

Watts New

Learning from Hardware: Design and Quality

Watts S. Humphrey

In the September column, I discussed what we could learn from hardware engineering, particularly about planning. Hardware engineers have developed a family of planning practices that they have used with great success. The prior column reviewed the pros and cons of using these planning methods for software work and discussed how these methods could help us meet our businesses' needs.

It is no fun to be late, to have unhappy customers, and to be unable to predict when you will finish a job. By following sound engineering planning methods, software work can be more productive, more predictable, and more enjoyable for the engineers themselves. In this column, I continue this same discussion with a focus on how engineering quality practices and design principles could be adapted to software and what we might gain from doing so.

Hardware Costs

Hardware development strategies are dominated by cost considerations. The principal costs are those the factory incurs in producing the products, as well as those the service organization expends in handling product warranty and repair work. Even though we don't need a factory to produce volumes of software products, we do have large and growing service costs and we can learn a great deal from the ways in which hardware engineers have addressed quality and design problems.

The Design Release

One of my first jobs when I joined IBM some years ago was to manage the development and release-to-manufacturing of a hardware product. We had built a working model and had complete parts lists, assembly drawings, and component specifications. While I thought we had a complete story, the manufacturing and service groups put us through the ringer for two exhausting days.

It took me a while to realize why they were being so difficult. They would not accept the release until we convinced them that our design provided the information they needed to meet cost, schedule, quality, and production volume commitments. Once they accepted the design release, these manufacturing and service groups would be committed to producing, warranting, and repairing these products on a defined schedule, with specified product volumes, and for the estimated costs. Their ability to do this would determine whether or not IBM made money on the product.

Since manufacturing and service were the two largest direct cost items for IBM's hardware products, these groups had learned how to manage costs and they were not about to accept a release that had potential cost problems. While the manufacturing and service people were hard to convince, they imposed a valuable discipline on the development engineers. By making us produce complete, precise, and clear designs, they motivated us to think about manufacturing and service quality during design. This release discipline provided a solid foundation for all of the subsequent hardware quality improvement programs.

The Need for Precise and Detailed Designs

Most hardware engineers quickly learn the importance of a precise and detailed design. On their very first jobs, they learn the difference between designing a laboratory prototype and releasing a design to the factory. Manufacturing groups will not accept a design release unless it provides the information they need to define the manufacturing processes, estimate the costs of production units, predict cost as a function of production volume, calculate warranty and service costs, order and fabricate all of the parts, and assemble and test the system. Many people need the design information and it is essential that they all get precisely the same story. Design documentation is also essential to enable the inevitable design changes and to track and control these changes.

The Need for Documented Software Designs

The need for precise and documented designs in software is both similar to and different from hardware. There are five principal reasons to document a software design:

- to discipline the design work
- to facilitate design reviews
- to manage change
- to preserve and communicate the design to others
- to enable a quality and cost-effective implementation

Some people can hold very complex designs in their heads. However, regardless of how gifted you are, there is some upper limit beyond which you will no longer be able to do this. When you hit this limit, your design process will fail and the failure will not be graceful. Even very complex designs are not beyond our mental capacities when we follow sound and thoroughly documented design practices. Then we will not face the design crises that often result in complete project failures.

By documenting your designs, you also facilitate design reviews. This will both improve the quality of your designs and improve your personal productivity. The connection between quality and productivity is easy to see in hardware because a major redesign generally results in a lot of scrapped hardware. For software, however, these scrap and

rework costs are equally significant, although not as visible. In the simplest terms, it is always more productive to do a design job correctly the first time than it is to do and redo the design several times.

Furthermore, a precise and documented design facilitates design changes. Anyone who has worked on even moderate-sized systems knows that, to control the inevitable changes, they must have precise records of the design before and after the change. Also, many programs will still be used long after their designers are no longer available, and many of these programs must be modified and enhanced. Without reasonably clear and complete design documentation, it is expensive to maintain or enhance almost any product. A well-documented design will add significantly to the economic life and value of the programs you produce. What is even more important, by increasing the economic value of your products, you also increase your personal value.

Separate Implementation

Another reason to thoroughly document your designs is to facilitate the growing practice of subcontracting software implementation and test. You cannot efficiently use people in lower-cost countries to do this work unless you have a thorough and well-documented design. Otherwise, these off-shore groups would have to complete the designs themselves. This would waste much of the time and money the subcontract was supposed to save.

With a complete and well-documented design, the designers can move on to newer jobs while the implementing groups build and test the products. Without a complete and well-documented design, the designers will be needed throughout the implementation and test work. One way to ensure that the software designs are complete and implementable would be to require that the implementing groups review and sign off on the design before they accept a design release. While the software designers might initially object to this practice, it would impose the discipline needed to truly capitalize on the implementation and test talent potentially available in developing countries.

New and Innovative Products

One argument against producing complete and well-documented designs is that software requirements are often imprecise and rapidly changing. When this is the case, the requirements, design, and implementation work must all be evolved together. This permits the users to test early product versions and to provide feedback on their improving understanding of the requirements. If these development increments are small enough and are done quickly enough, there will be fewer requirements changes to address and development can proceed rapidly and efficiently.

These requirements problems are most severe with new product development. However, in most established software groups, new product development is the exception. Only a small percentage of development time is generally spent on building new products. Most of the development work in most software organizations is devoted to repairing and enhancing existing products. Since the original designers are rarely available for this work, a documented design is needed to allow other groups to modify and enhance the original designs.

Software Service Costs

Software service costs are largely a function of product quality. While the largest proportion of user service calls are usually for what are called no-trouble-founds (NTF), we once did a study and found that over 75% of these unreproducible NTF calls were attributable to latent product defects. NTF problems are enormously expensive. They waste the users' time and they require multiple service actions before they can be found and fixed. To minimize service costs and to reduce testing time and cost, early attention to quality is essential.

Measure and Manage Quality

Quality products are not produced by accident. While most software professionals claim to value quality, they take no specific steps to manage it. In every other technical field, professionals have learned that quality management is essential to get consistently high quality products on competitive schedules. They have also learned that quality management is impossible without quality measures and quality data. As long as software people try to improve quality without measuring and managing quality, they will make little or no progress.

The lessons from hardware quality practices are instructive. Hardware quality problems have the same categories as software. They include requirements mistakes and oversights, design problems, and implementation defects. In addition, hardware groups must also worry about raw materials defects. Because of their rigorous design and design release procedures, most hardware manufacturing organizations find that their quality problems are generally due to manufacturing problems and not to design or requirements issues. This is why manufacturing quality programs concentrate almost exclusively on raw materials quality and on the quality of the manufacturing processes. The quality of the design is managed by the product developers and verified during the design release to manufacturing.

These manufacturing quality control practices are based on two principles. First, that the quality of the product is determined by the quality of the process that produced it. Second, that the quality of the process can be managed by measuring the work. The

manufacturing engineers then use these measures to control and improve the manufacturing processes.

Quality and Fix Time

One way to think about quality is to consider how the process would change as a function of defect fix times. For example, programmers generally think that it takes only a few minutes to fix defects in test. They base this on their experience with most of the defects they find in unit testing. In system test, however, the time to find and fix defects typically extends to many hours or even days. While most of these defects are fixed rather quickly, some take much longer. The average time to find and fix each defect is generally 10 to 20 or more hours.

Suppose, however, that the fix times in test were much longer, how would that affect the software process? The lessons from the hardware community are instructive. Some years ago, my laboratory had a small semiconductor facility for making special-purpose chips. The turn-around time for producing a custom chip from a completed design was six months. As a result, correcting any design or fabrication errors required at least six months. Since our products were for a highly competitive marketplace, the number of chip-fabrication turn-arounds was critical. The engineering objective was to release products with only one turn-around. With a little practice, their designs were of such high quality that they were generally able to meet that goal.

In the software business, the time to fix defects in final test is increasing and in some cases it can run into months. For example, for imbedded products like television sets and appliances, the general practice is to use more expensive technologies for the initial models so that they can quickly make any software corrections. Then, when the change rate drops sufficiently, they switch to a cheaper technology. As technology continues to get more complex and as competitive forces continue to increase, we will soon have to produce defect-free software before system test. At least we will for high-volume imbedded products. While testing will always be required, the software quality objective should be a one-cycle system test.

Engineered Software

While the quality management methods for the software process are necessarily very different from those used in hardware manufacture, the same principles apply. In summary, these principles are: product quality is determined by process quality; produce and document clear, complete, and precise designs; and measure and manage quality from the beginning of the job. By following these principles, many software groups are now delivering defect-free products more predictably and faster than they ever delivered products before [Humphrey].

Acknowledgements

In writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful comments and suggestions of Dan Burton, Julia Mullaney, and Bill Peterson.

In closing, an invitation to readers

In these columns, I discuss software issues and the impact of quality and process on engineers and their organizations. However, I am most interested in addressing the issues that you feel are important. So, please drop me a note with your comments, questions, or suggestions. I will read your notes and consider them when planning future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey

watts@sei.cmu.edu

Reference

Humphrey, W. S. *Winning with Software: An Executive Strategy*. Reading, MA: Addison-Wesley, 2002.

The views expressed in this article are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

® Capability Maturity Model, Capability Maturity Modeling, Carnegie Mellon, CERT, CERT Coordination Center, CMM, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

SM Architecture Tradeoff Analysis Method; ATAM; CMM Integration; COTS Usage Risk Evaluation; CURE; EPIC; Evolutionary Process for Integrating COTS Based Systems Framework for Software Product Line Practice; IDEAL; Interim Profile; OAR; OCTAVE; Operationally Critical Threat, Asset, and Vulnerability Evaluation; Options Analysis for Reengineering; Personal Software Process; PLTP; Product Line Technical Probe; PSP; SCAMPI; SCAMPI Lead Assessor; SCAMPI Lead Appraiser; SCE; SEI; SEPG; Team Software Process; and TSP are service marks of Carnegie Mellon University.

TM Simplex is a trademark of Carnegie Mellon University.