Software Productivity Measurements

Weider D. Yu, D. Paul Smith, and Steel T. Huang

AT&T Bell Laboratories
1200 East Warrenville Road
Naperville, Illinois 60566

ABSTRACT

In the arena of software metrics, measuring productivity associated with software products and their development has historically been a difficult process. A reliable set of operational productivity measurement procedures is needed to analyze a software product and its development process. Software productivity measurements are the roots of the tree of productivity improvement. Without good measurements, progress is unlikely. To improve the quality of a software product, and the productivity of the development process, accurate measurements of input to and output from the development process must be made and appropriate productivity factors must be identified and understood.

Standardizing the measurement procedures within a development organization is a critical step. To achieve accurate software productivity measurements, the software development process should be well understood and the software productivity measurements should be closely linked to the development process and the development environment. This paper describes the software productivity measurement metrics, the important productivity factors, and some applications of the software productivity data for the 5ESS® switch developed for the United States market. The measurement of production rate and the study of productivity factors have established a solid foundation for the pursuit of productivity and quality improvement within the 5ESS switch development community.

1. INTRODUCTION

The software industry has been making slower progress in productivity improvements than the hardware industry. The problems existing in software production and maintenance, such as cost and schedule overruns, project cancellations before completion, and quality problems, have been major concerns for many large corporations and industries.

In the last decade of the twentieth century, a 20 percent improvement in software productivity will be worth $90 billion worldwide [1]. Obviously many corporations have been looking for better technologies to improve software productivity and quality, effective ways to reduce software costs, and more reasonable controls to run software development schedules. However, few good results have been achieved. Improving software productivity is a difficult process. The ways to effectively improve software productivity can be varied in different development environments. It is risky to pursue productivity improvement process without understanding where a project stands. Changes to the development process, or to the environment, that are needed to improve productivity should be selected based on expected impact and cost to implement.

Basically there are three major stages in the process of software productivity improvement: measuring, analyzing, and improving software productivity. The first stage, software productivity measurements, is the fundamental basis for the other two stages. With a set of reliable and operational software productivity metrics, the software product and its development environment can be analyzed to identify which productivity factors have great impact on software productivity and quality. Furthermore, the appropriate modifications to the development process and to the environment can then be chosen on the basis of these metrics and the identified productivity factors.

This paper describes the specific effort that has been taken to establish and improve the measurement of software productivity in the US 5ESS® Switch project, and also summaries some of the results of the effort.

2. SOFTWARE DEVELOPMENT ENVIRONMENT AND METHODOLOGY

The 5ESS Switch project is one of the most extensive software projects at AT&T Bell Laboratories. The total size of the delivered source code and internal support code is several million lines of code. The software development work for a typical release usually is about 75 to 80% of the effort compared to the hardware development work (20-25%).

The 5ESS Switch has a modern distributed architecture in software and hardware to better utilize the system resources in a real-time execution environment. The software architecture

Reprinted with certain editorial revisions from *AT&T Technical Journal*, Vol. 69, No. 3, pp. 110-120, May/June 1990. ©1990 AT&T.

implements a number of proven software engineering techniques, such as distributed computing, layered architecture, a relational database, and high level design and programming languages. The 5ESS Switch software is produced in the C programming language environment, augmented by some high-level design languages (e.g. Specification and Description Language, or SDL) and special purpose implementation languages.

Multiple 5ESS Switch software releases are under development simultaneously. Each release is an incremental functional addition to a very large existing base. Due to the high degree of complexity involved during design and testing, a feature usually requires fairly extensive effort in verifying requirement coverage. Typically, several hundred engineers are involved in a release of switching software.

A waterfall phased methodology is used to plan, design, code, and test each release of 5ESS Switch software. The methodology breaks down the planning and development process into the following phases:

1. Feature Planning,
2. Architecture Planning and Design,
3. Feature Requirements,
4. Feature Design,
5. Design Unit Design,
6. Coding,
7. Unit Testing,
8. Feature Testing,
9. System Verification, and
10. Site/Acceptance Testing.

The work accomplished at each phase is validated through formal peer reviews, code inspections, or structured extensive tests. The exit criteria at each phase must be satisfied before entering the next phase.

3. SOFTWARE PRODUCTIVITY MEASUREMENTS

Making improvements in the quality of a software product, and the productivity of the development process that is used to produce it, requires the ability to accurately measure software product attributes and the productivity factors that affect the software development process.

3.1 Software Productivity Measures

To measure software productivity, two fundamental measures must be established:

— The measure of the output from a software development process

— The measure of the input to a software development process.

Conceptually, software productivity can be expressed as a ratio factor derived from "output measure divided by input measure" [2]. In order to make the software productivity measure meaningful, clear and objective definitions and measurement methods on input and output measures are necessary.

Some typical output from a software development process are software source programs, internal development and external user documentations, and delivered software feature functions. Some typical input to a software development process are various types of development effort, computing and networking costs for developing and testing software, and various costs incurred on employee, such as benefits, vacations, office rent, education and training, and materials, etc.

The common measures used for the output can be software lines of code, documentation pages, and number of delivered software feature requirements. The common measures used for the input can be average person-years and dollars. The selection of input/output measures for software productivity in a development organization should be based on the availability of consistent and reliable measurement methods and tools. One of the purposes of the paper is to illustrate this concept by focusing software productivity on the effectiveness of direct software production process. Some cautious steps need to be taken to understand various factors associated with input/output measures. For example, if the input measure is dollars, then the inflation factor should be considered when computing software productivity data.

In the following sections, the definitions and measurement methods used for the major input and output measures in the US 5ESS Switch project will be discussed. The major output measure used is *software program size* and the major input measure used is *development effort*.

3.2 Software Program Size Measurement

Software productivity measurements require an accurate and consistent representation of software program size. The lack of standard software sizing methods for source programs written for a project could cause confusion and misunderstanding about the project size.

To establish a common ground for software productivity and quality studies, it is necessary to select an appropriate line-counting method for source programs, to decide which types of source programs should be counted to determine the size of a software product, and then to standardize it within a software development organization. Although in some cases, using lines of code as a measure is inherently paradoxical [3], it is a measure that can be well-defined and implemented in the 5ESS software development environment. In fact, most corporations in the software industry still use "lines of code" as the primary measure in doing productivity measurement work. Furthermore, because most of the source programs of the 5ESS software are written in the C language, the paradox caused by high-level and low-level programming languages does not apply to the 5ESS software development in most cases.

The software size measurement can be viewed at project and at program levels respectively.

3.2.1 Software Size Viewed at Project Level

When a software release is viewed at the project level, there are two types of code: *production code* and *support code*.

Production code is the software code developed for a release and delivered to customers as a part of a release. *Support code* is the software code developed for a release but not delivered to customers as a part of a release. Most support code is used for testing and generating production code. Support code development is an essential part of the development process. The size of the support code can be 15% to 25% of the size of the total code developed for a release.

We distinguish between production and support code because estimating the total effort required to produce software, both production code and support code are important. However, when estimating the memory size for a 5ESS release, only production code is important.

When the software code is viewed from the standpoint of source code structure, it can be further divided into five types of code: *base code*, *modified code*, *new code*, *bug code*, and *ported code* (see Figure 1). These definitions support the concept of software production code coupled to the effort which produces the code.

- *Base code* is the code developed in the previous releases and remaining in the current release.
- *Modified code* is the code originally developed in previous releases but modified in the current release for functional enhancement.
- *New code* is the code in new files written in the current release for new functions.

  We distinguish between new code and modified code in a large embedded software base because it is inherently more difficult to modify the existing code and test it than to create new code in new source files which interact with existing code in more structured ways.

- *Bug code* is the code written to fix bugs in the base, modified, and new code for the current release.

  We distinguish "bug code" from "new code" to allow the accurate tracking of change activity in the software source during development. The ability to distinguish bug code allows the development of objective quality metrics within the modules of a release.

- *Ported code* is the code developed in other projects or organizations and used or re-used in the current release.

  We distinguish "ported code" from other types of code to allow for the fact that some of the existing code may be re-used or delivered from organizations outside the effort tracking system. Ported code is incorporated into a system with far less effort in general than new code and modified code.

- *Total delivered code* is the code actually delivered to customers in a release. The total delivered code includes base code, modified code, new code, bug code, and ported code.
- *Net developed code* of a new release includes modified code, new code, and bug code.

  Most of the effort in developing a new software release is covered in this definition. The size definition is used in the study of software productivity and quality.

- *Net growth code* of a new release is the difference between the size of the new release and the old release.

  Net growth code of a new release is useful for memory sizing purpose.

3.2.2 Definition of 5ESS Switch Software Size

The key concepts and definitions of the size of a 5ESS Switch software release are illustrated in Figure 1. For a measure of "software size" to be useful for software productivity and quality studies, and in the software cost estimation process, it would have to correlate well with the measure of software development effort. The "net developed code" was found to have a strong relationship to the corresponding software development effort, and so its size was chosen as "software size."

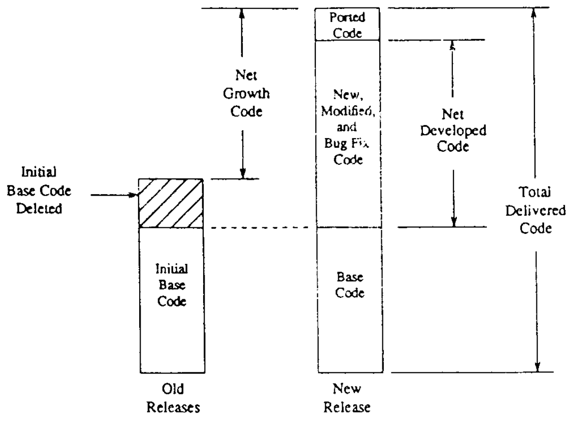

Figure 1. Illustration of 5ESS Switch release size versus release growth

The size of "net developed production code" has been used as a standard software size in the 5ESS Switch development community for computing productivity and quality metrics such as release and feature sizes, software production rates, and fault densities.

3.3 Software Size Viewed at Program Level

The method of counting "lines of code" in a software program has a number of variations throughout the software industry. Before knowing how to count "lines of code" in a software program, the meaning of "lines of code" needs to be defined. Without a proper definition of "lines of code," the *number* of lines of code of a software program is ambiguous.

Software program size is measured in thousands of non-commentary source lines (KNCSL). Each physical line (a source line ended with a carriage control character), that is not a comment or blank line, is counted as one non-commentary source line (NCSL).

In general, a NCSL may have one or more "logical" statements, or just a portion of a "logical" statement. It may or may not have embedded comments. Recognizing that individual programming style can affect the size and maintainability of code written in this way, a set of 5ESS coding standards has been adopted for the project.

The source code for the 5ESS Switch is managed by using the Change Management System (CMS) to recognize inserted and deleted code in new and modified source files, the Initial Modification Request (IMR) System to record new code and problems and fault reports, and the Source Code Control System (SCCS) to count numbers of physical lines inserted and deleted.

Using these systems, it is possible to recognize the difference between "new code" and code that subsequently modifies existing code (bug fixes) as well as identifying deleted code. Code counting tools have been developed to consistently produce metrics for software size. The source code counting algorithm gives "production credit" for new or modified software but does not give credit for rework.

3.4 Development Effort Measurements

The measure of input to the development process is "development effort." In general the development effort can include various kinds of effort generated by different activities (e.g. software development, project management, productivity and quality study, training), personnel (e.g. technical staff, administrative staff), and resources (e.g. computing equipment, communication facilities, office space and supplies). The effort information can be very complicated in a large development organization. It can be further complicated by the cost accounting and reporting schemes used in an organization.

To measure the development effort, it is necessary to consider the "availability" as well as the "correctness" of the development effort data in an organization. In the 5ESS project, the effort classifications are based on the organization accounting and reporting structure. The major development effort categories are mapped to a list of different effort charging numbers, which are used by staff to log their work hours. The basic measurement unit of effort expenditure is called the *Average Technical Head Count Year* (ATHCTY). Overtime effort is not explicitly accounted in the project.

3.4.1 5ESS Development effort definitions Basic definitions of 5ESS development effort are defined as follows:

- *Direct hardware development effort* represents all effort expended directly on hardware development, including circuit design, physical design, diagnostic and resident software (firmware) development.
- *Direct software development effort* represents all effort expended directly on software development from functional feature requirement through developer testing, plus post development feature support.
- *Support effort* includes all effort indirect to the hardware and software development effort, such as architecture, integration, load building, tools, system verification, field testing, training, field documentation, resource improvement, system labs, management and project coordination.

Total effort of development includes direct hardware and software development effort and support effort. The support effort may in fact exceed the direct software or hardware development effort in a release.

The "direct software development effort" is the primary measurement used to keep track of the actual effort expended on the software development work for a feature, which is a marketable unit of a 5ESS Switch release.

The effort expended on the tasks in the category of "support effort," such as architecture, integration, system verification, system labs, quality, project management, etc. is added to the direct software development effort. This effort charging scheme also applies to the "direct hardware development effort."

A diagram illustrating the effort charging and loading scheme is shown in Figure 2.

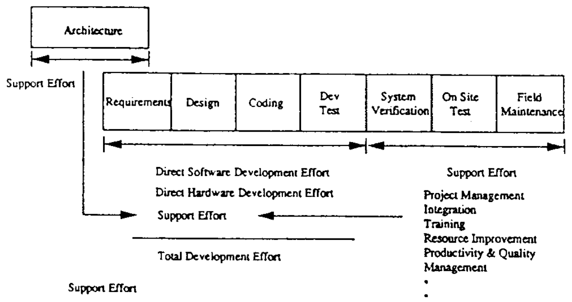

Figure 2. 5ESS® Effort Charging and Loading Scheme

4. 5ESS SOFTWARE PRODUCTION RATES

One metric used to measure the development process is called the *software production rate*. The *software production rate* can be expressed as:

$$\textit{Software production rate} = \frac{\textit{software program size}}{\textit{direct software development effort}}$$

The software production rate for a 5ESS release is defined as the net developed size of production code, (not including the code for diagnostic, diagnostic control, and resident software, which are included in the hardware productivity calculations), divided by the release "direct software development effort." The net developed code only includes new code, modified code, and bug code (base code and ported code are excluded). Software program size is measured in KNCSL. Direct software development effort is measured in ATHCTY.

The software production rate is only a small part of the overall productivity of the R&D development organization. The *overall productivity* can be expressed as:

$$\text{Overall productivity} = \frac{\textit{functionality}}{\textit{total R\&D cost of development}}$$

The numerator represents functionality delivered by the R&D development organization to customers and includes 5ESS release software, source information for generation of customer documents, database specifications for generation of office data, source information for external training, etc. Since this product is an aggregate of several different products, there is no single measurement unit which can be used for measuring the *functionality*. The *total R&D cost of development* includes hardware and software development staff effort, support staff effort, computing cost, and testing laboratory support resources. A 5ESS cost model has been developed for R&D managers to baseline the costs associated with the development of software and hardware of a 5ESS release. A good understanding of the costs is necessary to monitor, control, and reduce them. Currently more and more cost information has been collected from the project to facilitate continuing cost and productivity studies.

A relative comparison of 5ESS Switch release software production rates are shown in Figure 3. The releases span a period of 8 years.

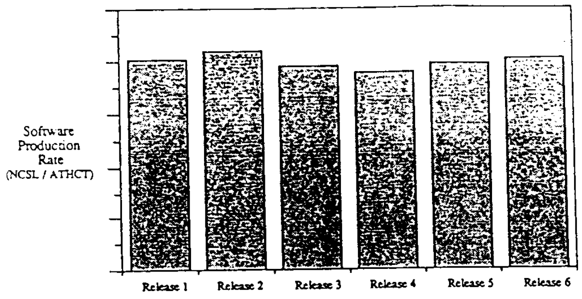

Figure 3. 5ESS® Switch Production Rates

The 5ESS-US software production rate has been stable over the past six releases. The development environment over this period of time has undergone continuous improvement, however the basic effort of the software developer is still involved with understanding the existing structure, designing and testing software at the C source level of complexity.

The fact that the software "base" for 5ESS-US is several million NCSL makes the job of designing and testing new software ever more difficult. The improvements in the development environment are evidently offset by this increasing complexity.

A feature within a 5ESS release represents a major functional capability such as Operator Services. A relative comparison of individual 5ESS feature software production rates in a recent release is shown in Figure 4.

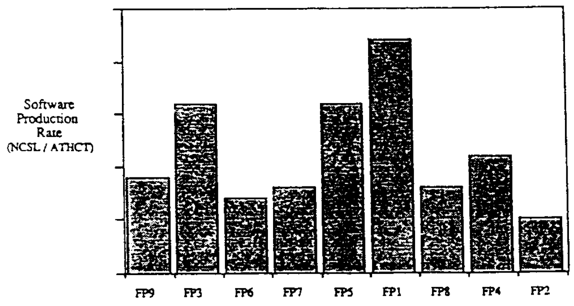

Figure 4. 5ESS® Feature Software Production Rate

Feature software production rates have more variation than the production rates for entire releases. The software production rates for features are impacted by productivity factors such as staff experience and feature interaction complexity.

4.1 Other Software Production Rate Views

The "net developed production code" used in the computation of the software production rates for releases and features includes three types of code by the software size definition given in the Section 3.1.1, *New code*, *Modified code*, and *Bug code*. For each release or feature, the source code counting techniques can generate size information for each of the three code components.

The software production rate for a release/feature can be further decomposed into three components for new code, modified code, and bug code respectively. This view can provide another level of understanding of the software production rate. The past experience indicates that modified code and bug code required more effort to develop than new code.

The concept of software production rate also can be extended to the arena of support code to understand the effectiveness of support code production. The denominator of the equation for the software production rate needs to be modified to reflect the related effort for developing the support code.

5. 5ESS SOFTWARE PRODUCTIVITY FACTORS

A list of potential 5ESS software productivity factors were identified by a group of experienced project planners, system engineers, architectural engineers, and software engineers. In 1987, a software productivity study was conducted among five 5ESS development laboratories. The study was based on the well defined productivity metrics: size, effort, software production rate, and the list of potential productivity factors. The lead software development engineers and managers were asked to provide information regarding the impact of productivity factors on the features which had shown high and low software production rates using methods similar to those employed by Jones [2] but adapted to the unique environment of 5ESS.

The study found that 5ESS software production rate was significantly impacted by requirement completeness, staff experience, software interface complexity, and testing environment stability.

Table 1 shows the productivity factors and the level of impact on software productivity as identified in the 1987 study.

SOFTWARE PRODUCTIVITY FACTORS	LEVEL OF IMPACT
Feature requirement completeness and stability	High
Feature interaction complexity	High
Staff experience	High
Feature development environment (tools, etc) impact	High
Feature hardware application novelty and change	Medium
Software architecture impact	Medium
Feature novelty and synergy with other features	Low
Feature program complexity	Low
Static and dynamic data impact	Low
Feature performance constraints	Low
Work environment	Low

Table 1. 5ESS Software Productivity Factors

The productivity factors and impact on productivity listed in Table 1 may be unique to the 5ESS development environment. Each project and its development environment may have its own set of productivity factors with different levels of impact on software productivity.

5.1 Software Productivity Factor Impact

Productivity factor impacts were measured either by quantitative methods or by qualitative multiple choices. Quantitative measurements were defined and applied to objective factors. Qualitative measurements were used to assess subjective factors. The following two examples illustrate quantitative and qualitative techniques.

A. Quantitative Measurement

- 5ESS static database impact?

  1. How many new static relations are created for the feature?
  2. How many existing static relations have population rule changes?
  3. How many existing relations have modified data attributes?
  4. How many existing relations have data tuple/element changes?

B. Qualitative Measurement

- What is development staff experience with this type of feature?

  1. Most development staff are experienced in this type of feature
  2. Half of development staff are experienced in this type of feature
  3. Most development staff are inexperienced in this type of feature

The sensitivity of two high impact productivity factors are illustrated in Figures 5 and 6. These factors were identified by asking experienced managers to rate all the factors in Table 1 for a large number of features whose software production rates were known. The correlation between software production rates and productivity factors were then compiled for the project.

Figure 5 illustrates the impact of staff experience. As illustrated, the difference in production rate between "most development staff are experienced" and "most development staff are new to 5ESS" is about 100%. Figure 6 illustrates the impact of feature interaction complexity. As illustrated, the difference between "minor interactions with other features" and "significant interactions with other features" is about 40%.

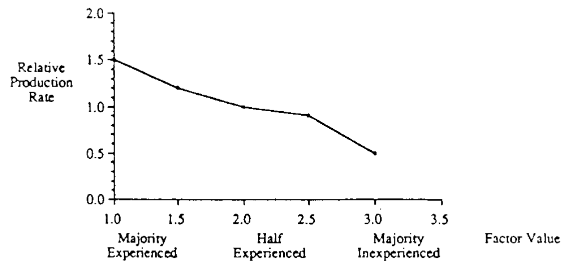

Figure 5. Staff Experience Factor

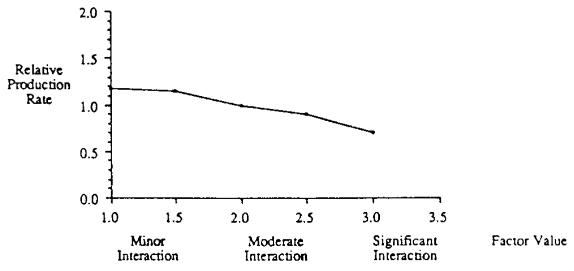

Figure 6. Feature Interaction Complexity

6. APPLICATIONS OF SOFTWARE PRODUCTIVITY MEASUREMENTS

The results of software productivity measurements have been applied in the following areas:

1. The productivity measurements and the identified software productivity factors have been used to develop a 5ESS estimation model, 5ESS SIZER [4]. 5ESS

historical data, based on the productivity measurements, has been used in the estimation model as a reference. The estimation model has been used in the 5ESS planning estimation process to assist project managers and feature estimators to plan and staff the projects consistently [5].

2. Software size measurement has been used extensively as an important normalizing factor in baselining the major characteristics of the 5ESS releases and development process, such as a variety of code sizes, fault density profiles, software production rates and test densities [6]. The consistent sizing measurement allows the project to develop and implement guidelines and entry/exit criteria with respect to inspection and review preparation effort, number of tests, testing time in lab hours, number of errors found per phase, etc. In addition, the size measurement has been used to derive other important quality metrics such as churn rate (frequency of code changes) and bad fix rate (frequency of fixes caused by other fixes). The effort measurement has been used by the project management to keep track of the direct software/hardware development by feature and by major development phases.

3. The identification of high impact productivity factors has helped the project focus its process improvement efforts. The following are some examples of improvements made in the high impact areas shown in Table 1:

   - *Feature requirement completeness and stability.* A requirement traceability methodology has been introduced to the 5ESS development community to emphasize requirement specification and requirement verification. It assists 5ESS engineers to identify requirement faults and omissions during the earlier development stage and to facilitate further feature design and testing work.

   - *Feature interaction complexity.* For some large and complex features, a feature interaction matrix is constructed to show all the interactions with other features. This improves the effectiveness of feature design and testing.

   - *Staff experience.* Critical expertise and shared resources were reorganized into functional units to better utilize subject experts. Management is currently considering a proposal, which gives additional incentives to engineers who are willing to stay on the same job function for a specific period of time.

   - *Feature development environment impact.* A number of development tools have been introduced to improve the development environment, such as DOC, which allows multiple engineers to develop a document simultaneously.

7. CONCLUSIONS

Establishing a reliable and operational software productivity measurement procedure is a critical step in the process of pursuing software productivity improvement. Productivity measurements actually are the roots of the tree of productivity improvement.

While the measurement of "software program size per unit of development effort" has some known problems as a productivity metric, it is a useful metric to study the software development process where the development organizations use a common source programming language. Studies of those productivity factors which impact the metric "software production rate" identified productivity and quality improvements which could be made in the 5ESS development process. Similar studies of other development processes would be expected to identify other factors which may or may not be similar to those identified for 5ESS.

8. REFERENCES

[1] [Boehm, 1981]. Barry W. Boehm, *Software Engineering Economics,* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981

[2] [Albrecht, 1984]. Alan J. Albrecht, *AD/M Productivity Measurement and Estimation Validation - Draft,* IBM Corp., Purchase, New York, 1984

[3] [Jones, 1986]. T. Capers Jones, *Programming Productivity,* McGraw-Hill, Inc., New York, 1986

[4] [Lehder, Smith, Yu, 1988]. Wilfred E. Lehder, Jr., D. Paul Smith, Weider D. Yu, *Software Estimation Technology,* AT&T Technical Journal, Volume 67, Number 4, July/August 1988

[5] [Yu, 1990]. Weider D. Yu, *A Modeling Approach to Software Cost Estimation,* IEEE Journal on Selected Areas in Communications, Volume 8, Number 2, February 1990

[6] [Yu, 1990]. Weider D. Yu, *Software Productivity Measurements and Estimation,* IEEE International Symposium: Software Quality and Productivity in the 1990's, Callaway Gardens, Georgia, April 1990