Orphans Preferred

https://www.gamasutra.com/view/feature/131820/orphans_preferred.php

"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

The stereotypical programmer is a shy young man who works in a darkened room, intensely concentrating on magical incantations that coax the computer to do his bidding. He can concentrate 12-16 hours at a time, often working through the night to realize his artistic vision. 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, SCSI, ISA, ACM, and IEEE!" 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 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. *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." 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."

How much of the stereotype is true, and what effect does it have on the programming occupation? To find out, 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 category 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.

After a person takes the MBTI test, that person is assigned 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. For example, some people might have a natural preference for I (introversion) but have developed their E (extroversion) so that they can be more effective in a business setting. Test results might indicate such people are introverts even though most business associates would classify them as extroverts.

MBTI Results for Software Developers

Two large studies have found that the most common personality type for software developers is ISTJ (introversion, sensing, thinking, judging), a type that tends to be serious and quiet, practical, orderly, logical, and successful through concentration and thoroughness. ISTJs comprise 25-40 percent of software developers.

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. One reason the majority of software developers are *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 25 percent of the general population.

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 ninety percent of software developers are *T*s, compared to about 50 percent of the general population. 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 Ns because *S*s go deep into technical details before Ns feel the breadth has been adequately explored. *N*s sometimes aggravate *S*s because *N*s 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

MBTI provides some insight into typical programmer personalities, but it isn't the final word. Many programmers aspire to be great designers. What are the personality characteristics of great designers? One study of designers in general (not just software developers) found that the most creative problem solvers seem to move easily between the *S/N*, *T/F*, and *P/J* distinctions. These individuals move back and forth between the holistic and sequential, the intuitive and logical, and the theoretical and specific; and

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 out and discard many possible solutions.

Great designers usually have experience on failed projects and have made a point of learning from their failures. They experiment with alternatives. Their creativity often leads them to dead ends, but they discover and correct their mistakes quickly. They have the tenacity to continue trying options even after other designers have given up.

Great designers 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. 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 on creativity has revealed 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 reason that software developers' affinity for Monty Python makes more sense than it might at first appear. Monty Python flouts social conventions using extremely unorthodox juxtapositions of elements of time and culture. The same independent, 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.

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 the contributions of highly creative individuals. Movie animation, the space program, computer games, medical technology-it's hard to find a leading-edge area that doesn't depend on the software developer's creativity. Software developers know that computer programming gives them a medium in which they can create something out of nothing, an experience that provides them with the same satisfaction that some individuals obtain from sculpting, painting, or writing. "Mind numbingly boring?" I don't think so.

Total and Absolute Commitment

The stereotype of the programmer working 12-16 hours at a time contains more than a grain of truth, however, and the Pony Express ad at the beginning of this chapter could almost describe some of today's software developers. To be an effective developer, you must be able to concentrate exclusively on the programming task. Such 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 missed lunch. One Friday evening you look up, and it's 11:00 P.M.-you 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 spent the past three months concentrating on an interesting project. Work can exclude family, friends, and other social ties. Here is Pascal Zachary's description of programmer commitment on the Microsoft 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.

At the end of the Windows NT project, some developers left the company. Others were so burned out that they left the software field entirely. Recognizing this phenomenon, some experienced developers are reluctant to sign up for new projects because they know that they might once again expose themselves to lost evenings, spent weekends, and missed summers.

Developers can avoid this work pattern by adopting an engineering approach to software development. The average project spends 40-80 percent of its time correcting defects. Project teams following a software engineering approach don't create the defects in the first place, or they position themselves to eliminate the defects more quickly and easily. Eliminating 50 percent of the work is one quick way to reduce the work week from 80 hours to 40.

The commitment software developers' have to their projects as compared to their commitment to their companies is unusual. In my experience, no matter how much software developers dislike their companies, they rarely quit mid-project. Workers in other fields might say, "I hate my company. I'm going to wait until right in the middle of the project, then quit. That'll show them!" But 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!"

Despite their lack of commitment to company, programmers do seem committed to their occupation. Many programmers feel more loyal to their colleagues at other companies than they do to their employers, and one consequence is that companies have difficulty enforcing nondisclosure agreements. I have observed that software developers routinely discuss confidential company material with colleagues who are not covered by nondisclosure agreements. In their judgment, the free exchange of information between developers is more important than any one specific company's need to protect its trade secrets. As an example of loyalty to colleagues over companies taken to the extreme, consider programmers in the Open Source movement, who advocate that all source code and related materials should be disclosed for the public good.

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.

Programmers are willing to commit to something beyond themselves-to their teammates, to their projects, or to their colleagues industry-wide. This willingness to make strong occupational commitments bodes well for establishing a profession of software engineering, which can provide a constructive focus for 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. The age structure of the workforce peaks at 30-35 years old, which is about 10 years younger than the peak for other types of technical workers. The average age is 38 years old, which is younger than the average age of the United States labor force overall.

The majority of software developers are male. In the latest year for which data is available (1996), 72 percent of the bachelor degrees in computer and information science and 85 percent of the PhDs were awarded to men. In high school, only 17 percent of students taking the advanced placement test for computer science are female, which is the lowest of any subject.

The comparison of programmers to Pony Express riders begins to look less and less like an exaggeration (though I don't have any evidence that computer programmers are any more "wirey" than average).

Education

Many 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 remaining problems seemed even harder to figure out than the syntax errors. I adopted a new belief, "Once I get the program debugged, I'll have computer programming figured out." That belief held true until I started creating larger programs and began having problems because the various pieces I implemented 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 some of them had to be changed 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, some not. As I mentioned earlier in the chapter, and as Table 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. 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 25,000 computer science and related degrees per year, whereas about 50,000 new software development jobs are created each year.

Highest Level of Education Attained	**Percent of Software Developers**
High school graduate or equivalent or less	**10**
Some college, no degree	**21**
Associate's degree	**10**
Bachelor's degree	**45**
Graduate degree	**14**

Table 1. Software Developer Education

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

Total current employment for software workers in the United States is about 2 million. As Table 2 shows, jobs are divided among computer programmers, systems analysts, computer scientists, computer operators, and network administrators. (These government-statistic job titles might sound old fashioned, but they do include modern software jobs.)

Job Title	Current Number of Software Personnel in the U.S.
Computer programmers	**617,000**
Systems analysts	**671,000**
Database administrators, computer support specialists, and other computer scientists	**289,000**
Computer operators/network administrators	**249,000**
Total	**1,826,000**

Table 2. Job Breakdown of Software Workers

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 job category between 1996 and 2006, with a projected increase of more than 100 percent during this period. The job category in second place, health services, has a projected increase of less than 70 percent. All computer-related job categories except computer operators are expected to increase.

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

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**

With a 25,000 job-per-year gap between bachelor's degrees awarded and jobs created, demand for computer programmers should remain high in the United States for at least the next several years. This labor shortage has been a perennial feature of the software world at least since the mid-1960s. Software-related jobs are rated well in terms of salary, benefits, work environment, job stress, job security, and other factors. Desirable as these jobs are, programmers know that there isn't much competition for them.

Programming Heroes and Ball Hogs

Combine a shortage of 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 these programmers are smart, temperamental, and sometimes a little self-righteous, and because the managers don't see any way to complete projects without them. In a tight labor market, replacing them isn't an option.

Unfortunately, the reality is that, for every programming hero 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. The rest is spent working with teammates, with customers, and on interactive activities. Another study of 31 software projects found that the greatest single contributor to overall productivity was team cohesiveness. 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 engineering 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 worst part of the stereotype-pathological heroism-might be due in part to industry demographics. The perennial labor shortage means that anyone with a strong enough interest in software development work can get a job as a computer programmer. The job market protects workers who become self-styled heroes.

The labor shortage contributes to increased hours for all available workers-heroes and others-which means less time for their self-education and professional development. This situation gives rise to a sort of "catch 22": 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 moving toward true software engineering professionalism is the fact that software developers are getting older. The longer the software field exists, the more the average age of software developers will begin to match the age of the rest of the working population. The extreme personal sacrifices that are tolerable to workers in their 20s become harder to justify as those workers marry, have children, buy homes, and move into their 30s, 40s, and 50s. As the current cohort of software workers grows older, the present hero-based approach to software development may naturally give way to an approach that relies more on working smart than on working hard. Software workers will become increasingly interested in the practices that allow them to complete their projects as promised and still be home in time for dinner.

Steve McConnell is president and chief software engineer at Construx Software, where he divides his time between leading custom software projects, teaching classes, and writing books and articles. He is the author of the Microsoft Press books *Code Complete* (1993), *Rapid Development* (1996), and *Software Project Survival Guide* (1998). His books have twice won Software Development magazine's Jolt Excellence Award for outstanding software development book of the year. In 1998, readers of Software Development named Steve one of the three most influential people in the software industry along with Bill Gates and Linus Torvalds. In his spare time, Steve serves as editor in chief of *IEEE Software* magazine. He is on the panel of experts that provides advice to the Software Engineering Body of Knowledge (SWEBOK) project, and is a member of IEEE and ACM.

Steve earned a bachelor's degree from Whitman College and a master's degree in software engineering from Seattle University. He lives in Bellevue, Washington with his wife, Tammy; daughter, Haley; and dog, Daisy. If you have any comments or questions about this book, please contact Steve via e-mail at stevemcc@construx.com or via his web site at http://www.construx.com/stevemcc/.