Complexity in Design

Bob Colwell

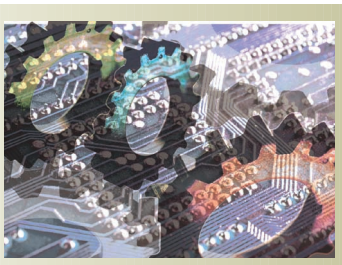
At its most basic, what engineers do is pit their intellect, training, experience, and intuition—and that of their design teams—against an implacable, relentless adversary: nature itself. I don't mean that an actual battle is going on; after all, only one of the would-be combatants even knows (or cares) that there is any contest.

When nature is the adversary, all that stands between the engineered product and disaster is the product designers' foresight, wisdom, and skill. Nature may not have maliciously chosen just the right wind pattern to demolish the Verrazano Narrows Bridge, but bridges must not fall down no matter how the wind is blowing. It may not literally be true that nature is conspiring against you, but it's a very good way to view the general engineering challenge. Forewarned is forearmed.

Students of engineering are first taught to "follow the rules"—guidelines for design that have proven over time to result in systems that behave as intended. Civil engineers learn about traffic patterns and human driving behaviors so they can design better roads. They study strengths of materials so that their bridges will be economical and reliable. When a bridge collapses, the rules are changed to incorporate the hard-won learning.

THE ART OF ENGINEERING

But that's not all there is to it. If engineers only had to follow a set of directions, we wouldn't need engineers; computers and robots can do that much. The real art of engineering, its sine qua non, is in evaluating a proposed design from every angle and vantage point to make sure a design will achieve its goals and prove reliable over its intended lifespan.



How well you handle your design comes down to how well you handle complexity.

When a simulation says a design is working, ask whether the simulation is correct and complete. If a formal proof asserts that some aspect of the design is correct, ask whether the proof itself is trustworthy. What makes you so sure the implementation technology will work as needed? What if the product specification itself has holes or blind spots? What if your product's buyers like it so much that they begin using it in ways you hadn't intended—can you anticipate that so the product will gracefully accommodate its new uses?

If the designer knows what she's doing, the design incorporates existing lore—only the desperate or suicidally naive would attempt a product with no familiar or known-trustworthy components—but it can't be based *only* on known components.

The nature of engineering is to never design exactly the same thing twice. Every new design pushes the envelope somewhere: performance, cost, reliability, features, capacity. Inevitably, some aspects of the new design will be outside the existing experience base. That's the part of engineering you don't learn in school. And assuming two competing design teams are technically proficient and reasonably well led, it's what these teams decide to do in these unknown areas that will largely determine which design ultimately triumphs.

So how do you handle the wilderness areas of your design, those places beyond your comfort zone and the safety of your tools and direct experience? I think the answer comes down to how well you handle complexity.

WHAT IS COMPLEXITY?

People who study chaos theory speak of a related complexity theory. They're trying to figure out why certain patterns seem to appear at many different levels of abstraction in natural phenomena; fractals are an example.

Scientists and engineers learn early that patterns often mean something important, so when the chaos theorists see the same pattern appear in seemingly unrelated places, they sense that there is something here worth investigating. However, I don't know if the way they use the word "complexity" is what I mean by that word. They adopted the term before I did, but I don't know of a better one, so for the sake of discussion, let's assume the chaos folks are chasing a different chimera than I am.

Even though you can't tell it by looking at the math they use, physicists strongly prefer simplicity in their models. Dissatisfaction with the "particle

Published by the IEEE Computer Society

zoo" that had accumulated over time led Murray Gell-Mann to look for a simpler underlying structure, which he named *quarks*.

Physicists know well the pattern by which an imperfect understanding of what nature is exhibiting often leads to a complicated, unsatisfying scientific model that a simpler, more fundamental model will eventually replace. The tendency for better knowledge to lead away from complexity and toward simplicity is a useful signpost for their intuitions (www.santafe.edu/sfi/People/mgm/complexity.html).

Another group that adopted the word "complexity" is wine connoisseurs. When they say a wine is complex, they mean they think it tastes good, and it strikes a good balance between all the different ways a wine can be measured. (Rich and deep, oaky with a hint of fresh bougainvillea blossoms after a rainstorm on a Tuesday. … Okay, I admit it—I don't know anything about wines, and I made that up. But they may as well say that as what they do say about wines, for all it helps me in selecting one. "Dry" wines still stain the rug.) I doubt that the wine experts have much to offer design engineers in terms of identifying and properly handling complexity, other than in much-appreciated liquid form.

A fourth group that I mention with considerable trepidation are the "intelligent design" proponents. I do think there are ideas here that can inform a design team's choices, if only we can distinguish the essence of the irreducible complexity argument from the emotional tar pit from which it comes.

Irreducible complexity

The basic notion of irreducible complexity is that, to achieve some end, certain physical systems must have a minimum amount of complexity. No isolated pieces of that system are very useful in and of themselves, but taken as a whole, they could achieve something very useful indeed.

I've heard human eyesight described in this way, for example: Without a clear cornea, nothing else matters. But if all you had was a cornea, without the light-sensitive rods and cones, you still couldn't see. And if you have a cornea plus light-detecting facilities, you still need a neural pathway to carry the information and some amount of mental processing with which to interpret it.

The intelligent design community uses such arguments to attack evolution, expressing incredulity that somehow all the necessary pieces could develop together when none of them are useful until the whole mechanism is complete. It's an interesting argument, and there are reasonably compelling counterarguments, but it's not my intention to take on this debate here. I mention it only because I want to borrow the idea of "irreducible complexity," and attribution must be given.

An engineer's implacable, relentless adversary is nature itself.

Quantifying complexity

You can't quantify complexity, but you can feel it. In fact, during a leading-edge design, if anything, complexity feels more real than many of the design's more quantifiable aspects. Your simulation tools can tell you a microprocessor's projected die size, but there's still time before the tapeout. Things happen, and as long as you believe that aspect of the design is on track, then for today it's mainly a theoretical concern.

Performance and power dissipation feel that same way. But complexity is a monster you can hear breathing right outside your cubicle. It whispers your name during planning meetings, but if you're not paying attention you may not hear it. Later on, you find yourself at the moment of truth in a project, suddenly realizing that things aren't right and it's by no means clear if there *is* a way to salvage them, let alone what that way might be.

Complexity factors

There are some hallmarks to complexity that I've noticed over the years. I believe design complexity is a function of the

- number of ideas you must hold in your head simultaneously;
- duration of each of those ideas; and
- cross product of those two things, times the severity of interactions between them.

For instance, all else being equal, a deep microarchitecture pipeline carries more intrinsic complexity than a shorter one. Different activities occur at each pipe stage (otherwise why does that pipestage exist?), and each activity can take more than one clock cycle. If a pipestage doesn't exist, it can't interact with any others; conversely, if the pipestage does exist, you must consider those interactions and make sure they're all congruent with the project's goals.

For nominal pipeline operation, this kind of analysis is difficult but doable. But for exceptional conditions, such as reset, fault, interrupts, test scan, performance counter interactions, power-downs to lower thermals, breakpoints, and stalls, the degree of intellectual difficulty can reach nightmarish proportions.

Project unknowns

This definition of design complexity makes it a little clearer why project success is so sensitive to unknowns. The very fact that these unknowns are, well, unknown, means that they could inject a wide range of behavior into the design. That uncertainty range increases the number of ideas you must simultaneously consider, and it might also increase the predicted time durations needed.

It doesn't take much uncertainty to make the complexity-derived behavior range too big to mentally handle. And

this is perhaps the most insidious issue of all: When faced with a complex situation, you generally know that you aren't yet in command of all the necessary details. This in itself isn't alarming; it takes time to understand what hundreds of people are doing or intend to do. But you can't leave it like that.

One option is to insist that all of the unknowns be researched and quantified so that your spreadsheet can tell you what to do. But this scheme doesn't work. Your project doesn't have enough time or idle engineers to do this much new work; besides, not everything you want to know about your project is knowable, much less quantifiable.

Complexity exacts a toll in many ways. In my January 2004 At Random column ("Design Fragility," pp. 13-16), I argued that complex designs are more fragile and lead to more surprises (which are always bad). Complexity leads to longer development schedules; it directly causes design errata; it fosters suboptimal tradeoffs between competing goals; it makes follow-on designs much more difficult; and it's cumulative, with new designs inheriting all of the complexity of the old and with new complications layered on top.

Increased project complexity also shrinks the available pool of engineers who can help when things go awry. Any competent designer can make easy, straightforward choices, but for truly gnarly situations, only wizards will do. And there are never enough wizards.

Stanford's John Hennessy once said that it's always possible to design something so complicated that you can never get it right. As a project leader, are you smart enough to mentally absorb the remaining project unknowns on top of what you already know to the extent necessary to make good decisions on any remaining tradeoffs (some of which haven't even surfaced yet)?

To some extent, everyone on the design team has this same problem. In considering a design's overall complexity, you're making the "smart enough" choice on their behalf as well. If, after deep reflection, you believe you have a practical grasp of the project's complexities, with enough margin to handle the usual surprises downstream, great. Today will be a good day and you won't have to think about this again until next week, when you must confront the same issue again. But what if you decide that things seem to have gotten a bit too close to the edge? What do you do then?

It's always possible to design something so complicated that you can never get it right.

WHY NOT JUST SIMPLIFY EVERYTHING?

Why not just make everything simple? As Einstein said, everything should be as simple as possible, but no simpler.

Have you ever taken a close look at an acoustic grand piano? The complicated mechanism in which the player's fingers cause hammers to fall on strings, which are damped to varying degrees and at varying times, is exquisitely subtle and quite involved. Perhaps there's a way to simplify that mechanism without sacrificing the combination of tone and playability that has selectively evolved over hundreds of years. But it seems likely that most of a piano's complexity is necessary because of how our fingers, feet, and ears work. A skilled performer can coax thunderous waves of sound or delicate susurrations; the mechanics of the keys have the requisite dynamic range. This complexity works.

A simple bow and arrow can be easily made; if the materials and workmanship are good, a serviceable weapon can be achieved. But there are other variations on that theme that work better—a compound bow using pulleys achieves much higher arrow speed at less draw weight, but at the cost of considerably higher complexity in its design. For what it achieves, that complexity is well justified.

It's also true that what initially looks impossibly complicated may seem quite reasonable after a period of study and experience. In part, you just get used to it; you find that you can remember the various pieces, their roles, and how they interact. You also find ways of thinking about what those pieces do, mental models that you internalize—thus they constitute no tax on your cognition.

I remember how thunderstruck I was in college after struggling with some formidable mathematical concept only to have a fellow student point out how simple it was when viewed in the right way. This familiarization effect must also be considered when you're judging how much complexity a design project should include.

A design's complexity must serve a project's major goals. If your design is complicated but coherent, challenging but understandable, you may have struck a good balance between irreducible complexity and the project's goals.

Strive to avoid creeping complexity, the kind that arises from unintended interactions among multiple unrelated design decisions. Eschew complicated machinery that isn't clearly and provably necessary to attain one or more of the major project goals. If you aren't sure you need some logic in your design, keep asking questions until someone either justifies it or agrees to toss it overboard.

Entropy always drags a project in the direction of increasing complexity; things never get simpler on their own. Except for bad wine. On Tuesdays. ■

Bob Colwell, *the 2005 recipient of the IEEE Computer Society/ACM Eckert-Mauchly Award, was Intel's chief IA32 architect through the Pentium II, III, and 4 microprocessors. He is now an independent consultant. Contact him at bob.colwell@comcast.net.*