# Myths, beliefs and superstitions about the quality of software and of its teaching

Adolfo Guzman-Arenas

Centro de Investigacion en Computacion (CIC), Instituto Politecnico Nacional, Mexico
a.guzman@acm.org

**Abstract.** It is a surprise to see how, as years go by, two activities so germane to our discipline, (1) the creation of quality software, and (2) the quality teaching of software construction, and more generally of Computer Science, are surrounded or covered, little by little, by beliefs, attitudes, "schools of thought," superstitions and fetishes rarely seen in a scientific endeavor. Each day, more people question them less frequently, so that they become "everyday truths" or "standards to observe and demand." I have the feeling that I am minority in this wave of believers and beliefs, and that my viewpoints are highly unpopular. I dare to express them because I fail to see enough faults in my reasoning and reasons, and because perhaps there exist other "believers" not so convinced about these viewpoints, so that, perhaps, we will discover that "the imperator had no clothes, he was naked."

## 1. Myths and beliefs about the production of quality software

This section lists several "general truths," labeled A, B…, G concerning quality of software, and tries to ascertain whether they are reasonable assertions ("facts," sustainable opinions) or myths.

### 1.1 About measuring software quality

**A. It is possible to measure the main attributes that characterize good quality software.** The idea here is that software quality can be characterized by certain attributes: reliability, flexibility, robustness, comprehension, adaptability, modularity, complexity, portability, usability, reuse, efficiency… and that it is possible to measure each of these, and therefore, characterize or measure the quality of the software under examination. To ascertain whether point A is a fact or a myth, let us analyze three facets of it.

*1) It is possible to measure above attributes subjectively,* asking their opinion to people who have used the software in question.

*Comment 1.* **Opinions by experienced users are reliable.** That is, (1) is not a myth, but something real. It is easy to agree that a program can be characterized by above attributes (or similar list). Also, it is convincing that the opinions of a group of qualified users respect to the quality, ergonomics, portability… of a given software are reliable and worth to be taken into account (subjective, but reliable opinions).

2) Another practice is to try to measure above attributes *objectively,* by measuring surrogate attributes if the real attribute is difficult to measure [Myth B below].

*Comment 2.* **Measuring surrogate attributes.** To measure the height of a water tank when one wishes to measure its volume, is risky. Objective (accurate) measurements of surrogate attributes may be possible, but to think that these measures are proportional to the real attribute, is risky. "If you can not measure beauty of a face, measure the length of the nose, the color of eyes…" If you can not measure the complexity of a program, measure the degree of nesting in its formulas and equations, and say that they are directly related. More in my comments to Myth B.

3) Finally, instead of measuring the quality of a piece of software, go ahead and measure the quality of the *manufacturing process* of such software: if the building process has quality, no doubt the resulting software should have quality, too (Discussed below as Myth C).

*Comment 3.* **To measure the process, instead of measuring the product.** In old disciplines (manufacturing of steel hinges, leather production, wine production, cooking…) where there are hundred of years of experience, and which are based in established disciplines (Physics, Chemistry…), it is possible to design a process that guarantees the quality of the product. A process to produce good leather, let us say. And it is also possible to (objectively) measure the quality of the resulting product. And to *adapt* the process, modifying it to fix errors (deviations) in the product quality: for instance, to obtain a more elastic leather. Our problem is that *it is not possible* to do that with software. We do not know what processes are good to produce good quality software. We do not know what part of the process to change in order, let us say, to produce software with less complexity, or with greater portability. More in my comments to Myth C.

**B. There exists a reliable measurement for each attribute.** *For each attribute to be measured, there exists a reliable, objective measurement* that can be carried out. The idea is that, if the original attribute is difficult to measure,[1] measure another attribute, correlated to the first, and report the (second) measurement as proportional or a substitute for the measure of the original attribute.

1. Reliability (reliable software, few errors): measure instead the number of error messages in the code. The more, the less errors that software has.
2. Flexibility (malleability to different usage, to different environments) or adaptability: measure instead the number of standards to which that software adheres.
3. Robustness (few drastic failures, the system rarely goes down): measure through tests and long use (Subjective measurement).

[1] Or we do not know how to measure it.

4. Comprehension (ability to understand what the system does): measure instead the extent of comments in source code, and the size of its manuals.
5. The size of a program is measured in bytes, the space it occupies in memory (This measurement has no objection, we measure what we want to measure).
6. Speed of execution is measured in seconds (This measurement has no objection, we measure what we want to measure).
7. Modularity: count the number of source modules forming it.
8. Program complexity (how difficult it is to understand the code): measure instead the level of nesting in expressions and commands ("cyclomatic complexity").
9. Portability (how easy it is to port a software to a different operating system): ask users that have done these portings (Subjective measurement).
10. Program usability (it is high when the program brings large added value to our work. "It is essential to have it."): measure the percentage of our needs that this program covers (Subjective measurement).
11. Program reuse: measure how many times (parts of) this program have been used in other software development projects. (Objective measurement, but only obtained in hindsight).
12. Ease of use (ergonomics) characterizes programs that are easy to learn, tailored to our intuitive ways to carry out certain tasks. Measure instead the quantity of screens that interact with the user, and their sophistication.

*Comment 4.* **Measuring surrogate attributes.** These "surrogate measurements" can produce irrelevant figures for the quality that we are really trying to measure. For instance, the complexity of a program will be difficult to measure using point 8, for languages that use no parenthesis for nesting. For instance, it is not clear that a software with long manuals is easier to comprehend (point 4). To measure the temperature of a body when one wants to measure the amount of heat (calories) in it, is incorrect and will produce false results. A very hot needle has less heat that a lukewarm anvil.

*Comment 5.* It is true that in the production of other goods, say iron hinges, is easy to list the qualities that a good hinge must possess: hardness, resistance to corrosion... And it is also easy to objectively measure those qualities. Why is it difficult, then, to measure the equivalent quantities about software? Because hinges have been produced before Pharaohnic times, humankind has accumulated experience on this, and because its manufacture is based on Physics, which is a consolidated science more than 2,000 years old. Physics has defined units (mass, hardness, tensile strength…) capable of objective measurement. More over, Physics often gives us equations ($f = ma$) that these measurements need to obey. In contrast, Computer Science has existed only for 60 years, and thus almost all its dimensions (reliability, ease of use…) are not susceptible (yet) of objective measurements. Computer Science is not a science yet, it is an art or a craft.[2] Nevertheless, it is tempting to apply to software characterization (about its quality,

[2] Remember the title of the book "The Art of Computer Programming" of Donald C. Knuth. In addition, we should not be afraid that our science begins as an art or a craft. Visualize Medicine when it was only 60 years old: properties of lemon tea were just being discovered. And physicians talked for a long time of fluids, effluvia, bad air, and witchcraft. With time, our discipline will become a science.

say), methods that belong and are useful in these more mature disciplines, but that are not (yet) applicable in our emerging science. We are not aware that methods that work in leather production, do not work in software creation. Indeed, it is useful at times to talk of software *creation,* not of software production, to emphasize the fact that software building is an art, dominated by inspiration, good luck… (see Comment 7).

## 1.2 Measuring the process instead of measuring the product

An indirect manner to ascertain the quality of a piece of software, is to review the quality of the process producing it.

**C. Measuring the quality of the process, not the product quality.** Instead of measuring the quality of the software product, let us measure the quality of its construction process. To have a good process implies to produce quality software.

*Comment 6.* It is tempting to claim that a "good" process produces good quality software, and therefore, deviations of programmers with respect to the given process should be measured and corrected. The problem here is that it is not possible to say which process will produce good quality software. For instance, if I want to produce portable software, what process should I introduce, versus if what I want to emphasize is ease of use? Thus, the definition of the process becomes very subjective, an act of faith. Processes are used that sound and look reasonable, or that have been used in other places with some success. Or that are given by some standard or international committee. "If so many people use them, they must be good." We need to recognize that our discipline is not (yet) a science nor an Engineering discipline, where one can design a process that guarantees certain properties in the resulting product, much in the same manner that the time and temperature of an oven can be selected to produce hinges of certain strength. Instead, our discipline is more of an art or a craft, where inspiration counts, "to see how others do it," "to follow the school of Prof. Wirth," to follow certain rites and traditions or tics that a programmer copied (perhaps unconsciously) from his teacher.

*Comment 7.* A more contrasting manner to see that certain measurement processes are not applicable to certain areas, is to examine an art, such as Painting or Symphony Composition. Following the rules of the hard disciplines (manufacturing of hinges), we would first characterize the quality symphonies as those having sonority, cadence, rhythm… Here, measuring those qualities becomes (as in software) subjective. Then, we would establish the rules that govern the *process* of fabrication of symphonies (by observing or asking notable composers, say Sergei Prokoffiev): the pen needs to have enough ink, use thick point; the paper must have a brightness no less than x, its thickness must be at least z; it must be placed on the desk forming an angle not bigger than 35 degrees. Light shall come from the left shoulder. Certainly, these rules will not hurt. But there is no guarantee that anybody that follows them will produce great quality symphonies, even if the very same rules in hands of Prokoffiev produce excellent results, over and over.

**D. If you have a controlled process, you will produce good quality software.** It is easy to know when you have a "good" (reasonable) process. It is easy to design a "good" process to produce software.

*Comment 8.* On the other hand, it is not really known which processes will produce easy-to-use software, which other processes will produce portable software, or software with good real-time properties, etc. The "process design" bears thus little relation to the goal: to produce a software product with this and that features. The problem resembles that of hospital surgeons in the pre-Pasteurian period, when bacteria were not yet discovered. Many people who underwent surgery died, full of infection and pus, without anybody knowing why. Of course, it was easy to measure the quality of a product of a surgery process: "birth-giving woman died of septicemia." Therefore, processes were designed to make sure that patients with surgery would not die: when coming to work, surgeons should pray to Saint Diego. Then, hang from your neck a chain of garlic bulbs. Then, wash your hands. You shall not commit surgery during days with nights having full moon… These rules certainly did not hurt (did not produce worse results), but they were not very related to the quality of the final results. Once bacteria were discovered, the rules were simplified, fine tuned and complemented with others: "wash your knives," "disinfect your hands." It is my impression that, in software creation, we are in a pre-Luis Pasteur epoch, and that we invent rules and processes "to have one at hand," but that the results (the quality of the resulting software) of these processes have little to do with the invented process, and with its fulfillment or lack thereof.

**E. It is necessary to create "quality champions," quality committees,** and other human organizations whose goal is "to promote quality (of software)." Generally, a committee of this type (1) generates norms and rules saying how the construction of software is to be handled (regulations about the process; they *define* the process), including formats that certain intermediate and final documents (manuals, say) shall have, and (2) it observes if the programming team follows the rules (1), seeking to correct deviations.

*Comment 9.* These committees, since they do not know for sure neither how to measure the quality of the product (Myths A and B) nor how to alter the fabrication process if certain output attributes are unacceptable (Myth D), end up becoming hindrances and stereotyped bureaucracies. What they can demand (and they do) from the programming and design team is adherence to the process invented by said committee (or copied from an international organization). If they adhere and follow the process, "that is good," and (by faith) "good quality software shall be the result." If the team deviates, that is bad; offenders should be punished and be blamed for the bad quality of the results. This is equivalent to have, in a pre-Pasteurian hospital (see Comment 8) a committee that, watching that this week more patients died of general infection than in the previous week, strengthens its efforts and detects surgeons that did not pray to Saint Diego, while others hanged from their necks garlic bulbs that were not fresh. Let us reprehend these offenders, and less patients shall die.

**F. Attitude matters.** The right mind-set towards quality shall permeate and impregnate each coder. The designer or programmer must be constantly thinking about quality, must have faith in that he will produce good quality software; he shall watch that the quality of his works be above a (high) minimum.

*Comment 10.* This is an act of faith, that certainly will not hurt. But it helps little. Software confection should not be based on faith or beliefs. Certainly, it helps somewhat that a programmer says each morning "today I am going to produce high quality software, I am sure of that," much in the same manner as a pre-Pasteurian surgeon said "Today, no one of my patients undergoing surgery will die; today, no one of my patients undergoing surgery will die." With respect to the idea that a programmer "shall watch the quality of his production," this is commendable, but it is certainly difficult, since *it is difficult to measure software quality,* even if the person measuring is the software builder.

### 1.3 The myth of standards

**G. Adhesion to standards means high quality software**. "If we do software construction following the rules dictated by a standards organization, we will be producing good quality software." "Following software construction standards ensures the quality of the process and the quality of the resulting software." That is to say, of the many processes that we could follow when creating software, let us use one that is part of a norm or standard (preferably, an international one), or let us follow the process used by a company that produces good quality software (Oracle, say).

*Comment 11.* Nothing wrong can be perceived in this practice. It is as if the surgeons of a low quality hospital (of Comment 8) decide to copy the surgery process of The Hospital of the Holy Virgin Mary, which has low post-surgery mortality. Or if I want to use Prokoffiev's "rules for composing good symphonies" (Comment 7). No evil will come out of this. Nevertheless, subjectivity and the scanty relation between these "preferred" procedures and the quality of the resulting software must be clear, as Comment 8 explains.

## 2. Myths and beliefs about the quality of teaching in Computer Science

We now examine how quality in Computer Science schools is measured.

**It is enough to measure the product**

It seems very reasonable and obvious (but let us examine it) to measure the quality of the product, in order to ascertain its quality.

**H. Measure the product and see if it is of good quality.** If I make a test or examination to two young undergraduate alumni of different Computer Science schools, and one of then knows more computer science than the other, certainly the person knowing more is of better quality. Example: the organism "Ceneval" in Mexico, who does just that.

*Comment 12.* This measurement is "almost right," except that it does not measure the great *obsolescence* in our field. I estimate that the mean half-life[3] of a computer science concept is about 5 years. That is, every five years, half of what we know becomes useless; not because we forget concepts or because they were inadequately learnt. Just because these concepts are no longer useful, they are obsolete: bubble memories, remote job entry, punch tape… The measurement of point H simply measures the *today* quality, the quality *as measured today.* What will happen in five or ten years with alumnus 1 versus alumnus 2? May be alumnus 1 still displays useful knowledge, while alumnus 2 (the more knowledgeable today) no longer has. One rusted faster than the other. That is, alumni formation (specially in high obsolescence fields, such as ours –this is due to its youth, scantly 60 years old) depends on two factors: (a) the basic, theoretical knowledge, which will enable our alumni to *keep acquiring knowledge* through his productive life, outside College; and (b) the "today" knowledge, the knowledge that is "fashion today" (in 2004, objects, UML, Java, say) that will allow them to become immediately productive. "To go out into the sugar cane field and start cutting cane." Knowledge acquired in College is like the quality of a machete, which depends on two attributes: its *temper,* that permits it several resharpenings along its productive life (and we need to prepare alumni capably of a productive life of 40 years), and the *sharpness* of the cutting edge (which allows immediate productivity, "to go out and start cutting cane." My problem with procedure H is that only measures the *sharpness* of the edge, the "today usefulness." Add measurements every five years (longitudinal studies), or add measurements (today) about the degree of theory and basic subjects (those that can hardly change in 50 years, say) that the alumnus has; this knowledge is what renders him resistant to obsolescence.

**I. Quality of a College is measured by the quality of its alumni.**

*Comment 13.* Again, this is "almost true." Certainly, we tend to give high quality to those Colleges that produce high-quality alumni. But often these schools require the entering students *to have already high quality.* At entrance time. High entrance requirements. I only accept the best. Obviously, these people will exit school better prepared than students at another College who, due to incoming deficiencies, finish their studies less well prepared. To

[3] The half-life of a radioactive substance is the time taken by that substance's mass to decay to half its original mass.

be fair, the quality of a school should be measured by the *added value*. That is, measure the student at entrance and exit times, and perform a subtraction.

## 2.2 To measure quality of the alumni, it is sufficient to measure quality of the process

The certification process of a College implies measuring the teaching process that it carries. There is a thought that good (certified) colleges produce high quality alumni.

**J. Good teaching processes will produce good alumni.** To know the quality of an alumnus, it is enough to measure the quality of the teaching process.

*Comment 14.* This looks like §1.2, "let us measure the process instead of measuring the product." Again, the idea here is that it is possible to design a teaching process that guarantees the quality of the alumni. "He shall have laboratory practices." "Exams should be designed by a Department commission." "Every student shall have his own computer." Nevertheless, unlike §1.2 (which is just a belief, called Myth C), point J is true, it is a truth (not a myth).. This is due to the many centuries that humankind has devoted to education, which shines much light on how to design good educational processes. And also tells us what part of the process to modify if, say, students are not acquiring enough knowledge in Experimental Chemistry. I will only add that in educational processes (as in cooking) two things are also important: (a) the *ingredients,* the books, the available software, the transmitted knowledge, the syllabus, and (b) the *artisans,* the cookers, that is, the teachers, the instructors.

## Bibliography

These works are not referred in the text.

1. Roger S. Pressman. *Software Engineering, a practical approach.* McGraw Hill. A standard textbook on software engineering, with many methods for software construction, and software metrics.
2. I. Sommerville. *Software Engineering.* Addison-Wesley. Idem.