Mitos, creencias y supersticiones sobre la calidad del software y de su enseñanza

Adolfo Guzmán-Arenas
Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), México
a.guzman@acm.org

RESUMEN. Sorprende cómo, a través de los años, actividades tan básicas a nuestra profesión como la creación de software de calidad, y la buena calidad en la enseñanza en computación, se han rodeado o cubierto paulatinamente de creencias, actitudes, "escuelas de pensamiento", supersticiones y fetiches que raramente se ven en un campo científico o técnico, y que cada vez más personas cuestionan cada vez menos, de manera que se convierten en "verdades cotidianas" o "estándares a observar y a exigir."

Tengo la impresión de que soy minoría en esta ola de creyentes y creencias, y que mis puntos de vista son altamente impopulares. Me atrevo a expresarlos porque no alcanzo a ver fallas suficientes en mi razonamiento, y porque quizá halle a otros "creyentes" de las creencias actuales no tan convencidos de ellas, de modo que, tal vez, descubramos que al fin "el emperador no llevaba traje, estaba desnudo".

1. MITOS Y CREENCIAS SOBRE LA PRODUCCIÓN DE SOFTWARE DE CALIDAD

1.1 Sobre la medición de la calidad del software

A) *Es posible medir los principales atributos que forman o caracterizan a un software de buena calidad.* La idea aquí es que la calidad del software se caracteriza por ciertos atributos: fiabilidad, flexibilidad, robustez, comprensión, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia...., y que es posible medir cada uno de ellos, y por consiguiente, caracterizar o medir la calidad del software en cuestión.

1) *Es posible medir los atributos anteriores subjetivamente*, pidiendo su opinión a gentes que han usado el software que se está midiendo.

   Comentario 1. **Son confiables las opiniones de usuarios experimentados.** Es decir, (1) no es un mito, es algo real. Es fácil estar de acuerdo en que un programa puede caracterizarse por los atributos anteriores. Además, es convincente que las opiniones de un grupo de usuarios respecto a la calidad, ergonomía, portabilidad... de un software, sean confiables y dignas de tomarse en cuenta (opiniones subjetivas, pero confiables).

2) Otra forma de proceder es medir los atributos anteriores *objetivamente,* midiendo atributos alternos en caso de que el atributo real sea difícil de medir (Mito B).

   Comentario 2. **Medición de atributos alternos. Medir la altura de un tanque de agua cuando lo que se desea medir es su volumen, es arriesgado.** La idea de medir (objetivamente) atributos alternos es tentadora, pero arriesgada. "Si no puedo medir la

belleza, mide la longitud de la nariz, y el color de los ojos". Si no puedo medir la complejidad de un programa, mide el número de anidamientos en sus fórmulas o ecuaciones, y dí que están directamente relacionados. Más en mis comentarios al Mito B).

3) Finalmente, en vez de medir la calidad de un software, podemos medir la calidad del proceso de fabricación de tal software: si el proceso de fabricación es de calidad, el software resultante debe ser de calidad (Mito C).

Comentario 3. **Medir el proceso, en vez del producto.** En disciplinas antiguas (fabricación de bisagras, curtido de cueros, fabricación de vino, cocinar alimentos) donde hay experiencia de cientos de años, que además están apoyadas en ciencias sólidas (Física, Química...), es posible diseñar un proceso que garantice la calidad del producto (un proceso que produzca cueros de buena calidad, digamos). Y también es posible medir (objetivamente) la calidad del producto resultante. Y *adaptar* el proceso, modificándolo, para corregir errores en la calidad, por ejemplo, hacer un cuero más elástico. El problema es que *no es posible hacer esto en software,* porque no se sabe qué procesos son buenos para producir software de buena calidad, y porque no se sabe qué parte del proceso cambiar para, digamos, producir software con menor complejidad, o con mayor portabilidad. Siguen los comentarios en el Mito C).

B) *Para cada atributo a medir, hay una medición confiable* (objetiva) que puede llevarse a cabo. La idea es que, dado que el atributo deseable es difícil de medir, se mide otro atributo que está correlacionado con el primero.

1) La fiabilidad (software confiable, pocos errores) se mide a través del número de mensajes de error, mientras más haya, contiene menos errores este software.
2) La flexibilidad (capacidad de adaptación a diferentes tipos de uso, a diferentes ambientes de uso) o adaptabilidad se mide viendo a cuántos estándares este software se adhiere.
3) La robustez (pocas fallas catastróficas, el sistema "no se cae") se mide a través de pruebas y uso prolongado.
4) La comprensión (capacidad de entender lo que el sistema hace) se mide viendo la cantidad de comentarios que posee el software, y la extensión de sus manuales.
5) El tamaño de un programa se mide en bytes, el espacio que ocupa en memoria. (Esta medición no tiene objeción, se mide lo que se quiere medir).
6) La eficiencia de un programa se mide en segundos, es la rapidez en su ejecución. (Esta medición no tiene objeción, se mide lo que se quiere medir).
7) La modularidad de un programa se mide contando el número de módulos que lo forman.
8) La complejidad de un programa (qué tan fáicl es entender el código) se mide contando el número de anidaciones en expresiones o postulados ("complejidad ciclomática").
9) La portabilidad de un programa es la facilidad con que se eche a andar en otro sistema operativo distinto al que fue creado. Se mide preguntando a usuarios que han hecho estos trabajos.
10) La usabilidad de un programa es alta cuando el programa aporta gran valor agregado a nuestro trabajo. "Es indispensable contar con él". Se mide viendo qué porcentaje de nuestras necesidades cubre ese programa.
11) La reutilización de un programa se mide por la cantidad de veces que partes del mismo se han reutilizado en otros proyectos de desarrollo de software.

12) La facilidad de uso (ergonomía) caracteriza a los programas que no cuesta trabajo aprender, que se amoldan al modo intuitivo de hacer las cosas. Se mide observando la cantidad de pantallas que interaccionan con el usuario, y su sofisticación.

Comentario 4. **Medición de atributos alternos.** Estas "mediciones alternas" pueden dar números irrelevantes para la cualidad que realmente se quiere medir. Por ejemplo, la complejidad de un programa será difícil de medir según el punto 9), para lenguajes que no usan paréntesis para anidar. Por ejemplo, no está claro que un software que tiene manuales extensos, sea más fáciles de comprender (punto 4). Medir la temperatura de un líquido cuando lo que se desea obtener es la cantidad de calor (calorías) en él, es incorrecto y puede producir resultados falsos. Una aguja muy caliente contiene menos cantidad de calor que un yunque tibio.

Comentario 5. Es cierto que en la fabricación de otros satisfactores, digamos bisagras, es fácil listar las cualidades que una buena bisagra debe tener: dureza, resistencia a la corrosión... Y también es fácil medir objetivamente esas cualidades. ¿Porqué es difícil entonces medir las cualidades equivalentes si lo que se fabrica es software? Porque las bisagras se han estado fabricando desde los romanos (la humanidad ha acumulado experiencia), y porque su fabricación se basa en la Física, que es una ciencia madura de más de 2,000 años, y que ha definido unidades (masa, dureza, resistencia al corte...) que se pueden medir objetivamente. En contraste, la computación existe desde solo hace 60 años, y casi todos sus atributos o medidas (la confiabilidad, la facilidad de uso...) no se saben cómo medir objetivamente. Es decir, la computación no es una ciencia aún, es un arte o artesanía.[1] Pero la idea de aplicar al software métodos de caracterización (de calidad) que son útiles en disciplinas más maduras, más avanzadas, es atractiva, y las aplicamos sin percatarnos de que no son aplicables.

1.2 Medir el proceso, en vez de medir el producto

C) En vez de medir la calidad del producto, midamos la calidad del proceso. Tener un buen proceso implica producir software de buena calidad.

Comentario 6. Es tentador definir que un proceso produce software de buena calidad, y entonces medir las desviaciones que los programadores hacen con respecto de tal proceso. El problema es que no se sabe cómo deducir cuál proceso producirá buena calidad en el software. Por ejemplo, si quiero hacer un software transportable, ¿qué proceso debo implantar, versus si lo que deseo es enfatizar la facilidad de uso del software? Entonces la definición del proceso se vuelve muy subjetiva, un acto de fé. Se recurre a procesos que suenan o se ven razonables, o que han sido ensayados en otros lados con éxito. O que están dados por algún estándar o comité internacional. "Si tanta gente los usa, deben ser buenos." Hay que reconocer que nuestra disciplina aún no es una ciencia o ingeniería o desarrollo sistemático, donde se puede deducir un proceso que garantice ciertas cualidades en el producto resultante. Es más bien un arte o artesanía, donde cuenta "la inspiración", "ver cómo lo hicieron los demás", ciertas costumbres o tics que un programador le copió (inconscientemente, quizá) a su maestro.

Comentario 7. **Medir el proceso en vez del producto.** Una manera más contrastante de ver que ciertos procesos de medición no son aplicables a ciertas áreas, es exa-

[1] Recuerdo el título del libro "The Art of Computer Programming" fde Donald C. Knuth. Además, no debe asustarnos que nuestra ciencia comience como un arte o artesanía. Imagine usted la Medicina cuando solo tenía 60 años de edad: aún se estaban descubriendo las propiedades curativas del té de canela. Y se hablaba de fluidos, miasmas y malos aires, y de "mal de ojo." Con el tiempo, nuestra disciplina se convertirá en Ciencia.

minar un arte, como la Pintura o la Composición de Sinfonías. Siguiendo las reglas de las ciencias duras (fabricación de bisagras), primero caracterizaríamos a las pinturas o sinfonías de calidad como aquellas que son sonoras, que tienen cadencia, que evocan emociones placenteras... Es concebible que mucha gente esté de acuerdo con una lista como la anterior. Luego, diríamos cómo medir la sonoridad de una sinfonía, su cadencia... Aquí, la medición se torna (como en software) subjetiva. Luego, estableceríamos reglas que norman el *proceso* de confección de sinfonías: la pluma debe tener suficiente tinta, el papel debe tener una reflectancia no mayor a tal cosa, su grosor debe ser de cuando menos z, el papel debe colocarse en el escritorio formando un ángulo no mayor a los 35 grados... Ciertamente, estas reglas lo más seguro es que "no estorben" la creación de buenas sinfonías. Pero sería risible asegurar que todo aquél que las siga, producirá obras de gran calidad.

1.2.1 Si se tiene un proceso controlado, se producirá software de buena calidad

D) Es fácil saber cuándo se tiene un buen proceso. Es fácil diseñar un buen proceso para producir software.

Comentario 8. Por el contrario, no se sabe a ciencia cierta qué procesos producirán software fácil de usar, cuáles otros producirán software transportable, etc. El "diseño del proceso" entonces es una cuestión poco relacionada con el fin que se persigue. El problema se parece al que tenían los cirujanos de los hospitales en la época que precedió al descubrimiento de las bacterias por Luis Pasteur. Los operados se morían, se llenaban de infección y pus, sin saber por qué. Desde luego, era fácil medir la calidad del producto de una operación: "parturienta muerta por septicemia". Entonces se diseñaron procesos para asegurar que los pacientes operados no se murieran: al entrar a trabajar, los cirujanos deberían elevar una oración a San Dionisio. Luego, colgarse del cuello un collar de ajos. Luego, lavarse las manos. No se debía operar en días con noches de luna llena... Estas reglas ciertamente no estorbaban, pero no estaban muy relacionadas con el resultado final. Una vez descubiertas las bacterias, esas reglas se simplificaron y complementaron con otras: "lavar los cuchillos". "desinfectarse las manos". Es mi impresión que estamos, los desarrolladores de software, en una época pre-Pasteuriana, y que inventamos reglas y proceso, "para tener uno a la mano", pero que nuestros resultados tienen poco que ver con el proceso inventado, y con que lo sigamos o no.

E) Deben crearse "campeones de la calidad", "comités de calidad", y otras organizaciones humanas cuya meta sea "promover la calidad." Generalmente, estos comités (1) elaboran normas que dicen cómo llevarse a cabo la confección de software, incluyendo formatos que ciertos documentos intermedios y finales (manuales, digamos) deben seguir, y (2) observan que el grupo productor se apegue a las normas (1).

Comentario 9. Estos comités, por no saber a ciencia cierta ni cómo medir la calidad del producto ni cómo alterar el proceso si la calidad de salida se observa baja, se convierten en rémoras y burocracias estereotipadas. Lo que pueden exigir (y lo hacen) es que el grupo de diseñadores y programadores se adhiera al proceso; si se adhieren, está bien ("saldrá de buena calidad el software"), si no se adhieren, está mal (se debe castigar a los infractores). Es equivalente a tener en un hospital pre-Pasteuriano (Comentario 8) a un comité que, al ver que en esta semana se murieron más pacientes de infección que en la semana pasada, redoblara sus esfuerzos y detectara a cirujanos que no le rezaron a San Dionisio, o que se colgaron collares de ajos no muy frescos. Regañemos a los infractores, así se morirán menos pacientes.

F) La actitud frente a la calidad debe permear a cada codificador. El diseñador o programador debe estar constantemente pensando en calidad, tener fé en que él hará trabajos de calidad, vigilará que la calidad de sus obras rebase un mínimo.

Comentario 10. Esto es un acto de fé, que puede no estorbar. Ayuda poco. La confección de software no debe estar basada en la fé o la creencia. Ciertamente, ayuda algo que un programador diga al principio de la jornada "hoy voy a programar con gran calidad, qué caray", así como un cirujano pre-Pasteuriano podría decir "hoy no se me va a morir ningún paciente, me comprometo". En cuanto a que el programador "vigilará la calidad de sus obras", es difícil, pues *es difícil medir la calidad del software,* incluyendo si el que mida es el que fabrica.

1.3 El mito de los estándares

G) Si hacemos las cosas de la manera que dicta un comité internacional, lo estamos haciendo bien. Así se asegura la calidad del proceso y del software resultante. Es decir, de los muchos procesos que podemos seguir, usemos uno que sea parte de una norma o estándar internacional, o que sea el procedimiento que sigue una empresa que produce software de calidad (Oracle, digamos).

Comentario 11. No veo mal en esto; es como si los cirujanos de un hospital con baja calidad del Comentario 8, decidieran copiar los procedimientos del Hospital Mariano, que tiene baja mortandad. Algo ha de ayudar. Empero, la subjetividad y posible poca relación entre estos procedimientos "estándar" y la calidad del software resultante debe quedar clara, según lo vertido en el Comentario 8.

2. MITOS Y CREENCIAS SOBRE LA CALIDAD DE LA ENSEÑANZA EN COMPUTACIÓN

2.1 Es suficiente medir el producto

H) Si yo hago un exámen o medición a dos recién egresados de licenciaturas en computación de distintas escuelas, y veo que uno sabe más cosas que el otro, claramente el que sabe más es de mejor calidad. Ejemplo: Ceneval.

Comentario 12. Esta medición "casi está bien", excepto que no toma en cuenta la *obsolescencia* tan grande que ocurre en nuestra profesión. Estimo que el tiempo de vida medio[2] de un concepto en computación es de 5 años. Es decir, cada cinco años, la mitad de lo que sabemos se vuelve inútil, no porque lo olvidemos o lo hayamos aprendido mal, sino porque simplemente ya no sirve, es obsoleto (memoria de burbujas, procesamiento por lotes, remote job entry...) La medición del punto H) simplemente mide la calidad *con respecto al día de hoy.* ¿Qué pasará dentro de cinco o diez años? Quizá uno de los egresados aún siga teniendo conocimientos útiles, en tanto que el otro ya no. Uno se oxidó más pronto que el otro. Es decir, la formación de un egresado (sobre todo en ciencias de alta obsolescencia, como la nuestra –lo es debida a su juventud, solo tiene 60 años de existir) depende de dos cosas: (a) los conocimientos básicos, teóricos, que le permitirán *continuar adquiriendo conocimientos* a través de su vida productiva, ya fuera

[2] El tiempo de vida medio de una substancia radioactiva es el tiempo en el cual la masa de esa substancia decae a la mitad de su masa original.

de la escuela; y (b) los conocimientos "de moda" (programación con objetos en el 2003, digamos) que le permitirán ser inmediatamente productivo, "salir y cortar caña." Se parece a la calidad de un machete, que depende de dos cosas: el *temple* del machete, que le permitirá reafilarse a lo largo de su vida productiva (y debemos de preparar egresados que tengan una vida productiva de 40 años), y el *filo* del machete, que le permite ser útil inmediatamente. Mi problema con el procedimiento H) es que solamente mide el filo. Sugiero tener mediciones cada cinco años, estudios longitudinales, para así evaluar también la componente del temple. Son los conocimientos teóricos o básicos (que no van a cambiar en 40 años) los que dan al egresado su resistencia a la obsolescencia.

I) La calidad de una escuela se mide por la calidad de sus egresados.

Comentario 13. De nuevo, esto "casi" es cierto. Ciertamente, tendemos a dar gran calidad a aquellas escuelas que producen egresados de alta calidad. Pero a menudo estas escuelas exigen que los alumnos que ingresan *ya tengan alta calidad.* Que vengan bien preparados de Preparatoria. Los selecciono con un examen de admisión. Y solo dejo pasar a los buenos. Es obvio que saldrán mejores que los egresados de otra escuela, que por tener grandes deficiencias cuando entran, saldrán menos preparados. Para ser justos, pues, la calidad de una escuela debe medirse por *el valor agregado* al estudiante, es decir, medir al estudiante a la entrada, medirlo a la salida, y hacer una resta.

2.2 Es suficiente medir el proceso de enseñanza

J) Buenos procesos de enseñanza producirán buenos egresados. Para conocer la calidad de un egresado, es suficiente medir el proceso de enseñanza.

Esto se parece al §1.2, "midamos el proceso en vez de medir el producto". De nuevo, la creencia es que es posible diseñar un proceso de enseñanza que asegure la calidad del egresado. "Que tenga prácticas de laboratorio". "Que los exámenes sean departamentales." La antigüedad del proceso educativo (los profesores vienen enseñando desde tiempos prehistóricos) garantiza que se tenga mucho más claro que en el §1.2 cuál es un buen proceso de enseñanza, y qué partes de él modificar si, digamos, los estudiantes no adquieren experiencia en Química experimental. Solo agregaría que, en procesos educativos (como en confección de alimentos, o cocina), también son importantes dos factores más. (a) los *ingredientes,* los libros, el software disponible, los conocimientos transmitidos, los planes de estudio; y (b) *los artesanos,* los cocineros, o sea, los maestros, los instructores.

Bibliografia

[1] y [2] son libros de ingeniería de software, sin referencia en el manuscrito..

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.
3. Phillip G. Armour. The business of software. Real work necessary friction, optional chaos. *Comm. ACM* **47,** 6, 15-18. It debunks the myth that it is possible to measure the effort of building a piece of software in staff-hours. It introduces "real work" (hours actually devoted to writing the code), "necessary friction" (hours devoted to learning something we need to know before writing a piece of code, related to 2OI, second order igno-

rance, “the things we don’t know we don’t know), and “optional chaos” (useless effort if programmers are pressured to produce results in a hurry). It explains why projects done in more time (calendar days) usually take less staff-hours: because “optional chaos” is reduced.