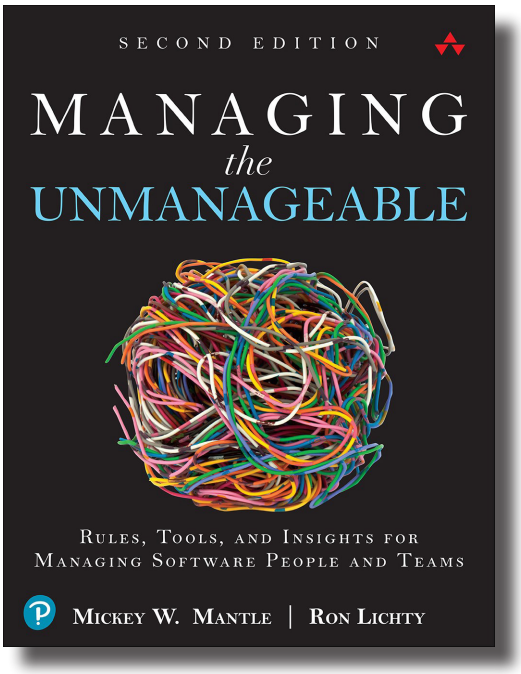
Effectively Manage Developers So You Can Deliver Better Software



ORDER & SAVE

Save 35% When You Order

from informit.com/infoq/managing and enter the code **INFOQ** during checkout

FREE US SHIPPING on print books

Major eBook Formats
Only InformIT offers PDF, EPUB, & MOBI together for one price

OTHER AVAILABILITY

Through O'Reilly Online Learning (Safari) subscription service

Booksellers and online retailers including Amazon/Kindle store and Barnes & Noble | bn.com

"Lichty and Mantle have assembled a guide that will help you hire, motivate, and mentor a software development team that functions at the highest level. Their rules of thumb and coaching advice form a great blueprint for new and experienced software engineering managers alike."
—Tom Conrad, CTO, Pandora

"Reading this book's nuggets felt like the sort of guidance that I would get from a trusted mentor. A mentor who I not only trusted, but one who trusted me to take the wisdom, understand its limits, and apply it correctly."
—Mike Fauzy, CTO, FauzyLogic

Today, many software projects continue to run catastrophically over schedule and budget, and still don't deliver what customers want. Some organizations conclude that software development can't be managed well. But it can—and it starts with people. Mickey W. Mantle and Ron Lichty show how to hire and develop programmers, onboard new hires quickly and successfully, and build and nurture highly effective and productive teams.

Drawing on over 80 years of combined industry experience, the authors share Rules of Thumb, Nuggets of Wisdom, checklists, and other Tools for successfully leading programmers and teams, whether they're co-located or dispersed worldwide.

- Find, recruit, and hire the right programmers, when you need them
- Manage programmers as the individuals they are
- Motivate software people and teams to accomplish truly great feats
- Create a successful development subculture that can thrive even in a toxic company culture
- Master the arts of managing down and managing up
- Embrace your role as a manager who empowers self-directed agile teams to thrive and succeed

Pearson
Addison-Wesley

**Discount code INFOQ confers a 35% discount off the list price on informit.com only. Discount may not be combined with any other offer, including the book + eBook "Best Value" bundle, and is subject to change.*

informIT®
the trusted technology learning source

Praise for *Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams*

Over 50 five-star reviews on Amazon.com (U.S.)!

"Lichty and Mantle have assembled a guide that will help you hire, motivate, and mentor a software development team that functions at the highest level. Their rules of thumb and coaching advice form a great blueprint for new and experienced software engineering managers alike."

—TOM CONRAD, CTO, Pandora

"I wish I'd had this material available years ago. I see lots and lots of 'meat' in here that I'll use over and over again as I try to become a better manager. The writing style is right on, and I love the personal anecdotes."

—STEVE JOHNSON, Senior Architect, Inlet Digital

"Managing the Unmanageable *is a well-written, must-have reference book for anyone serious about building sustainable software teams that consistently deliver high-quality solutions that meet expectations. It is loaded with incredibly useful and practical tips and tricks to deal with real-life situations commonly encountered by software managers anywhere in the world. It tearlessly peels back the onion layers of the process of managing software developers—whether a handful of co-located programmers or thousands dispersed across the world—through a balance of battle-tested approaches and keen understanding of the various personalities and backgrounds of software team members. Finally, a book on software engineering that focuses on the manager's dilemma of making a team of programmers work efficiently together. Every single software manager should have it on their bookshelf."*

—PHAC LE TUAN, CTO, Reepeet, and CEO, PaceWorks

"Becoming a great engineering leader requires more than technical know-how; Ron and Mickey's book provides a practical cookbook for the important softer side of engineering leadership, which can be applied to any software development organization."

—PAUL MELMON, VP of Engineering, NICE Systems

"EXCELLENT. Well-structured, logical, filled with great personal color and many little gems. You guys have done a great job here. Terrific balance between theory and practice, rich with info."

—JOE KLEINSCHMIDT, CEO, Obindo, former CTO, Leverage Software

"I started reading the nuggets section and it took fewer than four pages to improve my thinking. What struck me about the nuggets was that I could sense the genesis of this book: two masters of their craft learning from each other. Most books feel like a teacher describing a sterile version of what 'ought to be done' that leaves you wondering, 'Will this work in the "real world"?' Reading the nuggets felt like the sort of guidance that I would get from a trusted mentora—a mentor who I not only trusted, but one who trusted me to take the wisdom, understand its limits, and apply it correctly. It's concentrated like a Reader's Digest *for technical management wisdom."*

—MIKE FAUZY, CTO, FauzyLogic

"Managing the Unmanageable *is a great collection of sometimes-obvious and sometimes-not-obvious guidance for software managers. I wish that I had had this book when I first started managing teams, and it still is illuminating. For programmers who step into management, the hardest thing is to learn the soft skills. Ron and Mickey do a great job of illustrating not just the why but also the how."*

—BILL HOFMANN, VP of Engineering, Klamr.to

"Unique dialogue around the human aspects of software development that is very much overdue."

—MARK FRIEDMAN, CEO and founder, Greenaxle Solutions

"The advice provided herein about what to do on a new employee's first day of work seems unique and very helpful!"

—STEVEN FLANNES, PhD, author, People Skills 3.0: Next-Generation Leadership Skills for Project Success

"I just wish that I had this book when I started as a first-time manager five years ago!"

—KINNAR VORA, VP, Product Development and Operations, Sequoia Retail Systems

"The book provides insight to a unique group of people: programmers. Companies around the planet have struggled and are still struggling with how to best develop software products. Managing programmers is at the heart of developing software products successfully. Many project and organization leaders are ill-equipped to deal with programmers and software development in general. I think this book can bring insight to leaders of software organizations and help them understand and even get inside the head of programmers and therefore be more effective leaders."

—MICHAEL MAITLAND, CEO (geek-in-charge), WhereTheGeeksRoam

"I have enjoyed reading the book very much, and I wish I had it ten years ago—probably would have saved me from making certain mistakes. A lot of what I read is not new to me, but I have never seen so much relevant material assembled in one place. This book was just what I needed. I already feel that I've benefited from it."

—DAVID VYDRA, Continuous Delivery Advocate and Software Craftsman, TestDriven.com

"I found the book very helpful. It heightened my sensitivity to my staff, even having managed for decades."

—MARGO KANNENBERG, Assistant Director, Application Development, HighWire Press

"Mickey was my manager in my first role as programming manager. His real-world, pragmatic, hands-on guidance was a profound positive influence on everything I've ever done with management since. His is still my go-to advice as I develop and mentor managers. I'm pleased that he's taken the time to canonize it in this book so that many more new and experienced managers can benefit from it."

—H. B. SIEGEL, Director, Amazon.com

"Mantle and Lichty cut through abstract principles and present proven techniques that can increase the effectiveness of software development organizations. This book deserves a place on the real (or virtual) bookshelf of every software manager who wants to build an outstanding development team and create a culture where everyone enjoys coming to work. It's especially valuable in telling managers what not to do, and how to address the inevitable problems that affect all organizations."

—ANTHONY I. (TONY) WASSERMAN, Professor of Software Management Practice, Carnegie Mellon University—Silicon Valley; ACM Fellow; and IEEE Life Fellow

"Mickey was there on Long Island in the mid-1970s when the group now known as Pixar first formed, delivering successful software products then, and was still doing so, as manager, almost two decades later at Pixar itself. He knows what he's talking about."

—ALVY RAY SMITH, cofounder of Pixar

"Ron and Mickey clearly understand how important it is for programmers to work on projects that make a difference and how essential it is for managers to create and foster a unique and innovative culture."

—KATHY BALDANZA, VPE, Perforce Software

"This book is a treasure trove of real-world experiences that will make you a more effective software development manager."

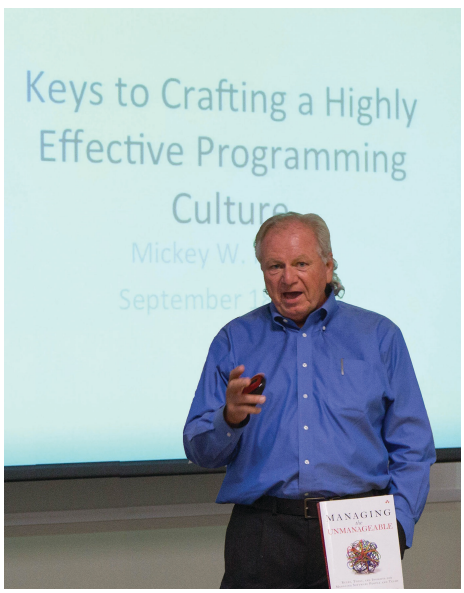
—CHRIS RICHARDSON, founder of the original CloudFoundry.com, and author of POJOs in Action

About the Authors

Mickey and Ron's software careers have spanned systems software, 2-D and 3-D graphics, multimedia, interface development, shrink-wrapped products, software-as-a-service, embedded devices, IT, Internet applications, mobile apps, professional services, and data warehousing and analytics. In recent years, they have taken on advising organizations in making software development more effective, stepped into interim and fractional VP Engineering roles, and trained and coached teams and executives in agile and in scrum. But they have seldom found the problems that plague software development to be domain or channel specific, and while problems have certainly been unique to organizations, they have nonetheless had much more in common than not.

Mickey W. Mantle

Mickey has been developing software for 50 years, creating hardware and software products and managing development teams. After graduating from the University of Utah (where he was contemporary with computer industry notables such as the founders of WordPerfect, Silicon Graphics, Netscape, Adobe Systems, and Pixar), Mickey was hired for his first programming job in 1971, developing the overall control software and real-time robotic controls for a six-acre aircraft rework facility for the



Author photo by Thomas De Lora

U.S. Navy at Kenway Engineering (later Eaton-Kenway). He thereafter joined 3-D computer graphics pioneer Evans & Sutherland (E&S), where he coauthored the original 3-D graphics library that paved the way for Silicon Graphics's GL. At E&S he was a contributor to many notable computer graphics products and first started managing programmers and programming teams.

After leaving E&S in 1984, Mickey joined Formative Technologies, a spin-off from Carnegie Mellon University, where he worked with the industry's first workstations (PERQ and Sun Microsystems) dealing with large-scale bit-mapped graphics for mapping and CAD applications. But his heart was in 3-D graphics, and he was hired by Pixar shortly after it was bought by Steve Jobs and spun out of Lucasfilm Ltd. in 1986. At Pixar, Mickey managed the development of all of the software for its external products, including the Pixar Image Computer, the Pixar Medical Imaging System, and RenderMan. RenderMan is the gold standard of 3-D photorealistic rendering software and by 2019 had been used on 27 of the 30 films to win the Academy Award for Best Visual Effects over the past 30 years.

Mickey left Pixar in 1991, as its focus shifted to making feature-length 3-D animated films and away from external software products, and joined Brøderbund Software as Vice President of Engineering/CTO. At Brøderbund he managed a vast development organization, including applications and systems programming, art and animation, sound design and music composition, and quality assurance that produced numerous award-winning PC/Mac games such as *Where in the World Is Carmen Sandiego?*, *Kid Pix*, *Myst*, and *Living Books*.

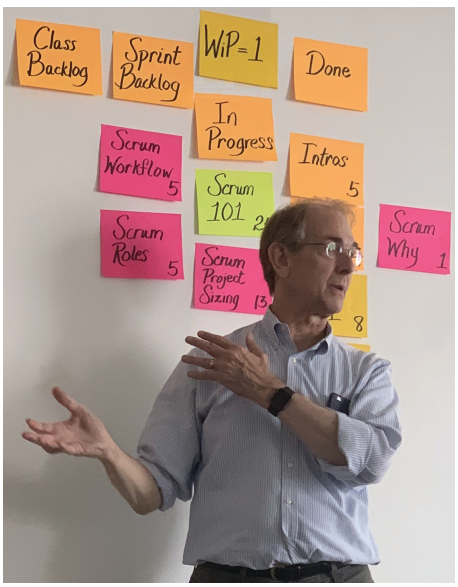
In late 1997 Mickey joined International Microcomputer Software, Inc., as Vice President of R&D/CTO, where he managed on-site and offshore development and support for numerous Windows/Mac applications such as MasterClips and professional-level products such as TurboCAD.

In 1999 Mickey joined Gracenote, where he was Senior Vice President of Development. (Since 2008 Gracenote has been a wholly owned subsidiary of Sony, Tribune Media, and most recently Nielsen—the global measurement and data analytics company.) At Gracenote he managed all development, operations, and professional services associated with the pioneering Web-based CDDB music information service that provides metadata and cover art to digital music player applications such as iTunes as well as hundreds of consumer electronics products. Gracenote's products utilize technology ranging from Web services and relational databases to embedded systems and mobile applications, giving Mickey a unique perspective on

the wide-ranging needs of the various types of software developed today. Mickey retired from Gracenote in 2011 to finish the first edition of this book and develop mobile, tablet, Windows, and macOS applications at his company Wanderful, Inc. He also consults with a variety of companies and organizations about effectively leading and managing technical people and teams.

Mickey's experience includes directing R&D teams around the world and managing multidisciplinary teams working 24/7 to deliver successful products. With experience in selecting, establishing, and managing offshore development organizations in India, Russia, Canada, Japan, and Korea, he brings insight into the challenges of managing software development using diverse staff and teams that are hours and oceans apart.

Ron Lichty



Author photo by Tammy Baker

Ron has been developing software for over 35 years, 30 of them as a Development Manager, Director of Engineering, and Vice President of Engineering in organizations ranging from tiny start-ups to Fortune 500 companies. This followed his first career as a writer in New York, Wyoming, and California, during which he wrote hundreds of articles, published scores of photographs, and authored two books.

His software development career began at Softwest in the heart of California's Silicon Valley, coding word-processing products, programming compiler code generators, crafting embedded microcontroller devices such as SmartCard–based postage meters and magnetic-keycard hotel locking systems (in the '80s!), and designing and developing the computer animation demo that Apple used to launch and promote a new line of personal computers. He was awarded software patents for compression algorithms and wrote two widely used programming texts.

Ron cut his teeth as a manager when Apple recruited him, in 1988, to create a product-management group for a line of its software development tools. He twice, at Apple, returned to programming, only to be made a manager, before he embraced management to lead the Finder and Applications teams, managing development of Apple's "special sauce," its user interface.

In 1994 Berkeley Systems recruited Ron to direct development of the then most widely used consumer software in the world, the *After Dark* screen saver line, to make engineering predictable and repeatable for the seven development teams creating its entertainment products. Brought into Fujitsu to make sense of its long-overdue *WorldsAway* entertainment product, he lopped off six months of overengineering to take it live in just 11 weeks.

Ron then led software development of the first investor tools on Schwab.com, part of remaking a bricks-and-mortar discount brokerage into the premier name in online financial services. He was promoted to Schwab Vice President and was recognized as Schwab's Technologist of the Year while leading his CIO's three-year technology transformation across every business unit from any-language-goes to a single, cost-effective platform companywide.

Since Schwab, he has been a Vice President of Engineering and Vice President of Products both as an employee and as a consultant, and he has continued to focus on making software development "hum." He headed technology for the California offices of Avenue A | Razorfish, the largest Internet professional services organization in the world; products and development for Forensic Logic, the crime detection and prevention company; development for Socialtext, the first commercial wiki company; engineering of the consumer ZoneAlarm line for Check Point; and product development for Stanford's HighWire Press, the largest Internet provider for scholarly publishing.

Ron left Stanford in early 2012 to finish the first edition of this book and launch a consulting practice to take on interim VP Engineering roles and to advise and coach engineering and product leaders both about role effectiveness and about making their organizations hum. In consulting engagements in the United States, Canada, Europe, and Asia, he has helped development teams overcome roadblocks, embrace agile, untangle organizational knots, and become more productive.

In his continued search for effective best practices, Ron coauthors the periodic *Study of Product Team Performance*. His developer conference and professional group talks and webinars include facing down the challenges of implementing agile and scrum; the critical roles managers play whose teams have gone agile; the importance of user groups, teamwork, and community in software development; and transforming software development from chaos to clarity. He has been an adviser to a half-dozen start-ups. He cochairs the Silicon Valley Engineering Leadership Community[1] and previously cochaired SVForum's Emerging Technology Special Interest Group (SIG); cofounded SVForum's Software Architecture SIG; cochaired EBIG's Software Development Best Practices SIG; and served on the board of SVForum, Silicon Valley's largest and oldest developer organization.

1. www.meetup.com/SV-ELC/.

Table of Contents

Preface

All too often, software development is deemed unmanageable. The news abounds with stories of software projects that have run ridiculously over schedule and budget. While strides made in formalizing the practice of software development have improved the situation, they have not solved the problem. Given that our craft has amassed over 60 years of experience and our industry has spent enormous numbers of hours and dollars/yen/won/yuan/rupees/euros trying to bring this discipline under control, how can it be that software development remains so unmanageable?

In this book we answer that persistent question with a simple observation: You first must learn the craft of managing programmers and software teams. That is, you must learn to understand your people—how to hire them, motivate them, and lead them to develop and deliver great products. Based on our own experience, and that of effective managers we have known in virtually every type of software business, we aim here to show you how. Combined, we have spent over 80 years working on and delivering a wide spectrum of software programs and projects—over 65 of those years managing the programmers and teams that delivered them. We hope that this book will help you avoid many of the mistakes we have made, as well as leverage for your own success the insights and skills we have learned.

Early in our careers as programmers, we both read Fred Brooks's 1975 book *The Mythical Man-Month*. An instant classic among programmers, it is full of wisdom still relevant today and is widely regarded as a definitive work in the art of software management. Like many others who read it, we found the most memorable parts to be Brooks's one-line nuggets of wisdom, such as "*Adding manpower to a late software project makes it later*." We can't recall the number of times we've used this quote when managing

software projects. The desire to find other such memorable rules of thumb[1] was the inspiration and driving force behind the writing of this book.

We were already seasoned managers when, as friends, we began meeting regularly to compare notes on our current work and software development challenges. We found ourselves getting help from each other and sharing an occasional nugget of wisdom or rule of thumb, which we would then take back to our jobs, integrate into our management approach, and share with our teams. We gleaned rules and nuggets from the books we read and the Web sites we surfed, but we never found a collection of them specific to managing programmers and teams developing software. Eventually our own desire to have such a collection led to our decision to write this book.

A broader perspective emerged as we began writing and talked to managers, directors, and CTOs. It became clear that we could draw from the breadth of our industry experience to offer considerably more than the rules of thumb we'd collected. We could also share the tools[2] we'd developed and the insights we'd gleaned from working in start-ups and in organizations of every size.

There are certainly areas we haven't touched in our careers—domains such as large-scale government contracting and defense systems. But our experience is relevant to most companies developing software today, including those companies whose managers are working on the edge of innovation. That latter group tends to be young and is seldom offered any formal management training or organizational support—or has time for it anyway. Unfortunately, that's how all too many managers learn today: on the job.

We wanted to write a book that could be a mentor of sorts for programming managers—a book filled with insights, stories, and guidance gained from years of learning the hard way how to do it successfully.

We realized we could also share the tools we have developed over the years that make managing easier—tools such as job descriptions, rankings spreadsheets, project workbooks, team technology inventories, programmer first-day schedule templates, and hiring checklists. They can save managers many hours developing tools from scratch when they find themselves working in organizations that are too immature to provide their people with the tools they need (all too common, unfortunately, in the fast-moving world of

1. See the 300 Rules of Thumb and Nuggets of Wisdom in what one reviewer called the "soft, creamy center" of this book.

2. See the tools for each chapter in the Tools section at the end of the book.

software development). These are the tools we wished we'd had when we first started managing.

We wondered if there needed to be another book about software development. Surely—with no end of books, articles, and Web sites about engineering software, managing process, and managing projects—some number of gifted engineering managers must have shared their secrets. Yet we found scant more examples focused on managing programmers and software development teams than we had when we began our careers.

There is no methodology for newly anointed development managers charged with managing, leading, guiding, and reviewing the performance of a team of programmers—often, the teams they were on just days before. There are no off-the-shelf approaches. Unlike project managers, who devote hours and hours of study toward certification in their chosen career path, development managers often win their management roles primarily from having been stellar coders while displaying a modicum of people skills.

Among the books we did find, none contained the kinds of behind-the-scenes stories and anecdotes we have incorporated into this book—stories and anecdotes that speak directly to how to handle specific situations that managers face.

Organization of the Book

In the chapters of this book we share our hard-won experience gained from programming, managing, and delivering software spanning two managerial lifetimes of companies and situations. We have distilled our insights into ten chapters sprinkled with anecdotes from our experience, as well as Rules of Thumb and Nuggets of Wisdom collected from others.

Chapter 1 reviews why programmers are special when it comes to managing them as individuals and managing them as teams. It's thinking about the qualities that characterize programmers that makes it obvious why you can't just pick up any book on management to start managing a team of programmers.

Chapter 2 provides a number of lenses through which to view the programmers on your teams that will help you see the individuality each of them brings—and inform your managing each of them uniquely.

Chapter 3 is a step-by-step guide to finding, recruiting, and hiring great programmers. Early readers of this chapter found themselves tearing it out of the manuscript to use separately. You may, too, but you'll leverage it best

from the context of the prior two chapters—knowing just who it is you're hiring—and from incorporating culture and motivation from Chapters 7 and 8.

Chapter 4 counsels how to keep candidates' enthusiasm between "yes" and start; prevent "buyer's remorse"; and, when they do arrive, integrate them quickly, effectively, and productively into your processes and practices. New managers tend to think their recruiting role is finished when a candidate accepts an offer, but too many have learned otherwise when a candidate failed to show up for their first day, floundered in fusing with the team, or never became productive.

Chapter 5 walks through the core of management—managing down. These are the mechanics and how-to of the day-to-day with your team, the tasks and interactions to successfully manage programmers.

Chapter 6 addresses the fact that success as a programming manager also demands that you become skillful at managing up—managing your boss (and possibly his boss); managing out—managing your relationships with your peers, leveraging other departments or folks within your company, and marshaling external resources and relationships; and finally managing yourself—your priorities, your style, your time, your growth, your life.

In an interlude called *Rules of Thumb and Nuggets of Wisdom*, inserted between Chapters 6 and 7, we've collected hundreds of, well, rules of thumb and nuggets of wisdom that have proven valuable to us over the years, denoted by lightly shaded pages for ease of access. We collected them from a broad cross section of programmers, development managers, and software luminaries.[3] The wisdom drawn from these adages, used judiciously, can help you make a point, win an argument, reframe a conversation, or defuse a tense discussion with a bit of humor that still drives your position home.

Chapter 7 turns the focus back to the team and the critical task of motivating programmers to accomplish great feats and deliver difficult projects. The chapter opens with grounding in the motivational theories of Maslow, McGregor, and Herzberg. The differentiation of motivators from demotivators—they are very different, contrary to popular thinking—was

3. If we have misattributed a rule of thumb or quote, we apologize in advance (and please let us know). Some of them are available only through word of mouth or indirect sources, making completely accurate attribution almost impossible. The titles given in the attributions are those for which the person is best known or, in many cases, their title when we knew them and heard their insights directly.

essential to our own managerial growth. Given that each programmer is unique, there's no motivational silver bullet, but our framework can help you think about ways to motivate—and how to recognize and avoid the potholes that demotivate—your team.

Chapter 8 provides context to think about your corporate culture and about how you can create the development subculture you need for success within even the most toxic of corporate cultures. Too few managers realize their critical role in creating a team culture that supports success. Chapters 5 and 6 cover the mechanics basic to managing, but Chapters 7 and 8 cover the two subtle sets of soft skills that can differentiate your management and help pave your way to success.

Chapter 9 returns to basics. The eight preceding chapters ultimately point to this objective: delivering software successfully. This chapter is not about project management but about the role seldom addressed: the team manager's essential role in delivery, even in agile environments. Success depends on synthesizing all the skills and efforts outlined in the previous chapters, as well as a mindset that is all its own.

Chapter 10 expands on the topic of agile development that is sprinkled throughout the book and answers the important question of what the role of a manager is when a company transitions to self-directed agile teams.

The *Tools* section provides a collection of useful tools, among them checklists, forms, reports, and so on, that we devised to aid our efforts to recruit, hire, and effectively manage and motivate programmers to deliver quality software successfully. We're certain they will aid your efforts as well and save you the time of having to create them anew. These tools are available online at www.managingtheunmanageable.net/tools.html.

Use This Book as a Reference

Many of the readers of the first edition of this book told us that they not only read the book but, more importantly, also used it as a reference that they turned to whenever they found themselves confronted with a management problem. We originally intended it for exactly this purpose, so we are glad to see that some have embraced it as we intended. We encourage you to pull it off your bookshelf when you wonder, *What would Mickey or Ron do now?*, and look in the detailed table of contents or in the carefully constructed index to find the section or sections that might apply to your problem. It's like having a personal mentor, always available to help.

Lessons Learned

Programmers and software teams need not be unmanageable, but it takes talented managers who are dedicated to doing the hard work of managing seemingly unmanageable personalities to succeed. We can certainly affirm that writing this—and the rules, tools, and conversations we shared as we transformed our thinking into words—made both of us better managers, made our jobs easier, made our teams happier, and made our projects more successful. We hope the rules, tools, and insights we have provided in this book will make your jobs easier, as well.

Remember: *Managing the Unmanageable* comes in three parts. You can forge straight ahead into the chapters. Or you can dip into the Rules of Thumb and Nuggets of Wisdom, which appear as an interlude between Chapters 6 and 7. And at any time you can look through the tools, summarized in the Tools section at the back of the book, for extra resources.

Acknowledgments

There are many people to thank who have helped us write this book. First and foremost, we want to thank our wives for encouraging us in our efforts to draft, redraft, and craft this book. Without their patience, help, and advice this book would not have been possible. Second, we want to thank Peter Gordon and Kim (Boedigheimer) Spenceley of Addison-Wesley for their continued support and advice over the years and for having faith that the work we would create was worth their time and energy to help make it happen. Peter's advice on organizing the book was especially helpful in the late stages. Kim bravely brought us into the multimedia world, signing us up to, in 2017, transform these words into video training as *LiveLessons: Managing Software People and Teams*.[4] And many thanks to Haze Humbert, editor for this second edition.

Next, we must thank the many originators of the rules of thumb that we have included in this work. The sage wisdom that they repeatedly imparted was the primary motivation for this book, and we marvel at the depth of insight that can be conveyed in so few words.

4. www.managingtheunmanageable.net/video.html.

We must also thank the many reviewers who spent considerable time and effort to provide detailed feedback that helped guide us to revise and improve our writing over the years. Among them were Brad Appleton, Carol Hoover, Carrie Butler, Clark Dodsworth, Daniel J. Paulish, David Vydra, Dr. Dinesh Kulkarni, George Ludwig, Harinath V. Thummalapalli, Jean Doyle, Joe Kleinschmidt, Kinnar Vora, Margo Kannenberg, Mark Friedman, Michael Maitland, Patrick Bailey, Rama Chetlapalli, Stefano Pacifico, Steve Johnson, Steven Flannes, and others who remained anonymous to us. We are grateful to Niel Nickolaisen, Rich Mironov, Cathy Simpson, Ed Burnett, Travis Klinker, Rob Parker, John Colton, Scott Henderson, Matthew Leeds, Anthony Moisant, and Thomas Barton for their insights into the new material in this second edition. Thanks to Marty Brounstein for clarity on his intervention technique. Special thanks to Georgia McNamara, who showed us the way through the vagaries of the English language to rid this writing of unintended male-gender references and make it as friendly to all as we had originally wanted; we are so, so appreciative. This work is definitely better because of all of their thoughtful feedback.

Figure 10.1 was created using the people artwork designed by rawpixel.com/Freepik. Figures 10.7 and 10.8 are courtesy of agilemanifesto.org.

Finally, we would like to thank the legions of programmers, managers, and executives with whom we have worked in all the various companies throughout our careers. It is because of them, and the experiences we gained working with them, that this book is possible.

Mickey W. Mantle
Ron Lichty
October 2019

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, Anniversary Edition* (Addison-Wesley, 1995).
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 programmers 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, Chris Abts, A. Winsor Brown, Sunita Chulani, and Bradford Clark, *Software Cost Estimation with Cocomo II* (Addison-Wesley, 2000).

6. Glass, "Frequently Forgotten Fundamental Facts about Software Engineering," *IEEE Software* 18, no. 3 (2001): 112–13.

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.

Think through whether your focus will be on experience, or on energy and passion.

Do you need

- A programmer who can mentor the team in best practices?
- A coder with a mind wired to ferret out the gnarliest design flaw?
- A designer who can sense the big picture and envision how your requirements can be broken up into modules and components?
- A developer who is comfortable being proactive and collaborative with management?

Or do you need

- To churn out thousands of lines of code in short order?
- To prototype features important to your customers that your veteran programmers blow off as "fluff"?
- The flexibility and speed to iterate routines over and over as their essence becomes clear?

These are not mutually exclusive sets of characteristics. But the former type of programmer is more likely highly experienced. And the latter type is more likely a fresh, passionate one. Be conscious of which you need.

When it comes to getting things done, we need fewer architects and more bricklayers.

—COLLEEN C. BARRETT, President and Corporate Secretary of Southwest Airlines

You also need to know whether you are better off with a full-time employee or a contractor.

Do you need

- A programmer for the long term?
- A fully integrated member of the development team?
- A developer with an evolving set of skills and tools whom you expect (and are willing) to train and grow over time, as needs or technologies change?

Or do you need

- A highly developed set of specialized skills and tools now?
- To fill a short-term need?

The former is likely an employee; the latter, a contractor.

Finally, will you consider distant candidates, either to move them to your location or to have them work remotely? Are you unable to find the candidates you want in the local pool, or is the skill set you need so rare that there is no pool of candidates, short of thinking regionally or nationally? Can you afford to pay moving costs? Or are you up to managing a geographically distributed team?

> *Distributed development can be made to work, but a distributed team will never perform as well as a collocated team.*
>
> *—MIKE COHN, Agile and Scrum Thought Leader*[8]

If so, you're likely to find yourself conducting some or all of your interviews by phone or videoconference. Conversely, you can leverage national trade shows and conferences to meet and recruit programmers who are uniquely qualified.

While at Apple, Ron frequently sought out hires at the Applefest and MacWorld and OOPSLA conferences. After giving a talk one year, he was approached by a programmer with laryngitis who was madly scribbling messages on pages of a 3″ × 5″ pad to communicate his interest in Apple. It was an odd approach, but he soon became a stellar Apple hire.

Writing the Job Description

Your hiring effort begins with writing a job description suitable for posting. Keep in mind that the objective of this description is to attract the largest number of qualified candidates—it's a marketing brochure for the position. It should be specific about what you're looking for to discourage the unqualified, broad where you're open to a wider set of talents, and persuasive about

8. Mike Cohn, *Succeeding with Agile: Software Development Using Scrum* (Addison-Wesley, 2010), p. 387.

the company as a great place to work and the position as an ideal opportunity for your kind of programmer, highlighting the social significance, career visibility, and lasting value of the contribution the right candidate can make.

In a small company, writing job descriptions will likely fall to you. In medium-size and larger companies, the staffing or HR organization will prepare these, but you should plan to collaborate on or edit them if not outright provide all or most of the job description and requirements. Unless you actually write the posting and are certain it will be run verbatim, it's wise to ask to be included in a review cycle. We have seen job postings (some of them appearing in expensive display advertising space in Sunday newspapers) that are inadequate and sometimes downright embarrassing due to rewrites of requirements and use of maddeningly ancient boilerplate by well-meaning but nontechnical recruiters.

If you're writing the posting yourself, you're probably thinking you'll draw it from the internal job description; we showed you a sample in Chapter 2. But that's really barely a start. It lacks both job-specific detail and the sparkle to attract candidates.

Your internal job description is only the starting place for your external posting. Buff it up, add spice, make it appealing.

For example, while "Programmer 3" may be a fitting title within your organization, you'll need one that is both more meaningful and more descriptive of the background and technology experience you're looking for candidates to have. The other key qualifier that most job seekers use to determine if a job might be a fit is location. "San Francisco Bay Area" is much too general; in a good economy, a programmer trying to minimize the commute will skip right by your listing rather than try to figure out where in the 100-plus-mile Bay Area the job actually is. Be specific.

That will lead to titles such as

- Entry-Level Ruby on Rails Programmer—SF Peninsula
- Experienced Full Stack Programmer—Cambridge
- Oracle Programmer with BI Experience—South Bay (SF)
- Java Architect—Denver
- Support Engineer, .Net/Sharepoint—Vancouver
- Principal Programmer, Engines Team, Search Technologies—Austin

(The latter title, since it doesn't specify C++ or Java or Python, might be for a role where you're much more concerned about finding a candidate with strength in algorithms, data structures, and system software internals than with specific language or platform experience.)

If telecommuting three days a week is possible, the top of the listing is the place to put that, as the example in Figure 3.1 shows.

Now come the key elements of your job posting. The first is a brief summary of the company and its product(s) with your focus on why a programmer would want to join you there. This paragraph is about selling, so if you're not used to writing compelling copy, get one of your sales or marketing colleagues to help.

Next is a job description that is specific to the job for which you're hiring. Here again, the internal job descriptions in Chapter 2 are too general for recruiting purposes. What coding, design, and architecture do you need this programmer to create? What special technical skills and knowledge do you need? What best practices experience do you expect as a minimum qualification? Are you looking for someone to lead or mentor other programmers, or give them technical direction? Do you need a communicator who can collaborate with your business partner? A technical guru who can translate business requirements into technical ones, or even directly into architecture? A mathematical wizard who can turn business requirements into complex analytical algorithms? A UI designer who from business requirements can conjure up a brilliant UI that users just intuit?

Now you're ready to describe the skills you're looking for. This is the time to describe the language and platform experience you need, in detail, along with the level of skill and knowledge you expect. You should also be specific about the experience you need candidates to have with leadership, management, project management, communication, analysis, design, architecture, and coding. Are you hoping for a programmer who takes direction and just codes? Or a programmer who gives direction? How many years of experience? Education? (There is no consensus regarding the extent to which education is a predictor of programmer success, so we would not recommend making a degree an absolute requirement short of some statutory requirement or some organizational quirk that would make lack of a degree a predictor of failure.)

Sometimes follow-through, attention to detail, and a sense of ownership can be more important than specific skills. Don't ignore these "soft skills" when crafting your job descriptions and during the interview.

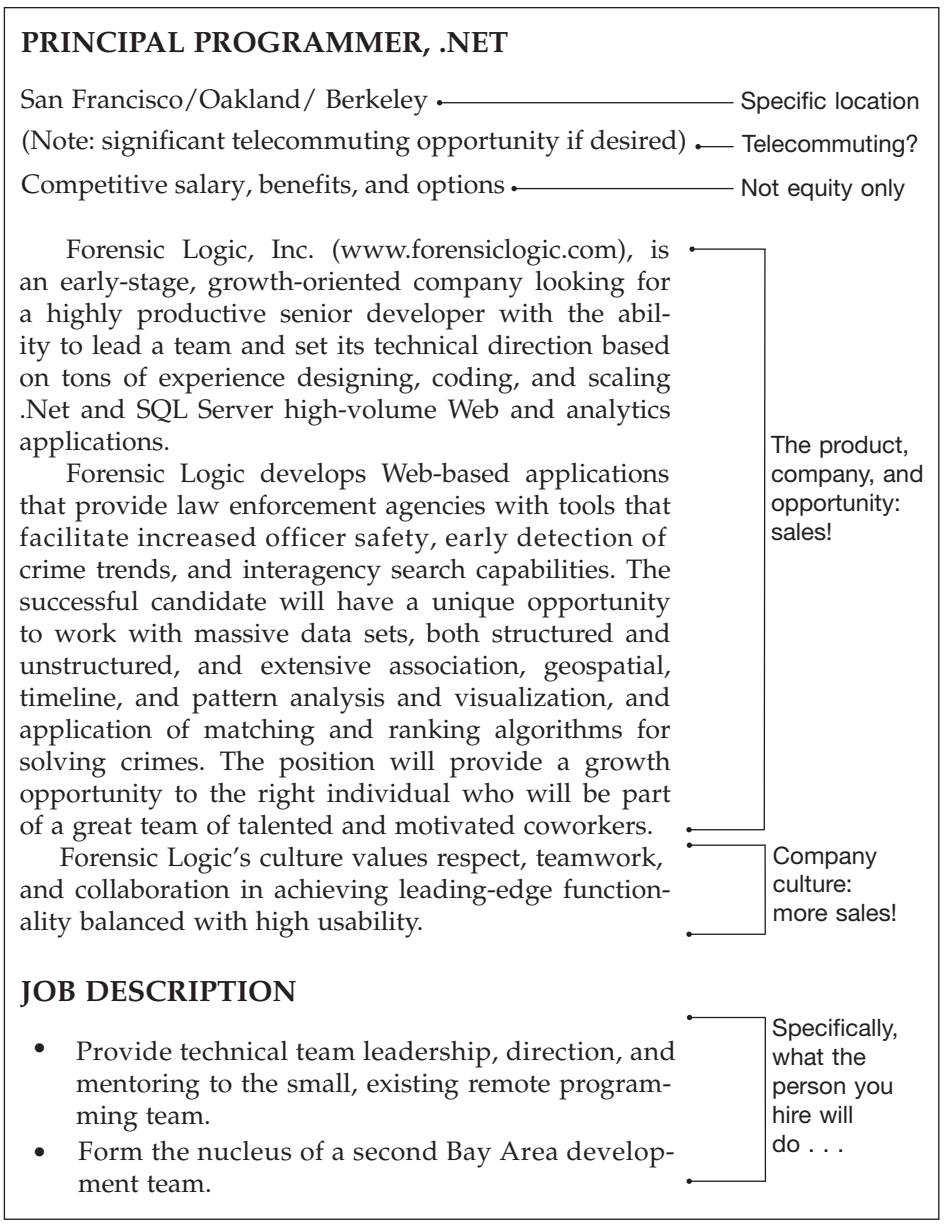


Figure 3.1 Sample job description (continued next page)

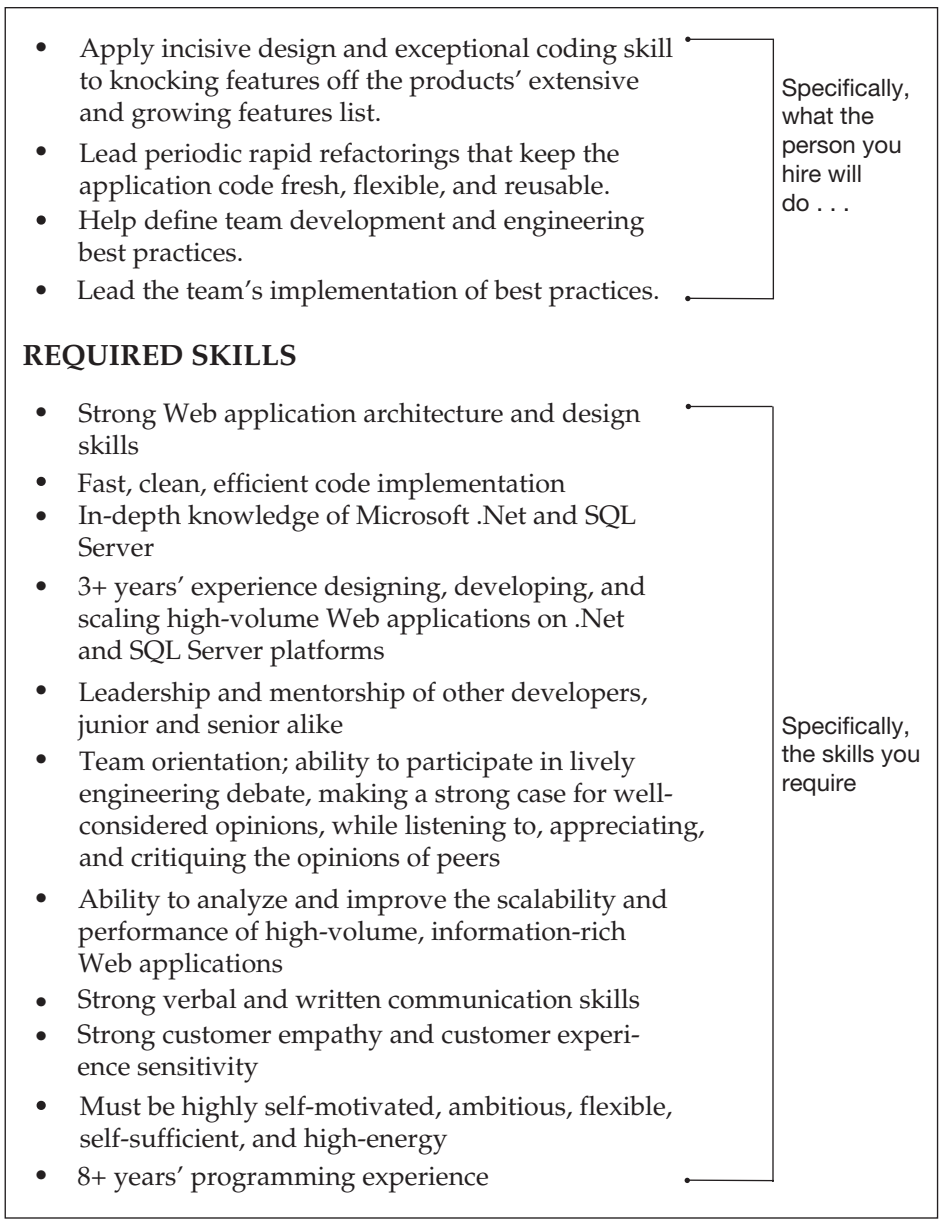


Figure 3.1 Sample job description (continued next page)

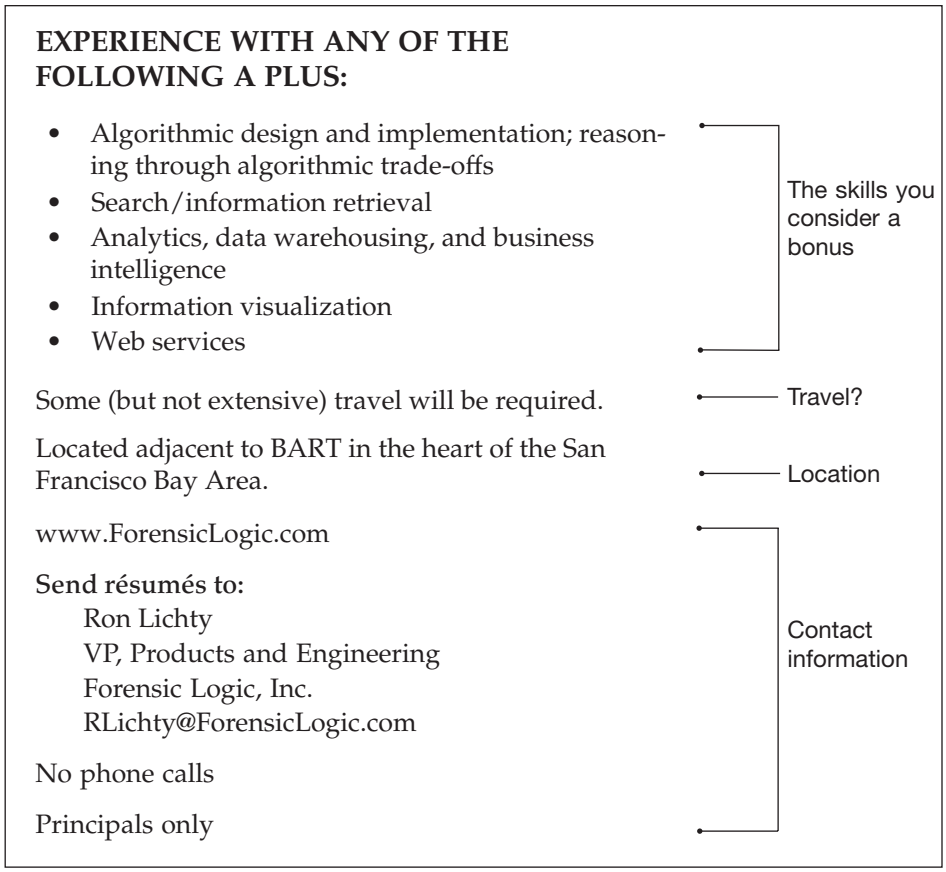


Figure 3.1 Sample job description

I emphasize communication, collaboration, energy, potential.

—MARK HIMELSTEIN, Interim VP of Engineering, San Francisco Bay Area

Divide skills into "required"—what you absolutely won't hire without—and those that you would really consider a bonus in a candidate who has the required skill set.

Also consider whether travel will be required. Disclosing it here will increase your chances for a good fit long-term.

Finally, don't forget to give candidates a way to contact you—an e-mail address, usually, as well as your Web address.

Selling the Hire

Are you budgeted for another programmer? No? Then you're going to have to sell your management on why you need to hire. In most cases, that means making a business case.

Think through your need thoroughly first. Do you need the new person because the team is missing a specific expertise? Can you change technologies to mitigate the need or to make the resource easier to find? With the hybridization of applications—programs have not only become systems made up of multiple objects, components, and services, but of multiple languages—you don't have to convert your entire application. Some years ago, one of our colleagues stopped battling Ruby's scalability constraints by sectioning off the critical area and recoding it in Scala. Another got around a display-layer bottleneck of scarce XSLT gurus by layering PHP Drupal on top.

When the expertise seems truly needed, you can make your case visually by taking a census of your team members and an inventory of their skills and presenting compelling visuals of your existing expertise against the expertise required by your customers, your products, or your marketplace.

On the other hand, perhaps there simply seems to be too much work. Be wary if that's a short-term need: Remember the Mythical Man-Month rule of thumb that adding resources to a late project will make it later. Your management may remember it if you don't, even if they're not reluctant to fund a hire.

Prepare to show that you have thought through every alternative to hiring. Instead of hiring, can you carve off a piece of functionality and have it programmed out of house/offshore to the lowest bidder? Can you convert your process to agile and your product managers to Software by Numbers[9] to focus on limiting development to a smaller but earlier initial release of

9. Mark Denne and Jane Cleland-Huang, *Software by Numbers: Low-Risk, High-Return Development* (Prentice Hall, 2003). This book introduced Minimum Marketable Features (MMFs) and an Incremental Funding Methodology (IFM) based on the notion that software is never complete, and it showed how to prioritize a project based on return so that it can become self-funding earlier.

just the highest-return features using the team you already have? Can you improve productivity and throughput by improving your team's processes?

Often enough, even with those analyses, you come up short and need to pitch resources. When Ron was faced at one firm with having a team of 30 programmers—a fifth of the firm's heads—and yet short what he needed to deliver customers' work, he helped make his case by drawing a new organization chart based on customers. The chart was, for the first time, visual evidence that once each of the most influential customers had been assigned the dedicated programmers its work deserved; the remaining customers were left without any. In another case, he gathered statistics of incoming requests for work and projects completed and graphed the rapidly growing backlog of work requests.

Regardless of the analysis, you may run up against the hard fact that your budget (or the department's budget or the company's budget) will not let you add headcount. To get the resource you need, you'll have to lay off someone or terminate a poor performer. If you're doing the right thing for your team and your project and your company, you'll likely sooner or later be faced with making tough decisions like this one to get the resource(s) you need.

Recruiting Full-Time Employees (FTEs)

Now that you can describe the type of employee you're looking for, you need to think through where you are going to find candidates and how much you can spend to do so.

You may luck out. If you're in a large organization, there may be a programmer in another part of the company with an established reputation who wants to work for you. As with any other candidate, express enthusiasm while privately checking the facts, verifying the person's reputation, and satisfying yourself that their credentials and qualifications apply to and are a fit with your project and your needs.

Be aware that most large organizations have an established process for employees to check out opportunities elsewhere in the company. There may be requirements that they spend a year in the job into which they were hired before they're eligible to move. They may be required to give a heads-up to their current manager before talking to you. Or they may be allowed to talk with you informally about what you have available but be required to post a form to HR before they can apply. They may be required to resolve

issues with their current organization before being allowed to look outside it. You as a hiring manager may have constraints. There may be rules to prevent or at least discourage the "cool" projects from "raiding" more mundane ones. Or conversely you may be expected to consider internal candidates first. These are questions only HR can answer definitively, and you should always ask.

At Fujitsu, Ron found a "diamond in the rough" programmer in the quality assurance (QA) organization and worked with the business unit's executive director and his peer to ascertain the tester's interest in development and then to transition him gently. That and some mentoring made him a stellar hire.

Always Be Recruiting

To start your recruiting, post positions on your own Web site. Include not only the active positions you are recruiting to fill, but also positions for which you always seem to need new talent. "At Gracenote," says Mickey, "we were always looking for Oracle database developers and embedded programmers. So we continued to collect résumés and review them even if we did not have positions open. If we saw a bona fide superstar, we would bring the person in to interview and make the case for increasing headcount (which is always easier if you've found a bona fide superstar candidate)."

When Ron first went to Razorfish, his teams were working with almost every technology but Microsoft's. "I didn't even have a folder set up for Microsoft coders when an information architect from upstairs came by to give me a résumé of a guy she'd worked with before, a .Net senior coder. I took a look at the résumé and knew I couldn't use him—and I sure didn't anticipate that changing—but I also knew if I ever did need a C# programmer, I was looking at the résumé of one I'd want. It doesn't happen like this often, but a few weeks later one of our clients asked us to help them solve problems with one of their C# apps! I sure felt lucky."

Mickey has numerous examples of interviewing candidates but not having the right position for them at the time. One example comes from Brøderbund: "I interviewed a guy who was not right for the job we were recruiting to fill, but I liked him and stayed in touch with him occasionally. Almost three years later the right position opened up and he was hired almost immediately. He turned out to be a superstar and was well worth the patience and waiting for the right position to open up."

By thinking of recruiting not as a series of one-time challenges but as ongoing relationship building, you'll add value to your network in the short term and your recruiting in the long term. You should always be recruiting!

> *You should always be thinking about building a network of possible employees and referrers and staying in touch; the person who turns you down this year may be next year's awesome hire. And the candidate you would think would never come and join your company may have their own perspective, or may refer a friend.*
>
> *—TIM DIERKS, Programmer, CTO, and VP of Engineering, Apple, Google, and elsewhere*

If you're at a start-up, you may find that your own network, your list of potential candidates, and referrals from your colleagues are all you have to work with. You can make some great hires with nothing more; you'll just have to work your limited paths harder.

Budgeting for Recruiting

One of the first things to know about hiring is how much you can spend to find candidates. Marketing costs to attract and recruit full-time employees can include

- Paying commissions to headhunters
- Engaging an internal recruiter or retaining an external one for this or a group of hires
- Paying employees bonuses for making successful referrals
- Paying to list your position online with the likes of LinkedIn
- Organizing a special recruiting event, perhaps around the time and location of a conference focused on a key technology in which you need expertise
- Paying to fly in remote candidates to interview (and potentially paying for moving expenses to relocate them, should you decide to go that route)

For any given hire, large companies will likely set strict limits on what avenues you can pursue and how much you can spend (mitigated sometimes by providing recruiters in-house and by letting you recruit from those already part of your organization—lateral hires). Smaller companies may

be more flexible with outside recruiting resources and dollars. Early-stage start-ups may give you no budget whatsoever.

Resolve the headhunter question first. If you're in a rush to hire, or you're anxious to increase the certainty of making a hire, particularly in a fast economy, working with two or three effective headhunters can vastly improve your candidate pool.

There are lots of mediocre recruiters. The recruiter you want is one with whom you can do a quick mind-meld, one who will almost instantly understand your needs and mirror them back to you first verbally and then in the form of perfect candidates.

The cost of using headhunters—you don't pay "contingency recruiters" unless you hire a candidate whom you had not previously contacted regarding a specific position—is usually a percentage of the new hire's first-year salary. The percentage was once 15 percent, but these days it is seldom less than 20, and 25 percent is not uncommon.

Companies of any size will have their own standard contract stipulating the conditions under which candidates are presented and commissions are earned, including an absolute ceiling on the commission percentage. Just make sure you have a contract in place with a recruiter before you accept résumés or interview any of the recruiter's candidates—or risk heartbreak when your HR department tells you that you can't hire the perfect candidate whom you just had 12 people invest their time interviewing because the recruiter won't meet your company's terms.

Beware: There are some less-than-ethical recruiters. Look to engage only exceptional recruiters with absolute integrity.

Avoid boiler-room operations. These are people who would, but for the fortuitous offer of a job in recruiting, be calling you at home during dinnertime to sell you carpets or drapery cleaning or credit repair. When they interrupt you with phone calls at work, they're no less annoying. Some of them will lie and tell you that your colleague "Bob" (pick a name in your organization they just heard) pointed them to you. Some of them will lie and tell you they have a "perfect candidate" with the very set of skills you're looking for and a pedigree so perfect any manager would leap to hire the candidate. On the flip side, programmers get calls about "perfect jobs" that may or may not exist and soon realize the "recruiters" cold-calling them don't know anything.

The worst for you—worse even than being unable to shake a recruiter who hounds you with phone calls—is the recruiter who "introduces" a

candidate to whom your organization is already talking, then accuses you of lying and threatens to sue you for a commission.

You can mostly avoid the bad apples and find stellar recruiters by relying on recommendations from other managers and being nothing more than polite to the cold callers.

Ron's feeling is that no interruption by an unsolicited recruiter on the telephone is acceptable. He is civil and asks them to e-mail him. He keeps expecting this to change, but so far it hasn't: The boiler-room operators never, ever e-mail. Their game is a telephone game. He won't hear from them for a month or more, when they phone again. He is always very cordial (until they try to keep him on the phone instead of listening to how he wants to communicate with them). He tells them he loves to communicate with recruiters—which generally truly throws them off their spiel—then after a pause says, "But until I get to know them, I only want to communicate via e-mail." They pester him with questions, to each of which he replies, "I'll look for your e-mail." And after a few of those, he says goodbye and hangs up the phone.

There are, of course, the rare real recruiters who cold-call, but they will be more than willing to contact you however you want. From them you'll see e-mails and candidates and interaction on your terms. And some of them will make it onto your personal list of preferred recruiters.

Recruiter Case Study

In late 2009, Elaine Wherry, one of the cofounders of Meebo, created a fictitious online persona, a JavaScript developer at her company. The fictitious programmer launched his own Web site, LinkedIn profile, and Facebook page. She was fishing for recruiters, hoping her persona would show her who were the best. Over the next 18 months, as her fictitious developer received 237 e-mails from 180 recruiters and 195 companies, she unexpectedly stumbled upon some great insights about recruiting.

Elaine was looking for prize JavaScript superstars; she needed to double the size of her team. She had already tried all the guerrilla recruiting tactics she could think of. She had placed Google AdWords (to pretty much no effect); embedded a "secretjobs" e-mail address into the gnarliest source code on her company's site to snare anyone daring enough to read it; put logo'd T-shirts on students' chairs during Stanford finals for the classes likeliest to deliver her talent; set up a jobs page chat widget (which she described as "useful"); devised JavaScript blog puzzlers and bingo; networked at JavaScript meetups; set up a résumé spider engine; spoke at events; participated

in Stanford's computer science classes; advertised in student newspapers; and placed Twitter keywords. And she created a map of the JavaScript community that proved useful to her recruiters. But when all that wasn't enough, she hit on creating the persona—a honeypot she hoped would attract recruiters who would be best able to find and deliver the coders she needed.

Initially, she gave her persona a guy's name, a great résumé, and a good blog but got nothing for two months. Then she filled out a profile for him on LinkedIn—and was flooded. What she learned was that despite what every recruiter told her about how broadly they looked, by 2009 recruiters were relying almost exclusively on LinkedIn. So she began turning over rocks for non-LinkedIn-listed programmers.

When she found that her competition for the coders she wanted was not just the big guys—Google and Amazon and Apple and their ilk—but predominantly the midsize and smaller companies, she began working harder to differentiate her company from the rest.

She found that every single recruiter her company had ever employed who was no longer contractually prevented from doing so tried to recruit away her "programmer"—and realized how important it is to keep your prized programmers happy: free food, great people to work with, and interesting stuff to work on.

When she realized how poorly prepared most recruiters were—how many were shotgunning impersonal, canned e-mails—she made sure her own recruiters were armed with her company's mission statement, had specifics about the role being recruited for, and referred to something in candidates' profiles and on their blogs that made them a good fit for the job requirements. Realizing how few stellar recruiters she came across, she determined to treat her few good ones like gems.[10]

Employee Referrals

While we think you should make your initial decisions with respect to recruiters right away, in our opinion the number-one source of candidates (and in a start-up with limited funding, virtually your only source) is employee referrals. With referrals, you're leveraging the people you already have in your organization to recommend their friends and former colleagues. Every study we've seen supports our experience: Good people recommend other good

10. Elaine Wherry shared her lessons learned in her Silicon Valley Code Camp 2011 session, "Winning the Engineering Talent War Online," and later in her blog at www.elainewherry.com/2012/06/26/the-recruiter-honeypot.

people. And you get a built-in reference, usually with contact information for other former colleagues who will vouch for the candidate as well.

> *If your current employees are happy, they will refer other great employees to you. So make your place a desirable place to work—including offices for programmers, good leadership, and perks.*
>
> *—GREGORY CLOSE, Manager, Project Manager, and Start-up Founder, San Francisco Bay Area*

It would be nice to think that your entire organization would recruit their friends to your team every time you have an opening. But the fact is that people can be hesitant to solicit their friends; however, that can be overcome with money. You can expect to pay a headhunter a big commission to find a candidate who will be less predictable than the ones your own employees will recommend. If you were to offer a bonus of just half that for employee referrals ($10,000 for a $100,000 hire, say), employees would feel richly rewarded and highly motivated. Justified as they would be, we have never, ever seen referral bonuses that high. Nor have we seen a single study quantifying the difference between $2,000 and $500 bonuses, both of which are common. But we do know referral bonuses work.

By the way, hiring managers are a special case when it comes to employee referrals. In every program we've seen, as the hiring manager you are not eligible for referral bonuses; you are expected to lure former employees from your network to your current team. It's not uncommon for managers to be asked, when interviewing, about their networks of programmers and their ability to hire from their own pool. Like many job expectations, doing so is not bonusable.

It is important to keep in touch with peers and former employees. In fact, a large number of employers, possibly a majority, would not hire you if they knew you hadn't stayed networked with the best of the developers with whom you've worked throughout your career. That said, don't solicit developers from the last company you worked for. Even if you didn't sign a nonsolicitation agreement, it's bad form. But stay in touch, connect with your former colleagues on LinkedIn, be friendly, let everyone know where you are, and let them contact you. That is OK. So is nonspecific recruiting like posting an update on your LinkedIn profile and other social networks to broadcast your need.

One note of caution: While the rule is that good people recommend good people, always, always, always listen to your "gut." Ron recalls,

"I progressed through one employee's referrals from one of my best hires to one of my worst. My employee had been stellar at his job, so when he told me that his referral candidate was even better, I was skeptical; but after interviews I thought she would at least be good. She was better. She knocked my socks off. So when I next needed a hire and the guy had another 'even better than me' candidate, I ignored the odd feeling in my gut and chose his candidate over another that my team and my gut really liked. My entire group suffered when he turned out not to be stellar—and in fact was not even competent; it was a month of pain until he made it easy for me and left the company." The rule of thumb: Trust your gut about the candidate, not about the referrer.

One more note of caution: You must avoid cronyism and the appearance of cronyism. Your job is to make great hires of people who are a superb fit, not to hire a team of your friends. Your hires should be the best candidates. Yes, that's a subjective decision, and you're the one making the decision, and you have experience with your candidates that no one else in the organization has, and all that is worth something. But if you have a history with a candidate, you should be explicit about communicating why that candidate is your choice; you should share the experience you had with the individual that makes you confident he is the right hire, especially in the face of a competing candidate who interviewed well. As always, communication is your "job one" as a manager, and maintaining interpersonal trust is essential.

Effective Recruiting

Our experience with advertising tech jobs in print media has been to do it rarely, only when the company has a large number of positions to fill, ideally when the number is large enough that it leads you to hold a recruiting event (perhaps in connection with a tech conference nearby) so that the advertising can focus on getting candidates to the recruiting event.

One way to be cost-effective with your recruiting budget, if you have time, is to tier your efforts. Give employees a two-week lead to bring in candidates (and you might make the bonus higher for candidates they bring in during that initial period, possibly saving you the additional work of the next steps). During that time, scare up candidates yourself from your own network of former employees and colleagues.

Simultaneously post your job on your company's Web site to ensure that candidates can get a clear idea of what you're looking for and can feel confident in approaching you. If your company uses an applicant tracking

system such as Greenhouse or Lever, you will post your job via that tool. Since smart candidates have learned to set triggers on such tools to notify them when an appropriate job has been posted, you may find you get traction from this move alone.

After two weeks, turn the recruiting over to your internal staffing department recruiters; if they're any good, the number of résumés you will now have to read will multiply fourfold if not tenfold. Simultaneously, brainstorm professional organizations, meetups, and social-media professional groups to which you can post your need, some of which might additionally have job boards that you can leverage.

If you still aren't finding your hire, advertise on low-cost classified networks such as Craig's List, AngelList, Indeed, and LinkedIn. Your résumé reading list should increase again.

Finally, go to a few contingency headhunters. If they're good, your résumé pile will grow by only a small number, but the candidates will be perfect and you'll owe the recruiter a lot of money. (If your organization has a lot of money and little time, skip directly from employee referrals, head start or not, to headhunters.) If you find yourself working with a headhunter who doesn't "get" what you're looking for—who sends you one inappropriate résumé after another—drop that recruiter. Your time is too valuable.

Recruiting Tips

There are a few other items to pay heed to when recruiting full-time employees.

First, given that your most important jobs are to recruit and retain the right people, the staffing and HR departments are the most important groups in your company to bond with. Staffing will play more of a role in your success than any other group. Make internal recruiters your friends. Their care and feeding should be a top priority for you.

The typical staffing department is wildly understaffed. And with your technical positions to fill, you're at an additional disadvantage, since 95 percent of recruiters barely have a technical bone in their bodies, truly struggle to make sense of your list of required skill sets, and don't really understand the people you're looking for (even if they're good at finding them!). Internal recruiters are typically either touchy-feely HR people who happen to demonstrate a bent for external networking, or marketing people who wish their colleagues would stop typecasting them as HR people. Either way, they have little in common with you.

Make it your mission to make these people your friends. Drop by. Be a friendly face; bring them a smile, coffee, a stuffed animal, or perhaps food (but not to recruiters who are dieting); learn to explain what you're looking for in their lingo; ask about their kids and their hobbies and their interests; help them to figure out where to look for the candidates you're seeking; review résumés with them to show them the words and phrases that jump out at you (both positive and negative). If they ask you for anything, get it to them by return mail. If they give you résumés to review, return them within hours, commented and prioritized by desirability against your criteria. Don't ever make them track you down. Make their job easier in every way. Be their best friend. Figure out how to genuinely like them, and they'll like you back.

Don't ever assume, when you don't hear from them, that they're working on your hire. Ask them how it's going. Ask if there's anything you can do to help, or if there's any additional information you can supply that would help. Follow these suggestions, and you'll be one of their favorite managers.

Staffing may be located elsewhere. Find excuses to wander by. Schwab's staffing department was on the same floor as its cafeteria, making it easy for Ron to drop by before or after lunch, or when visiting the vending machines at snack time. At Razorfish, Ron formed deeper bonds with the team upstairs when his recruiter was relocated to an office there. It worked. His job requisitions got the attention they needed.

Mickey has used contract recruiters quite successfully at Brøderbund and Gracenote: "When I had a bubble of critical positions to fill, I worked with HR to bring in a contract recruiter who can focus on those positions. Contract recruiters work for a lower fixed fee or on an hourly basis, which can greatly reduce the recruiting costs and result in more progress by focusing strictly on the critical positions. Like programmers, you can sometimes find contract recruiters who are passionate about their areas of interest. You can work closely with these contract recruiters to make sure they thoroughly understand the ideal candidate profiles and the critical skill sets, and they work very closely with hiring managers to optimize their time by presenting only highly qualified candidates. At Brøderbund we had one contract recruiter who became a specialist in locating great multimedia talent. He immersed himself in the technologies and prowled the technical forums and special-interest groups looking for talented individuals. He became almost obsessive about looking for and being successful at finding talent. I saw him a few years ago at a SIGGRAPH trade show where he was working for Intel recruiting 3-D graphics specialists and still obsessed by his mission. He was a special recruiter."

Finding such passionate recruiters is hard, but when you do, your life will be easier and your recruiting almost a pleasure.

There's another class of headhunters besides those who work on commission, but they mostly don't apply to you. Retained recruiters are ones you retain and pay regardless of whether they find the candidate you hire. Retained recruiters mostly work on senior and executive management positions, where they specialize in having senior-level networks they can access to find candidates. Sometimes they specialize in or undertake searches in secrecy, to avoid putting the word on the street that someone senior is being replaced. For programmers, though, you'll almost always pay recruiters a commission only if they find the right candidate for you—a contingency search.

Recruiting Contractors

Recruiting contractors is different from recruiting employees.

A large organization may well have a list of six, eight, or ten "preferred vendors" of contractors through which you will be required to hire contractors. One or more of them will be designated as "pass-through" vendors; should you find independent contractors you want to bring in, you'll typically be required to bring them in through one of the pass-through vendors, which will provide payroll services and bill you enough more to pay taxes and take a cut themselves.

If you're lucky enough to have this system in place and enforced in your company, your phone won't ring except with legitimate business. You'll never be plagued by the swarm of job shops trying to be the one to find you contractors. On the other hand, there goes your largest source of free lunches and presents at Christmas. The real downside is that you'll have to leave behind the contractor recruiters who have served you so well in the past and whom you have long cultivated to bring you great people (and buy you lunches).

If you don't have a preferred vendor system in your company, ask your programming manager peers and colleagues for referrals of good contractor houses and recruiters to work with.

Go out of your way to find a "boutique" contracting house that you can trust to find especially skilled contractors when you need them. Mickey says: "While at Gracenote I found a contract house that always seemed to find exactly the right 'specialty' contractor when I needed one. They had access to a network of contractors and had them categorized very well, because they found me a contractor in Seattle, a contractor in Toronto, and many

local to the Bay Area with exactly the specific skills I was looking for at the time. These skills were not simply programming skills; they were as exotic as experience with Japanese *and* Korean Morphological Text Matching, or experience in implementing UPnP servers (when the technology was first emerging), and others. I was amazed at how quickly they could respond to my seemingly exotic requests for contractors."

Preferred or no, cultivate those relationships. You want these folks to see your needs as their top priority and to think of you when their best people become available.

Of course, the best place to look for contract talent is within your own network. LinkedIn provides instantaneous and always updated access to your network, though it is no real substitute for a carefully cultivated database of your contacts that you maintain throughout the years.

Mickey uses LinkedIn for his close set of personal contacts (hundreds, not thousands), but also an address book application that has the ability to store preset fields as well as free-form data that is word indexed. He uses this program religiously to maintain all his contacts, including those whom he would not dream of inviting into his personal LinkedIn network. "This has been one of my best weapons in accelerating the recruiting process for employees and contractors."

Reviewing Résumés

If you're lucky, all that recruiting will result in a flood of résumés. But how do you identify the potential stars in a stack of résumés?

Reading résumés is an art. You need to look for your requirements expressed in someone else's words. You need to read between the lines. You need to connect the dots. You need to read the words and imagine the activities the candidate would have had to undertake to be able to write those words. You need to think through whether the range of experiences candidates have had will have readied them for your company and your position.

> *College degrees don't impress me, and lack of school doesn't scare me (see: Jobs, Steve, and Gates, Bill). At some point, when a person is far enough removed from school, the degree is all but meaningless. Experience is what matters most.*
>
> *—ERIC MULLER, Software Architect and VP of Technology, San Francisco Bay Area*

Pretty soon, you have to make a value judgment regarding what requirements are truly required, how experienced a candidate really has to be with each of those technologies, how many applications you need to see, and how big they need to be to prove a candidate truly has the skills you're looking for.

> *I assume that a good candidate rewrites their résumé and cover letter after reading my Web site. Checking them out on LinkedIn tells me what their résumé really says.*
>
> *—BRUCE ROSENBLUM, CEO of Inera, former VP of Software Development at Turning Point*

If you're seeking arcane and unusual skills, the pickings may turn out to be scarce. You'll have to decide whether to redouble or rethink your recruiting efforts in order to find the candidates you need or to scale back your expectations and plan to train. Keep in mind that though you can train FTEs, you should expect contractors to have each and every skill you need, coming in the door.

> *I want people who can write, because we spend a lot of time writing to each other. We're writing e-mail or documentation. We're writing plans. We're writing specifications. I want to know that the people on my team are capable of doing that, and that turns out to be a really difficult skill. So I would actually rather see people start as English majors than as math majors to get into programming.*
>
> *—DOUGLAS CROCKFORD, Inventor of JSON, Software Architect, and Entrepreneur*[11]

As you read résumés, jot notes on your copy (not on an original, since you want other interviewers to reach their own conclusions, not base their judgments on yours). Highlight the skills and tools you're looking for, where they appear. Draw arrows to gaps in employment history, so you can follow up with a question. Circle spelling errors, bad grammar, and sloppy formatting; you may end up making a decision between two candidates based on knowing that one can write well enough that you won't have to review every word. Note where candidates have changed jobs frequently; if you're looking for someone to stay on your team long-term, you may need to formulate

11. Quoted in Peter Seibel, *Coders at Work: Reflections on the Craft of Programming* (Apress, 2009), p. 124.

a question that elicits why a candidate jumped around. And write questions on the résumé as you're reading it (e.g., "What was your role in this accomplishment?" "What part of this project did you do?" "Why were you at this company for such a short time?" "What was the result of this effort for the company?" "What was the most difficult aspect of implementing this technology?" "What technologies and tools did you use on this project?" "What language did you write this in?" "What was the toughest challenge you overcame on this project?" "How did you learn this new skill?").

> *I often look for people that have done a lot of stuff on their own that wasn't asked of them. Not just their school project or just what their previous employer had them do. Somebody who was passionate about something and had some side project. How did they maintain it and how serious did they get with it? Or do they do a lot of quick hacks and abandon them?*
>
> *—BRAD FITZPATRICK, Founder of LiveJournal and Chief Architect at Six Apart*[12]

Résumé reading is a skill in which new managers will find it helpful to be mentored. Ask around to identify experienced and talented interviewers and hiring managers. Ask if you can help them read résumés for their next hire. Few managers will turn down that offer since even those skilled at it find reading résumés a thankless, but critical, chore.

You will find the résumé-reading checklist in the Tools section useful.

Narrowing the Field

In a slow economy or if you're hiring into a hot company, you may still have more candidates than you can interview.

> *IQ-like questions and quizzes are stupid.*
>
> *—DAVE WILSON, Software Architect, San Francisco Bay Area*

One way to narrow the field is to send candidates a programming challenge. Work with your team to identify a coding challenge that requires

12. Seibel, *Coders at Work*, p. 77.

skills consistent with your team's needs, has a correct answer, and should be able to be coded in a reasonable amount of time. Ask candidates to send you their answer and their code. Or leverage a platform like HackerRank that is purpose-built for this use.

When you get results, interview the candidates who submitted correct answers and whose code shows the kind of thinking, rigor, and documentation you expect.

A suggestion worth pursuing is to do the hands-on programming challenge live using WebEx, IM, or a Web site like typewith.me or sync.in that allows you as the interviewer to watch the remote candidate type. Even better, if you leverage pair programming to any extent on your team, is to pair one of your team with the candidate. It will give you a virtual hands-on feel for candidates before bringing them in for in-person interviews.

What ultimately narrows the field for us is simple: careful screening.

Preparing to Interview

Once you've got candidates who look like they might be a fit, it's time to interview.

The first interview is by phone, a screening interview. You need to find out

- If the candidate is still interested
- Whether the candidate is interviewing with other companies (and what the time frame is with those companies, whether the candidate already has other offers, is considering them seriously, and when a decision must be made)
- What kind of job the candidate is looking for
- What the candidate considers to be their areas of expertise
- What compensation is expected
- Why the candidate is looking for another job
- The candidate's availability to start working for you
- Whether the candidate is willing to commute to your location if working in your offices is a requirement

Ask candidates to describe in detail what they have worked on, both for you to gain confidence that they actually did what they said, but also to know that they can explain what they have done and what they know. Drill down into one or two of the accomplishments they cite to confirm that they have the skills you need.

I avoid prima donnas. One candidate told me he only needed to work two days per week because he could do in 16 hours what everyone else would do in 40. No, thank you.

—BRUCE ROSENBLUM

For a highly specialized technical position, you may want to choose a highly technical team member to conduct a second screening to test expert knowledge the candidate claims to have and that you need.

Once you confirm that candidates are credible—that they appear to meet all your criteria—you'll assemble an interview team. Then bring in two or three leading candidates for one or more rounds of interviews with your team, your colleagues, and perhaps your boss.

The job description you prepared earlier, such as our sample one in Figure 3.1, should provide all the criteria you'll use to qualify a candidate and measure one against another. The challenge is to remember to test every candidate against all those criteria, and then keep track of how the candidates stack up against them and each other. Mickey long ago came up with the spreadsheet format in Figure 3.2 to help him do that. Enter your criteria into a similar spreadsheet to keep track of your candidates and their qualifications.

Plan a strategy for who will pursue which skills and qualities, and additionally who will help you sell the candidate on joining the company. (It works both ways.)

The interviewers you assemble may include

- You
- Your HR or staffing person
- Programmers who are the technical leadership on your team
- Programmers from related teams with whom your hire will need to interface
- Your UI designer
- The product manager
- The project manager
- Another development manager or two (particularly if you're green at hiring; another manager's observations and feedback can help you with what to look for and how to look for it)
- Others in the business from whom the programmer will get requirements or collaborate around product and support issues
- Your boss (or even your boss's boss)

Principal Programmer Interview Summary	Bill Smith	Cathy Llu	Arnold Lai	Lucy Miller	Andy Jones
Received résumé on					
Phone screen on					
First interview round on					
On time, early, or late?					
Second interview round on					
On time, early, or late?					
Bachelor's Degree (optional)					
Minimum 8 years programming experience					
Wrote first program ever in (year, language)					
Wrote first professional program in					
Experience with what languages					
Experience with what databases					
Minimum 3 years .Net programming experience					
Wrote first .Net program in					
Most recently wrote for .Net in					
Minimum 3 years SQL Server programming experience					
Wrote first SQL Server program in					
Most recently wrote for SQL Server v. (???) in (year)					
Web application architecture and design skills?					
Ability to analyze & improve scalability and performance					
Experience scaling high-volume, information-rich Web apps					
Fast, clean, efficient coder?					
Refactoring skills					
Has defined development and engineering best practices					
Experience leading and mentoring other developers					

Figure 3.2 Principal programmer interview summary
(continued next page)

Principal Programmer Interview Summary	Bill Smith	Cathy Llu	Arnold Lai	Lucy Miller	Andy Jones
Communicates designs effectively					
Listens					
Critiques others' designs					
Writing skills					
Customer Experience empathy/ awareness/design sense					
Intangible qualities					
Energy					
Flexibility					
Self-direction					
Smart					
Articulate					
Passionate					
Fit in with team					
Overall desire to work at our company					
Experience w/algorithmic design, coding, trade-offs					
Search/information retrieval					
Analytics, data warehousing, and business intelligence					
Information visualization					
Web services					
Sent us a follow-up thank you?					

Figure 3.2 Principal programmer interview summary

In one company, Ron's CEO asked to interview every candidate to whom Ron thought he would want to make an offer (provided the CEO's travel plans or other conflicts did not hold up the hiring process); he wanted a head start with new hires for his goal to know everyone in the company, considered it a "touch test" to build confidence in his senior managers' hiring IQs, and offered the gift of his time to assist with the sometimes challenging task of luring highly qualified developers who were choosing among competing offers.

On the other hand, earlier in his career at a much larger company, Ron's midlevel boss gave him carte blanche to hire without the boss interviewing a single candidate. At a third company, not only his boss but also his boss's

boss were on the interview schedule. You'll likely have managers of every stripe as well, but if they interview, they'll almost always want to be last to do so; most will want to see just the "keepers."

Mickey and Ron would both rather have more interviewers than fewer. When hiring FTEs, Ron typically selects two teams of four to five interviewers for a first and second round of 45- to 60-minute one-on-one interviews. Get to know how long your interviewers prefer for an interview. Some will be like Ron, who wants a full hour with candidates, whether his or another manager's candidates; others are happy with 30 minutes and uncomfortable with even five minutes longer than their requested time.

Programmers are critical interviewers. They will have to work and team with the new person. They also likely know the skills and experience that are needed or missing better than anyone. But programmers are also generally the least prepared to interview. You need to spend time with new interviewers to go through the technical and team qualities you want candidates to bring. Then you can work together on questions and exercises they can pose that will help reveal the candidate's facility.

Assign areas of focus for your interview team members: the various technical skills you need; analytical, problem-solving, communication, and interpersonal skills; and résumé red flags and omissions. Make sure you have interviewers who will ask technical questions that demand technical answers. Divide up the candidate's projects and companies among your interviewers, so that someone digs into the details of each one. And divide up the qualities you're looking for, both to ensure that among your team someone is pursuing understanding of that quality as well as to avoid a day of interviews where everyone asks the same questions. A wiki page or other collaborative online space is perfect for letting your team sign up for those areas about which they feel most competent or most passionate.

All that preparation will help ensure that your interviewers are prepared. Too many interviewers in too many companies read a résumé five minutes before the person comes in—or on their walk to the lobby to pick up the candidate—and end up contributing a fraction of the thoughtful, in-depth understanding that a well-grounded, well-thought-out interview should produce. The entire interviewing team must be clear on the need for this hire, with whom the new hire will work, and what the new employee will be tasked to contribute. Then each interviewer can create initial specific questions, using the résumé to guide further questions when probing into the candidate's experience and background. Interview training is something that is not often given to employees, but the cost of hiring the wrong people far outweighs that time and effort.

When you're hiring programmers, you have to get at their ability to code. It's essential to answer questions that elicit a picture of their understanding of programming. It's critical that you ask them to do some design and to write some code.

> *I have had a couple of profound wake-up-call cases lately that pointed out how important it is to ask a programmer candidate to write code. In both cases, we had candidates whom we considered to be A or A+ level matches to what we were looking for. They'd had all the right experience, listed just the right skills for the job, seemed to have the right people skills, and genuinely seemed like nice and well-rounded individuals. But then, almost as a formality, we asked them to write some code. The term deer in the headlights best describes the result. Both these guys fell flat on their faces. We couldn't believe it. They did so badly that it caused a stir throughout our whole department and led to multiple discussions about how this could have happened. Long story short: What we learned was that asking candidates to write code and to answer questions about code is absolutely critical.*
>
> —*STEVE JOHNSON, VP of R&D*

Encourage candidates to bring a portfolio of projects—documentation they've written, designs they've created, samples of their work, and even demos on their laptops or online that demonstrate their prowess.

> *I invite the candidate to bring in a piece of code he's really proud of and walk us through it. I'm looking for quality of presentation . . . how effectively they can communicate, that's a skill that I'm hiring for.*
>
> —*DOUGLAS CROCKFORD*[13]

> *Ask a candidate to bring along some of their source code. Inspect their code, and you'll know if they are any good.*

13. Quoted in Seibel, *Coders at Wor*k, p. 129.

Then ask the candidate to show you an app that they built. Evaluate the user experience.

—DAVE WILSON

Sometimes doing this can have unexpected effects. Ron's youngest hire was, at Apple, an intern just out of high school. Ron had heard, through a connection, about the young programmer's prowess but wasn't sure his team would embrace a high school kid. As it turned out, the young programmer brought in samples of random-dot wall-eyed auto-stereograms; he had read about the technique of hiding 3-D scenes in images that at first glance appear to be nothing but random dots, and he'd figured out how to reverse-engineer a program to create them. As Ron watched his team squint wall-eyed at the samples pinned to the team wall, willing the 3-D images to emerge, he knew he had a fit.

Pair programming for half an hour during an interview will save everyone's time.

—DAVID VYDRA, Continuous Delivery Advocate and Software Craftsman, TestDriven.com

As you set up a morning or afternoon or day of interviews, you need to plan for someone to be the first to greet the candidate, and someone to see them out. As the hiring manager, you're a strong candidate to fill at least one of those roles. Taking the closing role can be a great opportunity to debrief candidates on their perceptions of your company and your team, try to correct any misperceptions, and send the candidate off with a positive impression.

Ron also tries to have a trusted strong interviewer lead off—and report back at once if the candidate seems at all a bad fit. There's no use wasting the candidate's or the team's time further if you determine up front that the match isn't there.

If possible, take the candidate and at least part of your team to lunch. The interactions you'll see will be priceless for making a decision about whether the candidate has "team fit."

Mark Himelstein, an Interim VP of Engineering in the San Francisco Bay Area, prepares his interviewing teams by going over

- What the person is being hired for
- Issues/areas to be covered (making sure that someone is covering the basics)
- How to sell the company consistently

He notes, regarding selling the company, "I have used role-play to teach developers how to sell the company consistently. We agree on the point I want each to make, then I have them use that point to sell the company to a colleague in 120 seconds. I have the team offer critiques to their peers on how to improve the pitch."

Finally, do candidates a favor by presenting them with an interview schedule when they arrive that includes times and interviewers with their titles and (should the candidate want to follow up) e-mail addresses. Having your greeter not only present it but draw a verbal picture of the interviews ahead and who the interviewers are will put your candidate at ease and get logistics out of the way so that you can all focus on fit.

Take a look at the sample interview schedule in the Tools section.

Interviewing

Take notes! Walk into interviews prepared to take notes on what candidates say. It's amazing how a series of candidates will blur together without notes to tell them apart.

Make time before the interview to prepare your questions. Write them down. Carry them into the room with you.

Make eye contact (and make sure the candidate can make eye contact with you). Make note of what candidates communicate nonverbally; how they comport themselves; whether they're on time, early, or late; and afterward whether they send a thank-you. And make notes about what your team members, colleagues, and boss have to say about the candidates.

At the same time, don't be so busy writing down what candidates say that you don't notice who they are.

Ron makes it a practice never to interview a programmer in a room that doesn't have a whiteboard; he looks for candidates' willingness, even eagerness, to get up and explain to him how they approached a problem they faced in a previous company, to explain the architecture and design of one or another of their previous projects, or how they would face one of his team's problems now. It can help differentiate the talkers from the doers.

> *I like to talk about design patterns, like how would you design something. Candidates should be able to identify all the parts of objects. For example, if you were designing a game of blackjack, you have cards, hands, and*

> *players. They should be able to identify properties of these objects and their relationships. When would they use inheritance? When would they use "is-a" versus "has-a" relationships? There may be more than one correct answer, but the approach should be workable. I find I am usually willing to give answers and instruct so that it is not a "gotcha" interview, but more of a conversation. The goal is to determine how well we work together on a problem.*
>
> *—PAUL OSSENBRUGGEN, Senior Staff Developer*

You're going to want to know the answers to questions like these:

- What aspects of your last job did you most like?
- What were your colleagues and your management like?
- Tell me about some of the things you and your supervisor disagreed about.
- What led you to leave the companies you previously worked for?
- What attracts you to our company?
- Why are you looking for another job now?
- What do you want to get out of your job?

Learn to ask questions that are open-ended—that candidates can't answer with a yes or no—like these:

- Tell me about. . .
- How were you able to accomplish . . . ?
- What was your role in . . . ?
- If you had led the development effort on that project, what would you have done differently?
- What best practices are you most fond of?
- What are your strongest technical strengths?
- What are your strongest nontechnical strengths?
- If you think of the fabric of programming as triangular, with the points representing design, coding, and debugging, tell me about the part of the fabric on which you would most like to spend your time.
- Where would you place yourself on a continuum where one end is developing gnarly algorithms and the other is developing customer-focused UI?

- Imagine a line. One end is leadership. On the other is teamwork that's so fully collaborative that leadership is totally shared and no one on the team would be able to identify a leader. Where on that line would you place yourself?
- How would your manager describe you?
- Tell me about your comfort level with asking for assistance from others.
- Where do you fall on a continuum that ranges from highly structured, where your tasks are spelled out completely, to one that is entirely free-form and you have to make decisions, often without having all the information you'd like?
- How do you like to be managed?

Ask for examples:

- Think of a time when you knew you could not make a deadline. What did you do?
- What was the most interesting problem you faced in a former project? How did you solve it?
- Tell me about a time when you. . .
- Give me an example that illustrates your leadership style.
- Think for a minute about the most stressful situations you've been in at work and tell me about the one you think was most stressful of all. What did you do to deal with it?
- Have there been times when you needed to formulate a new solution? Tell me about that time, and about what you devised.
- Tell me about a best practice you played a role in getting your team to adopt.
- Describe a time when you displayed extraordinary initiative.
- Have you worked with a UI designer [product manager, business analyst . . .] to translate customer needs into technical requirements? Tell me about that collaboration.
- Tell me about a time when your manager was annoyed with you or with your role on the team. How did you respond?
- Think about the teams you've been part of and tell me about a peak teamwork experience. What contributed to making that memorable?
- What have you done when you've had far too many tasks assigned to you than you can handle? When that's been the situation and you could see yet another task coming your way, what did you do?
- Tell me about a time when you successfully persuaded your manager or your team to adopt your position.

- How have you handled making formal presentations in front of large and small groups? Tell me what that was like.
- Have you had to present technical solutions to highly nontechnical audiences? How did that go?
- Describe a time when you advocated creating a better customer experience.
- Describe a situation in which you had to tear down code and redesign and recode from the ground up.

I'm no longer a full-time developer and my skills have gotten a little rusty. When I'm interviewing someone, I focus on basic concepts. If I can stump them, they are done.

—ERIC MULLER

Get candidates to give you details. What role did they play in the projects they cite on their résumés? Get them to tell you how they accomplished the achievements they claim. Ask them about the most difficult problems they had to solve in accomplishing them, and ask them to walk you through their solutions.

I have always found that getting candidates to talk about a project they have done in detail brings me the best info: how well they communicate, what roles they actually had, do they have a big picture about what they did, do they really understand the technical details.

—MARK HIMELSTEIN

Think of a problem situation that vexed a team you've managed (and would be appropriate for candidates to solve) and ask them to suggest how to solve it.

But don't ask leading questions. Do truly make your questions open-ended; give candidates room to answer as they think appropriate.

Ron writes questions with the intent not only to understand what candidates know and how they think, but to learn something from each and every candidate that he has the opportunity to interview, no matter for what job. "We're interviewing these candidates because we think they can bring something to our company. My attitude is to figure out what they know that I don't (yet), and to start learning from them. Sometimes it's technical; other times it's

how other companies or managers have handled challenges or, from college hires, how the computer science curriculum is being taught these days."

If I can't learn something significant from a candidate in an hour's interview, it's almost certain I will decline to hire the person.

There are also key logistical questions you'll want to ask:

- What will commuting to our offices from where you live be like for you?
- How much travel do you like (and how does that fit with the amount of travel I foresee in the job)?
- What are your compensation expectations?

One final note on preparing questions: Keep them legal. Your questions should never in any way suggest or encourage candidates to tell you about their

- Marital status
- Parental status
- Age (particularly whether 40 or over, but don't go there with anyone)
- Current salary or compensation (at least in some parts of the United States and the world)
- Ethnicity or nationality
- Disability or perceived disability
- Religion
- Sexual orientation

Making the Decision to Hire a Programmer

As each round of interviews completes, get timely feedback. Our experience is, unfortunately, that you will likely have to remind (even hound) your interviewers to give you their feedback. You need to get it the same day, at the latest the next day, while it's fresh and memorable, and also because, if you like the candidate, you want to take action, whether to bring the candidate back for another round of interviews or to make an offer.

Debriefing your interviewers, as a team, is critical: Not only is it an opportunity for you to understand the team's perspective, but for them, observing how others

can perceive different aspects of the candidate can help each team member improve their interviewing skills.

—PHAC LE TUAN, VP of Engineering and CEO, Silicon Valley

Ask your team to look for indicators and for red flags. An indicator might be a candidate who has programmed in a lot of languages. It's a rule of thumb: The more languages, the better the programmer. On the other hand, red flags might be that the candidate arrived late for the interview, took a call during the interview, never seemed to establish eye contact, was sharply critical of former managers and former companies, arrived knowing nothing about your company or your products, was unable to explain a previous design, didn't show interest in the work you do, didn't share anything from which you could learn, or didn't follow up with a note or e-mailed thank-you.

Will the candidate be able not only to contribute to the current need, but can you anticipate their skill set contributing for years to come? Make sure you're not hiring a narrow fit for a short-term task that, when complete, will leave you with a long-term problem requiring that you either train or terminate.

Weight the feedback from your interviewers. Some interviewers' feedback is worth a lot more, whether because they know the technical hiring requirements cold, or because they have proven themselves to have a great feel for hiring, or for one of a dozen other reasons. Think about the weighting *before* you hear the feedback.

If you're dithering, don't hire them.

—STEVE BURBECK, Manager at Apple, IBM, a small wholesale company, two start-ups, and a research institute

While dismissing candidates as not appropriate is easy, making a decision to hire is often difficult. Be clear with your interviewing team that the decision will be yours. (Actually, it will likely be yours in concert with your boss and HR.) It is *not* a consensus decision.

Sooner or later, you'll find yourself convinced that you have a stellar candidate, and every interviewer is on board but one—an interviewer who is adamant that hiring the candidate would be a mistake. Listen carefully to that person's feedback; it's possible the feedback is dead-on. It's also possible the interviewer is not looking at the same criteria you are. If you make it clear you're taking input (not looking for consensus), and you bring all your reflective listening skills to bear so that the person feels heard, you're likely

on solid ground to hire the candidate based on all your other feedback that says "stellar." On the other hand, if the interviewer, or worse, your entire interview team, gets it in their heads that it's a consensus decision, you'll never break a meeting deadlock without bad feelings, very possibly not only from the one person who demurs but from the team as a whole.

A quick meeting of all interviewers can be useful; a discussion can prompt memories and *ahas* that had been only subconscious. But meetings can also communicate to interviewers that they have more say than they do. And going on the record with one's input can make it more difficult for an interviewer to give you the power to make your own decision. These days Ron tends to get feedback one-on-one with each interviewer, ideally in person or by e-mail or phone.

You need to learn not only to listen to others who interviewed, but to trust your gut about what you heard and saw. At one company, Ron let his team talk him into hiring a candidate when all his internal signals were saying no. The candidate, who wanted onto the team for all the wrong reasons, turned out to be mediocre. While she in fact made some good project contributions, she never really fit in with the rest of the team and was at the top of the layoff list when times turned bad.

Ron made the first hiring decisions of his career at Apple, at a time when the company couldn't interview and hire fast enough. So it was memorable when CEO John Sculley, speaking to a full auditorium of Apple managers, urged everyone to hire carefully. His sage advice: "Hire people you want to sit next to, both tomorrow and a year from now."

Different organizations have different customs and practices around hiring. When Steve Jobs's NeXT Computer company hired technical staff, the decision to hire someone had to be unanimous; every person who interviewed the candidate had to agree that the candidate should be hired or they would pass (thumbs-up or thumbs-down). This led to some very intense interviews, and many of those who were hired survived grueling programming problems, one-on-five interviews, and a process that lasted many hours. The approach led to a team of extremely bright and talented members—but they were not that diverse. Make sure you clearly understand the culture you are working to staff.

To help you understand other hiring cultures, we suggest that you research some of the top technology companies to get some insight into how they work. Try Googling "interviewing at" and you'll get suggestions for several companies to review. There are some very interesting stories about interviewing experiences you can easily access online. Don't feel compelled

to copy them, but learn from the good and the bad that are painted in these stories to help mold your own hiring culture.

Call references.

With a candidate chosen, it's time to check references. Ask the candidate to provide you with a list of references you can call. You're going to ask for at least two peers and two former managers, with phone numbers and e-mail addresses. If you're hiring a manager, also ask for two former employees. Pick and choose to call at least one from each category.

To the candidate's list you'll add your own "back-channel" references. The candidate presumably listed colleagues who will all deliver praise and recommendations. What you're looking for are random others to corroborate that feedback but also to fill in gaps. You may know someone or have a teammate who worked at a company at the same time the candidate did. The shortest route these days is to search LinkedIn to identify whom you know who worked there when the candidate did.

Never be satisfied talking only with the references your candidate supplies. If they're a friend of the candidate, they often won't mention the candidate's faults—and everyone has them. Find an independent source—someone you know who has worked with the candidate as a peer as well as someone who managed or worked for them.

—DAVE CURBOW, User Experience Architect, Cisco

HR may volunteer to take care of reference checking, but you should always have at least two or three of the conversations yourself. After introducing yourself, begin by asking how and when the reference worked with the candidate.

Like interview questions, the best reference-check questions are open-ended. You want to know about the work that candidates did and about their skills, teamwork and collaboration, work habits, initiative, thoroughness, follow-through, reliability, need for supervision, ability to learn, strengths and weaknesses, and values and ethics. Ask for examples. Get the reference to be descriptive, to draw verbal pictures for you. Ask about any red flags that came up for you or your interviewers. Ask references where they would rank the candidate with the others on their team. Describe the job you're hiring for, and ask references whether they think the candidate is

a fit. Ask former managers if they would hire the candidate again, former teammates if they would gladly work with the candidate again.

We suggest you use a reference checklist like the one we've provided in the Tools section.

Making the Right Offer to a Programmer

Making the right offer to a programmer starts with timeliness. Every day lost is an opportunity for your candidate to discover, interview with, and receive an offer from another shop. During the dot-com hiring frenzy, every hour lost was an hour risked.

Don't be hasty but be quick.

But how do you know what offer to make? Determining the right offer starts early in the interview process with the question, "What are your compensation expectations?"

Even if it's legal where you live to ask candidates for their last salary—and it's not legal in a rapidly growing number of places—that information is less useful than it looks at first glance. Past salary equates with neither present value to you and your team nor market expectations. A candidate's last salary might have been exorbitant; salaries required to lure top programmers at the peak of the dot-com boom were downright unrealistic just a few months later, after the bust. Or it might be drastically below market; a programmer hired just before boom times probably didn't get raises to match the salaries of developers hired later. A programmer working for a struggling start-up may not have had a raise for years. Or a female or minority programmer may have suffered from recent—or even long-ago—wage gaps.

> *Salary compression is a fact of life. Don't let it make you miss a candidate.*
>
> —*MARK HIMELSTEIN*

Most programmers know if their last salary was out of line. In the post-boom-time case, they may respond to the question of expectations with a number or a range considerably lower than their last salary. In the second, third, and fourth examples, their expectations may justifiably be a big increment from what their previous company got away with paying.

You need to be prepared, from the moment you begin recruiting, to know what range you can afford to pay. Be prepared, when you ask a candidate

for their expectations, to get a reply that is a request for the range you expect to pay. You need to know the answer to that question.

You may hear expectations that are out of your range. Or in sharing your range, you may hear back that your candidate was expecting more. Be frank: "Our base salary ranges just don't go that high. We can offer <options, bonus opportunity, special benefits>, but not that kind of base salary." If you end up cutting the phone screening short, you'll have saved your interviewing team a lot of time, trouble, and false hopes and given yourself back a little time you can use to scout for other candidates.

Be aware, though, that while you'll run across an occasional candidate with an inflated sense of self-worth, more often you're getting a signal. It may be that you've overshot and are interviewing a candidate much more qualified than you need. On the other hand, you may be hearing a signal that salaries have moved. If the latter is the case, as you hear high expectations from subsequent candidates, you may kick yourself for dismissing the first.

Compensation is not just a salary number but a package. While some candidates won't lower their base salary expectations even for a great package that includes outstanding options, exceptional bonus potential, unusual and special benefits, or the like, some will. If you're excited about hiring them, then sell them on the company, the position, your team, and your package.

Every programmer's motivation is different.

If candidates are wary about telling you their compensation expectations, give them some time to think about it. Let it go during the interview, but follow up afterward if you're interested in pursuing them for your position. If you don't, you could end up negotiating with yourself by putting an offer on the table that is inappropriate—either too low or too high. In either case, you're now at a disadvantage in formulating the right offer for the candidate. Make sure you get wary candidates to clearly state their compensation expectations and consciously decide they are acceptable before moving forward in the hiring process.

Once you know what they expect and that it's a match for your range—and once all the feedback from interviewers and references has led you to want to hire—you need to think through a specific number and package. If the candidate's expectations are low, you may be tempted to make a lowball offer. We think you'll regret it.

We think your salary number has to be in line with the going rate in the market. The last thing you want is for a candidate—realizing, just after

arriving, that many if not most companies are paying a lot more—to keep their pipeline of job opportunities open, leading to a departure for a better one after only a short time on your team. If possible, work with your HR department to verify your target number with an industry standard salary survey service such as Radford Surveys.[14] Radford, and other similar services, provide benchmark data comparing companies situated similarly to your own. This will help you determine if you are paying competitive market rates for the new hire and provide the ammunition you may need to help convince your management to hire someone for more than what is budgeted for the position.

We think your salary number also has to be in line with salaries you're already paying comparable programmers on your team and across your company. You can exhort your team all you want to keep their salary numbers to themselves, but sooner or later they'll all know what the others are making. Bad inequities will lead, at that point, to carping, bitterness, disgust, and an exodus of your best people.

I'd rather do big bonuses than out-of-range salaries.

—MARK HIMELSTEIN

However, sometimes you need to bring in a programmer who doesn't fit your current internal equity. You hate to do it, but you may choose to because you need the programmer desperately, or because your programming staff, in general, is paid below market rates. By bringing someone in above your internal equity rankings, you have ammunition to bring to HR and your management to try to increase the salaries of the top performers on the staff you already have. This is a painful tactic, but it's sometimes necessary to satisfy your short-term hiring needs.

The one exception where equity can be less of an issue is with geographically dispersed teams, since salary decisions may have been made based on geographical differences in both market and cost of living. You can get an idea of how to derive geographical equity—how to come up with a salary number for the same person in different locales—by using the research data at www.salary.com.

At some point you may find yourself hounded by management or HR to bring contractors on board as employees, a challenge made especially

14. Radford is a market leader in compensation intelligence: https://radford.aon.com/surveys.

difficult by compensation numbers. Contractors' hourly net is almost always higher than the salary you could equitably pay them, and often more than the salary and benefits put together. And if they have figured out a way to make their benefits work (e.g., getting benefits through their spouse's employer), such that they don't need those that come with an employee offer, converting them to employeehood becomes monetarily nearly impossible. On the other hand, contractors often face corporate policies, put in place to avoid tax and legal problems, that decree they can consult for only a limited period (e.g., six months or a year), then must be gone for six months to reestablish eligibility. If they like you and your team and your work, they may be willing to talk, at that point, about conversion.

Ron had one contractor who wanted $10,000 in salary above the rest of the company's programmers at his level. The company's pay system required breadth to qualify for the next software development grade, but like many contractors his skill set was narrow and vertical. Ron created a win-win by formulating a package (which required his getting sign-off all the way to the general manager) including

- A salary at grade level
- A guaranteed $10,000 in training over the next 12 months that could be used only for coursework in related technologies that the company needed and would also broaden the contractor's narrow skill set to a much more versatile and valuable one
- A promise to provide him with mentoring support from one of the team's most senior architects
- A promise to evaluate him, in 12 months, for possible promotion to the higher grade and a possible $10,000 raise to go with it

Complicated as that was (and hard as it was to get HR to go along, which was where selling it to the general manager came in), it reduced Ron's personnel budget, extended the programmer's tenure, motivated the new employee both to deliver and to expand his skills, and gave Ron a productive, valuable resource on the team who was gratified at the investment being made in him and on his behalf—for less than he would have cost as a contractor.

You'll forget what you gave them in ten minutes so don't get too worried.

—MARK HIMELSTEIN

Once he has a salary number, Ron will sometimes test it: "I need someone to start Monday. It's difficult adjusting an offer later—I need to have an offer that I know you would accept before I go to get it approved. If I were able to get you an offer by Thursday of $*xx*,000 in base salary, along with *n*-thousand options and the opportunity to make a 20 percent increment over your base in bonuses, would you accept it and start Monday?"

Questions like that will help you understand what a winning offer looks like, typically clue you in about competing interviews and offers, and often begin to build commitment on the part of the candidate—get them practiced in saying yes.

Before making and writing up the offer, you'll want to think about a start date. If the candidate is working, it will almost certainly be at least two weeks after giving notice. If you have flexibility, you may want to give candidates an opportunity to take a week or two between jobs. They'll come to you fresher, happier, and less needy.

Ready to make the offer? You'll actually make it in two ways: first verbally, followed by a written offer.

Staffing and HR organizations are often in the habit of making the verbal offer themselves, but we suggest you volunteer to present it. In our experience, managers who ask for the task are seldom refused. From the standpoint of selling the candidate on taking the offer, unless your staffing person is an exceptional salesperson, we think the implied relationship of having the hiring manager present the offer makes the stronger sell. (There is also an argument to be made for having your boss make the offer, if your boss is willing; candidates are, in general, impressed that someone senior would know who they are and call to urge them to take your company's offer.)

Your goal, the moment you have the offer approved and have mentally rehearsed your pitch at least once, is to reach the candidate voice to voice. "I have exciting news. I'm calling to make you an offer to join <our company> as a Senior Software Engineer for Database Development. The salary is the one we discussed, $*xx*,000 annually. You'll have an *n* percent bonus potential. And you will be awarded *n*-thousand stock options, 25 percent of which will vest after the first year, with vesting monthly thereafter. In addition, you'll get <a few great/unexpected/unusual benefits>. Will you accept? Can we set your start date for <date>?"

If you hear hesitation, try to find out what the objection is and resolve it.

If you've done your homework well and have a good read on what really motivates the candidate and have made that part of the offer, you should get

an acceptance without hesitation. However, some candidates always want to push the envelope by asking for more—even if you've met the compensation requirements they asked for when you had that discussion. When they hesitate or ask for more, don't get flustered. It is part of the hiring process. Take your time and determine what their objections to your offer really are and how deeply seated they are. Rarely, if ever, should you counter immediately with more than your standing offer. Mickey says: "Rarely is the hesitation really about money at this point. Often it is about the job title, or another desire that the candidate has surfaced since you verbally sounded out the offer. Title, office space, additional training, ability to attend technical conferences, and permission to work at home (at least occasionally) have all come up as I've presented offers over the years. The key is to stop and get the person to fully articulate these concerns; then you can see if you can address them."

Ron has promised unofficial days off (provided he is not reorg'd away from being the candidate's manager) when a candidate asked for vacation days that had already been set aside at the candidate's current employer. He has gone back through the approval process with a changed offer due to a just-arrived competing offer. He has clarified that telecommuting two or three days per week was absolutely acceptable (with the proviso of good communication, availability, and productivity). He has clarified to a known stellar candidate that starting at a later hour to accommodate a combined train/bike commute was perfectly acceptable. He has responded to questions about child care and flexibility for sick kids. He has reassured candidates that the job was not a dead end and has explained the opportunities for transfer and promotion that it could afford.

Many candidates will ask for a few days or even a week or more to consider your offer. It's seldom the answer you want to hear, but it is reasonable. Know beforehand how long you're willing to give them. Once you've arrived at a date, enter it into the written offer as the candidate's deadline to respond. And then stay in touch during that time.

Invite candidates with pending offers to team events, connect them to people on your team, and make them feel welcome and like they're already team members. Have someone senior—CEO, CTO, VP of Engineering—make a special call to the candidates to sell them on the position and the company and really connect with them, if possible. Often these calls are an opportunity to paint a more strategic picture of positions and how they fit into the organization and the company's goals. Help candidates understand why each position you offer is the best they could ever encounter!

We recommend that you try all of these things before considering sweetening the offer in any way. If you can do creative things that do *not* add to inequities for your staff, do that. One-time hire-on bonuses are often the easiest. Such one-time actions can be effective at handling salary issues without causing internal inequity problems, but they may not address the fundamental issue (too low a starting salary). In such cases you may reach an impasse and the candidate may not, in the end, accept the offer. You have to know how far you can and will go to make the hire and stick by that, even at the risk of losing a potential new hire. Sometimes you've just got to let go.

Benefits questions will likely come up. Let your staffing or HR organization answer them. Those people are far more versed in the intricacies (and the questions candidates ask about them) than you will ever be. Make sure your benefits package includes links to all online benefits information that is available externally.

The written offer will be drafted by HR and will include salary and benefits information, a limit on how long the offer is good, and a proposed start date. Make sure you get a copy, preferably to quickly review and approve before it is FedExed to the candidate. With the offer should go a confidentiality agreement (which should include a nondisclosure agreement, along with the caveat that the company will own any inventions created at work), forms, and collateral that portrays the company as the terrific place to work that it is.

Follow Up until the Programmer Accepts

Sending candidates a signed offer letter in a FedEx package for them to sign and return speaks volumes to how important the candidates are to you and how important getting the offer in their hands is. Often the letter will have been sent out in e-mail already, so this may seem like a needless expense. But you want to make sure candidates realize that you want their commitment to coming on board as soon as possible, and that your organization doesn't cut corners.

Also, since there is a FedEx tracking number, you can use that information to time your follow-up with the candidate to make sure the offer was received (you'll know it was delivered), and make sure that there are no other questions or issues. Use this as an opportunity to make sure the candidate is excited about joining you for the position they verbally accepted. Keep following up until your offer is finally accepted (or rejected).

The next chapter will elaborate on more follow-up activity, even after an offer has been accepted.

Summary

Hiring is one of the most important jobs, if not the most important job, you'll do as a programming manager. It needs to be treated with both care and purpose. Just as in a project, you're unlikely to get it right without getting good requirements down first. In fact, for critical positions or in a hot hiring climate you'll need to treat it like a project, setting short deadlines for yourself for each step: identifying candidates, phone screening them, moving them along or rejecting them, interviewing them face-to-face, making a decision, and presenting your offer. Leverage your team and your network to bring in prequalified candidates. Choose your interviewing team with care. Mete out assignments—interviewing objectives—to each member of the team, and make sure they understand how important you think their participation is to hiring the right person.

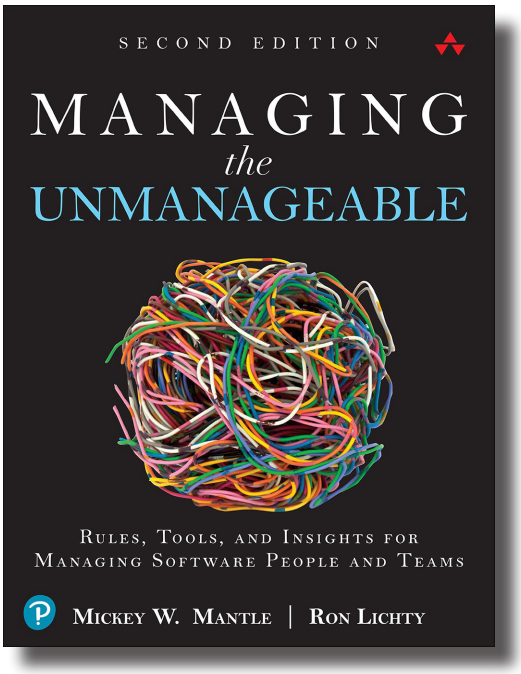
And remember that while hiring is an event, recruiting is a part of your job you should always have turned "on."

Tools

We have prepared a number of tools to assist you in managing your team. The spreadsheets and Word documents provide full examples you can easily adapt for your organization. See the Tools section, after the chapters, for the link to the Tools Web site, from which you can download the following tools:

- Sample job description
- Résumé-reading checklist
- Candidate-screening spreadsheet
- Sample interview schedule
- Sample interview questions
- Sample interview summary
- Reference checklist
- Hiring checklist

Effectively Manage Developers So You Can Deliver Better Software



ORDER & SAVE

Save 35% When You Order

from informit.com/infoq/managing and enter the code **INFOQ** during checkout

FREE US SHIPPING on print books

Major eBook Formats
Only InformIT offers PDF, EPUB, & MOBI together for one price

OTHER AVAILABILITY

Through O'Reilly Online Learning (Safari) subscription service

Booksellers and online retailers including Amazon/Kindle store and Barnes & Noble | bn.com

"Lichty and Mantle have assembled a guide that will help you hire, motivate, and mentor a software development team that functions at the highest level. Their rules of thumb and coaching advice form a great blueprint for new and experienced software engineering managers alike."
—Tom Conrad, CTO, Pandora

"Reading this book's nuggets felt like the sort of guidance that I would get from a trusted mentor. A mentor who I not only trusted, but one who trusted me to take the wisdom, understand its limits, and apply it correctly."
—Mike Fauzy, CTO, FauzyLogic

Today, many software projects continue to run catastrophically over schedule and budget, and still don't deliver what customers want. Some organizations conclude that software development can't be managed well. But it can—and it starts with people. Mickey W. Mantle and Ron Lichty show how to hire and develop programmers, onboard new hires quickly and successfully, and build and nurture highly effective and productive teams.

Drawing on over 80 years of combined industry experience, the authors share Rules of Thumb, Nuggets of Wisdom, checklists, and other Tools for successfully leading programmers and teams, whether they're co-located or dispersed worldwide.

- Find, recruit, and hire the right programmers, when you need them
- Manage programmers as the individuals they are
- Motivate software people and teams to accomplish truly great feats
- Create a successful development subculture that can thrive even in a toxic company culture
- Master the arts of managing down and managing up
- Embrace your role as a manager who empowers self-directed agile teams to thrive and succeed

Pearson
Addison-Wesley

**Discount code INFOQ confers a 35% discount off the list price on informit.com only. Discount may not be combined with any other offer, including the book + eBook "Best Value" bundle, and is subject to change.*

informIT®
the trusted technology learning source