I am the silver bullet

As the article describes, people have sought a silver bullet for software engineering. This silver bullet is supposed to help software engineering reach the efficiency of hardware engineering. Whereas hardware is continually getting faster and cheaper, software is still plagued by budget and time overruns. While this article was written nearly two decades ago, it could have been written two weeks ago. Consequently, it would appear that no silver bullet has been found yet.

Fredrick Brooks Jr. discusses several essences of software engineering that I believe are even more significant now then when he wrote his article. The first is software changes. Brooks states that end users often push the limit of the software and consequently request new features that go beyond what is currently possible. After developing applications for OTS, this statement is completely true. End users assume that because they can do tasks A, B and C that they should also be able to do D, E and F regardless of the software was originally intended to do. Also, instead of changing how an end user uses an application, they expect the application to conform to their individual expectations. The second is software invisibility. I think invisibility has the greatest effect on the end user. To the end user, the application is just a tool. They have no appreciation for the engineering behind application. Several times I have encountered end users who make a simple statement like why can’t this perform like this. Because end users don’t see or understand how something works, they assume that making changes to it is no big deal. I think that changeability and invisibility work together to doom software engineering. If end users, and people other than programmers, could learn to appreciate that a simple change is never simple then budgets and timelines would be better developed to allow for “simple” changes. And this is not entirely the fault of the end users. Programmers by nature are problem solvers and by nature present a product that hides everything that is not essential for the end user to know. But hiding all the details and inner workings inadvertently give the impression to end users that behind the scenes everything is as simple as the user interface they use.

Brooks also examines several potential silver bullets. One of these was high level language. In my opinion, high level languages have already made their contribution to improving software engineering. In other words, no new true gains can be achieved by developing a new high level language. In comparing high level to low level languages, I came across an interesting analogy. With high level languages, a programmer tells the computer what to do. With low level languages, a programmer tells the computer what to do and how to do it. In my mind, there is nothing currently better than telling it what to do. Perhaps someday, as Brooks suggests, a programmer can simply tell a computer what the problem is and it can solve it. If that is ever possible, that very well may be the silver bullet. However, I don’t think it will happen in the near future.

Brooks points out another interesting trend – that more software is bought off the shelf than developed. He points out that originally people thought that off the shelf software could never work for their specific needs. However, people learned to modify their needs and procedures to make off the shelf software work. While this is certainly true, it does not provide the entire picture. For example, Datatel was purchased off the shelf (so to speak) yet countless hours have gone into making it work for UE. While Datatel provides a basis, it does not meet every need. Therefore, even off the shelf software still needs to be developed and refined. And this brings software engineering full circle. Once Datatel, for example, was implemented, end
users pushed the boundaries and wanted more features. End users also wanted it to work like the old system worked. Because the software changed, people expect it to continue to change to meet their needs.

In conclusion, I don’t think there is a silver bullet yet. I don’t think there will ever be a silver bullet. In the near future, the closest thing to a silver bullet will be education. If end users and other non programmers can learn to understand what all really happens behind the scenes and what it actually takes to bring an application together then I think a greater appreciation will develop for what exists. When programmers and end users can relate to what the other is doing then everyone will understand what needs to happen and how it will happen. Personally, I did not truly understand how to design applications for OTS until I went and spent time doing the jobs that my end users would be using my applications for. As someone once said, knowledge is power. Therefore, knowledge and education are the only true silver bullets.