The Link Between Software Quality and Software Maintenance

What is software quality? Most commonly, it is defined as a set of attributes. If software is to be of good quality, then it must have:

- reliability
- efficiency
- human engineering
- understandability
- modifiability
- testability
- portability

What is software maintenance? It is the task of keeping software functioning to the satisfaction of a user.

It is interesting to note that, of those seven quality attributes, nearly all of them relate directly or indirectly to the key concepts in the definition of maintenance. Two of them, in fact, are specific to software maintenance and most of the others are critical to software maintenance. In other words,

viewed properly, the task of building quality into software is almost the same as the task of making it maintainable.

That is an all too unusual view of software quality. Normally, we think of quality as an activity that developers create and quality assurance people monitor, almost for its intrinsic value. You build software? You give it quality, of course.

But that's a myopic view because the maintainability aspects of quality are the most difficult to achieve, and therefore the most easily forgotten when we see quality as kind of an "automatic" thing.

Let's look at the preceding key "ilities" to try to expand that myopic view to include maintenance concerns. The first two "ilities" that should catch the eye, from a maintenance point of view, are:

- *Understandability*—the ability of a reader of the software to understand its function.
- *Modifiability*—the ability of the software to be changed by that reader.

Now those two quality attributes are almost entirely about maintenance. Hardly anyone else needs to modify software, and few others need to understand it! (And certainly, they are also the quality attributes most difficult to assess.) What about the others? In order of interest from a maintenance point of view, they are:

- *Reliability*—the ability of the software to perform its intended function without failure. Now there's a key element in maintenance! If software isn't reliable, it's the maintainer's job to fix it.
- *Efficiency*—the ability of the software to operate with minimal use of time and space resources. Once again, that's as much about maintenance as it is about any development activity. Software that takes too long or grabs too much space will have to be shrunk by the software maintainer.
- *Testability*—the ability of the software to be tested easily. Usually we think of testing as a development activity, but certainly a large part of the maintainer's job is testing each change made to the product, and regression testing the things not changed.
- *Human engineering*—the ability of the software to be easily used. If the software is *not* easily used, who bears the brunt of the pain? The maintainer.
- *Portability*—the ease with which software can be made useful in another environment, such as a different computer or operating system. Once again, if software is ported, it may very well be the maintainer who gets called upon to do the job.

Not only are two of the quality attributes intimately linked to maintenance, but the others are linked solidly as well. Software quality, then, is as much about maintenance as it is about anything else.

In fact, there are other "ilities" of software quality that those involved with data processing usage talk about. They are the service attributes, related to the user's prime concerns rather than the developer's. They are timeliness, accuracy, reliability, and cost. Now the continuing achievement of those additional "ilities" lies entirely in the domain of the maintainer.

What's the point of these thoughts? Too often, software maintenance is an afterthought in everyone's definition of computing. Computer science and software engineering rarely talk about it. Until recently, tools and techniques were nonexistent, and even today research pays maintenance scant attention. Perhaps if we begin to see how important maintenance is to something right at the heart of the field of software, it will at last get the attention it deserves.

Barry Boehm has suggested that a key member of the software quality assurance team should be someone who will eventually maintain the product in question. Now there is a first step in the right direction. Not only does it work the problem, but it clearly relates the two topics of software quality and software maintenance that seem so seldom related to each other now.