
Table of Contents

Introduction	1.1
Beginner	1.2
Personal Skills	1.2.1
Learn to Debug	1.2.1.1
How to Debug by Splitting the Problem Space	1.2.1.2
How to Remove an Error	1.2.1.3
How to Debug Using a Log	1.2.1.4
How to Understand Performance Problems	1.2.1.5
How to Fix Performance Problems	1.2.1.6
How to Optimize Loops	1.2.1.7
How to Deal with I/O Expense	1.2.1.8
How to Manage Memory	1.2.1.9
How to Deal with Intermittent Bugs	1.2.1.10
How to Learn Design Skills	1.2.1.11
How to Conduct Experiments	1.2.1.12
Team Skills	1.2.2
Why Estimation is Important	1.2.2.1
How to Estimate Programming Time	1.2.2.2
How to Find Out Information	1.2.2.3
How to Utilize People as Information Sources	1.2.2.4
How to Document Wisely	1.2.2.5
How to Work with Poor Code	1.2.2.6
How to Use Source Code Control	1.2.2.7
How to Unit Test	1.2.2.8
Take Breaks when Stumped	1.2.2.9
How to Recognize When to Go Home	1.2.2.10
How to Deal with Difficult People	1.2.2.11
Intermediate	1.3
Personal Skills	1.3.1
How to Stay Motivated	1.3.1.1
How to be Widely Trusted	1.3.1.2
How to Tradeoff Time vs. Space	1.3.1.3
How to Stress Test	1.3.1.4
How to Balance Brevity and Abstraction	1.3.1.5
How to Learn New Skills	1.3.1.6
Learn to Type	1.3.1.7
How to Do Integration Testing	1.3.1.8
Communication Languages	1.3.1.9
Heavy Tools	1.3.1.10

How to analyze data	1.3.1.11
Team Skills	1.3.2
How to Manage Development Time	1.3.2.1
How to Manage Third-Party Software Risks	1.3.2.2
How to Manage Consultants	1.3.2.3
How to Communicate the Right Amount	1.3.2.4
How to Disagree Honestly and Get Away with It	1.3.2.5
Judgment	1.3.3
How to Tradeoff Quality Against Development Time	1.3.3.1
How to Manage Software System Dependence	1.3.3.2
How to Decide if Software is Too Immature	1.3.3.3
How to Make a Buy vs. Build Decision	1.3.3.4
How to Grow Professionally	1.3.3.5
How to Evaluate Interviewees	1.3.3.6
How to Know When to Apply Fancy Computer Science	1.3.3.7
How to Talk to Non-Engineers	1.3.3.8
Advanced	1.4
Technological Judgment	1.4.1
How to Tell the Hard From the Impossible	1.4.1.1
How to Utilize Embedded Languages	1.4.1.2
Choosing Languages	1.4.1.3
Compromising Wisely	1.4.2
How to Fight Schedule Pressure	1.4.2.1
How to Understand the User	1.4.2.2
How to Get a Promotion	1.4.2.3
Serving Your Team	1.4.3
How to Develop Talent	1.4.3.1
How to Choose What to Work On	1.4.3.2
How to Get the Most From Your Team-mates	1.4.3.3
How to Divide Problems Up	1.4.3.4
How to Handle Boring Tasks	1.4.3.5
How to Gather Support for a Project	1.4.3.6
How to Grow a System	1.4.3.7
How to Communicate Well	1.4.3.8
How to Tell People Things They Don't Want to Hear	1.4.3.9
How to Deal with Managerial Myths	1.4.3.10
How to Deal with Organizational Chaos	1.4.3.11
Appendix A * Bibliography/Websiteography	1.5
Appendix B * History (As of January 2016)	1.6
Appendix C * Contributions (As of January 2016)	1.7

How to be a Programmer: Community Version

Robert L. Read with Community

Copyright 2002, 2003, 2016 Robert L. Read

Licensed under [Creative Commons Attribution-ShareAlike 4.0 International License](#).

Introduction

To be a good programmer is difficult and noble. The hardest part of making real a collective vision of a software project is dealing with one's coworkers and customers. Writing computer programs is important and takes great intelligence and skill. But it is really child's play compared to everything else that a good programmer must do to make a software system that succeeds for both the customer and myriad colleagues for whom she is partially responsible. In this essay I attempt to summarize as concisely as possible those things that I wish someone had explained to me when I was twenty-one.

This is very subjective and, therefore, this essay is doomed to be personal and somewhat opinionated. I confine myself to problems that a programmer is very likely to have to face in her work. Many of these problems and their solutions are so general to the human condition that I will probably seem preachy. I hope in spite of this that this essay will be useful.

Computer programming is taught in courses. The excellent books: *The Pragmatic Programmer* [Prag99], *Code Complete* [CodeC93], *Rapid Development* [RDev96], and *Extreme Programming Explained* [XP99] all teach computer programming and the larger issues of being a good programmer. The essays of Paul Graham [PGSite] and Eric Raymond [Hacker] should certainly be read before or along with this article. This essay differs from those excellent works by emphasizing social problems and comprehensively summarizing the entire set of necessary skills as I see them.

In this essay the term boss is used to refer to whomever gives you projects to do. I use the words business, company, and tribe, synonymously except that business connotes moneymaking, company connotes the modern workplace and tribe is generally the people you share loyalty with.

Welcome to the tribe.

Contents

1. [Beginner](#)
 - o Personal Skills
 - [Learn to Debug](#)
 - [How to Debug by Splitting the Problem Space](#)
 - [How to Remove an Error](#)
 - [How to Debug Using a Log](#)
 - [How to Understand Performance Problems](#)
 - [How to Fix Performance Problems](#)
 - [How to Optimize Loops](#)
 - [How to Deal with I/O Expense](#)
 - [How to Manage Memory](#)
 - [How to Deal with Intermittent Bugs](#)
 - [How to Learn Design Skills](#)
 - [How to Conduct Experiments](#)
 - o Team Skills
 - [Why Estimation is Important](#)
 - [How to Estimate Programming Time](#)
 - [How to Find Out Information](#)
 - [How to Utilize People as Information Sources](#)

- How to Document Wisely
- How to Work with Poor Code
- How to Use Source Code Control
- How to Unit Test
- Take Breaks when Stumped
- How to Recognize When to Go Home
- How to Deal with Difficult People

2. Intermediate

- Personal Skills
 - How to Stay Motivated
 - How to be Widely Trusted
 - How to Tradeoff Time vs. Space
 - How to Stress Test
 - How to Balance Brevity and Abstraction
 - How to Learn New Skills
 - Learn to Type
 - How to Do Integration Testing
 - Communication Languages
 - Heavy Tools
 - How to analyze data
- Team Skills
 - How to Manage Development Time
 - How to Manage Third-Party Software Risks
 - How to Manage Consultants
 - How to Communicate the Right Amount
 - How to Disagree Honestly and Get Away with It
- Judgment
 - How to Tradeoff Quality Against Development Time
 - How to Manage Software System Dependence
 - How to Decide if Software is Too Immature
 - How to Make a Buy vs. Build Decision
 - How to Grow Professionally
 - How to Evaluate Interviewees
 - How to Know When to Apply Fancy Computer Science
 - How to Talk to Non-Engineers

3. Advanced

- Technological Judgment
 - How to Tell the Hard From the Impossible
 - How to Utilize Embedded Languages
 - Choosing Languages
- Compromising Wisely
 - How to Fight Schedule Pressure
 - How to Understand the User
 - How to Get a Promotion
- Serving Your Team
 - How to Develop Talent
 - How to Choose What to Work On
 - How to Get the Most From Your Team-mates
 - How to Divide Problems Up
 - How to Handle Boring Tasks
 - How to Gather Support for a Project
 - How to Grow a System
 - How to Communicate Well

[Click here to download full PDF material](#)