No Silver Bullet: Essence and Accidents of Software Engineering

By Frederick P. Brooks, Jr.

Reviewer: Morten Christiansen
Frederick Phillips Brooks, Jr.

Ph.D. in Applied Mathematics (Computer Science) at Harvard
Managed development of OS/360
Author of The Mythical Man-Month

Founded and chaired Dep. Of CS at the University of North Carolina for 20 years

Turing award in 1999 (amongst others)
"It is a very humbling experience to make a multi-million-dollar mistake, but it is also very memorable."
The article


The software crisis

Has spawned much discussion, even today

Why I chose this article
Main message

Programming is hard!
The silver bullet and the quest to finding it

Example: Atira
No Moore's law in software

Essential and accidental difficulties

Skepticism is not pessimism, there are still good things
Essence of modern software systems 1/2

Complexity
- No two things are the same
- Scaling is non-linear
- Abstracting away complexity abstracts away the essence
- Management issues (training, communication)

Conformity
- No unifying principles
- Arbitrary complexity from interfaces
- Software must conform because it is the most flexible
Essence of modern software systems 2/2

Changeability
Software embodies function, function changes
Software is easier to modify than physical objects
Software can survive its original platform, usage, culture

Invisibility
Unvisualizable (Has no geometric representations)
Cross cutting concerns
Lack of visual abstractions makes thinking about and communicating ideas harder
Past breakthroughs solved accidental difficulties

High level languages
Time sharing
Unified programming environments

Summed up: "The hard thing about building software is deciding what one wants to say, not saying it. No facilitation of expression can give more than marginal gains."

We haven't seen any progress as great as these since and it's hard to image how it would happen
Potential silver 1/2

Ada and other high-level advances
  No silver bullet but good for training modern design practices
OOP - Promising
  Easier to express the essence of the system
  Does not attack the essence itself
Artificial intelligence
  Automating tasks – Domain specific
  Expert systems – Promising but requires expert
  Separate application complexity from the application
  Give new developers the experience of old developers
Potential silver 2/2

Automatic programming
   Euphemism for programming with higher level language than available
Graphical programming – Not interesting, hard to visualize
Program verification
   Makes no guarantees
   Meeting the spec does not make the spec easy to get
Environments and tools – Not much more to get
   The biggest benefit of file management has been solved
Workstations
   More MIPS does not give more productivity
Promising attacks on the conceptual essence

Buy vs. build
The market is becoming large enough
Requirements refinement and rapid prototyping
  Turned into agile (A good bet)
  Helps decide what to build
Great designers
  Find the talents and train them
The metaphors of programming

Writing software
Building software

Other parts of the metaphor: Specifications, Assembly of components, and Scaffolding

Growing software

Sums up the current approach to software development
Other views


Agrees with the conclusion that there is no silver bullet

All potential bullets have a painful part which prevents it from being pure silver
Breakthroughs and potential silver bullets in the last 20 years

Ruby on Rails ("10-fold improvement" - The Atira people)
Agile development
Visual Studio
Language Integration (.NET)
Aspect oriented programming
Frameworks
They are still only incremental improvements
The future of language design

If there is no silver bullet, what does this mean for language design?

Diminishing returns on productivity gains of new languages? (As we remove accidental difficulties there is less and less room for improvement)

There will always be the need for creative minds to solve the problems of tomorrow? (We can't ever automate the task of software development)
My opinion of the article

Serious, well written and objective

I generally agree with the article and the statement that there is no silver bullet (I have seen no breakthroughs yet)
No silver bullet: So what?

Do we want a silver bullet?

Daniel M. Barry sees it as a positive

"It says that software engineering can never be automated, that it will always require thinking, creative, human beings. Therefore, we programmers are always assured of jobs!"

The lack of a silver bullet suggests that software engineering might, after all, be an art

Would a silver bullet be fun?