well-defined module interfaces and that are inherently testable, so that the construction progress is transparent and visible.

Next, we want architectures that demonstrate Persistence—that is, architectures that have stood the test of time. We work in an era when the technical environment is changing at an ever-increasing rate. A beautiful architecture should anticipate the need for change, and allow expected changes to be made easily and efficiently. We want to find architectures that have avoided the "aging horizon" (Klein 2005) beyond which maintenance becomes prohibitively expensive.

Finally, we would want to include architectures that have features that delight the developers and testers who use the architecture and build it and maintain it, as well as the users of the system(s) built from it. Why delight developers? It makes their job easier and is more likely to result in a high-quality system. Why delight testers? They are the ones who have to attempt to emulate what the users will do as part of the testing process. If they are delighted, it is likely that the users will be, too. Think of the chef who is unhappy with his culinary creations. His customers, who consume those creations, are likely to be unhappy, too.

Different systems and application domains offer opportunities for architectures to exhibit specific delightful features, but Conceptual Integrity is a feature that cuts across all domains and that always delights. A consistent architecture is easier and faster to learn, and once you know a little, you can begin to predict the rest. Without the need to remember and handle special cases, code is cleaner and test sets are smaller. A consistent architecture does not offer two (or more) ways to do the same thing, forcing the user to waste time choosing. As Ludwig Mies van der Rohe said of good design, "Less is more," and Albert Einstein might say that beautiful architectures are as simple as possible, but no simpler.

Given these criteria, we propose some initial candidates for our "Gallery of Beautiful Architectures."

The first entry is the architecture for the A-7E Onboard Flight Processor (OFP), developed at the Naval Research Laboratory (NRL) in the late 1970s, and described in Bass, Clements, and Kazman (2003). Although this particular system never went into production, it meets every other criterion for inclusion. This architecture has had tremendous influence on the practice of software architecture by demonstrating in a real-world system the separation of a design-time Information Hiding Module and Uses structures from the runtime Process Structures. It showed that information hiding could be used as a primary decomposition principle for a complex system. Since the U.S. government funded and developed the architecture, all project documentation is available in the public domain.† The architecture had a well-defined Uses structure that facilitated incremental construction of the system. Finally, the Information Hiding Module structure provided clear and consistent criteria for decomposing the system,

† See, for example, Chapters 6, 15, and 16 in Hoffman and Weiss (2000), or conduct a search for "A-7E" in the NRL Digital Archives (*http://torpedo.nrl.navy.mil/tu/ps*).