



## References

- Davis, Alan M. 1995. *201 Principles of Software Development*. New York: McGraw-Hill.
- Glass, Robert L. 1979. *The Power of Peonage*. Computing Trends.

## PEOPLE

---

### Fact 1

**The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.**



## Discussion

People matter in building software. That's the message of this particular fact. Tools matter. Techniques also matter. Process, yet again, matters. But head and shoulders above all those other things that matter are people.

This message is as old as the software field itself. It has emerged from, and appears in, so many software research studies and position papers over the years that, by now, it should be one of the most important software “eternal truths.” Yet we in the software field keep forgetting it. We advocate process as the be-all and end-all of software development. We promote tools as breakthroughs in our ability to create software. We aggregate a miscellaneous collection of techniques, call that aggregate a methodology, and insist that thousands of programmers read about it, take classes in it, have their noses rubbed in it through drill and practice, and then employ it on high-profile projects. All in the name of tools/techniques/process over people.

We even revert, from time to time, to anti-people approaches. We treat people like interchangeable cogs on an assembly line. We claim that people work better when too-tight schedules and too-binding constraints are imposed on them. We deny our programmers even the most fundamental elements of trust and then expect them to trust us in telling them what to do.

In this regard, it is interesting to look at the Software Engineering Institute (SEI) and its software process, the Capability Maturity Model. The CMM assumes that good process is the way to good software. It lays out a plethora of key process

areas and a set of stair steps through which software organizations are urged to progress, all based on that fundamental assumption. What makes the CMM particularly interesting is that after a few years of its existence and after it had been semi-institutionalized by the U.S. Department of Defense as a way of improving software organizations and after others had copied the DoD's approaches, only then did the SEI begin to examine people and their importance in building software. There is now an SEI People Capability Maturity Model. But it is far less well known and far less well utilized than the process CMM. Once again, in the minds of many software engineering professionals, process is more important than people, sometimes spectacularly more important. It seems as if we will never learn.



### Controversy

The controversy regarding the importance of people is subtle. Everyone pays lip service to the notion that people are important. Nearly everyone agrees, at a superficial level, that people trump tools, techniques, and process. And yet we keep behaving as if it were not true. Perhaps it's because people are a harder problem to address than tools, techniques, and process. Perhaps it's like one of those "little moron" jokes. (In one sixty-year-old joke in that series, a little moron seems to be looking for something under a lamp post. When asked what he is doing, he replies "I lost my keys." "Where did you lose them?" he is asked. "Over there," says the little moron, pointing off to the side. "Then why are you looking under the lamp post?" "Because," says the little moron, "the light is better here.")

We in the software field, all of us technologists at heart, would prefer to invent new technologies to make our jobs easier. Even if we know, deep down inside, that the people issue is a more important one to work.



### Sources

The most prominent expression of the importance of people comes from the front cover illustration of Barry Boehm's classic book *Software Engineering Economics* (1981). There, he lays out a bar chart of the factors that contribute to doing a good job of software work. And, lo and behold, the longest bar on the chart represents the quality of the people doing the work. People, the chart tells us, are far more important than whatever tools, techniques, languages, and—yes—processes those people are using.

Perhaps the most important expression of this point is the also-classic book *Peopleware* (DeMarco and Lister 1999). As you might guess from the title, the entire book is about the importance of people in the software field. It says things

like “The major problems of our work are not so much technological as sociological in nature” and goes so far as to say that looking at technology first is a “High-Tech Illusion.” You can’t read *Peopleware* without coming away with the belief that people matter a whole lot more than any other factor in the software field.

The most succinct expression of the importance of people is in Davis (1995), where the author states simply, “People are the key to success.” The most recent expressions of the importance of people come from the Agile Development movement, where people say things like “Peel back the facade of rigorous methodology projects and ask why the project was successful, and the answer [is] people” (Highsmith 2002). And the earliest expressions of the importance of people come from authors like Bucher (1975), who said, “The prime factor in affecting the reliability of software is in the selection, motivation, and management of the personnel who design and maintain it,” and Rubey (1978), who said, “When all is said and done, the ultimate factor in software productivity is the capability of the individual software practitioner.”

But perhaps my favorite place where people were identified as the most important factor in software work was an obscure article on a vitally important issue. The issue was, “If your life depended on a particular piece of software, what would you want to know about it?” Bollinger responded, “More than anything else, I would want to know that the person who wrote the software was both highly intelligent, and possessed by an extremely rigorous, almost fanatical desire to make their program work the way it should. Everything else to me is secondary. . . .” (2001).



## References

- Boehm, Barry. 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.
- Bollinger, Terry. 2001. “On Inspection vs. Testing.” *Software Practitioner*, Sept.
- Bucher, D. E. W. 1975. “Maintenance of the Computer Sciences Teleprocessing System.” Proceedings of the International Conference on Reliable Software, Seattle, WA, April.
- Davis, Alan M. 1995. *201 Principles of Software Development*. New York: McGraw-Hill.
- DeMarco, Tom, and Timothy Lister. 1999. *Peopleware*. 2d ed. New York: Dorset House.

# About Management

**Fallacy 1**

You can't manage what you can't measure.

**Discussion**

The purpose of this saying is to point out that measurement is invaluable to managers. Clearly, managers need to know the answer to questions such as how much, when, and how well. There is a whole software engineering subfield devoted to the topic of software metrics, and proposals of new things to count—and how to count things long understood—are rampant.

What is interesting about the field of software metrics is that it is used little in practice. In surveys of tools and techniques used by software managers, metrics generally come in close to last. There are exceptions, of course—some enterprises (for example, IBM, Motorola, and Hewlett-Packard) place heavy emphasis on metric approaches. But for the most part, metric approaches are soundly ignored. Why is that? Perhaps managers are unconvinced of the value of metrics. Perhaps some of the necessary data is too hard to collect.

But there have been lots of studies of both the value and the cost of metrics, most of which have positive findings. At NASA-Goddard, for example, studies have shown that the ongoing cost of collecting the necessary metrics should be no more than 3 percent (data collection and analysis) + 4 to 6 percent (processing and analyzing the data) = 7 to 9 percent of the total cost of the project (Rombach 1990). NASA-Goddard considers that to be a bargain, given the value of their results.

Some of the history of metrics approaches has been tainted, however. Originally, managers all too often collected data that didn't matter or that cost too much to obtain. Such helter-skelter metrics collection was expensive and, as it turned out, pointless. It wasn't until the notion of the GQM approach (originally proposed by Vic Basili)—establish Goals to be satisfied by the metrics, determine what Questions should be asked to meet those goals, and only then collect the Metrics needed to answer just those questions—that there began to be some rationality in metrics approaches.

There was also the problem of software science. Software science was an attempt by the brilliant computing pioneer Murray Halstead to establish an underlying science for software engineering (Halstead 1977). He defined factors to measure and ways of measuring them. It seemed a worthy and, at the time, an important goal. But study after study of the numbers obtained showed neutral or negative value to the software science data. Some even likened software science to a form of astrology. The collection of “scientific” data about software projects eventually fell into disrepute and has, for the most part, been abandoned. Those who remember the software science debacle tend to taint all software metrics activities with the same brush.

Nevertheless, the collection of software metric data now happens often enough that there is even a “top 10” list of software metrics, the ones most commonly used in practice. To present an answer to the question “what are software metrics?” we present that list here.

Software Metrics	% Reported Using
Number of defects found after release	61
Number of changes or change requests	55
User or customer satisfaction	52
Number of defects found during development	50
Documentation completeness/accuracy	42
Time to identify/correct defects	40
Defect distribution by type/class	37
Error by major function/feature	32
Test coverage of specifications	31
Test coverage of code	31

Perhaps equally interesting is the list of the bottom 5 metrics:

Software Metrics	% Reported Using
Module/design complexity	24
Number of source lines delivered	22
Documentation size/complexity	20
Number of reused source lines	16
Number of function points	10

(This data comes from Hetzel [1993]. There is no reason to believe that the intervening years since 1993 would have changed this list a great deal, although advocates of function points claim a recent rise in their usage.)



### Controversy

The problem with the saying “you can’t manage what you can’t measure”—what makes it a fallacy—is that we manage things we can’t measure all the time. We manage cancer research. We manage software design. We manage all manner of things that are deeply intellectual, even creative, without any idea of what numbers we ought to have to guide us. Good knowledge worker managers tend to measure qualitatively, not quantitatively.

The fact that the saying is a fallacy should not cause us to reject the underlying truth of the message it brings, however. Managing in the presence of data is far better and easier than managing in its absence. In fact, it is the nature of managers—and human beings in general—to use numbers to help us understand things. We love batting and fielding and earned run averages. We love basket and rebound and assist counts and invent terms like *triple double* to accommodate combinations of them. We even invent data to cover subjects when there is no natural data, such as ice skating and diving (where judges assign scores to performances).

This is a case in which the fact is that measurement is vitally important to software management, and the fallacy lies in the somewhat-cutesy saying we use to try to capture that.



### Source

The saying “you can’t manage what you can’t measure” appears most frequently in books and articles on software management, software risk, and (especially)

software metrics. An interesting thing happened when I set out to track down where the saying originally came from. Several metrics experts said that it came from *Controlling Software Projects* (DeMarco 1998), and so I got in touch with Tom DeMarco himself. “Yes,” said DeMarco, “it’s the opening sentence in my book, *Controlling Software Projects*. But,” he went on to say, “the saying is actually ‘you can’t control what you can’t measure.’” Thus the fallacy version of the saying is actually a corruption of what DeMarco really said!



### References

- ➔ DeMarco, Tom. 1998. *Controlling Software Projects: Management, Measurement, and Estimation*. Englewood Cliffs, NJ: Yourdon Press.
- ➔ Halstead, M.H. 1977. *Elements of Software Science*. New York: Elsevier Science.
- ➔ Hetzel, Bill. 1993. *Making Software Measurement Work*. Boston: QED.
- ➔ Rombach, H. Dieter. 1990. “Design Measurement: Some Lessons Learned.” *IEEE Software*, Mar.

## Fallacy 2

**You can manage quality into a software product.**



### Discussion

This is a reprise of an idea presented previously in this book. In the section About Quality, I asked the question “whose responsibility is quality? My answer, as you may remember, was that no matter how many people believed” that management was responsible for product quality, there was too much technology to the subject of software quality to leave it up to management. I then went on at that point to say that nearly every one of the quality “-ilities” had deeply technical aspects, aspects that only a technologist could work with.

Not only is the achievement of quality a technical task, but those who believe that it is a management task often go about it in the wrong way. Over the years, managers have tried to use motivational campaigns to instill a quality viewpoint, as if the average technologist would be interested only in quality if he or she were pushed to do so. Sloganeering—“Quality Is Job One”—and methodologizing—“Total Quality Management”—seem to be management’s chief approaches to achieving software product quality. Far from accepting these approaches, technol-