

Robert L. Glass

The Ups and Downs of Programmer Stress

It is well known that the 1990s trend toward downsizing/rightsizing has led to trauma among people in industry. Those axed must deal with the severe trauma of losing a job; those who are left behind must add the work of the axed to what they were already doing, leading to a different kind of trauma. Although it is better to be one of those who remains behind (usually), there is a traumatic price paid by all.

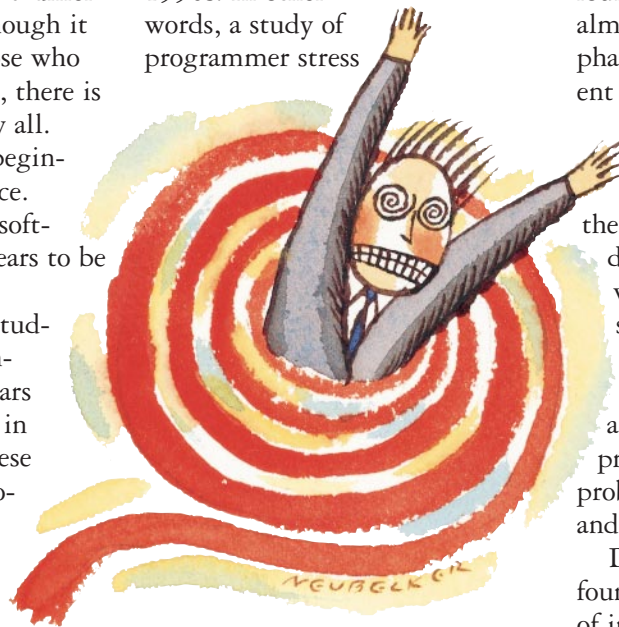
Some researchers are beginning to measure that price. And, particularly in the software field, the price appears to be high.

The most important studies of this topic were conducted in Japan a few years ago, and were published in 1993. The subject, in these Japanese studies, was programmer stress.

Researchers found that programmer stress is both extremely common and extremely problematic. We will talk about that stress and the problems it causes later. But first, a bit of background.

Japan's software industry is in some ways different from that of the rest of the world. For one, according to [1], software specialists are not held in high esteem in Japanese society. That manifests

itself in industry with programmers who are not encouraged to take much responsibility, and who are not paid well. Of course, those kinds of conditions would tend to exacerbate any stress problems that might be prevalent in the software industry of the 1990s. In other words, a study of programmer stress



in Japan may both mirror and magnify similar problems elsewhere in the world.

Now let's get to the specifics of those research findings. There were two studies: [Fujigaki, 1993] and [Furuyama, 1993]. The first study measured the degree of stress present in various programming tasks, and sought to identify

the cause. The second study measured the impact of stress on the number of errors introduced into the software product by the stressed programmers. Both studies produced what I think are fascinating results.

In the first study, Fujigaki found stress present during almost all software life-cycle phases. But the stress took different forms in different phases.

During requirements definition and design, the research found the stress in the form of high anxiety and depression. The programmers were struggling, Fujigaki speculated, to anticipate the needs of the users, which were often ambiguously presented and in constant flux, and to produce a design to solve the problem defined by those unclear and unstable requirements.

During coding, the research found the stress taking the form of irritability and falling morale. The programmers here were struggling to mold a solution out of the myriad details that tend to lie unidentified beneath the high level of most formal designs.

During testing, the research found that stress of all categories ran rampant. The programmers struggled to tie all the loose ends of the previous phases into a

working, solid and specific whole, and inherited all the stresses of the previous phases.

Stress was measured using standardized stress measurements tests, defined for use in other disciplines. Programming was studied because the researchers suspected its practice was a high source of stress. (For example, they noted that “Japanese managers deliberately underestimate software development time,” a practice that heaps stress on the backs of those subject to unrealistic expectations—and a practice not unknown in the rest of the programming world.) The findings of their studies confirmed their suspicions.

But why was there so much stress in the programming profession? Here, Fujigaki was quite clear. The management model currently used for programmers, the author said in no uncertain terms, was wrong. The tendency is currently to manage programmers with techniques derived from the manufacturing industry. The researcher called them “manufacturing-derived management techniques,” and also identified them as a “time-management system.” (I have referred to this tendency elsewhere as “management by schedule.”) But that approach simply won’t work, according to Fujigaki. “The software process is not the manufacturing process. The time-management system that developed in manufacturing should not be applied to the software process without modifications,” the researcher went on. And what does Fujigaki think those modifications

should be? “The software process is a learning and communication process,” Fujigaki concludes. Clearly the researcher’s expectation is that management’s role is to facilitate this learning and communication, although the study does not say anything about what form of style management might take.

So the conclusion of the first study is that programmers are highly stressed, and that poorly chosen management technique is the problem.

The second study measured the effects of this stress. Furuyama and co-authors examined the toll stress took by studying the results of the testing activity. Programmers operating under stress, the researchers found, make far more mistakes than they would under more normal circumstances.

The findings were quite specific. Examining collections of software faults, the researchers found that 37% of mistakes would have been avoided “by appropriate scheduling and placing no stress on the developers.” Another 34% of those faults were attributed by the researchers to “human nature” rather than to technical difficulties. Adding these percentages together, the researchers conclude “71% of all faults are the responsibility of the project manager, the quality assurance manager, and senior management” (in the sense these faults arise from human-factors considerations).

These numbers are shockingly high. But there is more to come in the study. Design, the researchers say, is particularly vulnerable to stress-caused errors.

They found that 42% of all design faults were directly attributable to programmer stress. Why? The study concluded “deep thinking, such as searching for solutions in a huge problem space . . . is required in the design phase Deep thinking is easily effected by stress, which causes imperfect investigation.”

It is easy, of course, to be suspicious of these findings. In particular, attributing programmer-caused errors to management seems like a shift of responsibility that in many ways could be counter-productive. That is, programmers who allow themselves to believe errors are management’s fault may simply lower their efforts, inevitably leading to even more errors. But at the same time, regardless of the specifics of the findings, the overall discovery these researchers have made seems important.

Downsizing and rightsizing, it is well known, lead to stress among those who survive the cuts. Add to that the stress resulting from an inappropriate management style, one simply not suited for knowledge workers like programmers. The result? High levels of errors in the resulting software products. That is the gist of these Japanese research findings. I may find it hard to believe percentage numbers like 37 and 42 and 71. But I have no trouble believing that, at its essence, this research is telling us something we need to hear.

Is there any evidence corroborating these findings? The answer is yes. U.S. researcher

Practical Programmer

Robert A. Zawacki (Colorado), has conducted a number of studies of programmer needs and characteristics. Beginning in 1979, he (and his colleague Dan Couger) found some interesting differences between programmers and the remainder of society—programmers had a high need to achieve, and they had a low need to socialize with other people.

Zawacki and Couger continued over the years to update this sociological research. In 1993, for example, Zawacki found that those two unique needs remained for the most part the same. But he also found something else—what he called an “alarming drop” in job satisfaction. Programmers were considerably less satisfied with their work in 1993 than they were in 1979.

What did Zawacki see as the cause? There were several, he found, but one was a need for “better management,” a management better prepared to deal with the changing needs of programmers in the 1990s. Zawacki suggested these changes in management style:

1. Find ways to improve motivation of programmers.
2. Improve feedback between managers and programmers.
3. Add more people to the individual contributor mix with higher social needs (to match the more team-oriented, user-focused approaches of the 1990s).

High stress? Increasing numbers of programming errors? Less job satisfaction? According to these researchers, those all seem to be true of the programming

profession in the 1990s. And they fit together into an intuitively believable whole.

What is the solution? All three of the researchers who identified these problems agree. It is time to change our ways of managing programmers. And they suggest some specifics of how that might be done. This is both an interesting research finding, extremely relevant to (and dependent on a study of) practice, and an interesting challenge to future researchers (to find management techniques to replace the current flawed approaches).

There is still much to learn in this fledgling software world, and it should not surprise us that we are still struggling to find better ways of doing—and managing—the job. What we see here is that researchers who employ a sociological approach to the problem may be helping us see some things more clearly than our more purely computing-oriented research approaches of the past.

REFERENCES

1. Cusumano, M. A. *Japan's Software Factories*. Oxford University Press, 1991.
2. Fujigaki, Y. Stress analysis: A new perspective on peopleware. *Amer. Prog.* 6, 7 (Jul. 1993), 33–38.
3. Furuyama, T. Arai, Y., and Iio, K. Fault generation model and mental stress effect analysis. In *Proc. of the Second International Conference on Achieving Quality in Software*, (Venice), Oct. 18–20, 1993.
4. Zawacki R. Motivating IT people in the 90s: An 'alarming drop' in job satisfaction. *Softw. Pract.* 3, 6 (Nov. 1993), 1, 4–5.

ROBERT GLASS is the publisher of the *Software Practitioner* newsletter and editor of Elsevier's *Journal of Systems and Software*. He welcomes feedback: 1416 Sare Rd., Bloomington, IN 47401.