Panel

"No Silver Bullet" Reloaded – A Retrospective on "Essence and Accidents of Software Engineering"

Steven D. Fraser
Director (Engineering)
Cisco Research Center
Cisco Systems, San Jose

Frederick P. Brooks, Jr.
Kenan Professor
Dept of Computer Science
UNC, Chapel Hill

Martin Fowler
Chief Scientist,
ThoughtWorks
Boston

Ricardo Lopez
Principal Engineer
Qualcomm
San Diego

Aki Namioka
Development Manager
Cisco Systems
Seattle

Linda Northrop
Director
Product Line Systems
SEI (CMU), Pittsburgh

David Lorge Parnas
Director, Software
Quality Research Lab
University of Limerick

David Thomas
Founder
Bedarra Labs
Ottawa

Abstract

Twenty years after the paper *No Silver Bullet: Essence and Accidents of Software Engineering* by Frederick P. Brooks first appeared in *IEEE Computer* in April 1987 (following its 1986 publication in *Information Processing,* ISBN 0444-7077-3) does the premise hold that the complexity of software is not accidental? How have the "hopes for silver" which included high-level language advances, object-oriented programming, artificial intelligence, expert systems, great designers, etc. – evolved? Panelists will discuss what has changed and/or stayed the same in the past twenty years – and the paper's influence on the community.

Categories & Subject Descriptors:
D.2.9 Management
H.4 Information Technology and Systems
K.0 Computing Milieux
K.4.3 Organizational Impacts

General Terms: Design, Management

Keywords: Complexity, Silver Bullet, Design, Software

1. Steven Fraser *(panel impresario),* sdfraser@acm.org

STEVEN FRASER recently joined the Cisco Research Center in San Jose California as a Director (Engineering) with responsibilities for developing and managing university research collaborations. Previously, Steven was a member of Qualcomm's Learning Center in San Diego, California with responsibilities for technical learning and development and creating the corporation's internal technical conference – the QTech Forum. Steven also held a variety of technology management roles at Nortel/NT/BNR including: Process Architect, Senior Manager (Disruptive Technology and Global External Research), and Design Process Engineering Advisor. In 1994 he spent a year as a Visiting Scientist at the Software Engineering Institute (SEI) collaborating with the Application of Software Models project on the development of team-based domain analysis (software reuse) techniques. Fraser is the Corporate Support Chair for OOPSLA'07 and was the General Chair for XP2006. Fraser holds a doctorate in EE from McGill University in Montréal – and is a member of the ACM and a senior member of the IEEE. The *IEEE Computer* version of the paper "*No Silver Bullet*" memorably appeared the morning of his doctoral defense.

Fred Brooks' seminal "No Silver Bullet" paper – often referenced simply by its initials "NSB" – begins with the chilling introduction:

> *"Of all the monsters that fill the nightmares of our folk- lore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest…"*

Panelists will discuss what "werewolves" remain, as systems grow in complexity and age – and what "hopes for silver" have evolved or have been laid to rest.

2. Frederick P. Brooks, Jr. *brooks@cs.unc.edu*

FREDERICK P. BROOKS, JR. received an A.B. summa cum laude in physics from Duke and a Ph.D. in computer science from Harvard. He joined IBM, working in Poughkeepsie and Yorktown, NY, 1956-1965. He was an architect of the Stretch and Harvest computers and later the project manager for the development of IBM's System/360 family of computers. For this work he received a National Medal of Technology jointly with Bob O. Evans and Erich Bloch Brooks. With Dura Sweeney in 1957 – Brooks patented an interrupt system for the IBM Stretch computer that introduced most features of today's interrupt systems. He coined the term computer architecture. His System/360 team first achieved strict compatibility, upward and downward, in a computer family. His early concern for word processing led to his selection of the 8-bit byte and the lowercase alphabet for the System/360, engineering of many new 8-bit input/output devices, and introduction of a character-string data type in the PL/I programming language.

In 1964 he founded the Computer Science Department at the University of North Carolina at Chapel Hill and chaired it for 20 years. Currently, he is Kenan Professor of Com-

OOPSLA'07 October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

puter Science. His principal research is in real-time, three-dimensional, computer graphics—"virtual reality." Brooks distilled the successes and failures of the development of Operating System/360 in *The Mythical Man-Month: Essays in Software Engineering*, (1975, 20th Anniversary Edition, 1995). He further examined software engineering in a 1986 paper, *No Silver Bullet.* He is a member of the National Academy of Engineering, the National Academy of Science, and the American Academy of Arts and Sciences. He has received the ACM A.M. Turing Award, the IEEE John von Neumann Medal, the IEEE Computer Society's McDowell and Computer Pioneer Awards, the ACM Allen Newell and Distinguished Service Awards, the AFIPS Harry Goode Award, and an honorary Doctor of Technical Science from the Swiss Federal Institute of Technology (ETH-Zürich).

"No Silver Bullet" argues that:

1. The difficulties of making big software systems consist of **essential** difficulties and **accidental** (or incidental) difficulties.
2. The great leaps of progress in the past were accomplished by eliminating accidental difficulties, e.g., by high-level languages, time-sharing, and workstations.
3. Of the remaining difficulties, at least half seem to me to be essential, the very inherent complexity of what we build.
4. Therefore, no attack on accidental difficulties can bring an order-of-magnitude improvement—indeed, more than a factor of 2.
5. And yet, most of the proposed "radical improvements" proposed continue to address only accidental difficulties.

The soft spot in this argument is the estimate that at least half of our remaining trouble is essential. In the many challenges to the paper that have been published, none have challenged this point, even though it would seem easy to do so if one could enumerate a lot of particular accidental difficulties remaining.

The 1986 paper predicted that there would be "no order-of-magnitude improvement within a decade in software productivity, reliability, or simplicity." 1996 has come and gone. 2006 has come and gone. Few have stepped forward to claim such an improvement. In the 1995 Anniversary Edition of *The Mythical Man-Month,* Chapter 17 discusses progress in the first decade after "NSB". What has happened in the next decade?

Of the candidates enumerated in "NSB", object-oriented programming has made the biggest change, and it is a real attack on the inherent complexity itself. The most promising attack continues to be re-use, and re-use of COTS programs in particular. And we see much progress there; a 2004 paper by Boehm, Brown, Basili, and Turner describes the substantial difficulties remaining. It remains true that "The most radical possible solution for constructing software is not to construct it at all."

3. Martin Fowler, fowler@acm.org

MARTIN FOWLER is an author, speaker, consultant, and general loud-mouth on software development. He concentrates on the issues of designing enterprise software, in particular: object-oriented technology, refactoring, patterns, agile methodologies, domain modeling, the Unified Modeling Language, and Extreme Programming. He has written five books and writes regularly at *martinfowler.com*. Martin was born in Walsall, England and lived in London for a decade before coming to New England were he has enjoyed living in the US even though he misses the beer, the deep pointlessness of Cricket, and the English countryside.

No Silver Bullet came out not long after I graduated. It was an extremely influential paper on me, in no small part due to the big influence it had on my early mentors. The term *silver bullet* has entered our industry's vocabulary, although in my experience it's never something directly claimed. It's common for people to criticize a technology on the basis that it isn't a silver bullet, but I can't remember anyone claiming their technology to be a silver bullet. (Amusingly tons of technologies claim an order of magnitude improvement, which is what this paper defines as a *silver bullet*.)

My greatest lessons from this paper were the importance of separating essential and accidental difficulties, and the central role of iterative development. Over the years the essential/accidental separation has become fuzzier for me - it's impossible to think of the essence of a problem without some form of representation, and in enterprise software the accidental difficulties increasingly appear as the essence of our problems. In contrast the strengths of iterative development have grown clearer and stronger. Within the agile movement we've seen huge developments in rapid turnaround of ideas into running tested code. I have no idea how to measure the consequences of iterative development on productivity, but I would place it as one of the great leaps of the last two decades.

A similarly large leap over this time foreshadowed by this paper is the presence of pre-built software, either in packages or libraries. These days nobody can propose a new language environment without access to a wide range of libraries. The open-source community has made an astonishing array of software available which I can use by just typing 'apt-get install'. When I compare what I'm working on now to what I had 20 years ago, I think I'd say that this software makes a greater difference to me than even the hardware that it drives.

If a message from this paper has been missed, it's the central importance of "great designers". It is something accepted in places like this, but when I think of our corporate clients I still see a huge gap between what they say and what they do. Our industry still has a long way to go here.

In the end core message behind the paper for me is that programming is fundamentally a hard task and that no single technology is going to leap up and make that difficulty go away. As a result we must distrust anyone who claims such a thing. I think technologists instinctively know this to be true, which is why silver bullet is only used as an accusation in our circles. Sadly many big purchasing decisions in the corporate world are not made by technologists - these days the werewolves are on the golf courses.

4. Ricardo Lopez, rjlopez@qualcomm.com

RICARDO LOPEZ is a Principal Engineer in the Office of the Chief Scientist at QUALCOMM. He is responsible for software architecture, software process, and sometime *Just Plain Old Software* (JPOS). Architecting and designing software for over thirty years, he has been an evangelist for OO technology for the last twenty years and he has the arrow heads to prove it.

There is a "silver bullet" – it is the pursuit of personal and professional excellence – this when achieved, easily gives us an order of magnitude improvement in software productivity. There is no "silver bullet" from without – it must come from within.

5. Aki Namioka, anamioka@cisco.com

AKI NAMIOKA has been a Software Development Manager for more than 9 years. She is a member of Cisco Systems' Voice Technology Group in Seattle. Prior to Cisco, Namioka held roles with IBM Global Services and Boeing's Advanced Technology Center.

In Fred Brook's seminal article, he names several promising technologies that have become standard in current software development projects. We use code generation, third-party packages – notably open source projects, static analysis tools, etc. All of these have made our jobs easier.

However, the biggest problem I am faced with isn't fixing accidental errors or managing essential complexity - rather, it's creating products that meet expectations. Design by Defect is time consuming and wasteful. We rely on our Product Managers, Quality Assurance, and other members of the development team, to indirectly represent the interests of our real customers. However, no amount of requirements writing will create a thorough representation of our final output. There have been several suggestions presented at past OOPSLAs to help mitigate the lack of on-site customers in the development process, e.g. customer stand-ins, iterative design, etc. I can share some positive experiences we've had during this panel.

6. Linda Northrop, lmn@sei.cmu.edu

LINDA NORTHROP has more than 35 years of experience in software development as a practitioner, researcher, manager, consultant, and educator. She currently is director of the Product Line Systems Program at the SEI where she leads the work in software architecture, software product lines, and predictable component engineering. She recently led a yearlong study including leaders in the software community to define technical and social challenges to the creation of ultra-large-scale systems that will evolve in the next generation. The report, *Ultra-Large-Scale Systems: The Software Challenge of the Future* (ISBN 0-9786956-0-7), has just been published. She is coauthor of *Software Product Lines: Practices and Patterns* and chaired both the first and second international Software Product Line Conferences (SPLC1 and SPLC2). She is a past chair of the OOPSLA Steering Committee and OOPSLA 2001 conference Chair. Before joining the SEI, she was associated with both the United States Air Force Academy and the State University of New York as professor of computer science, and with both Eastman Kodak and IBM as a software engineer. As a private consultant, Linda also worked for an assortment of companies covering a wide range of software systems. She is a recipient of the Carnegie Science Award of Excellence for Information Technology and the New York State Chancellor's Award for Excellence in Teaching.

When Brooks' "No Silver Bullet" article was published I was mid-way through the college professor stage of my career. I immediately made it required reading for Lecture 3 of my undergraduate software engineering course. That lesson's student outcomes included: awareness that software engineering involves more than programming; awareness of the software crisis; and familiarity with innovations, approaches, and movements introduced to combat the software crisis. Over the years, the article inspired scores of students, software continued to be in crisis state, and the article's key phrases lodged in my memory. I find myself consistently drawing on those phrases and their underlying meaning in my post-professor career. In fact, as time progresses they have more meaning, not less. I have not seen a silver bullet.

Progress has been made in software engineering in the last twenty years and we have indeed constructed increasingly complex systems. In cases where we have focused on the essence, the results have been breathtaking. Yet conceptual errors still abound and crisis mode continues in far too many quarters. Innovations have too often focused on the accidents not the essence and in some cases have added greater complexity to software production.

At the same time, our global appetite for complex software and software-intensive systems continues to increase at a rate comparable to the increases in computational capacity of hardware. Sandwiched in the middle is our need to understand the problem domains and model solutions that harness the computational power as well as the attendant power of today's sensor and wireless technologies. Current trends are leading us to systems of unthinkable scale in not only lines of code, but in the amount of data stored, accessed, manipulated, and refined; the number of connections and interdependencies; the number of hardware and computational elements, the number of system purposes and user perception of these purposes, the number of rou-

tine processes, interactions, and "emergent behaviors," and the sheer number of people involved in some way. Brooks' essential software difficulties of complexity, conformity, changeability, and invisibility are ever more poignant. In such systems the "grow not build" imperative is a fact of life, the boundary between developers and those we have called users blurs, and our accidental innovations help little. To wrestle these future werewolves we still need great designers and still have too few, but we also need to cultivate an interdisciplinary perspective that takes us uncomfortably out of our coding world.

7. David Lorge Parnas, david.parnas@ul.ie

DAVID LORGE PARNAS is Professor of Software Engineering, SFI Fellow, Director of the Software Quality Research Laboratory at the University of Limerick, Professor Emeritus at McMaster University and Adjunct Professor at Carleton University. Parnas received his B.S., M.S. and Ph.D. in Electrical Engineering – Systems and Communications Sciences from Carnegie Mellon University and honorary doctorates from the ETH in Zurich and the Catholic University of Louvain. He is a Fellow of the Royal Society of Canada and of the Association for Computing Machinery (ACM) and a Member of the Royal Irish Academy. He is licensed as a Professional Engineer in Ontario. Parnas won an ACM 'Best Paper' award in 1979, two 'Most Influential Paper' awards from the International Conference on Software Engineering, the 1998 ACM SIG-SOFT 'Outstanding Research Award', the 'Practical Visionary Award' given in honour of the late Dr. Harlan Mills, and the 'Component and Object Technology' award presented at TOOLS99. He was the first winner of the Norbert Wiener Prize from Computing Professionals for Social Responsibility and received the FifF prize from the Forum Informatiker für Frieden und Verantwortung in Germany. He is the author of more than 240 papers and reports. Many of his papers have been repeatedly republished and are considered classics. A collection of his papers can be found in Hoffman, D.M., Weiss, D.M. (eds.), "Software Fundamentals: Collected Papers by David L. Parnas".

Lead Bullets, Why are they not used?

The attractive thing about the mythical silver-bullets is that they are easy to use. With a silver bullet in your arsenal, you need not work hard or hone skills, just point and whoosh - the enemy is defeated. Anyone who has the bullet can kill a werewolf, no education or training is required.

Building software will never be like that. As Fred Brooks said, the essential difficulties will remain. However, I object to the phrase "accidental difficulties". The word "accident" is often used in an attempt to escape blame. "It was an accident" is often a weak apology, a statement that the damage was unintentional and a claim that it could not be avoided. In fact, the accidents that I know about have all resulted from a combination of negligence, momentary carelessness, haste, greed, and poor training. This is also true of the many poor software products that I have examined. In each of them, the mistakes that led to the complexity and errors are obvious violations of principles and techniques that were known many years ago. These principles and techniques are not easy answers; they are hard work, but they are what we need to do.

It is a poor workman that blames his tools. In software, there are many who blame their programming languages and believe that new ones will be the silver bullet that so many seek. Programming languages are undoubtedly the great contribution of computer science to computer system development. The concept of (what were then called) higher level languages and the first implementations of that concept were incredibly useful. It is interesting that these useful tools were developed by people who had never studied about programming languages but knew what they were trying to do with them. Today, programming languages seem to be the province of people who have been taught all about them but know little else. Many bring to mind another old proverb, "to the man with a hammer, every problem is a nail". If you examine the systems that consist of millions of lines of poor code, you will see that the problem is not "low level tools" but poor use of those tools. While I have no doubt that today's languages can be improved (mostly by unification and simplification), they are not the cause of our problems and no new language will be the "silver bullet" that developers seek.

One of the stranger silver bullets that I see popularized should be called "business as usual". For as long as I have observed software development, we have practised forms of iterative development, taken short-cuts when it comes to documentation and reviews, based our decisions on informal meetings, solved our problems by sitting next to others at a keyboard, and avoided "wasting time" by planning for the future. Where documentation was produced, it was seen as a required evil, and not taken seriously. Today I hear gurus advocating these old practices as if they were novel.

Another silver bullet has been the belief that we must depend on "great designers" to be "chief programmers" or "master architects" by means of great insight and creativity. If we were relying on such people to build our bridges, the ferry industry would be in great shape. There is a lot of routine work to be done and it is essential that it be done properly.

The belief that we can find a silver bullet is kept alive by another myth, the belief that we have made great progress in software engineering because we are managing to build increasingly complex systems. In fact, if there is progress it will because we will be building increasingly simple systems.

The belief that we have made progress is also encouraged by the impressive capabilities of the tools on our desktops. In fact, if you look closely you will see that these are made possible by highly improved hardware, not improved soft-

ware engineering. Many of the things we enjoy today (e.g. multiple-window displays) were invented and implemented in the 70s but only on very expensive specially developed hardware. If we tried to use today's software on the standard hardware available then, we would be sharpening our pencils as we wait for a response.

The only solution to the never-ending software "crisis" is to try to emulate the science-based, disciplined, document based, development we see in good engineering projects. Our road builders (who are far from perfect) follow a fairly rigid process in which increasingly precise documents (often annotated drawings) are prepared, reviewed, analysed mathematically where needed, and then carefully used in the next phase of the process. Entry to the profession is also controlled; professional titles are restricted to people who have had a professional education and passed a series of exams. These restrictions are backed up by a regulatory regime in which those who are negligent or incompetent are examined and lose the right to practice. Introducing such professionalism in the software field is not easy, but it is the only road forward.

8. Dave Thomas, dave@bedarra.com

DAVE THOMAS (*www.davethomas.net)* is the founder of Bedarra Research Labs and is associated with Object Mentor, Carleton University and the Queensland University of Technology. Dave was a principle in the object conspiracy which sold OO to unsuspecting application developers who got lost in hopelessly complexity and technologists who used them to create hopelessly complex middleware frameworks. He helped create tools for those most addicted. Then he conspired with the Agile Alliance to try to convince companies that the best software is actually made by craftsmen whose work practices are not driven by a big process water wheel. Dave still labors under the illusion that it is possible for knowledgeable end users to create much of their own software if only our industry would enable and empower them.

Accidental Complexity – Footsteps in the Sand:

By many objective measures we have significantly increased rather than decreased the accidental complexity of software development.

- Languages, while supporting better abstraction mechanisms, have greatly increased complexity with generics, attributes, XML etc.
- Software professionals are increasingly being certified rather than apprenticed into true craftsmen.
- Class Libraries and Frameworks present an increasingly complex and constantly changing sea of APIs that must be mastered to create even the simplest of applications. There are too many ways to do the same thing.
- Tools are much more complex, often unnecessarily so.
- Competing fads and pseudo standards force market decisions to override sound technical decisions. Constant change is the enemy of quality.
- Open source frequently obligates the developer to understand an even larger code base and all too often maintain some of it.

Essential Complexity – *We have seen the enemy and it is us!*

- We ignore the KISS *(Keep it Simple, Stupid)* principle, ever introducing more concepts, frameworks and tools.
- We tackle too many domain intensive problems without the proper domain expertise.
- We jump to the new language, technology or release without considering the impact, often orphaning perfectly good assets because they are in the older version or language.
- We maintain a dedication to low level languages, albeit OO, when we know that their KLOCS will eventually kill us.
- We insist on taking our technology further from the end users who are increasingly more computationally literate than some of their developers.
- We are always in too much of a rush to build quality in and are therefore constantly trying to test defects out.
- We constantly break the rules of software physics only to relearn them down the road as the product or application goes into its 3rd release.

Lessons:

- Emphasize craftsmanship and mentor to really instill best practices. Good code really does need to be written at least 3 times! Writing compact high function code is a challenge for even the best developers.
- Written and verbal communication should be an essential part of education and professional development which is as valued as efficient code. We need Literate Programmers.
- In education we need emphasize skills such as discovery since the majority of development is enhancing or repairing an existing code base.
- The most important decisions for a requirement, architecture or design are *"Is this really necessary?"* and *"Can we leave it out?"*
- Exploit simpler languages with clear semantics for both the language and the libraries.
- Consider using a more powerful mechanism only where it will give you substantial leverage in terms of reduced code or greatly improved quality.