IN-PROCESS INSPECTIONS OF WORKPRODUCTS AT AT&T

Priscilla J. Fowler

Priscilla J. Fowler *is a supervisor in the Technology Training Department at AT&T Bell Laboratories in Piscataway, New Jersey. She is responsible for the System Development Technology group, which provides consulting services and training to software development and hardware design projects within AT&T. Ms. Fowler, who joined AT&T in 1970, holds a B.A. from Drew University.*

In-process inspections are examination meetings held to find defects in design and development workproducts, including intermediate versions of the product or system in requirements and design documents. Because these inspections delimit the phases of design and development processes, they can prevent the passage of defects from one phase to the next and significantly reduce the number of defects released to customers. Software development projects within AT&T's research and development community have been using in-process inspections effectively for several years to reduce defects, and hardware projects began using them one and a half years ago. In addition, the experience of installing in-process inspections in project organizations has yielded a wealth of information on technology transfer. This article defines inspections, describes the installation process, and discusses some uses for inspection data.

Effective Mechanisms

Since the early 1970s and the beginning of large software development projects, various types of peer review meetings have been used to resolve problems and find defects. Early forms of peer review, best exemplified by the *structured walkthrough*,[1] addressed code quality. This focus on code is consistent, because software engineering, as a new field, was concerned almost exclusively with techniques for building and debugging its end product, the code.

However, practices have evolved, and software development organizations of any size now take for granted that there must be a software development "methodology."[2] (The methodology is a handbook of policies and methods for developing software in a particular design or development organization. It defines the project's phases and intermediate deliverables, or workproducts.)

Organizations no longer question the need to validate and verify interim versions of the product being developed. They assume that they will check requirements, design specifications, code, test plans, and other workproducts in some way[3-6] to prevent the propagation of defects from one workproduct to another, especially to one later in the development process. Such checking will be necessary as long as software development continues to be a human-intensive[7] activity.

Thus, peer reviews are now commonplace[2,8-9] throughout the development process at AT&T Bell Laboratories and elsewhere. But all reviews are not equally effective in detecting defects. With the current push toward greater quality and productivity, it is imperative that we use the most effective mechanisms possible to develop products.

The software inspection process, which was developed and documented ten years ago at IBM, has been in use at AT&T since 1981. It is an extremely effective peer review process that can filter out critical defects and improve productivity through its in-process data collection mechanism.

Throughout this article, we will draw primarily on the experiences of software development projects in the AT&T research and development community and of the Bell Laboratories Software Engineering Technology Transfer (SETT) program in installing inspections.

Software Inspections

Software inspections or, more correctly, software development workproduct inspections are meetings where development workproducts—such as design specifications, test plans and code—are "read." That is, the workproduct is examined meticulously and systematically by its author and his or her peers.

The examination tries to find defects, mismatches between the workproduct and the specification or between the workproduct and the standards. The sharp focus on detecting defects is the key to the effectiveness of in-process inspections.

Inspection meetings are not problem-solving sessions. If open issues still exist for the workproduct, then inspection is premature and a review meeting to consider alternative solutions is more appropriate. In any case, it does not pay to detect defects in a workproduct that may not yet be stable.

Inspectors collect and then analyze data about defects to tune the inspection process, anticipate the need for special attention to particular workproducts downstream in development, or correct current problems in the development phase that generated the workproduct being examined.

In addition, the meetings have positive side-effects; they disseminate product information and development experience, and enhance team spirit.

Prerequisites for Using Inspections. The inspection process assumes well-defined workproducts. In this context, "well-defined" means that a standard exists or, at least, there is a good example that can serve as a standard for the workproduct being inspected. This standard becomes part of entry criteria, the criteria that a workproduct must meet before it is considered ready to inspect.

The standard identifies the form of the workproduct. For example, a software design specification must contain a design unit overview, shared data structures, a list of lower level design units, integration guidelines, and areas of potential change. All these items must be present in the design specification, otherwise the workproduct is not ready to be inspected. Also, the specification must have been checked (by the UNIX® system's `spell` command), its lines must be numbered, and associated documents must be available for use during inspection.

But inspecting requirements usually is quite different from inspecting the design specification. Although the requirements document's format can be standardized, there is no preceding specification to check the requirements against. Therefore, requirements inspections rely heavily on the inspectors' expertise and knowledge, and often require more participants than other inspection types. The best moderators must be used for requirements inspections, because the inspectors are checking requirements against undocumented information sources.

The group of peer developers who attend the inspection meeting prepare for their role as inspectors by studying the workproduct to be examined and reviewing the associated material.

The Inspection Meeting. One developer moderates the meeting, assures at its start that all participants are ready to inspect, and keeps the meeting on target and objective. Another developer acts as reader, paraphrasing the material in the workproduct and pacing the inspection process, while a third developer records the defects discovered. (In a small group, the moderator may take this as a second function.)

The moderator, the reader, and the workproduct's author all act as inspectors. The author is often the most effective inspector, because he or she combines perspective—gained from hearing the reader present the workproduct—with intimate knowledge of the workproduct's function and structure. The author does not read or moderate.

Besides assuming the role of moderator, reader, or recorder, each meeting participant takes a particular point of view when examining the workproduct. This point-of-view role follows the current thinking on quality: That everybody, even someone internal to a design or development process, is a customer and has customers.

Because different workproducts have different customers, an inspection participant may be a customer of a workproduct's author. For example, a system tester may inspect a requirements document on which he or she must base a test plan. Or, the author may be a programmer who developed code based on a designer's workproduct and, thus, is a customer of the designer who participates to assure accurate use of his or her design.

Tables I and II identify customer types, and therefore inspection participants, for software and hardware workproducts.

The success of the inspection meeting depends directly on how the moderator conducts it. Besides being skilled in running meetings, the moderator must be someone who is seen as both a leader and a strong technical contributor.

The combined efforts of three, four, or five people who concentrate on one workproduct in this format result in more defects detected than the sum of their individual efforts.

Checklists of common defects for the type of workproduct being inspected guide and enhance the meeting's defect detection process. Also, management should not attend the meeting, because this could discourage participants from energetically looking for and disclosing defects.

Classifying Detected Defects. Once the workproduct has been completely inspected and all defects have been recorded, they must be classified, usually in three ways:

- *Class*—The basic defect classes are: missing, wrong, and extra.
- *Severity*—Defects are categorized by severity, usually on two levels: the more severe defects would prevent customers from using all or an important part of the system or product; and the less severe would not cause such limitation.
- *Type*—It is helpful to categorize defects in terms of type, such as a logic defect in code, an interface defect in design, or a missing menu screen in a requirements document (human-interface defect).

As part of the start-up process[10] to using inspections, a defect-classification scheme must be defined (and later maintained) for all inspection types in a development organization. Certainly, the inspection meeting is not an appropriate forum for discussing what scheme to use. However, once the classification scheme is established, the on-going analysis of inspection data is reasonably straightforward and can yield results that are useful for process control during software development.

Followup. After the inspection meeting, as a final step in the inspection process, the author corrects all defects in the workproduct, which is then either reinspected or verified by the moderator. Defects discovered

Table I. Software Development Workproduct Inspections

Inspection Type	Customer Type (Participant)						
	Requirements Author	Architect	Detail Designer	Verifier (Tester)	User Documentation Author	Maintainer	Coder
High-level design	X	X	X	X	X	X	—
Detail design	—	X	X*	X	—	X	X
Code	—	—	X	X	—	X	X*

*Including author.

Table II. Hardware Design Workproduct Inspections

Inspection Type	Customer Type (Participant)							
	Requirements Author	Architect	Circuit Designer	Physical Designer	Design Engineering Representative	Test Engineer	Power Engineer	Manufacturing Engineer
Functional architecture	X	X	X	—	X	—	—	—
Electrical design	—	—	X*	X	—	—	—	—
Electrical implementation	—	—	X	X	—	X	X	X

*Several, including author.

in associated workproducts as a result of an inspection are documented and tracked, for example, through modification requests (MRs). Once this work has been done, the workproduct has passed the exit criteria for that inspection type.

Meeting Milestones. On many projects, passing an inspection is the final step in completing the workproduct's development for that phase and gives real meaning to the term milestone. Robertson and Secor[2] suggest frequent demonstrations of a software product under development as a way to verify that milestones have been completed. However, demonstrations depend on code. Inspections can verify milestone completion early in the development process when no code is available.

In-process Data

A major distinction between software inspections and other peer review techniques is the data collection and analysis process. Besides numbers of defects and their classes, severity, and type, the data collected include inspection type (design, code, etc.), date of the meeting, length of the meeting, number of inspectors, total preparation time, and both estimated and real rework time for correcting defects.

Uses of Data. These data are used in various ways that are highly compatible with the phased development and incremental release approach typical of software development in AT&T's research and development community.

First, data from a few inspections on volume of material inspected, staff time used, and defects found can quickly show the usefulness of the process in a particular environment.

Second, inspections often unearth defects that many concede would be extremely costly to find with system tests. Even apparently minor defects that inspections found may be difficult to find during testing. An inspection finds the defect, while testing—which usually occurs one or more development phases after the opportunity to inspect has passed—finds only the symptom.

Finally, data from several inspections can show where trouble may be brewing in either the development process or the product. If many defects occur in interface design, for example, there may be a process problem. Perhaps the interface section of the design document standard is not clearly specified, or perhaps a standard has not been written. If defects in design suggest an unusually difficult technical issue, system testers can prepare to test the affected software with special care.

Studies of Inspection Results

Fagan's original article on inspections[11] includes the only controlled study on the effectiveness of software inspections. In that study, design and code inspections found 82 percent of the defects uncovered; 18 percent were found in later tests. Coding productivity increased 23 percent, because less rework was needed to correct defects.

Several Bell Laboratories studies, published as talks or memoranda, document the effectiveness of inspections. In one study, a major software development organization (more than 200 members of technical staff) increased its productivity by 14 percent from one release to the next by using improved project phasing and tracking mechanisms, including inspections and reviews. Early software-fault-density data showed a tenfold improvement in quality. (Unfortunately, later fault-density data were not available, so further comparisons were not possible.)

Although this study is not definitive—we cannot attribute the results solely to inspections—it is consistent with anecdotal reports from the project's technical staff and management. They credit inspections with an important influence on both quality and productivity.

Another study examined the use of inspections on the 5ESS™ switch project. The conclusion: reviews and inspections are cheap and effective mechanisms for early defect detection. Defects detected in inspections cost an average of ten times less to fix than defects found during development but outside inspections and reviews. The study also concluded that inspections and reviews could be even more effective if inspectors were better prepared.

Graden and Horsley[9] studied inspection results from the 1/1A ESS™ switch project. Both document inspections (of requirements and design documents and test plans) and code inspections were used. Document inspections that produced defect counts different from the mean pointed out, prior to code inspection, areas for quality concern. The study also found that defects detected in document inspections correlated highly with MR and demerit performance. (*Demerit* is a severe MR.)

Eisele and Rathburn[8] measured various aspects of the testing process for generic 1 of the network control point's direct services dialing application. They determined that "it is about twenty times more effective to find bugs in tests during an inspection than to find them during lab sessions" of testing. Inspections of test documents cost two percent of the total testing effort for that generic.

Designing the Inspections Program

In-process inspections are widely and effectively used throughout AT&T's research and development community. They have made a major contribution to the productivity and quality of software projects. As such, they are significant and interesting.

However, the way the inspection technique and its attendant process were disseminated is also interesting. It did not consist solely of classroom training, but required considerable involvement with and understanding of projects that adopt in-process inspections. Therefore, it yielded significant information about the technology transfer process for software engineering. It is worth considering what has been learned from inspections in this area as well.

Early History. By mid-1981, the standards development effort[12] of the IEEE (Institute of Electrical and Electronics Engineers) Computer Society had produced the IEEE Software Quality Assurance Plan (SQAP). The SQAP standard[13] presented the requirements for a software development methodology. It provided a basis for organizations to write methodologies that are appropriate to their environments and technical problems.

The Bell Laboratories Quality Assurance Center had recognized that software quality issues needed to be worked in-process (i.e., within the development process). Therefore, it devoted a small staff to publicizing the SQAP and establishing its use within Bell Laboratories.

In addition, the Quality Assurance Center's staff had determined that fault density measures, patterned after software reliability modeling,[14] could indicate the quality of one release of a system over another, and was defining a mechanism for comparable in-process measures. At that time, the Bell Laboratories Systems Training Center was also preparing to teach software inspections as a peer review technique.

The Quality Assurance Center and Systems Training Center efforts were joined, and software inspections gradually became the in-process check mechanism of choice in Bell Laboratories.[6]

But choosing the mechanism and preparing a course on inspections were just the beginning. During early efforts to teach software developers the inspection technique, it soon became clear that, while the basic technique was straightforward, getting developers to use it routinely and systematically was not.

Addressed Organization Needs. Pilot sessions of the Software Inspections Workshop produced comments from developers: Was the "overhead" of inspections justifiable? Would their management support the use of inspections? Was the data collected worth the effort? Would the data be used inappropriately, for example, as part of an individual's performance review process?

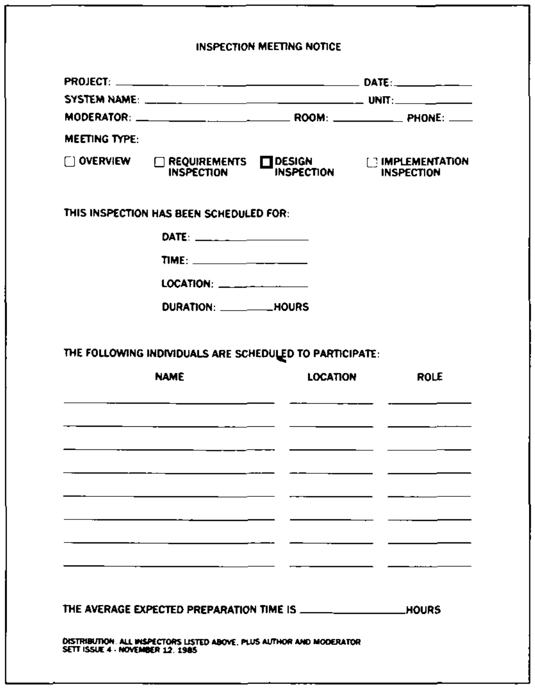
Supervisors had similar questions. In addition, they questioned how inspections could be used when specification documents were informal and incomplete, if not nonexistent, and schedules were tight and seemingly inflexible.

Clearly, there were issues to address far beyond teaching the skills that developers needed to participate in inspection meetings. These issues cut across the levels of personnel and technical responsibilities of a software development organization.

As the issues were analyzed, solutions were proposed and tested, and the Software Inspections Program was created. (Later, it became the Software Inspections Unit of the SETT program.)

Organizational Learning. Early on, the Software Inspections Program recognized and addressed many technology transfer issues articulated by the program's staff and others at the IEEE's Computer Society Workshop on Software Engineering Technology Transfer in 1983.[15] Issues revolved around the need to deal with a software development organization, not just in terms of its workers, but also in terms of its culture, management, software environment, budget, and quality and productivity goals.

Thus, key to the Software Inspections Program's design was to recognize that, rather than giving individuals a set of skills, the program was installing a process in an organization. Training for individuals made sense only after managers could be counted on to support the new process, and the new process was carefully designed to serve in the organization's environment and culture.



INSPECTION MEETING NOTICE

PROJECT: DATE:
SYSTEM NAME: UNIT:
MODERATOR: ROOM: PHONE:

MEETING TYPE:

☐ OVERVIEW ☐ REQUIREMENTS INSPECTION ☐ DESIGN INSPECTION ☐ IMPLEMENTATION INSPECTION

THIS INSPECTION HAS BEEN SCHEDULED FOR:

DATE:
TIME:
LOCATION:
DURATION: HOURS

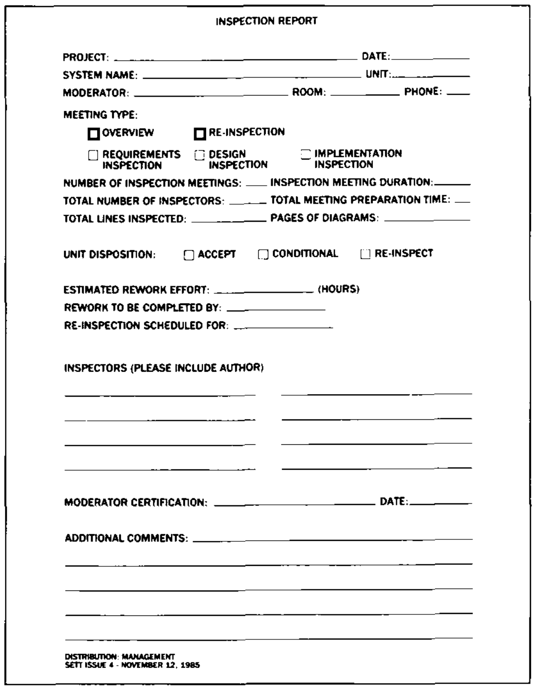
THE FOLLOWING INDIVIDUALS ARE SCHEDULED TO PARTICIPATE:

NAME	LOCATION	ROLE

THE AVERAGE EXPECTED PREPARATION TIME IS HOURS

DISTRIBUTION: ALL INSPECTORS LISTED ABOVE, PLUS AUTHOR AND MODERATOR
SETT ISSUE 4 - NOVEMBER 12, 1985

Figure 1. Typical form for Inspection Meeting Notice.



INSPECTION REPORT

PROJECT: DATE:
SYSTEM NAME: UNIT:
MODERATOR: ROOM: PHONE:

MEETING TYPE:

☐ OVERVIEW ☐ RE-INSPECTION
☐ REQUIREMENTS INSPECTION ☐ DESIGN INSPECTION ☐ IMPLEMENTATION INSPECTION

NUMBER OF INSPECTION MEETINGS: INSPECTION MEETING DURATION:
TOTAL NUMBER OF INSPECTORS: TOTAL MEETING PREPARATION TIME:
TOTAL LINES INSPECTED: PAGES OF DIAGRAMS:

UNIT DISPOSITION: ☐ ACCEPT ☐ CONDITIONAL ☐ RE-INSPECT

ESTIMATED REWORK EFFORT: (HOURS)
REWORK TO BE COMPLETED BY:
RE-INSPECTION SCHEDULED FOR:

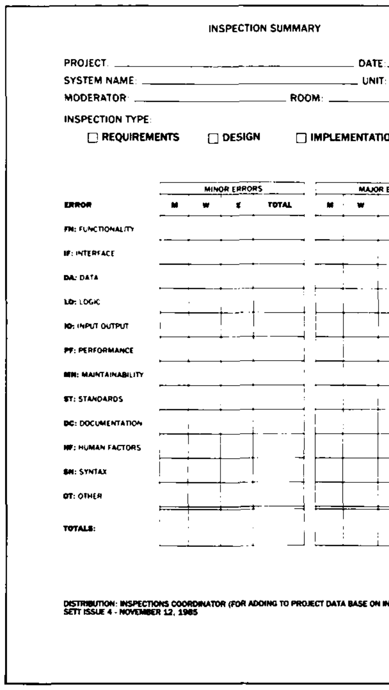
INSPECTORS (PLEASE INCLUDE AUTHOR)

MODERATOR CERTIFICATION: DATE:

ADDITIONAL COMMENTS:

DISTRIBUTION: MANAGEMENT
SETT ISSUE 4 - NOVEMBER 12, 1985

Figure 2. Typical form for Inspection Report.



INSPECTION SUMMARY

PROJECT DATE:
SYSTEM NAME UNIT:
MODERATOR ROOM:

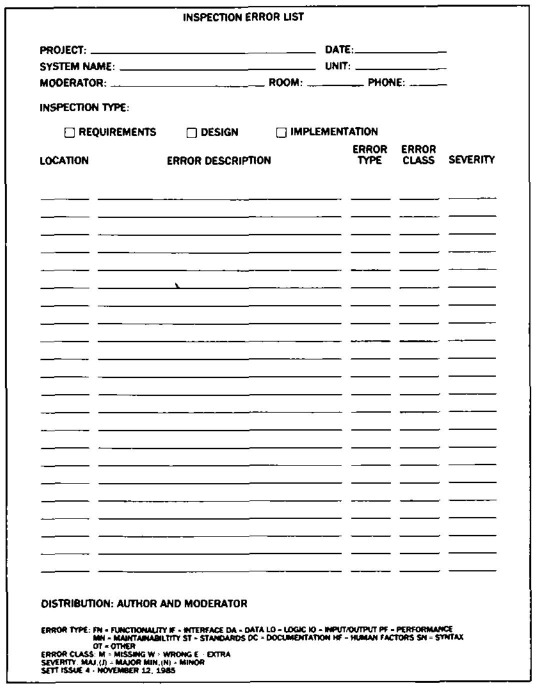
INSPECTION TYPE:

☐ REQUIREMENTS ☐ DESIGN ☐ IMPLEMENTATIO

	MINOR ERRORS				MAJOR E	
ERROR	M	W	E	TOTAL	M	W
FN: FUNCTIONALITY						
IF: INTERFACE						
DA: DATA						
LO: LOGIC						
IO: INPUT OUTPUT						
PF: PERFORMANCE						
MN: MAINTAINABILITY						
ST: STANDARDS						
DC: DOCUMENTATION						
HF: HUMAN FACTORS						
SN: SYNTAX						
OT: OTHER						
TOTALS:						

DISTRIBUTION: INSPECTIONS COORDINATOR (FOR ADDING TO PROJECT DATA BASE ON IN
SETT ISSUE 4 - NOVEMBER 12, 1985

Figure 3. Typical form for Inspection Summary.



INSPECTION ERROR LIST

PROJECT: DATE:
SYSTEM NAME: UNIT:
MODERATOR: ROOM: PHONE:

INSPECTION TYPE:

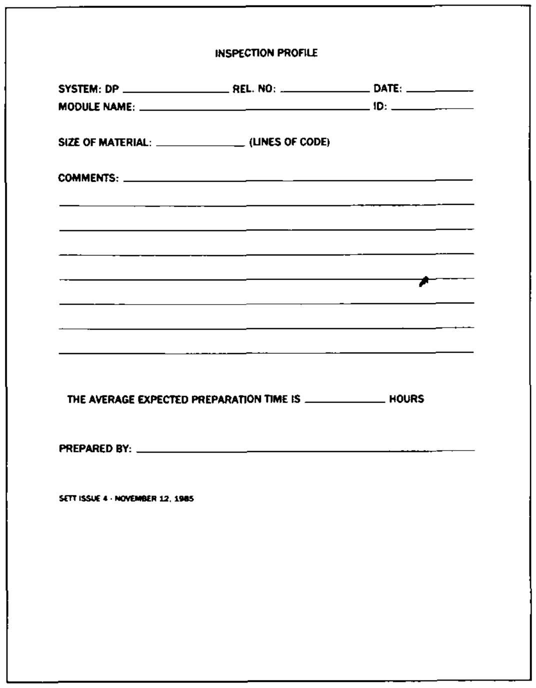
☐ REQUIREMENTS ☐ DESIGN ☐ IMPLEMENTATION

LOCATION	ERROR DESCRIPTION	ERROR TYPE	ERROR CLASS	SEVERITY

DISTRIBUTION: AUTHOR AND MODERATOR

ERROR TYPE: FN - FUNCTIONALITY IF - INTERFACE DA - DATA LO - LOGIC IO - INPUT/OUTPUT PF - PERFORMANCE
MN - MAINTAINABILTITY ST - STANDARDS DC - DOCUMENTATION HF - HUMAN FACTORS SN = SYNTAX
OT = OTHER
ERROR CLASS: M = MISSING W - WRONG E - EXTRA
SEVERITY: MAJ.(J) = MAJOR MIN.(N) - MINOR
SETT ISSUE 4 - NOVEMBER 12, 1985

Figure 4. Typical form for Inspection Error List.



INSPECTION PROFILE

SYSTEM: DP REL. NO: DATE:
MODULE NAME: ID:

SIZE OF MATERIAL: (LINES OF CODE)

COMMENTS:

THE AVERAGE EXPECTED PREPARATION TIME IS HOURS

PREPARED BY:

SETT ISSUE 4 - NOVEMBER 12, 1985

Figure 5. Typical form for Inspection Profile.

The Software Inspections Program

The program consists of five steps: Overview Seminar, Needs Assessment and Planning Meeting, Inspection Workshop, Management Seminar, and Review and Evaluation.

The *Overview Seminar* provides a general understanding of the inspection process. How is it carried out? What are the expected benefits? What are the costs? What project and other resources are required to implement software inspections in an organization? Managers and key technical personnel from the same project attend.

The format is presentation, with ample time for and encouragement to ask questions. At the conclusion of this seminar, the project has enough information to decide whether to attempt a trial use of inspections. Equally important, the attendees have had the opportunity to "buy in" and, thus, are far more likely to understand and support the use of inspections.

The *Needs Assessment and Planning Meeting* determines the best approach for inspections in that organization, and begins specific implementation activities. Here, attendees:

- Discuss the organization's software development methodology, including development phase demarcations and the related workproducts.
- Select project people for further inspection planning and coordination.
- Determine the inspection types to be used, related entry and exit criteria, and checklists; adapt forms (e.g., Figures 1 through 5) for inspection data collection procedures; and determine administrative procedures to support inspections, such as moderator selection, copying and distributing workproducts to inspectors, room reservations, etc. They also prepare a project inspections manual that includes all this. (The Software Inspection Program's staff maintains a generic manual for use as a base to save time in preparing a project-specific manual.)
- Determine qualitative and quantitative goals for the use of inspections.
- Determine the inspection implementation and training schedule.
- Determine trial groups that will use inspections (if inspections will not be used project-wide).
- Review the Inspection Workshop's first module to determine what revisions are needed for the particular project. The first module deals with inspections as a software quality technique. It works best when the vocabulary and concepts are specialized to the organization's environment, because there is not yet a universal vocabulary for software development.
- Select an Inspection Workshop case study, a practice inspection meeting of the type that the project will use first.

The *Inspection Workshop*, a one-day course to train inspection moderators and participants, is held close to the time in a project's schedule when inspections are needed (for example, at the start of a design or code phase). The workshop focuses on the steps needed to prepare for, participate in, and conduct an inspection meeting, and includes the practice meeting (case study). It concludes with a discussion on implementing inspections in the organization, along with distribution of the project's inspection manual.

The results of the needs assessment study done earlier are used to orient the workshop and select a case study problem that matches the organization requirements and inspection implementation plan. In addition, the knowledge that management supports inspections and that the use of inspections is imminent enhances the effectiveness of this training. Students rarely drop out of the course and begin using inspections immediately, because all organization and support issues are addressed before the course is held.

Once a project uses inspections routinely, new project members can learn the inspections process from a generic workshop and apply these skills successfully in their specific work assignments. Generic skills can be adapted quickly with the help of the inspection coordinator and by using the project inspection manual.

The optional half-day *Management Seminar* occurs after ten or more inspections have been held. Here, the presentations and discussions focus on how inspections affect the project's estimation and budget profiles, personnel requirements, staffing, and milestones. In addition, managing the inspections process is covered, including administration, inspection process quality, and inspector attitudes. Finally, the use of inspection data for process control (as early indicators of quality problems in the design or development process) is discussed in detail.

If this seminar is not held, the equivalent information is conveyed informally.

Review and Evaluation activities also occur after at least ten inspection meetings have been held, and whenever an organization perceives its inspection process is not behaving as expected. These activities distill the organization's experience with inspections into specific recommendations for tuning the use of inspections. They include an analysis of time spent in inspection preparation and meetings, defects detected (type and count), and rate of inspection.

Adapting the Program

It is not reasonable to expect one project person who just learned a technique or process from a course, book, or conference tutorial to also have the expertise, management, marketing, and training skills to establish the technique or process in his or her organization. Therefore, the Software Inspections Program deals directly with the substantial difference between learning a new technique or process and applying it successfully. It also deals with the disruption that inevitably accompanies change. The program's five steps allow a project to prepare systematically for the new activity, and disruption is minimal.

Besides recognizing and dealing with organizational issues of installing a new process, the Software Inspections Program recognizes and deals with technical issues. Not surprisingly, an environment as diverse as AT&T's software development community did not have—and, indeed, could not have agreed on—a universal, corporate-level, software development methodology.

But the use of inspections was critical to improving both personnel productivity and product quality. Somehow, the inspections process had to be adapted to work equally well in as many development contexts as needed.

Inspection Parameters. After studying Fagan's article[11] and reviewing our consulting experience on some projects that use inspections, we concluded that the inspections process could be used anywhere there were reasonably well-defined workproducts.

For each inspection type (i.e., requirements, design, code, test plans, etc.), inspections could be specified by considering entry and exit criteria, checklists of common defects, a defect classification scheme, and participants (Tables I and II). Further, these specifications could, and should, be refined using inspections data and other feedback from the inspectors.

In this way, preparing the organization to use inspections dovetails with adapting the inspections process to local technical issues. Also, with a parameter-driven adaptation process, inspections could be installed in any organization that is willing to take a few staff days to specify one or more inspection types.

Ideally, one corporate-level or several area-wide software development methodologies would have been defined, with consistent inspections training. If this were the case, AT&T would not have needed a highly adaptive process. In reality, developing such a methodology would have taken years, and the need for immediate improvement was urgent. By proceeding in a more flexible, even "grass roots" way, the practice of inspections has become commonplace.

Consultative Training. The project-oriented approach of the Software Inspections Program is used, with modifications, to disseminate other software engineering techniques via the SETT program. This approach, called *consultative training,* builds on an assessment of a project's or organization's needs, which is used to tailor the generic formal courses and performance aids to a particular environment. The training also addresses all levels of personnel in an organization.

Like inspections, consultative training adapts to many different situations, but works particularly well in design and development environments. Consultative training and inspections are well-suited to processes, such as software development or the early stages of hardware design, that depend highly on people.

Other Applications of Inspections

With minor revisions, the inspections process can be used in any design or development environment where the design or development process is well-understood, if not documented. Inspections are now being used successfully in AT&T Bell Laboratories on several hardware design projects, including the D5 channel bank and the Metrobus and FT Series G lightwave systems.[16] They are also part of course design and development for technical training materials.

Acknowledgments

The success of in-process inspections over the past several years is due in large part to the efforts of many people working on design and development projects. They understood the importance of engineering the design and development process to achieve improvements in quality and productivity. More significantly, their extra effort facilitated the use of inspections and related techniques.

First are the people—too many to name individually—who wrote design and development methodologies on projects such as the 5ESS switch, UNIX Real-Time Reliable (RTR) operating system, network operations systems, Metrobus, and D5. These methodologies provided the context for the successful use of inspections and process-oriented design and development techniques. Second are the local champions of inspections, including W. W. Kremer, E. H. Black, C. H. Kolbenson, B. Garg, R. H. Yacobellis, S. E. Schwab, A. Sheng, R. A. Maione, S. M. Poet, T. Pingel, R. S. Cooper, N. M. Scribner, and J. M. Kalmanek. Third is the management that was not only supportive, but assertive, in encouraging the use of inspections in their organizations, including D. C. Opferman, H. L. Bosco, C. L. Pettijohn, J. J. Lang, E. M. Prell, and R. J. Sanferrare.

Next are early advocates for inspections from the Quality Assurance Center: W. K. Comella, V. S. Hiering, and B. F. Gundaker. Also important and very influential are the people who measured and quantified the effects of inspections: W. Kremer, T. Pingel, D. A. Christensen, W. Ku, R. C. Eisele, and W. K. Wiener-Ehrlich. Finally, there is E. E. Sumner, who had the idea to use inspections in the first place, and started it all.

References

1. E. Yourdon, *Structured Walkthroughs*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
2. L. Robertson and G. Secor, "Effective Management of Software Development," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 94-101.
3. G. J. Surette, "The AT&T Quality System," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 21-29.
4. C. L. Pettijohn, "Achieving Quality in the Development Process," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 85-93.
5. J. Inglis, "Standard Software Quality Metrics," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 113-118.
6. E. M. Prell and A. P. Sheng, "Building in Quality and Productivity to a Large Software System," IEEE *Software*, July 1984, pp. 47-54.
7. R. Balzer, "A 15 Year Perspective on Automatic Programming," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 11, 1985, pp. 1257-1267.
8. R. C. Eisele and D. L. Rathburn, "What We Learned from Functional Testing of the DSD1 Generic," *Proceedings AT&T Software Quality Symposium: Achieving Productivity Through Quality*, December 11-12, 1985, pp. 1-6.
9. M. E. Graden and P. S. Horsley, "Effects of Inspections on 1/1A ESS™ Software Quality," *Proceedings AT&T Software Quality Symposium: Achieving Productivity Through Quality*, December 11-12, 1985, pp. 36-51.
10. A. F. Ackerman, P. J. Fowler, and R. G. Ebenau, "Software Inspections and the Industrial Production of Software," *Proceedings Symposium on Software Validation*, 1983, pp. 13-40.
11. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.
12. A. F. Ackerman and F. J. Buckley, "Software Standards Take Shape," *Datamation*, October 1983, pp. 259-262.

13. *IEEE Standard for Software Quality Assurance Plans,* ANSI/IEEE Standard No. 730-184, IEEE, Inc., 1984.
14. J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application,* McGraw-Hill, New York (scheduled for publication in 1986).
15. T. J. Emerson, A. F. Ackerman, A.S. Ackerman, P. J. Fowler, R. G. Ebenau, and S. A. Rosenthal, "Training for Software Engineering Technology Transfer," *Proceedings IEEE Computer Society Workshop on Software Engineering Technology Transfer,* 1983, pp. 34-41.
16. R. M. Kanter, *The Change Masters,* Simon & Schuster, New York, 1983.

(Manuscript received November 12, 1985)