CHAPTER SEVEN

Orphans Preferred

Wanted: Young, skinny, wirey fellows not over 18. Must be expert riders willing to risk death daily. Orphans preferred. Wages $25 per week.

PONY EXPRESS ADVERTISEMENT, 1860

We realize the skills, intellect and personality we seek are rare, and our compensation plan reflects that. In return, we expect TOTAL AND ABSOLUTE COMMITMENT *to project success—overcoming all obstacles to create applications on time and within budget.*

SOFTWARE DEVELOPER ADVERTISEMENT, 1995[1]

The stereotypical programmer is a shy young man who works in a darkened room, intensely concentrating on magical incantations that make the computer do his bidding. He can concentrate for 12 to 16 hours at a time, often working through the night to make his artistic vision a reality. He subsists on pizza and Twinkies. When interrupted, the programming creature responds violently, hurling strings of cryptic acronyms at his interrupter—"TCP/IP, RPC, RCS, ACM, and IEEE!" he yells. The programmer breaks his intense concentration only to attend Star Trek conventions and watch Monty Python reruns. He is sometimes regarded as an indispensable genius, sometimes as an eccentric artist. Vital information is stored in his head and his head alone. He is secure in his job, knowing that, valuable as he is, precious few people compete for his job.

USA Today reported that the techie nerd stereotype is so well-entrenched that students in every grade ranked computer jobs near the

bottom of their lists of career choices.[2] *The Wall Street Journal* reported that film crews have difficulty presenting stories about leading-edge software companies in an interesting way because every story starts with "an office park, a cubicle, and a guy sitting there with a box on his desk."[3] Sometimes the stereotype is fostered even inside the profession. The associate director of Stanford University's computer science program was quoted by *The New York Times* as saying that software jobs are "mind-numbingly boring."[4] This is all in spite of the fact that sources such as the *Jobs Rated Almanac* consistently rate software jobs at the top or near the top of the most desirable occupations.[5]

How much of the stereotype is true, and what effect does it have on the programming occupation? Let's look first at the programmer's personality, then at the other elements of the stereotype.

The Meyers-Briggs Type Indicator

A common means of categorizing personality was developed by Katherine Briggs and Isabel Briggs Meyers and is called the Meyers-Briggs Type Indicator, or MBTI. The MBTI categorizes personality types in four ways.

Extroversion (E) or Introversion (I). Extroverts are oriented toward the outside world of people and things. Introverts are more interested in the inner world of ideas.

Sensing (S) or Intuition (N). This category refers to how a person prefers to receive decision-making data. The sensing person focuses on known facts, concrete data, and experience. The intuitive person looks for possibilities and focuses on concepts and theories.

Thinking (T) or Feeling (F). This mode refers to a person's decision-making style. The thinker makes decisions based on objective analysis and logic; the feeler relies on subjective feelings and emotions.

Perceiving (P) or Judging (J). The perceiving person prefers flexibility and open-ended possibility, whereas the judging person prefers order and control.

To determine MBTI type, a person takes a test that assigns one letter from each of the four categories, resulting in a designation such as ISTJ or ENTJ. These letters indicate an individual's personality tendencies or preferences. They don't necessarily indicate how a person will react in specific circumstances. A person might naturally prefer to be an I, but might have developed her E to be more comfortable in a business setting, for example. Such a person might test as an I even though most business associates would classify her as an extrovert.

MBTI Results for Software Developers

Two large studies have found the most common personality type for software developers is ISTJ,[6] a personality type that tends to be serious and quiet, practical, orderly, logical, and successful through concentration and thoroughness. ISTJs comprise from 25 to 40 percent of software developers.[7]

Consistent with the stereotype, programmers are indeed introverts. One-half to two-thirds of the software development population is introverted compared to about one-quarter of the general population.[8] Part of the reason for the majority of software developers being *I*s might be that more *I*s pursue higher education, and programmers are more educated than average. About 60 percent of software developers have attained at least a bachelor's degree, compared to about 30 percent of the general population.[9]

The S/N (sensing/intuition) and T/F (thinking/feeling) attributes are particularly interesting because they describe an individual's decision-making style. Eighty to 90 percent of software developers are *T*s compared to about 50 percent of the general population.[10] Compared to the average, *T*s are more logical, analytical, scientific, dispassionate, cold, impersonal, concerned with matters of truth, and unconcerned with people's feelings.

Programmers are approximately evenly split between *S*s and *N*s, and the difference between the two will be immediately recognizable to most

software developers. *S*s are methodical, live in the world of what can be accomplished now, are precise, concrete, and practical, like to specialize, and like to develop a single idea in depth rather than several ideas at once. *N*s are inventive, live in the world of possibility and theories, like to generalize, and like to explore many alternative ideas. An example of an *S* is an expert programmer who is intimately acquainted with every detail of a specific programming language or technology. An example of an *N* is a designer who considers wide-ranging possibilities and shrugs off low-level technical issues as "implementation details." *S*s sometimes aggravate *N*s because they go deep into technical details before *N*s feel the breadth has been adequately explored. *N*s sometimes aggravate *S*s because they jump from one design idea to the next before *S*s feel they have explored any particular technical area in sufficient depth.

Personality Characteristics of Great Designers

The MBTI gives some insight into typical programmer personalities, but it isn't the final word. One important group of software development skills is software design skills. Many programmers aspire to be great designers. What are the characteristics of great designers? One study[11] of great designers in general (not just software developers) found that the most creative human problem solvers seem to move easily between the S/N, T/F, and P/J ends of the continuum. They move back and forth between holistic and sequential, intuition and logic, theory and specific details. They are able to look at problems from many different points of view. Leonardo da Vinci and Albert Einstein are examples of such great designers (although I don't believe they ever took the MBTI).

Great designers have a large set of standard patterns that they apply to each new problem. If the problem fits an existing pattern, the great designer can easily solve it using a familiar technique.

Great designers have mastery of the tools they use.

Great designers aren't afraid of complexity, and some of the best are drawn to it. But their goal is to make the seemingly complex simple. As Einstein said, everything should be made as simple as possible, but no

simpler. The French writer and aircraft designer Antoine de Saint-Exupéry made much the same point when he said, "You know you have achieved perfection in design not when you have nothing more to add, but when you have nothing more to take away."

Great designers seek out criticism of their work. The feedback loop that criticism supports allows them to try and discard many possible solutions.

Great designers usually have experience on failed projects and have made a point of learning from their failures. They try out and discard more alternatives. They are often wrong, but they discover and correct their mistakes quickly. They have the tenacity to continue trying alternatives even after others have given up.

They are not afraid of using brute force to solve a problem. Thomas Edison worked on the problem of designing a filament for an electric light bulb for nearly two years, and he tried thousands of materials. An assistant once asked him how he could keep trying after failing so many times. Edison didn't understand the question. In his mind, he hadn't failed at all. He is supposed to have replied, "What failure? I know thousands of things that do not work."

Great designers must be creative to generate numerous candidate design solutions. A great deal of research has been done on creativity, and there are some common themes. Creative people are curious, and their curiosity covers a wide range of interests. They have high energy. They are self-confident and independent enough to explore ideas that other people think are foolish. They value their own judgment. They are intellectually honest, which helps them differentiate what they really think from what the conventional wisdom says they should think.

Great designers have a restless desire to *create*—to make things. That desire might be to create a building, an electronic circuit, or a computer program. They have a bias toward action. Great designers aren't satisfied merely to learn facts; they feel compelled to *apply* what they have learned to real-world situations. To the great designer, not applying knowledge is tantamount to not having obtained the knowledge in the first place.

Programmers live for the "aha" insights that produce breakthrough design solutions. I think this is one of the reasons software developers' affinity for Monty Python makes more sense than it might at first

appear. Monty Python flouts social conventions using unorthodox juxtapositions of elements of time and culture. The same independent-minded, out-of-the-box thinking that gives rise to Monty Python's scripts can also give rise to the innovative technical design solutions that programmers strive for.

Some of these characteristics match the programmer stereotype, and some do not. People outside software development might think of computer programming as dry and uncreative. People inside software development know that some of the most exciting projects of our times could not be accomplished without extreme software creativity. Movie animation, space exploration, computer games, medical technology—it's hard to find a leading-edge area that doesn't depend on software developers' creativity.

Software developers know that, stereotypes aside, computer programming gives them a medium in which they can create something out of nothing. That provides them with the same satisfaction that others obtain from sculpting, painting, writing, or other activities that are more obviously creative. "Mind-numbingly boring?" I think not.

Total and Absolute Commitment

The stereotype of the programmer working 12 to 16 hours at a time contains more than a grain of truth. To be an effective developer, you need to be able to concentrate exclusively on the programming task. That concentration exacts a penalty. While concentrating on a programming project, you lose track of time. One morning you look up, and it's 2:00 p.m. You've missed lunch. One Friday evening you look up, and it's 11:00 p.m. You've stood up your date or neglected to tell your spouse you were coming home late. One October you look up and realize that the summer is over and you missed it again because you've spent the past three months concentrating on an interesting project.

The Pony Express ad at the beginning of the chapter could be applied to some of today's software developers. It can be hard to have a family,

friends, or other social ties when you work as much as some software developers do. Here is Pascal Zachary's description of programmer commitment on the Windows NT project:

> Work pervades their existence. Friends fade into the background. The ties of marriage fray or rip apart. Children are neglected or deferred. Hobbies wither. Computer code comes to mean everything. If private dreams are nursed at all, it is only to ease the pain of creating NT.[12]

At the end of the NT project, some developers left the company. Some were so burned out that they left the software field entirely.

Recognizing this phenomenon, some experienced developers have difficulty signing up for new projects because they know that in so doing they will once again choose to lose evenings, weekends, and summers.

This pattern is avoidable by moving to an engineering approach to software development. The average project spends 40 to 80 percent of its time correcting defects.[13] With software engineering, the team doesn't create the defects in the first place or it positions itself to eliminate the defects more quickly and easily. Eliminating 50 percent of the work is one quick way to reduce the workweek from 80 hours to 40.

Developers' high task completion need creates an unusual relationship between their commitment to their projects and their commitments to their companies. In my experience, no matter how much software developers dislike their companies, they rarely quit in mid-project. Workers in other fields would say, "I hate my company. I'm going to wait until right in the middle of the project, then quit. That'll show them!" Software developers say, "I hate my company. I'm going to finish this project to show the company what they're losing, then I'll quit. That'll show them!"

Programmers also seem committed to their occupation. One difficulty companies have in enforcing nondisclosure agreements is that many programmers feel more loyal to their colleagues at other companies than they do to their own employers. I have observed that software developers routinely discuss company confidential material with colleagues who are

not covered by nondisclosure agreements. In their judgment, the free exchange of information between companies is more important than any one specific company's protection of its trade secrets. Programmers in the Open Source movement apply the idea more broadly and advocate that all source code and related materials should be disclosed for the public good.[14]

I think this loyalty to a project, tendency to work long hours, and high need for creativity are all related. Once a programmer has visualized the software to be built, bringing the vision to life becomes paramount, and the programmer feels tremendously unsettled until that can be done.

This ability to make a strong commitment to a vision bodes well for the establishment of a profession of software engineering. Programmers have a desire to commit to something beyond themselves—to their colleagues on a project or to other colleagues industry-wide. A profession of software engineering and its related professional societies can provide a constructive focus for this occupational commitment.

Software Demographics

The stereotype of programmers as young men appears to have some merit too. The average software worker is significantly younger than the United States labor force. As Figure 7-1 shows, the age structure of the software workforce peaks at about 30 to 35 years old, which is about 10 years younger than the peak for other types of technical workers.

The majority of software developers are male. In the latest year for which data is available (2000), 72 percent of computer and information science bachelor degrees and 83 percent of the Ph.D.s were awarded to men.[15] In high school, only 17 percent of people taking the advanced placement test for computer science are female, which is the lowest of any subject.[16]

Programmers are younger than average and tend to be male. The comparison to Pony Express riders begins to look less and less like an exaggeration (though there's no evidence that computer programmers are any more "wiry" than average).

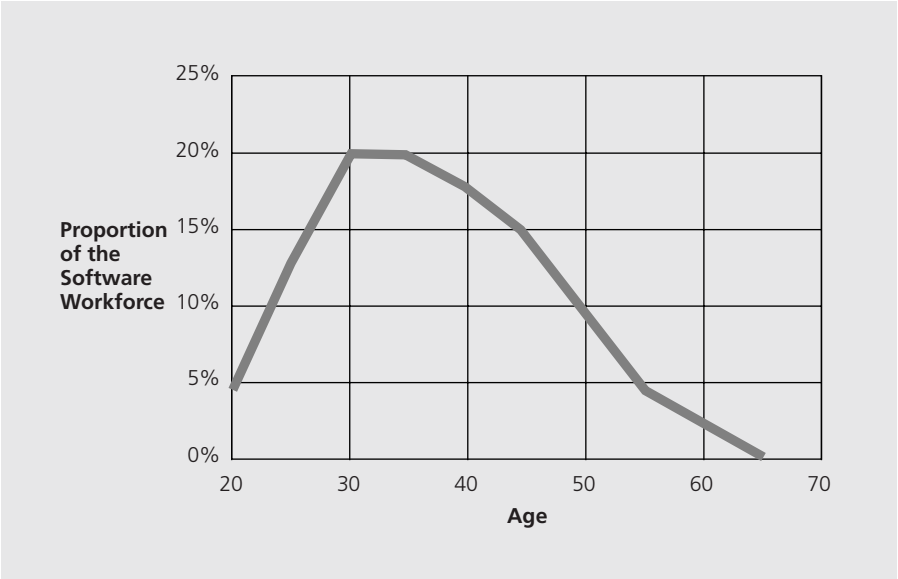

FIGURE 7-1

The biggest group of software workers is 30 to 35, which is about 10 years younger than the biggest groups of workers in other technical occupations.

Source: "A Moving Target: Studies Try to Define the IT Workforce"[17]

Education

Most programmers go through a gradual occupational awakening. When I wrote my first small programs, I thought, "Once I get the program to compile and quit getting all these syntax errors, I'll have computer programming figured out." After I stopped having problems with syntax errors, sometimes my programs still didn't work, and the problems that were left seemed even harder to figure out than the syntax errors were. I adopted a new belief, "Once I get the program debugged, I'll have computer programming figured out." That worked until I started creating larger programs and began to have problems because the various pieces I created didn't work together the way I thought they would. I came to rest on a new belief, "Once I figure out how to design effectively, I'll finally have software development figured out." I created some beautiful designs, but

then some of my designs had to change because the requirements kept changing. At that point, I thought, "Once I figure out how to get good requirements, I'll finally have software development figured out." Somewhere along the path to learning how to get good requirements I began to realize that I might *never* get software development figured out. That realization was my first real step toward software engineering enlightenment.

Programmers take many circuitous paths to personal enlightenment, some resembling mine, and some different. Most developers are well-educated in general but self-taught about software development. As Table 7-1 shows, about 60 percent of software developers have obtained bachelor's degrees or higher. According to the United Engineering Foundation, about 40 percent of all software workers obtained their degrees in software-related disciplines.[18] About half of those who eventually obtained a software-related degree did so after first obtaining a bachelor's degree in some other subject. Another 20 percent of all software workers obtained degrees in subjects such as mathematics, engineering, English, history, or philosophy. The remaining 40 percent completed high school or some college but did not obtain a four-year degree.

Universities in the United States currently award about 35,000 computer science and related degrees per year[19] while about 50,000 new software development jobs are created each year.

The implication of all these statistics is that a great many software developers have not received any systematic training in computer science,

TABLE 7-1

Software developer education[20]

HIGHEST LEVEL OF EDUCATION ATTAINED	PERCENT OF SOFTWARE DEVELOPERS
High school graduate or equivalent or less	11.8
Some college, no degree	17.2
Associate's degree	11.0
Bachelor's degree	47.4
Graduate degree	12.8

much less in software engineering. What education they have obtained has been acquired through on-the-job training or self-study. Providing more consistent education in software engineering represents a significant opportunity to improve the level of software development practices.

Job Prospects

The total current employment for software workers in the United States is about two million. As Table 7-2 shows, jobs are divided among computer scientists, computer programmers, systems analysts, network analysts, and software engineers. (Some of these government-statistic job titles might sound old fashioned, but they do include modern software jobs.)

Job prospects for software developers in the United States are very good. According to the Bureau of Labor Statistics, computer and data processing services will be the fastest growing industry from 2000 to 2010, with a projected increase of 86 percent during this period. Software engineering is expected to be the fastest growing job category overall. All computer-related job categories are expected to increase.[21]

TABLE 7-2

Job breakdown for software workers[22]

JOB TITLE	CURRENT NUMBER OF SOFTWARE PERSONNEL IN THE U.S.
Computer and information scientists, research	28,000
Computer programmers	585,000
Computer software engineers, applications	380,000
Computer software engineers, systems software	317,000
Computer systems analysts	431,000
Network systems and data communications analysts	119,000
Other computer specialists	203,000
Total	**2,063,000**

TABLE 7-3
Software development jobs worldwide[23]

YEAR	TOTAL PROGRAMMERS
1950	100
1960	10,000
1970	100,000
1980	2,000,000
1990	7,000,000
2000	10,000,000
2010	14,000,000
2020	21,000,000

Worldwide, software development jobs are expected to increase as dramatically as they are increasing in the United States. Table 7-3 shows the projected increase.

With a 15,000-job-per-year gap between baccalaureate degrees awarded and jobs created, demand for computer programmers should remain high in the United States for at least the next several years, despite cyclical ups and downs in the job market. Labor shortages have been a perennial feature of the software world at least since the mid-1960s.[24] Software-related jobs are rated well in terms of salary, benefits, work environment, job stress, job security, and other factors.[25] Programmers know that, desirable as their jobs are, there isn't much competition for them.

Programming Heroes and Ball Hogs

Combine a shortage of skilled workers with the common tendency to set overly optimistic schedules, and the stage is set for the programming hero. Programming heroes take on challenging assignments and write mountains of code. They work vast amounts of overtime. They become indispensable to their projects. Success, it seems, rests squarely on their shoulders.

Project managers both love and fear hero programmers because they are smart, temperamental, and sometimes a little self-righteous, and

because the managers don't see any way to complete the project without them.[26] In a tight labor market, replacing them isn't an option.

Unfortunately, the reality is that for every programming hero who is capable of monumental coding achievements, there are other pathological programming disasters who just don't know how to work well with others. They hoard design information and source code. They refuse to participate in technical reviews. They refuse to follow standards established by the team. The sum total of their actions is to prevent other team members from making potentially valuable contributions. A significant number of programming heroes don't turn out to be heroes at all; they turn out to be prima donna programming ball hogs.

Individual heroics can contribute to project success, but teamwork generally contributes more than individual accomplishment does. A study at IBM found that the average programmer spends only about 30 percent of the time working alone.[27] The rest is spent working with teammates, with customers, and on other interactive activities. Another study of 31 software projects found that the greatest single contributor to overall productivity was team cohesiveness.[28] Individual capabilities also significantly influenced productivity but were less influential than team cohesiveness.

Many people like to take on challenging projects that stretch their capabilities. Those who can test their limits, follow sound software development practices, and still cooperate with their teammates are the true programming heroes.

Cult of Personality

Upon examination, many aspects of the programmer personality stereotypes turn out to be accurate. The worker shortage contributes to increased hours for all workers who can be found—heroes and others—which means less time for those workers' self-education and professional development. This gives rise to a chicken-and-egg problem: We can't implement better development practices until we find the time for education and training, and we can't find the time for education and training until we implement better practices.

Working in favor of greater software professionalism is the fact that programmers are getting older. The longer the software field exists, the more the average age of working programmers will begin to match the age of the rest of the working population. The extreme personal sacrifices that are tolerable to young workers in their 20s become harder to justify as we become married, have children, become homeowners, and move into our 30s, 40s, 50s, and 60s. As the current cohort of programmers ages, the present hero-based approach to software development may naturally give way to approaches that rely more on working smart than on working hard. More software workers will adopt practices that allow them to complete their projects as promised and still be home in time for dinner.

Notes

1. *The Seattle Times,* October 8, 1995. Emphasis in original.
2. *USA Today,* February 16, 1998, pp. 1B–2B.
3. Bronson, Po, "Manager's Journal," *The Wall Street Journal,* February 9, 1998.
4. "Software Jobs Go Begging," *The New York Times,* January 13, 1998, p. A1.
5. Krantz, Les, *Jobs Rated Almanac,* NY: St. Martin's Press, 1999.
6. Thomsett, Rob, "Effective Project Teams: A Dilemma, a Model, a Solution," *American Programmer,* July–August 1990, pp. 25–35. Lyons, Michael L., "The DP Psyche," *Datamation,* August 15, 1985, pp. 103–109.
7. Thomsett, Rob, "Effective Project Teams: A Dilemma, a Model, a Solution," *American Programmer,* July–August 1990, pp. 25–35. Lyons, Michael L., "The DP Psyche," *Datamation,* August 15, 1985, pp. 103–109. Bostrom, R. P., and K. M. Kaiser, "Personality Differences within Systems Project Teams," *Proceedings of the 18th Annual Computer Personnel Research Conference,* ACM No. 443810, 1981.
8. Thomsett, Rob, "Effective Project Teams: A Dilemma, a Model, a Solution," *American Programmer,* July–August 1990, pp. 25–35. Lyons, Michael L., "The DP Psyche," *Datamation,* August 15, 1985, pp. 103–109.
9. National Center for Education Statistics, *2001 Digest of Educational Statistics,* Document Number NCES 2002130, April 2002.
10. Thomsett, Rob, "Effective Project Teams: A Dilemma, a Model, a Solution," *American Programmer,* July–August 1990, pp. 25–35. Lyons, Michael L., "The DP Psyche,"

Datamation, August 15, 1985, pp. 103–109. Bostrom, R. P., and K. M. Kaiser, "Personality Differences within Systems Project Teams," *Proceedings of the 18th Annual Computer Personnel Research Conference,* ACM No. 443810, 1981.

11. Glass, Robert L., *Software Creativity,* Englewood Cliffs, NJ: Prentice Hall PTR, 1994.
12. Zachary, Pascal, *Showstopper!: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft,* New York: Free Press, 1994.
13. Software Engineering Institute, quoted in Fishman, Charles, "They Write the Right Stuff," *Fast Company,* December 1996. Mills, Harlan D., *Software Productivity,* Boston, MA: Little, Brown, 1983, pp. 71–81. Wheeler, David, Bill Brykczynski, and Reginald Meeson, *Software Inspection: An Industry Best Practice,* Los Alamitos, CA: IEEE Computer Society Press, 1996. Jones, Capers, *Programming Productivity,* New York: McGraw-Hill, 1986. Boehm, Barry W., "Improving Software Productivity," *IEEE Computer,* September 1987, pp. 43–57.
14. See, for example, R.M. Stallman, "The GNU Manifesto," 1985, www.fsf.org/gnu/manifesto.html. Raymond, E.S., "Homesteading the Noosphere," 1998, www.catb.org /~esr/writings/homesteading.
15. National Center for Education Statistics, *2001 Digest of Educational Statistics,* Document Number NCES 2002130, April 2002.
16. "Software Jobs Go Begging," *The New York Times,* January 13, 1998, p. A1.
17. Lowell, Bill, and Angela Burgess, "A Moving Target: Studies Try to Define the IT Workforce," *IT Professional,* May/June 1999.
18. Lowell, Bill, and Angela Burgess, "A Moving Target: Studies Try to Define the IT Workforce," *IT Professional,* May/June 1999.
19. National Center for Education Statistics, *2001 Digest of Educational Statistics,* Document Number NCES 2002130, April 2002.
20. *Occupational Outlook Handbook 2002-03 Edition,* Bureau of Labor Statistics, 2002.
21. *Occupational Outlook Handbook 2002-03 Edition,* Bureau of Labor Statistics, 2002.
22. Hecker, Daniel E., "Occupational employment projections to 2010," *Monthly Labor Review,* November 2001, vol. 124, no. 11.
23. Jones, Capers, *Applied Software Measurement: Assuring Productivity and Quality,* 2d Ed., New York: McGraw-Hill, 1997.
24. Bylinsky, Gene, "Help Wanted: 50,000 Programmers," *Fortune,* March 1967, pp. 141ff.
25. Krantz, Les, *Jobs Rated Almanac,* NY: St. Martin's Press, 1999.

26. Bach, James, "Enough about Process: What We Need Are Heroes," *IEEE Software,* March 1995, pp. 96–98.

27. McCue, Gerald M., "IBM's Santa Teresa Laboratory—Architectural Design for Program Development," *IBM Systems Journal,* vol. 17, no. 1, 1978, pp. 4–25.

28. Lakhanpal, B., "Understanding the Factors Influencing the Performance of Software Development Groups: An Exploratory Group-Level Analysis," *Information and Software Technology,* 35 (8), 1993, pp. 468–473.