



InfoTech Update

Information Technology for CPAs by CPAs

RESEARCH AND DEVELOPMENT

COMPONENT-BASED SOFTWARE DEVELOPMENT AND THE SOFTWARE FACTORY

By Philip Friedlander, CPA, CITP, DBA and Doug Collins

Philip Friedlander, CPA, CITP, DBA, is CEO of Friedlander Advisory Services LLC, a firm specializing in strategy and Information Technology Process Improvement, as well as a senior consultant with ADD/Ware Development Systems, a firm specializing in IT project management. He has 30 years' business experience, including the last 15 years in methodologies, and helping organizations improve their systems development and business processes. Doug Collins is co-founder of Synergistic FrameWorks, Inc. a software factory company specializing in distributed, scalable Web-enabled applications. He has worked with, and built, distributed application architectures since 1982.

As users and resellers of accounting and other software programs, most of us do not usually think about the ins and outs associated with the process of actually *developing* the software. We buy it, sell it, train ourselves, use it and rely on the manufacturer's integrity to ensure programs work.

As CPAs and accounting professionals, our clients and customers rely on us for our expert opinions and business savvy, and now that so many professionals consult in the technology arena, having a grasp on software development is more important than ever. Not only does this knowledge benefit a CPA firm or company; it also benefits the client and customer when the professional can pass along his or her knowledge in a meaningful and useful way.

Take a look at some disturbing realities in today's software development. A recent study by the Standish Group indicates that:

- ❖ only 16 percent of software development projects are successful, producing what they were supposed to produce — on time and on budget;
- ❖ fifty-three percent of projects will be “challenged” with schedule overruns of 25-200 percent and/or cost overruns of 50-200 percent; and
- ❖ thirty-one percent of software development projects are cancelled before actually producing anything.

These statistics do not speak well for the Information Technology (IT) discipline. One might hesitate to even call it a discipline with these kinds of undisciplined results. It's even more chilling to know that systems that *do* get implemented still only have spent 20 percent of the total life-cycle cost. In other words, the total maintenance of that system will be four times its development cost over the useful life of the system.

The History of the Problem

Over the past 20 years, there were many so-called answers to the problem. In the early '80s, there were structured techniques preached by people like Ed Yourdon, Chris Gane and Chris Sarson. Then came “information engineering” — espoused by James Martin and Clive Finklestein. In the early '90s, “object orientation” began to take hold. Each of these new disciplines claimed to be the saving force of system development. Yet, the sad fact is that while they had potential, each one ignored fundamental barriers in modern, especially American, systems development shops.

The Software Engineering Institute at Carnegie Mellon University has dealt with these issues for years. Their Capabilities Maturity Model (CMM) is an effort to help systems development organizations gauge their “maturity” along a five-level scale. However, CMM expands beyond just measurement; the impact of the CMM on accounting consulting is vast. There is big business in consultants helping organizations get their systems development process up to the levels of the CMM. Most organization can barely rate a two or three on the five-level scale.

Part of the problem is the influence of short-term thinking on American IT departments. The average tenure of a chief information officer, even today, is about two years. With that kind of time frame, how can we expect any real investment in long-term solutions? In many organizations, IT is treated as an expense rather than an investment, and one investment that often is lacking is the

continued on page 2



Component-Based Software Development and the Software Factory *continued from page 1*

investment in people. Many software development organizations face the statistics noted in the beginning of the article because their people are not yet adequately trained on the proper system development process.

The Factory Process

Let's use an analogy. In the late 1800s, early 20th Century craftsmen made many different things by handcrafting each one. Everything was custom-made, even though a craftsman might be making a piece very similar to the one he made the day before. Manufacturing, then, was individual based, and the quality of the product was highly dependent on the skill of the individual. The process was very expensive compared to some innovations that were about to come. Most importantly, the customer was highly dependent on knowing and trusting the individual artisan.

When Henry Ford and other inventors introduced mass production to America, they brought not only a less expensive way to manufacture, but innovations with other more important qualities as well. Mass production:

- ❖ was less dependent on critical skills in short supply — you did not exclusively need master craftsmen in the process;
- ❖ made it easier to bring junior people up to productive status because apprentices had to understand only a small part of the entire manufacturing process instead of having to know the entire process;
- ❖ enabled a higher volume of output per person; and
- ❖ made the process repeatable — once workers knew the process, it could be performed over and over again with consistent quality.

As ideal as this sounds, mass production was not without its drawbacks. For example, it did not allow for as much customization. Even Henry Ford said, "You can have any color you want, as long as it's black." Some people say the craftsman as artisan was lost in the process, but that is not entirely true. Master craftsmen still were needed to translate the product requirements into a manufacturable process.

Applying the Factory Concept

Fast forward almost a century, and we're into mass production of software. While the concept sounds good, there are some barriers to instituting factory principles to software manufacturing, the most challenging one being the "craftsman" mentality. Most IT professionals consider themselves craftsmen and are, accordingly, resistant to methodologies, processes or anything else that takes away their "artistic creativity." For now, let's look at what it takes to have a reliable factory process and see how the principles apply to software manufacturing.

Design for Manufacturability

In manufacturing, a significant amount of product design time is spent in manufacturing or producing the product, and sometimes, the product's design is adjusted, and perhaps even sub-optimized to a degree, to make it more efficient to manufacture. This is done in a number of ways, such as designing components for what our current manufacturing line can produce, or more simply, manufacturing what we have the equipment and skills currently to build. Alternatively, if there is a required part that we currently cannot manufacture, we make a conscious decision to either develop those skills or outsource that component part.

The big difference between the manufacturing process and current software development is that these issues are identified up-front, and costing and scheduling estimates are adjusted accordingly. Part of the "tooling-up" process is to understand issues like learning curve and outsource dependencies.

Design Component Parts for Reuse

The most common way to reduce the cost and time for manufacturing is to identify opportunities to use existing component parts in the product. Again, for comparison, when a car manufacturer designs a new car, it seeks to reuse as many existing parts as possible for a number of reasons:

- ❖ existing parts have already been designed, manufactured, tested and proven, thereby reducing the risk of using unproven components in a new product;
- ❖ an equivalent new part would not just have to be manufactured, but also designed, tested and inventoried;
- ❖ a part reused in a number of products has more economies of scale opportunities in inventory and maintenance — if 20 products have the same carburetor, then there only has to be one set of documentation, maintenance instructions and training, instead of 20 different instances of each; and
- ❖ concept to production time is significantly less due to reduced design time for component parts.

During the mid-'90s, automobile companies reduced their component parts inventories by an average of 200,000 parts to fewer than 40,000, and saved billions of dollars. Part of the challenge in reusing components is to design the components themselves with high potential for reuse. If a carburetor is designed for a specific kind of car and the design, while optimal for that car, does not allow it to be used in other contexts, that carburetor has limited value to the factory process.

Even though a software program may be a marvel and parts of it are the absolute best for that program's single use, the fact

continued on page 3

Component-Based Software Development and the Software Factory *continued from page 2*

Figure 1 — Traditional vs. Software Factory

	Traditional Factory	Software Factory
Design	Product is designed with manufacturability in mind. Consideration is given to both the core competencies of the organization as well as the tools and equipment that are available.	Product is designed with manufacturability in mind. Consideration is given to both component tools available as well and the software manufacturing software (equipment).
Tooling Up	The factory floor production line is configured to efficiently and effectively manufacture the new product. Issues like quick change-over of tooling is considered.	The factory production line is configured to produce the specific types of system functionality needed.
The Production Process	Consideration is given to the layout of the production line. Concepts such as lean manufacturing drive issues like cell manufacturing, batch-of-one manufacturing, point-of-use inventory, etc. are used to minimize wasted motion and effort. Appropriate steps are automated.	The production process is well-defined and very similar to cell manufacturing. The line is designed to minimize wasted labor. Appropriate steps are automated.
Quality	Quality is built into the process. Small batch sizes allow quality problems to be detected early before significant cost and rework is incurred.	Quality is built into the process. Small batch sizes, in the form of iterative development, allow quality problems to be detected early before significant cost and rework is incurred.
People	People are given the training, motivation and tools they need to do their job properly. Every job/role is valued in the process.	People are given the training, motivation and tools they need to do their job properly. Every job/role is valued in the process.

that it is not adaptable makes it a barrier to good manufacturing. In order to maximize productivity, developers had to redesign many components so that they were more easily reused across product lines.

Stabilize the Process

The key element of a good manufacturing process is stability, and once the process starts, there may be adjustments. However, once those adjustments are made, the process is stable over time. People do not capriciously change or bypass the process, or say they are in a hurry and haven't got time to take the product through the process.

Have the Right Machinery in Place

Successful manufacturing has the right tools and equipment in place to optimize the process. An astute manufacturer will make an investment in tools and machinery because it will save money and time over the life of the process. The right equipment can automate an otherwise painstaking manual task. It not only makes the task faster, but often more reliable and of higher quality. It also makes the process less dependent on specialized skills.

Invest in Training

Smart manufacturers understand the payback of good training. From the standpoint of product quality, safety and throughput, as well as morale and turnover, investments in appropriate, focused training almost always pay off on the factory floor. Proper training also can reduce key-person dependency; a lesser skilled, but properly trained individual is often more available and possibly less expensive than a highly skilled craftsman.

Taking These Principles to Software Manufacturing

Software manufacturing is not significantly different from any other sort of manufacturing, and many of the ideas for manufacturing software have their roots in object orientation. However, object orientation, during its initial introduction in the late '80 and early '90s, relied too much on technology and abstract analysis, and did not focus on implementation. At that time, object orientation was much like someone theorizing on the ideal factory floor, but never actually building one.

Here are some principles of object orientation and how they align with good manufacturing philosophy:

continued on page 4



Component-Based Software Development and the Software Factory *continued from page 3*

Design for Manufacturability — integration into the manufacturing process requires clearly identifiable interfaces. A foundational principle in object orientation is called encapsulation, the principle of binding related data (properties), process (methods) and triggers for the process (events) together. In this way, a reusable unit with a clearly defined interface contains (is encapsulated with) everything it needs in order to be used over and over again without the user having to seek out other dependent parts.

Design Component Parts for Reuse — in order for an object to be very reusable, it must be designed to behave differently when used in different contexts. This is known as Polymorphism. For example, an object may have an interface that supports printing that can be used for displaying on the screen or printing on a printer. The object will know its context and behave differently to accomplish the same objective (displaying data).

Stabilize the Process — requires an environment that categorizes and facilitates the product production in all aspects of the product manufacturing lifecycle. One of the problems from which object orientation has suffered is that no one had developed an effective, stable process in which to use “object orientation (OO)” techniques. For many years and still to a great deal today, OO suffers from the “arts and crafts approach.” There are many good, effective experienced practitioners, but there was no mass production process developed that did not require the “artisans.”

Have the Right Machinery in Place — efficient factories have tools to automate the factory floor and facilitate the production line. While object orientation did have some CASE tools for quite some time, these were mainly analysis and design tools. There were no effective manufacturing tools or software to automate the software production process.

Invest in Training — the single biggest obstacle to good software manufacturing is still training. A factory needs the artisans, but it also needs well-trained journeymen workers. Because of the short-term thinking discussed earlier, American management has not invested in the right kind of training for the masses. In addition, up until now there was no mass production process defined around which one could train.

The New Software Factory

Let’s take an example of an organization that, on average, yields savings of 35 percent in time and cost of software projects over traditional methods, always delivers on time and on budget, and eliminates risks and surprises.

Anyone who has worked with IT organizations and has experiences that more closely align with the statistics cited in

the beginning of the article knows these facts may sound amazing. However, results were achieved by merely following the principles in object orientation. This organization has produced a process that practices what we preach.

Design for Manufacturability — they have developed a design process that immediately breaks requirements into “features.” Features are clearly identifiable, measurable functional units of requirements that are expressed in a usability perspective independent of technology. The features units are categorized and cost factors applied using highly accurate (because they have been measured over time) predictors of the downstream effort. The predictors are based on the effective implementation of components and a rigorous manufacturing process, much in the same way an experienced industrial manufacturer measures the cost of doing everything on the factory floor. This company prepares for manufacturability by partitioning requirements into features.

Design Component Parts for Reuse — they have developed a library of highly reusable, polymorphic components based on industry standards like JAVA, CORBA and XML. These components implement common, recurring functionality in business systems.

Stabilize the Process — they have put in place a rigorous process that takes into account the benefits of the components and factory floor automation. Software development is not arts and crafts here — it is manufacturing.

Have the Right Machinery in Place — they have developed software manufacturing software to automate all of the pieces of the process that are repetitive and tedious, freeing up the people for the parts of the process requiring thinking. As a result, people have time freed up to be creative when creativity is needed.

Invest in Training — the people involved in the manufacturing process understand it along with their role. They truly consider themselves software manufacturers instead of artists. This is what overcomes the “artistic resistance” to a methodical process.

Figure 1 (see page 3) shows a comparison between effective traditional manufacturing and the effective software factory.

Let’s close with an example project. In order to understand the scale and scope of this project, one must understand a bit of the vocabulary of the software factory. In the early stages of a software factory process — what one might call the “tooling” stage — requirements are partitioned into “features.” While the rules for factoring out features are too complex and lengthy for this article, suffice to say that a feature is a large unit of functionality to the end user of the system.

It is important to understand that these units of functionality are in terms of what the end user understands. There are explicit

continued on page 5

Component-Based Software Development and the Software Factory *continued from page 4*

features for which the end user asks, and implicit features that are suggested by the functionality encompassed in the explicit features. Because of the rigor of the software factory process, not only can these implicit features be anticipated, identified and included in the end product; these implicit features often are those implemented via the reusable components and the software factory automated process. The results are that the user gets functionality that they may not have known they needed. Typically, in a normal system development process, these additional features would be “Revision 1.1” enhancements. In addition, the functionality is implemented at a relatively low cost, due to the experience set of the software factory process, staff and automation.

This example project contained 22 features, and was estimated, under normal software development estimates, to take two developers 30 weeks to complete, or 60 man weeks of work. The software factory process completed the work in 10 weeks. However, what is even more important than the 66 percent reduction in elapsed time and 40 percent reduction in cost is the fact that more than 35 implicit features also were implemented that the buyer didn’t even identify in the initial set of requirements. The most critical indicator, however, is that this system was brought in on time, on budget and with no surprises.

The software factory produces production-ready systems. This project was for a client that needed a facility management system. The delivery was a functional facility management system that was built using a component-based solution with CORBA providing scalable distribution. The client used the system in a networked environment with multiple users to demonstrate their product direction. Subsequent projects with the client produced the same level of metrics or better. The subsequent projects produced an intelligent work management system that is touted as a leading product in the facility management market.

Continuous Improvement

Another aspect of the software factory, and one that is extremely important to the CMM, is the constantly improving process. The software factory is constantly adding to its inventory of reusable components, improving both the people side of the development

process and the automation that supports it. As good as the software factory was a year ago, it is much better now and not nearly as good as it will be a year from now. This is very much aligned with a modern traditional factory.

Change is Needed Across the Industry

There is an important principle of organizational change: *nobody changes unless they have to*. The status quo is comfortable to people and change is painful. Therefore, people will not change until there is more pain in the status quo than in changing. The changes in modern manufacturing and in the automobile industry, for example, did not come about because the automobile industry decided it wanted to change. The change came about due to outside pressure, specifically pressure came from Japanese car manufacturers who made American car buyers understand things could be improved, and that they did not have to settle for the quality and cost levels that were then standard in the automobile industry. American carmakers did not change until they began to feel pain in their bottom line.

We cannot expect the IT discipline to be any different. IT practitioners will continue to practice as they always have because their status quo is neither currently painful nor will it be until the buyer begins to put pressure on them to improve. Even with the current downturn in the economy, experienced IT practitioners are still in big demand. Even ones producing the mixed results cited in the beginning of the article can find a place in the IT departments of any large organization. Until IT departments are made to compare their results with the types of results cited here and made to “compete” with outside forces, significant wide-scale improvement is still fleeting.

There *are* pockets of good practice in existence today. There are organizations that understand how to do it right, and more importantly, are willing to invest the time and money into truly improving the process and profits for the long-term. We hope that this information will inspire organizations to improve their system development process.

Contact Phil Friedlander at PhilFriedlander@hotmail.com, and Doug Collins at douglas.collins@sfworks.net. 