

## **What makes a good engineer?**

**By Don Rowe**

**Revised Nov. 30, 2005**

**Canzona Technologies**

**www.canzonatech.com**

### **Make it happen, or wonder what happened**

An important part of engineering is proving something will work before it's built. Would you drive across a new bridge if the chief engineer said, "This bridge is sure pretty, but I don't know how many vehicles it can support. Lets send a bunch of cars across and see what happens. If we notice any cracks, we'll slap on some duct tape." Part of the engineer's job is to design the bridge to safely support the expected vehicle weight, and withstand expected earthquakes, winds and water levels. Civil engineers "prove" a bridge design with calculations based on anticipated stresses, strength of the materials, and construction techniques. Despite the engineers best efforts, bridges still occasionally collapse, possibly due to extreme stress beyond what could reasonably be anticipated, defective materials, construction "shortcuts", or long term wear with inadequate maintenance.

### **Anticipate and prepare, or wait and react**

Careful carpenters say "measure twice, cut once". Good engineers think things through twice, and design once. Those who don't often end up like the reckless carpenter who laments, "I've cut it three times and it's still too short."

### **You want it when?**

Software engineering is especially easy to do badly, and quite difficult to do well. It's tempting to create folders of source code files, and then "try it and see what happens". If it crashes, just patch it with the software equivalent of duct tape, or the latest industry/management buzzword. Oh well, everyone expected it to be late anyway, and customers are used to bugs and feature-challenged first releases.

### **Is next month OK?**

Good software design is a process of continual reengineering. The complexities of most programs make it very unlikely that an engineer can fully comprehend everything at once. The first round of "calculations" might be like an overall outline, showing which components are needed to get the job done, and how they will fit together. A more refined analysis should consider which components are totally new to this project and are most likely to contain unintended consequences or unanticipated limitations. Refining and testing these critical sections should be followed by reconsidering the overall structure. It's also important to consider which sections might slow down overall performance, and design those sections as separate modules that can be optimized as needed without changing much else. This process repeats, working through the various levels, while continually reevaluating the overall design. The design evolves quickly because "surprises" are found and fixed early, as are major changes to the overall design. The final program will be flexible, modular, easy to understand and flexible enough to accept enhancements to one part of the program without causing problems elsewhere. Oh, and by the way, it will be ready on time and a pleasure to use.

### **Testing – It's not just for customers**

Testing isn't "watching it work". Meticulous testing should be part of the design process and includes methodically seeking out problems by identifying likely bug habitat. More complexity requires more thorough testing. Subjecting the code to "stress tests" by considering unlikely scenarios and alternate paths through the code helps identify problems before the official testers try it out. Testing by others who may know nothing of the program's inner workings can be helpful, but the testers need guidance so they

don't just keep repeating the same successful tasks while overlooking more obscure features or combinations that may never get tested.

### **The quick fix**

Despite everyone's best efforts, problems occasionally sneak through. A well designed program is easy to diagnose and fix without creating new problems elsewhere. The fixing process is just an extended design cycle, and should be approached the same way. There are only so many problems lurking inside the code. Each one found and fixed puts the design one step closer to perfection.

### **Make all the improvements you want to – just don't change anything**

Sometimes after extensive changes to the inner workings of a design, it becomes clear that a high-level reorganization would provide significant benefits. These high-level changes take time, but not making them usually leads to even more low-level changes and added complexity. The total time to a fully functional design is often reduced after doing the necessary cleanup throughout the program, and future changes and fixes will go much quicker. A thorough house-cleaning may also eliminate other mysterious, nagging problems that previously defied all attempts at a fix. An occasional review may also lead to new insights and potential improvements.

### **Poetic license**

Is engineering art or science? Grammar or poetry? Meticulously following the rules may produce a sturdy but ugly bridge. A fanciful design might look beautiful, but collapse in the first breeze. Good engineers understand both concepts, and seek new applications for traditional rules and develop new rules to prove an innovative design. Although software doesn't have a visible physical structure, similar principles apply to the overall program organization. Like the good civil engineer who designs a sturdy bridge that's truly beautiful in its elegant simplicity, a good software or hardware engineer finds simple, fundamental relationships between each part of the design. Discovering these simple, but often hidden relationships helps good engineers develop a flexible "toolkit" that not only speeds the current design, but provides reliable components for the future. The ability to seek and discover these "hidden" relationships is a skill that applies to any technology and often leads to simpler designs. Simplicity often mean fewer "parts", which means fewer potential problems, and designs that are smaller, better, faster and cheaper.

### **Been there, done that**

Experience is important, as long as each experience involves more than simply repeating the same mistakes. Like good engineering, experience has multiple levels. Understanding a particular component or technique may be relevant to future designs, but understanding the underlying principles can be even more important. Many components and techniques are "variations on a theme", or simply combine things that were previously independent. Understanding the underlying concepts usually leads to a very short learning curve when using something new. The reverse is seldom true. A solid understanding of the fundamentals become extremely important when a problem occurs, or the design is too slow, too big, or too expensive. A good engineer will often find a way to rearrange things, or tweak one part of the design without creating new problems elsewhere.

### **Haven't been there ... yet**

The best engineers are problem solvers and innovators. It can save some time if they've already solved exactly the same problem, but the latest problem may be different in some way. Good engineers may find a solution based on similar experiences, or invent a totally new solution. They look at the "traditional" capabilities of a system or component, and see new, untapped potential. The "grammar" of engineering is most easily learned. A good engineer may need only a few days to learn about a new component or technique, and has the ability to apply fundamental technical knowledge within the context of an overall design. Learning the "poetry" of engineering requires much more time and effort, but allows good engineers to incorporate new concepts into original and elegant solutions.