The past, present, and future of the model-based digital design thread

Paul Eremenko

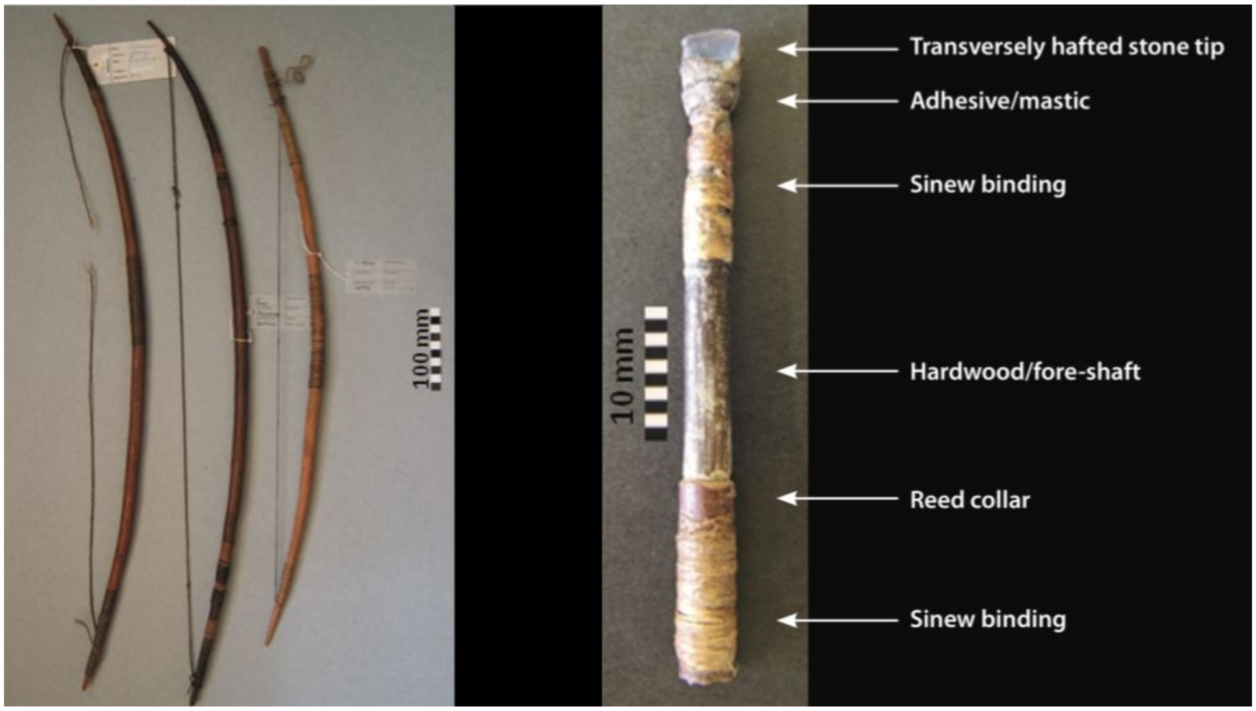
National Science Foundation Cyber-Physical Systems Annual Meeting
November 21, 2019
Arlington, Virginia

I want to talk to you today about design of large-scale, highly-integrated, cyber-physical systems. I imagine that of greatest interest to this audience is a perspective on the future. But I've found in my career that it pays to be a student of history in order to really understand the present and to frame the future. And so I'd first like to take you through a brief history of engineering design.

Although design has been around for a long time, there have been only a few distinct design epochs. Today, I'll talk about three. The shift between them is pretty much always driven by cost and schedule. It is rare that you truly can't build something with existing design methods. It's usually a situation where you can't build the next thing because the last one cost so much and took so long. For instance, could the Apollo program have gotten us to the moon without modern systems engineering? Perhaps. But it could not have done so in only 8 years and at an average annual cost of about half a percent of GDP.

It is also no coincidence that advancement in design methods has almost always been driven by either military or transportation applications, and frequently military transportation applications. They have been at the leading edge of what we, as a civilization, are capable of creating and capable of affording. The notable exception is VLSI design for integrated circuits, and we'll talk more about that in a bit.

So, let's begin at the beginning. Probably the first real example of engineering design was the bow and arrow, which emerged in sub-Saharan Africa during the Middle Stone Age, around 60 thousand years ago. Why was it significant? It's not the first composite tool—which means one that's manufactured from multiple components. The Neanderthals actually made axes with a wooden clamp shaft and a stone blade some 300 thousand years ago. The bow and arrow, however, are the first time that we see one composite device employed to effectively use another composite device. The cognitive complexity of such a combination is thought to be a capability unique to *Homo sapiens*.



Source: M. Lombard & M. Haidle, "Thinking a Bow-and-arrow Set: Cognitive Implications of Middle Stone Age Bow and Stone-tipped Arrow Technology," Cambridge Archaeological Journal, Vol. 22, No. 2, 2012, pp. 237–64.

It's also the first example of modularity in engineering design. Granted, it's a very loosely coupled modular system. But it is coupled with a few design parameters like the length and weight of the arrow, and the string tension in the bow. It's also the first example of a fairly complex manufacturing process, including water, fire, an adhesive, a lubricant (to prevent cracking of the bow string), as well as a variety of subsidiary tools needed for carrying, stirring, and cutting, each with their own manufacturing process.

This was a remarkable breakthrough 60 thousand years ago. But to me, what's even more remarkable is that this was pretty much the state of the art in engineering design for the next 59.7 thousand years. Progress in weapons technology eventually became synonymous with advances in metallurgy, and shipbuilding became the industry which was pushing the limits of engineering design. And until the 1700s, shipbuilding was done very much in the same "craft tradition" of design as the Stone Age bow and arrow.

MIT's David McGee describes it in these terms: "In the absence of drawings, the defining characteristic of the craft approach is that craftsmen work immediately with their materials. Final dimensions are determined only as the materials are actually worked and the artifact actually made. There is no separation of designing from making. The two activities take place at the same time. There is no separation of designer from maker. They are the same person. Craftsmen compensate for errors, inaccuracies, and inconsistencies in the material as they work. They change their minds as they go along. They consider what has been done and respond to it, maintaining a feedback loop with the object in a process traditionally referred to as cutting and fitting."[1]

19
VERAGVA PARS.

Men from Francisco de Orellana's expedition building a small brigantine, the "San Pedro."

The craft tradition of engineering suffered from three types of limitations. First, it was inefficient. It resulted in significant waste of materials, as each part would start out being oversized and be cut down to shape. This cutting and fitting was also extremely laborious. And because each fitting decision was an exercise in engineering judgment, it required labor that was quite skilled and therefore expensive. In fact, the system of apprenticeship arose to provide a trained workforce of craftsmen. Since a lot of

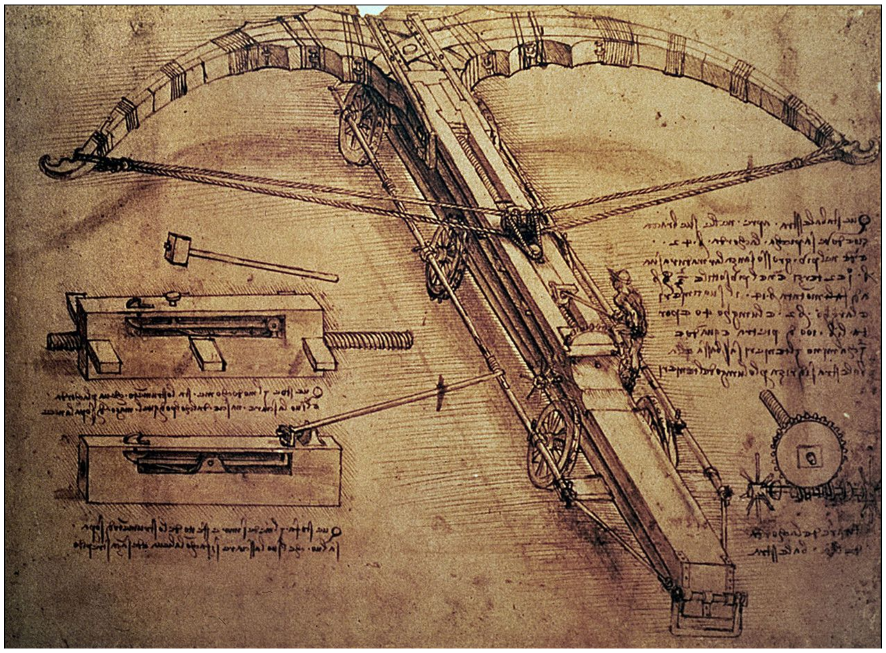
[1] D. McGee, "From Craftsmanship to Draftsmanship: Naval Architecture and the Three Traditions of Early Modern Design," *Technology and Culture*, Vol. 40, No. 2, 1999, pp. 209-36.

individual judgment went into the cutting and fitting process, the overall time and cost to complete a product of any complexity was highly variable. And so was the outcome—the product itself.

Second, this craft system disincentivized innovation. Because there was no way to predict whether a particular change would work and would actually improve the product, there was a natural reluctance to deviate from the last known working design. Apprentices were selected not for their creativity, but for their ability to faithfully emulate their master. So innovation crept pretty slowly for many millennia.

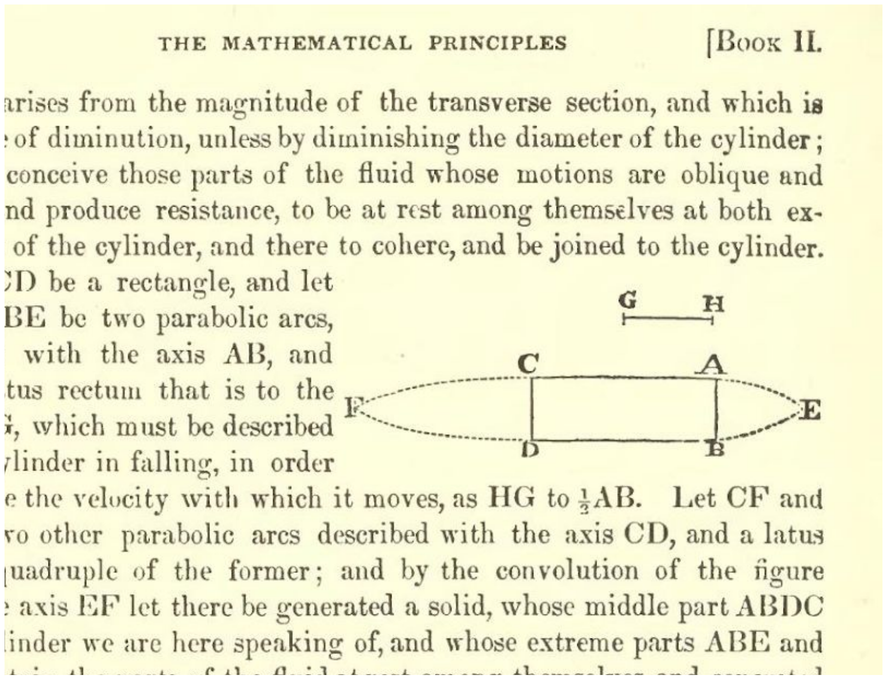
And the third big limitation of the craft approach was that it limited the complexity of the overall product which could be built. Because each part was uniquely cut and fitted, there was no systematic approach for splitting the work among multiple craftsmen or multiple shifts. If you did have a team working on a product, they had to be very closely supervised by the master craftsman who took charge of ensuring that all the parts fit together. This limited the practical size of a team, and more importantly, it limited the complexity of the design to the cognitive capacity of the master craftsman.

The Renaissance era saw the invention of linear perspective, which enabled the first engineering sketches, like those of Leonardo Da Vinci. These might have stirred the imagination, but they did not seem to have had a practical impact on engineering. It was not until the 17th century that we saw the craft era draw to a close, and give way to something radical: the use of dimensioned drawings for engineering design.



Source: L. Da Vinci, Codex Atlanticus, ca. 1500.

At this point, a few things were happening in Europe. Britain was on the rise as a great naval power. The cost of the navy was becoming a significant draw on the royal treasury. There was pressure to find more cost-effective ways to design and build ships. Also, the Scientific Revolution was well under way, and the best scientific minds—people like Boyle, Hook, Wren, Bernoulli, and even Newton—were very much focused on the shipbuilding problem. Today we think of Newton's *Principia* as famous for the laws of classical mechanics. At the time, it made a much bigger splash (so to speak) for what turned out to be an incorrect solution to the problem of finding the shape of minimal resistance in a fluid.



THE MATHEMATICAL PRINCIPLES [Book II.

arises from the magnitude of the transverse section, and which is
of diminution, unless by diminishing the diameter of the cylinder;
conceive those parts of the fluid whose motions are oblique and
nd produce resistance, to be at rest among themselves at both ex-
of the cylinder, and there to cohere, and be joined to the cylinder.
D be a rectangle, and let
BE be two parabolic arcs,
with the axis AB, and
tus rectum that is to the
, which must be described
linder in falling, in order
e the velocity with which it moves, as HG to $\frac{1}{2}$AB. Let CF and
o other parabolic arcs described with the axis CD, and a latus
uadruple of the former; and by the convolution of the figure
axis EF let there be generated a solid, whose middle part ABDC
inder we are here speaking of, and whose extreme parts ABE and

Source: I. Newton, Philosophiæ Naturalis Principia Mathematica, 1687, Book II, p. 342.

As the cost of the navy continued to grow, King Charles I made the fateful decision to bring back a medieval tax custom, called ship money, to raise taxes to support the navy beyond what Parliament had approved. Although this action was endorsed by the courts, it proved so unpopular, that it became one of the major issues that incited the English Revolution, the rise of Oliver Cromwell, and Charles' ultimate beheading.

After the restoration of the monarchy, Charles's son, Charles II, took a different approach, and instead made a number of reforms to professionalize the Royal Navy. The introduction of dimensioned drawings was huge. It enabled the standardization of the design, the decomposition of the design for production, and the separation of design and production. The naval architect was now based in London. And the dockworker was no longer the master craftsman. He could be paid much less. Many of them could work at once. And they could work in shifts. The worker’s only task was to make shapes determined by the designer. Any creative contribution could only be a mistake, since a part altered by one worker would not fit parts made properly by others.

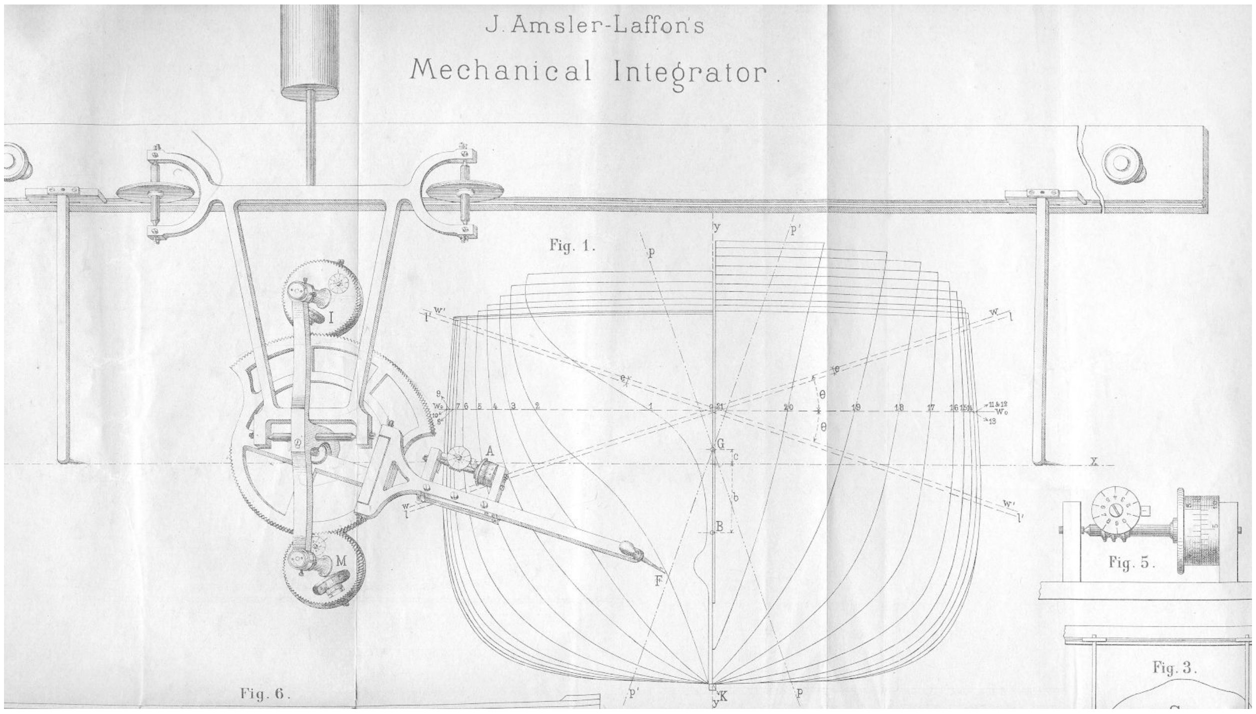
Artist Unknown, “The Execution of Charles I,” Scottish National Portrait Gallery, ca. 1649.

There was recognition at the time that ship behavior was highly sensitive to its geometry, but apart from some general rules of thumb, the relationship was an enigma. Samuel Pepys, who headed the British Admiralty under Charles II, observed that “it is worthy of note how small things are sometimes found to mar or mend a ship’s quality.”

The introduction of measured drawings and the intense scientific focus on the problem meant there was a flurry of activity to try and predict the behavior of the ship’s design before actually building it. The 17th and 18th centuries yielded a number of scientific theories and models. Some, like Newton’s solid of minimal resistance, were simply wrong. Many were right and form the foundation of modern fluid mechanics.

But, the path from a scientific theory to engineering practice is not straightforward. While engineers of the time recognized that the problem of design is all about tradeoffs between competing objectives, the scientists instead looked for perfection. It was Pepys again who reported that Robert Boyle (of Boyle’s Law) tried to develop a way to "prove the true body of a ship" and Christopher Wren (best known as the architect of St Paul's cathedral) tried to create "the truest figure for any body to pass through water."

For all that effort, by the end of the 18th century, the only predictive models that were actually used in shipbuilding were of a ship's displacement and its stability. Both calculations began with a laborious accounting of the distribution of mass on a ship. It could take up to two years to calculate the location of the center of gravity and center of buoyancy for a large ship—longer than it took to build it! It wasn't till the end of the 19th century, with the invention of the mechanical integrator, that the stability calculation became a practical one.



Source: A. Amsler, Instructions for Using J. Amsler-Laffon's Mechanical Integrator, Schaffhausen, 1883.

By the beginning of the 20th century, aeronautical design was starting to displace naval design as the great technological problem of its time. But the design approach to airplanes picked up right where naval architecture left off. There were dimensioned drawings. Predictive behavioral models of the airplane weren't much better than those of the ship in moving from scientific theory to practical engineering applications. Instead, empirical models were starting to fill the need, with rapid growth in wind tunnel testing, covering vast numbers of possible design variations.

World War II and the years immediately following it saw an avalanche of technological progress, most notably the invention of the atom bomb and the digital electronic computer. The atom bomb is significant because the Manhattan Project, perhaps like nothing else in the history of humankind, built the link between science and engineering. In the course of just a few short years, the nuclear fission chain reaction went from a theoretical concept to a practical weapon that ended the war. There was now a model for transitioning scientific concepts to practical applications.

The digital electronic computer is why we are here today, talking about cyber-physical systems. It presented, for the first time, a discrete mathematical abstraction over an analog electrical device—at the time, the vacuum tube. It was the first meaningful use of abstraction to hide complexity in an engineering system.

It did not take long after the first computer to recognize that a new approach to engineering design was going to be needed. One of the precipitating events was the U.S. Air Force's SAGE AIR DEFENSE PROGRAM, which took the better part of the 1950's to complete.

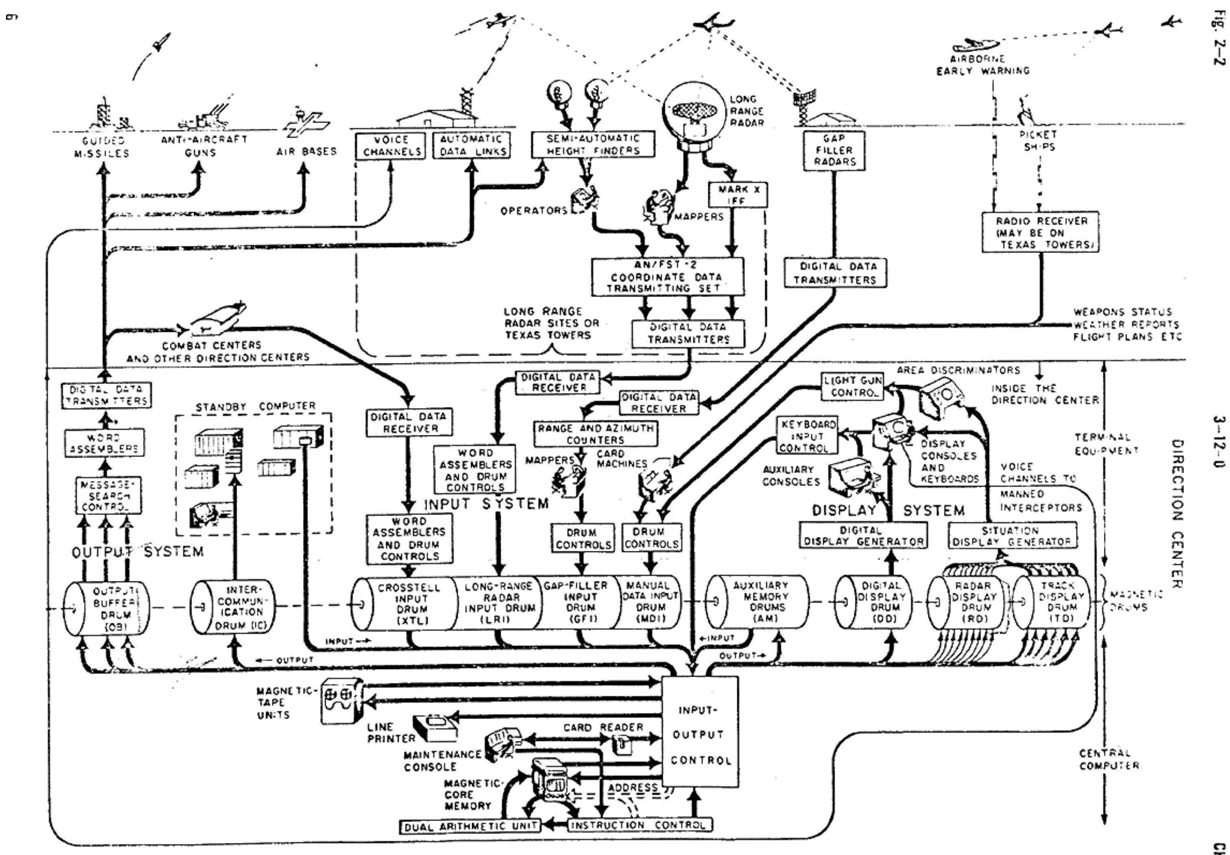


Figure 2—2. Information Flow in the Sector

Source: Introduction to AN/FSQ-7 Combat Direction Central and AN/FSQ-8 Combat Control Central, IBM Military Products Division, 1959, ch. 2, p. 6.

The purpose of SAGE, which stands for Semi-Automatic Ground Environment, was to gather and process data from ground radar sites to produce targeting information for intercepting aircraft and missiles. The SAGE program was both a spectacular success and a resounding failure. It resulted in many innovations in computing hardware, networking, and software development methods, and is firmly enshrined in the annals of computing history. It was a massive step change in system complexity over anything we had seen before, and it was the first time we saw significant software complexity in a system.

As a result, the program took too long, was too expensive, and by the time it was fielded in 1959, the long-range bomber threat against which SAGE was designed was largely superseded by intercontinental ballistic missiles (ICBMs). At completion, it was the single most expensive program in history, costing over $10 billion in then-year dollars, nearly five times the cost of the Manhattan Project.

SAGE's contribution to the history of engineering design was to highlight the urgent need for a new design methodology. The Air Force rushed to do just that on the Atlas ICBM program, which was just starting just as SAGE was hitting some of its greatest challenges in 1954. The program was entrusted to General Bernie Schriever, with the charter that the program be "accelerated to the maximum extent that technological development will permit" in light of the perceived missile gap with the Soviet Union. Schriever made the best of his mandate and went on to create a new systems engineering methodology that by 1957 harnessed the efforts of 17 principal contractors, 200 subcontractors, and a workforce of 70,000. For a single product development, this was unprecedented.

Systems engineering is a methodical approach for decomposing a large design problem across teams of design engineers and subsequently integrating the pieces back into a whole design. The systems engineering workflow goes something like this: you partition the system, typically along lines of functional expertise; then you decompose and flow down the requirements, and allocate them to each partition; you specify the interfaces between the partitions; then you design and optimize each partition to meet its allocated requirements; you integrate all of the individual partition designs; as you do that, you do verification testing to confirm that the system meets its requirements; and finally, you validate that the system performs its mission (in other words, that the requirements were correct). What I am describing here is what most of you probably know as the "systems engineering V."

This was the first time that the complexity of a design was not limited by the cognitive capacity of the chief engineer. It was a way of bringing together teams of experts in aerodynamics, structures, propulsion, guidance, and control, and have them all effectively contribute toward the design of the whole. Notably, controls were the cybernetic portion of the missile, and systems engineering brought a welcome structure to the nascent field of software development.

The SAGE operational program consisted of some 100,000 machine language instructions. Straight away two opportunities for improving programming productivity became apparent. The first was modularity for purposes of reusing segments of code. Once you wrote the code to compute the sine of an angle, for instance, the advantage of reusing that code segment every time you need to find the sine is obvious. Modularity also had the immediate benefit of enabling the parallel development by multiple programmers.

The second was abstraction. The really low-hanging fruit was to write instructions in a human-friendly form and have them automatically translated to machine code. Next came the first high-level programming languages like FORTRAN, that would present the programmer with a mathematical abstraction independent of the machine. By the end of the 1950's, a half dozen such languages were in widespread use. Since then, at least two additional levels of abstraction have been introduced in software design. These enable the programmer to express high-level functional requirements or even intent, from which the software is automatically generated.

Software presented a third major opportunity for productivity enhancement, but it went largely ignored for a number of decades. I blame, in large part, the subjugation of software development to the systems engineering methodology. Design closure for a physical system is not a trivial task. It requires combining the results of behavioral models of all of the different domains in the system to predict its overall performance against requirements. It happens only a handful of times throughout the design process—typically to support major design review milestones. And so, software development followed this same model well into the 1990's—what came to be known as the Waterfall Model—with only a small handful of major "builds" throughout the design cycle.

Design closure for a software system, of course, is much more straightforward than for a physical system. The behavioral models are the software itself, and functional performance prediction is a matter of compiling and running it. Today, standard practice is agile development with daily, weekly, or bi-weekly software builds, incorporating learnings and changes from each into the next, and incrementally building functionality to ultimately meet requirements.

The systems engineering methodology remains the dominant design method for large-scale, complex, cyber-physical systems in aerospace, defense, and most commercial sectors. In my career, I practiced it first-hand at DARPA, Airbus, Collins, and Pratt & Whitney, as well as in the commercial sector at Motorola, Carrier, and Otis Elevators. And it's pretty much the same everywhere. Software teams that are part of the systems engineering process typically use agile development and then have to freeze occasionally and "fake the inputs" to the major waterfall design milestones.

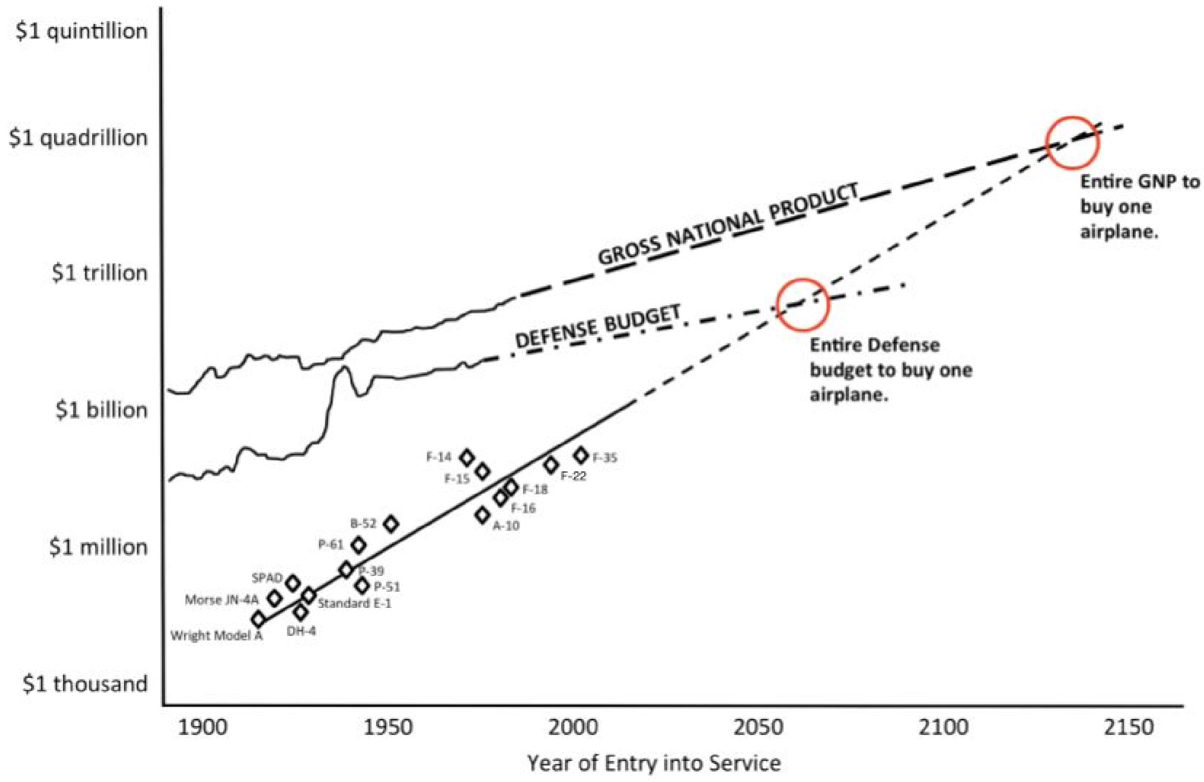
Before I go on to critique modern systems engineering, let me just say that it's been the most successful design methodology in history. It solved all of the historical shortcomings of the design epochs that came before it. It enabled the efficient deployment of labor by separating design from manufacturing and further partitioning the design problem along functional discipline lines. This allowed engineers to specialize, and also enabled lower-cost, junior engineers to perform a lot of the low-level tasks which could be codified as standard work processes. It facilitated innovation because systems are designed to requirements, not as an incremental departures from prior designs.

We all like to complain about the slow pace of innovation in aerospace and defense. And certainly large organizations exhibit risk aversion, group think, and other bureaucratic maladies. But here it is important to separate organizational will from the design methodology. Systems engineering is not to blame. In fact, we should thank it for many incredible innovations ranging from Apollo to GPS to the modern commercial air transportation system. It made possible the creation of products of complexity that would've been unimaginable with prior design methods.

But the complexity does come at a cost—both literally and figuratively. There are many ways of showing the cost and complexity progression of aerospace systems. I think the fact that both are growing rapidly is relatively uncontroversial at this point, and complexity metrics tend to get a little wonky. So I will stick with the slightly cliched, but veritably true Augustine's 16th Law.

Norm Augustine, who was the CEO of Lockheed Martin, called this Calvin Coolidge's Revenge, in reference to the President Coolidge, in 1928, quipping, "Why can't we buy just one aeroplane and let the aviators take turns flying it?" after being presented with a budget request of $25,000 for a whole squadron of aircraft. Augustine's point is that

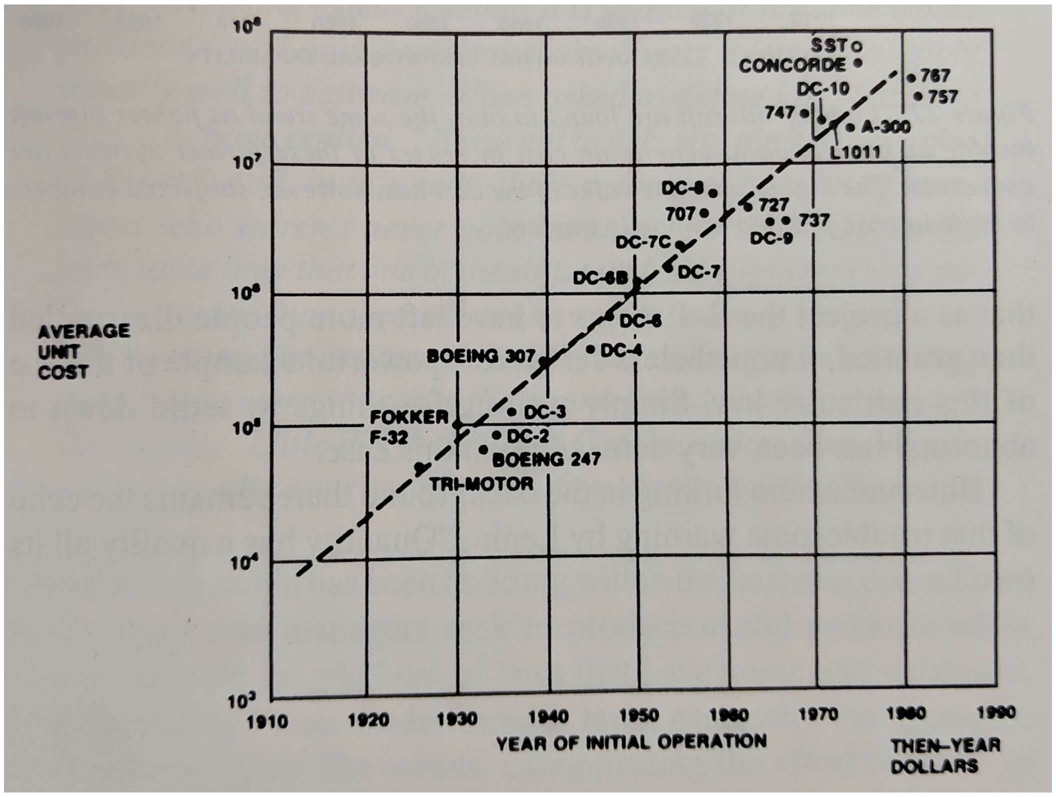
Coolidge's wish will come true if the exponential cost trend continues through the year 2054.



Source: N. Augustine, Augustine's Laws, AIAA Press, 6th ed., 1997, p. 106. Extended by M. Nowack, DARPA.

We should all be wary, of course, of falling into the Malthusian trap of extrapolating exponential trends into the distant future. Augustine first published this in 1983. And so I've helpfully plotted a couple of the new fighter aircraft since 1983, including the Joint Strike Fighter, and they have pretty faithfully followed the exponential trendline. And lest you think that this is unique to defense, and a consequence of red tape or cost-plus contracting or whatever your favorite critique of the Pentagon is, Augustine obliged us with a comparable exponential cost trend for commercial airplanes.

There is actually a pretty straightforward explanation for why the systems engineering process increasingly struggles with complexity growth. As you integrate the individually designed and optimized partitions of the system, in addition to the interfaces between these partitions that were specified and tracked during the decomposition and requirements allocation phase, there are inevitably undesired and unanticipated interactions and coupling that occurs. This frequently takes the form of mechanical vibrations, thermal leakage, or electromagnetic interference.

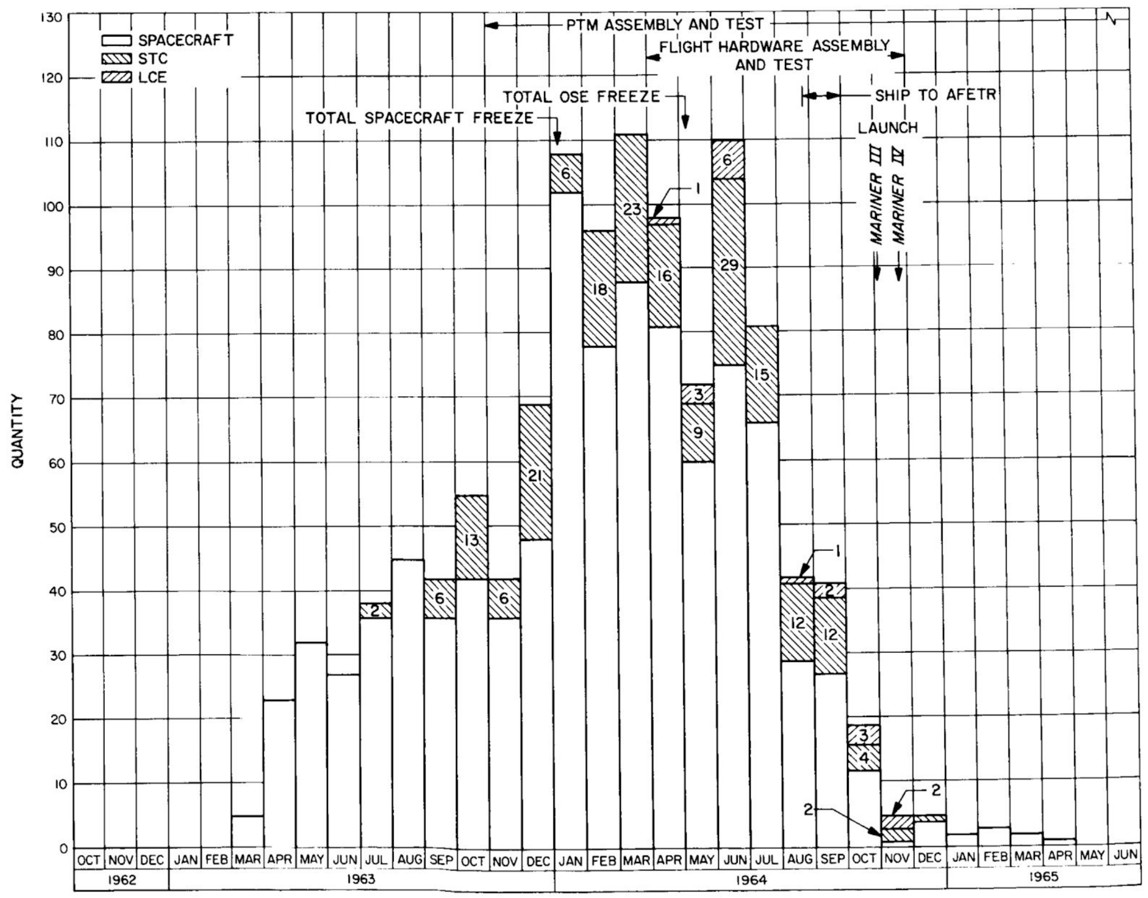


Source: N. Augustine, Augustine's Laws, AIAA Press, 6th ed., 1997, p. 109.

Because airplanes, satellites, rockets, and other vehicles aren't getting any bigger but are getting more complex, the "complexity density" is growing, more components are miniaturized and packed ever more tightly, and the number of these unanticipated interactions grows. They get discovered, of course, but typically this happens during verification testing, which is quite late in the design cycle, and so changes to redesign the system are relatively expensive.

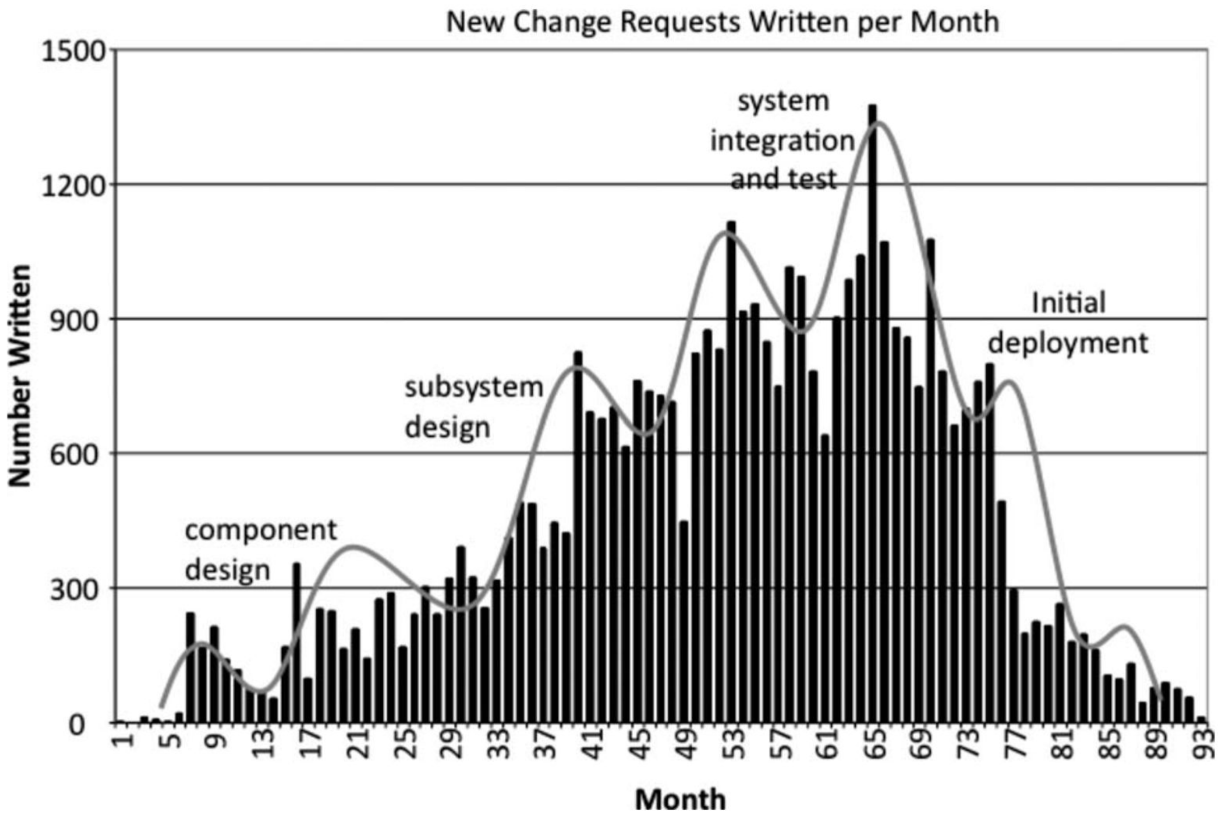
This is nicely illustrated in the following series of plots that show engineering change requests (or ECRs), these are basically formal documentation of a redesign, over the life of a program. The program details are not terribly important for our purposes.

First, here they are for the Mariner spacecraft, which was a Mars exploration spacecraft from the 1960's. And the thing to note here is that the ECRs peak after the design is ostensibly "frozen." So these are all changes that are taking place as a result of things uncovered during the verification phase.



Source: Mariner Mars 1964 Project Report, Vol. 1, NASA Jet Propulsion Laboratory, 1965, p. 32.

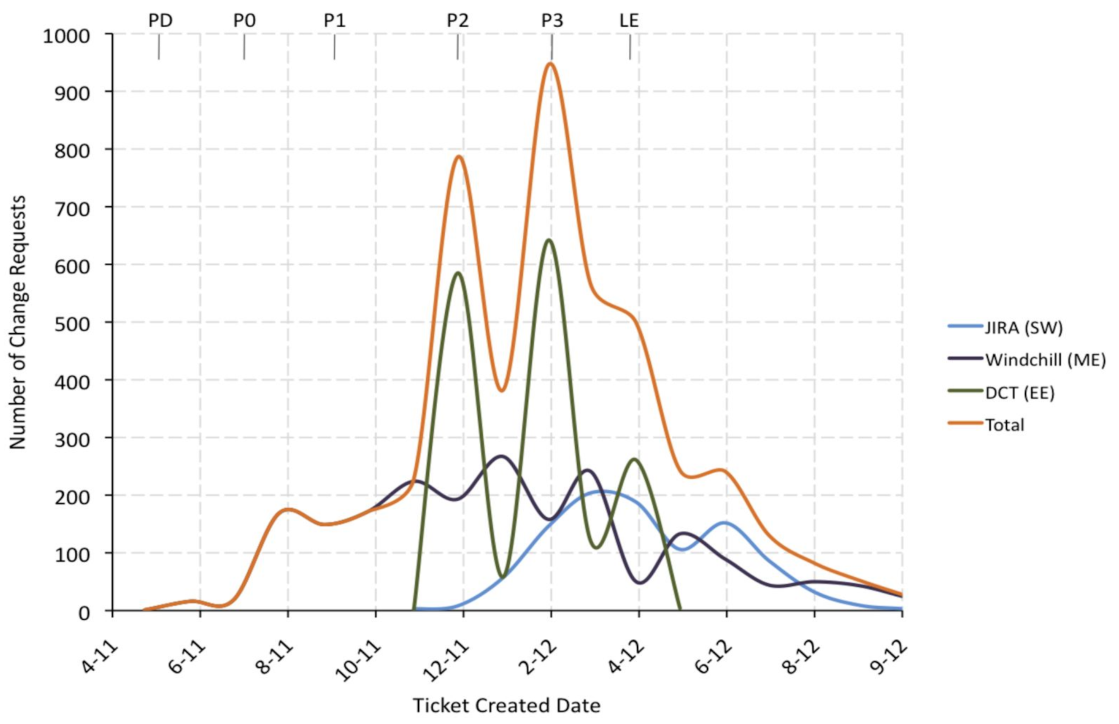
Next, here are the ECRs for a large Raytheon ground radar system from the early 2000's. This is a much more complex system, so absolute numbers of ECRs are an order of magnitude higher, but the basic shape is the same. The peaks, by the way, likely correspond to major design closure milestones, when there is a flurry of integration activity and design iteration taking place.



Source: M. Griffin & O. de Weck, "Change Propagation Analysis in Complex Technical Systems," Journal of Mechanical Design, Vol. 131, No. 8, 2009.

And lastly, here is the ECR history for a Motorola smartphone in the early 2010's. And here they are broken out by discipline—mechanical, electrical, and software. And you see that mechanical and electrical exhibit this sort of modulated behavior where they are chasing design reviews. While software is smooth and steady, because they are using agile development, and so there is a fairly constant stream of improvements in each sprint.

Also, the verification problem itself does not scale well with system complexity. This is called the "state space explosion"—the exponential growth in the number of possible system states or modes which must be tested. As we start to develop systems that learn and adapt over the course of their operating life—which is not yet something we see in aerospace, but we are starting to see in cars—the verification problem is going to get really hard!



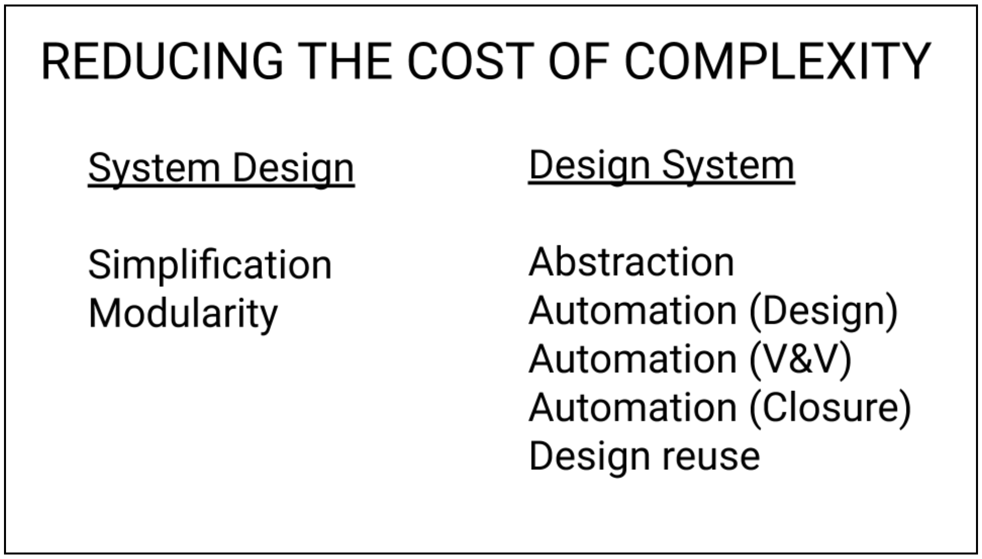
Source: Data assembled by author and first presented at the 2014 Complex Systems Design Management Symposium.

We are starting to see other creaks in modern systems engineering. For instance, the fairly primitive way in which it deals with software and the cyber-physical seam is coming to haunt us on cyber security. We seem to have gotten pretty good at building reliability—which is more-or-less a static attribute—into systems. But we are coming to realize that security—where there is an intelligent, rapidly-evolving adversary on the other side—requires a different approach.

So what can be done? I think that the Joint Strike Fighter, today's most expensive development program in history by a long shot ($400 billion in development and acquisition costs), is likely to be the straw that breaks the camel's back. Hopefully no beheading will be necessary. But a new design epoch may be coming soon. What will it look like?

Let's consider all the options for reducing the cost of complexity in cyber-physical systems. Here's my list, heavily inspired by what has already been accomplished on the cyber side of cyber-physical systems. If you think I am missing any important ones, I'd like to hear from you.

There are two categories of things we can do. First, we can make the product itself less complex. That's on the left. And there are basically two ways of doing that. One is just make it simpler. Sometimes a technological breakthrough or sheer ingenuity can help find a way to meet a given requirement more simply. An example might be an electric drone replacing a manned helicopter for a reconnaissance mission, for instance. But this is rare and hard to anticipate. So it's not really a systemic solution for engineering design.



REDUCING THE COST OF COMPLEXITY

System Design	Design System
Simplification	Abstraction
Modularity	Automation (Design)
	Automation (V&V)
	Automation (Closure)
	Design reuse

Alternatively, almost any engineering design contains some superfluous complexity beyond the absolute minimum level of complexity needed to meet the requirement. I am describing something like a physical analogue to Kolmogorov complexity for algorithms. Then anything beyond that would be superfluous complexity. Although I know of no systematic study of this notion of unnecessary complexity in physical systems, I am skeptical that it would yield significant gains. But there have been a few examples of successful applications of design automation to create designs superior to those made by humans. So there is no reason in principle that similar approaches cannot be used to identify lower-complexity designs. I think this would be a fascinating topic to explore further.

Another approach to reducing product complexity is through modularity. Modularity partitions the systems into less complex pieces and severs most of the interactions between them so that they can be designed and verified independently. Personally, I am a big fan of modularity. I championed modular satellites at DARPA, a modular smartphone at Motorola and Google, and a modular aircraft cabin at Airbus. The main downside of modularity is that it introduces overhead into the design. This explains why it is much more popular in software—where the cost of overhead is minimal—than it is in physical systems. With miniaturization and careful partitioning, modularity <u>CAN</u> buy its way into many more system designs. In addition to reducing complexity, it can also improve the upgradeability, maintainability, and other aspects of flexibility of the design. Unfortunately, all too often these “ilities”—which can be difficult to quantify—don’t make it into the design requirements. The requirements focus instead on performance and cost only, and incentivize the opposite of modularity: a tightly-integrated point design. Modular design does not require a whole new design methodology, although it can benefit from many of the things on the right. It requires only a different approach to requirements, such that the value of modularity is properly reflected in the design trades.

I think there is some fruitful work to be done on reducing product complexity. But I do not think that any of these approaches will be a panacea, at least not for aerospace systems that operate on the bleeding edge of what the laws of physics allow. The complexity of these kinds of systems is going to continue to grow. So the second category of things we can do it is change the design system to better manage complexity. These are on the right.

The first and most potent tool in this arsenal is abstraction. Abstraction has supported incredible complexity growth in the cyber domain. The software of a modern aircraft or weapons system is 3-4 orders of magnitude more complex than the software of the SAGE air defense system. The application of abstraction is not limited to software. The design of integrated circuits has sustained some 8 orders of magnitude increase in complexity from the first transistor-based CPU to a modern-day multi-core processor, largely through the introduction of several levels of abstraction in the design process. And yet abstraction has found no application in the design of mechanical systems. Today, a design engineer sees and “touches” every single component of an airplane, just as they did a century ago.

Design automation has also found widespread application in electronics. Certainly in integrated circuits, but also in the automated layout of printed circuit boards. There’s been a bit of work on design automation for structural elements and for the creation of new materials, but these techniques are not in widespread use. There is a lot of potential here, particularly with the introduction of AI techniques like adversarial learning methods which allow us to parts of the design space which we could never efficiently find before.

Automation of verification is a bit more common, with automated test procedures becoming standard on some classes of aerospace systems, again mostly electronics. It’s unclear if test automation is going to keep up with the growth in the size of the state space that needs to be tested. Other approaches, like proof-based methods, have not found their way to physical systems mainly because of lack of suitable models.

Automated design closure is closely related to the automation of verification, just a more ambitious version of it. Rather than just improving the productivity of verification, here we are talking about doing design closure quickly and on-demand any time during the design process. This would be revolutionary for a physical system. As I mentioned, today this is a manual and laborious process that typically happens a few times over the course of development, yielding a flurry of engineering changes. Doing it frequently would smooth out the profile of engineering changes and pull many of them much earlier in the development, when they are cheaper to implement.

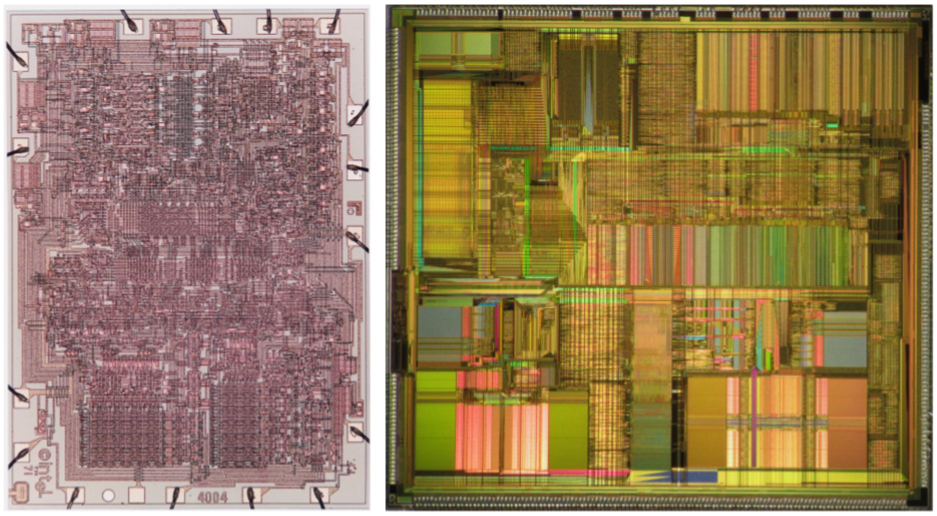
The last item on the list is design reuse. Of course any smart design engineer would use a prior design as a point of departure for a new one if they are similar enough. But here I am talking about black box reuse. And this we see only in highly modular designs. This is, of course, due to the difficulty of describing and guaranteeing the behavior of the black box at the interfaces, except in these very simple modular cases. If you could reuse pieces of any design, this would be a significant savings.

I mentioned that all of these approaches are borrowed from software. But as you heard me mention a few times now, they have also found great success in at least one class of physical systems—integrated circuits.

So I think it's worth spending a few minutes examining chip design to see what lessons we can extract for the design of mechanical—and ultimately cyber-physical—systems. The first integrated circuits were designed by hand and manually laid out. However, this quickly became impractical to do for individual transistors, so even early layout was largely done with blocks of transistors that comprised Boolean logic gates. As the transistor counts grew quite rapidly, so did the cost and time to develop new chips. Here it was the Intel 286 processor that broke the camel's back. So by the time Intel was gearing up to do the 386 in the mid-1980's, they adopted a new design methodology called Very Large Scale Integration (VLSI), devised by Carver Mead and Lynn Conway, which systematized the design process and added yet another level of abstraction on top of logic gates called Register Transfer Level (RTL).

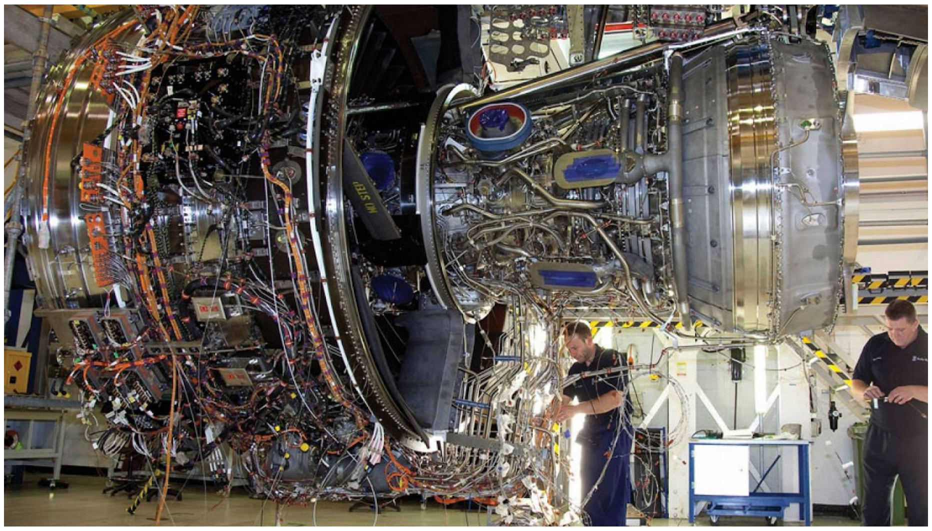
VLSI design required a new generation of design tools. These came to be known as Electronic Design Automation (EDA) tools. True to its name, EDA allowed designers to interact with the design at a very high level of abstraction, while automating the design synthesis at the lower level. The tools also verified that the resulting design met requirements, both in terms of its digital logic, as well as its analog byproducts such as heat and electromagnetic interference. The key underpinning of VLSI and EDA is a model library that fully describes the behavior of the low-level components from which a design can be composed and a set of design rules governing the composition.

Here is a visually illustrative comparison of a pre-VLSI chip design on the left. This is the Intel 4004, introduced in 1971. And it was designed using a manual process. On the right is a modern processor designed with a VLSI design flow and EDA tools. I think it's a fun juxtaposition.



Source: Intel

And by the way, here is what a modern jet engine looks like. It very much looks like a product of human ingenuity!



Source: Rolls-Royce

I should point out that there have been some very thoughtful critiques of the idea that the design of physical—meaning mechanical—systems can ever resemble VLSI design. Most notably, MIT's Daniel Whitney has a whole paper 20 years ago titled "Why Mechanical Design Will Never Be Like VLSI Design." And although I disagree with his categorical conclusion, I do agree with two key points that he raises.

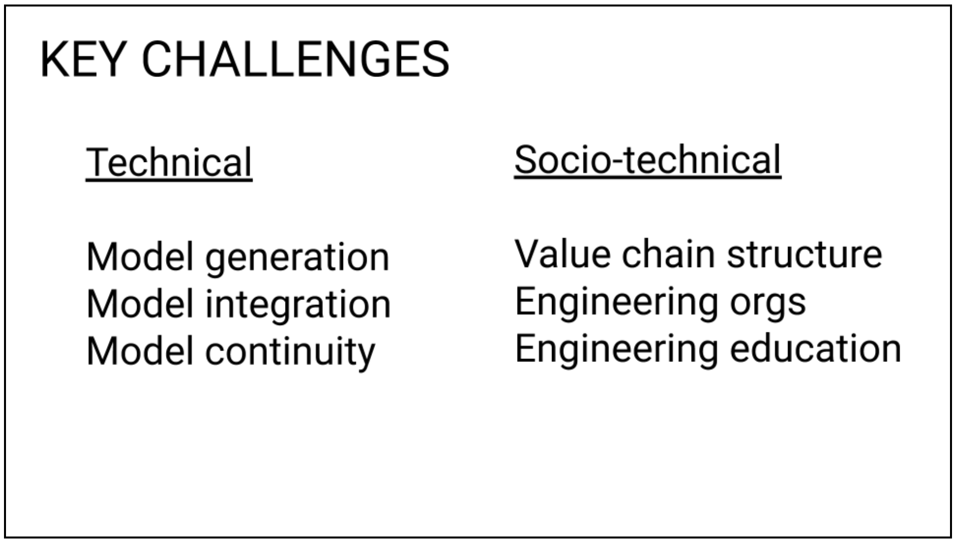
First, the behavior of mechanical components in a jet engine, for instance, is much richer and much more nonlinear than a transistor in an integrated circuit. For the transistor, you have to account principally for its electrical and thermal properties. For the mechanical component, you have additional considerations of its motion, deformation, and degradation. These must indeed be modeled, and the modeling challenge is certainly a big one, but nothing about it says that it's insurmountable.

The second point is that because the power levels and degree of integration are so much greater, components in mechanical systems cannot be modeled in isolation. Each component must incorporate the effects of other relevant components. That's very true, and I would further add that component models must also reflect the effects of the external environment. A microchip interacts only weakly with the environment around it, while an engine experiences not just airflow, but also the wing that it's attached to, weather, and the occasional bird. All of these must be modeled. So each component has its own model as well as a set of context models from adjacent components or from the external environment. Again, it's certainly a difference and a complication, but does not strike me as an insurmountable challenge.

Because it's really predicated on the primacy of behavioral modeling, I like to call this VLSI-inspired design methodology as applied to large-scale, heterogeneous, cyber-physical systems a "model-based digital design thread." It's a thread because there are many model-based design efforts and tools that tackle a specific point in the lifecycle of a product. And what I am describing is an end-to-end design process. In fact, it should extend through the manufacture and operation of the product as well. We might also consider calling it Newton's Revenge, since it would finally deliver on the dream that we've had since the 17th century, of having a full physics-based predictive model of the product before we ever build it.

I've tried three different times over the course of my career so far to develop this kind of design system. When I was at DARPA ten years ago, the META program which was part of an overarching Adaptive Vehicle Make initiative, aimed to do this for an Army ground vehicle. At Airbus, we launched the Digital Design, Manufacturing, and Services initiative which continues and strives to do this for the next clean-sheet commercial airplane. And at United Technologies, we started the Model-Based Digital Thread demo for a small jet engine, which Alberto Ferrari told you about in last year's keynote at this very meeting.

And so I would like to share with you some learnings from these efforts. In particular, here are three key technical challenges and three socio-technical challenges to implementing such a design system. I should note, by the way, that my own thinking on this subject was profoundly shaped by UC Berkeley's Alberto Sangiovanni-Vincentelli, who, having founded two of the first and to this day most prominent EDA tool vendors, can rightfully be called the father of EDA. Alberto's paper "Quo Vadis, SLD?" is a must-read on this topic.[2]


KEY CHALLENGES

Technical	Socio-technical
Model generation	Value chain structure
Model integration	Engineering orgs
Model continuity	Engineering education

Building a library of component models and context models is certainly the single biggest obstacle to a truly end-to-end model-based design system. It is expensive. It is time-consuming. It's never been done for something at the scale of an airplane. In VLSI design, the cost of the model library is amortized over a very large number of chip designs. New airplane programs happen infrequently, and the airplane makers treat each program based on an stand-alone business case. Some automotive companies have taken a consortium approach to model development. And aerospace does have one big advantage—because of the sophistication of the products, a lot of models already exist. They are scattered throughout the supply base, and model sharing and intellectual property will surely emerge as a significant issue.

[2] A. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System-Level Design," *Proceedings of the IEEE*, Vol. 95, No. 3, 2007, pp. 467-506.

Also a comment about model creation. Most of the successful modeling efforts I have seen are physics-based models that are calibrated and validated with experimental data. If I had a nickel each time someone asked me: "Aren't we are already over-reliant on simulation? Don't we need more test rigs?" It's not an either/or proposition—you need both! But instead of feeding test rig data into a spreadsheet, you use the data to tune a physics-based model.

The second technical challenge is model integration. How do you integrate thermal, electrical, mechanical, and other models such that you can operate across them and reason about the properties of the entire system? This one, I think, we convincingly demonstrated in the DARPA META program can be solved and solved at scale. The approach rests on over two decades of work by Janos Sztipanovits and many of his colleagues and students. Including, by the way, Ethan Jackson, who is the morning keynote tomorrow. Their key insight is semantic integration of existing domain-specific modeling languages; not requiring designers and modelers to change from the thousands (literally, thousands for an airplane, for instance!) of tools they are already using.[3]

And the final technical challenge I want to mention today is model continuity throughout the design process. High-fidelity models that you would want to use for design verification are not the models you would want to use for conceptual design space exploration. Another way to look at it is that you want to reduce uncertainty smoothly through the design process, until the final design artifact is correct-by-design and is assured to meet requirements if it's built as designed. Here the approach is typically to start with the high-fidelity models and apply some combination of model order reduction, linearization, or surrogate modeling in order to create the lower-fidelity models. Modern machine learning methods seem to be an especially interesting area to explore for surrogate modeling. But this is definitely a hard one and there are some juicy problems left to solve.

Finally, I'd like to say a few words about the socio-technical challenges associated with the creation and adoption of the model-based digital thread design system. First, on the industry value chain, who develops and owns the models is definitely up for grabs and could reshape the relationship between system integrators, suppliers, and tool vendors.

[3] J. Sztipanovits et al., "Model and Tool Integration Platforms for Cyber-Physical System Design," *Proceedings of the IEEE*, Vol. 106, No. 9, 2018, pp. 1501-1526.

Even more interesting is the relationship between design and manufacturing. As you will recall, in the craft epoch of design, there was no separation between design and construction. With the introduction of dimensioned drawings, the two separated, primarily so that the builders could be paid less. In the systems engineering era, the pendulum swung back a bit. Although design and manufacturing are two distinct professions, more often than not they live under one roof or at least in the same company. And although we frequently lament the seam between the two—like when engineers design a part that is difficult to manufacture—there is actually quite a bit of interaction between them. Handing over a modern aircraft design from engineering to manufacturing is a pretty engaged process. It is only in the lower tiers of the supply chain do you find pure manufacturing companies that will build to print.

It is not so in the VLSI world, where, with only a few exceptions, there is clear separation between fabless design and design-agnostic manufacturing foundries. This is not entirely without precedent in other industries. The automotive industry, for instance, has a few vehicle-level contract manufacturers. And Will Roper, the Air Force acquisition chief, has said that separation between design and scale production is something the Air Force is considering for its next-generation air dominance platform.

The last two items on this list are around the structure of engineering design organizations and how we train engineers, which are closely related. The key feature of the systems engineering approach, as we discussed, is the ability to partition complex systems. In theory, this partitioning could occur in any number of ways. In practice, it almost always happens along engineering discipline lines—aerodynamics, structures, power, controls, etc. The design team is organized accordingly. And most engineers are trained that way and rewarded for becoming experts in their discipline. Systems engineers, on the other hand, are trained in managing the design flow, but they are typically not designers.

Here I foresee the need for significant change. And it is change that will not come easily nor quickly. Certainly the disciplines corresponding to physics domains are here to stay. But they are likely to employ modelers, not designers, whose principal job will be to build domain-specific, physics-based models. Designers are likely to be generalists, as they will not need to have especially deep expertise in any one discipline. As in VLSI design, there will probably be a distinction between front-end designers, who are focused on the functional architecture of the product, and back-end designers, who are responsible for its physical realization, component placement, routing, etc. Today we have very few who think this way, and our engineering curricula are not designed this way. I highlight this for you because I know many of you are not just researchers, but also educators. And I see line-of-sight to the realization of the model-based digital thread design system at industrial scales in the next 5-10 years. But the socio-technical, and especially educational, issues have generational lag times. So I think they require increased emphasis and urgent attention.

With that, let me close and take your questions. I am also always reachable on Twitter at @PaulEremenko; if you'd like to have a more extended conversation on any of these topics, I welcome it.