First, Do No Harm: A Hippocratic Oath for Software Developers?

Phillip A. Laplante, Penn State University

When asked about the Hippocratic Oath, most people are likely to recall the phrase, “First, do no harm.” It’s a logical response, as even those unfamiliar with the oath could figure out that avoiding additional injury in the course of treatment is critical. In fact, it’s natural to strive in any endeavor not to break something further in the course of repair. In software engineering, as in medicine, doing no harm starts with a deep understanding of the tools and techniques available. Using this theme and some medical metaphors, I offer some observations on the practice of software engineering.

FOUNDATIONS OF HEALTHY SKEPTICISM

The Hippocratic Oath was proffered by the Greek scholar, Hippocrates, the “Father of Medicine,” around 500 B.C., and since then it has guided the practice of medicine. Variations of the oath are sworn at nonmedical school graduations. For example, the “Nightingale Pledge” for nurses is an adaptation of the Hippocratic Oath and includes the phrase, “I will abstain from whatever is deleterious and mischievous, and will not take or knowingly administer any harmful drug.” A Hippocratic Oath for Scientists has been proposed, though it deals largely with the ethical issues of developing weapons. Some more “progressive” colleges and universities have students swear oaths to social responsibility.

Software engineering is a profession that frequently involves life-critical systems, yet nowhere have I found the equivalent of the Hippocratic Oath for software engineers. The IEEE Computer Society adopted a “Software Engineering Code of Ethics and Professional Practice,” but it relates to personal responsibility as opposed to the adoption of safe practices. It’s a shame, but a Web search for a “Hippocratic Oath for Software Engineers” yielded only a collection of jokes (for example, “Type fast, think slow” or “Never write a line of code that someone else can understand”).

“First, do no harm” makes a lot of sense in the practice of medicine, especially given its history. In prehistoric times, drilling holes in the head to release evil spirits (called trepanning) was an accepted treatment, and according to evidence from remains, some patients

What’s wrong

WITH TAKING OUR
PROFESSION A LITTLE
MORE SERIOUSLY?

actually survived. Physicians from medieval times through even the late 19th century would poke and prod with gruesome, dirty

instruments, attach leeches, perform blood-letting, and administer powerful drugs, such as mercury and laudanum, without understanding the side effects. These practices, though seemingly barbaric today, were undertaken with good intention and were in keeping with the state of the art. But the Hippocratic Oath at least made physicians aware that any new regimen always had the potential to injure. Consequently, many doctors chose (still choose) no treatment at all in lieu of one they didn’t understand completely, or at least waited until indisputable supporting scientific evidence appeared.

PROBING AROUND WITH DIRTY FINGERS

Software engineering procedures, like medical procedures, can be intrusive and destructive. Likewise, the tools and techniques that we use can be new and untested (or barely tested). Moreover, we don’t have the equivalent of medical licensing boards or the U.S. Food and Drug Administration (FDA) to regulate the practice of software engineering and the tools that we adopt. Thus, we sometimes subject our “patient”—the software—to unnecessarily risky procedures, without really understanding the risks.

What is the software engineering equivalent of poking and prodding with dirty fingers, bloodletting, trepanning, and lobotomizing? One example is code refactoring; though the intention of refactoring is noble and a certain amount is usually beneficial, even necessary, there is some point at which there are diminishing returns, even the risk of irreparable harm. The old saw, “If it ain’t broke, don’t fix it,” still holds merit. I think we sometimes go overboard trying to achieve the equivalent of a software extreme makeover.

In the course of fixing a problem we sometimes do more harm than good. In his software engineering classic, *Code Complete*, Steve McConnell opines that if you are not fixing the underlying source of a problem, and just

the symptom, then you're doing more harm than good in that you are deceiving yourself into thinking the problem has gone away. "If you don't thoroughly understand the problem, you're not fixing the code."¹ In fact, you may be transforming a relatively easy-to-find defect into a more insidious one that occurs much less frequently, and hence is harder to find.

The simplest example involves debugging statements. Imagine debugging an embedded realtime system. You add some kind of output statement to display some intermediate calculations (or you use a source-level debugger). Suddenly, the problem goes away. You remove the output statement (or turn off the source debugger) and the problem is back. After hunting and pecking around for the source of the problem, you give up and leave a dummy statement in place to overcome what you assume is a timing problem. Only a telltale comment remains, something akin to, "If you remove this code, the system doesn't work. I don't know why."

PLACEBOS AND PANACEAS

Throughout history, doctors have embraced quackery and offered placebos or harmful treatments simply because they didn't know any better. While I am not accusing anyone of deliberate misrepresentation, the relative merits of, for example, clean-room software development, pair programming, and other practices can have placebo-like or detrimental effects when misused. Even agile methodologies, while clearly useful in some settings, can lead to complacency in situations where they are not intended to be used. For example, it is always easier, in the name of agile development, to let a team self-organize, declare that the code is the primary artifact, and eschew documentation. But agile isn't appropriate in large mission-critical systems or in those with far-flung engineering teams. In other situations, it's really unclear whether agile is appropriate or not.

In *Code Complete*, McConnell warns of the equivalent of snake oil, "method-o-matic"—that is, the methodology du jour. He challenges us to skeptically ask, "How many systems have been actually built using method-o-matic?"² If we adopt method-o-matic and it works, we celebrate. When it doesn't work, we just say that it wasn't meant to be used in that context, anyway. It's a no-lose situation for those promoting method-o-matic.

BEWARE THE BLACK BOX

Historian and philosopher Thomas Carlyle said, "Nothing is more terrible than activity without insight." Has software engineering degraded into this state? In the

early days of computing, electrical engineers, mathematicians, and physicists programmed computers by adjusting circuits, programming in binary assembly language, and later using compilers that they understood intimately. These few skilled practitioners were often referred to as a "priesthood" because to the outside observer, they performed supernatural feats attributable only to the gods or magic. But to them, it wasn't magic. Everything they did was understood down to the minutest detail. I am not longing for the days of punch cards, but it seems that "back then" the "physicians" had a deep understanding of the remedies they employed.

From complex IDEs (integrated development environments) and frameworks, refactoring tools, and vast and mysterious code libraries to ready-baked architectures, software engineers are armed with a bag of tricks that most of us could not explain in meaningful detail. The age of reusable, abstract components has also become the age of black boxes and magic. I am not a Luddite, but my fear—based on observations of hundreds of practitioners—is that we adopt these aforementioned technologies without fully understanding what they do or how they do it. Thus, the chance of doing more harm than good when we use these technologies is ever present.

Why should we believe that the development, maintenance, and extension of software for anything but the most trivial systems should be easy? No one would ever contend that designing spacecraft is easy (it is, after all, "rocket science")—or even that designing modern automobiles is a breeze. So why do we assume that development of complex software systems should be any easier?

OUTSOURCING: DOING HARM IN A BIG WAY?

Doing harm while trying to do good can happen in the large, too. Those who contend that the price to be paid for easy-to-use, portable software is black-box reusability are condemning themselves to eventually outsourcing to low-cost competitors. The wide availability of GUI-based tools, easy-to-build Web solutions, and an inflated demand for Web programmers led to a whole generation of dot-com, barely-out-of-high-school whiz kids who could cobble components together and whip up ready-made solutions, without really knowing what they were doing. An inflated demand for IT professionals also helped to bloat salaries for even the most modestly prepared individuals.

Now, competitors in India, the former Soviet bloc, and elsewhere can just as easily use the same tools and

techniques at lower cost. So, managers outsource. But outsourcing can do more harm than good. Loss of control, loss of intellectual property, unsatisfactory performance, hidden costs, and hard-to-obtain legal remedies are the harm that can occur when things go wrong when projects are outsourced.

AN OATH FOR SOFTWARE ENGINEERS

It is widely believed that the single most important new medical procedure was hand washing. We need to adopt the equivalent of hand washing in software engineering practice, whether in the small, such as in the adoption of a new tool or practice, or in the large, such as in endeavoring to outsource. I think the software equivalent of the Hippocratic Oath can help. Let me take a stab at such an oath. It is a variant of the Nightingale Pledge for nurses:³

I solemnly pledge, first, to do no harm to the software entrusted to me; to not knowingly adopt any harmful practice, nor to adopt any practice or tool that I do not fully understand. With fervor, I promise to abstain from whatever is deleterious and mischievous. I will do all in my power to expand my skills and understanding, and will maintain and elevate the standard of my profession. With loyalty will I endeavor to aid the stakeholders, to hold in confidence all information that comes to my knowledge in the practice of my calling, and to devote myself to the welfare of the project committed to my care.

Perhaps this is too tame, and too long. People may dismiss it or ridicule it. Perhaps we need a punchy, mantra-like slogan that we can play over and over again like Muzak in software development houses. Or maybe a variation on the Star Spangled Banner, evoking patriotic visions of heroic coding deeds. Perhaps we need to create some epochal story of the trials and tribulations of the development of a major software system and have undergraduates memorize it, with portions to be recited at key moments in the software life cycle (e.g., "A move method refactoring, the guru applieth and when it was done, 'all is well,' he lieth"). Whatever, it's the purview of committees and organizations to evolve these things, not for some individual to decide.

I know, you say, "What is the point of any such oath? It's just a waste of time." My response is this: It says something that in the professions of medicine and nursing an oath of fidelity is important enough to recite at graduations, whereas the only semblance of an oath that software engineers have is a collection of jokes. Perhaps the symbolic act of adopting some kind of oath is a state-

ment that we want to take the practice of our profession more seriously.

More importantly, I remind you that the Hippocratic Oath is the basis for healthy skepticism. After all, the FDA exists largely to ensure that medical innovations don't do more harm than good. If nothing else, such an oath is a reminder to exercise caution in adopting new tools, methods, and practices. But if after all this I still haven't convinced you, be aware that such luminaries as Edsger Dijkstra⁴ and Steve McConnell,⁵ among many others, have suggested the adoption of a Hippocratic Oath for software.

Whatever we do, first, do no harm. After all, we do not want critics 100 years from now ridiculing practices that we now believe to be legitimate, especially if the only reason we adopt them is faith and not deep understanding. Q

REFERENCES

1. McConnell, S. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, Redmond: WA, 1993.
2. See reference 1.
3. Florence Nightingale Pledge. Nursing Network: see <http://www.nursingnetwork.com/florencepledge.htm>.
4. Dijkstra, E. The end of computing science? *Communications of the ACM* 44, 3 (March, 2001), 92.
5. McConnell, S. *After the Gold Rush: Creating a True Profession of Software Engineering*, Microsoft Press, Redmond: WA, 1999.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

PHILLIP A. LAPLANTE, Ph.D., is associate professor of software engineering at the Penn State Great Valley School of Graduate Studies. His research interests include realtime and embedded systems, image processing, and software requirements engineering. He has written numerous papers, 17 books, and cofounded the journal, *Real-Time Imaging*. He edits the CRC Press Series on image processing and is on the editorial boards of four journals. Laplante received his B.S. in computer science, M.Eng. in electrical engineering, and Ph.D. in computer science from Stevens Institute of Technology, and an M.B.A. from the University of Colorado. He is a senior member of the IEEE, a member of ACM and the International Society for Optical Engineering (SPIE), and a registered professional engineer in Pennsylvania.

© 2004 ACM 1542-7730/04/0600 \$5.00