You’ve committed a dangerous act by printing Grady Booch’s article “The Irrelevance of Architecture” (May/June 2007). Way too many people already feel that architecture is irrelevant, and they’ll be tempted to read the misleading title, skip the article, and forever use Booch as proof of their erroneous beliefs about architecture.

Additionally, I should admonish Booch for the invalid assumptions and errors of omission his article propounded on the way to reaching an otherwise good conclusion—namely, that architecture is quite relevant. A better conclusion would have been that a proper systems architecture is vital, desires of the agile cowboy coding hackers notwithstanding.

One always has an architecture. It might have been done explicitly; otherwise, it’s ultimately defined by whatever the developers came up with. It all depends on which end of the dog is wagging the other. And this is not some Big-Endian vs. Little-Endian religious dispute (see Danny Cohen’s article “On Holy Wars and a Plea for Peace” in the October 1981 issue of Computer). When it comes to dogs, the tail should be the thing wagged, just as a design should be dictated by an architecture.

And architecture is relevant to the end users. Without starting with a proper systems architecture, the end result won’t serve users well. They might not care about the implementation details, but they do care that the interface and functionality suit their needs when delivered.

Although Booch observes the generic relationship between architecture and implementation, he omits the details that let software-oriented people persist in their delusion that software is the system and that software architecture is synonymous with systems architecture—and that they can consider their part of the system without regard for the whole.

Software is but one component supporting a systems architecture. To optimize the entire system, the systems architect must also consider the as-yet-undefined supporting architectures for hardware, communications, information, and security, as well as others that can impact the system, such as those for processes, procedures, and maintenance.

All the supporting architectures must be balanced. The fundamental theorem of systems states that you can’t optimize a system by optimizing the subsystems, yet many people continue to make that mistake.

Software types are outstanding examples. Who hasn’t suffered from delays and added expense when some cowboy coder made an unapproved, unwanted, and unnecessary “improvement” that delayed testing and integration as well as breaking code?

For the designer, the systems architecture is the keystone that defines success. The other building blocks the designer must rely on are the domain architectures, high-level domain designs, detailed designs, and designs for the smallest buildable thingie, no matter what that domain names it.

A critical part of any systems architecture is defining the nonfunctional requirements. As Barry Boehm showed in Computer magazine (“Unifying Software Engineering & Systems Engineering,” March 2000), these nonfunctional requirements drive both the feasibility...
and quality of the end system, as well as schedule and cost. Systems often fail because the nonfunctional requirements were arbitrarily promulgated or were merely erroneous. Ignoring or omitting them leads to the same results, often with lawsuits based on misunderstandings of what was actually built versus what was wanted.

What the systems architect defines essentially become the requirements that are used as input for the domain architecture’s definition. In reality, these are just strong guidelines, which the systems architect can modify to improve the overall system. But, they can’t be unilaterally changed (or “improved”) by domain architects, engineers, or designers without severe risk of harming the total system.

Naturally, the only real requirement is to solve the problem or improve the situation depending on the initial motivation for change. And there may well be a range of feasible solutions.

The systems architect, having considered the full range of alternatives, has to make a decision in order to move forward. So, for all practical purposes, the systems architecture, while really being a specification of the selected solution, is the set of requirements for the domain architects to use; they would have had ample opportunity to suggest improvements and note trade-offs and impacts on their domain during the development of the systems architecture.

Ultimately, for the designer, the systems architecture will be refined into domain architectures, with inputs from the chief domain systems engineers. These domain architectures will in turn be used to define high-level designs for each domain. We’ll repeat this decomposition process, addressing ever-smaller building blocks, until we’ve defined a buildable device or module.

Unless the developers build the exact things specified, their deviations or shortcomings can cause major problems elsewhere to the detriment of the total system, as well as often ensuring total failure. Consider the butterfly effect, and also recall the grade school lesson about the lack of a nail losing the war.

This isn’t a waterfall process, it’s helical. It ensures that optimization will occur by a heuristic dynamic programming process and that some undisciplined improvements—which may or may not be of benefit locally but which can be disastrous to the overall system—won’t cause unintended consequences.

Success depends on the skill of the systems architect and on how well management controls the execution of the development.

Projects fail for several reasons. Lack of reality often dooms them, and the good-fast-cheap mentality is erroneous in that it omits risk and completeness of the solution as measured by functionality desired. Lack of sufficient resources also ensures partial if not complete failure. And finally, the lack of a proper systems architecture, including the critical nonfunctional requirements, dooms projects to failure.

William Adams
williamadams@ieee.org

Grady Booch responds:

A dangerous act? Happily, when I work up the morning after the issue hit the streets, the sun was still shining, most major civilizations were still relatively intact, and my cat was purring. It was a good day. I do respect, however, that you’ve taken exception to my article. Indeed, you raise some valid points.

You note properly that “way too many people already feel that architecture is irrelevant.” I agree, but if they take me out of context in support of their view, then they’re fools if they quote me thus, and their ignorance will only be amplified by a citation that is so vacuous. Unfortunately, there’s no law of conservation of stupidity in the world; this is a property that seems to be increasing exponentially. Alas, no amount of software will deter us from that. I’m glad, however, that you seem to agree with my conclusion, although I respect that you may disagree with the manner in which I got there. I suppose it’s like an editor changing the phrase “it was the best of times, it was the worst of times” to something like “some times were better than others.”

You do suggest that “a better conclusion would have been that a proper systems architecture is vital.” Well, it depends on what you call a system, but I wouldn’t disagree. John Gall’s classic Systemantics (General Systemantics Press, 1986) makes a compelling case for your position. I also agree that “one always has an architecture.” As I’ve often said, every system has an architecture; some are accidental, others intentional. Furthermore, any architecture is relevant to a variety of stakeholders (not just end users), each of which come to that architecture with a unique perspective and viewpoint.

Continuing, you suggest that I encourage the delusion that software architecture and systems architecture are equivalent. They most certainly are not, and if I’ve given that impression, then I clearly have erred in my presentation. You later suggest that “the systems architecture is the keystone that defines success.” I somewhat agree, but counter by observing that, in my experience, focusing on a system’s architecture is a necessary but insufficient condition for success. I must also observe that, for sufficiently complex systems, you can’t really know its final architecture until the level of risk has become tolerable. The mere presence of a system disrupts the problem space itself, and thus there are questions and details of a system’s architecture that you simply can’t know a priori for novel systems until elements of the system have been built.

You made an intriguing comment in passing, in which you said “there may well be a range of solutions that are feasible.” Indeed, one of the things that I’m trying to understand with the handbook (www.booch.com/architecture) is if certain architectures are an accident of history or if a given solution space naturally converges to an envelope of possible satisfactory architectural solutions. I simply don’t know.

We agree on so many other levels: systems development isn’t a waterfall process; the architect’s skills contribute greatly to a system’s success; the absence of a focus on a system’s architecture is often a predictor of eventual failure.

And by the way, my cat is still purring, happily ignorant that the software-intensive system on which I compose this message even has an architecture. ☺️