Familiar Metric Management -
Productivity of the Reuse Organization

Lawrence H. Putnam
Ware Myers

The management metrics (effort, development time, and functionality) and the process productivity index computed from them can be applied, not only to organizations developing application systems from scratch, but also to the software organization units specializing in the development of reusable components.* However, as project organizations developing application systems make more use of reusable components, we expect their "total" productivity indexes to be high, reflecting the value of reuse. In fact, in the case of a group of projects that QSM Associates measured, the PI's averaged 26.2, compared to the average of business-system projects, 16.9. Because of the exponential nature of the PI scale, that difference is a factor of 9.4 times, as Paul Bassett explained. [1]

Moreover, as the reuse program gains steam and application development units utilize increasing proportions of reusable components, their total PI's will increase. If they do not, something is wrong with the reuse program. That is one of the functions of metrics. It is a signal to look into a problem. Their "working" productivity indexes, reflecting an application unit's effectiveness in developing new code, serve the same function as the management metrics have all along.

The PI is also a useful metric for assessing the effectiveness of reusable-component development. Because the development of components that can be reused in many different application systems is much more difficult than the development of single-use components, we expect the PI's to be correspondingly lower.

---

* You may recall from earlier columns that we derive process productivity for a single project from measures of development time, effort, and functionality in source lines of code for the one project:

Process productivity = (Quantity of product in SLOC) / (Effort/Skills Factor) $^{1/3}$ (Development Time) $^{4/3}$

Reuse-organization productivity

The component-system and application-system organization units, however, are only part of the entire organization needed to carry out reuse. Other units include up-front domain analysis, architectural planning, component support, and reuse management, as detailed by Ivar Jacobson and his co-authors. [2] We can hardly expect this complicated sequence of activities to function of its own accord. It will need overall management to set direction, allocate appropriately trained or experienced people, standardize the collection of metrics, and adjudicate differences.

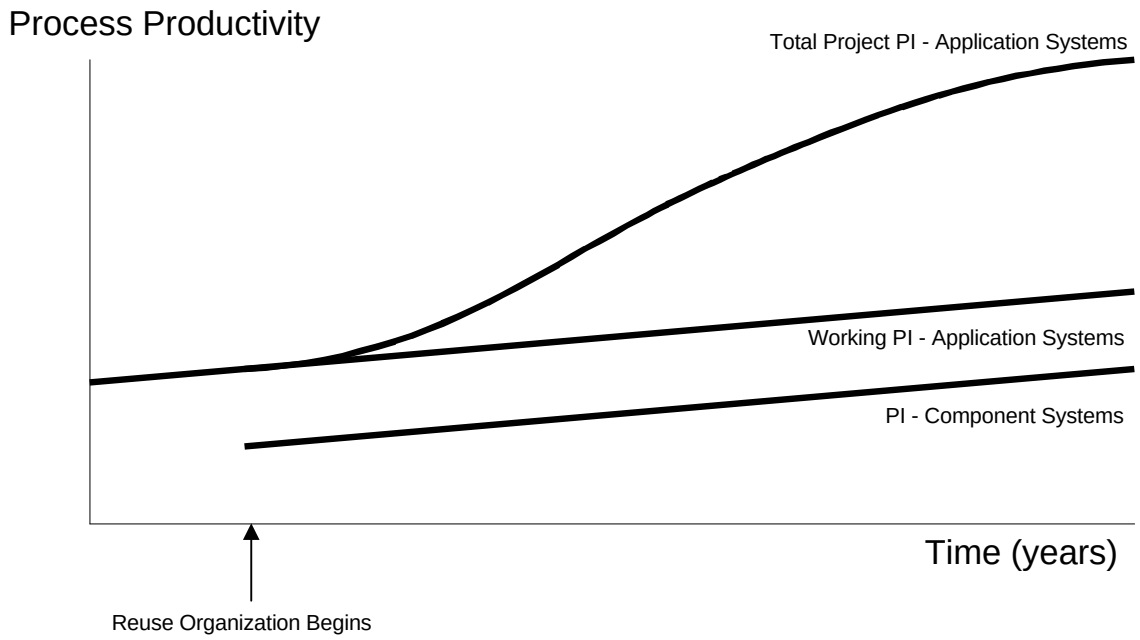
These units are not directly responsible for the production of code for either a component or application project. Therefore, they cannot be brought directly under the PI umbrella. That is, their work, being spread over many projects, is not part of a project PI, whether the project is a component system or an application system. The reason is that the software equation, from which the PI is derived, measures the process productivity of a project. The effort figure in that equation

represents person-years directly chargeable to the project. It cannot readily include the person-hours devoted to domain analysis, architectural planning, component support (that is, operation of the component library), and management of the overall reuse organization. Perhaps these "overhead" hours could be allocated on some basis to the projects, but the allocation scheme, whatever it might be, would muddy up the purity of the project PI's.

Let's ask instead, what is it about the reuse organization that we are interested in measuring? Well, it is the effectiveness of the reuse organization as a whole. We are already gauging the effectiveness with which we develop individual systems, whether an application system or a component system, with the PI. The domain analysis, architectural planning, component support, and management units, together with the two types of development units, make up a reuse organization that can, over a period of years, build many individual systems. At first, with few reusable components, the reuse organization will be having little effect on the company's ability to develop software. After several years sorting out and building reusable components, the effect will be greater. It is that increasing effect that we want to measure. We can observe that effect in the PI's of the application projects. By taking an average of the PI's of all the application projects completed during a period, an organization will have an indication of its effectiveness in applying reuse.

In the first year of a budding reuse organization, the average productivity index of the application-system projects will rise slowly, probably at about the same rate as they have been rising in the past. That is because they do not yet have many components available for reuse. The reuse organization as a whole is beginning to expend considerable effort on domain analysis, architecture, and initial component-system development. In fact, the component-system units may just be getting organized.

As soon as component systems become available and application developers learn to use them, the application-system project PI's go up. That average PI reaches a peak some years later when reuse approaches the limit, probably somewhere above 90 percent. The figure illustrates the general behavior of the several productivity indexes.



[Figure 1. The "working" productivity indexes of the application groups and the component groups rise slowly over the years, reflecting the slow growth in process productivity. Once component systems are available in some quantity for reuse the "total" productivity of the application groups grows exponentially until it reaches a new plateau, where reuse begins to near its limit.

Recovering costs

If the application-system projects are producing products for external customers, they naturally charge for them. Even if they are developing products for internal users, they may make a charge. This charge should include not only the application-system unit's own costs, but also the cost of the component systems they employ. Fundamentally, there are only two ways to transfer this cost from the component units to the application units. One is to add up the costs of domain analysis, architecture, component-system development, component support, and reuse management, based on accounting data. Then distribute this cost as an overhead charge to the application-system projects, again by accounting methods. This method leads to higher overhead. It also leads to dissatisfaction by those generally distressed by overhead. More precisely, it leads to dissatisfaction on the part of those application projects that, for good reasons or bad, are using less than the typical proportion of component systems.

The second method is to put a transfer price on each component system and charge each project for the component systems they use. This price would include a proportionate share of the costs of domain analysis, architecture, component development, and component support. At first the transfer price would be high, as the reuse organization recovers its initial investment. After three reuses, say, the transfer price might drop to cover only continuing running costs. By that time dissatisfaction would drop to a low level. Not only would the price of component systems be low, but the great gains from reuse would have become apparent.

In the overhead distribution method the costs are distributed rather arbitrarily under the supervision of accounting people. By the nature of their location in the organization, they can have little familiarity with the value of individual component systems. Under the transfer-price arrangement, the component-development part of the reuse organization sets these prices and the application-development part, if need be, challenges them. The advantage is that both sides know the technical circumstances that they are negotiating. The prices that stick are likely to be fairly accurate measures of the value of the component systems. They amount to another metric with which to evaluate the worth of the reuse operation.

"We have learned that even with simple technology and only a few people focused on systematic reuse, you can incrementally build a reuse program that will produce significant improvements in quality, productivity, time to market, and competitive posture in just a few years." Martin L. Griss and Marty Wosser [3]

1. Paul G. Bassett, *Framing Software Reuse: Lessons from the Real World,* Yourdon Press, Prentice Hall, 1997, 365 pp.
2. Ivar Jacobson, Martin Griss, and Patrik Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success,* Addison Wesley Longman, 1997, 497 pp.
3. Martin L. Griss and Marty Wosser "Making Reuse Work At Hewlett-Packard," *IEEE Software,* Jan. 1995, pp. 105-107.