Software productivity and its management

by JAINENDRA K. NAVLAKHA
Florida International University
Miami, Florida

ABSTRACT

Software productivity is one of the most important attributes of the software development process. Our current knowledge of software productivity is not sufficient to define it properly, let alone measure it accurately. Indeed, for lack of something better, the industry has used some version or other of lines of code per man-month as the measure of productivity. To my mind, this measures programmer productivity, not software productivity. In this paper, I discuss several possible definitions of software productivity from different viewpoints. I also discuss an important issue on the subject, the management of software productivity.
INTRODUCTION

Software engineering refers to the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation. 1 The annual cost of software in the United States, which was approximately $40 billion in 1980 or about 2% of GNP, is estimated to grow to 13% of GNP by the year 1990. 2 Over the past two decades, the cost of hardware has steadily decreased while the cost of developing software has consistently increased. The demand for software is increasing faster than our ability to supply it.

In such an economic atmosphere, it is necessary that methods be developed to measure various attributes of computer software in order to continually enhance our ability to increase productivity. The sector of software engineering that deals with these measurements is called software metrics. Software metrics has evolved from the gradual recognition of the need and importance of measuring and controlling the quality and development cost of software product.

Measurement discipline is as fundamental to programming as it is to any other area of engineering. Boehm, et al. define a software metric as a measurement of the extent or degree to which a software product possesses and exhibits a certain quality, property, or attribute. 3 Software metrics can be used to measure many properties and attributes of software including reliability, complexity, productivity, quality, correctness, availability, maintainability, portability, and development effort. They can also be used to provide a precise definition of certain measurements to be used in legal contracts, to manage resources, and most of all to evaluate the quality of a design and pinpoint potential problem areas of that design so that changes and improvements can be made at any stage during the software development process.

Thus, it is evident that software metrics is a very important area of software engineering. However, it is also one of the most difficult areas of research. As Munson and Yeh report, "... the definition of the attributes of software which materially affect cost, quality and productivity are still unknown, or when known, are unmeasurable. Until we can measure and compare we cannot consider computer science a science." 4

One of the most important attributes of the software development process that needs to be measured is software productivity. In most software houses, productivity has been measured as follows:

\[
\text{Productivity} = \frac{\text{output}}{\text{input}}
\]

Traditionally, productivity has been defined as follows.

\[
\text{Productivity} = \frac{\text{Total deliverable source instructions}}{\text{Number of man-months to develop and correct}}
\]

A slight variation of the above definition of the form of number of modules in the software per unit time period has also been used by some. This variation essentially reduces to the original definition by approximating the number of source lines per module, assuming proper modularization.

Note that the above mathematical definition does not truly define software productivity as much as it defines programmer productivity. While it is quite important to have a measure for programmer productivity, this is not the subject of this paper. Hence, I will stay away from the above definition. It is quite conceivable that until a better measure for software productivity is available, most people will continue to use programmer productivity instead.

In this paper, I first discuss a couple of possible but incomplete definitions of software productivity and list some of the attributes on which it depends. Also included are different viewpoints about software productivity and associated measurements. There are many other aspects of software productivity like advantages, means to improve, etc., which one may discuss, but I intentionally omit them here. However, I do discuss various issues related to software productivity management.

DEFINITION AND DIFFERENT VIEWPOINTS

It should again be emphasized at this point that this paper will discuss software productivity as opposed to programmer productivity. Thus, the definitions and their corresponding metrics that we refer to are not intended as a means of setting goals for the evaluation of individual performance.

This section discusses a couple of definitions of software productivity and some dependent attributes. It also discusses an economic viewpoint, a management viewpoint, and a way to look at software productivity as the software progresses through various life-cycle phases.

Traditionally, productivity has been defined as follows.

\[
\text{Productivity} = \frac{\text{Total deliverable source instructions}}{\text{Number of man-months to develop and correct}}
\]
put and what constitutes the input. Some of the output attributes on which productivity depends are as follows.

1. Software quality
2. Software reliability
3. Software usability
4. Software maintainability—this may depend on the methodology used to develop software, and on many other factors
5. Software verifiability
6. Timely delivery
7. Customer satisfaction

Some of the input attributes on which productivity depends are as follows.

1. Personnel cost to develop software
2. Cost of equipment
3. Capital investment
4. Incentives given to software personnel

These lists are clearly not exhaustive. There is always an economic viewpoint when looking at productivity. The economic definition of productivity along with the associated input and output factors for each term is shown as follows.

\[
\text{Output value} = \text{Quantity used} \times \text{Initial price} \uparrow
\]

\[
\text{Productivity} = \text{Product} \times \text{Price recovery} \uparrow
\]

\[
\text{Input value} = \text{Quantity produced} \times \text{Initial cost} \uparrow
\]

Along the same lines, the March 1981 IEEE Software Engineering Productivity Workshop chose to use an economic definition of software productivity for their deliberations. Their definition is as follows.

Software productivity = \( \frac{\text{values}}{\text{costs}} \)

where value is defined in profit-oriented terms either as an increase in revenues attributable to the product, or a cost reduction resulting from the use of the product. This definition focuses attention on management decisions rather than on programmer output. Table 1 lists the fundamental elements of economic measurement.

Another way of viewing software productivity is to concentrate on the software system not only in its entirety, but as it progresses through various phases in its life-cycle. The IEEE Standard defines software life-cycle as the period of time that starts when a software product is conceived and ends when the product is no longer available for use. Typically, it includes a requirements phase, design phase, implementation phase, test phase, installation and check-out phase, operation and maintenance phase, and sometimes a retirement phase.

Software productivity of any software development process as a whole depends upon the productivity as obtained during its progress through various life-cycle phases. The output of one life-cycle phase becomes input for the subsequent phase; thus, any metrics used to measure software productivity must predict the resources required for the subsequent phases of the life-cycle based upon the characteristics of the software at a given point in its life-cycle.

During the lifetime of a software system, it is not very uncommon to find that a life-cycle phase is iterated many times. Any measure for software productivity should be inversely proportional to the number of times a phase is iterated in the lifetime of a system because the iterations of most life-cycle phases induce iterations of some other phases thereby driving up the development cost.

It should be born in mind that productivity measurement result takes on meaning only when compared with other results of productivity measurements. Things which a productivity measurement may be compared with include the following.

1. Actual historical data
2. Existing standards in the industry
3. Other currently available actual data

Changes in future productivity metrics depend on the extent to which industry participates in using standard metrics. (We do not have such standard metrics yet, but IEEE has set up a working group on standards for software productivity metrics, and we may expect their recommendations by the middle of 1986.) Greater industry participation implies better understanding of the metrics. This in turn implies improvements in its definition which further provides more accurate measurements.

Management's viewpoint of software productivity is a little different because they have a different set of priorities and responsibilities. Certainly, they like the economic definition of productivity, but they view software productivity as the ability of a corporation to develop software. A metric for software productivity is useful to them if the measurements help each level of management to make appropriate decisions with regard to software development while keeping the economic goals of the company in mind. This means that for each level of management, productivity takes on a different meaning depending on the concerns of that level of management. The industry-wide concern may be technological capability and international competition; corporate level concern may be profits and market position; unit management's concern

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Costs</td>
<td></td>
</tr>
<tr>
<td>Cash</td>
<td>Rate of spending</td>
</tr>
<tr>
<td>Human effort</td>
<td>Present value of</td>
</tr>
<tr>
<td>Mechanical effort</td>
<td>money</td>
</tr>
<tr>
<td>Values</td>
<td></td>
</tr>
<tr>
<td>Financial</td>
<td>Present value of</td>
</tr>
<tr>
<td>Information</td>
<td>revenue market</td>
</tr>
<tr>
<td>Quality of Life</td>
<td>Present value of</td>
</tr>
<tr>
<td></td>
<td>information</td>
</tr>
</tbody>
</table>

TABLE 1—Fundamental elements of economic measurements (from Munson, J. B., and Yeh, R. T., Report by the IEEE at the Software Engineering Productivity Workshop, San Diego, March 1981).
may be resource allocation; project management may worry about actual progress against designated milestones; the project team’s concern may be the integration of individual products; and an individual’s concern may be the quality of the product and the work rate.

The variety of viewpoints and the fundamental difficulty of measuring software make it extremely difficult to establish a standard definition of software productivity. It is interesting to note that the IEEE Standard Glossary of Software Engineering Terminology defines many attributes of software including quality and reliability, but does not define software productivity.

SOFTWARE PRODUCTIVITY MANAGEMENT

Software productivity management is quite different from productivity management for a hardware manufacturing unit. This is true because

1. Software attributes that form software productivity issues are not all clearly understood, and
2. No appropriate metrics for measuring software productivity exist.

As if this were not enough, the problem of software productivity management is compounded by the fact that there is a shortage of computer scientists skilled in managerial issues. This is due to the fact that computer science is a relatively new area of expertise. In software houses, first line managers are usually chosen for their technical skills only, whereas in most cases the higher-level managers do not understand the software development process completely. The best way to alleviate this problem is to train more computer scientists to be managers.

There are many issues which can be considered central to software productivity management. With a little help from Munson and Yeh, I list some of them.

1. The role of software in the organization
2. The quality of software management
3. Understanding the approach to software development
4. How management fosters new ideas and creativity
5. The use of proper personnel for available tasks
6. The availability of software development tools

I will not discuss each of these issues in detail directly, but ask that you consider a productivity management process that concentrates on most of them.

A productivity management process consists of the following two fundamental issues discussed in detail below.

1. Efficient use of the following resources
   a. manpower
   b. materials
   c. technology
2. Satisfaction of consumer requirements related to the following
   a. cost
   b. schedule
   c. performance

Manpower

Management must use available manpower efficiently. To achieve this, it must identify individuals with special skills and assign them to projects so as to maximally exploit those skills. To get the most out of people, it is necessary to allow them direct access to top-level management, opportunity for outside recognition, personal professional growth, and increased benefits. A dual ladder with salary, status, and visibility for the most skilled software developers is required.

Software managers must also assume responsibility for a smooth transition from one phase of the life-cycle in the software development process to another. Frequently, various life-cycle phases overlap. Even so, the transition from a partially complete phase to a new one should be smooth. This is particularly important for transition from the implementation to the testing and maintenance phases where about 50 to 60% of software cost is incurred. Thus, software managers should make sure that, after implementation, there is an orderly transition to the maintenance phase, particularly with respect to the assignment of personnel already familiar with the project to subsequent phases. This maintains continuity during the most important transition in the software development process, thereby reducing the total system cost.

Material

Management needs to provide materials to software developers so that productivity can be consistently improved. These materials could be provided in many forms including increasing travel budgets for personnel to attend professional seminars and conferences that keep them abreast of the latest developments in the field; increasing the contents of the library to make sure that the latest research results are available to the software staff in time; improving office facilities; and providing more terminals and better secretarial support to the software engineers.

Technology

Management should provide the software staff with appropriate equipment and the means to use it such that they do not lag behind their competition. Management must realize that such tools are very important instruments to achieve productivity gains. It is important to note that hardware costs in the computing industry have gone down consistently over the last 15 to 20 years whereas software costs have continually increased. Management should recognize this and be prepared to buy the hardware needed to simplify the tasks of its software staff. Providing the latest technological developments in the software area to the software engineers can also increase productivity.

Cost Management

Management must keep the cost of software development and maintenance as low as possible. This provides the organi-
zation with a competitive edge and increased profit margin on the product. There are many ways of managing and controlling the cost of software development.

Schedule

Management at all levels must assure that software development is taking place within the appropriate budget and on schedule as the software progresses through its life-cycle phases. Project managers play a big role in maintaining and adhering to prescribed schedules.

Performance

Management must guarantee the performance of the delivered software at or above the required levels of customer satisfaction. Software productivity metrics are directly proportional to the performance level of software and customer satisfaction because both increase the value of the software. Additionally, as software productivity depends on software quality, it is the duty of managers at different levels to ascertain that at the end of each phase of the life-cycle, the developed software meets its requirements specifications. Verification and validation of software with respect to its specifications are important aspects that ensure good software performance in the future.

ADVANTAGES OF SOFTWARE PRODUCTIVITY MANAGEMENT

Management actions and reactions to the increasingly pervasive impact of software development on corporate products and processes have considerable impact upon a corporation’s ability to develop software, i.e., its productivity. Each level of management has a natural set of decisions to make which impact productivity.

When some results of productivity measurement become available to managers at all levels in the hierarchy, the decision making capability of management increases. Once the relationship of software to corporate products and processes is recognized, the productivity measure becomes the basis for evaluation of capitalization for software development, corporate strategy for acquiring additional technology, and evaluation of the performance of groups within the company and other organizations. Such measures also help to identify the specialized skills required to improve productivity and thereby support corporate goals.

Middle-level management can use productivity measures to explain to the top-level management how corporate goals can be achieved through software development. Measurements of productivity allow for the tracking of progress within budget and schedule. Sometimes, they can be useful in imposing a methodology on the development or maintenance process.

For software managers, productivity measures allow for the accurate prediction of group performance. This allows management to accurately project the cost and schedule of subsequent projects. At the upper levels of management, it may be used to decide on the life or death of a project or to evaluate the benefits of new tools, etc. Appropriate measures may also be used by managers to identify support requirements and to foster creativity, initiative, and innovation.

SUMMARY

The state of the art of software productivity, its associated measures, and its management is that it is still an art, not a science. We know quite a bit about the attributes that determine the cost of software production processes or “input” to the productivity measurement, but we have yet to identify many software attributes that determine the value of a software system. This, to my mind, is the biggest stumbling block in precisely defining software productivity. Lack of a precise definition results in our inability to understand software productivity and to measure it effectively.

It is clear that the study and management of productivity requires a measurement discipline. Measurement is as fundamental to programming as it is to any other area of engineering. If programming is to keep pace with technological advances in other areas of engineering, productivity improvement is critical. But, improvements in measurement techniques have not kept pace with the technological improvements in software engineering. Unless this happens, software costs will be ever increasing.

Software managers at all levels have to play a crucial role in these productivity improvements. It is not going to be an easy task, but the economic realities of the present day environment are forcing us to analyze and improve our software productivity management. We will have to look at traditional management techniques and improve and/or adjust them so that they can be applied to the software development process.

REFERENCES


From the collection of the Computer History Museum (www.computerhistory.org)