



# Good Enough

Jeffrey Voas , IEEE Fellow

*The computer science community has always faced the challenge of what is “good enough” software. The challenge remains unsolved and will remain that way under the prevailing mindset of a quantitative formula. This message offers a more pragmatic approach.*

“**G**ood enough” is two words. We understand good versus bad. Good versus evil. We teach this to our kids.

Growing up, “good enough” usually meant that “the fix” that we were trying to implement was completed and we hoped the fix was sufficient. In those days, I was working on physical systems, for example, lawnmowers and bicycles. But how about today’s nonphysical systems? What is “good enough” for them? And do we understand “enough”?

“Enough” implies “amount” and “time”: for example, “time has expired” at a parking meter. So how do we determine “good enough” for something that we cannot touch? This is the conundrum.

Physical and nonphysical systems are hard to compare from a trustworthiness standpoint—it is essentially

an apples-and-oranges problem. The software reliability community has experienced this for decades because of numerous attempts to apply hardware reliability models (that account for physical fatigue and wear-out over time) to nonphysical software.

Hardware is usually mass produced. “Good enough” for mass-produced hardware is different from that for a singular nonphysical software product. For mass-produced products, sampling and testing a handful of items coming off an assembly line are traditional. But for software, your sample size is one, and it’s your current version.

So how do you determine “good enough” for a singular virtual product? Beauty is supposedly in the eye of the beholder. That’s subjective. Is “good enough software” in the eye of the beholder, or can we start to create plausible algorithms and processes (or maybe a single equation) to quantify “good enough”?

After more than 30 years of thinking about this problem, I still believe that we must assess virtual function

## DISCLAIMER

The authors are completely responsible for the content in this message. The opinions expressed here are their own.

quality from a behavioral standpoint. To me, a software product's behavior should be viewed as a cross-product between the required "ilities" and its environment. (It seems straightforward to include environments here since it is the environment that causes hardware fatigue and failure.) But what about the "ilities"? How do they fit in? (It is unclear how many "ilities" there are, though examples include availability,

Let's assume I1 represents reliability, and the software is unreliable. That slice will then be small or totally missing. If we were to do this for all 10, the chart offers a quick visual indicator of the current "evidence of trustworthiness." This is only a statement about the evidence of trustworthiness—it is not a statement about the confidence in actual trustworthiness. And recognize that you cannot size

The unsolved research problem is how to quantitatively or nonmathematically estimate the size of the slices.


composability, compatibility, dependability, discoverability, durability, fault tolerance, flexibility, interoperability, insurability, liability, maintainability, observability, privacy, performance, portability, predictability, probability of failure, readability, reliability, resilience, reachability, safety, scalability, cybersecurity, sustainability, testability, traceability, usability, visibility, and vulnerability.<sup>1</sup>) So here, I propose an idea of color-coding "trustworthiness." The idea is that we want some evidence of trustworthiness knowing that we'll never get it all.

So, for example, in my view of a potential approach, let's assume we have 10 "ilities" of importance: {I1, I2, I3, .... I10}. Now, visualize a pie chart with 10 equal slices. Each slice represents an "ility," and each has a different color.

a slice without consideration of the environment.

I know that this is not easy. The unsolved research problem is how to quantitatively or nonmathematically estimate the size of the slices. This is essentially a trustworthiness dashboard. And while this is not precise, so long as the methods to estimate the pie slices are consistent, the methods need not be perfect. Anyone should be able to look and see what slices are missing (or minimal) to get a "warm fuzzy" feeling. So, for example, if 99% of the pie is white space, I'd probably run away from the product. Warm fuzzy feelings are not optimal, but given the difficulty in defining "ilities," they may be a last resort.

The bottom line is that we've spent decades running away from certification or any notion of warranties of

"good enough software."<sup>2,3,4,5</sup> The top-line question is: Will we ever quit running, or is software failure an expectation and not an exception? 

### REFERENCES

1. J. Voas, "Software's secret sauce: The 'ilities' [Software Quality]," *IEEE Softw.*, vol. 21, no. 6, pp. 14–15, Nov./Dec. 2004, doi: 10.1109/MS.2004.54.
2. J. Voas, "Software quality's eight greatest myths," *IEEE Softw.*, vol. 16, no. 5, pp. 118–120, Sep./Oct. 1999, doi: 10.1109/52.795111.
3. D. R. Kuhn, I. Dominguez, R. N. Kacker, and Y. Lei, "Combinatorial coverage measurement concepts and applications," in *Proc. IEEE 6th Int. Conf. Softw. Testing, Verification Validation Workshops (IWCT)*, Luxembourg City, Luxembourg, Mar. 2013, pp. 352–361, doi: 10.1109/ICSTW.2013.77.
4. J. Voas, F. Charron, G. McGraw, K. Miller, and M. Friedman, "Predicting how badly 'Good' software can behave," *IEEE Softw.*, vol. 14, no. 4, pp. 73–83, Jul./Aug. 1997, doi: 10.1109/52.595959.
5. J. Voas and P. A. Laplante, "IoT's certification quagmire," *Computer*, vol. 51, no. 4, pp. 86–89, Apr. 2018, doi: 10.1109/MC.2018.2141036.

**JEFFREY VOAS**, Gaithersburg, MD 20899 USA, is the editor in chief of *Computer*. He is a Fellow of IEEE, IET, AAAS, and the Washington Academy of Sciences. Contact him at [j.voas@ieee.org](mailto:j.voas@ieee.org).