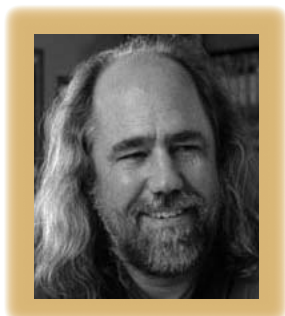


Speaking Truth to Power

Grady Booch

Whenever I conduct an architectural assessment—well, really, I try to apply the following principle in all my dealings—I endeavor to speak truth to power: those with true power never fear the truth.

That being said, sticking to that precept has gotten me kicked off at least two projects. In one case, I'd suggested to management that they simply cancel their project because it had a corrupt architecture and a dysfunctional process that were beyond repair. They eventually did cancel the project, but only after they had spent several more tens of millions of dollars of taxpayer money. In the other case, my recommen-



dations were clearly contrary to the project manager's political aspirations, so my papers were buried and I was shuffled out the door. Rumor has it that this project was also later canceled, but not before the manager in question moved up the ladder, leaving the morass and the resulting blame to his successor.

Sam Guckenheimer has observed that in code there is truth. Code represents the stark reality of a software development organization's labor, and no deluge of PowerPoint slides, marketing spin, or glossy packaging will hide that reality. There is also truth to be found in a system's architecture, a truth that's at the same time

- fundamental in the sense that a system's architecture is a manifestation of the significant design decisions its developers have made to shape the system, and
- emergent in the sense that a system's architecture reflects patterns that transcend indi-

vidual lines of code and give intellectual integrity to the as-built system.

An aside: Be gentle

Before I go on, I should offer a bit of practical advice to those of you who wish to follow my lead. Be careful not to speak the truth too harshly or too personally. If you tell someone that "your architecture is like a rotting pile of fish guts and furthermore your children are ugly," you might indeed be speaking the truth but you won't be an effective agent of change. My style tends to be far more gentle and a bit Socratic.

Complexities and contradictions

Who needs to hear this truth? To answer that question, let me take a slight diversion that will actually set up the context for an answer.

In preparing for some lectures in London, I studied several of Alan Turing's papers and some biographies of him. Turing was a complex man—I suppose that all pioneers are complex in their own way—and full of contradictions, an enigma, as his biographer Andrew Hodges put it. Perhaps the ultimate contradiction of his life was that although his cryptanalysis work helped defeat Adolph Hitler's oppressive forces, the society he helped preserve oppressed him in his personal life (as a homosexual, in an era when society fiercely condemned such behavior).

This contradiction led me to realize that, despite all the promise that software brings, its presence carries its own set of contradictions. For example, the Internet changes the way individuals communicate and businesses collaborate, but it also gives terrorists and other criminal enterprises an environment to operate in with little fear of detection. The Web provides

unprecedented mechanisms for social networking; it also makes it easier for criminals to exploit children as well as defraud and steal from individuals and organizations. Software-intensive systems permit real-time and distributed information access, but they can erode personal privacy and other basic human rights. Email and other software-intensive mechanisms increase the velocity of communication; yet email and the aging of digital archives threaten the preservation of history. Software-intensive systems create new forms of artistic expression, whereas piracy disrupts the economic underpinnings of traditional media companies and can dilute artists' intellectual property. Such systems enable and accelerate scientific research, but they're also at the center of a new generation of offensive and defensive weapons. Software is part of the very fabric of civilization, living in its interstitial spaces. It continues to grow more complex, affecting its users as well as the stakeholders who develop, deploy, operate, and evolve it.

Telling the truth

The stakeholders who have an interest in a software-intensive system are many, varied, and sometimes unexpected. The most obvious collection encompasses those who build the software itself; this includes the system's architects, project manager, code warriors, analysts, and testers. The most important group covers those who use the software, including its end users, third parties who extend or integrate other systems with the system, and its administrators. The not-quite-so-obvious-but-still-very-important stakeholders include those who fund the system and its other patrons. Beyond those orbits, we move out to more metaroles. As my list of contradictions suggests, any significant piece of software touches many classes of people, and they are impacted by and can impact software-intensive systems. Historians, legal professionals, criminals, artists, companies, and governments might all be drawn in by the gravitational pull.

The closer we are to a system's code, the more visible its architecture. A code warrior toiling away to introduce a new feature into that system will need to do

so in the context of its architects' design decisions. Someone maintaining a piece of software must honor the system architecture or it will slowly rot away. A third party who wishes to extend a system must understand where its edges lay and have some sense of its underlying architecture to properly and cleverly use the resources it offers at the edge.

End users, for the most part, want software to be invisible. However, even for them, some essence of a system's architecture will emerge, because that architecture shapes the way users see the system and provides a model that enables users to interact with the system. At the outermost orbits, where all the unexpected stakeholders live, the fact that a software-intensive system even exists is irrelevant because all they really care about is the system's impact on their lives.

In *IEEE Standard 1741, Recommended Practice for Architectural Description*, a clear separation of concerns is made among stakeholders, their issues, their viewpoints, and the system's architectural description. According to the standard, we should organize the description of a system's architecture by a set of views, each of which consists of a set of models and conforms to a particular viewpoint. In turn, the set of concerns that are important to certain stakehold-

ers determines that point of view.

Every system's architecture is molded by the forces that swirl around it, and the collective concerns of all the stakeholders represent the most dynamic forces shaping a system. (Other forces include limiting factors around the laws of physics and the laws of software.) The software development organization's unique task is to address all the essential concerns of all the important stakeholders and to ensure that they aren't blindsided by unexpected problems and stakeholders. This is why employing a process that incrementally and iteratively grows a system's architecture through the regular release of testable executables is so important. Such a process lets the software team engage the right stakeholders at the right time and to make the right decisions, neither too early nor too late.

In creating a software-intensive system that's both relevant and beautiful, every stakeholder, no matter how close or how far from the code, deserves the truth. ☺

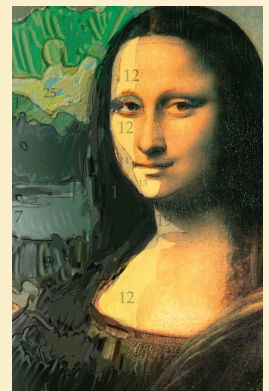
Grady Booch is an IBM Fellow and one of the Unified Modeling Language's original authors. He also developed the Booch method of software development, which he presents in *Object-Oriented Analysis and Design*. He's now working on a handbook of architectural patterns, available at www.booch.com/architecture. Contact him at architecture@booch.com.

Master your software—look for these future topics:

- Test-Driven Development
- Software Patterns
- Rapid Application Development with Dynamically Typed Languages

Visit us on the Web at

www.computer.org/software



IEEE
Software