Software productivity measurement

by J. S. COLLOFELLO, S. N. WOODFIELD, and N.E. GIBBS
Computer Science Department
Arizona State University

ABSTRACT

Productivity is a crucial concern for most organizations. This is especially true for software development organizations. Although the term productivity is widely used, the difficulty of defining it leads to serious problems in productivity measurement. This paper will attempt to survey some current productivity measures for software development organizations and discuss their deficiencies. A theoretical productivity model that overcomes these deficiencies will also be presented. A practical productivity measure that exceeds current measures by including a quality component will also be described. Although this measure is only a small improvement over contemporary measures, it is a promising step in the direction of better productivity measurement.
INTRODUCTION

Productivity has become a major concern for most organizations attempting to survive in today's competitive market. Productivity improvements are being sought to produce a higher quality product at a lower cost. To assess the effect of steps taken to improve productivity, it is first necessary to have an objective measure. This measure normally assumes the form of a ratio of acceptable products developed to unit of time. Thus, measures such as manufactured components per hour are typical.

Productivity is a major concern for software development organizations. As in other organizations, productivity improvements are being sought to produce higher quality software at reasonable costs. High-level programming languages, software engineering techniques and tools, and new project management methods are all being proposed as ways to improve productivity. For an organization to assess the effects of these new approaches, an objective software development productivity measure is needed. With this type of measure a software development organization can assess its current productivity and determine if and where improvements are needed. Appropriate new techniques can then be applied and a determination made whether productivity was increased by the new techniques.

Productivity measures can also be used by many organizations for comparative purposes. Comparisons may be made of productivity differences between individuals, projects, or competitive organizations. In software development wide variations of productivity are common. Observations of large productivity differences for individuals performing the same task have been cited. When productivity is used to evaluate different software organizations, this comparison must be performed carefully. The reason for this is that although all software developers are producing programs, their products may be no more comparable than transportation manufacturers' products, such as bicycles and aircraft.

Another major use of productivity measures occurs in the product development time and cost estimation phase. An organization must have an estimate of production productivity to project development time and expense adequately. Although software development is often regarded more as research and development rather than manufacturing, costs must nevertheless be estimated. Although still in their infancy, most software cost estimation models require an estimate of the organization's software development productivity as a major input.

Thus, productivity measures are needed for most organizations, including software development organizations. The remainder of this paper will focus on productivity measurement for software development.

STATE-OF-THE-ART PRODUCTIVITY MEASURES

Several different approaches to software productivity measurement are currently used. Historically, the two major approaches are lines of code per programmer month, which expresses productivity in terms of work units; and cost per line of code, which expresses productivity in terms of cost units. These productivity measures can be used to assess the productivity of the entire software development effort. Other productivity measures for specific tasks in the software life cycle can also be used, such as pages of documentation per programmer month and number of test case runs per programmer month.

Another approach to productivity measurement is the software science approach. In this approach lines of code are decomposed into operators and operands in an attempt to produce a more invariant measure than lines of code. Based upon some ratios and counts of unique operators and operands and total numbers of operators and operands, some interesting work and complexity estimates have been formulated.

A new approach to productivity measurement, which abandons the lines-of-code approach completely, is quantifying the functions of programming. This approach concentrates on the number of external user inputs, inquiries, outputs, and master files to be dealt with by the program and uses a weighting approach to calculate a dimensionless number called a function point. A variation of this approach is a cost-per-function method, in which a function is defined as a program segment with a single input and output that performs a single transformation or action.

Of the major approaches to productivity measures just discussed, the measure applied most frequently is lines of code per programmer month. This measure as well as the others have several severe deficiencies, which will now be discussed.

DEFICIENCIES OF CURRENT PRODUCTIVITY MEASURES

One of the biggest problems with software productivity is the difficulty of measuring the work accomplished. In other industries, work can easily be measured in terms of products, such as radios produced per unit of time. In software the products are not as easy to quantify. This fact has left the lines-of-code measure as the traditional unit of work. This measure, as well as the software science and functionality types of measures, all suffer from the same problem of not representing adequate work units. Since the difficulty in developing software depends upon the problem being solved, implementation-based metrics such as lines of code, software science size or complexity measures, or function counts will vary with the complexity
of the task. Thus, all current software productivity measures fail to take into account the problem size of the task being performed.

Another major deficiency with contemporary measures is the difficulty of extending them into the maintenance phase. One measure for estimating maintenance productivity, reported by Boehm,\(^4\) consists of the average number of instructions that can be modified per man-month of maintenance effort. Modified instructions consist of new instructions or changed instructions. This measure suffers from the same deficiencies of the lines-of-code measure discussed for software development, and it also presents difficulties during maintenance. Since modification of a line of code requires understanding the program, changing it, and accommodating to a possible ripple effect, this measure may be a poor estimate of the amount of work performed.\(^5\)

Another severe problem of all current productivity measures is their failure to incorporate a quality component. The quality of a product must be reflected in productivity measurement. For example, if two organizations achieve the same lines-of-code measure for identical problems, but one organization produces a flawless system and the other is plagued with errors, the software productivity of the organizations should not be regarded as equal. This implies that defect removal costs must be factored into the productivity calculation. In general, other quality factors must also be incorporated into the productivity calculation, and none of them are included in any current measures.

Current productivity measures thus have several deficiencies that have prevented their being reliable or widely used. In the next section a proposed theoretical productivity model will be discussed that overcomes these deficiencies.

PROPOSED PRODUCTIVITY MODEL

For a productivity measure to gain universal acceptance, the problems discussed in the previous section must be resolved. The magnitude of these problems is so great that many feel that the development of such a measure is nearly impossible. Despite the difficulty of solving these problems, it is still essential to identify the characteristics of the ideal solution so that a direction of research can be planned and progress charted toward an eventual solution. It is to attain this objective that a theoretical productivity model is proposed.

The proposed productivity model is calculated as a function of problem size, resources consumed in production, and quality of the product produced. This productivity will be denoted as:

\[ P = f(PS, RC, Q) \]

where \(PS\) is the problem size, \(RC\) is the resources consumed, and \(Q\) is the quality of the product.

The problem size component of this productivity measure is meant to correspond to the magnitude and complexity of the task under development. It replaces the lines-of-code and functions measures of work in current productivity measures. This concept is essential to avoiding the current pitfall of productivity measures: rewarding long, inelegant solutions to problems. For example, the current lines-of-code productivity measure would conclude that a programmer who writes a 2000-lines-of-code solution to a problem in 1 month is twice as productive as a programmer who writes a 1000-lines-of-code equivalent solution to the same problem in 1 month. If, instead of lines of code, the problem size is used as a measure of work, the productivity of the two programmers would be regarded as equal, assuming the product was of equal quality.

The problem size for a particular software development effort must be calculated at the requirements level. This is essential, because problem size is independent of implementation size. This notion of problem size rewards cost- and timesaving measures such as reusing code and identifying simple and elegant solutions. The problem size notion is also consistent with the way productivity is measured in other industries. For example, in the automotive industry, productivity is measured by the assembly time of a car; the car corresponds to the problem size. It is not normally measured by the number of welds per hour, since this is an implementation metric. The number of welds per hour is, however, analogous to the lines-of-code measures; and they both significantly affect productivity once a development approach is selected. The key here is that productivity measurement must focus upon the problem size and not upon implementation approaches. The implementation approach is a factor that affects productivity but does not serve as a measure of work in its calculation.

The resources-consumed component of this productivity measure is equivalent to that used in contemporary measures. Thus, this measure may correspond to either a time or a cost calculation of the resources consumed to complete the development of a particular problem size. The time component may be measured in programmer months and the cost component in dollars.

The quality component of this productivity measure recognizes the necessity of evaluating the quality of a product developed. This concept is essential to avoiding the current pitfall of productivity measures: encouraging quantity of code produced without taking into account the reliability of the resulting software product. Unless the assumption is made that all software produced is of the same quality, this component must be a part of any productivity measure.

The measurement of software quality is a complex task. In fact, it is impossible to establish a single figure for software quality, since it has many attributes. Some typical software quality attributes are correctness, flexibility, portability, reliability, efficiency, integrity, and maintainability.\(^6\) In a given software development effort, some of these quality attributes may be stressed more than others. Several of the quality attributes, such as efficiency and portability, may also be in conflict with each other. The point here is that a single measure of software quality is unlikely; instead, software quality must be examined in terms of attributes affecting it.

This quality component and the problem size component of the proposed productivity measure must be combined to form a measure of the work to be performed. Thus, for a given problem size, the quality desired will determine the actual amount of work to be done. For example, consider a scientific
Thus, the only significant difference between this productivity calculated as the amount of rework time required to fix development and current measures is the incorporation of the reliability component. The reliability component will be calculable today will be described. This productivity measure will also not attempt to calculate a problem size, but instead can be calculated as the amount of rework time required to fix development methodologies that call for extensive reviews throughout the life cycle. These reviews take time, and with current productivity measures the added time detracts from productivity. Under the proposed productivity model, these reviews can increase productivity as long as they are cost effective.

Thus, the proposed productivity measure can be used in an organization to assess its current productivity and measure the effect of development improvement on productivity. This measure can be calculated at the end of the coding phase or at the end of the testing phase to assess the productivity of different life cycle phases. If it is calculated at the end of the coding phase, then error rework costs incurred during the testing and maintenance phases are factored into the productivity equation. If it is calculated at the end of the testing phase, then some interval may be selected, such as the next release, where error rework costs will be calculated.

This productivity measure may also be used (with caution) to compare productivity with that of other organizations. In addition to comparing work measures across organizations (which presents potential problems), error detection and correction mechanisms in the organizations must be examined. They must be examined because the rework component due to errors in the productivity calculation depends on error detection and removal costs. Thus, the productivity of two development organizations producing identical software with identical resources consumed may vary if the error detection and correction tools and techniques of the organizations vary. This forces meaningful productivity comparisons only among organizations with equivalent error detection and correction mechanisms.

Although this proposed productivity measure suffers many of the same problems as other contemporary measures and is limited in scope, the measure does advance the state of productivity measurement to include one facet of software quality. It is hoped that future research efforts will continue to progress toward the development of a complete productivity measure similar to the model proposed in this paper.

FUTURE RESEARCH

We plan to continue our research efforts on improving our pragmatic productivity measure. An attempt is being made to
include other quality factors besides reliability in its calculation. The problem size calculation problem is also being addressed. This research has led us into an examination of cost estimation models. Finally, the feasibility of productivity calculation based upon predictive quality metrics, which would enable productivity measurements to take place earlier, is being examined.

REFERENCES