SOFTWARE PRODUCTIVITY MEASUREMENTS

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

Weider D. Yu, D. Paul Smith, *and* ***Steel T. Huang*** *are with AT&T Bell Laboratories at Indian Hill, Naperville, Illinois. Mr. Yu is a distinguished member of technical staff in the 5ESS® Switch Systems Quality Department, where he works on software productivity and quality; software estimation, cost analysis, and modeling; software requirements traceability methodology; and software development process effectiveness. He received a B.S. in mathematics from Fu-Jen Catholic University, Taiwan, an M.S. in computer science from the State University of New York at Albany, and a Ph.D. in computer science from Northwestern University. He joined AT&T in 1983. Mr. Smith is head of the Switching Systems Quality Department, where he is responsible for quality management of the software development process for the 5ESS*
(continued on page 120)

Measuring productivity associated with software products and their development historically has been difficult. Reliable operational measurements are needed to analyze a software product and its development process. To improve both the quality of the software product and the productivity of its development process, the input to and output from the development process must be measured accurately and appropriate productivity factors must be identified and understood. To measure productivity accurately, the software development process must be well understood and the 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.

Introduction

The history of information processing clearly indicates that the software industry has been making slower progress in productivity improvements than the hardware industry. Problems in software production and maintenance, such as cost and schedule overruns, project cancellations before completion, and inadequate quality, have been major concerns for many large corporations and industries.

In the 1990s, 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 on 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 without understanding where a project stands. Changes to the development process, or to the environment, that are needed to improve productivity should be selected on the basis of 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 measurement, is the basis for the other two stages. With a set of reliable software productivity metrics, the software product and its development environment can be analyzed to identify those productivity factors having great impact on software productivity and quality. Furthermore, the appropriate modifications to the development process and the environment can then be chosen on the basis of the metrics and the identified productivity factors.

This paper describes the specific steps that have been taken to establish and improve the measurement of software productivity in the U.S. 5ESS switch project and summarizes some of the results.

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 amounts to about 75 to 80 percent of the total effort; the hardware development work accounts for only 20 to 25 percent.

The 5ESS switch has a modern distributed architecture in both software and hardware to better utilize the system resources in a real-time execution environment. The software architecture 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 (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. Because of the complexity of design and testing, a software feature usually requires fairly extensive effort to verify that it covers the requirements. 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 waterfall model, described by Royce in 1970,[2] views software development as proceeding through successive phases. It is still the best known and most widely used framework for the software development process. 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
10. Site/acceptance testing.

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

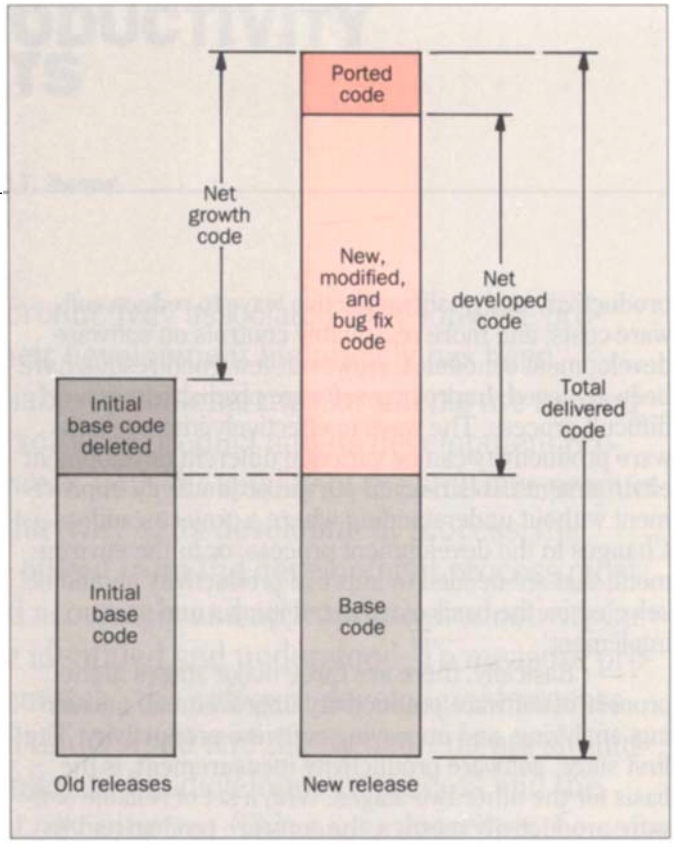

Figure 1. Key concepts relating to the size of a 5ESS switch software release. Total delivered code for a new release comprises new code; previously developed code, whether carried over from the previous release, modified from it, or imported from another project; and code developed to fix bugs in new and old code.

Software Productivity Measurements

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

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 the software development process.

Conceptually, software productivity can be expressed as a ratio factor, derived from the output measure divided by the input measure.[3] To make the software productivity measure meaningful, clear and objective definitions and measurement methods for input and output measures are necessary.

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

Software Program Size Measurement. Software productivity measurements require an accurate and consistent representation of software program size. A 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:

1. Select an appropriate line-counting method for source programs
2. Decide which types of source programs should be counted to determine the size of a software product
3. Standardize the counting method and types within a software development organization.

Although, in some cases, using lines of code as a measure is inherently paradoxical,[4] it is a measure that can be well-defined and implemented in the 5ESS switch software development environment. In fact, most corporations in the software industry still use lines of code as the primary measure in productivity measurement work. Furthermore, because most of the source programs of the 5ESS switch software are written in C language, the paradox caused by mixed high-level and low-level programming languages does not apply in most cases.

Software size measurement can be viewed at both project and program levels.

Software Size Viewed at Project Level. When a software release is viewed at the project level, there are two types of code: *Production code* is the software code developed for a release and delivered to customers as part of a release. *Support code* is the software code developed for

Panel 1. Definitions of Terms in Figure 1

Base code is the code developed in previous releases and remaining in the current release.

Modified code is the code originally developed for 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 that interact with existing code in more structured ways.

Bug fix code is the code written to fix bugs in the base, modified, and new codes for the current release. We distinguish bug fix code from new code so that change activity in the software source during development can be tracked accurately. The ability to distinguish bug fix code allows the development of objective quality metrics within the modules of a release.

Ported code is code developed in other projects or organizations and used or reused 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 reused 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 fix code, and ported code.

Net developed code of a new release includes modified code, new code, and bug fix 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 purposes.

a release but not delivered to customers as a part of the 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 support code can be 15 to 25 percent of the size of the total code developed for a release.

We distinguish between production and support code because for estimating the total effort required to produce software, both production code and support code are important, but for 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 (Figure 1):

- Base code
- Modified code
- New code
- Bug fix code
- Ported code.

These subdivisions support the concept of software production code coupled to the effort that produces the code. These terms and others in the figure are defined in Panel 1.

Definition of 5ESS Switch Software Size. 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."

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.

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 lines of code in a software program can be

counted, the meaning of lines of code must 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 noncommentary 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 noncommentary source line (NCSL).

In general, an NCSL may have one or more logical statements, or just a portion of a logical statement. It may or may not have embedded comments. In recognition of the fact that individual programming style can affect the size and maintainability of code, a set of 5ESS switch 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) and to identify 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.

Development Effort Measurements. The measure of input to the development process is *development effort*. In general, the development effort can include various kinds of 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). Information about the effort can be complicated in a large development organization. It can be further complicated by the cost accounting and reporting schemes used in the organization.

To measure the development effort, it is necessary to consider the *availability* as well as the *correctness* of the development effort data in the organization. In the 5ESS switch project, the effort classifications are based on the organization accounting and reporting structure. The major development effort categories are mapped to a list of 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.

Definitions. The following definitions apply to the 5ESS switch development effort:

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

The 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 can be added to the direct software development effort through loading. This effort charg-

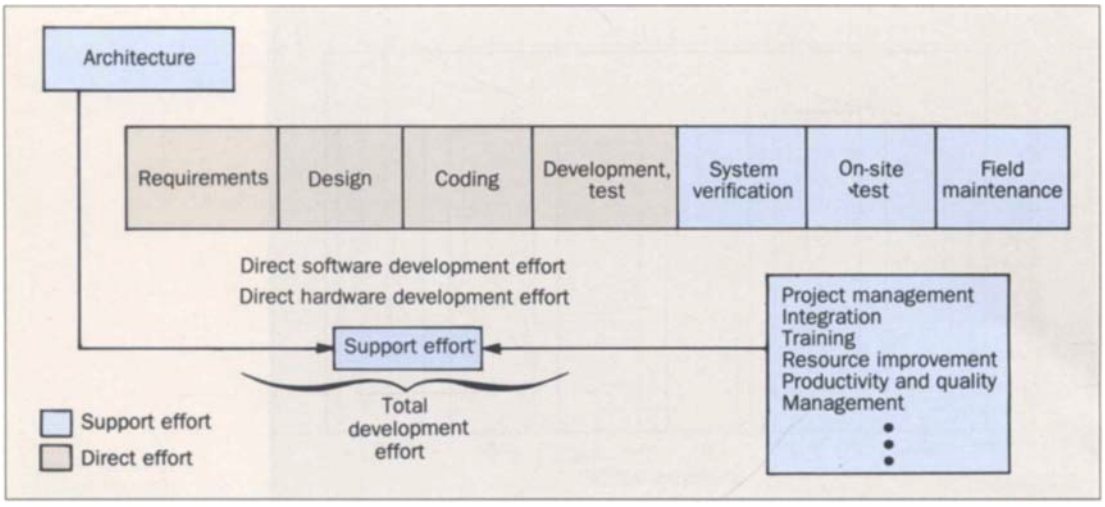

Figure 2. Effort charging and loading scheme for 5ESS switch software development. Support effort (shaded areas) is added to the direct software development effort. The support effort often exceeds the direct effort.

ing scheme also applies to the direct hardware development effort.

The diagram in Figure 2 illustrates the effort charging and loading scheme.

5ESS Switch Software Production Rates

One metric used to measure the development process is called the *software production rate*, which can be expressed as

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

The software production rate for a 5ESS switch release is defined as the net developed size of production code (not including the code for diagnostic, diagnostic control, and resident software, which is included in the hardware productivity calculations) divided by the release direct software development effort. The net developed code includes only 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

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

The numerator represents functionality delivered by the R&D development organization to customers and includes 5ESS switch 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 that can be used for measuring the functionality. The denominator, 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 switch cost model has been developed for R&D managers to baseline the costs

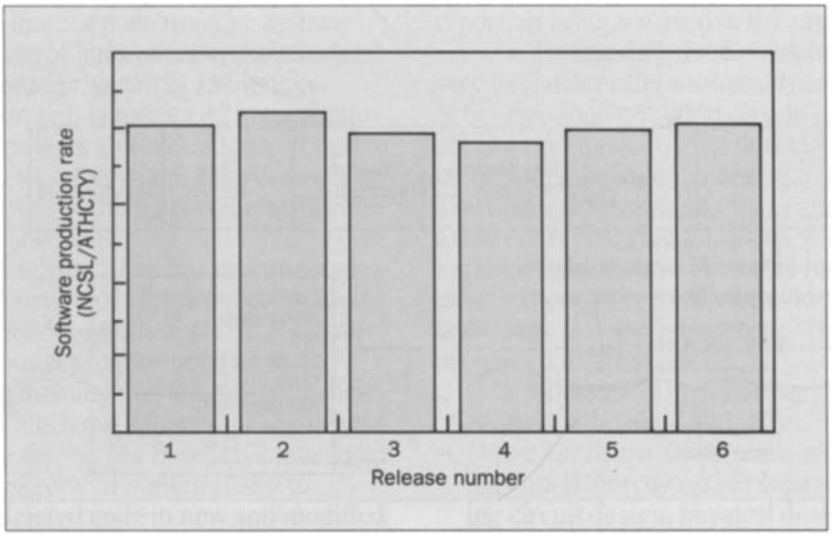

Figure 3. Software production rates for 5ESS switch software releases. The rate is fairly stable, especially for Releases 3 to 6, despite increasing software complexity, largely because of improvements in the development environment. The production rate is measured in terms of NCSL and ATHCTY.

associated with the development of software and hardware of a 5ESS switch release. A good understanding of the costs is necessary to monitor, control, and reduce them. Currently, more and more cost information is being collected from the project to facilitate continuing cost and productivity studies.

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

The software production rate for the U.S. 5ESS switch 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 and designing and testing software at the C source level of complexity.

The fact that the software base for the U.S. 5ESS switch 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 switch release represents a major functional capability such as operator services. A comparison of production rates for individual software features for the 5ESS switch in a recent release is shown in Figure 4.

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

5ESS Switch Software Productivity Factors

A list of potential 5ESS switch software productivity factors has been 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 switch development laboratories. The study was based on 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 that

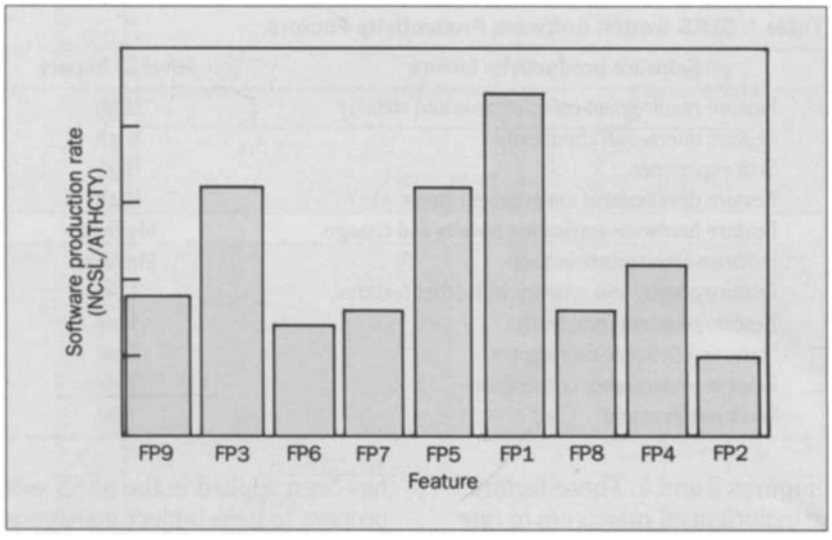

Figure 4. Unlike software releases, software features for the 5ESS switch vary widely in production rates. This is because features are strongly affected by the level of staff experience, by the complexity of interaction with other features, and by other factors.

had shown high and low software production rates. The engineers and managers used methods similar to those employed by Jones,[4] but which were adapted to the unique environment of the 5ESS switch.

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

Table I shows the productivity factors and the level of impact on software productivity as identified in the 1987 study. The productivity factors and impact on productivity listed in the table may be unique to the 5ESS switch development environment. Moreover, each project and development environment may have its own set of productivity factors with different levels of impact on software productivity.

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.

Quantitative measurements assessed impact on the 5ESS switch static database, for example, by eliciting answers to questions such as these:

- How many new static relations are created for the feature?
- How many existing static relations have population rule changes?
- How many existing relations have modified data attributes?
- How many existing relations have data tuple/element changes?

Qualitative measurements sought information on staff experience with feature-development projects, for example, by asking subjects to choose a response from lists such as the following:

- Most development staff are experienced in this type of feature
- Half of development staff are experienced in this type of feature
- Most development staff are inexperienced in this type of feature

The sensitivities of two high-impact productivity

Table I. 5ESS Switch Software Productivity Factors

Software productivity factors	Level of impact
Feature requirement completeness and stability	High
Feature interaction complexity	High
Staff experience	High
Feature development environment (tools, etc.)	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

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

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

Applications of Software Productivity Measurements

The results of software productivity measurements have been applied to estimation, baseline, and process improvement.

Estimation. The productivity measurements and the identified software productivity factors have been used to develop a 5ESS switch estimation model, 5ESS Switch Sizer.[5] Historical data for the 5ESS switch, based on the productivity measurements, have been used in the estimation model as a reference. The estimation model has been applied in the 5ESS switch planning estimation process to help project managers and feature estimators to plan and staff the projects consistently.

Baselining. Software size measurement has been used extensively as an important normalizing factor in baselining the major characteristics of the 5ESS switch releases and development process, such as a variety of code sizes, fault density profiles, software production rates, and test densities. The consistent sizing measurement allows workers on 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, and number of errors found per phase. 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). Project management has used the effort measurement to keep track of direct software/hardware development by feature and by major development phases.

Process Improvement. The identification of high-impact productivity factors has helped the project staff focus their process improvement efforts. The following are examples of improvements made in the high-impact areas shown of Table I:

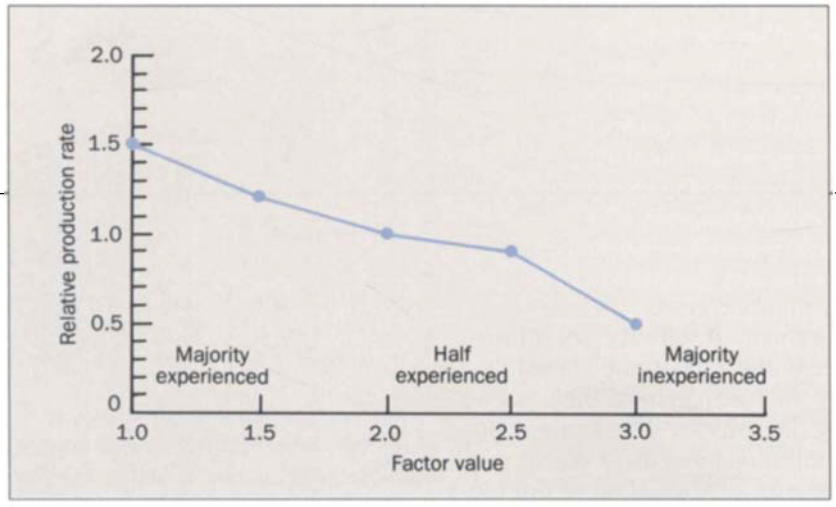

Figure 5. Software productivity for features is highly sensitive to staff experience; for experienced staff it is twice that for inexperienced staff.

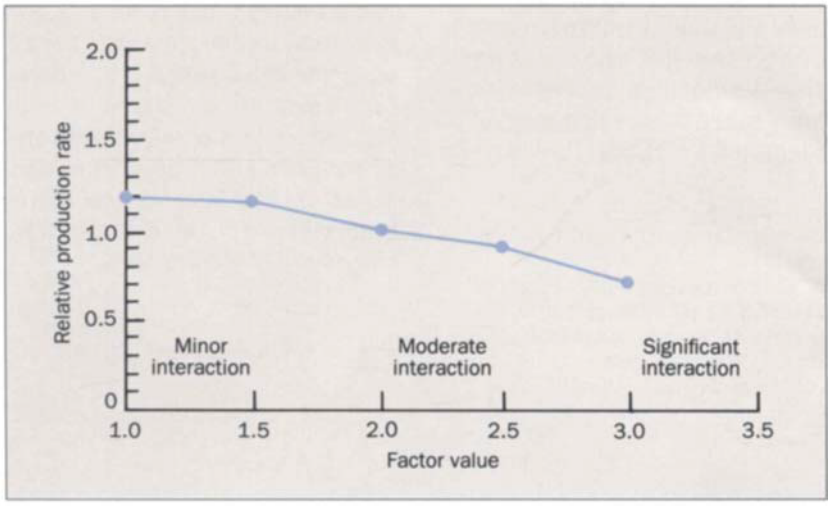

Figure 6. Productivity for software features is also highly sensitive to the extent to which a software feature interacts with other features. The rate with significant interaction is roughly three-fifths that with minor interaction.

- *Feature requirement completeness and stability.* A requirement traceability methodology has been introduced to the 5ESS switch development community to emphasize requirement specification and requirement verification. It helps 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 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 that would give additional incentives to engineers who are willing to stay on the same job function for a specific period of time.
- *Feature development environment.* A number of development tools have been introduced to improve the development environment. An example is DOC, a software tool that allows many engineers to develop a document simultaneously.

Conclusion

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 nevertheless a useful metric for studying the software development process in development organizations that use a common source programming language. Studies of those productivity factors that affect the software production rate metric identified productivity and quality improvements that could be made in the 5ESS switch development process. Similar studies of other development processes would be expected to identify other factors that may or may not resemble those identified for 5ESS.

References

1. B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
2. W. W. Royce, "Managing the Software Development of Large Software Systems," *Proceedings of IEEE WESCON*, August 1970.
3. A. J. Albrecht, "AD/M Productivity Measurement and Estimation Validation," IBM Corp., Purchase, New York, 1984.
4. T. Capers Jones, *Programming Productivity*, McGraw-Hill, New York, 1986.
5. W. E. Lehder, Jr., D. P. Smith, W. D. Yu, "Software Estimation Technology," *AT&T Technical Journal*, Vol. 67, No. 4, July/August 1988, pp. 10-18.

Biographies (continued)

switch. He previously worked on design and development, project management, and quality improvement of 5ESS system hardware. He received a B.S.E.E. degree from Brigham Young University and M.S.E.E. and Ph.D. degrees from Stanford University. He joined AT&T in 1968. Mr. Huang is supervisor of the 5ESS Switch–U.S. Process/Current Engineering Department. He is responsible for planning, definition, modeling, and analysis of software quality and productivity metrics for the 5ESS switch development project. He received a B.S. in mathematics from National Taiwan University and a Ph.D. in statistics from the University of North Carolina at Chapel Hill. He joined AT&T in 1979.

(Manuscript received March 5, 1990)