I doubt anyone could ultimately prove or disprove that software-quality standards improve the quality of our products. Standards are only one of many factors that influence quality. However, there is ample evidence to suggest that, on balance, standards and their related guides and recommended practices do improve product quality. Although some standards have shortcomings, the software community is better off with imperfect standards than no standards at all.

There are many examples of standards improving software-product quality.

- **Space Shuttle.** Loral Space Information Systems, a level 5 organization on the Capability Maturity Model, has developed a process for the onboard shuttle software systems that is based in part on ANSI/AIAA Recommended Practice for Software Reliability. According to Ted Keller, manager of project coordination, the process has contributed significantly to the prevention of operational failure of space shuttle software. In addition, Loral is essentially in compliance with the ANSI/IEEE Standard for a Software Quality Metrics Methodology.

- **World Wide Web.** We would not have the WWW or be able to surf the Internet without the underlying protocols that serve as its basis: Transmission Control Protocol (MIL-STD 1778) and Internet Protocol (MIL-STD 1777). These US Department of Defense protocols provide interoperability across diverse platforms on the Internet.

- **Local area networks.** Although PCs and workstations operating over local area networks are not perfectly compatible, they would be far less compatible if not for the IEEE 802 LAN standards. Without the protocol standards that have been implemented in software, it would be impossible to use a variety of vendor hardware and operating-system platforms on the same Ethernet or token-ring network. Without a high degree of software compatibility, we cannot achieve high software quality.

**Information and education.** In addition to specifying product design or operation, standards can educate and inform us about structured software-engineering processes, current practices, and how to set up processes such as quality-assurance programs.

This informal use of standards is probably far more important than the traditional one because it initiates significant cultural change in the workplace. When people transfer from more traditional jobs such as programming into jobs in quality assurance, for example, they need information to help them do their new jobs. Standards can help provide this information.

By increasing understanding, standards improve quality. Without standards, project personnel have a difficult time communicating technical ideas and information. For example, different engineers may have different definitions and interpretations of things as fundamental to software quality as "reliability." The ANSI/IEEE Standard Glossary of Software Engineering Terminology can help minimize this confusion.

**Developing standards.** Another important factor to consider when judging standards is that the ones produced by professional organizations — AIAA, IEEE, and ISO — are developed by consensus in working groups, and thus diverse points of view are considered. The drafts receive extensive review before going to ballot. In ballot, drafts must receive a specified percentage of positive votes before being issued as standards, and all negative votes must be resolved. In the case of IEEE standards, they must be reaffirmed by ballot every five years. Thus, practitioners are assured that the standard represents a community viewpoint and not the
Between 1990 and 1994, a group of us at the Centre for Software Reliability were involved in Smartie, a collaborative project whose primary objective was to propose a method for assessing the efficacy of software-engineering standards. As far as possible, the method was intended to be objective (based on measurement). We can best summarize the point of our inquiry by rephrasing the title of this debate: a software standard is effective if, when used properly, it improves the quality of the resulting software products cost-effectively. We found no evidence that any of the existing standards are effective according to this criterion.

This will come as no surprise to anyone who has sought quantitative evidence about the effectiveness of a software-engineering method or tool. However, what concerned us more was that, in general, software-engineering standards are written in such a way that we could never determine whether they were effective or not.

No shortage. We came across more than 250 standards that fell within the scope of software engineering. They ranged from those that define general quality-assurance procedures to those that define specific techniques, such as unit testing. All defined a perceived “best practice” for developing high-quality software systems or components.

Unfortunately, there is no consensus on what constitutes best practice, and thus there is no consensus on how to choose the techniques that should always be applied. The result? For standards with similar names and objectives, we came across very different models of software quality and the software-development process.

For example, there are now many standards dealing with safety-critical software. Some are based on formal methods; others totally ignore this approach. Even relatively old issues like configuration management take on radically differing appearances in different standards. All of this suggests that there is little in the domain of software engineering that is sufficiently mature for standardization.

Better than nothing? It has been argued that using any standard is better than using none. However, unlike other engineering disciplines — where this assumption may be valid — software-engineering standards pose unique problems.

Software standards overemphasize process. Traditional engineering standards assure product quality by specifying properties that the product must satisfy, including extensive product testing. Software standards almost entirely neglect the product and concentrate on the development process. Unfortunately, there is no guarantee that a “good” process will lead to a good product, and there is no consensus that the processes mandated in many standards are even “good.”

Many software standards aren’t standards. Traditional standards are a set of mandatory requirements. Such requirements must be sufficiently precise so that conformance can be determined objectively by appropriate tests. When precision — and hence mandatory enforcement — is impossible, traditional standards bodies use the terms “codes of practice” or “guidelines.” Software engineering makes no such distinction. We are thus subject to a proliferation of “standards” that are at best guidelines that could never be realistically mandated.

It is impossible to measure conformance to software standards. We found that a vast majority of requirements contained in software standards are presented in such a way that it is impossible to determine conformance in any objective sense. Thus, in general, it would be impossible to determine whether or not the standard has been applied. This makes a mockery of many of the assumed benefits of standardization.

Many software standards prescribe, recommend, or mandate the use of technology that has not been validated objectively. The standards may therefore be mandating methods that are not effective for achieving high-quality systems.

Many software standards are simply too big. For example, numerous standards attempt to address the complete system-development life cycle. This results in extremely large...
Point

opinion of a single individual or corporation.

Portability and certification. In today's distributed-computing environment, applications, information, and users must be portable across multiple platforms. Otherwise, the costs of writing software and training users for vendor- or application-specific interfaces are prohibitive. In fact, many believe that the creation of "a computing environment based on widely implemented vendor-neutral standards is essential to compete in the global marketplace of the '90s and beyond."\(^7\) ANSI/IEEE Std. 1003.1-1991 Portable Operating System Interface (Posix) is the major standard designed to achieve portability. Without portability, there is no software quality.

Perhaps the most significant example of standards influencing quality is the certification process of ISO 9000 quality system standards. Whether you like this process or not, it is hard to deny that it forces vendors to put greater emphasis on product quality — including software quality.

The verdict is in: Customers do receive higher quality software products when software-quality standards are used in product development, because standards require that the developer comply with a documented, formal, rigorous, disciplined, and repeatable process.

REFERENCES

Norman Steneideuwedt is a professor of information sciences at the Naval Postgraduate School, where he specializes in research on software measurement. He is a fellow of the IEEE.

Counterpoint

documents containing sets of unrelated requirements, many of which will be irrelevant in a given application. Such standards are almost impossible to apply and are generally left on the shelf.

I, along with my Smartie colleagues, presented some simple, practical advice to standards' bodies for avoiding problems like these.\(^1\) Unfortunately, the standards-making process is long and tortuous and often contributes to some of these problems itself. Perhaps it is time the software industry paid for the development of good, timely standards rather than continuing to rely on the contributions of individuals who volunteer their effort to standards-making bodies. While such contributions are, more often than not, heroic and unsung, they are nevertheless entirely ad hoc. As such, we deserve nothing better than the ad hoc standards we have at present.

REFERENCE

Norman Fenton is a chartered engineer and a professor of computing science at the Centre for Software Reliability, City University, London. His current research interests are in the areas of software metrics, safety-critical systems, and formal development methods.