Without Software, No Megatrends

Lawrence H. Putnam

Ware Myers

Quantitative Software Management, Inc.

If we have virtual corporations operating over the Internet, they will be using software. If we have household appliances that you control from your office or car, your control will be implemented by software. If e-business is a megatrend, as indeed its position on the covers of leading business magazines declares it to be, it, too, will depend upon software.

It's fun to speculate about the megatrends of the first part of the 21st century. The management challenge is to support these exciting trends. Yet, unless the software works, on time, and for a price, many of the megatrends will derail.

What all the trends foreshadow is more software. Moreover, this software will be more inter-dependent and more highly integrated. It will have to operate in a "smarter" environment. Individual systems may not be much larger. In fact, average project size reported to the QSM database during the last decade of the 20th century remained about the same--around 40,000 lines of source code. The reason appears to be that, more and more, a software system does not stand alone. It interfaces to other systems. That will be still truer when a large share of everything becomes Internet-enabled. That is, most software systems will have to be capable of operating on, or at least interfacing to, the Internet. Individual systems, however, tend to be of manageable size.

So, the over-riding trend for software developers stays the same--getting the software to work and getting it to that point within a time-to-market and within market-set economic constraints (costs to you). The problem is the same, but we expect its magnitude to grow. Several decades ago, when today's planning, estimating, and control systems originated, we were developing most systems from scratch and most of them operated independently of other software systems. In other words, our estimating problem in particular and management control in general were bounded. Even so, the software industry as a whole did not do a very good job of estimating and managing software development.

We won't bore you with another recital of the horror stories. We do draw a conclusion. Implementing the software to support the megatrends is going to place software organizations under still more strain. We doubt that the people in most organizations can work much longer or harder. There are some indications that students are not specializing in computer science and that people already in the field are leaving it. The only way out of this trap may be to work smarter.

That is possible, and the existence proof is in Figure 1. First, the wide bars represent the average level of process productivity of the projects of all kinds in the QSM database completed each year. The trend has been generally upward since 1970. In other words, the companies reporting to the QSM database have been "working smarter" year after year since 1970. Of course, to

report to this database, a company has to be keeping metrics. The companies not keeping metrics may be doing more poorly. Nevertheless, the wide bars demonstrate that working smarter is possible. It has been occurring.

Second, the thin line above the wide bar represents the proportion of projects where process productivity exceeded the mean. The top on the thin line marks one standard deviation above the mean, or the 84th percentile. In other words, each year a substantial number of projects are working smarter than the average project. Note that, just as the average process productivity has been increasing over the years, so has the 84th percentile level. The already smart companies have been getting steadily smarter. Each year these companies demonstrate that it is possible to do better.

Process productivity is not an estimated number. It is not based on management assessment. It is derived from size, project duration, and effort applied--all hard metrics after a project is completed.

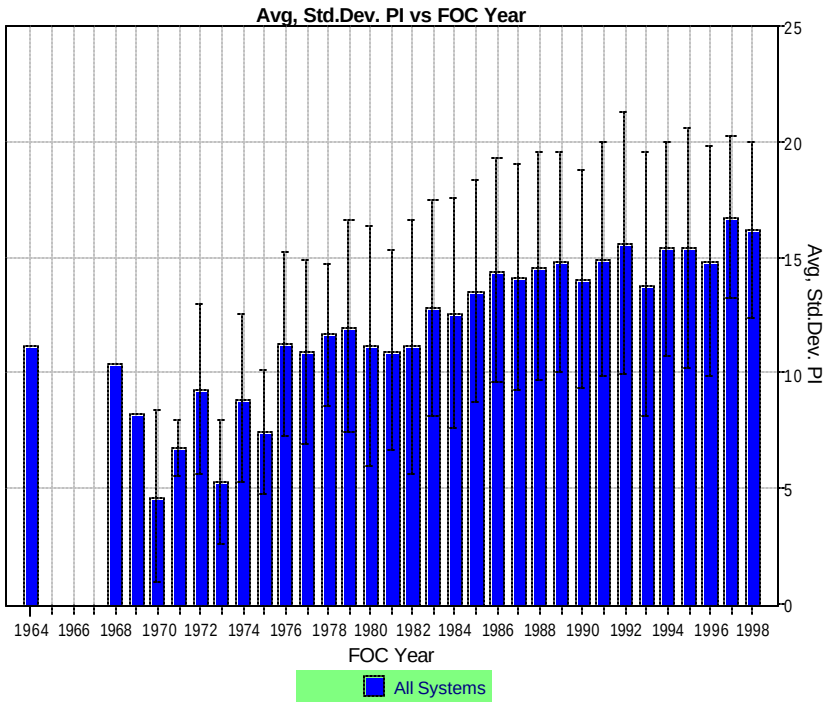

Figure 1. The process productivity of the projects in the QSM database has been growing, with occasional setbacks, for three decades. Each bar represents the projects that reach Full Operational Capability in the year shown. When we divide the database into business systems, engineering systems and real-time systems, the diagrams are similar, but the means or averages differ. Some types of development are more difficult than others.

Each yearly bar in Figure 1 represents a bell curve (or *normal* statistical curve). The position of a particular organization on the curve--its process productivity--tells it *where* it falls, but not *why* it is at that point. Consequently, it is not going to be easy for software organizations now below the mean to pull themselves up--particularly with the mean itself advancing most years.

Moreover, there are four trends within the software field itself that promise to make development more productive, but at the same time to make estimating and management more difficult.

Up-front emphasis moves time and effort up front

One of these trends moves more of the development effort to the early phases of the development process, as we discussed last April in CITJ. Briefly, we ought to determine that a project is feasible before we start spending a lot of money on doing it. We ought to make the business case--that the project is worth doing. We ought to nail down at least the pertinent requirements. We ought to look into the risks that could upset the project. We ought to have some grasp of the architecture.

The early phases, in Department of Defense terminology are *feasibility study* and *high-level design,* followed by *main build* and *operation and maintenance*. Recently, the Unified Process has used the terms, *inception* and *elaboration,* for the first two phases. They are followed by *construction* and *transition.*[1]

If we did all those things up front, the main build itself ought to proceed very smoothly. In fact, we ought to accomplish it in less time and at less cost than it takes under the present regime of little up-front work. Well, that sounds attractive.

What we don't know how to do, in general, is persuade clients to fund and allocate time for the up-front phases. Moreover, there is little data on how much time and effort they take. QSM estimates that the feasibility study typically takes about 14 percent of the total project time and about 4 percent of the total effort, diagrammed on Figures 2 and 3. However, most projects nowadays are beyond the initial development cycle and feasibility is in little doubt. There may be little need for this initial phase for such projects.

In cases where feasibility is in doubt, this initial phase appears to be comparable to research. It probably should be assigned to a research-oriented group on a time and materials basis. In industry that group might be some senior people whose time is charged to overhead. Still, to the extent that a company still finds a feasibility phase needed on some projects, it would be valuable to accumulate some time and effort data to guide estimating it.

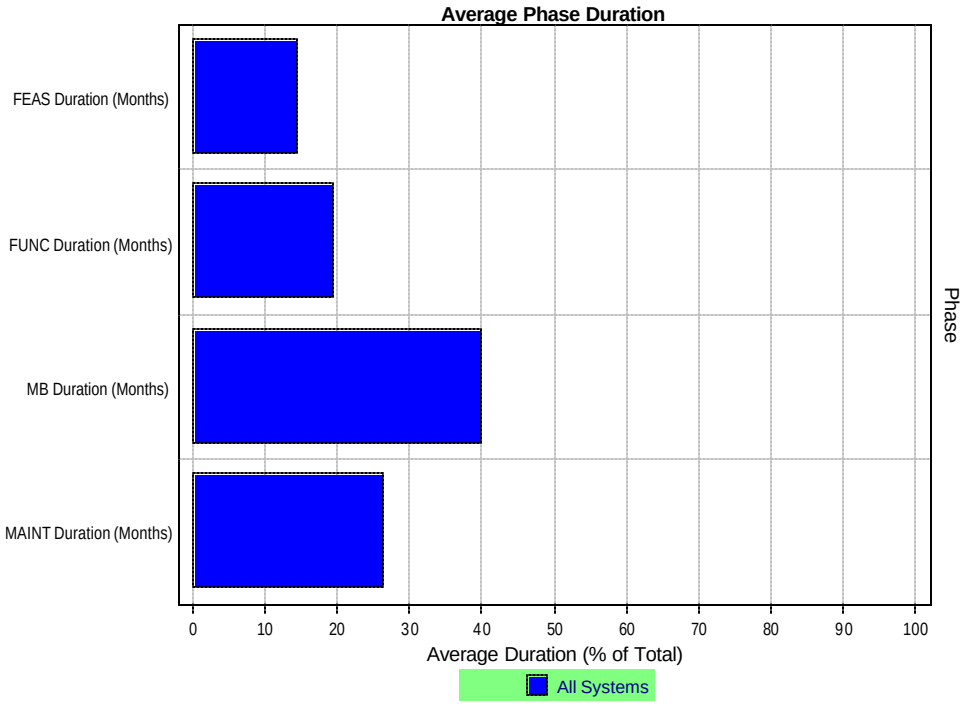

Figure 2. Reports to the QSM database indicate the percentage of time spent in each of four phases. The four phases are probably comparable to feasibility study (inception), high-level design (elaboration), main build (construction), and initial operational maintenance (transition), but phase definitions used by the reporting organizations differ.

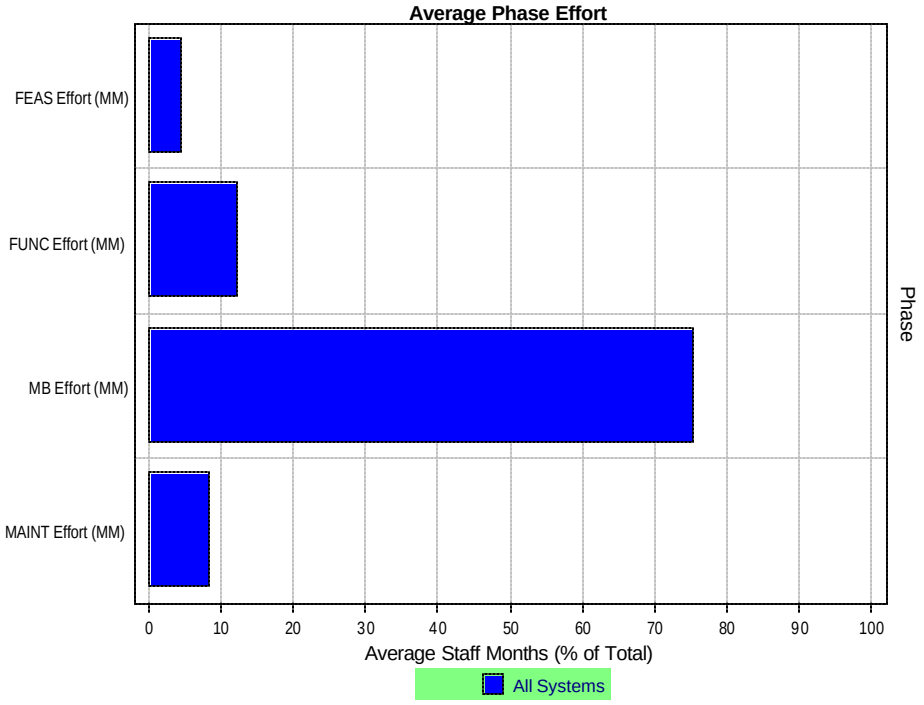

Figure 3. Reports to the QSM database indicate the percentage of effort (person-months) spent in each phase.

The high-level design or elaboration phase takes about 20 percent of the project time and about 12 percent of the effort. That is a pretty good chunk of the total and few clients are accustomed to financing that kind of time and effort that early, that is, before enough information is in hand to make a firm bid. Ideally, this phase should identify the significant risks and mitigate them to the point where they will not cause time or effort overruns in the main build. It should carry architecture to the point at which management can estimate system size close enough to support a reasonably accurate bid.

The management problem is this. A good many software organizations are accustomed to locating most of this up-front work in the main-build phase. That means they are bidding more time and effort in this phase and doing little work in the inception and elaboration phases. It also means that they often underestimate the main-build time and effort. The 1998 Standish Group report found, for example, that about three-quarters of sizable projects overrun budget or schedule or finish short of the originally intended requirements. That suggests that many organizations start the main build with an inadequate grasp of the system's problems. Re-arranging the work into these phases, estimating time and effort for them, controlling the progress of work in them--these are all new tasks for many software managements.

Setting a basis for planning the next generation

Providing software to support the coming megatrends will involve some work in initial development cycles. In most cases, however, there will be a current generation of the software. Estimating size, time, and effort for the new system involves understanding what is already available. If the only documentation that has been kept up to date is the source code, penetrating it will take considerable time and effort. If this up-front work is not done, then the project plan and estimate are likely to be unrealistic. The project eventually turns up in the Standish 75 percent.

It is evident that appropriate documentation would reduce the time and effort that lack of past information causes. It follows that standardized documentation that management and developers can easily use would help still more. A step in this direction was taken by the Object Management Group when it standardized the Unified Modeling Language in 1997. [2] Standardized documentation will also reduce the time and effort to interface new software to existing systems. Anything that is going to interface to or operate on the Internet will, of course, encounter existing systems.

Estimating the cost of reuse

The day after the software age began half a century ago, some machine-code programmer probably stuck a sequence of code in his hip pocket. In recent years some organizations have been turning the possibility of reuse into a reality. That leads to the secondary problems: How are reusable components to be costed? How much schedule time should a project manager allow for acquiring and integrating components?

There are two sources of reusable components: in-house and vendors. In both cases the preparation of the component takes place outside the organizational scope of the project that is going to use it. (We pass over the occasional exception, where a project itself is the originator of a component that other projects later employ.) In much of the software industry, the project is the dominant organizational focus. There is little supra-project structure to conceive an architectural pattern for all the projects, to specify interfaces between components, to acquire or to build reusable components, to stock and price them, and to convey knowledge about them to projects that could use them. Component vendors do all these things more or less well.

In the ideal situation, a reusable component has a price. It is adequately described in a catalog. It fits into the proposed system without time and effort-consuming integration problems. In this happy situation an estimator simply adds the price to the bid price and plans no time for integration. Some day the industry may reach that blissful level. At present most reusable components present estimators with problems.

For example, what should an in-house developed component cost a project that contemplates using it? If it was developed on another project, should that project bear the entire development cost? Often, to make a component reusable, there are costs beyond what the project would normally spend. What organization entity should carry those costs until a second (or third) project begins to pick them up? Who bears the costs of whatever supra-project organization administers the reuse process? Well, you can add questions for quite a while. Software organizations need to give the issue some thought.

Allocating supra-project costs

There has always been a certain amount of cost above the project level--for higher level management, facilities, staff departments, and so on. These costs have been tacked on to project estimates by multiplying direct effort costs by a "labor rate." Developing domain architectures, standardizing interfaces, and producing reusable components are going to add costs above the project level. Just how are these costs going to get into estimates?

On the one hand, they could be added to the labor rate. In effect, they would be treated as overhead and passed on to all the projects in proportion to their direct costs. Projects that didn't use these components would be unhappy.

On the other hand, these components could be priced by the reuse organization and paid for by the projects that used them. That might lead to projects bypassing the components because project managers believe the components are not worth the price. And maybe they aren't! It's another issue.

So, as the millenium turns over, we are moving into a brave, new world. We are going to be beset with megatrends as our tiny mammalian ancestors were beset with dinosaurs 65 million years ago. While the dream of immense riches drives the stock price of Internet startups to levels never before achieved, the sub-world of software development will still be one of limited resources and short schedule time. Dream of the approaching mega-wonders, yes, but cathedrals are always built stone by stone.

---

[1] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process,* Addison-Wesley, Reading, Mass., 1998, 463 pp.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide,* Addison-Wesley, Reading, Mass., 1998, 512 pp.