REPORT:

Mark E. Graden *and* ***Palma S. Horsley*** *are members of the information systems staff in the Quality Systems Engineering department at the Network Software Center in Lisle, Illinois. They assist NSC organizations in developing integrated quality systems. Mr. Graden received a B.S. in statistics from Purdue University. He joined AT&T in 1983. Ms. Horsley received a B.S. in mathematics from the University of Montana. She joined AT&T in 1983.* ***Thomas C. Pingel*** *is department chief, Quality Systems Engineering. He is responsible for quality management systems and quality improvement at the Network Software Center. He received a B.S. in mathematics from Purdue University and a M.S. in operations research from Lehigh University. He joined AT&T in 1969.*

THE EFFECTS OF SOFTWARE INSPECTIONS ON A MAJOR TELE-COMMUNICATIONS PROJECT

Introduction

Software inspections are a highly formalized and rigorous technique used for the identification and removal of errors in software products.[1,2] Faithfully applied, they have beneficial impact on the productivity and quality of a project. As a result, software inspections were selected as a critical ingredient in the overall Software Quality Assurance Plan to guide the development and evolution of a major, real time telecommunications software project.

This paper describes how the results of software inspections have been used to explain differences in end-product quality and identifies useful techniques for applying the results of software inspections to manage the software development process.

Background

Although there are few, if any, measures of end-product software quality which are agreed upon and widely used within the software industry, there are even fewer measures which are able to quantify the quality of a software product as it moves through its development life cycle (Figure 1). Of particular interest is the quality of the product as it leaves the design phase and enters the implementation or coding phase. It is at this point that the quality of the software must already be built-in.

With this in mind, it became one of the primary objectives of the overall Software Quality Assurance Plan to instrument the front end of the development process so that objective qualitative and quantitative data could be captured for later study of their relation to end-product quality. If any statistically significant results could be observed, there would then be the opportunity to more effectively manage the quality of future software releases as they moved through development.

The characteristics of a formal life cycle are such that each of the phases typically is defined by a series of inputs, specific activities which transform the inputs, and one or more outputs or deliverables. For example, these deliverables may be in the form of requirements documents, design documents, test plan documents, or code. Typically there are completion criteria for each deliverable which are specified to assure uniform and consistent progress of the product. A software inspection is a common technique used to specify completion of the deliverables from coding and precoding phases. The software inspection identifies errors in the various deliverables and requires rework and verification by the inspection moderator and producer before the deliverable is considered complete. Careful instrumentation of the inspection process can provide a rich body of data about the number and nature of errors identified at the inspection as well the environment in which the inspection was held.

For the project under study, the instrumentation of the inspection process, the detailed inspection procedures, and the overall software development life cycle have been documented in a comprehensive Software Quality Assurance Plan. Initially, all of the development

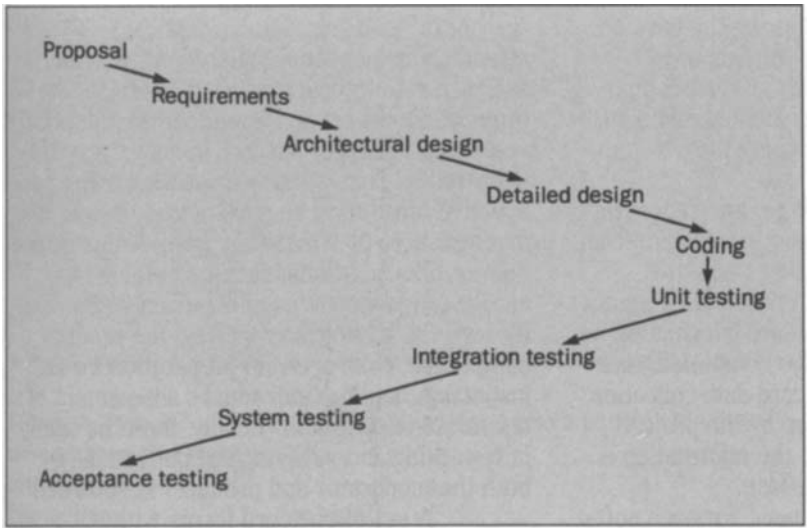

Figure 1. Typical software development life cycle.

staff were trained in the use of inspections as well as their overall responsibilities under the plan. A strong management commitment assured adherence to the plan and the availability of the data necessary for subsequent analysis.

Data Collection

The development and deployment of our software involves a staff of more than 300 technical personnel. Any release of our product is composed of a collection of features which provide new or enhanced functional capabilities for the end users of the product. Separate departments are responsible for the planning, engineering, development, testing, installation, and support of the software. Within the development area alone, the evolution of a feature requires the interfacing of multiple departments, each having responsibility for a specific software subsystem (e.g., recent change, call processing, etc.). The nature of this complex organization, both in size and in the vast communication paths that must be established to successfully develop and deploy any given product, necessitates a great expenditure of effort to collect, verify, analyze, and use data relating to both process and product quality throughout the life cycle. Furthermore, the difficulty in assuring consistently reproducible data under the conditions of diverse staffing and product profiles is evident.

Despite these environmental constraints, a significant body of data is collected on an ongoing basis for the purposes of confirming that the standard practices outlined in the Software Quality Assurance Plan are being followed and identifying potential problem areas for subsequent corrective actions. It is important to note that at the onset of the Software Quality Assurance Plan, the life cycle of a typical major release exceeded 20 months. Because of this interval and the lack of prior results in the analysis of software inspections, our strategy was to collect any data that might affect end-product quality.

The following paragraphs highlight three important classes of data which form the basis for the analyses described in this paper. Included are not only the types of information collected, but also the methods of collection and the sources of the data.

Project Management Data. The success of the data collection efforts is, to a great extent, dependent upon the project management organization. Their role in defining and monitoring a plan by which a product is developed and deployed is crucial. Accordingly, all product components are specified and tracked via a unique identifier. The project management organization is also responsible for authorizing the set of deliverables to be completed by the

appropriate organizations during the development of each product. Finally, data are gathered on the size of each product component and the effort, both estimated and actual, expended to develop and deploy the component.

To collect, assimilate, and report this vast amount of data, a project tracking mechanism is utilized. On a monthly basis, the organizations responsible for individual product components provide up-to-date information on size, schedules, deliverables completed, and effort estimates via a standard data collection form. After data verification by the project management organization, the information is entered into a central data base.

Software Inspection Data. For each software inspection that is conducted during precoding and coding phases, two types of information are provided: product and process data. Product data consist of general descriptive information: a product component identifier assigned by the project management organization, the type of deliverable being inspected (requirement document, design document, test plan document, or code), and producer information (name, organization, years of experience in the telecommunications field, and a rating of familiarity with the technical aspects of the product component). Also collected on a product basis is a detailed description of each error identified during the inspection. The description includes the category of the error and the source of the error. Also recorded are the final disposition upon inspection of the deliverable, acknowledgement by the moderator that the deliverable has been subsequently reworked, and any errors that can not be resolved.

Process data include the date and duration of the inspection plus inspection participant information. Those individuals attending an inspection must identify their assigned role (moderator, reader, scribe, or required reviewer) and any required functional responsibilities they assume in inspecting the deliverable. That is, each must denote the perspective (customer, requirements, design, test, maintenance) by which they have evaluated the deliverable. Additional data are retained for each individual on level of experience, familiarity with the technical aspects of the product component, time spent in preparation for the inspection, and the individual's assessment of deliverable disposition. Finally, the time spent in reworking and verifying the deliverable by both the moderator and producer is recorded.

A set of standard forms with documented instructions is used to manually collect all inspection data. It is the responsibility of the moderator and the producer to verify that the information is both complete and correct before the data are entered into an inspection data base. Furthermore, an inspection is not recognized as complete until all information has been submitted.

Fault Data. Once a product, in the form of code, reaches the testing phase, all faults are documented via a problem-reporting and corrective-action-tracking mechanism. For each fault, the date found, the phase of the life cycle (unit test, integration test, system test, acceptance test, released to customer), the fault category, the fault severity, the source of the fault (in terms of the product component identifier), and the impact to the customer are detailed on a standard data collection form by the individual who located the fault. After the fault evaluation and correction, any necessary updates to the above information are reflected on a corrective action implementation form. The fault data are verified by a review board

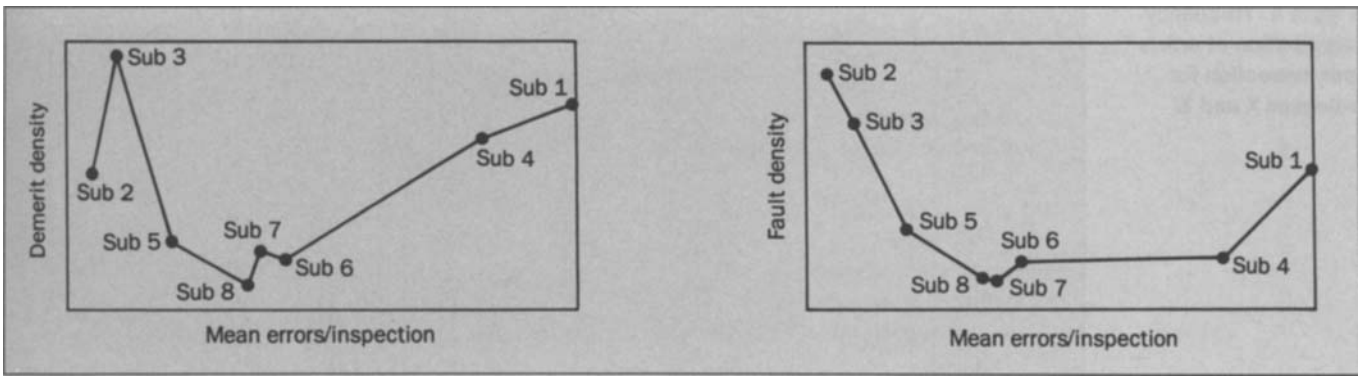

Figure 2. Plot of demerit density versus mean errors per inspection.

Figure 3. Plot of fault density versus mean errors per inspection.

before being entered into a centralized data base.

Analysis

Thorough analysis of both product and process data is imperative in order to understand and improve the software development process. Since rigorous product inspections were implemented to assure quality, an analysis of the effects of inspections on software quality was conducted.

The purpose of this study was to determine:

- Those inspection characteristics which are critical to the end-product quality of our software
- A method of accurately identifying, as early as possible, software features and subsystems of potential quality concern

The basis for the analysis was data collected from inspections (which included inspections of requirements, design, and test plan documents as well as code) along with field and test data for a major product release, denoted by release X. Also analyzed were inspection data from the release under development, denoted by release Y, in order to identify, if possible, features and subsystems for quality concern.

The measures of end-product quality used were:

- Fault density—faults normalized by size against new and changed code after completion of integration testing
- Demerit density—demerits normalized by size against new and changed code after completion of integration testing (where demerits are a function of fault severity)

Initial investigations of the data identified a number of promising relationships between inspection results and end-product quality. Plots of mean errors detected by subsystem against fault/demerit density consistently showed a parabolic relationship (Figures 2 and 3). Plotting the data by feature showed similar results. These plots together with the observation that the process mean of errors detected per inspection was located near the lowest fault/demerit densities led us to the following hypothesis:

Software features and subsystems with errors per inspection "different" from the process mean have the highest fault and demerit densities.

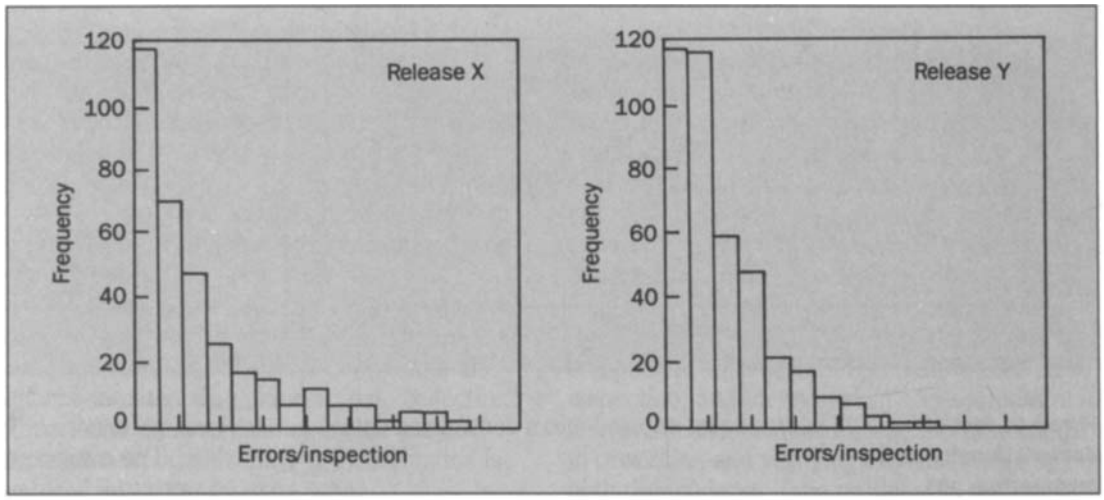

Figure 4. Frequency distribution of errors per inspection for releases X and Y.

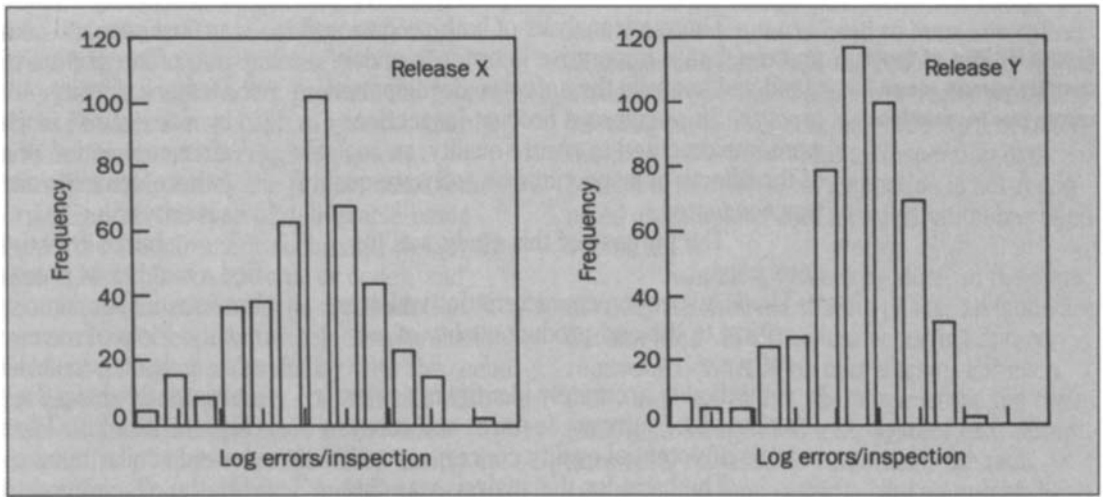

Figure 5. Frequency distribution of the logarithm of errors per inspection for releases X and Y.

Figure 4 shows the distribution of inspection errors for both release X and release Y development. Note that the distributions are very similar, each highly skewed with a large spread in the data values. In order to compare the means by classic statistical techniques, it was necessary to transform the data before proceeding. As seen in Figure 5, the log transformation succeeded in normalizing the inspection error data. Furthermore, the initial relationship between mean errors detected and fault/demerit density held true for the trans-

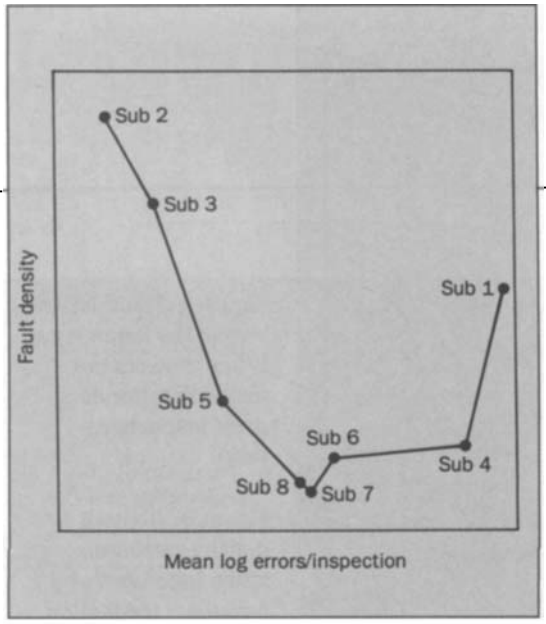

Figure 6. Fault density as a function of the mean logarithm of errors per inspection (for all inspections).

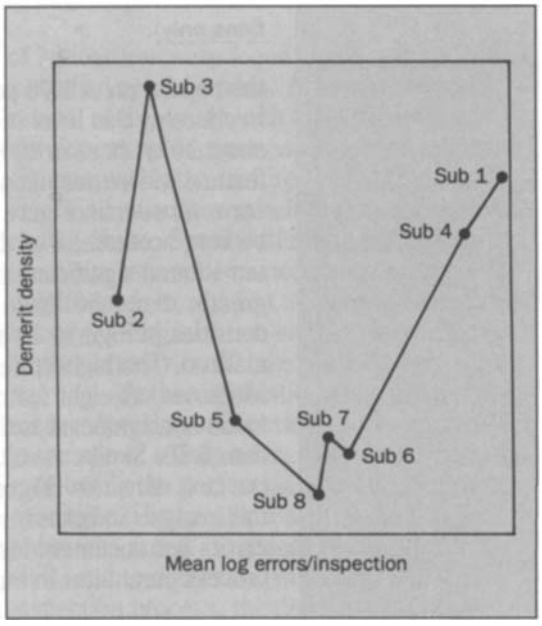

Figure 7. Demerit density versus the mean logarithm of errors per inspection (for all inspections).

formed data (Figures 6 and 7).

To this point the body of data consisted of both documentation and code inspections. However, since inferences about documentation inspections would provide indication for concerns earlier than code inspections, only these inspections were selected for further evaluation. In addition, the status of release Y development, which was at the end of its design phase, would allow for comparison. Initial results of the study were substantiated for the population of document inspections alone (Figures 8 and 9).

Previous graphs implied that release X features and subsystems with mean errors near the process mean had the lowest fault/demerit densities. To quantitatively determine which efforts had documentation mean errors different than the population, a test of hypothesis was required.[3] Since the population was bell-shaped and samples were small, a T test was appropriate:

$$T = \frac{\bar{x} - u}{o_s}\sqrt{n-1}$$

where $\bar{x}$ = sample mean
u = population mean
n = sample size
o_s = sample standard deviation

A T statistic was computed for each feature and subsystem and compared against a standard T score for a given level of confidence. A

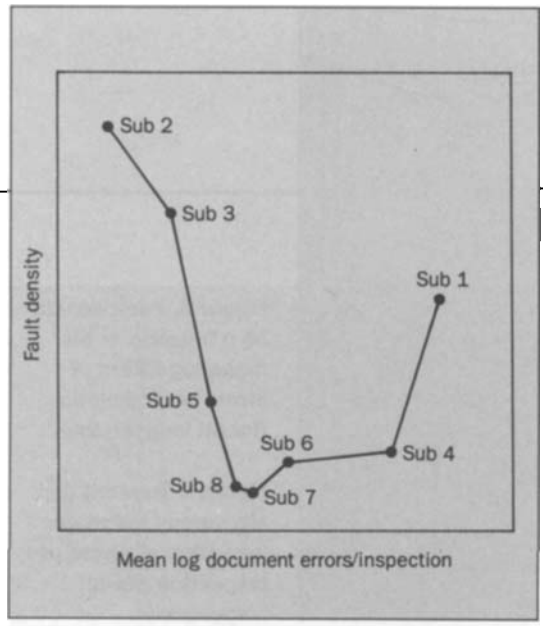

Figure 8. Fault density versus the mean logarithm of errors per inspection (for document inspections only).

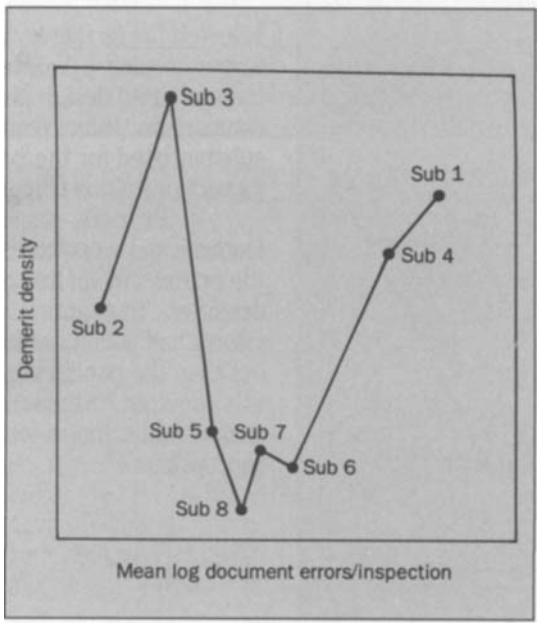

Figure 9. Demerit density versus the mean logarithm of errors per inspection (for document inspections only).

computed T greater than the standard T implies that the sample (feature or subsystem) mean is significantly different than the population mean.

When the T test was applied to release X, 9 of 17 features or subsystems were found to have means significantly different than the population at a 90 percent confidence level. In choosing this level of confidence, there are about 10 chances in 100 that a subsystem or feature will be mistakenly identified for concern. As seen in Figure 10, the test proved to be very accurate: 8 of the 9 features or subsystems found significant had demerit densities greater than 0.50. Note that the fault/demerit densities in Figures 10 and 11 have been normalized. The highest density appears as 1.00. Moreover, all eight features or subsystems found nonsignificant had demerit densities less than 0.50. Similar results were observed for the fault densities (Figure 11). The results of this analysis indicated that the number of errors per document inspection was a critical process parameter in our development process.

Application

Because the T test was successful in explaining differences in end-product quality for release X, it was applied to release Y in order to identify features and subsystems for concern. This would allow appropriate corrective action to be initiated before release of the product. For release Y, 3 of 7 subsystems and 5

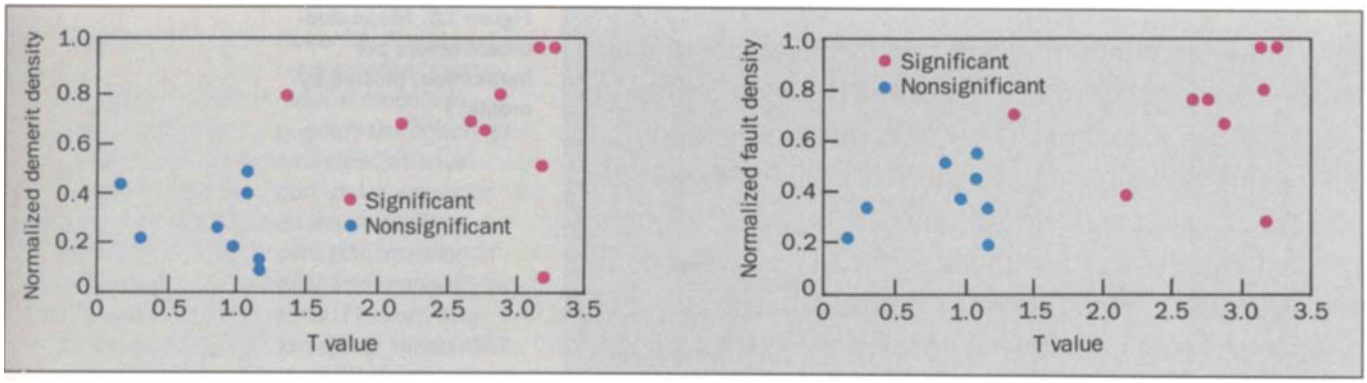

Figure 10. Normalized demerit density versus *T* value in the application of the *T* test to development efforts for release X.

Figure 11. Normalized fault density versus *T* value in the application of the *T* test to development efforts, release X.

of 18 features were found significant and thus identified as exceptions. After sharing the results of this study with management and developers we recommended reevaluation of documents for the features and subsystems identified as concerns before proceeding to the coding phase. Special attention in the form of more rigorous code inspections was also emphasized. In addition, results were explained to our test organization so that more effective testing activity could take place.

Another application of errors per document inspection as a critical process parameter is the use of a time series plot. As a result of monitoring the performance of this parameter over time (Figure 12), a shift in the process mean was noted. To assess reasons for this shift, and to determine weaknesses in the inspection process, the development organization implemented an in-process audit of requirements, design, and test plan inspections during the second quarter of 1985.

The audit was conducted by management-selected senior technical personnel who evaluated key attributes that impact inspection effectiveness: preparation, attendance, and participation, which included an evaluation of performance of inspection roles. Also, any significant errors not detected by the inspection teams were noted by the auditors.

The value of tracking errors per document inspection to identify shifts in the process mean was illustrated through the results of the audit. Several problems in the inspection process were identified and acknowledged by the development organization. Currently, recommendations submitted by the audit team are being pursued in order to improve the overall effectiveness of the inspection process.

Conclusion

The usefulness of inspection results in managing the software development process depends upon strict adherence to the Software Quality Assurance Plan. Inspections should not be held only when convenient, at the discretion of the developer, or waived in a schedule crunch. The strict adherence to the Software Quality Assurance Plan in our project not only provided data for relation of inspection results to end-product quality, but also yielded significant reductions in first-time development costs, long-term maintenance costs, and product errors found after development. There was

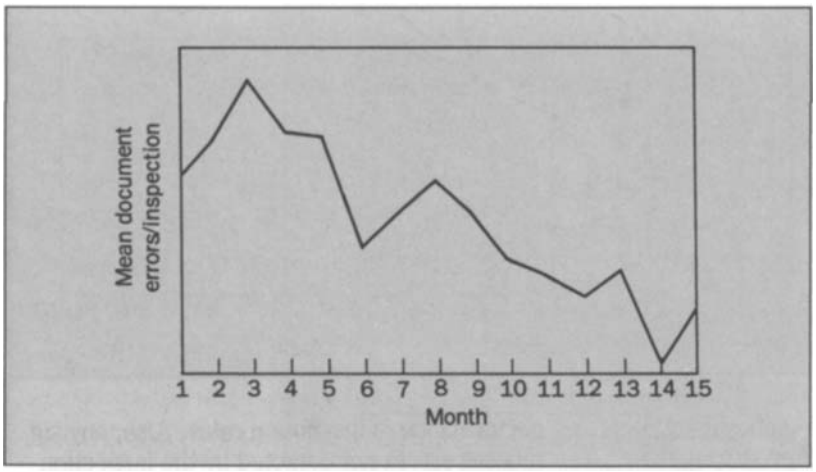

Figure 12. Mean document errors per inspection, plotted by month.

also significant improvement in the development organization's ability to meet internal schedules.

Given consistent application of the Software Quality Assurance Plan, the following conclusions and recommendations can be offered:

1. Results of document inspections can be used to explain differences in end-product quality.
2. Classic statistical techniques have application in monitoring and controlling the software development process.
3. Number of errors per document inspection is a critical process parameter.
4. Since the inspection process exhibits stability over time, a shift in errors per document inspection indicates a change in process.
5. A shift in errors per document inspection should be investigated for root cause.
6. Errors per document inspection can be an early indication of areas for concern on specific software development efforts.

Further investigation is required to validate the results of this analysis on subsequent product releases. Additionally, data from other projects need to be gathered and analyzed in a similar manner. Should these further studies confirm the results of this analysis, a significant step will have been made in transforming the "art" of software development into a science.

References

1. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, No. 3, 1976.
2. D. P. Freedman and G. M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, third edition, Little, Brown and Co., Boston, 1982.
3. Frank M. Gryna, Jr., and J. M. Juran, *Quality Planning and Analysis*, McGraw-Hill Book Company, New York, 1980.

(Manuscript received December 4, 1985)