Familiar Metric Management -
Management Vision Precedes Metrics

Lawrence H. Putnam
Ware Myers

By way of obeisance to the great God of *metrics*—before we consider the matter of *vision*, let us proclaim our belief that metrics is a necessary underpinning of good management of software development (even though so far in the software age it seems to be honored more in the breach than in the observance). Metrics underpins excellent management in three important ways:

1. It provides the means of projecting budget, schedule, manpower needs, and plans, giving you the ability to acquire business that will be profitable;
2. It provides the means of controlling the execution of those plans, that is, by comparing actuals as they develop against the plan, giving you the ability to assure the profitability of the project;
3. It provides the means for improving your process by comparing the numbers for this year against the numbers for last year, giving you the ability to stay ahead of competitive organizations.

The growth of a vision. Long ago and far away there was a river valley and a range of hills beyond it. A trail led across a rocky ford, up a ravine, across a low pass, down another ravine, into the plain beyond. To a 12-year old boy, the trail had come down from prehistoric times, possibly originally beaten down by great bears, then used unchanged by generations of native Americans. This trail did not change the hillside; it just wound along it. It had no mileposts—no metrics. One had to judge how far along one had hiked by how thirsty one had become. That was good enough for a 12-year-old.

There may be such trails yet, way out in the back country, but the ones we travel now are called superhighways, four lanes in either direction. They don't wind along the contour of the hills. They cut through the hills. Where you are is proclaimed by signs every mile or so. You need not get thirsty or hungry, because a rest area comes up before you need it.

Both the primitive path and the modern highway are comparable to a "vision." They guide you to a geographic "somewhere" or in the case of the management vision, to the future. The moral: *management vision* charts the software path. Metrics measures your progress along that path. Without a path, you are hacking your way through unknowns and pretty soon you are lost. In those circumstances, metrics may be interesting, but you are still lost.

Laying out the path has become increasingly difficult. The boyhood path simply followed a route established long before. The physical path of the superhighway rests upon more complicated surveying, and before surveying began there may have been long political fights over whether to have a highway, where the funds would come from, what townships to connect or avoid, and to whom to award the contracts. These are the "dimensions" that beset modern projects. They require "management" over a long period.

Software development dimensions. Development of major software systems faces these highway-project problems and more besides. What are the dimensions of software development?

What to build. Forty years ago this dimension was fairly clear. We transferred to software business processes, like payroll, that were heavy on computation. Now, the direction in which to take software is far from obvious. Let us just list some of the complications:

- Software systems should implement existing business processes more efficiently, or
- The existing business processes should be reengineered to better meet emerging customer needs and these improved business processes should then be implemented in software, or
- The fact the elaborate software systems are now feasible feeds back on what the "improved business processes" should be and that, in turn, has an effect on "what to build."

Dizzy yet? Let's go on.

Requirements capture. Given a rough idea of *what to build,* stakeholders and developers face the task of converting that concept into detailed form. As the effort progresses, however, both sides learn more about the real-life application. What they learn is that it is more complicated than they thought originally. As they learn more about it, the requirements keep changing. Constant change is a "dimension" of software development.

Functional design. The requirements have to lead to an overall pattern of what the software system is to be. This architecture should not only guide the current development cycle, but should be good for several later generations. In other words, the architecture and the resulting system should be capable of adaptation to needs now perceived somewhat dimly.

Risk. Doing something not fully defined over months and years into the future is accompanied by risk. The definition of what to build will evolve; technical risks that were not foreseen may appear; business risks, such as competition getting there first, may materialize.

Staffing. Software development is carried out by people, involving all the well-known complications that accompany the human element, such as hiring, training, pay scales, motivation, morale, turnover, and disputes. Having a path moderates at least one source of disputation.

Process and tools. The Software Engineering Institute calls its Capability Maturity Level one "ad hoc" because it has no repeatable process. But there are software development processes and they make a difference.

Economic constraints. All this has to be done within business limits of cost, effort, and schedule and has to meet a quality goal.

Management. Back in 1992, when we published *Measures for Excellence,* Larry wrote: "Similarly, on the basis of my own consulting experience, management is high on my personal list. I used to put tools at the top. Now, I think the influence of management is much more important. Management can make all the other factors come together."

Vision of the business. A generation ago Peter F. Drucker (in his book, *Management: Risks, Responsibilities, Practices,* 1973) saw that a "theory of the business" had to underlie the commitment of resources "to an ever longer future." The mission and purpose of every business, he said, is to satisfy the customer. Moreover, for the vision "to be good for ten years is probably all one can normally expect."

Management's vision of the future is an ongoing task. It is never-ending because enterprises, whether private or public, are in the throes of continuous change, because of factors such as;

- Growing size,
- Increasing geographic reach, currently called globalization,
- Intricate relationships with vendors and customers,
- Rapidly changing product technology, leading to rapid generational turnover,
- Changing process technology, including the process of software development.

This complexity makes modern enterprise increasingly difficult to manage. In fact, some say top management can no longer manage in the detail sense of past generations; it can only lead. It must depend on decentralized managers for detailed management—the level at which software metrics, for instance, come into play. Top management leadership still embraces many facets, one of which is laying out a path—a vision.

Andrew S. Grove, Chairman of Intel Corporation, took a stab at the issue in his 1997 book, *Only The Paranoid Survive.* He examined the plight of top executives in high-technology businesses likely to be impacted by more high technology, such as the Internet.

However, executives in low-technology concerns are also in peril of being blind-sided. How about all the local hardware stores Home Base and the like have supplanted? The mom-and-pop stationery stores replaced by Staples? The department stores out-competed by Wal-Mart. A chief feature of these newcomers has been their ability to apply software systems to old lines of business.

Of one thing we can be reasonably sure. Few organizations will be able to lay out a path that extends very far into the unknown future. It follows then that the ability to develop software systems in less than the many years it now takes will be a significant competitive advantage. That means:

- Figuring out *what to build* right on the heels of the unfolding future;
- Crafting system *architecture* capable of adaptation to future requirements;
- Identifying both technical and competitive *risks* early in the development cycle while you still have time to do something about them;
- Improving your development *process* and automating more of it in *tools*;
- Moving to *component-based development*, also known as reuse;
- And, of course, our own pet subject—better *planning* of project effort and schedule.