My Response to “No Silver Bullet: Essence and Accidents of Software Engineering”

In “No Silver Bullet”, Fredrick Brooks asserts that Software Engineering is a fundamentally difficult task: that, although some of the difficulties of software engineering can be eliminated with better programming environments, better programming languages, and better paradigms, the fundamental difficulties of software engineering are inherent to the field. The above distinction, I believe, can be likened to that of optimizing an algorithm: an inherently-hard polynomial-time task can perhaps be optimized in the sense of reducing the constants in front of the big-Oh notation, but the optimization will never put the algorithm into a different asymptotic family.

To me, as a programmer, it comes as no surprise that programming is hard! Like the author, I believe that the difficulty of programming lies in the complex interactions within the system. A human mind can only attend to a limited amount of items at a time, and even “small” software often far exceeds the human attention capacity. Moreover, people are not well-equipped to visualize such complex interactions, and, even when they do grasp the overall outline of the program, they have a hard time imagining how their software might integrate with other software, or how to explain this complex and convoluted web of thought to somebody else.

Though the above problems are inherent to the very essence of programming, other problems (like foolish mistakes, or even some higher-level difficulties) can be eliminated with better tools and languages. For example, having programmed in Visual Basic, Java, Objective-C, and C++, and Assembly, I definitely feel that some languages are more conducive than others
for crafting will-written and easily-readable programs. Likewise, having used various
programming environments and debugging tools, from notepad to Visual Studio to Eclipse, I
recognize that productivity can be boosted significantly with the magic of auto-complete, real-
time code errors analysis, and visual debugging. Even so, while better tools and languages can
increase programming productivity, they are inadequate at combating the one true challenge:
namely, actually designing the intricate and inter-connected underlying structure of the software.

Brooks suggests that that a silver bullet to solve all software engineering problems is
unrealistic: and, indeed, no silver bullet has appeared in the 20+ years since the writing of the
article, and nor do I expect that one can ever be found. To this day, software continues to be
released woefully late, woefully buggy, and woefully incompatible (an example of all three can
be found in Microsoft Vista / Office 2007). Yet at the same time, I agree with Brooks’
suggestion that the combined forces of better languages, object-oriented paradigms, “intelligent”
programming environments, automated testing tools, and so forth can (and have) increase(d)
productivity tenfold, if not more! Eclipse does not let me write flawless code, fast computers
don’t guarantee perfect execution, and visual debugging tools can’t ensure that my code is
entirely bug-free -- but all three certainly do help! In short, I believe that an array of regular
bullets, though not as desirable as one magnificent silver bullet, is still better than facing
software engineering completely unarmed!