No Silver Bullet Revisted

American Programmer Journal
November 1995

by Brad Cox,
bcox@virtualschool.edu
http://virtualschool.edu/mon

First published in
American Programmer Journal
Cutter Information Corp
Ed Yourdon, Editor

Abstract

Superdistribution could let software engineers overcome the software crisis as tangible domains surmounted the same problem, by encapsulating complexity so thoroughly that everyone else can forget it. The solution requires enforcing property rights in digital goods as robustly as conservation of mass enforces them for tangible goods.

Crises versus Silver Bullets

The software crisis has been with us now for nearly a quarter-century. During this time, the computer industry progressed at breakneck speed through the computer revolution, the personal computer revolution, and most recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web. Yet during the very period that the computer industry has been delivering exponential improvements in price-performance, software remains intractable; immune to the organizational innovations that make us think of computer manufacturing as a mature industrial age enterprise.

In one of the most influential books of this era, *The Mythical Man-month*, Dr. Fred Brooks observed that adding more people to a late software project only seems to make matters worse. And in an equally influential article, *No Silver Bullet; Essence and Accidents of Software Engineering*, he argued that the difficulties are inevitable, arising from software's inescapable *essence*, as distinct from *accident*; something we're doing wrong in producing it.

> *Denver's new international air port was to be the pride of the Rockies, a wonder of modern engineering. Twice the size of Manhattan, 10 times the breadth of Heathrow, the airport is big enough to land three jets simultaneously-in bad weather. Even more impressive than its girth is the airport's subterranean baggage-handling system. Tearing like intelligent coal-mine cars along 21 miles of steel track, 4,000 independent "telecars" route and deliver luggage between the counters, gates and claim areas of 20 different airlines. A central nervous system of some 100 computers networked to one another and to 5,000 electric eyes, 400 radio receivers and 56 bar-code scanners orchestrates the safe and timely arrival of every valise and ski bag.*
>
> *At least that is the plan. For nine months, this Gulliver has been held captive by Lilliputians-errors in the software that controls its automated baggage system. Scheduled for takeoff by last Halloween, the airport's grand opening was postponed until December to allow BAE Automated Systems time to flush the gremlins out of its $193-million system. December yielded to March. March slipped to May. In June the airport's planners, their*

bond rating demoted to junk and their budget hemorrhaging red ink at the rate of $1.1 million a day in interest and operating costs, conceded that they could not predict when the baggage system would stabilize enough for the airport to open.

--- Software's Chronic Crisis; Trends In Computing
by W. Wayt Gibbs;
Scientific American September 1994; Page 86

This paper will reexamine Brooks' argument from a different perspective to arrive at precisely opposite conclusions. Whereas he adopted the techno-centric perspective of the software industry's established paradigm, we'll step outside of this frame to examine it from a human-centric viewpoint. We'll not focus our microscope on how we build software today. We'll focus instead on the people, or more precisely on the incentive structures that might lead people and firms to build software in the future as mature engineering domains build baggage carts and airplanes today.

This difference in viewpoint leads to entirely different interpretation of the same bundle of facts that Brooks considered. The good news is that a silver bullet is indeed feasible to whoever musters the determination to forge it. The bad news is that, exactly as with the silver bullets that eliminated other crises throughout history, our silver bullet will also be a paradigm shift, not a technology. Paradigm shifts can be very bad news for they impact the most deeply held beliefs of established incumbents.

It is this resistance to new paradigms that has often caused established incumbents to resist to the end until they're ultimately displaced by newcomers who don't share the same system of beliefs. The indigenous freedom-loving Indians of the American frontier were once exactly such an incumbent. They were emphatically displaced by property-conscious European invaders during the confrontation that the winners called the taming of the wild west.

No Silver Bullet?

Let us consider the inherent properties of this irreducible essence of modern software systems: complexity, conformity, changeability, and invisibility.

--- Fred Brooks
No Silver Bullet
Essence and Accidents in Software Engineering

As Brooks suggests, "The complexity of software is an essential property, not an accidental one." We observe that this inherent complexity derives from four elements: the complexity of the problem domain, the difficulty of managing the development process, the flexibility possible through software, and the problems of characterizing the behavior of discrete systems.

--- Grady Booch
Object-oriented Design with Applications
Benjamin Cummings; 1991

Notice that something is missing from each of these lists. The 'essential' differences in these quotations are actually symptoms, not causes. They all originate from a causal difference that is so deep, yet so obvious, that neither author thought to even mention it in their lists.

This underlying difference is that electronic goods are made of bits but tangible goods are made of

atoms. This difference causes not only the symptoms that these authors listed, but the higher level differences in outcome that the the Denver International Airport suffered. The primal cause of the software crisis is the breakdown in human incentive structures originating from the problematics of buying, selling and owning goods made of bits.

At first, this difference may seem so obvious that no good can come of mentioning it. But it holds the key insight from which a silver bullet might someday be forged. Since tangible goods abide by conservation of mass, it is hard to replicate and transport them but trivial to buy, sell and own them. Therefore large commercial enterprises have arisen to provide them. These enterprises are capable cooperating across vast differences in space, time, language and culture, to master the vast complexity of even the most mundane of tangible goods. In the Denver International Airport example, these goods include not only the conveyor belts, airplanes and buildings at the top of the manufacturing pyramid but also the subcomponents used in building them right down to the raw materials from the mines, wells, and farms.

The situation with electronic goods is exactly the reverse. They can be replicated and transported at the speed of light, but it is hard to buy, sell and own them. Since bits do not abide by the physical laws upon which economics has been based since antiquity, we have been unable to mobilize a similar structure of production for reusable software subcomponents.

From this rediagnosis of first causes arises an as yet untried approach to solving the software crisis that, following Brooks, I'll call the silver bullet. If we could deploy a way of buying, selling and owning goods made of bits that is comparable to the incentive structure that underlies tangible manufacturing, we'd have a solution exactly like the one that built the tangible goods for the Denver Airport. The crucial aspect of this system is not whether it works for the large commercially significant objects at the top of this pyramid. The crucial aspect is that it work for multigranular subcomponents as they flow through a comparably elaborated software structure of production.

This article will outline how such a system might work should we ever muster the determination to actually build the infrastructure to support it. The good news is that the technical difficulty of building such an infrastructure is small, well within the capabilities of today's computer industry. The bad news is that, exactly as with all the other silver bullets of history, the real innovation isn't in the infrastructure technology itself. The innovation is a paradigm shift, redefining what it means, or better what it *should* mean, to say we buy, sell or own objects made of bits.

Manufacturing's Silver Bullet

What is more complicated, the mundane wooden pencil a Denver ticket agent might use to write your ticket or its electronic equivalent, a word processor? If you answered the word processor, you're in good company. Like Brooks and Booch, programmers invariably agree that computer software is the most complicated of all human activities. I've been heard word processors seriously compared with the complexity of Boeing's airplanes.

But according to the product manager of Microsoft Word (version 3.1), his software development team consisted of only eight programmers. This included only those doing hands-on work with the code but not those who documented, tested and marketed it. His numbers are comparable to my own experience with software development team sizes but perhaps not your own. Use your own estimate for the team size for your own last project. Then contrast the size of your team with the one that worked together to put those mundane wooden pencils in your desk drawer.

> *I, Pencil, simple though I appear to be, merit your wonder and awe, a claim I shall attempt*

to prove. In fact, if you can understand me--no, that's too much to ask of anyone--if you can become aware of the miraculousness which I symbolize, you can help save the freedom mankind is so unhappily losing. I have a profound lesson to teach. And I can teach this lesson better than can an automobile or an airplane or a mechanical dishwasher because--well, because I am seemingly so simple.

Simple? Yet, not a single person on the face of this earth knows how to make me. This sounds fantastic, doesn't it? Especially when it is realized that there are about one and one-half billion of my kind produced in the U.S.A. each year.

Pick me up and look me over. What do you see? Not much meets the eye--there's some wood, lacquer, the printed labeling, graphite lead, a bit of metal, and an eraser. Just as you cannot trace your family tree back very far, so is it impossible for me to name and explain all my antecedents. But I would like to suggest enough of them to impress upon you the richness and complexity of my background.

My family tree begins with what in fact is a tree, a cedar of straight grain that grows in Northern California and Oregon. Now contemplate all the saws and trucks and rope and the countless other gear used in harvesting and carting the cedar logs to the railroad siding. Think of all the persons and the numberless skills that went into their fabrication: the mining of ore, the making of steel and its refinement into saws, axes, motors; the growing of hemp and bringing it through all the states to heavy and strong rope; the logging camps with their beds and mess halls, the cookery and the raising of all the foods. Why, untold thousands of persons had a hand in every cup of coffee the loggers drink!

The logs are shipped to a mill in San Leandro, California. Can you imagine the individuals who make flat cars and rails and railroad engines and who construct and install the communication systems incidental thereto? These legions are among my antecedents.

Consider the millwork in San Leandro. The cedar logs are cut into small, pencil-length slats less than one-fourth of an inch in thickness. These are kiln dried and then tinted for the same reason women put rouge on their faces. People prefer that I look pretty, not a pallid white. The slats are waxed and kiln dried again. How many skills went into the making of the tint and the kilns, into supplying the heat, the light and power, the belts, motors, and all the other things a mill requires? Sweepers in the mill among my ancestors? Yes, and included are the men who poured the concrete for the dam of a Pacific Gas & Electric Company hydro plant which supplies the mill's power

My "lead" itself--it contains no lead at all--is complex. The graphite is mined in Ceylon. Consider these miners and those who make their many tools and the makers of the paper sacks in which the graphite is shipped and those who make the string that ties the sacks and those who put them aboard ships and those who make the ships. Even the lighthouse keepers along the way assisted in my birth--and the harbor pilots.

The graphite is mixed with clay from Mississippi in which ammonium hydroxide is used in the refining process. Then wetting agents are added such as sulfonated tallow--animal fats chemically reacted with sulfuric acid. After passing through numerous machines, the mixture finally appears as endless extrusions--as from a sausage grinder--cut to size, dried, and baked for several hours at 1,850 degrees Fahrenheit. To increase their strength and smoothness the leads are then treated with a hot mixture which includes candelilla wax from Mexico, paraffin wax, and hydrogenated natural fats.

> *My cedar receives six coats of lacquer. Do you know all of the ingredients of lacquer? Who would think that the growers of castor beans and the refiners of castor oil are a part of it? They are. Why, even the processes by which the lacquer is made a beautiful yellow involves the skills of more persons than one can enumerate!*
>
> --- I, Pencil
> My family tree as told to Leonard E. Read

This tale of the wooden pencil brings a crucial but slippery distinction into the software complexity debate. Although it might be academically amusing to compare the complexity of wooden and electronic pencils in absolute terms, arguing about absolute complexity doesn't really move the ball ahead.

What does is realizing that *encapsulated* complexity is no longer complexity at all. Its gone, buried forever in somebody else's problem. This shifts the focus to the human **systems** that manage complexity successfully versus those that don't. The productive question is "How do wooden pencil makers encapsulate complexity so successfully that the rest of us have the luxury of forgetting that complexity is even there?"

They do this by ensuring that somebody else has an incentive to hide the complexity for us. The quantum event that underlies their entire system is the commercial exchange transaction. This provides the office supply store its incentive to hide complexity by preventing us from having to build our own wooden pencils. And the same principle extends downwards through the structure of production pyramid underneath that store. It persuades the mill owner to remain in the business of hiding complexity from the office supply store. It persuades the lumberjack to remain in the business of hiding the complexity of logging from the mill. And it persuades the rape seed farmer to remain in the business of hiding the complexity of his farm from everyone else in this global food chain.

It really doesn't matter whether wooden pencils are more or less complex than electronic pencils. What matters is that, for wooden pencils, a thriving human system operates to hide complexity from everyone else. This system does its job so well that the rest of us have the luxury of entirely forgetting that it was ever even there. And if that doesn't qualify as a silver bullet for manufacturing's complexity, I'd have to agree with Brooks that there really is no hope for us.

Pencils as the Fruit of the Pencil Tree

This line of argument helps to see that complexity is not the fundamental cause of the software crisis. Complexity is itself a symptom of a much deeper disease. In fact, this disease is at the deepest level imaginable, so deep that it didn't even make it onto Brooks' and Booch's list of software's essences.

The disease is a breakdown in the very commercial exchange transaction upon which tangible manufacturing's complexity hiding mechanism is based. The path out of the software crisis involves appreciating how their solution actually works, and then designing one that has exactly the same human properties. The trick is that ours cannot rely on the conservation of mass laws that support commerce in tangible goods made of atoms. Our system needs a different fundamental basis, one that works for goods that do not obey the physical conservation laws upon which the traditional commercial exchange transaction relies.

The important thing to notice here is that conservation of mass has *reach*. It applies not only to the wooden pencils, but with equal force to every one of the subcomponents used in building them. It is as if the pencil in every customer's hand remains connected to the vast global tree of human relationships that produced it. Conservation of mass ensures that each trace of graphite a pencil leaves on paper

automatically draws precisely right amount of production from each of the lower levels of the tree. The consumption of a pencil in the Denver Airport is infallibly communicated, through price signaling, right down to the bottom levels of the tree. Price signalling tells the graphite miner in Ceylon exactly how much graphite should be mined. It tells the mill how much demand exists that they'll produce exactly the right number of pencils to satisfy it and not one pencil more. In other words, the pencil tree is an ecological system, exactly like a peach tree, in that it draws precisely the right amount of water and sunshine from nature, and transforms them into precisely the right number and amount of amino acids, sugars and proteins. All of these are precisely coordinated to support the production of fruit that we think of as the tree's commercially significant product.

In other words, the system that's operating here is *recursive* in a particularly powerful way. It doesn't just work at the level of peaches, pencils and word processors; the levels the world thinks of as as commercially significant. It doesn't only apply to the peaches, but with precisely equal force down through the leaves, the twigs, the branches, the trunk and the roots.

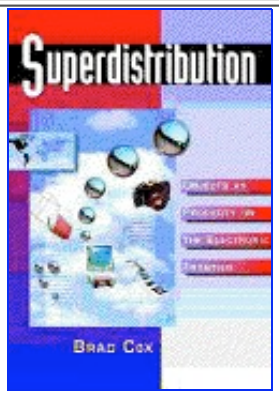
However since electronic goods, don't abide by conservation of mass, the vessels inside our electronic pencil tree get vapor lock. So the tree dies. Or more precisely, such trees never got a chance to evolve in the first place.

Demand for computer software is high, so high prices encourage us to provide large granularity computer software like word processors. But the smaller granularity objects we might use to build them can be replicated without revenue to their owner each time an application that relies on them is sold. So there is no incentive to persuade people to populate the lower levels of the tree. Since we all know that revenue and demand pulses cannot propagate recursively through the tree, we all naturally gravitate to the high granularity levels where we can get paid. We all struggle to build large applications unaided, so structure of production trees never evolve. Since supporting levels don't evolve, the entire task falls to the team at the very top of what would otherwise be a cooperative human pyramid. And we all gather at software engineering conferences to complain of the crushing complexity of our task.

Out of the Crisis

Unlike the hardware industry, which has organized itself into a fully elaborated rainforest of mutually interdependent structure of production trees, the software industry remains stuck in the unicellular, bacterial stage of the primordial ooze.

I'll avoid here the question of whether programmers might be individually better off if our industry remained in the primordial ooze stage. My book engages that question in detail by examining how the analogous confrontation between primitive and advanced social orders developed on the Wild West. Here, I'll simply take this industry's professed interest in "doing something" about the software crisis at face value and concentrate on showing what would be needed to evolve an advanced structure of production tree for goods made exclusively of bits.



Superdistribution; Objects as Property on the Electronic Frontier by Brad Cox; Addison Wesley 1996; ISBN 0201502089; at

Notice that this cannot be addressed solely at the level of large granularity computer applications. It involves attention to how we can evolve supporting layers, healthy roots, trunk and branches, to support the production of commercially significant fruit. However the fruit is the only level that is getting serious attention today. Copy protection technologies such as dongles, shareware unlocking technologies, license servers, and encryption are all unigranular

solutions. They enforce ownership of objects that are large enough to stand on their own as commercial objects. But they do nothing for smaller-granularity components from which large-granularity objects might someday be composed. Yet large applications are the one level of the tree that least needs greater protection. After all, applications can be protected by simply attaching them paper and cellophane that can be bought and sold like cornflakes in an ordinary industrial age software store.

The path out of the crisis involves finding a viable replacement for the quantum event at the core of the economic tree that produces tangible goods like pencils. This quantum event is the commercial exchange transaction. Most commonly, this event relies on conservation of mass to define exchange in terms acquisition of atoms. But even in tangible domains, acquisition of atoms is not the only basis of owning. The path out of the software crisis involves another interpretation of owning, which is based on acquisition of use instead.

We don't generally think of these two meanings as separate because most of our daily transactions bundle them together. For example, when I buy a burger at the fast food stand, I'm actually buying an ownership bundle. I acquire ownership of the physical atoms from which the burger is comprised. Conservation of mass immediately kicks in to guarantee all parties that the stand owns one less burger, or more precisely, the bun, meat and sauce from which burgers are assembled. My acquisition of these atoms immediately generates a scarcity pulse that immediately travels throughout a vast agrarian system, encouraging farmers and distributors around the globe to produce more.

But ownership of the atoms is not the only thing I buy with my burger. I've implicitly acquired as part of my rights bundle the right to *use* those atoms as I please. I can eat it myself, give it to a friend, feed it to my dog or just throw it away. I'm even entitled to take my burger to a competing stand and resell it to someone else. That is to say that my transaction involves not only copyright as enforced in the case of tangible goods by natural law. My transaction also involves implicit useright, which is rarely articulated as a separate right.

However defining owning in terms of acquisition of atoms is by no means the only way to buy and sell things, just the most common. The person that owns the atoms can retain the atoms on his balance sheet but sell the right to *use* those atoms to someone else. Hertz and Avis are masters at this business. They buy automobiles, and all of the headaches of owning and maintaining them on the assert side of their ledger. They sell the right to *use* those atoms to customers by the day or the mile.

This is where our escape from the software crisis could be launched. Today everyone is accustomed to buying software the way we buy burgers. We're accustomed to acquiring the right to use software bundled with the bits. However, we somehow realized that bits add absolutely no value to our balance sheet. Bits are actually a liability because they occupy physical storage space that might be used for something else. And again unlike atoms, they represent no cost to the provider because consuming them generates no scarcity pulse requiring him to buy more.

So why don't we just give them away? After all, nobody wants bits for bits are just packaging. The asset is the software utility that the bits encode. And it costs nobody to provide them. What people want is the right to use the utility that the bits encode.

So why not just give the bits away freely and base revenue collection on acquisition of use instead? This shift in what it means to buy, sell and own immediately shifts the ownership issue onto ground where technology can apply. For although computer software is intrinsically unable to monitor its *acquisition*, it is trivially able to monitor its *use*. Like this:

```
if (query()) { /* is this a paid up customer? */
```

```
        ... /* deliver requested service */
        commit(); /* record successful delivery */
} else { /* refuse service */
        ...
}
```

Those query and commit calls rely on communication infrastructure that doesn't exist today. But could be provided relatively easily. Since this infrastructure deals with money, it would have to be sufficiently tamper-resistant to be trusted by all parties. But an infrastructure capable of conveying the usage information captured by these query and commit calls with a financial institution for billing is "just a matter of engineering", and quite within the reach of today's computer industry.

The key to this approach is a paradigm shift in which buying, selling and owning is based on acquisition of *utility* encoded in the bits instead of acquisition of the *bits* themselves. Invocations have the exactly same recursive properties for software that conservation of mass has for commerce in tangible goods. For example, when a large object runs on an end-user's PC, the user will be charged a for using this application. However the application's *owner* will be charged for his application's use of any reusable subcomponents that it invokes.

Just as conservation of mass conveys scarcity pulses throughout global industries, metering of invocation can carry energy right from the top to the roots of the structure of production tree.