CREATING A
SOFTWARE
ENGINEERING
CULTURE

Karl E. Wiegers

FREE SAMPLE CHAPTER
SHARE WITH OTHERS
CREATING A SOFTWARE ENGINEERING CULTURE
Also Available from DORSET HOUSE

**Complete Systems Analysis: The Workbook, the Textbook, the Answers**  
by James & Suzanne Robertson  
foreword by Tom DeMarco  
ISBN: 0-932633-50-1  
Copyright ©1998, 1994  
624 pages, softcover

**Exploring Requirements: Quality Before Design**  
by Donald C. Gause and Gerald M. Weinberg  
ISBN: 0-932633-13-7  
Copyright ©1989  
320 pages, hardcover

**Managing Expectations: Working with People Who Want More, Better, Faster, Sooner, NOW!**  
by Naomi Karten  
foreword by Gerald M. Weinberg  
ISBN: 0-932633-27-7  
Copyright ©1994  
240 pages, softcover

**Peopleware: Productive Projects and Teams, 2nd ed.**  
by Tom DeMarco and Timothy Lister  
ISBN: 0-932633-43-9  
Copyright ©1999  
264 pages, softcover

**The Psychology of Computer Programming: Silver Anniversary Edition**  
by Gerald M. Weinberg  
ISBN: 0-932633-42-0  
Copyright ©1998, 1971  
360 pages, softcover

**Quality Software Management Series**  
by Gerald M. Weinberg  
- **Vol. 1: Systems Thinking**  
  ISBN: 0-932633-22-6  
  Copyright ©1992  
  336 pages, hardcover  
- **Vol. 2: First-Order Measurement**  
  Copyright ©1993  
  360 pages, hardcover  
- **Vol. 3: Congruent Action**  
  Copyright ©1994  
  328 pages, hardcover  
- **Vol. 4: Anticipating Change**  
  ISBN: 0-932633-32-3  
  Copyright ©1997  
  504 pages, hardcover

**The Secrets of Consulting:**  
**A Guide to Giving and Getting Advice Successfully**  
by Gerald M. Weinberg  
ISBN: 0-932633-01-3  
Copyright ©1988  
248 pages, softcover

**Surviving the Top Ten Challenges of Software Testing:**  
**A People-Oriented Approach**  
by William E. Perry and Randall W. Rice  
Copyright ©1997  
216 pages, softcover

Find Out More about These and Other DH Books:  
Contact us to request a Book & Video Catalog and a free issue of *The Dorset House Quarterly*, or to confirm price and shipping information.
To

Pretzels the Clown
This page intentionally left blank
Acknowledgments

The dedicated critiquing efforts of Marcelle Bicker, Tim Brown, Lynda Fleming, Anne Fruci, Kathy Getz, Leslie Hill, Trudy Howles, Kevin Jameson, Tom Lindsay, Lori Schirmer, Doris Sturzenberger, Mike Terrillion, and Nancy Willer led to countless suggestions for improvement. These reviewers caught many errors; any that remain are entirely my responsibility. I also appreciate the helpful comments provided by Linda Butler, Henrietta Foster, Beth Layman, Mark Mitchell, Carol Nichols, Tom Sweeting, and Peter Szmyt.

None of the achievements made by our Kodak software groups would have been possible without the active involvement of all team members. I am indebted to the original members of the Photographic Applications Software Team, who worked with me as we learned how to build better software from January 1990 through April 1993. We learned a lot from each other, built some useful applications, and had fun in the process. Many thanks to Ray Hitt, Tom Lindsay, Mike Malec, and Mike Terrillion. Mike Terrillion also allowed me to describe his work correlating the number of requirements in a software requirements specification with the work effort needed to deliver the application.

Thanks go as well to John Scherer and Jim Terwilliger, managers who didn’t know a lot about software, but were willing to learn. They provided solid management support and encouragement during the early years of our process improvement efforts. They also understood the importance of providing recognition for the team’s efforts to improve our work and to share our approaches with other software groups at Kodak. Judy Walter and Jeffrey Yu in the Kodak Research Library kept me in touch with the software literature by responding quickly to a steady stream of requests for books and articles.

I am grateful to the staff at Dorset House, particularly Wendy Eakin, David McClintock, Ben Morrison, and freelance editor Teri Monge, for their careful review and editing, insightful recommendations, and support throughout this project.
Special thanks go to Professor Stanley G. Smith of the University of Illinois at Urbana-Champaign, who taught me a lot more in graduate school than how to do physical organic chemistry.

On the home front, I wish to thank my cat, Gremlin, for keeping my lap warm whenever I try to read, and the musical genius of “Weird Al” Yankovic, for helping to put life into proper perspective. My deepest appreciation goes to my wife, Christine Zambito, for her limitless support, encouragement, listening ability, and sense of humor. Living with a clown is just slightly less peculiar than living with a computer person.
Permissions
Acknowledgments

The author and publisher gratefully acknowledge the following for their permission to reprint material quoted on the cited pages.

p. 16: Material from Yeager, An Autobiography by Chuck Yeager and Leo Janos, p. 235, reprinted by permission of Bantam Doubleday Dell Publishing Group. Copyright © 1985. All rights reserved.

pp. 23, 45: Material from Peopleware: Productive Projects and Teams by Tom DeMarco and Timothy Lister, pp. 135 and 12, reprinted by permission of Dorset House Publishing. Copyright © 1987. All rights reserved.

p. 35: Material from Managing Software Maniacs by Ken Whitaker, p. 45. Copyright © 1994, published by John Wiley & Sons. All rights reserved.


pp. 61, 214: Material reprinted from Software Reliability by Glenford J. Myers, pp. 39, 189-95. Copyright © 1976, published by John Wiley & Sons. All rights reserved.


p. 85: Figure 6.3 reprinted from IEEE Std 830-1993 IEEE Recommended Practice for Software Requirements Specifications, Copyright © 1994 by the Institute of Electrical and Electronics Engineers, Inc. The IEEE disclaims any responsibility or liability resulting from the placement and use in this publication. Information is reprinted with the permission of the IEEE.


p. 265: Figure 15.4 adapted from Robert B. Grady, "Practical Results from Measuring Software Quality," Communications of the ACM, Vol. 36, No. 11 (November 1993), p. 65. Figure adaptation courtesy of ACM. Copyright © 1993 Association for Computing Machinery, Inc. Reprinted with permission.

p. 316: Ratio provided by private communication from Capers Jones, Chairman, Software Productivity Research, Inc. Used by permission.

p. 326: Figure 20.1 reprinted courtesy of Corporate Ergonomics, Eastman Kodak Company.
Chapter 3: Recognizing Achievements Great and Small 35

The Importance of Being Visible 38
The Importance of Management Attitude 39
Rewards for a Job Well Done 40
Summary 41
Culture Builders and Killers 42
References and Further Reading 43

Chapter 4: So Much to Learn, So Little Time 45

What to Learn 47
Where to Learn 49
  Professional Seminar Sources 49
  Technical Conferences 50
  Publications 51
  Videotape Instruction 53
  On-Line Information Sources 53
  Professional Societies and Certification 53
Summary 55
Culture Builders and Killers 55
References and Further Reading 56

Part II: In Search of Excellent Requirements 59

Chapter 5: Optimizing Customer Involvement 61

Software Requirements: The Foundation of Quality 62
The Need for Customer Involvement 64
The Project Champion Model 66
Project Champion Expectations 69
When the Project Champion Model Fails 73
Summary 75
Culture Builders and Killers 75
References and Further Reading 76

Chapter 6: Tools for Sharing the Vision 78

Use Cases 79
Software Requirements Specifications 83
Dialog Maps 91
Prototypes 93
Requirements Traceability Matrices 96
Part IV: The Bug Stops Here  163

Chapter 11: The Quality Culture  165

The Cost of Quality  167
Assuring Software Quality  168
How Dense Are Your Defects?  170
Lines of Code versus Application Functionality  172
How Good Is Good Enough?  174
An Assault on Defects  178
Explicit SQA Responsibilities  181
Why Do We Think Quality Practices Pay Off?  184
Summary  185
Culture Builders and Killers  185
References and Further Reading  186

Chapter 12: Improving Quality by Software Inspection  189

Inspections and Culture  190
Benefits of Inspections  190
Inspections, Walkthroughs, and Reviews  194
Guiding Principles for Reviews and Inspections  198
Keeping Records  201
Making Inspections Work in Your Culture  204
Summary  207
Culture Builders and Killers  207
References and Further Reading  208

Chapter 13: Structured Testing  211

Testing and the Quality Culture  212
A Unit Testing Strategy  215
Cyclomatic Complexity and Testing  218
Test Management and Automation  220
Structured Testing Guidelines  222
Summary  224
Culture Builders and Killers  224
References and Further Reading  225
Part V: Methods, Measures, and Tools  229

Chapter 14: The CASE for Iteration  231
Types of CASE Tools  233
Hypes of CASE Tools  235
Lessons from Our CASE History  236
Fitting CASE into Your Culture  244
Other Benefits from CASE  247
Culture Change for CASE  248
Summary  249
Culture Builders and Killers  250
References and Further Reading  251

Chapter 15: Control Change Before It Controls You  254
Benefits of a Problem Tracking System  255
A Software Change Management Case Study  256
The Software Change Control Board  261
How Change Control Can Simplify Your Life  262
Learning from Bug Detection Trends  263
Proactive Failure Reporting  265
Making Change Management Work in Your Culture  267
Summary  268
Culture Builders and Killers  269
References and Further Reading  270

Chapter 16: Taking Measures to Stay on Track  272
Why Measurement Programs Fail  273
Metrics Programs Don’t Have to Fail  276
What to Measure  276
How to Design Your Metrics Program  278
Summary  282
Culture Builders and Killers  283
References and Further Reading  284

Chapter 17: Case Study: Measurement in a Small Software Group  287
Software Work Effort Metrics  287
Trends and Applications  295
Metrics-Based Project Estimation  297
Lessons from Work Effort Metrics 300
Predicting Maintainability with Metrics 300
Summary 301
Culture Builders and Killers 302
References and Further Reading 302

Chapter 18: If It Makes Sense, Do It 304
Summary 308
Culture Builders and Killers 308
References and Further Reading 309

Part VI: What to Do on Monday 311

Chapter 19: Action Planning for Software Managers 313
Action Item Menu 315
Summary 318
References and Further Reading 318

Chapter 20: Action Planning for Software Engineers 319
Action Item Menu 320
Building a Healthy Workplace 324
Summary 327
References and Further Reading 327

Epilogue 329

Appendix A: Sources for Continued Software Learning 331
Appendix B: Contact Information for Selected Resources 337

Bibliography 339
Author Index 349
Subject Index 351
Reviewers' Comments 359
Figures and Tables

Figure 1.1. A software engineering culture. 5
Figure 1.2. Characteristics of Constantine’s four organizational paradigms. 13
Table 1.1. Characteristics of a Structured Open Team. 14
Table 1.2. Aspects of “Roman” versus “Greek” Software Engineering Cultures. 15
Table 1.3. Software Engineering Cultural Principles. 17
Figure 2.1. The five dimensions of a software project. 29
Figure 2.2. Flexibility diagram for an internal information system. 30
Figure 2.3. Flexibility diagram for a quality-driven application. 31
Figure 2.4. Flexibility diagram for a competitive commercial application. 31
Figure 2.5. Sample form for documenting the negotiated dimensions for a project. 32
Figure 3.1. The simplest form of recognition. 36
Figure 3.2. Priorities for the enlightened software manager. 40
Figure 4.1. Growing better software engineers through continuing education. 47
Table 4.1. Technical and Nontechnical Skills for Software Developers. 48
Table 4.2. Professional Computing Societies. 51
Table 4.3. Professional Computing Certifications. 54
Figure 4.5. “Specification” for a Stonehenge replica. 62
Figure 5.2. The software development expectation gap. 63
Table 5.1. Effective Techniques for Requirements Engineering. 64
Figure 5.3. Project champion model for a large project. 68
Figure 5.4. The context diagram shows what is inside the system and what is not. 70
Figure 6.1. Process and deliverables from the use case method. 80
Figure 6.2. Use case flipchart created in a workshop with user representatives. 81
Figure 6.3. IEEE template for software requirements specifications. 85
Table 6.1. Characteristics of a High-Quality Requirements Statement. 87
Table 6.2. Some Quality Attributes for Software Operation. 88
Table 6.3. Some Quality Attributes for Software Revision. 88
Table 6.4. Some Quality Attributes for Software Transition. 89
Figure 6.4. Sample dialog map. 92
Figure 6.5. Prototyping reduces the software development expectation gap. 94
Figure 6.6. Sample requirements traceability matrix. 96
Figure 7.1. Activities involved in the effective practice of software engineering. 107
Figure 7.2. A process improvement task cycle. 113
Figure 7.3. Template for action planning. 114
Figure 8.1. Add new skills to your toolbox for future projects. 120
Figure 8.2. Emotional reactions to pressure for change. 122
Figure 9.1. Several evolutionary scales for software development. 128
Figure 9.2. Software Engineering Institute’s Capability Maturity Model for software. 129
Table 9.1. Key Process Areas for the Repeatable Level. 132
Table 9.2. Key Process Areas for the Defined Level. 133
Table 9.3. Key Process Areas for the Managed Level. 133
Table 9.4. Key Process Areas for the Optimizing Level. 134
Figure 9.3. Where do you fit along the CMM philosophy spectrum? 137
Figure 10.1. Software guidelines, procedures, and standards. 148
Figure 10.2. Sample source file header documentation template. 153
Table 10.1. IEEE Software Development Standards. 157
Table 11.1. Some Components of the Cost of Quality for Software. 167
Figure 11.1. Internal rework and post-release maintenance. 169
Table 11.2. Source Statements, Assembler Instructions, and Function Points. 171
Table 11.3. Defect Levels Associated with Different Sigma Values. 172
Figure 11.2. Example of how to calculate defect detection efficiencies. 176
Table 11.4. Reaching Six-Sigma Quality in a C Program. 177
Figure 11.3. A customer/supplier process model for software. 179
Table 11.5. Allocating Software Quality Responsibilities. 183
Figure 11.4. Trends in defect-correction work effort over five years. 184
Figure 11.5. Increase in the cost of fixing bugs throughout the life cycle [Boehm, 1981]. 191
Table 12.1. Benefits from Software Inspections. 192
Table 12.2. Errors Found by Code Inspection or Testing. 193
Figure 12.1. Stages in a formal software inspection process (adapted from [Ebenau, 1994]). 195
Figure 12.2. Defects found versus inspection rate. 200
Figure 12.3. Inspection summary report. 202
Figure 12.4. Inspection issues list. 203
Figure 12.5. Inspection issues list. 204
Figure 12.6. Record of code reviews carried out. 205
Table 13.1. Survey of Testing Practices Among Kodak R&D Software Engineers. 212
Table 13.2. Myers' Software Testing Axioms. 214
Figure 13.1. Data flow model for operation of an automated test driver. 221
Figure 13.2. Sample test report from the Graphics Engine. 222
Figure 14.1. Some upper-CASE and lower-CASE tool categories. 223
Figure 14.2. Sample data flow diagram that will generate a validation error. 234
Figure 14.3. Different platforms used for development steps. 243
Figure 14.4. The data stores connected to a DFD process define its external interfaces. 248
Figure 15.1. Architecture of the SWCHANGE problem tracking system. 257
Figure 15.2. SWCHANGE entry submission screen. 258
Figure 15.3. Example of reliability modeling from defect discovery rates [Walsh, 1993]. 263
Figure 15.4. Categorization of sources of software defects at Hewlett-Packard [Grady, 1993]. 264
Table 15.1. Top Level of Beizer's Taxonomy of Bugs [Beizer, 1990]. 266
Figure 15.5. Sample e-mail message from a trapped REXX error. 267
Figure 16.1. A recommended starter set of software metrics. 277
Table 16.1. Some Measurable Dimensions of Software Products, Projects, and Processes. 279
Figure 16.2. Sample goal/question/metric strategy for a goal of early retirement. 280
Figure 16.3. Goal/question/metric feedback model for software metrics. 281
Figure 17.1. Work effort distribution over time for one software project. 291
Figure 17.2. Sample work effort distribution report for one project. 295
Figure 17.3. New development work effort distribution for one software group, by year. 296
Figure 17.4. Development time as a function of number of requirements. 298
Table 18.1. Relating Software Engineering Practices to Organizational Goals. 306
Figure 20.1. Recommended configurations of a safe computer work environment. 326
Table A.1. Some Suppliers of Software Training Seminars. 332
Table A.2. Some Software Development and Quality Conferences. 333
Table A.3. Some Software Engineering and Software Quality Periodicals. 334
Table A.4. Sources of Information about Testing Tools. 335
Table A.5. Sources of Information about CASE Tools. 335
Preface

Rarely has a professional field evolved as rapidly as software development. The struggle to stay abreast of new technologies, to deal with accumulated development and maintenance backlogs, and to cope with people issues has become a treadmill race, as software groups work hard just to stay in place. A key goal of disciplined software engineering is to avoid the surprises that can occur when software development goes awry. Software surprises almost always lead to bad news: canceled projects, late delivery, cost overruns, dissatisfied customers, and unemployment because of outsourcing.

The culture of an organization is a critical success factor in its efforts to survive, improve, and flourish. A culture based on a commitment to quality software development and management differentiates a team that practices excellent software engineering from a gaggle of individual programmers doing their best to ship code. In a software engineering culture, the focus on quality is present at all levels—individual, project, and organization.

In this book, I share a cultural framework that was effective in improving the results obtained by several software groups at Eastman Kodak Company. Most of our projects involved small teams of one to five developers, with typical durations of six months to two years. Each part of the book discusses several guiding principles that shaped the way we chose to create software. I also describe the specific software engineering practices that we adopted to improve the quality and productivity of our work. We believe a culture based on these principles and practices has improved our effectiveness as software engineers, the relationship and reputation we have with our customers, and our level of collaborative teamwork. Many of the experiences related and suggestions offered are most relevant to workgroups of two to ten engineers. Since even large software products are often constructed by small teams of engineers working together, these technical activities are applicable in a wide variety of organizations.
With this book I hope to reach first-line software managers, project leaders, and practitioners who wish to drive progress toward an improved, quality-oriented culture in their organization. My goals are to provide practical ideas for immediately improving the way a team performs software engineering, and to show that continuous software process improvement is both possible and worthwhile. I am assuming that the reader has the ability to actually change the culture of his software group, or at least to positively influence those who can drive changes.

I present here a tool kit composed of many ideas and practices for those who wish to improve the quality of the software they develop, along with case studies of how these methods really worked. Our groups have applied all the methods described, and I have used nearly all of them personally. Every anecdote is real, although the names have been changed. While not every team member has used every good method on every project, we invariably obtained better results when we applied these solid engineering practices than when we did not.

An organization grows a quality-directed software culture by blending established approaches from many sources with locally developed solutions to specialized problems. To help point toward useful sources in the voluminous software literature, each chapter provides an annotated bibliography of references and additional reading materials. The references I feel are particularly valuable are marked with a bookshelf icon.

Each chapter contains several “Culture Builder” tips (marked with a handshake icon), which are things a manager or project leader can do to promote an attitude and environment that leads to software engineering excellence. “Culture Killers” are also described, and are marked with a skull and crossbones warning icon. Culture killers are management actions that will undermine a team devoted to superior software engineering or prevent such a culture from developing. Sadly, many of these are real examples. You can probably think of other culture killers from your own experience, as either victim or unknowing perpetrator. Although both builders and killers are written in the form of recommendations, remember that the culture killers are tongue-in-cheek. Don’t rush into work next Monday with an agenda of action items selected from the culture killers!

Some of the experiences of our software groups at Kodak were published originally in the following articles; material is included here with permission from the publishers:


This page intentionally left blank
CREATING A
SOFTWARE
ENGINEERING
CULTURE
This page intentionally left blank
Chapter 3

Recognizing Achievements
Great and Small

Another major software player decided to “dangle” incentive money in front of its developers. . . . The dictated delivery date came and went and no product had been delivered. . . . Senior management withdrew the incentive and threatened jobs since developers were employed to deliver even without the need for extra compensation. Some of the software engineers (the good ones) left the company, morale was horrible, and by the time a product was delivered, the company’s market share had eroded. What a terrible lesson to learn.

—Ken Whitaker, Managing Software Maniacs

When I first became the supervisor of the Kodak software group in which I had worked for several years, I initiated a simple (and slightly corny) recognition program. When someone reached a minor project milestone or made a small contribution such as helping another team member with a problem, I gave him a package of M&M® candies, with a message tag attached expressing congratulations or thanks, as appropriate. Bigger achievements generated bigger bags of M&Ms, or something more tangible. It wasn’t much, but it was more than we were used to.

As I expected, the candy disappeared immediately, but I was pleasantly surprised to see that some people kept the message tags visible around their desks. To them, the important thing was not the bag of candy, but the words indicating that their manager noticed and valued the progress being made. It soon became apparent that group members preferred to have the presentations made publicly at our weekly team meetings, indicating their desire for peer recognition of even small achievements.

M&Ms worked with our group, but some other social recognition technique might work better for you. We also gave this sort of micro-recognition award to people outside the group who helped us in some way. It brought smiles to their faces and goodwill to our relationships. However you choose to do it, appropriate praise and commendation help to build the culture of teamwork and striving for excellence that we all want, and it can motivate your team members to do an even better job in the future, since they know you appreciate their efforts.
The form and extent of recognition and reward is a visible indication of an organization’s culture. If managers believe the employees are lucky just to have jobs, they won’t go out of their way to offer even small gestures of appreciation or congratulations. Conversely, in a market characterized by competitive hiring and high staff turnover, an effective recognition program can help retain talented developers. M&Ms won’t make up for low salaries or unpleasant working conditions, but simple recognition is an important step in the right direction.

Software engineers are like other people (well, pretty much): We want to be appreciated, and we appreciate being wanted. Besides the internal satisfaction we obtain from interesting, challenging work and the tangible compensation in our paychecks, we want to feel that our efforts are noticed and valued by those around us. We all enjoy receiving compliments, especially when they come from various sources: peers, customers, team leader, senior managers, professional associates.

Praise for a job well done should be timely, direct, personal, and specific (see Fig. 3.1). If you are a manager, don’t wait until performance appraisal or salary adjustment time rolls around to pass along some positive feedback. Tell the individual exactly what he or she did well and why you appreciate it. A mumbled “Keep up the good work” in the hallway is more likely to confuse than motivate the recipient. Your team members must know you are sincere when you offer compliments on their work. Though many people feel awkward when they receive a compliment, they appreciate that someone took the trouble to say how pleased he was with your work.

Figure 3.1: The simplest form of recognition.
Recognition can take many forms. Donna Deeprose presents more than one hundred ideas about how to select appropriate and meaningful recognition and rewards for your employees [Deeprose, 1994]. Ask the members of your team what kinds of recognition are important to *them*. Do they prefer a public pronouncement at the weekly group meeting, or are they more comfortable with private ceremonies? Should recognition come just from you, or is it meaningful to have higher-level managers participate in certain recognition activities? Tailor the reinforcement you offer to be significant to the recipients. The following paragraphs describe some of the things the groups in which I have worked have done to express appreciation and to build a positive culture. Some of these apply to individuals, while others are appropriate for teams of people.

Spend a few moments at weekly team meetings to give everyone a chance to pass along some positive reinforcement ("R+") to others. Did a coworker help you solve a problem this week? Did someone take some action out of the ordinary that helped the team? If yes, say so! The group may be uncomfortable when you first try this, but they should warm to the idea over time. If group members are so isolated from each other that no one ever has any R+ to pass along, you may have some serious issues of team dynamics to address.

A traveling trophy that moves from project to project can be used to recognize team achievements. In keeping with the M&M motif, we used a framed three-pound M&M bag (empty, sad to say) as a traveling trophy. Recipients displayed the prize in their office area until another project reached a milestone worthy of recognition. The trophy was ceremoniously passed from one team to the next in our group meeting. If you try something like this, be sure to keep the trophy traveling every few weeks, or its significance becomes lost.

Food and entertainment are also good ways to recognize someone’s contributions or special achievement. Taking the team to a celebration luncheon when a milestone is reached can be fun for everyone involved. A gift certificate for dinner at a restaurant gives an individual recipient a chance to celebrate privately with friends or family. Whenever a member of my team earned a college or advanced degree, my wife and I took him and his significant other out to dinner. Maybe going to dinner with the boss is not everyone’s idea of a great time, but it worked for us. On another occasion, I gave each team member a pair of movie passes as a symbol of how much I appreciated their time and teamwork when, during an intense period of selecting new members for our group, the entire team of ten pitched in on short notice to participate in the interviews. It was a small gesture but a sincere way of saying, “Thanks for the help, gang.”

Recognize individuals outside your group for their contributions as well. It’s amazing how much future cooperation you can secure with a simple gesture of appreciation. As the recipient of a few such gestures myself, it always makes me feel good to know that someone really appreciated something I did, however routine it may have seemed to me. We have thanked project customer representatives by taking them to lunch, giving them certificates to hang on the wall, and bestowing restaurant gift certificates upon those who shouldered the most responsibility.
One customer even reciprocated, throwing a lunch bash for the development group. This sort of customer-developer interaction helps build a culture of constructive teamwork.

Be sure to recognize people for attaining minor milestones, as well as when they complete a big project. Interim pats on the back help provide team members with the incentive to keep pushing ahead. Again, it says to the recipient, "Congratulations on making progress toward your goals."

As a manager, you must actively look for recognition opportunities, and seize the moment as soon as you spot one. The manager who realizes an achievement is worthy of formal recognition but waits to figure out what he wants to do about it, and waits to execute his plan, may provide recognition too far removed from the achievement itself to mean much to the recipient. "Oh, so you finally noticed what I did," is a typical unspoken reaction to a belated recognition effort. A manager who expects such opportunities automatically to pop up in front of him will miss many of them, and will not deliver consistent recognition messages. Also remember that managers who fail to reward exceptional contributions are sowing the seeds of discontent. The absence of well-deserved recognition is highly demotivating.

The Importance of Being Visible

The antithesis of being recognized for your achievements is feeling that your managers do not know who you are, what you do on the project, how you do it, or what your contributions are to the company as a whole. When was the last time your supervisor stopped by your office just to say hello and to ask how things are going? How about a visit from a manager farther up the corporate hierarchy?

Some people are uncomfortable with an unannounced manager visit, but others welcome the opportunity to share their concerns and show the boss what they are working on. "Management By Walking Around" is one way managers express interest in the individuals in their organization; it can and should be practiced by managers at all levels. Think about it: If you are a first-line supervisor and you never see your boss in the engineering staff's offices, chances are you'll conclude that he or she is hopelessly out of touch with the group.

People are more motivated to put in extra effort when they know the higher-ups value it. We have all worked for managers who represented the other extreme, having a limited awareness of the group’s challenges and contributions. How excited can you get about trying to please such managers? As a supervisor, make sure you really know what the engineers in your department are doing. Who are the contributors, the innovators, the leaders? Who is just along for the ride? The team members will have no confidence that they can get fair performance appraisals if they rarely talk to their supervisor. Your employees need to have adequate opportunities to explain what they do and the problems they face.

When appropriate, a manager should also show interest in professional activities that are unrelated to specific projects. It is discouraging to put in extra time to
prepare a presentation for a local software conference, then not to see your boss’s face in the audience. Telling you he is pleased you are doing something extra is not nearly so meaningful as if he actually shows up for the event. Remember all those plays, dance presentations, and concerts you sat through because your children were on stage? Being a good manager demands some of the same actions as being a good parent. It means a lot to know your supervisor cares about what you are doing.

The Importance of Management Attitude

Here’s a radical idea: Think of you, as a manager, working for the people who report to you, as opposed to the more traditional view of subordinates working for the supervisor. The people you supervise are the customers for your leadership and management services, including

- coaching and mentoring
- setting project goals and priorities
- resolving problems and conflicts
- providing resources
- evaluating performance and providing feedback
- career development
- leading process improvement efforts
- providing technical guidance when appropriate

Give your own people top priority over the demands of others for your time. Your priorities as a manager should be those shown in Fig. 3.2.

Unfortunately, too many managers are busy looking up the corporate organization chart, not down. The sequence in Fig. 3.2 is frequently inverted, with the key driver being what you think will make your own boss happy. In a healthy, congruent workplace, the boss should be thrilled if you are meeting the needs of your team and its customers. Not everyone is fortunate enough to work in such an enlightened environment; priorities are usually defined by the perception of who you think you have to please to keep your job.

One way a manager can tell if his priorities are straight is whether he ever receives any recognition from his team members. It meant more to me to get an R+ from one of the people I supervised than to get one from my own supervisor. If you are an individual contributor, remember to thank your managers for special
contributions they make or extra help they provide to you. Managers are people, too, with the same desire to be appreciated that anyone else has.

![Figure 3.2: Priorities for the enlightened software manager.](image)

**Provide services to your team members.**

**Satisfy your department’s customers.**

**Work on your own projects.**

**Please your own managers.**

**Rewards for a Job Well Done**

The skillful manager will use recognition and rewards to reinforce desired behaviors, rather than offering tangible incentives to individuals or teams to achieve specific goals. The main incentive for most software people to go to work each day is the opportunity to work on challenging projects with stimulating colleagues, in an environment that encourages quality work, in which software skills can be applied and extended.

Dangling extra cash as a carrot in front of a software engineer to try to get him to work faster-harder-longer can backfire [Whitaker, 1994]. If the ambitious goals are not met, and developers know the pot of gold at the end of the rainbow won’t be forthcoming, how do you keep them motivated? Withdraw the incentives (thereby destroying morale), or renew the incentives (thereby showing that falling short of management’s outrageous goals is just as meritorious as achieving them)? Either way, everyone loses. Instead of offering incentives, design a reward program that matches your organization’s culture and means something to your team members. Motivate your team through frequent interim recognition activities, and reward them for a job well done when the job really is done. People should also be rewarded for taking intelligent risks, even if a great notion or great effort doesn’t
pay off for reasons outside the development team's control. Public rewards indicate to the rest of the group those behaviors you feel are desirable.

Rewards can be monetary or non-monetary. Slipping a few bonus bills in the pay envelope may seem like a sure way to please an employee, but often something else would be preferable. Talk to your people and understand what rewards they feel are significant. The corporate culture will have some influence in selecting feasible rewards. It may be easier to give someone a substantial, but non-cash, award, such as a trip to a conference or trade show. Some companies reward employees with frequent flyer miles, which can be purchased from airlines for a few cents per mile. Some developers might enjoy extra vacation time; others might want to buy a special software package with which to experiment, just because they heard it was interesting. If someone does an exceptional job on a project, he might appreciate a mini-sabbatical, a couple of weeks set aside to work on whatever he likes.

Another kind of reward is the opportunity to work on an exciting new project. In some groups, only the old hands have the skills and knowledge to keep the legacy systems alive. Less experienced people may be assigned to projects involving newer technology, which are more fun than maintaining ancient applications. This is a good way to drive your senior staff members out of the group, to search for opportunities where they can learn contemporary skills and work on the kinds of projects they read about in computer magazines. Creative and experienced engineers don't want to be mired in legacy code for the rest of their careers. Look for ways to reward your best workers by keeping them stimulated with new learning opportunities and project challenges.

Summary

✔ People need to feel the work they do is appreciated.

✔ Receiving positive reinforcement from peers, managers, and customers is highly motivating to most people. Failing to recognize someone's exceptional contributions and major achievements is demotivating. Why do extra work if your managers don't care?

✔ Recognition says, "I appreciate your effort," "Congratulations on your accomplishment," or simply, "I noticed what you did."

✔ Find out what kinds of recognition and rewards are meaningful to your people, and tailor your R+ program accordingly to foster a culture of desirable software engineering behaviors.

✔ Recognize minor accomplishments and milestones, to motivate individuals to keep working toward their major objectives.
 Managers can build better relationships with their team members simply by understanding the work they do and showing a sincere interest in it. Talk to your people informally, to find out what they are excited and unhappy about. Sometimes, this approach will let you deal with a concern before it becomes a crisis.

 A software manager should regard the people who report to him as his most important customers. The manager's top priority should be to address the needs of his team members.

 Reward your staff members, whatever their job, for a good performance, rather than offering big incentives to induce them to do great work in the future.

**Culture Builders and Killers**

**Culture Builder:** Distribute recognition awards equitably to your group members. Don't reserve recognition events only for project leaders, members of high-profile project teams, or your senior technical people. The scale and frequency of rewards does not have to be the same for everyone—after all, people are different—but it is demoralizing for an employee to see the same coworkers being recognized repeatedly without anyone noticing his own achievements.

**Culture Builder:** Make sure you are accessible to the people who report to you. Schedule one-on-one meetings with those who desire them, at whatever interval is mutually acceptable to you and each individual. A general open-door policy is important, too, but your team members may be reluctant to bother you if they know how busy you are. They deserve a slice of your undivided attention at regular intervals.

**Culture Builder:** If you make a verbal commitment to someone for a reward, be sure to follow through on it. Forgetting that you made this promise demonstrates a lack of sincerity. Be sure to reward the right people for the right reasons. If you aren't sure who made key contributions to a successful project, find out before you present any rewards or recognition. Few things are more infuriating than seeing a person receive praise (or more) for work that was actually done by someone else.

**Culture Killer:** In an era of political correctness carried to an extreme, you don't dare run the risk of anyone crying "Discrimination!" on any basis. Therefore, it is safest to offer exactly the same kinds of recognition to
all of your team members, whether they excel in the performance of highly challenging work or they struggle to meet minimal expectations.

**Culture Killer:** Here are some good reasons to cancel, cut short, or interrupt a regularly scheduled one-on-one meeting with one of your team members:

- Someone else has already stopped by your office to chat.
- You need to work on one of your own projects.
- Your telephone rings.
- You have been invited to join a new committee that meets at that same time.
- Your boss calls another meeting for that time.
- You have to travel to another site to meet with someone else.
- You forgot about it.

Whenever anything short of a true emergency interferes with a scheduled meeting with someone you supervise, you are sending a clear message: “Everything else I have to do is more important to me than you are.”

**Culture Killer:** Offer recognition, such as a luncheon with several important managers, to an individual who has done an ordinary job on an ordinary assignment, but offer nothing to other group members for taking significant initiative that extends outside the boundaries of their assignment. Word will get around that recognition depends on who you are, not what you do.

**References and Further Reading**


Deeprose presents ten guidelines that can give your recognition program more impact. She lists one hundred ways you might provide recognition and reward, in the form of structured reward programs, spontaneous rewards, and day-to-day feedback. You can recover the cost of the book the first time you provide an employee with recognition that motivates him to work a little bit harder or smarter on the company’s behalf.

In Chapter 3, “Attracting and Keeping Developers,” Whitaker argues that managers should reward developers after the project is completed, rather than trying to motivate them by promising wonderful rewards in advance as an incentive. This book contains many horror stories of management actions that led to undesirable results.
Author Index

Ambler, S., 82, 100, 339
Ash, D., 303, 340
Bach, J., 135, 141, 271, 339
Bankes, K., 200, 208, 339
Basili, V., 278, 284, 339
Beizer, B., 146, 211, 215, 216, 217, 225, 226, 264, 266, 270, 339
Bell, R., 252, 346
Bergin, T., 234, 251, 339
Binder, R., 126, 339
Boddie, J., 56, 339
Boehm, B., 19, 190–91, 208, 297, 302, 340
Bollinger, T., 340
Bond, S., 142, 341
Brooks, R., 11, 19, 231, 340
Card, D., 278, 284, 340
Carleton, A., 143, 343
Carmel, E., 76, 344
Carnegie Mellon University/Software Engineering Institute, 128, 130, 134, 142, 149, 160, 169, 186, 340
Caswell, D., 285, 343
Chidamber, S., 279, 284, 340
Chrisis, M., 144, 345
Christerson, M., 102, 344
Clark, J., 252, 343
Cohen, R., 52, 57, 340
Coleman, D., 300, 303, 340
Constantine, L., 12, 13, 20, 79, 100, 340
Cornell, J., 93, 100, 341
Covey, S., 52, 321, 327, 341
Crosby, P., 61, 76, 127, 141, 165, 167, 187, 313, 341
Curtis, B., 136, 142, 144, 341, 345
Daskalantonakis, M., 138, 142, 273, 276, 281, 284, 341
Davis, A., 61, 78, 95, 100, 341
Deecke, D., 37, 43, 341
DeGrace, P., 14, 20, 86, 101, 233, 245, 251, 341
DeMarco, T., 3, 19, 20, 23, 45, 51, 57, 127, 254, 272, 273, 278, 279, 284, 292, 297, 303, 317, 318, 341
Deutsch, M., 86, 87, 101, 187, 341
Dion, R., 128, 142, 342
Dixon, R., 235, 251, 342
Dorfman, M., 77, 346
Dreger, J., 173, 187, 342
Ebenau, R., 192, 194, 195, 196, 200, 208, 342
Fagan, M., 192, 194, 209, 342
Falk, J., 226, 271, 344
Fenton, N., 342
Freedman, D., 201, 209, 342
Gause, D., 76, 342
Gianturco, M., 215, 226, 342
Gilb, T., 86, 89, 101, 192, 209, 342
Glass, R., 86, 87, 101, 187, 342
Goodman, P., 342
Grady, R., 192, 209, 264, 265, 270, 274, 275, 276, 281, 284–85, 342, 343
Graham, D., 209, 342
Harwin, R., 324, 327, 343
Haynes, C., 327, 343
Hefley, W., 142, 341
Herbsleb, J., 128, 143, 343
Hetzel, B., 282, 285, 343
Hughes, C., 245, 252, 343
Iannino, A., 271, 345
IEEE, 84, 101, 160, 211, 214, 226, 264, 270, 279, 285, 343
Ince, D., 156, 160, 343
Jacobsen, L., 79, 102, 344
Janos, L., 348
Johnson, M., 216, 226, 344
Johnson, M.L., 286, 346
Johnson, P., 102, 344
Kaner, C., 216, 226, 271, 344
Karten, N., 34, 344
Keil, M., 65, 76, 344
Kemerer, C., 284, 340
Keuffel, W., 57, 271, 340, 344
Kidd, J., 286, 344
Konrad, M., 142, 341
Kuzara, R., 346
Lao-tzu, 319
Layman, B., 161, 344
Lazell, R., 252, 346
Lister, T., 3, 19, 20, 23, 45, 51, 57, 127, 292, 303, 317, 318, 341
Lorenz, M., 279, 286, 344
Lowther, B., 303, 340
Mack, R., 345
Marose, B., 286, 346
McCabe, T., 216, 218, 226, 344
McConnell, S., 21, 34, 52, 57, 188, 210, 344
McGowan, C., 340
Miller, S., 142, 341
Mosley, D., 217, 219, 227, 345
Mus, J., 263–64, 271, 345
Myers, G., 61, 181, 188, 213–14, 227, 345
Nguyen, H., 226, 271, 344
Nielsen, J., 345
Okumoto, K., 271, 345
Oman, P., 303, 340
Overgaard, G., 102, 344
Page-Jones, M., 128, 143, 252, 345
Paulk, M., 128, 144, 345
Perry, D., 303, 345
Perry, W., 188, 345
Pfleeger, S., 345
Pirsig, R., 52
Port, O., 287
Pressman, R., 57, 139, 144, 345
Raynor, D., 57, 345
Rettig, M., 94, 102, 345
Robertson, J., 234, 252, 346
Robertson, S., 234, 252, 346
Roetzheim, W., 158, 161, 346
Rombach, H., 284, 339
Rozum, J., 143, 343
Rubin, H., 273, 274, 276, 286, 346
Saidian, H., 346
Sellers, D., 324, 327, 346
Shafer, L., 100, 341
Sharon, D., 234, 252, 346
Siegel, J., 143, 343
Silver, D., 77, 347
Spurr, K., 245, 252, 346
Stahl, L., 14, 20, 101, 251, 341
Staudenmayer, N., 303, 345
Strauss, S., 208, 342
Szmyt, P., 77, 346
Thayer, R., 77, 346
Tripp, L., 156, 161, 346
Van Slack, T., 209, 343
Verdago, G., 286, 346
Votta, L., 303, 345
Walsh, J., 263–64, 271, 346
Weber, C., 144, 345
Weeks, K., 217–18, 227, 346
Weller, E., 191, 192, 210, 347
Whitaker, K., 21, 35, 40, 44, 347
Wiegers, K., xx–xxi, 347
Willis, R., 101, 187, 341
Wilson, J., 244, 253, 347
Wirth, N., 347
Wood, J., 69, 77, 347
Yeager, C., 16, 22, 348
Yourdon, E., 22, 52, 58, 117, 188, 253, 309, 348
Zubrow, D., 143, 343
Subject Index

   for software engineers, 319ff.
   for software managers, 313ff.
   template for, 114
Analysis, 97, 238, 242, 249
   iteration during, 232
   paralysis, 89ff., 232
   structured, 238
Applied Computer Research (ACR), 52, 337
Appreciation, need for, 36, 41
   (see also Recognition; Rewards)
Assess-Plan-Do-Verify, 112–13, 115, 316
Assessment, process, 103, 125, 137–39, 140, 149, 151
   CMM-based, 137–38, 139, 140
   mini, 138–39
   Software Productivity Research (SPR), 139, 143
Association for Computing Machinery (ACM), 50–51, 53, 334
Books, see Software literature, books
Brainstorming, 112, 117–20, 122–24, 125, 137, 140
Bugs, 163, 165, 189, 256, 258
   (see also Defects; Errors; Fault, software)
   categories of, 264–66
   cost of correction, 190ff.
   inspections and, 192, 205
   seeded, 175–76, 207–8
taxonomy of, 264–65, 270
   trends in detection, 263–65
Capability Maturity Model (CMM), 63, 128ff., 137ff., 148, 149, 151, 169, 186, 305, 340
   key process areas (KPAs), 129–34, 136, 138, 140, 149
   levels of, 129–34, 151
   objections to, 135–37
CASE tools, 93, 96–97, 231, 233ff., 246, 335
   benefits from, 236, 237, 241, 244, 247, 322
   communication and, 235–36, 238ff., 249
   culture change and, 241, 244, 248–49
   integrated, 234, 243, 251
   iteration and, 237, 321
   lessons, 236–44
   lower-, 233, 235, 242, 246
   success with, 244, 245–46
   training and, 245, 249–51
   types of, 233–35
   upper-, 233, 237–38, 241, 244, 247
   vendors, 235, 335
Certification, see Training, professional certification
Change, 6, 12, 16, 107, 109ff., 120, 122ff., 313ff.
   (see also Change management; Software change control board)
   agent, 107, 122, 314
assessments, 258–59
control, 149, 262, 254ff., 313
cultural, 16, 107
goals and, 109–110, 112, 124, 314
incentives for, 109–110
measurements and, 111
mini-projects, 112, 115, 118, 120, 123
need for, 120
pain and, 110, 112, 130
process, 18, 313
reactions to, 6, 122, 314
request, 254, 256, 269–70
request process guidelines, 153
resistance to, 109, 110, 314
sustaining momentum,
Change management, 71–72, 89–90, 134,
229, 254, 256ff.
culture and, 267–68
system, 181, 268
Communication:
software engineers and, 78, 108
tools for, 64, 78, 98, 235–36, 238, 240, 242,
321, 335
Conferences, 50–51, 55–56, 126, 313, 333
Congruent behavior, 7, 16
COnstructive COst MOdel (COCOMO),
297ff., 302
Context diagram, 69, 70, 239, 240
Control flow diagram, 239
Control flowgraph, 217, 226
Coverage, test, see Test coverage
Culture, xix–xx, 1, 4, 6–8, 18, 46, 107, 109,
125, 131, 135, 137, 229, 313ff.
(see also Software engineering culture)
definition of, 3–4
inspections and, 190–93, 204–6
organizational paradigms and, 12–13,
122
of quality, 165–88
Customer:
(see also Customer/supplier model for
software)
change requests, 269–70
failure reports, 265, 268
integrity with, 24–26
involvement in requirements, 9, 17, 59,
61ff., 64–66, 74, 75, 119, 180, 306
needs, 5, 24, 25, 33, 61, 66, 78, 79–80, 84,
232, 317
product errors and, 166, 168
projects without, 69
quality and, 166, 175, 329
representatives, 25, 37–38, 65, 66, 68, 10,
90, 232
satisfaction metric, 277
software expertise, 25, 33
voice of, 24, 59, 63, 66, 75, 79
Customer/supplier model for software,
178–79, 214, 320
Cyclomatic complexity:
basic, 218–20
extended, 218–20
testing and, 216, 218–20
tools for, 220
Data dictionary, 236, 246, 247
Data flow diagram (DFD), 66, 86, 90, 234,
236, 239, 246, 247, 248
Defects, 124, 254, 256
assault on, 178–81
classifying, 201–2, 264–65
correction of, 184, 270, 290, 296
culture and, 254–55
density of, 61, 170–72, 185, 186, 204, 277
found by peer, 168, 190, 207
per function point, 170
names for, 165
opportunity, 170–71
prevention of, 133–34, 167–68
removal efficiency, 193, 203, 224
six-sigma quality and, 171–72, 175
tracking of, 181, 254, 268
units of, 170
Design, 5, 97–98, 133, 231–32, 238, 242, 249
iteration and, 232, 319, 320
measured aspects of, 289
modeling, 232–33
object-oriented, 233, 234
structured, 244
tools, 234
Dialog map, 91–93
Documentation, 5, 10, 71, 78, 133, 255
CASE tools and, 236, 242–43, 320
errors in, 165
guidelines for, 152–53, 156
inspection of, 191
procedure definition and, 149
templates for, 152
work effort metric, 290
Dogma, 17, 111, 120, 135–36, 229, 240, 305ff.
Education, professional, 9, 46–47, 49, 54, 55
(see also Training)
National Technological University, 53, 55, 337
on-line information, 53
resources for, 331–35, 337
Entity-relationship diagram (ERD), 86, 234, 239
Ergonomics, 11, 311ff.
environment and, 311, 314, 317, 323ff.
recommended configuration, 326
Errors:
acceptable level of, 175
healthy engineering culture and, 166
by user, 259
Estimation:
and change, 90
metrics for, 27, 297–300
of projects, 27–28, 33, 106, 124, 130, 132, 297–300, 323
tools, 233
Expectations gap, 63, 93–94
Fagan inspection, 189, 194–95, 197
Fault, software, 165, 255
objections to term, 165
Features, 28–32, 34
Flexibility diagram, 29–31, 33
Function point:
backfiring estimates, 174
counting scheme, 172–74, 187
defects per, 170
International Function Point Users Group (IFPUG), 173, 337
Functional metrics, 173, 275, 297
vs. LOC counts, 173–74
Goal/question/metric (GQM) paradigm,
278–82, 283, 292, 316
definition of, 278–79
feedback model, 280–82, 292
Guidelines, software development, 147–49, 151–55, 158
areas omitted from, 151–52, 154
glossary section, 154
local development, 149–51
review of, 155, 158
as suggested approaches, 148, 151
team involvement in, 150, 158
for user interface, 154
for user manual, 154
IEEE Computer Society, 50, 51, 53, 64, 161, 334
IEEE standards for software development, 150, 154, 155–58, 159, 182
as guides, 149, 151, 154, 305
for productivity metrics, 173
for quality metrics, 285
for software anomalies, 270
for software quality assurance plans, 182
for software requirements specification, 84–86
for test documentation, 182, 214
Inspection, software, 180, 193, 194–96
(see also Fagan inspection)
benefits of, 190–93, 207, 321
of code, formal, 193, 201
culture and, 190–93, 204–6
efficiency of, 175–76, 177
etiquette, 199
failure, reasons for, 205–6
formal, 189, 194–98
forms used in, 200, 201–4
guiding principles for, 153, 198–201
management involvement in, 195, 205
participants, 194–95, 205–7
rates, 199–200, 203
records kept, 201–4, 207
seeded errors for, 207–8
of software requirements specification, 190, 191, 193, 207
stages of, 195–98
summary report, 202–3
time for, 191, 193, 205, 206
training, 205, 206
unit testing and, 191, 193
Iteration, 229, 231ff., 249, 319
Joint Application Design (JAD), 69, 77, 79

Key process area (KPA), see Capability Maturity Model

Kiviat diagram, 29–30

Kodak software groups, vii, xix–xx
development guidelines, 151ff.
Graphics Engine testing example, 220–22
IEEE standards and, 86, 155, 182
measurement programs and, 275, 276, 281, 287ff.
metrics, 180, 292

Magazines, see Software literature, periodicals

Maintainability, predicting, 300–301

Maintenance, 124, 133, 259
bug-fixing, 163
goal to reduce, 287–88, 300
of orphan software, 185
phases, 289, 290
post-release, 169
reduced by initial quality, 178, 180

Management:
behavior, 9, 11
consensus, 109, 110
goals, 287–88, 299
leadership, 15–16, 39, 109–10, 241, 274
measurement programs and, 274, 275, 283
priorities, 4, 39–40
problem tracking, support for, 268, 270
role in leading change, 16, 314, 317
services to staff, 39–40
visibility, 38–39, 274
by walking around, 38

Manager:
action planning and, 313ff.
attitude toward quality, 166
education, 121, 245
expectations, vague, 147
inspections and, 195, 205, 207, 208
integrity with, 27–28
personal improvement and, 48
process improvement and, 314, 329–30
quality-related tasks vs., 168
tracking QA activities, 183

Maturity, software process, 127–45
(see also Capability Maturity Model; Process, software)
culture and, 6, 139–40
Measurement, software development, 119, 123, 140, 229, 272–86, 287–303
culture and, 272, 273, 290
dimensions of, 274, 278, 279
failure of, 273–75
vs. of performance, 274, 283
programs, 273
reasons for, 272, 282
stakeholders and, 274

Methodology, software development, 7, 11, 69, 75, 147, 159, 239, 304ff.
(see also Dogma)
CASE tools and, 234, 236–37, 245

Metrics, software, 131
database, 276, 294, 299, 300, 301, 302
functional, 173–74
guidelines for, 153, 155
Halstead, 220
indicators, 278
individual evaluation of, 229
maintainability and, 300–301
of maintenance efforts, 184
privacy of, 274, 294
of productivity, 288
starter set, 277–78
trends in, 263–65, 275, 288, 295–97, 301, 302
work effort, 184–85, 287–95, 296, 297, 300, 301, 303

Metrics programs, 203, 229, 273ff., 276ff., 287–303, 335
accounting system vs., 293, 302
designing, 276–77, 278–82
failure, reasons for, 273–75
to monitor progress, 314, 316, 335
for project estimation, 297–300
reporting options, 294–95, 300
as self-controlling process, 294
standards for definitions, 275

Modeling, 232ff., 236ff., 249
CASE tools and, 231–53

Object-orientation, 79, 233, 234, 239, 250, 284
PC-Metric, 220
Peer deskcheck, 197–98, 203, 206, 320–21
Peer reviews, 133, 136, 189, 190, 206, 207, 208, 231, 301, 314, 316, 319, 320
(see also Inspection, software; Fagan inspection; Reviews, by group)
People Capability Maturity Model (PC-CMM), 136, 142
Performance:
  self-assessment by manager, 314, 315
  of software developers, 3, 124, 208
  of software groups, 3, 313ff.
Personal Software Process (PSP), 134–35, 143, 166–67, 187
Problem tracking system (PTS), 255ff.
  commercial, 256–57
  as communication tool, 259, 268
  culture and, 259, 260, 264, 269
  databases used in, 257, 261
  philosophy of using, 260–61, 268
  reports from, 256, 258–59, 261, 269
  repository, 255–56
  response time, 260, 269
  rules for, 260–61
  submission example, 258–59
  testing, uses with, 263ff.
Procedures, software development, 103, 146–62
(see also Guidelines)
  documentation of, 149
  engineers, new, and, 159
  project schedule and, 160
  realistic, 155
  resistance to, 150
  as step-by-step formula, 148
Process, software, 103ff.
(see also Assessment, process; Maturity, software process; Process improvement, software)
  customer/supplier relationships, 178–79
  defect prevention and, 168
  evolutionary scales for, 127–28
  maturity, 127–45
Process improvement, software, 4, 7, 8, 16, 17, 103ff., 106, 114, 117ff., 120, 125, 134, 137–38, 141, 149, 254
  culture and, 103, 106, 114
  errors as indicators for, 166
  evolutionary, 108–109, 114, 130
  goals for, 108, 139, 278, 313ff., 319ff.
  objectives of, 136
  principles of, 106–112
  problem tracking system and, 259–60
  time and, 106, 120
Productivity:
  CASE tools and, 321
  quality and, 163, 185
Programmers, 160, 168, 287
  reviews and, 190, 208
  standards and, 150, 307
  testing and, 181, 214, 225
Programming, egoless, 8, 190, 210
Project, software development:
  change request tracking, 268
  customer/supplier relationships, 178–79, 214, 320
  dimensions of, 28–33, 34
  estimation metrics, 297–300
  improvement goals, 313ff.
  internal rework and, 168–69, 292
  measurement of, 272–73
  negotiation, 27, 29, 33
  peer-review activity and, 189, 314, 316
  pilot, 150
  planning, 5, 28, 69, 124, 131, 132, 149, 205
  schedule, 28–32, 34, 186
  work effort distribution, 291
Project champion, 59, 64, 66ff., 73–74, 75, 79, 81, 90, 91, 246
  change management, 258–59, 260, 262
  change requests and, 256–57
  characteristics of ideal, 67
  expectations of, 69–73
  resource team, 68–69
  responsibilities, 66, 152, 153, 194, 258–59
Prototypes, 64, 69, 89, 93ff., 95ff., 231
  expectation gap and, 93–94
Publications, 51–52, 334
(see also Software literature)
Quality, 23, 28–32, 33, 59, 175, 186, 193
(see also Conferences; Quality control;
  Six-sigma quality; Software literature; Software quality; Software quality assurance; Software quality engineering)
as conformance to requirements, 61, 76
CASE tools and, 235, 241, 249
cost of, 46, 167–68, 180, 187, 191, 303, 329
culture and, 4–5, 163, 165–88, 301, 315
filters, 90, 224
improvement, first initiative, 186
individual concern for, 163, 166, 178, 319ff.
inspections and, 189–200
Management Maturity Grid, 127
methods, 193
productivity and, 163, 168, 184–85, 186, 313ff.
schedules and, 186
work effort metrics and, 184–85
Quality Assurance Institute (QAI), 332, 333
Quality control, 170, 175, 181, 268
defined, 169–70
efficiency, 175–76
Recognition, 35ff., 112, 313ff., 317
(see also Rewards)
culture and, 35–36, 40, 41, 123
forms of, 36–37
importance of interim, 38, 40
M&Ms, 35–37
of outside contributors, 37–38
timeliness of, 36
Repetitive strain injuries (RSI), 11, 311, 324
Requirements, 26, 59ff., 62–64, 102, 105, 131
(see also Requirements specification, software)
analysis, 63, 66, 76, 79, 91, 97, 133, 233
vs. business rules, 81, 82
changing, 26, 65, 89, 97, 98, 132, 254
correlation with effort, 297–99
creeping, 63, 76, 97
culture and, 5, 59, 63, 75, 98
cursory understanding of, 231
customer involvement with, 9, 17, 59, 61ff., 64–66, 74, 75, 119, 180, 306
documenting, 83–89, 256, 306
engineering, 61, 64
essential vs. chrome, 70
external functional, 215
gathering, 69–70, 256
inspection of, 190, 191, 192, 207
marketing department and, 65–66
new, 32–33, 66, 90
quality attributes negotiation, 175
tagging, 86
techniques for, 64
traceability matrix, 64, 96–97, 99, 183
Requirements specification, software, 5, 62, 64, 71, 73, 76, 82, 83–90, 91–92, 96–100, 105, 119, 124, 125, 149, 150, 180, 243, 289, 296, 297–98
IEEE, 84–86, 99, 101
inspection, 83, 90
iteration, 232
pitfalls, 89–90
quality attributes in, 86–89
Reuse, 84, 87, 98, 115
benefits of, 322–23
modeling and, 247
testing and, 214
Reviews, 10
(see also Inspection)
abuses of, 190
of development guidelines, 155, 158
development guidelines used in, 159
of documentation, 159
formal technical, 189
guiding principles for, 198–201
by group, 197–98, 203
impact on quality, 180, 321–22
of requirements specification, 90
by team, 192
Rewards, 40–41, 46–47, 107
(see also Recognition)
examples of, 41
instead of incentives, 40–41, 42
Rework, 167–69, 292
Risk, 188, 193, 221, 223, 232, 285
analysis and quality control, 175
-based decision after testing, 264
of failure, 175, 245
laboratory process control example, 175
Six-sigma quality, 171–72, 174, 175, 177, 315
vs. more realistic goal, 186
ways to achieve, 177
Skills, professional, 48, 119–20, 140, 244
assessment, 47, 55, 319ff.
Software change control board (SCCB), 256, 261–62
Software configuration management (SCM), 149, 154
Software development:
   (see also Measurement, software development; Methodology, software development; Procedures, software development; Process, software; Standards, software development)
   incremental, 89, 93, 231
   lifecycle (SDLC), 152, 290–91
   phases, 289–90
Software engineering, 139, 146
   (see also Software engineering culture)
   practices vs. goals, 306
   process group (SEPG), 132
   vs. quality control, 181
   standards and, 146ff.
Software engineering culture, xix–xx, 1, 4–5, 6, 8, 55, 112, 125, 229, 231, 264, 304
   (see also Culture; Quality, culture and)
   CASE tools and, 241, 244–46, 249, 321, 335
   change management and, 229
   changing, 117, 307
   through consensus, 151
   defined, 1, 4–5
   ergonomics and, 311ff.
   errors and, 190
   Greek versus Roman, 14–15, 251
   healthy, 4, 8–10, 46, 106, 108, 307, 311
   measurement and, 273, 283, 288, 307
   outsourcing and, 5
   peer deskchecks and, 198
   principles, 16–17
   problem tracking system and, 259, 260, 264, 269
   procedures, written, and, 158
   process maturity and, 137, 139–40
   quality and, 163, 329
   standards and, 147, 306–7, 308
   testing and, 183, 212–15, 307
   tools, adoption of, 304
   unhealthy, 10–12
Software Engineering Institute (SEI), 127ff., 134, 136, 137, 140–42, 144–45, 332, 337, 340
   (see also Capability Maturity Model)
Software failure, 165–66
   proactive reporting of, 265–67
Software literature, xx, 8, 45, 51–52, 126
   books, 48, 51–52, 55, 56, 119, 304, 314, 319
   periodicals, 52, 55–56, 119, 314, 319, 334
Software quality, 59, 61–62, 75, 119, 123, 181
   acceptable level, 174
   attributes, 175
   code quality and, 174
   customer and, 166, 175, 329
   engineering, 182–83
   good enough, 174–78
   initiatives, five major, 180–81
   low, causes of, 180
   responsibility for, 183
Software quality assurance (SQA), 132, 168–74
   as auditing function, 186
   coordinator role, 182
   defined, 169
   function, 159, 168–69
   goals for, 160, 313ff.
   guidelines for, 153
   plans, 182–83
   responsibility for, 180, 181–83
Software reliability, 263, 269, 271
Standards, software development, 7, 11, 146ff., 150–51, 306–7
   (see also IEEE standards for software development)
   Department of Defense, U.S., 148, 158
   ISO, 147, 156
   ISO 9000-3 guidelines, 156, 158
   ISO 9001 registration, 111, 149, 156, 158, 160
   sources of, 155, 156–58
State-transition diagram (STD), 86, 93, 234, 239
Team, 125
   behavior, 8–9, 10
   guidelines, involvement with, 150
   interactions, 319, 329–30
   leadership, 13–14
   reviews by, 192
   structured open, 13–14
Technologies, new, xix, 8, 9, 41, 244, 249, 250
Test cases, 71, 80, 83, 84, 96, 214, 219, 220–22
    benefits of, 82–83
    review of, 232
    software quality engineer, written by, 183
    SQA coordinator, written by, 182
Test coverage, 216
    branch, 215, 217
    condition, 215, 217, 225
    path, 216, 218
    statement, 215ff., 225
Test plans, 182, 214
    process, guidelines for, 153
Testing, 106, 123, 125, 133
    (see also Test cases; Test coverage; Test plans)
alpha, 263, 268
automated, 220–22
baseline method, 218ff.
beta, 123–24, 263, 268
black box, 215, 216, 217, 224
cyclomatic complexity and, 218–20, 224, 226
defined, 211
documentation, written, 211–13, 223
efficiency of, 175–76, 177
Graphics Engine example, 220–22
gray box, 217, 224
guidelines, 153, 222–24
vs. inspections, 193
integration, 186, 191, 213, 225
management of, 220–22
measured aspects of, 289–90
need for, 181
quality, impact on, 180
vs. quality assurance, 170
quality culture and, 212–15
regression, 217, 225
responsibilities for, 181–83
schedule delays and, 186
software quality engineer and, 183
structured, 180, 211–28, 319, 320, 322
system, 10, 180, 191, 213, 216
target quality level, 264
testers and, 186, 224
tools, 216, 217, 335
unit, 212–13, 215–18, 222, 224
white box, 215, 220, 223, 224
Tools, 229ff.
    CASE, 233–36
    problem tracking, 256–61
    testing, 216, 217, 335
Training, 45–58, 119, 133, 332
    (see also Conferences; Education)
as investment, 49, 51, 55
    CASE, 245, 249–51
culture and, 46
vs. education, 50
for inspections, 205
for metrics, 276, 282, 283
professional certification, 46, 53–54, 55
professional societies, 45, 46, 53–54, 55
seminars, 9, 48, 49–50, 55–56, 126, 332
for testing, 211–12
value of, assessing, 317
U.S. Government, as standards body, 156
Use cases, 64, 79, 83, 90, 95, 99
definition, 79
requirements, 79, 81–82, 83
test cases from, 80, 82–83
workshops, 79–81, 90
Usenet newsgroups, 53, 235, 256
User interface, 64, 79, 83, 91
graphical, 91, 154
guidelines for, 154
modeling, 91–93
prototyping, 93, 125, 256
User manual, guidelines for, 154
User needs, 64, 66, 231, 329
    (see also Customer, needs)
UX-Metric, 220
Videotape instruction, 53, 55
Vision of the product, 59, 63, 73–74, 78ff., 82, 83, 91, 98
Walkthrough, 98, 189, 194, 197–98
Work environment, health and, 311, 323, 324–25
    (see also Ergonomics)
World Wide Web, 50, 53, 235