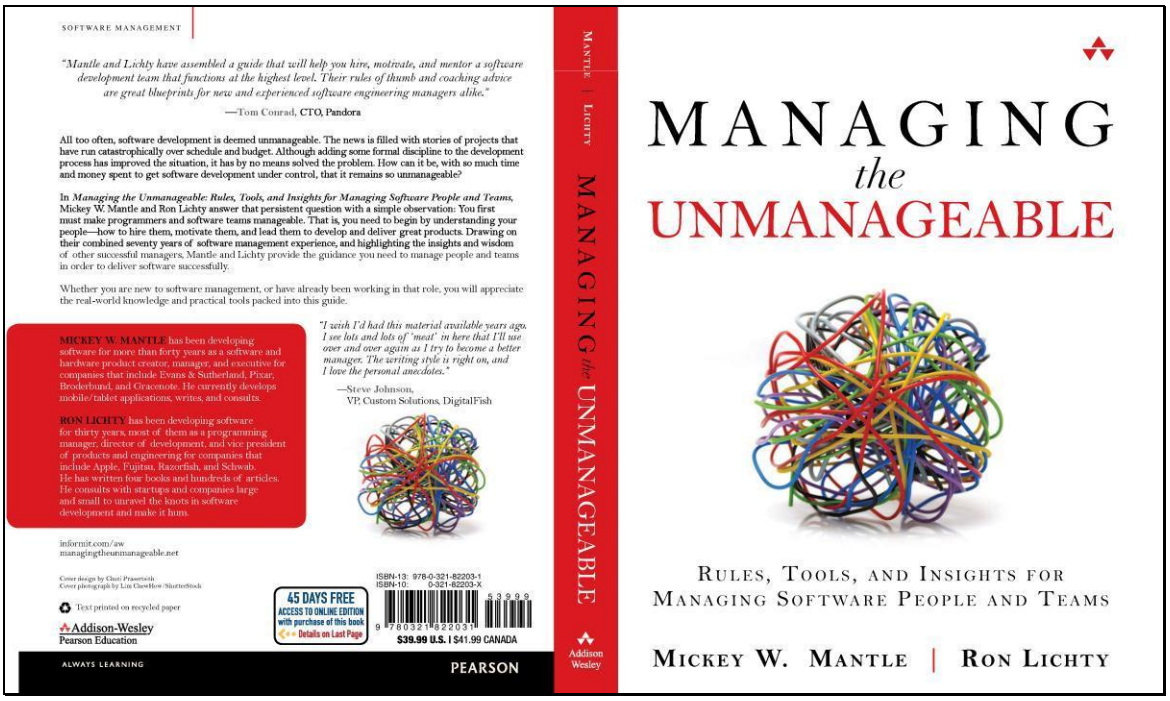
Managing the Unmanageable

Rules, Tools, and Insights for Managing Software People and Teams



Paperback: 422 pages

Publisher: Addison-Wesley Professional; 1st edition (September 15, 2012)

Language: English

ISBN-10: 032182203X

ISBN-13: 978-0321822031

managingtheunmanageable.net

Two software industry veterans with over 70 years of combined experience have written a book that will help any software manager be more successful. Having spent their careers developing software, leading software development projects, and managing programmers and teams, they have now distilled their experience into a book that every beginning programming manager should read and have on their bookshelves for reference.

It's a book that will also help executives who struggle sponsoring projects dependent upon software to succeed – CEOs, COOs, CTOs, and others – understand the craft of software development and the intricacies of how to manage software people and teams to deliver software projects successfully.

This book will help any software manager get a handle on their own unmanageable challenges.

True to its subtitle, the book is comprised of Rules, Tools and Insights for managing software people and teams:

Rules

Nearly 300 rules of thumb and nuggets of wisdom from software luminaries, managers, and developers – succinct insights into managing programmers and development teams – are collected into three sections:

- Managing
- Managing People
- Managing Teams to Successfully Deliver

There are two different forms of these seeds of advice:

Rules of Thumb – Proven, pithy wisdom for making things work
Nuggets of Wisdom – Concise insights to guide your actions and thinking

An excerpt of the Rules is included with the Chapter excerpts at the end of this document.

Tools

The book delivers dozens of tools the authors have collected or created over the years to help manage programmers: they share their sample job descriptions, interview summaries, interview questions, goal setting templates, developer skills inventories, code review guides, sample project workbooks, and many other tools that will save managers hours and days of time and effort.

Samples of the numerous Microsoft Word documents and templates and Excel spreadsheets available to the reader follow after the Insights section below.

Insights

The authors share, in nine chapters, their hard-won experience gained from programming, managing and delivering software spanning two managerial lifetimes of companies and situations. The chapters are sprinkled with anecdotes from their experience as well as rules of thumb and nuggets of wisdom they gleaned from their gurus. They are written in a straightforward first-person manner by the authors, as if they were sitting with you relating their experiences as personal mentors. Excerpts from Chapters 3, 5, and 8 are at the end of this document.

Chapter 1, *Why Programmers Seem Unmanageable*, reviews why programmers are special when it comes to managing them as individuals and leading them as teams.

Chapter 2, *Understanding Programmers*, provides a number of lenses through which to view programmers that will help reveal the uniqueness each of them brings – and inform the task of managing each of them uniquely.

Chapter 3, *Finding and Hiring Great Programmers*, is a step-by-step guide to finding, recruiting and hiring great programmers.

Chapter 4, *Getting New Programmers Started Off Right*, 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 established processes and practices.

Chapter 5, *Becoming an Effective Programming Manager: Managing Down*, walks through the core of management: the mechanics and how-to of the day-to-day with the team – the tasks and interactions to successfully manage programmers.

Chapter 6, *Becoming an Effective Programming Manager: Managing Up, Out, and Yourself*, addresses the fact that success as a programming manager also demands becoming skillful at managing up – managing the manager you report to; managing out – managing your many relationships with peers, leveraging others in your organization and marshaling external resources and relationships; and finally managing yourself – your priorities, your style, your time, your growth, your life.

Chapter 7, *Motivating Programmers*, focuses on the critical task of motivating programmers to accomplish great feats and deliver difficult projects. The framework provides different thinking about ways to motivate – and how to recognize and avoid the potholes that demotivate – the team you are entrusted to manage.

Chapter 8, *Establishing a Successful Programming Culture*, provides context to think about corporate culture and about how to create the development subculture needed for success within even the most toxic of corporate cultures.

Chapter 9, *Managing Successful Software Delivery*, deals with the true test of a successful manager, delivering software successfully. This chapter is not about project management but about the role seldom addressed: the team manager's essential role in delivery.

Tool Samples

Among the tools available are sample Job Descriptions for five levels of Client Programmers, System Programmers, and Database Programmers. An example of the entry-level job description for an Application Programmer is shown below.

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

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

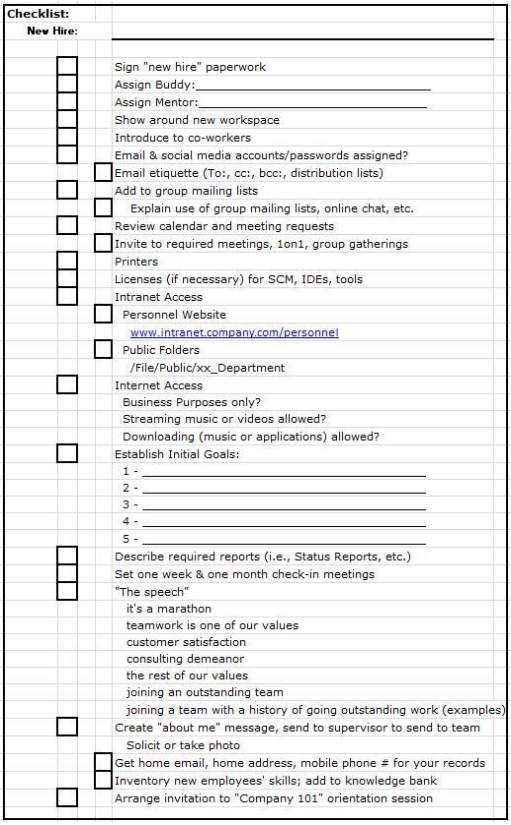
JOB REQUIREMENTS:

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

A structured job career track for Application Programmers, System Programmers, and Database Programmers is included to help address the often asked question, when will I be promoted. The Client Programmer guidelines are shown below.

Programming Levels	Client Programmers
Entry Level	Programmer 3
1-5 Years Experience	Programmer 2
Experienced (5-10 years)	Programmer 1
Experienced (10-20 years)	Senior Programmer 2
Very Experienced (12+ years)	Senior Programmer 1/Architect

A checklist to help make sure a new hire is started successfully is shown below.

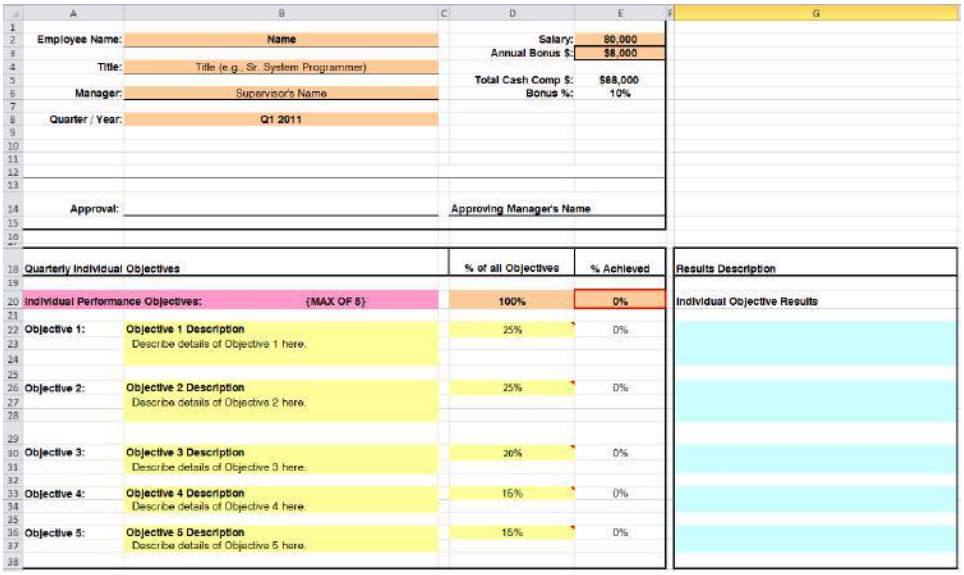


Checklist:

New Hire: _____

- ☐ Sign "new hire" paperwork
- ☐ Assign Buddy:_____
- ☐ Assign Mentor:_____
- ☐ Show around new workspace
- ☐ Introduce to co-workers
- ☐ Email & social media accounts/passwords assigned?
  - ☐ Email etiquette (To:, cc:, bcc:, distribution lists)
- ☐ Add to group mailing lists
  - ☐ Explain use of group mailing lists, online chat, etc.
- ☐ Review calendar and meeting requests
- ☐ Invite to required meetings, 1on1, group gatherings
- ☐ Printers
- ☐ Licenses (if necessary) for SCM, IDEs, tools
- ☐ Intranet Access
  - ☐ Personnel Website
    - www.intranet.company.com/personnel
  - ☐ Public Folders
    - /File/Public/xx_Department
- ☐ Internet Access
  - Business Purposes only?
  - Streaming music or videos allowed?
  - Downloading (music or applications) allowed?
- ☐ Establish Initial Goals:
  - 1 - _____
  - 2 - _____
  - 3 - _____
  - 4 - _____
  - 5 - _____
- ☐ Describe required reports (i.e., Status Reports, etc.)
- ☐ Set one week & one month check-in meetings
- ☐ "The speech"
  - it's a marathon
  - teamwork is one of our values
  - customer satisfaction
  - consulting demeanor
  - the rest of our values
  - joining an outstanding team
  - joining a team with a history of going outstanding work (examples)
- ☐ Create "about me" message, send to supervisor to send to team
  - Solicit or take photo
- ☐ Get home email, home address, mobile phone # for your records
- ☐ Inventory new employees' skills; add to knowledge bank
- ☐ Arrange invitation to "Company 101" orientation session

A spreadsheet to help manage quarterly goals for a programmer is included, as shown below. An advanced version of the spreadsheet that allows goals to be tied to quarterly bonus payouts is also among the tools included.



Employee Name:	Name		Salary:	80,000	
			Annual Bonus $:	$8,000	
Title:	Title (e.g., Sr. System Programmer)		Total Cash Comp $:	$88,000	
Manager:	Supervisor's Name		Bonus %:	10%	
Quarter / Year:	Q1 2011				
Approval:			Approving Manager's Name		
Quarterly Individual Objectives			**% of all Objectives**	**% Achieved**	**Results Description**
Individual Performance Objectives:		**{MAX OF 5}**	**100%**	**0%**	**Individual Objective Results**
Objective 1:	**Objective 1 Description**	Describe details of Objective 1 here.	25%	0%	
Objective 2:	**Objective 2 Description**	Describe details of Objective 2 here.	25%	0%	
Objective 3:	**Objective 3 Description**	Describe details of Objective 3 here.	20%	0%	
Objective 4:	**Objective 4 Description**	Describe details of Objective 4 here.	15%	0%	
Objective 5:	**Objective 5 Description**	Describe details of Objective 5 here.	15%	0%	

These are only a few samples of the many Tools available from the Author's website to the readers of this book.

About the Authors

Mickey W. Mantle

Mickey has been developing software for over 40 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 had his first programming job in 1971 developing the real-time control software for a U.S. Navy aircraft rework facility at Kenway Engineering (later Eaton-Kenway). He thereafter joined 3-D computer graphics pioneer Evans & Sutherland (E&S).

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 worked at Formative Technologies, Pixar (after it was bought by Steve Jobs and spun out of Lucasfilm Ltd. in 1986), Brøderbund Software where he was Vice President of Engineering/CTO, and International Microcomputer Software, Inc. as Vice President of R&D/CTO.

In 1999 Mickey joined Gracenote where he was Senior Vice President of Development. At Gracenote he managed all development, operations, and professional services associated with the pioneering Web-based CDDB music information service. Gracenote's products utilize technology ranging from Web services and relational databases to embedded systems and mobile applications, giving him a unique perspective on the wide-ranging needs of the various types of software developed today.

He retired from Gracenote in early 2011 to develop mobile/tablet applications, and consult with a variety of companies and organizations regarding the management of software people and teams. His 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, Europe, and Japan, he brings insight into the challenges of managing software development using diverse staff and teams that are hours and oceans apart.

Ron Lichty

Ron has been developing software for 30 years, over 20 of them as a Development Manager, Director of Engineering, and Vice President of Engineering. 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, 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.

Recruited to Apple in 1988, Ron product-managed Apple's development tools, then led Finder development, managing delivery 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. He was promoted to Vice President while leading his CIO's three-year initiative to migrate software development across Schwab to a single, cost-effective platform company-wide.

Since then, he has been a Vice President of Engineering and Vice President of Products both as an employee and as a consultant, and has continued to focus on making software development "hum." He headed technology for the California offices of Avenue A | Razorfish, products and development for Forensic Logic, programming for Socialtext, engineering of the consumer ZoneAlarm line for Check Point, and publisher services for Stanford's HighWire enterprise.

In consulting engagements in America and Europe, he has helped development groups overcome roadblocks, untangle organizational knots, and become more productive. Ron's developer conference and professional group talks and webinars include implementing Agile and Scrum and transforming software development from chaos to clarity. He has been an adviser to a half-dozen start-ups.

Managing the Unmanageable

Rules, Tools, and Insights for Managing Software People and Teams

Insights

Chapter 1 - Why Programmers Seem Unmanageable

What do Programmers Do? • Why is Becoming a Successful Programming Manager Hard?

Chapter 2 - Understanding Programmers

How Programmers Differ by Discipline, Type, Domain • Job Requirements and Abilities • Effect of Proximity • Employees vs Contractors • Differences by Generation and by Personality

Chapter 3 - Finding and Hiring Great Programmers

What Kind to Hire • Writing the Req • Selling the Hire Internally • Recruiting Employees • Recruiting Contractors • Reviewing Resumes • Narrowing the Field • Preparing to Interview • Interviewing • Making the Hire Decision • Making the Right Offer • Follow-up

Chapter 4 - Getting New Programmers Started Off Right

Get Them On Board Early • Prepare For Their Arrival • First Day Musts • Introductions • Ensuring Success • Initial Expectations

Chapter 5 - Becoming Effective: Managing Down

Earning Technical Respect • Hire Greatness • Turbocharge the Team You Have • Manage Every Type of Programmer • Facilitation • Protection • Judging and Improving Performance: Objectives, Reviews • Firing • Creating, Organizing and Reorganizing Teams • Troubleshooting a Dysfunctional Organization • Deliver Results • Celebrate Success

Interlude: Rules of Thumb and Nuggets of Wisdom

The Challenges of Managing
Managing People
Managing Teams to Successfully Deliver

Chapter 6 – Becoming Effective: Managing Up, Out, and Yourself

Managing Up • Managing Out • Managing Outside the Company • Managing Yourself

Chapter 7 - Motivating Programmers

Motivational Theories: Maslow, McGregor, Herzberg • Putting Motivational Theory into Practice • Dissatisfiers • Motivators • Personal Commitment

Chapter 8 - Establishing a Successful Programming Culture

Programming Cultures • Company Cultures • What Makes Programming Cultures Successful

Chapter 9 - Managing Successful Software Delivery

Defining the Project • Planning the Work • Kicking Off the Plan • Executing the Work • Running the End Game • Delivering Results

Appendix: Tools and Techniques

3
Finding and Hiring Great Programmers

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

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

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

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

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

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

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

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

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

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

4. Barry Boehm, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, October 1988.

5. Barry Boehm et al., *Software Cost Estimation with Cocomo II* (Addison-Wesley, 2000).

6. Barry Boehm et al., *Software Cost Estimation with Cocomo II* (Addison-Wesley, 2000).

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

> *A's hire A's. B's hire C's.*
>
> —*STEVE JOBS*

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

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

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

Determining What Kind of Programmer to Hire

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

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

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

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

5

Becoming an Effective Programming Manager: Managing Down

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

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

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

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

Earning Technical Respect

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8

Establishing a Successful Programming Culture

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

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

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

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

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

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

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

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

> *Under the right conditions, the problems of commitment, alignment, motivation, and change largely melt away.*
>
> —*JIM COLLINS*[1]

Defining "Successful"

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

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

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

The Programming Culture

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

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

Creating a powerful programming culture requires establishing

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

The challenge is: How do you do that?

Company Culture

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

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

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

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

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

directive CEO Jeffrey Skilling is said to have given his management team to "cut jobs ruthlessly by 50 percent."

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

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

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

Leveraging the Complexity of Your Company's Culture

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

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

Rules of Thumb and Nuggets of Wisdom

We all manage by using rules of thumb. Sometimes we create them ourselves—we have some hard experience that teaches a big lesson, from our having squeezed through it, or perhaps because we failed to. Sometimes the lesson comes to us already fully formed, and so succinct we can communicate it in a phrase or a sentence. Other times we struggle to boil complexity down to its pith.

Benjamin Franklin collected and published one of the first well-known sets of maxims, proverbs, and rules of thumb in *Poor Richard's Almanack*.[1] Around the same time, in 1732, Thomas Fuller published his own book of proverbs that included the ubiquitous "A stitch in time saves nine," a rule of thumb for much of life.

Rules of thumb we learned growing up and others are often as applicable to managing software development as to the rest of life. More than one software development manager has used "Measure twice, cut once," even though code is probably the most resilient of materials. Here are a few that predate our profession, many by hundreds of years, yet apply to us:

Measure twice, cut once.

—TRADITIONAL

This is perhaps the simplest and clearest example of a rule of thumb. Nearly everyone has failed to follow this rule, and most of us have regretted doing so at least once, perhaps none so much as the space scientists whose metric-system-based Mars Climate Orbiter spacecraft disintegrated when the ground-based system sent its Mars orbit trajectory in English units.

Life is simpler when you plow around the stump.

—TRADITIONAL

Sometimes bad practices are so deeply rooted that you need to route learning and the introduction of better practices around them.

1. http://en.wikipedia.org/wiki/Poor_Richards_Almanack.

Do not corner something that you know is meaner than you.

—TRADITIONAL

The corollary would have to be "Do not corner anything if you're not willing to find out just how mean it can be." As applied to corporate politics, the rule is simple: "Don't corner your people, don't corner your peers, and whatever you do, don't corner someone who outranks you!"

Don't interfere with somethin' that ain't botherin' you none.

—TRADITIONAL

There's way too much to fix to fool with the stuff that works.

When you wallow with pigs, expect to get dirty.

—TRADITIONAL

A corollary to "Do not corner anything."

Save early, save often.

—LEARNED COMMON WISDOM

Recommended always, and especially when running beta software, or version 1 of anything.

While we learned some rules of thumb growing up, we learn others from the counsel of managers and peers, whether to give us solace from having had similar experiences, to counsel us in how to identify and avoid future problems, or to mentor us with their hard-won wisdom.

We characterize two different forms of these seeds of advice:

- Rules of thumb—proven, pithy wisdom for making things work
- Nuggets of wisdom—concise insights to guide your actions and thinking

Separately and then together, we have long collected and used rules and nuggets to help us understand and manage programmers. Some come from our reading, some from our mentors and colleagues, some from the famous, some from our own experience. Many are well known and widely circulated. More than one software development manager has used "Underpromise, overdeliver," even though when first said it was probably not about software. Others are less well known. We find all of them relevant.

We hope you too will find them useful for understanding and managing programmers and teams.

The Challenges of Managing

Managers must manage.

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

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

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

—PETER DRUCKER[2]

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

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

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

—DAVE SMITH, Agile Software Development Coach[3]

People hate change but love progress.

—TERRY PEARCE, Author, Leading Out Loud

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

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

You miss 100 percent of the shots you never take.

—WAYNE GRETZKY, Hockey Phenom

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

—MICHAEL JORDAN, Basketball Phenom[5]

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

—TIM SWIHART, Engineering Director, Apple Computer

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

3. http://c2.com/cgi/wiki?WorstThingsFirst.

4. Spoken at BayALN, the Bay Area chapter, January 2007.

5. Quoted in *Guideposts*, August 2002.

it went wrong, and what's being done to correct the situation. Too often in business, witch hunts start because those in charge can't get straight answers when things aren't going well. Providing timely insight into what went wrong and what's being changed to minimize the chance of it happening again provides them with the knowledge needed to accurately assess the impact of that problem.

Don't let the day-to-day eat you up.

—DAVID DIBBLE, Schwab and Yahoo EVP and First Data CIO

David made this statement to make the point to his management team that managers have "real" work to do; that the seemingly urgent—e-mail, meetings, the routine—could easily fill a day. Only by being intentional about how we use our days can managers overcome letting that happen.

Every hour of planning saves about a day of wasted time and effort.

—STEVE MCCONNELL, CEO and Chief Software Engineer, Construx Software, and Author, Code Complete *and* Rapid Development [6]

Prioritize. Sometimes, it is urgent to wait.

—PHAC LE TUAN, VP of Engineering and CEO, San Francisco Bay Area

Chairman and CEO, Kinemo International, Phac explains: "When an unexpected issue comes up, engineers (mostly) tend to want to fix it right away to show their mettle, forgetting their actual priorities. It is actually more difficult to sit back and wait first to understand the actual priority of the new issue."

6. Steve McConnell, *Software Project Survival Guide* (Microsoft Press, 1998), p. 36.