

# The Software Engineering Silver Bullet Conundrum

Daniel M. Berry (dberry@uwaterloo.ca)  
Cheriton School of Computer Science, University of Waterloo  
Waterloo, Ontario N2L 3G1, Canada

In 1986, in the famous “No Silver Bullet” paper, Fred Brooks predicted, based on his experience, that “There is no single development, in either technology nor management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.” That is, he predicted that in the next 10 years no software development silver bullet will be found. In arguing for his claim he added “I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to conceptual errors in most systems. If this is true, building software will always be hard. There is inherently no silver bullet.” The conceptual errors he was talking about were those of failing to capture the essence of the system being built, the system’s conceptual construct, the system’s requirements.

While we have yet to find a sure-fire way to understand a system’s requirements, we have over the years made significant technological improvements that have combined to improve the production of software by more than an order of magnitude. These technological improvements include, high-level language compilers, configuration managers, testing tools and harnesses, debugging tools, graphical user interface (GUI) builders, etc. In other words, we have come up with many technological aluminum bullets that have combined to be almost a silver bullet while no bullet itself is silver.

In 2002, I went further and claimed that there would not ever be a silver bullet unless a technology were able to deal with not only the essence of software systems, their requirements, but also the relentless changes to these requirements. In my “Inevitable Pain of Software Development: Why There is No Silver Bullet” paper, I argued that in fact, the typical software development method is effective in the first application of it to any system development problem. However, once a version has been built with the method and an inevitable change to the requirements comes along, from client and user demand, the modification of the method’s documenting artifacts is so painful that doing the changes the right way, according to the method’s careful tracking of the effects of a change, gets put off in favor of the quick patch that increases the brittleness of the system.

Since writing this paper, I have come to the realization that there are two more fundamental reasons that we will never have a software development silver bullet.

1. A silver bullet kills itself and ceases to be a silver bullet simply because once we have a silver bullet, we will quickly solve all the formerly too tough problems that the silver bullet allows us to solve. Doing so bring us to a new frontier of not easily solved problems. Then, due to our very human ambition to advance, we begin to try to solve problems that are just beyond the frontier; the silver bullet has become an ordinary lead bullet with respect to these new problems.
2. The cause of the inevitable pain is the very act of writing something formal, so that it will be implemented. Once we have written that formal specification, we are stuck. Note that even if we do not write anything traditionally called a formal specification, we do eventually write executable code, and this code is a formal specification. Any subsequent change in requirements requires changing the specification in a way that preserves correctness. Therefore we have pain. Changing a specification consistently is painful; redoing a specification from scratch is painful; and deciding which of the two pains to endure is painful. This pain happens *even* if we use a silver bullet, which is about to convert itself into a lead bullet. The only way to avoid the pain is to not write any specification. However, then we get *no* implementation, unless we build a machine that reads our minds, intelligently fills in *all* details, and does what I mean (DWIM), an impossibility.

In conclusion, I believe that no software engineering bullet can exist in silver form for more than an instant. That each software engineering silver bullet very quickly becomes an ordinary lead bullet is the basic conundrum of software engineering silver bullets. Does this conclusion mean that we should stop trying to improve software engineering? No! However, we need to stop the search for silver bullets and to focus on finding lightweight aluminum bullets. Moreover, we need to stop pouncing on good bullets that we do find and hyping them as silver bullets, which will solve more problems than they actually can.