Epilogue

A Word on Leading

Occasionally I'll come across the idea that as the lead for a project, you cannot and never will be a part of the team, that you will always be a step removed, and that there is nothing you can do about it. In my experience, that isn't true. I've been a part of dozens of teams—as both lead and programmer—and without exception the teams that jelled were those in which the lead was just another person on the team, one who happened to have some nonprogramming responsibilities. There was never the feeling that the lead was superior.

To someone who didn't know much about American football, the quarterback might seem to be in a superior position with respect to the other players. After all, the quarterback calls each play, the quarterback is the focal team member who has control of the ball, and after a victory it's the quarterback who usually gets carried off the field by the other team members.

The quarterback might appear to be superior in rank to the other players, but we know better. The quarterback is just another team member who happens to have unique responsibilities. An effective project lead is no different. He or she understands that a focal team member is not superior to other team members:

> *The lead is just another team member, who, like every other team member, has his or her own set of unique responsibilities.*

Effective leads understand that team members play different roles on the team. Some team members are responsible for the data entry part of the project, others for the print engine, still others for foreign file converters and the user interface design. Leads may implement features along with everybody else, but in addition to that work, they have the responsibility for setting project goals and priorities, keeping dependent groups such as Testing and Marketing informed of progress, creating an environment in which the team members can work effectively, and ensuring that team members are learning new skills as a way of adding value to the company. A lead can do all those tasks without adopting the attitude that he or she is superior.

If a lead has the attitude that he or she is superior, a whole array of harmful behaviors follows. Here's what happens in extreme cases:

- The lead blames the team for failures but gladly takes the credit for successes.
- The lead doesn't care about the people on the team. They're just workers. Who cares if they work 80-hour weeks? The lead is concerned only that the team might make him look bad by missing a scheduled date.
- The lead expects team members to jump at every command and never question her authority. "I said 'do it,' so *do it*" is the motto.
- Anxious not to appear inferior in any way, the lead attacks any team member who threatens his authority or who appears to be more skilled or knowledgeable than the lead in any area.
- Because she must always be right, the lead never admits it when she is wrong.

- The lead shuts down anybody who suggests improvements to the development process or otherwise rocks the boat.
- The lead acts as if he is indispensable.

Granted, not all leads who think of themselves as superior behave so tyrannically, but even in mild cases the air of superiority still comes through. Do team members work *for* the lead or *with* the lead? The very language the lead uses reveals the underlying attitude.

A lead who views herself as a team member works better because she spends little or no time fighting to keep the other team members in their place—why should she? By choosing to adopt the attitude that she's not superior, she relieves herself of having to attack perceived threats to her authority. When such a lead discovers a superstar on the team she's just inherited, she doesn't raise her guard and start the territorial one-upmanship battle so common in people who must feel superior. Such a lead is more likely to be thankful and to work together with the superstar for the benefit of the project.

Your own attitude as a lead can influence everything you do. If you and a team member disagree over a performance review, how do you react? Do you stand firm because you feel you need to be "right," or do you discuss the problem to see if there's another valid interpretation of events? If you and the team member still disagreed, would you amend the review to describe both positions so that others who read the review later could make their own evaluations?

Look again at the bulleted list that characterizes the behaviors of the leads who insist on regarding themselves as superior. Would a lead who viewed herself as just another team member exhibit those kinds of behavior? Which type of lead would you be more willing to work with, one who behaves in a superior way or one who treats you with more respect? Be the kind of lead *you* would want to work with.

Leads should see themselves as members of their teams, not as superior to them.

REFERENCES

These books are explicitly referenced in the text.

Bentley, Jon. *Writing Efficient Programs*. Englewood Cliffs, N. J.: Prentice Hall, 1982.

DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*. New York: Dorset House, 1987.

Gerber, Michael E. *The E-Myth: Why Most Small Businesses Don't Work and What To Do About It*. New York: Harper Business, 1986.

Kernighan, Brian W., and P. J. Plauger. *The Elements of Programming Style*. 2d ed. New York: McGraw-Hill, 1978.

Koenig, Andrew. *C Traps and Pitfalls*. Reading, Mass.: Addison-Wesley, 1989.

Maguire, Steve. *Writing Solid Code*. Redmond, Wash.: Microsoft Press, 1993.

McConnell, Steve. *Code Complete*. Redmond, Wash.: Microsoft Press, 1993.

McCormack, Mark H. *What They Don't Teach You at Harvard Business School*. New York: Bantam Books, 1984.

Weinberg, Gerald M. *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold, 1971.

These educators are mentioned in the preface:

Anthony Robbins
Robbins Research International, Inc.
9191 Towne Centre Drive, Suite 600
San Diego, CA 92122
Phone: (800) 445-8183
FAX: (619) 535-0861

Michael E. Gerber
Gerber Business Development Corporation
1135 N. McDowell Blvd.
Petaluma, CA 94954
Phone: (707) 778-2900

INDEX

A

annual reviews, 120–22
applications. *See also* Microsoft Excel
  priorities for, 18–19
  and shared code, 141–43
arbitrary deadlines, 99
assignment bugs, 126–27
attack plans
  including in postmortem reports, 78–81
  need for detail, 79–80
  questions that elicit, 33
attitudes
  about bugs, 125–29
  about lead, 171–73
  leveraging, 144–49
  negative, changing, 131–35
  resistant, 129–31
  toward users, 136–40
  toward working long hours, 155–61, 168–70

B

backward compatibility, 14
bad coffee (example), 24–26
Bentley, Jon, 117, 119
bonuses, basis for, 161
books, recommended, 117
bugs
  attitudes toward, 125–29
bugs, *continued*
  fixing early, 128
  goal of bug-free code, 125–29
  as negative feedback loop, 27, 39
  and quality definition, 28
  questions to ask, 31–32
  researching problems, 50
  when to fix, 26–29

C

C code, rewriting in C++, 68–69
C compiler, 59–62, 102–4
"can'titude," 131–35
clipboard, Microsoft Excel, 67–68
code
  bug-free, 125–29
  goto statements in, 35–37
  line counts, 40–41
  master, 127–28
  multi-platform, 133–35
  portability, 17, 18, 19
  priorities for, 17–19
  reformatting source files, 68–69
  reusable, 141–43
  shared, 141–43
  variations among programmers, 108
Code Complete, 36, 117
coding wars, 108
coffee quality (example), 24–26
compatibility, backward, 14

compilers
  cross development project, 59–62, 102–4
  and linker quality, 140–41
  turning on warnings, 126, 127
"cool" features, 65–67
cross development system
  becomes product, 145–47
  and FORTRAN compiler, 59–62
  subprojects in development, 102–4
cross-pollination theory, 115
crutches, systems as, 30
C Traps and Pitfalls, 50, 117

D

deadlines. *See also* schedules; ship dates
  arbitrary, 99
  near-term, 98–101
  and subprojects, 98–104, 105
debug code, adding, 129, 130–31
debugging
  attitudes toward, 125–29
  questions to ask, 31–32
  research during, 50
  when to do, 26–29
decision making
  and meetings, 85, 86
  and priorities, 20, 130, 131
  and snap decisions, 20
delegation, 4–5
DeMarco, Tom, 108
dependency issues
  controlling, 15
  and saying No, 54–55
  and status meetings, 8
design meetings, 83–84
desk accessories, adding, 65–67
development process at Microsoft, xvii–xx
development teams. *See* programmers; project leads
dialog manager project, 48–51, 114, 153
Dijkstra, Edsger, 36

E

editing vs. writing, 23–24
The Elements of Programming Style, 117
e-mail
  answering, 5
  at Microsoft, xx
  as problem, 2, 3, 30, 163, 164
  for status reporting, 10
  when to read, 30, 165, 166
The E-Myth, 117
end-cut pot roast rule, 75
end users
  attitude toward, 136–40
  considering, 139–40
Excel. *See* Microsoft Excel

F

features. *See* products
feature teams, 11
feedback loops, 37–41
figure skating, 107–8
fixing bugs
  attitudes toward, 125–29
  questions to ask, 31–32
  research while, 50
  when to do, 26–29
flextime, 163
focus
  importance of, 2–4

focus, *continued*
and need for status reports, 7–10
removing obstacles to, 4–6
follow-up work, 3, 87–88
FORTRAN compiler, 59–62
"free" features and products, 61–62
function headers, adding, 68

G

Gates, Bill, 134
Gerber, Michael, 117
Gimpel Software, 50
goal setting
and bug-fixing, 26–29
and coding priorities, 17–19
and deadlines, 99
and debug code, 130–31
and decision making, 20, 130, 131
importance of, 16
in the moment, 119–20
and need to say No, 57
personal, 116–20
specificity of, 12–15
and subprojects, 98–104
goto statements, 35–37
guidelines vs. rules, 35–37, 75

H

headers, adding, 68
housekeeping. *See* process work
house moving (example), 5, 46

I

improvement goals, 116–20
inline directive, 19

K

Kernighan, Brian, 117
keyboard-driven menus
and end users, 139–40
and shared code, 141–42
Knuth, Donald, 36
Koenig, Andrew, 117

L

LAYOFF macro, 63
leads, types of, xvii–xviii. *See also* project leads, program managers
leverage
creating, 144–45
use of, 145–46
libraries. *See* user interface library project
linker, need for improvement, 141
Lister, Timothy, 108
little systems, 24, 25, 28, 29, 30
long hours
attitudes toward, 155–61
and personal life, 168–170
and time management, 162–67

M

Macintosh projects. *See* Microsoft projects
macros, 19, 63, 64
maintainability, 68
marketing teams, requests from, 58, 63–65
master source code, 127–28
master task lists. *See* task lists
mastery, 1–2

McConnell, Steve, 36, 117
McCormack, Mark, 117
meetings
  and action items, 87–88
  benefits vs. drawbacks, 84–85
  and decision making, 85, 86
  design, 83–84
  and follow-up tasks, 87–88
  good times for, 83
  and negative feedback loops, 88
  project review, 4–5, 86
  questions to ask before calling, 82, 84
  recurrent, 81–84
  status, 81
  worthwhile, 81–82
Microsoft Excel
  clipboard paradigm, 67–68
  and *LAYOFF* macro, 63, 64
  multi-platform version, 132–35
  schedule for, 91–95, 153
  Windows vs. Macintosh versions, 132–35, 139–40
Microsoft projects. *See also names of products*
  and Applications division, 141, 145
  compiler cross development, 59–62, 102–4, 146–47
  dialog manager, 48–51, 114, 153
  Excel for the Macintosh, 92–95, 132–35, 139–40, 142–43
  and Languages division, 140–41, 145
  Macintosh keyboard-driven menus, 139–40, 141–42
  Macintosh print preview feature, 142–43
  multi-platform, 133–35
  and shared code, 141–43
Microsoft projects, *continued*
  user interface library, 12–15, 51–53, 56, 65–67, 152–53
  Windows vs. Macintosh, 132, 133–34, 139–40
  Word for MS-DOS, 56
  Word for Windows, 48–51
Microsoft Windows vs. Macintosh, 132, 133–34, 139–40
milestones
  and personal growth goals, 116–18
  scheduling by, 98–104
multi-platform code, 133–35

N–O

naming conventions, 68
near-term deadlines, 98–101
negative feedback loops
  and bug-fixing, 27, 39
  defined, 38
  destructive, 39
  and follow-up work, 88
  vs. negative reinforcement, 40
No, saying, 54–56
object-oriented methodologies, 68–69
operating systems, priorities for, 18
optional compiler warnings, 126, 127
oral reports, 76

P

Pascal compiler, 60, 61, 62, 103
pay raises, basis for, 161
PC-Lint, 50
Peopleware, 108

personal growth goals
  aligning with project milestones, 116–18
  documenting in annual reviews, 120–22
  setting in the moment, 119–20
personal life, 153, 168–69, 170
personal schedules, 162–67
planning, 12–15. *See also* attack plans
Plauger, P. J., 117
portability, as coding priority, 17, 18, 19
positive feedback loops, 38, 40–41
postmortem reports
  acting on, 80–81
  attack plans in, 78–81
  importance of, 78
  when to write, 80
pot roast rule, 75
print preview feature, 142–43
priorities
  for coding, 17–19
  and decision making, 20, 130, 131
  and subprojects, 100–101
proactivity, 46–47
problems. *See also* questions
  anticipating, 46–48
  bringing up, 135
  defining correctly, 48–51
  e-mail as, 2, 3, 30, 163, 164
  and use of time, 162–64
process work, 3–4, 7–10, 88–89
products. *See also* Microsoft projects
  focus on improving, 2–4
  "free," 61–62
  inclusive definition, 141
  requests to add features, 63–65
  substandard features, 138
program managers, xviii, xix
programmers
  attitudes toward bugs, 125–29
  "average" skill level, 108–9, 112
  and bug-fixing, 27, 28–29, 31–32
  effectiveness of, 1–2
  vs. end users, 136–38
  on feature teams, 11
  as long-term specialists, 109
  need for focus, 2–4
  personal schedules, 162–67
  protecting, 4–6
  questions to ask, 32
  reassigning, 113–15
  and skill-building, 108–13
  and task decisions, 130, 131
  training, for promotion, 116–18
  from upstart companies, 123
  use of time, 162–67
  working long hours, 151–70
project goals
  and bug-fixing, 26–29
  and coding priorities, 17–19
  and debug code, 130–31
  and decision making, 20, 130, 131
  and need to say No, 57
  setting, 12–15
  specificity of, 12–15
  and subprojects, 98–104
project leads
  anticipating problems, 46–48
  asking questions, 32–35
  and delegation, 4–5
  effectiveness of, 1–2
  vs. leaders, xv–xvi
  need for focus, 3–4
  of other leads, 6
  proactivity of, 46–48
  as protectors, 4–6

project leads, *continued*
status meetings for, 8
as team members, 171–73
training for, 116–18
project review meetings, 4–5, 86
projects. *See* Microsoft projects; project goals
project task list. *See* task lists
The Psychology of Computer Programming, 117

Q

quality bars, 18, 19, 28, 49, 138
questions. *See also* problems; requests
defining context, 53
level of precision, 32–35
wrong vs. right, 51–53

R

raises, basis for, 161
recurrent meetings, 81–84
reports
follow-up, 3
oral, 76
postmortem, 78–81
problems with, 77
status, 3
trip, 74–76
requests. *See also* questions
for added product features, 63–65
defining context, 53
from superiors, 58–60
when to say No, 54
research, as problem-solving strategy, 50, 51
reusable code, 141–43
robustness, as coding priority, 17, 18
rules vs. guidelines, 35–37, 75

S

safety, as coding priority, 17, 18, 19
saying No, 54–56
schedules
aggressive vs. unattainable, 95–97
and arbitrary deadlines, 99
and bug-fixing, 27, 28, 29
and goal setting, 99
and long working hours, 151–70
and Microsoft Excel project, 91–95, 153
and milestones, 98–104
personal, 162–67
questions to ask, 33–34
and sense of urgency, 95–97
and status reports, 7–10
and subprojects, 98–104
undue focus on, 93–95
unrealistic, 94, 95, 97
scheduling meetings, 82
sense of urgency, 95–97
shared library, as goal, 13, 57
sharing code, 141–43
ship dates. *See also* deadlines
best case, 104, 105
questions to ask, 33–34
680x0 cross development system
becomes product, 145–47
and FORTRAN compiler, 59–62
subprojects in development, 102–4
size, as coding priority, 17, 18
skill-building, 1–2, 31, 108–13
by asking questions, 32–35
leveraging, 144–45

skill-building, *continued*
for promotion, 116–18
and versatility, 111
snap decisions, 20
solutions, 135
speed, as coding priority, 17, 18
speed bumps, 88–89
status meetings, 3, 7, 8
status reports
benefits vs. drawbacks, 8–10
as necessary evil, 7–10
need for, 3
negativity of, 8–9
positive, 9–10
strategies. *See* goal setting; systems, work
subprojects, 98–104
substandard features, 138
superiors, as team members, 171–73
systems, work, 24, 25, 28, 29, 30

T

task lists
breaking up, 98–104
for Microsoft Excel project, 93–95
and subprojects, 98–104
team leads. *See* project leads
team spirit, 82
technical leads, xvii
third party vendors, 65–67
time
efficient use of, 162–67
and scheduling meetings, 83
and sense of urgency, 95–97
training. *See* skill-building
trial and error, 1–2
trip reports, 74–76
trivial processes, 24, 25, 28, 29, 30

U

urgency, sense of, 95–97
usability studies, 137
user interface library project
responding to requests, 51–53, 56, 65–67
schedule problems, 152–53
setting goals for, 12–15
users
attitude toward, 136–40
consideration of, 139–40

V

Visual C++, 141
visual freeze point, xix

W

weekends, working, 159–60
Weinberg, Gerald, 117
What They Don't Teach You at Harvard Business School, 117
Windows Everywhere, 146
Windows vs. Macintosh, 132, 133–34, 139–40
Winter Olympics, 107
Word for MS-DOS, 56
Word for Windows, 48–51
working hours, 151–70
work systems, 24, 25, 28, 29, 30
Wow! factor, 101–4
Writing Efficient Programs, 117, 119
Writing Solid Code, xii, xvi–xvii, 27–29, 117
writing vs. editing, 23–24

ABOUT THE AUTHOR

Steve Maguire graduated from the University of Arizona with a degree in electrical and computer engineering, but he has always gravitated toward work in computer software. Steve has programmed professionally for the past 19 years in both Japan and the United States. In the late 1970s Steve regularly contributed developer tools, applications utilities, and the occasional video game to the Processor Technology and NorthStar users' groups. Steve has been responsible for numerous projects since then, including *valFORTH* in 1982, an award-winning FORTH development system that enabled Atari programmers to write high-quality graphics applications and video games.

In 1986 Steve joined Microsoft Corporation for the opportunity to work on high-end Macintosh applications. Steve worked on Microsoft Excel and led the development of Microsoft's Intel-hosted MC680x0 Macintosh cross development system. He was the driving force behind Microsoft's switch to a cross-platform shared code strategy in its applications development and is perhaps best known in the company for his efforts to increase the utility and quality of shared code libraries. As a veteran software design engineer and project lead, Steve spent several of his years at Microsoft working with troubled projects—enabling teams to work effectively and, not incidentally, to enjoy their work.

Debugging the Development Process is the second of several books Steve is writing to give programmers practical guidelines for developing professional, high-quality software. His first book, the critically acclaimed *Writing Solid Code* (Microsoft Press, 1993), focuses on strategies that programmers can use to write bug-free programs. It won a prestigious *Software Development* Jolt Productivity Award and awards from the Society for Technical Communication in 1994.

Steve lives in Seattle, Washington, with his wife, Beth, and their Airedale terrier, Abby. He can be reached at *stephenm@stormdev.com* or *microsoft!storm!stephenm*.

The manuscript for this book was prepared using Microsoft Word 5.0 for the Macintosh and submitted to Microsoft Press in electronic form. Galleys were prepared using Microsoft Word 2.0 for Windows. Pages were composed by Microsoft Press using Aldus PageMaker 5.0 for Windows, with text and display type in Palatino. Composed pages were delivered to the printer as electronic prepress files.

Cover Designer
Rebecca Johnson

Interior Graphic Designer
Kim Eggleston

Principal Compositor/Illustrator
Peggy Herman

Principal Proofreader/Copy Editor
Deborah Long

Indexer
Julie Kawabata