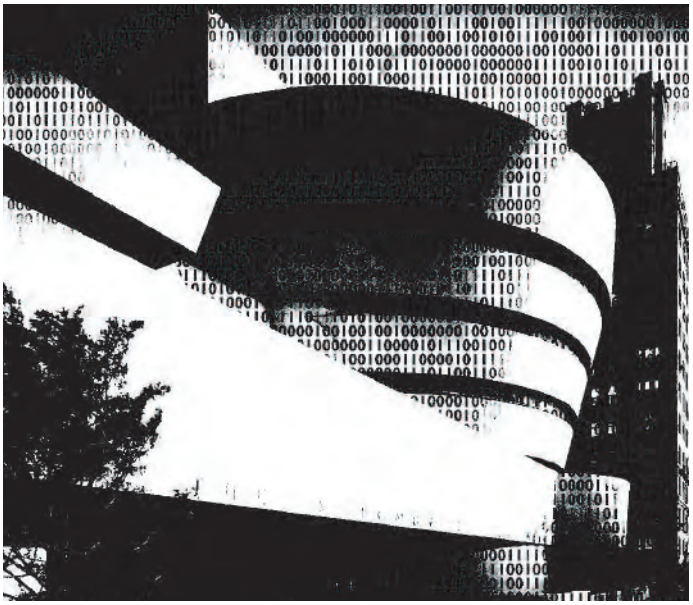
Design? We Don't Need No Stinkin' Design! (or "How to Fail Without Really Trying")

Let me start by pointing out that I am a compulsive list-maker and planner. As a military brat, I grew up overseas. Recently, I decided to take a vacation to Istanbul, Turkey where my father had been stationed from 1964 to 1966. To me, a large part of the fun of a vacation is the preparation and planning. Making lists – what to carry, what sites to visit ranked by importance (with categories of Essential, Important, or Optional) – is exhilarating. To me, half of the fun is examining all the tourist guides, and savoring the experience of the trip by preparing for it.

When I showed my co-workers my list of sites I wanted to visit, sorted both by district and importance, the word *geek* kept creeping up in their responses.



On the other hand, these obsessive-compulsive traits seem to be a good thing when talking about design.

When you mention *design* to most developers, they start thinking of program design. Now program design is indeed one part of design, and a very important part. Program (or module) design is where you convert the algorithms and logic of the design into executable code, assuming that the algorithm and logic were available – assuming that the developer doesn't know more than the designer – and *improve* the logic.

Module design is the easy part of design, however. Design, depending upon the source and reference cited, consists of at least four phases: architecture design, interface design, data design, and module design. This ordering also captures the correct order of importance and difficulty.

Architectural design is where a *grand designer* sets forth the vision of the system. It includes what the major subsections are, the major functionality, and general *feeling* of how the overall system is going to work. Even with automated tools, this step is hard. It is typically said that the architectural design should have a feel to it that says, "Well, of *course* this is the way that the system is supposed to look." It should appear simple. Unfortunately, as system designers know, simplicity is very difficult to achieve. Without a good architecture, however, the system never seems to work well – tasks have to be shared across major subsystems, increasing complexity, and decreasing cohesion.

As part of the architectural design, interfaces have to be established and enforced. In a perfect world, this step should be relatively simple. Unless, of course, you are dealing with legacy systems that already have existing interfaces; interfaces that were, at best, designed using 30-year-old functional decomposition, and do not fit well with object-oriented languages of the present.

When I teach design, I tell my students that 75 percent of all errors will eventually be traced to poor interfaces. Most don't believe me at first, but eventually find out that I was right. The problem with interface incompatibilities is that they often don't show up during unit testing, and cannot be discovered until integration testing. Of course, the later in the life cycle that errors are discovered, the more expensive they are to fix.

Do the words "Boyce-Codd Normal Form" mean anything to you? How about "First, Second, or Third Normal Form"? If so, you're in the minority. These terms refer to relationships between data and keys. Twenty years ago, you wouldn't have dreamt of designing a database without having an expert determine an optimum arrangement of keys and relationships. Now, unfortunately, cheap disk storage and fast processors allow data to be *thrown together*. However, as applications evolve, grow, and are modified due to changing and new requirements, bad data design can slow down your application. It can also contribute to problems with duplication of data, along with consistency and correctness of the data.

In short: design is hard, very hard. The larger and more complex the application, the more important design becomes. If you have a small application that's going to exist on the Web for a few months at the most, design is no problem. On the other hand, if you have a $10 million project, projected to interface with existing Department of Defense systems with an expected life of 20 years, design is critical.

Of course, you can probably get a working system without much design. Enough to get user sign-off. Of course, the first time you try and make any modifications or additions to the system, the lack of a design will cause it to fall apart like a house of cards.

Not to worry. There's always time and money to do it over with a good design, right?

— David A. Cook, Ph.D.
Senior Research Scientist
The AEgis Technologies Group, Inc.
dcook@aegistg.com