Software Development Amidst the Whiz of Silver Bullets...

Alex E. Bell, The Boeing Company

There are plenty of examples in the software engineering realm that demonstrate blatant disregard for Fred Brooks's sage advice¹ asserting that there are no silver bullets available now or in the foreseeable future with which to solve all difficulties. Regardless, the desperate, the pressured, and the ignorant are among those who continue to worship the silver-bullet gods and plead for continuance of silver-fueled delusions that are keeping many of their projects alive. It is difficult to be overly critical of the individuals who have been impacted by silver bullets, however, because the software engineering space is being barraged with them as never before. Even the most savvy of software engineers must occasionally liken themselves to the infamous Neo in the film *The Matrix* and gyrate wildly to avoid being stricken by the many bullets whizzing by.

Veterans of the software industry will attest to having seen a number of silver bullets come and go in their time. The argentum² projectiles of yesteryear, such as OO, high-level software languages, and IDEs, are now obvious to have been only low-grade alloys compared with the fine silver being discharged today. For example, today's silver bullets have demonstrated an unparalleled ability to provide implicit value to both text and diagrams, the power to shift the economics of software development, and a capacity to change the focus of long-established engineering disciplines. Only the passage of time will reveal the new and amazing capabilities promised of future silver bullets yet to whiz by.

I was recently reviewing a software design document that correctly paid much-needed attention to the objective of supporting configurable behavior. As opposed to simply documenting how the design would accommodate said configurability, the design description also included the following commanding statement a number of times: "The configuration data will be stored in XML." I immediately thought to myself, "Wow, this design must *really* be good" and was relieved from previous concerns that some form of ancient Egyptian hieroglyph may have instead been the representation of choice.

Still reeling from the powerful implications of what I had read, I began to wonder if placement of *any* content

A call to

TRANSYLVANIA

MAY BE NEEDED.

in the context of XML would somehow lead people to believe it to be of hallowed or divine origin, and having some implicit

warranty of accuracy or correctness. I decided to test this premise and composed an e-mail to my 12-year-old daughter, hoping to sway her on an opinion she has been very unwilling to yield upon many times in the past:

<sanctified_declaration>

<addressee> Alanah </addressee>

<message> Hi Sweetie, I really am not the weirdest Dad of all the kids in your school.

Love, Dad.

</message>

</sanctified_declaration>

I was optimistic that if she were to read this declaration within the context of XML, our long-standing dispute would finally get some resolution. Unfortunately, things did not work out as I had hoped, and my ploy served only to reinforce her unyielding position.

XML is not the only silver bullet in the software engineering space to which sanctity seems to be automatically attached by simply using a particular technology. The fact that a diagram has been created using the UML leads some to believe that the associated design is guaranteed to be implementable and ready for development, even if the laws of physics were ignored as constraints. Had my daughter not already convincingly dashed the notion of implicit technological sanctity, I would be tempted to create a "yoo-mel"³ sequence diagram for my wife detailing the steps I plan to take someday to be more sensitive, a better listener, and less of a slob. If she were to see these steps captured in UML, would she be more inclined to believe the sincerity of my intentions?

Continued on page 62

Continued from page 64

No discourse on silver bulletry would be replete without recognizing a current favorite of software management. “That’s just a Web service...” is a phrase often spoken so convincingly that it is hard not to believe that Web services are available, or soon will be, to solve all difficult engineering problems. Web services are already available that provide the stock price for any ticker symbol or the zip code for any city! Considerations for strict temporal determinism, minimum availability thresholds, or the fact that a service should not even have to endure the constraints of squirreling through HTTP/HTTPS ports are unimportant amidst the whiz of silver bullets.

It would be unfair to suggest that only management is caught up in the Web services silver-bullet parade because some developers are marching right along with them. Although vulnerability to error and productivity impacts of working directly with XML led to the innovation of technologies such as WSDL with which to improve the “face” of Web services, some developers feel that their usage impacts the open-endedness and flexibility made possible by method signatures of the ilk:

```
XMLResult DoAnything (XMLDoc Arguments)
{
.....
}
```

Additional benefits of methods with signatures such as the one above include freeing designers from the burdens of time-consuming negotiation with prospective clients and not being bothered with recompilation issues that typically accompany modification of more strongly typed interfaces. It would serve as great justice if the providers of methods such as `DoAnything()` also assumed their associated liabilities. The unfortunate reality, however, is that the method’s *users* have to absorb the liabilities in the form of increased testing and integration costs as the result of its misuse being virtually undetectable at compile time. In response to suggestions that a `DoAnything()` method be redesigned to take advantage of strong typing, it is not uncommon to hear its designers assert something of the sort, “An XML document *is* a strong type; let’s see you try to use an integer argument in its place!”

<utterance_in_a_state_of_speechlessness>

Gasp.

</utterance_in_a_state_of_speechlessness>

The challenges of software development are difficult enough without also having to endure the ricochet of silver bullets strafing supporting organizations. For example, many systems engineers have discovered that using the UML greatly simplifies the efforts that their predecessors unnecessarily struggled with in pre-UML days. As opposed to having to devote significant efforts to the consideration of constraints such as network bandwidth, processor speeds, and the speed of light when developing system architectures, such annoyances are now overcome by creating reams of UML diagrams containing very vague entities and equally vague navigation between them. Unfortunately for software engineers, the UML silver bullets whizzing through the systems engineering space translate into much more work for them.

I have a very vivid mental picture of this situation that is related to one of my all-time favorite TV commercials. While sitting in a police station, an apparent crime victim is providing a detailed description of the alleged perpetrator to a police artist who appears to be busily sketching a corresponding likeness on an unseen tablet. After some period of time, the artist believes that the likeness he has captured is sufficiently complete for the victim to assess its accuracy. Amidst a rising crescendo of expectation, the artist reveals his drawing to the victim, and it contains only a human stick figure that would be considered primitive even by kindergarten standards. All of the detail is missing from the sketch, just as in use cases such as `CreateWorldPeace` or `IncreaseTheSpeedOfLight` being delivered to software engineering organizations for development. Unfortunately, the UML mind-melds required for such use cases to have any value are usually omitted and dismissed as implementation detail.

Silver bullets of the past and present share a number of common properties that will also likely apply to those only taking shape now in the mental foundries of the fantasy-minded. They defy the laws of physics, they are not bounded by cost, they are not constrained by time, and they seem to rob otherwise intelligent people of their common sense. Their usage is typically accompanied by shifting engineering efforts to later in the software life cycle, shirking responsibilities to other organizations, and blatant denial. Probably the most consistent property associated with silver bullets is that the people who promote or endorse them have generally never been software developers nor have they directly contributed to the delivery of a successful program. Still one more timeless eventuality associated with silver bullets is that the software developers responsible for actually implementing argentum-fueled visions are usually left holding the

bag with a bunch of blanks.

It is ironic that a reverence for silver bullets usually brings more pain to software organizations than if they had simply rolled up their sleeves, focused on traditional engineering activities, and accepted that there is no such thing as a free lunch. Postponement of effort, production of useless artifacts, and illusion of progress are just a few of the factors that result in big price tags when reality eventually trumps denial. Even in the absence of Fred Brooks's well-known admonitions, it is startling that the failure of past silver bullets to hit their targets is not enough to make us very wary of them today.

Barring any events similar to the Hunt brothers' failed attempt at cornering the silver market in 1980, silver futures appear to be bullish. The supply of fine-grade silver required to manufacture the next generation of silver bullets is expected to meet demand, so the sound of their whiz will not subside anytime soon. Software organizations must accept that silver bullets will be a part of their future and must prepare strategies to defend against them rather than assume their demise. The only plausible defense strategy against silver bullets that I have been able to conjure up is to take advantage of an engineering staff that has a natural affinity for eluding these projectiles and one with an innate terror of them. Assembling such a staff, however, would involve an undertaking that is currently quite controversial: outsourcing. As opposed

to China or India, however, my outsourcing plan would focus on a small town in Romania—for it is only in Transylvania where the werewolf can be hired to work with an unrivaled vigilance to avoid the whiz of silver bullets. Q

Author's note: Because *Queue* did not offer this entire issue in which I could rant and rave about *all* silver bullets, I apologize for failing to provide honorable metallic mention to those whizzing by that were not specifically recognized.

REFERENCES

1. Brooks Jr., F. P. 1987. No silver bullet: Essence and accidents of software engineering. *Computer Magazine* 20 (4): 10-19.
2. The Latin word for silver and the basis of the periodic table symbol: Ag.
3. Said in a hillbilly drawl, a euphemism for insane usage of the UML.

LOVE IT, HATE IT? LET US KNOW

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

ALEX E. BELL is a software architect with The Boeing Company and author of "Death by UML Fever" (*ACM Queue* 2(1): 72-80).

© 2006 ACM 1542-7730/06/0600 \$5.00

acm
queue
architecting tomorrow's computing

Coming next month in *Queue*

**Beyond mouse
and keyboard—
new trends in HCI**

- Communicating with context
- Speech recognition
- Perceptual interfaces