Installation Management

Includes: GN20-3814

Inspections in Application Development – Introduction and Implementation Guidelines

IBM

GC20-2000-0

Inspections in Application Development – Introduction and Implementation Guidelines

Application development inspections is a disciplined technical project review technique used at the end of various application development phases to improve program quality and project manageability and to increase productivity. This manual is designed to provide enough information to the manager of the data processing activity and his technical staff to answer the question: "Should this activity use inspections?" The manual also provides guidance in preparing for and performing inspections. A case study is included.

This description of the inspections technique is made available by IBM with the objective of providing information that may assist others in improving their own application development procedures. It does not require the use of IBM products or services. However, IBM DP Services will, subject to the local availability of appropriate resources, respond to requests for assistance in implementing inspections.

Preface

Application development inspections is a technical project review technique held at the end of various application development phases.

This manual is designed to provide enough information to the manager of the data processing activity and his technical staff to answer the question: "Should this activity use inspections?" The manual also provides guidance for the moderator – the person who schedules inspections, invites participants, and controls inspection meetings and their follow-up – in preparing for and performing inspections. Appendixes provide a case study and solution that can be used in moderator training, offer sample materials that can be modified for use during inspections, and describe two types of inspections, test plan and test case, that installations may wish to adopt after gaining experience with the other types of inspections described in the text.

Although the inspections technique was developed within IBM in a system control programming environment one appendix reports on the use of inspections in an application development project within IBM. Each installation must evaluate the technique's usefulness for its own environment and, if adopted, tailor such elements as exit and reinspection criteria, checklists, and time estimates to fit the installation's characteristics.

The inspections technique is also discussed in *Design and Code Inspections to Reduce Errors in Program Development,* an article by M. E. Fagan that appeared in the IBM *Systems Journal,* Volume 15, Number 3, 1976. A reprint of this article is available from IBM under order number G321-5033.

First Edition (July 1977)

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

Address comments concerning the contents of this publication to IBM Corporation, Technical Publications, Dept. 824, 1133 Westchester Avenue, White Plains, N.Y. 10604.

Contents

List of Illustrations

Chapter 1. Introduction

Project reviews at the end of various application development phases have long been recognized as a vehicle for determining application development project status and identifying areas that need special attention. Such reviews are usually intended to accomplish the following:

1. Communicate project status to higher-level management and/or project personnel.
2. Establish that the project can be completed within the time and costs allotted, or adjust such schedules and costs.
3. Evaluate the technical accuracy of the project and arrange for the correction of technical errors uncovered as early as possible in the development cycle.

In practice, however, project reviews tend to omit this last function.

The Need for Technical Reviews

Technical reviews during which technical accuracy is evaluated and arrangements for necessary corrections made are needed at various points within the development cycle to help produce a product of higher quality that is easier to maintain while reducing the cost of finding and correcting errors and to help management to better control the development process (see Figure 1).

Higher quality and improved maintainability are central objectives of data processing management. The number of completed programs in use has been increasing; many of them still contain errors and frequently are difficult to modify; as a result, the number of personnel devoted to program maintenance has also been steadily increasing. It has been estimated that 40-60 percent of every programmer/analyst staff dollar is devoted to the maintenance of operational programs. As the size and complexity of new programs continue to rise, this is likely to increase further.

Achieving quality and maintainability goals while reducing the cost of finding and correcting errors dictates a strategy of *early* error detection. The cost of finding and correcting errors increases substantially as the development process continues. That is, the cost of detection and correction of errors during testing is dramatically higher then during design; the expense is higher still after the program has been placed into productive use.

Technical evaluation of product quality at the end of development phases can help improve management control by providing more realistic evaluations of the developing product's quality, cost, and schedule status, and can make possible timelier corrective action.

Structured Walk-Throughs and Inspections

Two types of technical reviews now in use are structured walk-throughs and application development inspections. The fundamental differences between the two are discussed in this text; the major concern, however, is inspections.

Structured Walk-Throughs

In a structured walk-through, a developer's work (program design, code, documentation, etc.) is reviewed by fellow project members invited by the developer. Structured walk-throughs, in various forms, are being used by many project development groups. The forms vary in their objectives and

Type of Review	Beneficiaries	Function/Objectives
Project status review	Management	● Communicate project status ● Check and adjust cost and schedules
Technical reviews	Management Developers	● Improve quality ● Improve maintainability ● Technical evaluation ● Reduce costs ● Improve management control

Figure 1. Technical reviews versus project status reviews

procedures, but most have the following characteristics:

1. They are arranged and scheduled by the developer of the work product being reviewed.
2. Management does not ordinarily attend the walk-through.
3. The developer selects the list of reviewers, but in most cases, management examines the list to ensure that developers of related products are invited. The walk-through is usually attended by four to six reviewers. Participants can include:
   - Designers of the system, to ensure compatibility and continuity of design
   - Individuals responsible for documenting the function being reviewed
   - Testers responsible for functional and system testing
   - Developers of other parts of the system
   - Developers of other systems that interface with the one being reviewed
4. The reviewers are given the materials four to six days before the walk-through and are expected to review them and come to the session with a list of questions.
5. A typical walk-through is scheduled to last for a specified period, not longer than two hours. If the materials have not been completely reviewed at the end of that period, or if a significantly large list of issues have been created, another walk-through is scheduled for the next convenient time.
6. One person – usually the author whose work is being examined, or perhaps a project team leader – is appointed or elected to guide the session. That person compiles an action list consisting of all errors, discrepancies, exposures, and inconsistencies uncovered during the walk-through.
7. All issues are resolved after the session. The walk-through provides problem detection, not problem resolution.

Inspections

Inspections provide a more formal and rigorous method of performing technical reviews at the end of development phases. Application development inspections, as used in IBM development groups, can be characterized as follows:

1. Inspections appear as separate scheduled activities in the project development plan, and the project schedule contains a time allowance for rework of deficiencies identified in the inspection process.
2. Each development phase, at the end of which work products are to be inspected, is defined, as are the exit criteria for each.
3. At the end of an inspection, formal approval is required that deficiencies have been satisfactorily reworked and exit criteria have been satisfied before work can proceed to the next development phase.
4. A specially trained moderator schedules and conducts the inspections and chooses the participants, who are usually selected on the basis of special skills or knowledge. The moderator is not the developer of the product being inspected nor a member of the development team; he does not usually devote full time to this role, but is a working analyst or programmer. A moderator's work may be inspected in turn by inspectors from other projects.
5. The inspection of a work product at the end of a development phase consists of six well-defined steps, as shown in Figure 2.
6. The inspections technique emphasizes the accumulation and analysis of data about the types of errors and their frequency of occurrence. As the data base of error patterns grows larger and the moderators gain experience with error detection, the moderators can better analyze the results of inspections, can better help designers and implementers avoid errors, and can help inspection teams learn to do a more thorough job of error detection. Checklists developed from the data base assist in this by assuring that all reasonable questions have been considered during an inspection. Also, by comparing current inspection results against the data base, both developers and management can become aware of situations in which corrective action must be taken to avoid schedule delays or to reduce the liklihood of creating excessively error-prone modules. Accumulation and analysis of error patterns can also highlight for management those development practices in need of revision or suggest some that could be initiated, thus leading to an improved development process.

   In an installation that requires several moderators, management chooses one to serve as an inspection coordinator to oversee the implementation of inspections for the installation. The coordinator controls the establishment and modification of the inspection procedures. He normally serves as the first moderator; trains subsequent moderators; standardizes record-keeping procedures, checklists, and exit and

Step	Inspection Team Participants	Objectives
1. Planning	Moderator	Scheduling/distributing materials
2. Overview	Designer and other participants	Education
3. Preparation	Participants – individually	Self-study
4. Inspection meeting	Entire group	Finding errors
5. Rework	Author	Correcting errors
6. Follow-up	Moderator, author	Assuring correct rework, improving development and inspections

Figure 2. The six steps of an inspection (summarized)

reinspection criteria; maintains the installation's inspections data base; and, by analyzing this information, provides such data as time estimates for the various inspection steps, expected error rates, and analyses of errors by type. The coordinator's analyses of the data also assist the moderators in maintaining a high level of effectiveness and efficiency during inspections.

7. An application development process that includes inspections can be compared to an industrial continuous flow process in that each provides the environment for better management control of the process because management (1) defines the operations in the process, (2) defines the criteria for completion of each operation, and (3) measures the process by collecting data about the functioning of the process.

Key Differences Between Inspections and Structured Walk-Throughs

Key differences exist although there are variations of both techniques. Because of the variations, one or more of these differences may not be applicable in some installations.

The differences may be characterized as follows:

- Typically, walk-throughs are characterized by informality, with the developer (or chief programmer) requesting them. They may be performed on completed activities or during development of an activity. Although proper follow-up usually ensues, no formal approval is generally required before proceeding with further development work as is the case with inspections, although a second walk-through is usually performed when a "large" number of errors has been detected. In the inspections discipline, "exit criteria" for each development phase must be satisfied, errors corrected, and formal approval given before the work can proceed to the next phase.
- The moderator of the inspection is not part of the team that developed the work product. He is trained in the skills required for the moderator's role: planning the inspections, selecting participants, preparing for the inspection meeting, maintaining an efficient pace during the inspection meeting, assuring that all reasonable error possibilities have been considered, keeping interpersonal friction to a constructive level, recording and categorizing errors, following up on rework, and analyzing inspection results. He carries the experience he gains forward from project to project, and becomes an expert in making the most effective possible use of the participants' time.

Procedural differences include the following:

1. The inspection procedure is divided into six distinct steps, each of which has its own stated objectives. In walk-throughs, these steps exist but are blended together, with several objectives being addressed simultaneously.
2. At a walk-through meeting, the developer of the work product usually conducts the meeting and "reads" the materials. In inspections, the moderator conducts the meeting and designates someone other than the developer to read the

materials so that the developer's interpretation cannot inadvertently cover up an error.

3. Errors detected during the preparation period and discussed with the developer are usually not brought up at a walk-through meeting. During inspection meetings, *all* errors are noted and described to establish error patterns, improve the skills of all the participants, and flag any schedule slippage as early as possible.
4. Management does not usually participate in meetings in either form of review. In the case of structured walk-throughs, they do not attend because their presence may interfere with the free flow of discussion among the working group. In the case of inspections, managers are not discouraged from attending, but usually cannot add valuable input to an essentially technical discussion; they are therefore not needed. Management is, however, informed of the results of inspection meetings. In addition to traditional quantitative data, such as lines of code completed, management receives data regarding program quality (error rates) and time expected for rework and retesting.
5. Inspections emphasize uniformity and completeness through the use of checklists.
6. Errors are categorized during an inspection meeting and entered into a data base to improve the developers' and management's understanding of the types of errors that are occurring and where they most often occur. This information can be used to develop acceptable error rate standards to determine whether a module should be reinspected or rewritten. It can also assist management, testers, and those who are developing related modules to plan their work better and maintain more accurate schedules.

Briefly stated, the walk-through procedure relies heavily on technical team self-control and confines the visibility of shortcomings to within the development team itself. On the other hand, the inspection process allows for such visibility to extend beyond the development team and imposes technical controls from sources (the moderator and management) that are external to the team.

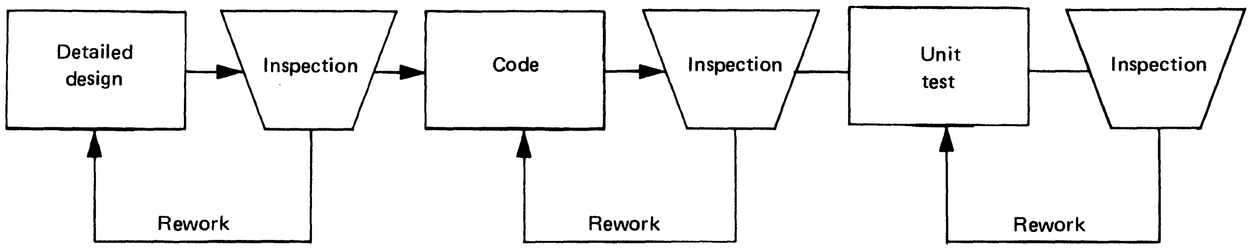
Note that the differences between structured walk-throughs and inspections tend to diminish as more of the formalities of inspections are included in walk-throughs. For instance, walk-throughs are now frequently scheduled into a project development plan.

A Controlled Experiment

Inspections are designed to improve both program quality and development productivity. This concept was tested in an experiment conducted by an IBM programming department. The inspection sample was considered to be representative in terms of its size and other characteristics. It was designed and developed by three designers and 13 programmers, was structured, and judged to be of moderate complexity. In the experiment, inspections were held, as shown in Figure 3, after detailed design and coding. After each inspection, the necessary rework was performed, and the work proceeded to the next phase. After including the time to prepare for and perform the inspections and do the rework, a net saving resulted when compared to similar projects using walk-throughs, the technique then in use. It was estimated that 94 programmers hours were saved per thousand lines of code as a result of the detailed design inspection, and 51 hours by the code inspection. This net saving translates into a 23 percent increase in coding productivity. (The unit test inspection did not result in a saving of programmer hours and therefore was eliminated subsequent to the experiment.

To determine that the productivity gain was not a result of the participants' knowledge that they were being studied and therefore producing at above normal rates, a control sample inspection was conducted later, after inspections had become accepted as normal practice. Inspection results for the control sample were essentially the same as those for the original experiment.

The study also compared the quality of the programs produced during the inspection sample study with a comparable component produced similarly, but using walk-throughs instead of inspections. Equivalent testing between post unit testing and system test showed the inspection sample to contain 38 percent fewer errors than the walk-through sample.



Net Coding Productivity

- Sample showed 23 percent net increase
- Poststudy sample showed 22 percent net increase

Net Savings in Programmer Hours/1000 LOC

- Design inspection: 94
- Code inspection: 51
- Unit test: -20

Program Quality

The inspection sample had 38 percent fewer errors/1000 LOC.

Figure 3. An IBM sample inspection study

Implementing Inspections

The first step in implementing inspections is a management decision that the following objectives and methods of the inspections technique will be appropriate for the installation.

The principal objectives of inspections are (1) to improve application system quality while reducing the cost of application development and maintenance and (2) to improve management's ability to control the development process. These objectives are accomplished by (1) providing a uniform method for inspecting development materials, (2) verifying that development materials are complete (satisfy the exit criteria for that development phase), (3) detecting errors and ambiguities in design and code at the earliest possible point in the development before they become progressively more expensive to rework, (4) verifying that all errors discovered during an inspection have been corrected before the work product (for example, a coded module) being inspected moves to the next development phase, (5) maintaining and analyzing records of errors found and time spent in inspections to improve both the development process *and* the inspections technique, and (6) providing early clues to the quality of developing products.

Following a management decision to implement inspections, full management support of the technique, as well as awareness by the development staff of management's support, is necessary for successful implementation.

The next step is to choose a pilot project. It is recommended that the project be currently in development and proceeding without any unusual difficulty. For best results, it should not be under any abnormal time constraints so that the technique can be judged in as neutral an environment as possible. It is advisable to choose a project for which all three major types of inspections (initial design, detailed design, and coding) can be used for each module to be inspected. No more than three or four months should elapse from the time that inspections are first used to the time that testing starts so that the developers can evaluate the results within a reasonable time period.

The moderator chosen for the pilot project, as for any inspection, should not be a member of the team whose work product is to be inspected because the moderator needs to bring an objective, outside viewpoint to the inspections. For the pilot project, however, it is desirable for the moderator to report to the same manager as that of the pilot project so that both the costs and the benefits of the inspections process are under the same control. The moderator will probably need to establish exit and reinspection criteria based on installation requirements and standards. There may also be a need to modify the reporting forms and checklists included in this text.

Beginning with the first inspection of the pilot project, the moderator should conscientiously maintain the records because the accumulation and analysis of inspection and man-hours data provide some of the major benefits of inspections. For example, the analyses can help to answer such questions as:

- How likely are we to meet our scheduled dates?
- How likely is this program or module to experience excessive maintenance?
- Are we spending too much or too little time on inspections?
- When should we reinspect a module?
- When should we rewrite a module?

As with any new technique, inspections will not experience immediate acceptance. Initially, there will be misgivings among the participants, and a certain amount of defensiveness can be expected from those whose work is inspected first, but these reactions should disappear as soon as the developers realize that they are not being singled out for criticism. The usual experience has been that the developers overcome their initial hesitation, begin to count on the constructive feedback they receive from their peers, and gain new confidence in their ability to predict project completion times. Once the pilot project has proven itself, the participants usually have a positive attitude toward inspections and should, by informal contact with other development personnel, help to generate interest in the technique. They can also be trained as moderators and can introduce additional groups or individuals to inspections.

Experience indicates that success with inspection pilot projects is usually associated with:

- Success of the inspections themselves (successful inspections find a significant number of serious errors)
- Conduct of the moderator (the moderator should encourage a constructive environment and use the participants' time effectively)
- Attitude of management (management must recognize the potential benefits of the inspection process, support the pilot effort, and use the results to improve the development process)

Chapter 2. The Inspection Technique

Inspections are described here as they have been used in a system control programming environment in IBM, although some terminology changes have been made to make the document more meaningful to an application development environment. Changes to the sample exit and reinspection criteria, checklists, time estimates, and reporting forms will be necessary to adapt the technique to each installation's environment.

Inspection Types

Inspections have been most commonly introduced at three "inspection points": immediately following initial design, detailed design, and coding. They are also used extensively within IBM following the development of test plans and test cases (see Figure 4).

Inspections can also be given an important role in the review of publications and requirements. The major part of this text discusses initial design, detailed design, and code inspections. Test plan and test case inspections are discussed in Appendix F.

Note that inspections can be used equally well in conventional and top-down program development environments. In the latter case, each module or group of modules in an application passes through the same development phases, although one module may do so weeks or months ahead of another.

Exit Criteria

One of the most important checks made in an inspection is to determine whether the work product has met the exit criteria for that phase and is thus eligible to proceed to the next development phase. Clearly defined exit criteria are necessary for this determination and for consistent error and man-hours inspection data. Appendix C contains sample exit criteria developed within IBM in a system control programming environment for the initial design, detailed design, and coding phases; some examples from this Appendix follow:

- For the initial design phase, each design process statement (HIPO statement, flowchart block, pseudocode statement) should be at a level of detail equivalent to 15-25 lines of executable source code.
- For the detailed design phase, each design process statement should be equivalent to 3-10 lines of executable source code.
- For the coding phase, the first clean compilation listing should be available.

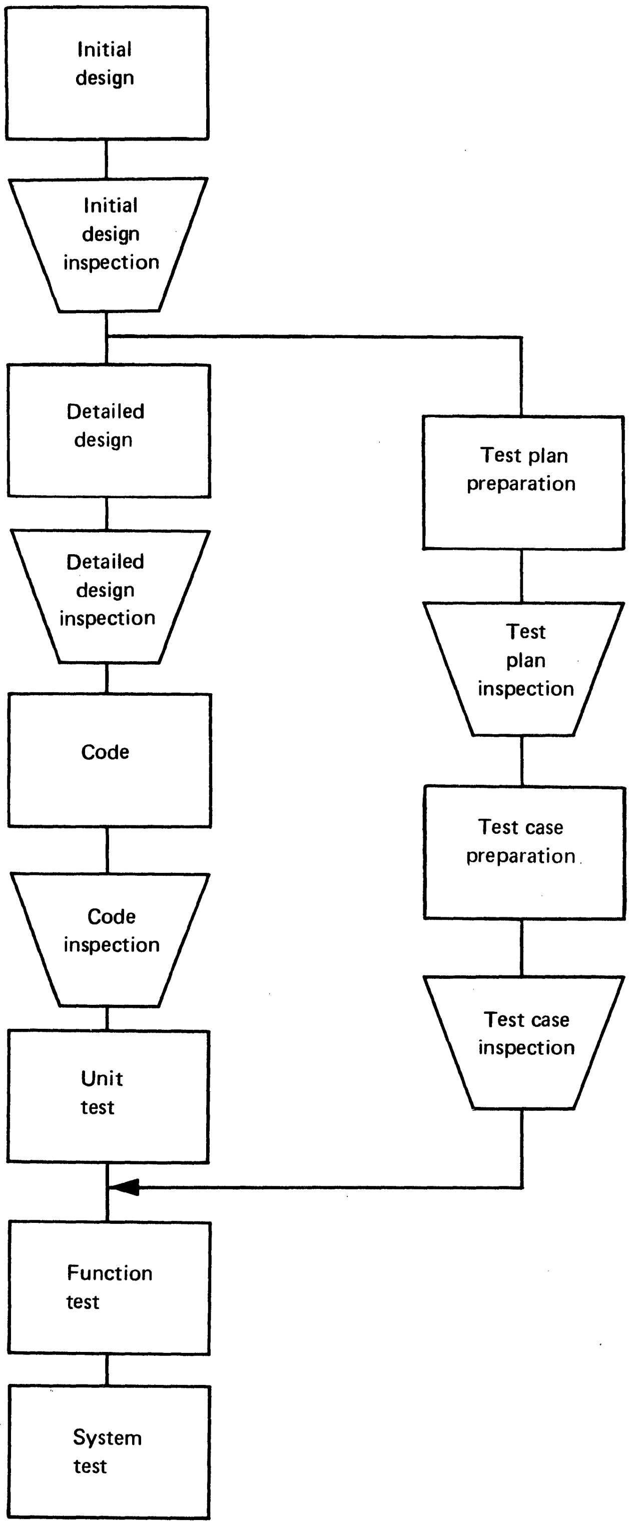


Figure 4. Inspection types

Steps

An inspection consists of six steps (see Figure 5):

1. *Planning*, during which the moderator schedules inspection activities and makes sure that inspection materials have been distributed
2. *Overview*, presented by the author of the materials to those who are to participate in the inspections
3. *Preparation*, during which the participants study the materials
4. *Inspection* meeting itself, during which the participants concentrate on finding errors in the materials
5. *Rework*, wherein the author corrects the errors found in the meeting and summarized and reported by the moderator
6. *Follow-up* (this step usually involves only the author, the moderator, and in larger installations, the inspection coordinator), during which the moderator certifies the author's rework and authorizes the next development phase, and during which inspection data is analyzed

Each step of the process has its own objectives. The moderator's objective in the planning step is to schedule the overview and inspection meetings and to ensure that all the materials are distributed to the participants. The purpose of the overview is to teach the participants the principles of the design or code to be inspected. Participants use the preparation step to become thoroughly familiar with the inspection materials. The moderator assures himself that the participants are adequately prepared before holding the inspection meeting. The only objective of the inspection meeting is to find errors. No education normally takes place at this session. In the rework step the author corrects errors found in the meeting and summarized and reported by the moderator. The error summary also keeps management informed about the quality of the developing product. The follow-up step has two objectives: to validate the correctness of the work done in the rework step (the moderator may call on others for assistance with this) and to make improvements in the application development process and the inspection process, through the analysis of inspection data. In larger installations, an inspection coordinator helps to perform this analysis function.

The cycle may be partially repeated by requiring a reinspection, as shown in Figure 6. At the end of the inspection meeting, the moderator determines whether the errors found were numerous enough to warrant a reinspection after rework has been completed.

Participants

An inspection team is composed of a *moderator*, the *author* or *developer* (designer, coder), and one or more *inspectors*. The moderator chooses the inspectors on the basis of special skills or knowledge they can bring to the inspection. The team normally consists of four persons. Large teams tend to expend too much manpower, while small teams do not generate the kind of human interaction that is effective in detecting errors. Under certain circumstances, however, a large team may be warranted. For example, if there are many linkages to other

Step	Inspection Team Participants	Objectives
1. Planning	Moderator	Scheduling/distributing materials
2. Overview	Designer and other participants	Education
3. Preparation	Participants — individually	Self-study
4. Inspection meeting	Entire group	Finding errors
5. Rework	Moderator, author	Correcting errors (after their summarization and reporting)
6. Follow-up	Moderator, author	Assuring correct rework, improving development and inspections

Figure 5. The six steps of an inspection

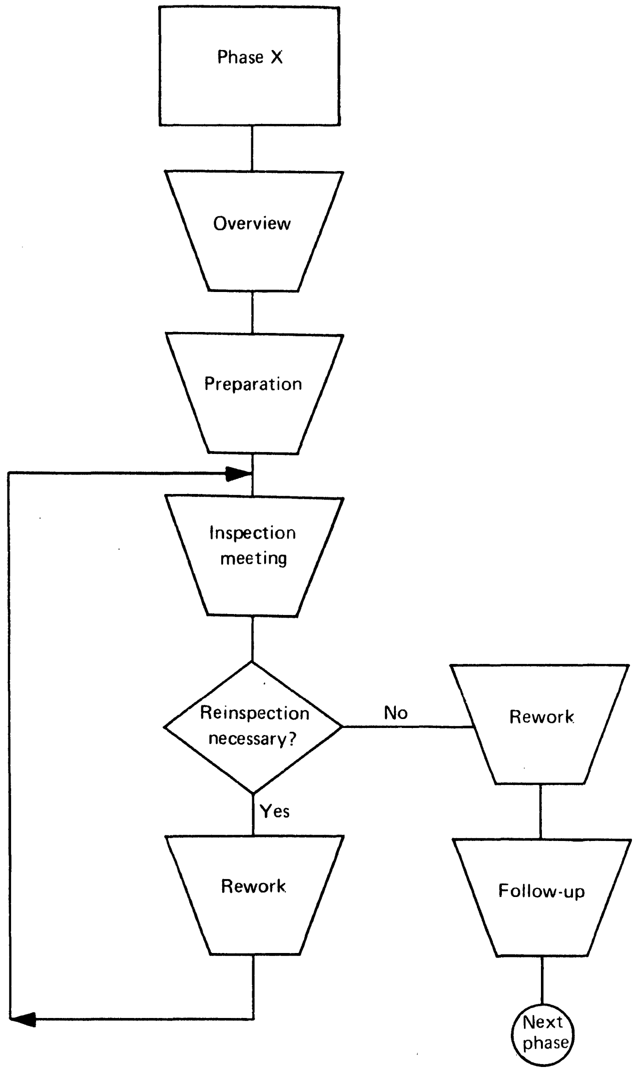


Figure 6. Reinspection points

modules, designers and/or coders of those other modules can contribute to the effectiveness of the inspection.

In many installations, project teams may consist of one or two people. In such cases, the inspection team may consist of the moderator, the author, and one or two inspectors recruited from other projects. The author may serve at some future time on an inspection team that is reviewing the work done by the inspectors of his work, and the current moderator may, in the future, have his work inspected at a meeting moderated by one of the inspectors from other projects.

Moderator

The moderator controls the activities of the inspection process and acts as the manager of the inspection meeting. To achieve impartiality, the moderator is usually not a member of the team that produced the work product to be inspected. (If a separate development assurance or development quality control group exists, the moderator is often chosen from that group.) Although the moderator must manage the inspection process, technical personnel (systems analysts or programmers) rather than managerial personnel serve as moderators. In system control programming development within IBM, the moderator is usually a skilled programmer because the same person is often called upon to moderate all inspections for a given project.

Throughout the inspection process, the moderator is responsible for efficient resource utilization and maximum problem detection. During the inspection meeting, the moderator must not allow the participants to engage in extended disagreements, solution hunting, or trivial matters. Thus, it is usually necessary for the moderator to be able to use personal sensitivity, tact, and drive, in balanced measures, to conduct successful inspections.

The moderator is responsible for the following:

- Scheduling each step of the inspection
- Selecting inspectors, with the assistance of the author and the approval of management, and assigning one of the inspectors to be the "reader" of the materials during the inspection
- Ensuring that inspection materials have been distributed to the inspection team by the development group
- Distributing checklists and error analysis reports to be used by the inspectors
- Training all inspectors in how to prepare for and participate in inspection meetings
- Planning the sequence of events for the inspection meeting; determining which parts of the materials will be inspected at given points in the meeting (the plan will be influenced by the kinds of materials, the complexity of the design, and whether changed design requires examination of other modules)
- Recording all errors detected and all problems raised
- Submitting reports after the inspection meeting
- Evaluating the error rate and determining whether a reinspection is required
- Estimating the amount of rework

Inspectors
Typically, the inspectors include the technical persons responsible for the prior and succeeding development phases. As a result, inspectors for a detailed design inspection may include the creator of the initial design, the coders of the design being inspected, and the testers responsible for testing the work product. For a code inspection, the general and detailed designers and the testers may be the inspectors. The development team leader or the chief programmer may be an inspector (but may not be the moderator). When the designer and the coder are the same person, an inspector may be an individual chosen from a closely related area, for example, the designer/coder of related components.

Inspections and the Project Plan

An important characteristic of the inspection process is that its time and costs are planned for and included in the overall project plan. Accordingly, when the project manager develops the project plan, he estimates the time and costs for each of the succeeding inspection steps for each inspection for each work product. While some projects may require only a few inspections, the volume of design and code materials involved in large projects may require many inspections. Estimated time and costs for rework are also included in the plan.

An inspection plan is usually prepared by the project manager before each program development project. It normally lists the names of the moderator and the inspectors for each planned inspection, the work product to be inspected, an estimate of the size of the work product*, and an estimate of the time required for each step of each inspection.

To develop time estimates, the estimator can use a table based on installation experience and developed by the installation's inspection coordinator. For example, the table of Figure 7 shows that the coordinator who prepared the table expects that 150 lines of noncommentary source statements can be inspected per hour during a code inspection, and that a code inspection meeting for a module with a size of 300 NCSS would take two hours. Note that only one overview meeting – an initial design overview meeting – may be necessary, especially for projects for which the inspectors for all phases are available at the time the meeting is held. For code inspections, the overview step is usually omitted, since the moderator and the inspectors are usually already familiar with the detailed design, as is the case when the code inspection team members also constitute the detailed design inspection team.

The estimates in the table of Figure 7 are based on a system control programming environment. Application programming environments have experienced different and much higher inspection rates. Therefore, each installation will need to develop its own table based on its own experience.

Maximums seem to be applicable for some activities; for example, inspectors probably cannot devote more than four hours per day to preparation activities, and inspection meetings seem to become much less efficient when their duration is greater than two hours. It has been found, however, that two inspections per day (separated by other activities) can be successful.

*This estimate is expressed in lines of code (LOC) or noncommentary source statements (NCSS), the sum of executable code instructions and declaratives. Instructions that invoke macros are counted only once, as are expanded macroinstructions. Comments are not counted.

Inspection Step	Inspection rates in NCSS/hour		
	Initial Design Inspection	Detailed Design Inspection	Code Inspection
Overview	500	500	–
Preparation for inspection	200	100	125
Inspection meeting	250	130	150
Rework	–	20 hrs/K.NCSS	16 hrs/K.NCSS

Figure 7. Inspection estimating table

Planning

Included in this initial step are the following moderator activities:

- Ensuring that the inspection materials are distributed to the inspection team by the development group
- Scheduling the overview and inspection sessions (scheduling of the meetings should permit the inspectors sufficient study time and permit flexibility in their schedules)

Overview Meeting

The purpose of the initial design overview meeting is educational. The designer's presentation should provide an understanding of the design's major functions and functional relationships as well as a detailed description of the materials. The designer includes in his presentation the design logic, external linkages, module relationships, data areas, etc. The overview is attended by all initial design inspectors and also by other project personnel who need a reasonably detailed description of the project.

Preparation for the Inspection Meeting

This step of the inspection process is performed individually by all inspectors and the moderator. The objective is for them to become thoroughly familiar with the inspection materials so that during the inspection meeting they will be better able to find errors. The inspection materials may include those from an earlier development phase. For example, in preparing for a detailed design inspection, the inspectors will need to refer to initial design materials.

While examining the materials, they:

- Check that the materials for the work product being inspected match materials from the previous phase. For example, detailed design specifications should not deviate from the requirements of the initial design specifications.
- Understand the required inputs and expected outputs (external linkages to and from each module).
- Understand the data area environment of each module.
- Comprehend the control flow and logic.
- Check that the exit criteria have been met.
- Note discrepancies or errors found so that they may be recorded during the inspection meeting. No attempt is made during this step to find solutions to any problems uncovered. That is the function of the rework step.

Inspectors should also familiarize themselves with the latest error analysis report for the type of inspection being conducted (see Figure 8). This type of analysis, prepared from data gathered from previous inspections, helps the inspectors focus on those errors which occur most frequently. (In Figure 8 these are design errors, logic errors, and prologue errors.) Such analyses also assist in the updating of checklists (see Figure 9) which list examples of each type of error. Inspectors should also use the appro-

Error Type		Error Category			Total Errors	Error %
		Missing	Wrong	Extra		
CC	Code Comments	5	17	1	23	6.6
DA	Data Area Usage	3	21	1	25	7.2
DE	Design Error	31	32	14	77	22.1
EL	External Linkages	7	9	3	19	5.5
LO	Logic	33	49	10	92	26.4
MN	Maintainability	5	7	2	14	4.0
OT	Other					
PE	Performance	3	2	5	10	2.9
PR	Prologue	25	24	3	52	14.9
PU	Prog. Lang. Usage	4	9	1	14	4.0
RU	Register Usage	4	2		6	1.7
SU	Storage Usage	1	8		9	2.7
TB	Test and Branch	2	5		7	2.0
		123	185	40	348	100.0

Figure 8. Code inspections error analysis

LOGIC

Missing

1. Are all constants defined?
2. Are all unique values explicitly tested on input parameters?
3. Are values stored after they are calculated?
4. Are all defaults checked explicitly tested on input parameters?
5. If character strings are created, are they complete? Are all delimiters shown?
6. If a keyword has many unique values, are they all checked?
7. If a queue is being manipulated, can the execution be interrupted? If so, is queue protected by a locking structure? Can the queue be destroyed over an interrupt?
8. Are registers being restored on exits?
9. Are all keywords tested in macro?
10. Are all keyword-related parameters tested in service routine?
11. Are queues being held in isolation so that subsequent interrupting requesters are receiving spurious returns regarding the held queue?
12. Should any registers be saved on entry?
13. Are all increment counts properly initialized (0 or 1)?

Wrong

1. Are absolutes shown where there should be symbolics?
2. On comparison of two bytes, should all bits be compared?
3. On built data strings, should they be character or hex?
4. Are internal variables unique or confusing if concatenated?

Extra

1. Are all blocks shown in design necessary or are they extraneous?

Figure 9. A portion of a detailed design checklist

priate checklist during the preparation step. Appendix D contains sample checklists developed within IBM in a system control programming environment. Each installation will probably wish to develop its own, based on its own environment and error experience.

Inspection Meeting

The inspection team and the author of the work product being inspected attend the inspection meeting. In general, others are not encouraged to attend, since the meeting is a working session with a fixed objective – to find errors.

At the beginning of the meeting, the moderator describes to the group the sequence in which the materials are to be examined. The author's role in the inspection is usually limited to answering technical questions. The author does not conduct the meeting – that is the responsibility of the moderator. The moderator has, before the meeting, appointed one of the inspectors (usually a "key" inspector) to "read" aloud the inspection materials. The inspection is more effective if the reader paraphrases the materials instead of reading them verbatim because paraphrasing tends to keep the other participants more alert and helps the author determine whether the materials can be understood. As the reading proceeds, each inspector looks for errors or ambiguities and for adherence to the exit criteria. The collective activity of group inspection tends to find more errors than the sum of individual inspector efforts.

As errors are recognized, the moderator records them in a problem list, estimating the rework time and classifying them by error type, error category, and error severity:

- *Error type* – for example, logic error, disagreement of code with design specifications, lack of adherence to maintainability or performance requirements.
- *Error category* – missing, incorrect, or extra design or code.
- *Error severity* – major or minor. An error which would cause malfunction or which precludes the attainment of expected or previously specified results is considered to be a major error.

The moderator might record the following to indicate a logic error (LO), categorized as incorrect code (W), of major severity (MAJ), with an estimated rework time of 20 minutes:

LO/W/MAJ - Line 169 - While counting the number of leading spaces in NAME, the wrong variable (I) is used to calculate "J."
Rework = 20 minutes

At the conclusion of the inspection meeting, the moderator seeks the team's agreement with the correctness of the error list and decides whether a reinspection is required after the errors have been corrected. This decision is based on installation-established standards. For example, an installation's standards may specify that detailed design materials may not exceed an error rate of five percent and that code may not contain more than one major problem per 25 lines of source code.

Initial Design Inspection Meeting
For an initial design inspection meeting, the moderator usually chooses either the team leader or the author of the detailed design as the inspector "to read" the design. As the "reading" proceeds, each inspector follows the logic looking for errors or ambiguities, and questioning the design logic as necessary.

Detailed Design Inspection Meeting
At a detailed design inspection meeting, the programmer often serves as the reader. Certain users of inspections have the initial designer perform that function, believing that the initial designer is in a better position to evaluate linkages to other parts of the system that may have been overlooked by the author of the detailed design. At the meeting, the team checks the detailed design materials for consistency with the initial design, covers every piece of logic at least once, and checks each design statement for ambiguities that could lead to coding errors. The emphasis in this inspection meeting is on detecting omissions; user experience has shown that most problems discovered during a detailed design inspection are design omissions. Inspectors also check for incorrect design and areas of overdesign (extra, unnecessary logic).

Code Inspection Meeting
At a code inspection meeting, the detailed designer (unless the detailed designer is also the programmer) usually serves as the reader. A statement-by-statement comparison is usually made between the current detailed design and the code. Every line of code is read and every path is checked. Code inspections emphasize the detection of wrong rather than missing or extraneous code. Correctness of structured code may be verified by either of the two following methods, although the second method is usually more successful:

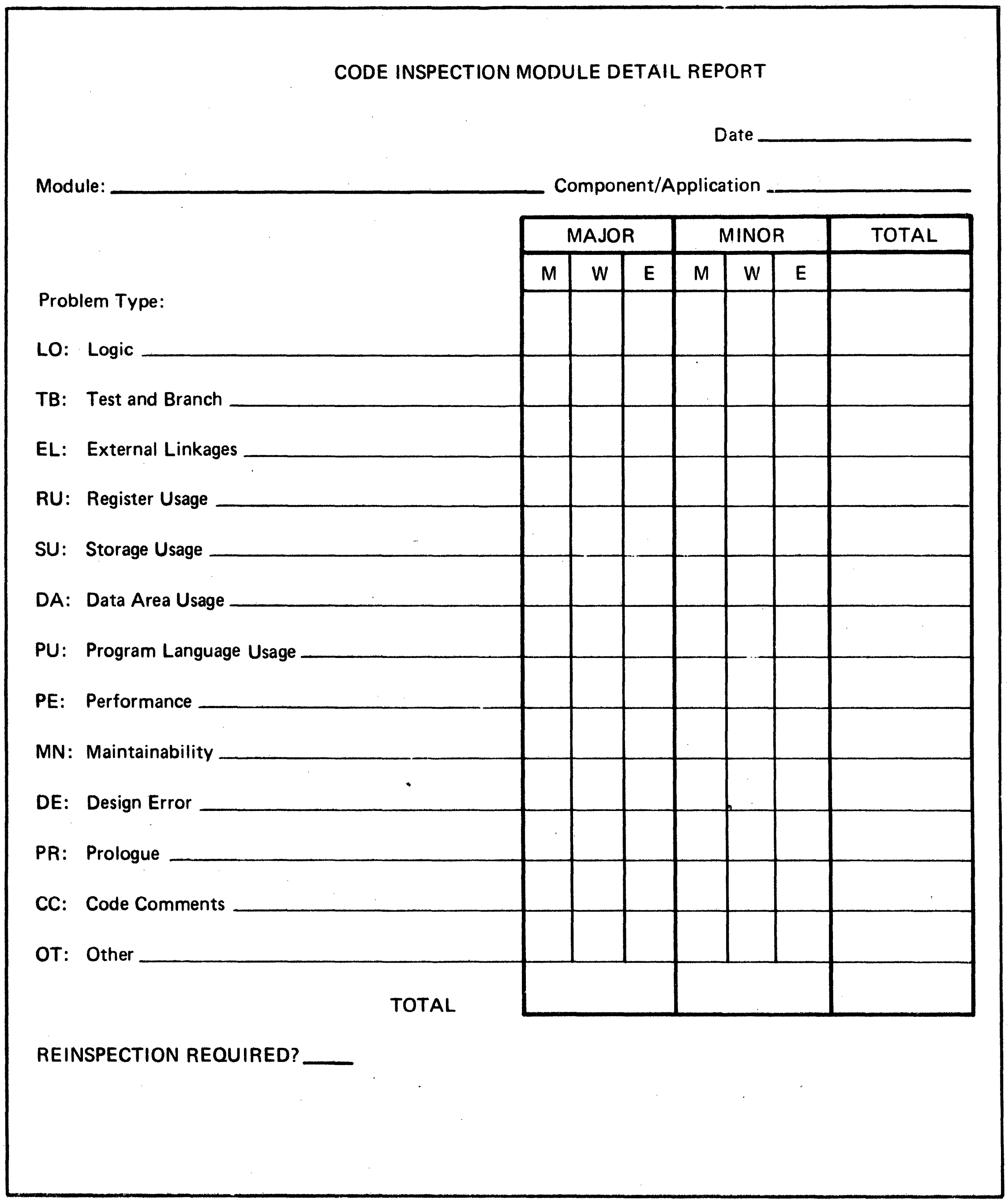
1. The code is traced in sequential page order, starting with the main line segment, followed by the lower-level segments.
2. Main line logic is traced completely through every subroutine until the main line logic has been completely traced. Then all remaining secondary paths are traced.

Rework

Within one day of the inspection meeting, the moderator distributes to the participants the code inspection module detail report (see Figure 10), which summarizes, and to which is attached, the problem list prepared by the moderator during the inspection meeting. The author then proceeds to correct the problems specified in the report and list. The moderator also distributes to the team and project management the summary inspection report (see Figure 11), which may contain the results of one inspection encompassing several modules – modules forming a particular part of the system. Note that in the summary report the moderator records actual hours spent for the first three inspection steps – overview, preparation, and meeting – and estimated rework and follow-up hours. Management can compare these estimates, which are based on errors actually found, to the estimates in the project plan and determine whether the project is still on schedule. Note that since estimates for rework *were* included in the project plan, project schedules and costs are much less likely to be affected by any necessary rework.

Management, then, can use the summary report to keep track of the project's quality and schedule. The author, by comparing the detail report to the error analysis report (see Figure 8) prepared by the inspections coordinator, can determine in what areas to strive for improvement.

Appendix E contains samples of the module detail reports (one for each type of inspection) and the summary report, along with explanations of the various report fields.



CODE INSPECTION MODULE DETAIL REPORT

Date ________________

Module: ________________________ Component/Application ________________

	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
Problem Type:							
LO: Logic							
TB: Test and Branch							
EL: External Linkages							
RU: Register Usage							
SU: Storage Usage							
DA: Data Area Usage							
PU: Program Language Usage							
PE: Performance							
MN: Maintainability							
DE: Design Error							
PR: Prologue							
CC: Code Comments							
OT: Other							
TOTAL							

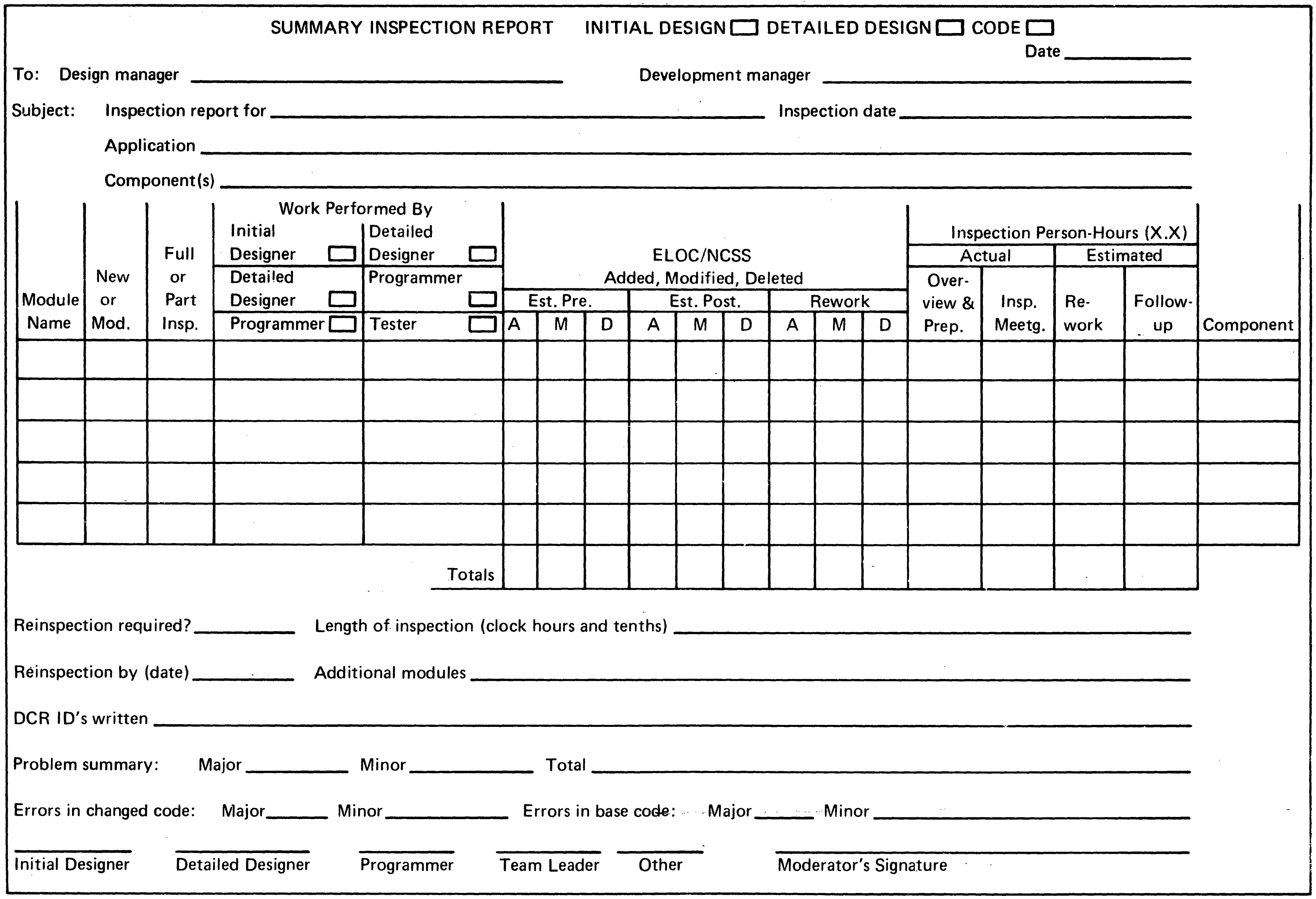
REINSPECTION REQUIRED? ____

Figure 10. Code inspection module detail report

Follow-Up

After the errors and ambiguities have been corrected, and if a reinspection has not already been scheduled, the moderator verifies the completeness and accuracy of the reworked materials and gives his formal approval, thereby allowing the development effort to move forward. If the moderator finds that the amount of rework warrants a reinspection ("more than five percent of the materials have been reworked" is a possible criterion), or if the moderator simply believes that the reworked materials should be reviewed by others, a reinspection may be scheduled at this time.

From the error count and inspection hours information in the detail and summary reports, the coordinator (or moderator, if the installation does not have a coordinator) develops the data base used to produce the error analysis report (Figure 8) and other reports used to improve the application development and inspection processes, as discussed in Chapter 3.



SUMMARY INSPECTION REPORT INITIAL DESIGN ☐ DETAILED DESIGN ☐ CODE ☐

Date ____________

To: Design manager ______________________________ Development manager ______________________________

Subject: Inspection report for ______________________________ Inspection date ____________________

Application ____

Component(s) ____

			Work Performed By		ELOC/NCSS Added, Modified, Deleted									Inspection Person-Hours (X.X)				
					Est. Pre.			Est. Post.			Rework			Actual		Estimated		
Module Name	New or Mod.	Full or Part Insp.	Initial Designer ☐ / Detailed Designer ☐ / Programmer ☐	Detailed Designer ☐ / Programmer ☐ / Tester ☐	A	M	D	A	M	D	A	M	D	Over-view & Prep.	Insp. Meetg.	Re-work	Follow-up	Component
				Totals														

Reinspection required? __________ Length of inspection (clock hours and tenths) ______________________________

Reinspection by (date) __________ Additional modules ______________________________

DCR ID's written ____

Problem summary: Major __________ Minor ____________ Total ______________________________

Errors in changed code: Major ______ Minor ____________ Errors in base code: Major ______ Minor ____________

Initial Designer ____________ Detailed Designer ____________ Programmer ____________ Team Leader ____________ Other ____________ Moderator's Signature ____________

Figure 11. Summary inspection report

Chapter 3. Using the Inspections Data Base

Comparing the application development process again to industrial continuous flow processes, it can be seen that two major concepts of industrial process control – feedback and feedforward – can be applied to the application development process. In feedback, a step in the process is measured with the measurement used to make adjustments to that step so that it will make a more significant contribution to the whole process. In feedforward, the measurement of a step is used to affect succeeding steps in the process.

Examples of feedback and feedforward in the application development process are:

- Data about a designer's or a programmer's work product resulting from an inspection is *fed back* to him so that he can see the types of errors he is making, compared to others in the installation, and so produce a better quality product in his next assignment.
- Data developed about a product during an inspection can, when known to people involved in succeeding development phases *(feedforward)*, help in improving the product. For instance, if a design inspection reveals that certain types of errors have been more common than usual, the tester should be prepared to design test cases that concentrate on that type of error.

A prerequisite for process control in an industrial process or in application development is the availability of data. The inspection process provides a disciplined technique for the collection of data about applications in development. When this data is systematically collected and analyzed, steps can be taken to improve those applications and the application development process. Some specific uses of this data follow.

Identification of "error-prone modules". A listing of initial design, detailed design, or code design data, or some combination thereof, as in Figure 12, which combines detailed design and code design inspection data, highlights the modules for which the error density was highest. A commonly used measure of error density is errors per 1000 lines of noncommentary source statements, or "errors/K NCSS. As can be seen from Figure 12, this measurement is developed using the following formula:

$$\frac{\textit{Number of errors in the module}}{\textit{Number of lines of noncommentary source statements in the module}} \times 1000$$

If the error detection efficiency of inspections is fairly constant, module and project quality can be predicted fairly early – certainly before unit testing – and steps can be taken to improve quality considered to be below standard. For example, if error detec-

Module Name	No. Errors	Lines of Code	Error Density (Errors/K NCSS)
ECHO	4	128	31
ZULU	10	323	31
FOXTROT	3	71	28
ALPHA	7	264	27 Average error density
LIMA	2	106	19
DELTA	3	195	15
.	.	.	.
.	.	.	.
.	.	.	.

Figure 12. A listing of most error-prone modules

tion efficiency of a design inspection is 50 percent and the inspection found ten errors in a module, it can be estimated that there are ten errors remaining in the module. If this is below standard, management could decide to reinspect or redesign the module before coding it. Another option might be to plan an especially rigorous test for the module and to concentrate the testing on those errors of the type found in the inspection.

Early identification of high-incidence error types. Comparing the distribution of error types in the first modules inspected for a project to the installation's normal or usual distribution of errors provides an early warning of possible project problems and can make it possible to take steps to remedy the problem for the remaining modules of the project. Figure 13, for example, shows that module X contained almost twice as many linkage errors as is usual. Provision could be made to test module X and other early modules to remove the unusually high incidence of such errors. In addition, other developers may be warned of the situation and of possible peculiarities in this application, and so be better prepared to perform their functions for this project.

	No. Errors	%	Normal/Usual Distribution %
Logic	23	35	44
Linkage	21	31	18
Data Areas	6	9	13
.	.	8	10
.	.	7	7
.	.	6	6
.	.	4	2
		100%	100%

Figure 13. Distribution of error types for module X

Self-improvement. Developers can use a listing such as that in Figure 14 to concentrate their efforts on the areas of their own work that most need improvement. They can see what types of errors they made in the subject module and how frequently they occurred compared to the normal or usual distribution of errors in the installation.

Improvement of the development process. Cumulative error distributions by error type for a particular type of inspection, as in Figure 14, can show development management which types of errors are most prevalent and thus help focus attention toward reducing their frequency. Such a report, may, for example, show overall shortcomings in the training given to development personnel and the need for changes in development standards and procedures. When compared to a cumulative error distribution of a later period, as in Figure 15, management can determine the progress made in reducing the frequency with which certain errors occur.

Improvement of the inspection process. Analysis of inspection results may show that inspections conducted by a particular moderator yield significantly fewer major and considerably more minor errors per man-hour of effort when compared with inspections conducted by other moderators. Such analyses can help moderators redirect and discipline their efforts.

Analysis of reports showing error distributions by type can assist moderators in developing checklists that will direct the attention of inspection teams to those errors that occur most frequently.

The person hours required by inspections can be continually compared to installation averages to evaluate the inspection process. Inspections should not only improve the quality of programs but should reduce the net time required to develop them; the total time required by all participants for all inspections, plus total development time, should be less than the development time along without inspections.

Inspections Data and Personnel Considerations

In extensive IBM use of inspections and some use by non-IBM installations, employee dissatisfaction has not been reported as a problem. Primarily this is because developers find inspections to be of assistance to them in assuring the quality of their work and improving their ability to meet schedules. Also, managers quickly learn that it is undesirable and unrealistic to rely solely on inspections data to evaluate an individual's work quality. Experience has shown that management sees the necessity of continuing to weigh all available facts when appraising an individual. Some of the reasons for this are:

1. Only the quality of the final product, and not of intermediate stages, should be considered when evaluating the developer. The same principle applies to productivity: the overall time spent by the developer is much more significant than the time spent at any one stage.
2. The number of errors uncovered depends largely upon the thoroughness of the inspection; the better the inspection, the greater the number of errors detected (all other factors being equal).

Error Type		Error Category			Total Errors	Error %
		Missing	Wrong	Extra		
CC	Code Comments	5	17	1	23	6.6
DA	Data Area Usage	3	21	1	25	7.2
DE	Design Error	31	32	14	77	22.1
EL	External Linkages	7	9	3	19	5.5
LO	Logic	33	49	10	92	26.4
MN	Maintainability	5	7	2	14	4.0
OT	Other					
PE	Performance	3	2	5	10	2.9
PR	Prologue	25	24	3	52	14.9
PU	Prog. Lang. Usage	4	9	1	14	4.0
RU	Register Usage	4	2		6	1.7
SU	Storage Usage	1	8		9	2.7
TB	Test and Branch	2	5		7	2.0
		123	185	40	348	100.0

Figure 14. Code inspections error analysis

ERROR TYPE	PERCENT Jan. 1	Aug. 1
Code Comments	2.4	.9
Data Area Usage	7.6	8.0
Design Error	35.0	22.9
External Linkage	7.0	11.8
Design Documents	3.0	3.0
Logic	30.7	34.5
Maintainability	.3	.2
Other	.3	1.8
Performance	.6	.2
Prologue/Prose	2.7	1.9
Program Language	3.3	4.4
Register Usage	3.0	3.7
Storage Usage	.8	2.2
Test and Branch	3.3	4.5
	100.0	100.0

Figure 15. Cumulative error distribution by type

3. The number of errors made by a developer is affected by the complexity of the program, the quality of prior development, the time available for development, the time actually spent by the developer before the inspection, and differences in the modes of operation of individual developers (such as "very slow and error-free" or "very fast and error-prone").
4. A tendency to minimize the number of errors uncovered and their severity may occur if the errors are used to appraise the quality of work performed by an individual, since the members of the inspection team will also have their development work subjected to inspections. Thus, the value of statistics gathered during inspections may be largely destroyed.
5. If errors are "punished", inspections may be delayed unduly by the developer who wants to detect and to correct as many errors as possible before the inspection to avoid having a "poor" record entered into the data base.

Chapter 4. Moderator Training Hints

This chapter is designed to assist an installation's first moderator in preparing for his role and in training other moderators. Subsequent moderators can also use this chapter when preparing to conduct inspections.

The moderator-trainee should first study this text, including Appendixes A-E. The case study presented in Appendix A consists of a COBOL source listing, together with supporting design documents. Although it is written in COBOL, the case study has been used with little difficulty by people with programming, but no COBOL, experience.

Then, the inspection steps for the case study inspection should be planned: the overview meeting, the preparation step, the inspection meeting, rework, and follow-up.

The participants in the case study inspection should be selected from members of the project selected by management to be the inspections pilot project. As indicated in Chapter 1, under "Implementing Inspections", it is desirable for the moderator to report to the same manager as that of the pilot project, although he should not be a member of the pilot project team. In addition, a moderator-trainee may be invited to act as an inspector. The participants can prepare themselves for their first inspection by reading Chapters 1 and 2 of this manual.

Since the author of the case study is not available, the moderator should appoint one of the inspectors to assume the role of author and present the overview of the case study during the overview meeting.

Sufficient time should be scheduled for the preparation step, and the case study materials, including the COBOL checklist (Appendix D), should be delivered to the participants early enough so that the inspectors can adequately prepare for the inspection meeting. The guidelines suggested in Figure 7 can initially be used to schedule preparation time. When delivering the materials to the inspectors, the moderator should notify the inspector he has chosen to be the "code reader" during the inspection meeting, because this person will want to make special preparations for that activity.

During the inspection meeting, the moderator should write a short description of each error discovered, or, if this is not possible, mark the errors on the listing and write the descriptions after the meeting.

After the program listing has been completely inspected, the moderator should read to the participants his interpretation of each of the errors and obtain agreement that it is accurate. Then the moderator should make a decision as to whether a reinspection is required and so inform the participants. (Since inspection standards unique to this installation have not yet been developed, the decision can be based on whether there are more than 25 major errors per 1000 lines of noncommentary source statements.) Finally, the participants should discuss the error list in the case study solution (Appendix B).

After the meeting, the moderator should prepare the detail and summary inspection reports (guidelines are given in Appendix E), compare the reports against the samples shown in the case study solution, and inform management of the results of the case study inspection.

Plans should then be made for the pilot project's first inspection. Before holding that inspection, however, the moderator should, in conjunction with management and the installation's standards group, develop the exit criteria to be used for the pilot project's inspections. The appropriate checklists in Appendix D may also be tailored to the installation's use, as may the reporting forms shown in Appendix E.

Beginning with the pilot project's first inspection, the error and inspections hours data from the inspection detail and summary reports is entered into the inspections data base. Only with a conscientiously maintained data base can the benefits described in Chapter 3 be achieved.

The second and subsequent moderators can learn the moderator role in a similar manner. In addition, it may be helpful for moderator-trainees to observe an inspection of one of the installation's development projects; afterward, they can benefit from a critique of the session with the session moderator.

Appendix A: Case Study

This case study provides materials (Figures 16, 17, and 18) for a code inspection of a COBOL program that reads student enrollment cards, checks them for errors, and prints out card images with errors flagged. Although the application may be somewhat trivial, it is of sufficient difficulty for a realistic code inspection.

The materials to be used by the participants in the inspection follow. They consist of design documents, a card layout form, a sample output listing, and a program listing. The COBOL checklist in Appendix D may be used during the preparation and inspection steps to focus the inspectors' attention on common errors. Appendix B is the "school solution" of the case study. Chapter 4, "Moderator Training Hints", should be reviewed before the case study materials are reviewed.

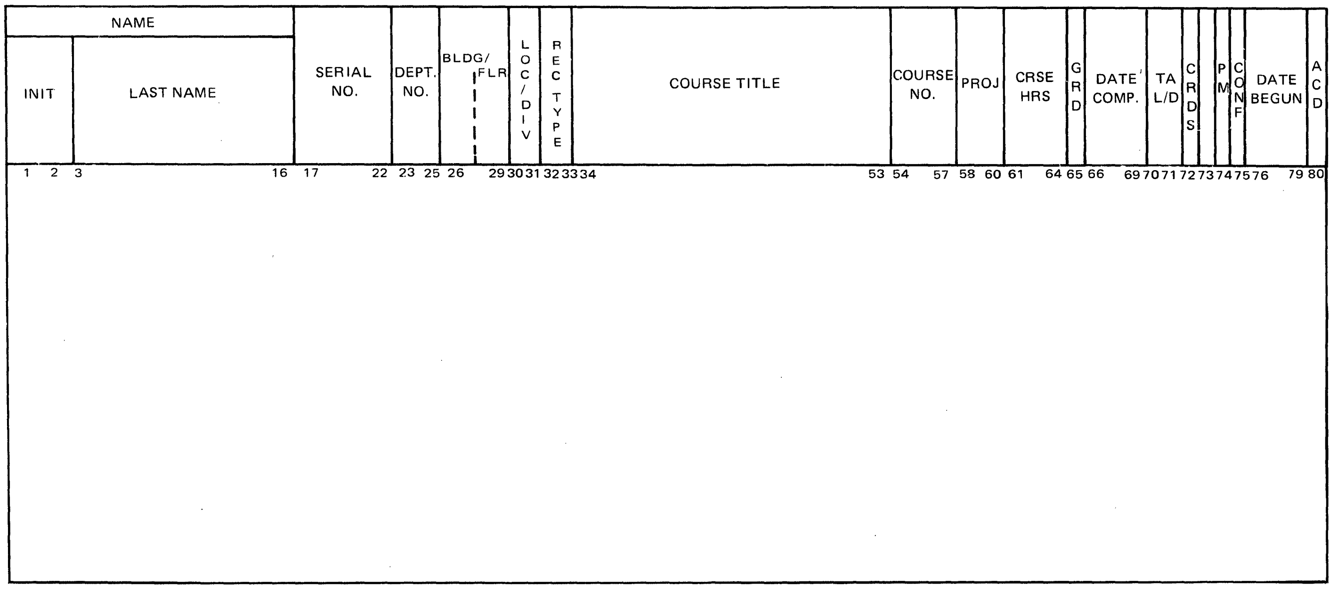
Initial Design Requirements

The "CHECKER" program is to read cards containing student enrollment and grade information, validity-check the data on the cards, and print out the card images with errors appropriately flagged. More specifically:

- Two types of data cards are processed: (1) completion cards, and (2) enrollment cards. The two types must be grouped separately in the input deck, with the completion cards followed by the enrollment cards, separated by a card with a $ in column 1.
- Enrollment cards are distinguished from completion cards by a blank in the grade field (column 65).
- Card images are printed sequentially, with enrollment cards printed on a separate page.
- Card fields that fail validity checking (that are in error) are flagged with an underscore and a vertical bar under the field in error.

Detailed Design Specifications

Figure 16a shows the layout of the enrollment and completion cards and Figure 16b shows the detailed specifications for each card field. (Any time "error" is indicated in these specifications, the error flag should be set for the column(s) in error.) Figure 17 shows a sample report.



NAME																			
INIT	LAST NAME	SERIAL NO.	DEPT. NO.	BLDG/ FLR	LOC/DIV	REC TYPE	COURSE TITLE	COURSE NO.	PROJ	CRSE HRS	GRD	DATE COMP.	TA L/D	CRDS		PM	CONF	DATE BEGUN	ACD
1 2	3 16	17 22	23 25	26 29	30 31	32 33	34 53	54 57	58 60	61 64	65	66 69	70 71	72	73	74	75	76 79	80

Figure 16a. Enrollment and completion card layout

Card Column	Field Name and Description
1-2	INITIALS
1	If $, this is a control card indicating that all following input cards should bear no grade (col. 65 = blank); if not $ or a letter – error.
2	If not letter or blank – error.
3-16	LAST NAME
3	Last name must start in col. 3; if not, all leading blanks are flagged as errors. If not letter, blank, single quote, or hyphen in cols. 3-16 – error.
17-22	SERIAL NO.
17	If not W or numeric – error.
18-22	If not numeric – error.
23-25	DEPT. NO.
	If col. 23-25 not letters or numbers – error.
26-29	BLDG/FLR
	1. If Record Type (col. 32-33) = 10 (Graduate Work Study)
	a. Cols. 26-29 must contain a valid school code. Valid school codes: 0293, 0946, 1126, 1361, 6012.
	b. If cols. 26-29 not valid school code – error.
	2. If Record Type not 10
	a. If LOC = K, cols. 26-28 must contain a valid Bldg. No. Valid Bldg. Nos.: 001,002, 003, 004, 005, 021, 022, 023, 024, 033, 034, 035, 042, 043, 045, 052, 057, 201, 202, 210, 211, 212, 959, 960, 964, 965, 966. If Bldg. No. not valid – error.
	b. If LOC = X, cols. 26-29 must contain a valid country code. Valid codes: AUST, BELG, CANA, ENGL, FRAN, GERM, ITAL, JAPA, NETH, SCOT, SWED, SWIT. If code not valid – error.
	c. If LOC = Z, cols. 26-29 must contain valid three-letter locations codes (col. 29 must be blank). Valid codes: BOT, LGC, LSC, PAA, RMD. If code not valid – error.
	d. If LOC not, K, X, or Z, cols. 26-28 must be a combination of blanks, numbers, or letters. If not blanks, numbers or letters – error.
	e. If LOC not X or Z and Floor (col. 29) not blank, numbers, or letters – error.
30-31	LOC/DIV
	1. If col. 30 not valid location code – error. Valid LOC codes: A, B, C, D, E, F, G, H, J, K, L, M, N, O, P, Q, R, S, T, V, W, X, Y, Z.
	2. If col. 31 not valid division code – error. Valid DIV codes: A, C, D, F, G, H, I, J, K, L, N, O, P, Q, R, S, T, W, X, Y.
32-33	REC TYPE
	If Record Type not valid – error (col. 32 and col. 33). Valid Record Types: 10, 35, 40, 46, 47, 52, 54.
34-53	COURSE TITLE
	Title must start in col. 34. If not, all leading blanks are flagged as errors.
54-57	COURSE NO.
	If col. 54 not numeric (0-9) or S, F, or D – error. If cols. 55-57 not numeric – error.
58-60	PROJ
	If col. 32 and col. 33 (Record Type) = 47, cols. 58-60 must be V30 or V31. If col. 32 and col. 33 = 46, cols. 58-60 must be V28 or V29. If col. 32 and col. 33 = 10, cols. 58-60 must be GWS. If col. 32 and col. 33 not 10, 46, or 47, cols. 58-60 must be V32 or V33. If any of the above do not match – error.
61-64	COURSE HRS
	If cols. 61-64 not numeric – error. If cols. 61-64 greater than 320 – error.
65	GRADE
	1. If checking completion cards ($ in col. 1 card not yet encountered), col. 65 must have valid grade code. Valid grade codes: A, B, C, D, F, I, N, O, S, T, U, W, Z. If valid grade code not found – error.
	2. If checking enrollment cards ($ card encountered), col. 65 must be blank. If not blank – error.
66-69	DATE COMP
	1. If cols. 66-69 not numeric – error.
	2. If col. 66 and col. 67 (month) greater than 12 – error.
70-71	TA L/D
	1. If col. 70 not K – error.
	2. If col. 71 not L – error.
72	CRDS
	1. If col. 32 and col. 33 = 1, col. 72 must be numeric. If not – error.
	2. If cols. 32-33 not 10, col. 72 must be blank or numeric. If not – error.
73	FLAG
	If col. 73 not blank, 1, or 2 – error.
74	PM
	If col. 74 not 1-9 – error.
75	B or F
	If col. 75 not blank, B, or F – error.
76-79	DATE BEGUN
	1. If cols. 76-79 blank, bypass further checking of these columns.
	2. If cols. 76-79 not numeric – error.
	3. If col. 76 and col. 77 (month) greater than 12 – error.
	4. If col. 78 and col. 79 (year begun) greater than col. 68 and col. 69 (year completed) – error.
80	ACD
	If col. 80 not blank, A, C, or D – error.

Figure 16b. Card field detailed specifications

```
 |     NAME    | ID  DPT     L RT|        TITLE        C_NO   HRS G    TA F B    D
 |             |    |  BLDG|D|  |                    |   PRJ|   |COMP| C M BEG |
 |             |    |  |    | | |                    |   |  |   |    | | | |   |
 |             |    |  |    | | |                    |   |  |   |    | | | |   |
NLBROWN        40460363M6012KL10DIGITAL SIMULATION  6000GWS0030A0674KL225 0474C
                                                                   | |
NLBROWN        40460363M6012KL10SYS ANALYSIS        6001GWS0030A0674KL225 0474C
                                                                   | |
REGREEN        50647001L6012KL10SYS ANALYSIS        6000GWB0030A0674KL225 0474C
                    | | |                               | | |        | |
*****************************************************************

      THE FOLLOWING ARE ENROLLMENTS--GRADES SHOULD BE BLANK

*****************************************************************

 |     NAME    | ID  DPT     L RT|        TITLE        C_NO   HRS G    TA F B    D
 |             |    |  BLDG|D|  |                    |   PRJ|   |COMP| C M BEG |
 |             |    |  |    | | |                    |   |  |   |    | | | |   |
 |             |    |  |    | | |                    |   |  |   |    | | | |   |
$
                    | | |                               | | |        | |
VSSMITH        2078145OL6012KL10STOCHASTIC MODELS OR6000GWS0030 0674KL225 0474C
                    | | |                               | | |        | |
JBWHITE        41783466L6012KL10ADV COMPILER DES    6000GWS0030 0674KL225 0474C
                    | | |                               | | |        | |
```

Figure 17. Checker output

```
PP 5740-CB1 RELEASE 2.1  OCT 11, 1977              IBM OS/VS COBOL

       IDENTIFICATION DIVISION.
       PROGRAM-ID.    'CHECKER'.
       REMARKS.  THIS PROGRAM READS CARDS CONTAINING STUDENT
                 ENROLLMENT AND GRADE INFORMATION, VALIDITY CHECKS
                 THE DATA ON THE CARDS, AND PRINTS OUT THE CARD
                 IMAGES WITH ERRORS FLAGGED APPROPRIATELY.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       OBJECT-COMPUTER.  IBM-370.
       SOURCE-COMPUTER.  IBM-370.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT ENROLLMENTS  ASSIGN TO UT-S-ENROLL.
           SELECT LISTING      ASSIGN TO UT-S-LISTING.

       DATA DIVISION.
       FILE SECTION.
       FD  ENROLLMENTS
           LABEL RECORDS IS STANDARD
           DATA RECORD IS CARD-IN.
       01  CARD-IN    PIC  X(80).

       FD  LISTING
           LABEL RECORD IS STANDARD
           DATA RECORD IS LINE-OUT.
       01  LINE-OUT   PIC  X(81).

       WORKING-STORAGE SECTION.
       77  EOF-IND                 PIC  X.
       77  GRADE-CHECK-SWITCH      PIC  X.
       77  COUNTER                 PIC  99  USAGE COMPUTATIONAL.
       77  THIS-YEAR               PIC  X(2).
       77  I                       PIC  99  USAGE COMPUTATIONAL.
       77  J                       PIC  99  USAGE COMPUTATIONAL.
       77  IT                      PIC  X.
       77  SPLAT                   PIC  X(20)  VALUE IS ALL '|'.
       01  HEADER                  PIC  X(81)  VALUE IS
           '1 |     NAME     | ID  DPT    L RT|       TITLE       C-NC
      -    '  HRS G   TA F B   D'.
       01  HEADER2                 PIC  X(81)  VALUE IS
           '  |              |     |  BLDG|D| |                   |   PR
      -    'J|   |COMP| C M BEG |'.
       01  HEADER3                 PIC  X(81)  VALUE IS
           '  |              |     |  |   | | |                   |   |
      -    ' |   |    | | | |   |'.
       01  HEADER4                 PIC  X(81)  VALUE IS
           '0 **********************************************'.
       01  HEADER5                 PIC  X(81)  VALUE IS
           '0         THE FOLLOWING ARE ENROLLMENTS--GRADES SHOULD BE BL
      -    'ANK'.
```

Figure 18a. Checker program (1 of 13)

```
01  CARD.
    05  INIT1                   PIC  X.
    05  INIT2                   PIC  X.
    05  NAME                    PIC  X(14).
    05  NAME-CHARACTER REDEFINES NAME          PIC  X  OCCURS 14.
    05  SERIAL-NC               PIC  X(6).
    05  SERIAL-NC-CHARACTER REDEFINES SERIAL-NC  PIC  X OCCURS 6.
    05  DEPT-CHARACTER          PIC  X  OCCURS 3.
    05  SCHOOL                  PIC  X(4).
    05  OTHER-LOCATION REDEFINES SCHOOL PIC  X(4).
    05  FILLER REDEFINES OTHER-LOCATION.
        10 BLDG                 PIC  X(3).
        10 FLOOR                PIC  X.
    05  LOCATION                PIC  X.
    05  DIV                     PIC  X.
    05  RECORD-TYPE             PIC  X(2).
    05  TITLE                   PIC  X(20).
    05  COURSE-NC               PIC  X  OCCURS 4.
    05  PROJECT                 PIC  X(3).
    05  COURSE-HOUR-VALUE       PIC  X(4).
    05  COURSE-HOUR-DIGIT  REDEFINES COURSE-HOUR-VALUE
                                               PIC  X  OCCURS 4.
    05  GRADE                   PIC  X.
    05  DATE-COMPLETED.
        10  MONTH-COMPLETED     PIC  X(2).
        10  YEAR-COMPLETED      PIC  X(2).
    05  DATE-COMPLETED-DIGIT  REDEFINES DATE-COMPLETED
                                               PIC  X  OCCURS 4.
    05  TEACHING-LOCATION       PIC  X.
    05  TEACHING-DIVISION       PIC  X.
    05  CREDITS                 PIC  X.
    05  FLAG                    PIC  X.
    05  PRESENTATION-MODE       PIC  X.
    05  BILL-OR-FREE            PIC  X.
    05  DATE-BEGUN.
        10  MONTH-BEGUN         PIC  X(2).
        10  YEAR-BEGUN          PIC  X(2).
    05  DATE-BEGUN-DIGIT  REDEFINES DATE-BEGUN PIC  X  OCCURS 4.
    05  ACD                     PIC  X.
```

Figure 18a. Checker program (2 of 13)

```
 01  LOCATION-CODES     VALUE 'KPFENMXZWTSLHGBYVQDACJOR'.
     05  LOCATION-CODE      PIC  X     OCCURS 24 INDEXED I-LOC.

 01  GRADE-CODES        VALUE 'NSABCDFWITCUZ'.
     05  GRADE-CODE         PIC  X     OCCURS 13 INDEXED I-GRADE.

 01  DIVISION-CODES     VALUE 'LNCWPDRACFGHIJKQSTXY'.
     05  DIVISION-CODE      PIC  X     OCCURS 20 INDEXED I-DIV.

 01  RECORD-TYPE-CODES  VALUE '47465410524035'.
     05  RECORD-TYPE-CODE   PIC  X(2)  OCCURS 7 INDEXED I-RT.

 01  SCHOOL-CODES       VALUE '60121361094611260293'.
     05  SCHOOL-CODE        PIC  X(4)  OCCURS 5 INDEXED I-SCHOOL.

 01  X-LOCATION-CODES   VALUE 'ENGLNETHGERMFRANCANASWEDSCCTJAPASWI
-                             'TAUSTBELGITAL'.
     05  X-LOCATION-CODE    PIC  X(4)  OCCURS 12 INDEXED I-XLOC.

 01  Z-LOCATION-CODES   VALUE 'RMD LSC BOT PAA LGC '.
     05  Z-LOCATION-CODE    PIC  X(4)  OCCURS 5 INDEXED I-ZLOC.

 01  KGN-BLDGS          VALUE '202201003211212210001002004005021O2
-                             '202302403403503304204304505205795990
-                             '60064965966'.
     05  KGN-BLDG           PIC  X(3)  OCCURS 27 INDEXED I-BLDG.

 01  ERROR-LINE  VALUE IS ALL SPACE.
     05  ERR-CONTROL                  PIC  X.
     05  ERROR-FIELDS.
         10  FILLER                   PIC  X(22).
         10  ERR-DEPT                 PIC  X(3).
         10  ERR-SCHOOL.
             15  ERR-BLDG             PIC  X(3).
             15  ERR-FLOOR            PIC  X.
         10  ERR-RECORD-TYPE          PIC  X(2).
         10  ERR-LOCATION             PIC  X.
         10  ERR-DIVISION             PIC  X.
         10  ERR-TITLE                PIC  X(20).
         10  FILLER                   PIC  X(4).
         10  ERR-PROJECT              PIC  X(3).
         10  ERR-COURSE-HOURS         PIC  X(4).
         10  FILLER                   PIC  X.
         10  ERR-MONTH-COMPLETED      PIC  X(2).
         10  ERR-YEAR-COMPLETED       PIC  X(2).
         10  FILLER                   PIC  X(6).
         10  ERR-MONTH-BEGUN          PIC  X(2).
         10  ERR-YEAR-BEGUN           PIC  X(2).
         10  FILLER                   PIC  X.
     05  ERR  REDEFINES ERROR-FIELDS  PIC  X  OCCURS 80.

 01  DATA-LINE.
     05  CARRIAGE-CONTROL             PIC  X  VALUE IS SPACE.
     05  DATA-FIELDS                  PIC  X(80).
     05  LINE-CHARACTER  REDEFINES DATA-FIELDS  PIC  X  OCCURS 80.
```

Figure 18a. Checker program (3 of 13)

```
 PROCEDURE DIVISION.
     MOVE CURRENT-DATE TO THIS-YEAR
*            TWO DIGITS RIGHT-JUSTIFIED
     OPEN INPUT ENROLLMENTS
     OPEN OUTPUT LISTING
     MOVE 'F' TO EOF-IND
     MOVE 'F' TO GRADE-CHECK-SWITCH
     READ ENROLLMENTS INTO CARD AT END
         MOVE 'N' TO EOF-IND.
     PERFORM 100-MAIN-PROCESS UNTIL EOF-IND = 'N'
     CLOSE ENROLLMENTS
     CLOSE LISTING
     STOP RUN.

 100-MAIN-PROCESS.
     IF INIT1 = '$'
         MOVE 'N' TO GRADE-CHECK-SWITCH
         WRITE LINE-OUT FROM HEADER4
         WRITE LINE-OUT FROM HEADER5
         WRITE LINE-OUT FROM HEADER4
         MOVE 59 TO COUNTER
     ELSE
         IF INIT1 = SPACE OR INIT1 IS NOT ALPHABETIC
             MOVE SPLAT TO ERR (1).
     IF INIT1 NOT = '$'
         PERFORM 200-MAIN-PROCESS-CONTD
         PERFORM 201-MAIN-PROCESS-CONTD
         PERFORM 202-MAIN-PROCESS-CONTD
         PERFORM 203-MAIN-PROCESS-CONTD
         PERFORM 204-MAIN-PROCESS-CONTD.
     IF COUNTER > 58
**       NEW-PAGE
         MOVE 4 TO COUNTER
         WRITE LINE-OUT FROM HEADER
         WRITE LINE-OUT FROM HEADER2
         WRITE LINE-OUT FROM HEADER3
         WRITE LINE-OUT FROM HEADER3.
**   PRINT
     MOVE CARD TO DATA-FIELDS
     WRITE LINE-OUT FROM DATA-LINE
     ADD 1 TO COUNTER
     IF ERROR-LINE NOT = SPACE
         PERFORM 205-BUILD-UNDERSCORE-LINE VARYING I FROM 1 BY 1
                 UNTIL I > 80
**       OVERPRINT
         MOVE '+' TO CARRIAGE-CONTROL
         WRITE LINE-OUT FROM DATA-LINE
         MOVE SPACE TO CARRIAGE-CONTROL
         WRITE LINE-OUT FROM ERROR-LINE
         ADD 1 TO COUNTER.
     READ ENROLLMENTS INTO CARD AT END
         MOVE 'N' TO EOF-IND.
 205-BUILD-UNDERSCORE-LINE.
     IF ERR (I) = SPACE
         MOVE SPACE TO LINE-CHARACTER (I)
     ELSE
         MOVE '|' TO LINE-CHARACTER (I).
```

Figure 18a. Checker program (4 of 13)

```
 200-MAIN-PROCESS-CONTD.
     IF INIT2 IS NOT ALPHABETIC
         MOVE SPLAT TO ERR (2).
     EXAMINE NAME TALLYING LEADING SPACES
     IF TALLY NOT = ZERO
         ADD 2 I GIVING J
         PERFORM 300-POINTER-INSERT VARYING I FROM 3 BY 1
                 UNTIL I > J.
     PERFORM 301-NAME-CHECK VARYING I FROM 1 BY 1 UNTIL I > 13
**   FIRST-CHARACTER-CHECK
     MOVE SERIAL-NO-CHARACTER (1) TO IT
     IF IT IS NUMERIC OR IT = 'W'
         NEXT SENTENCE
     ELSE
         MOVE SPLAT TO ERR (17).
     PERFORM 302-SERIAL-NO-CHECK VARYING I FROM 2 BY 1 UNTIL I > 6
**   DEPARTMENT-CHECK
     PERFORM 303-DEPT-CHECK VARYING I FROM 1 BY 1 UNTIL I > 3.

 300-POINTER-INSERT.
     MOVE SPLAT TO ERR (1).

 301-NAME-CHECK.
     MOVE NAME-CHARACTER (I) TO IT
     IF IT IS ALPHABETIC  OR  IT = SPACE OR QUOTE OR '-' OR '.'
         NEXT SENTENCE
     ELSE
         ADD  2  I  GIVING  J
         MOVE SPLAT TO ERR (J).

 302-SERIAL-NO-CHECK.
     IF SERIAL-NO-CHARACTER (I) IS NOT NUMERIC
         COMPUTE  J  =  I  +  17
         MOVE SPLAT TO ERR (J).

 303-DEPT-CHECK.
     MOVE DEPT-CHARACTER (I) TO IT
     IF IT IS NUMERIC  OR  IT IS ALPHABETIC AND IT IS NOT = SPACE
         NEXT SENTENCE
     ELSE
         ADD  22  I  GIVING  J
         MOVE SPLAT TO ERR (J).
```

Figure 18a. Checker program (5 of 13)

```
201-MAIN-PROCESS-CONTD.
    IF RECORD-TYPE = '10'
        PERFORM 304-SCHOOL-CHECK
    ELSE
        PERFORM 305-RECORD-TYPE-CHECK
        PERFORM 306-LOC-AND-BLDG-CHECK
        IF FLOOR IS NUMERIC OR FLOOR IS ALPHABETIC
            NEXT SENTENCE
        ELSE
            MOVE SPLAT TO ERR-FLOOR.

304-SCHOOL-CHECK.
    SET I-SCHOOL TO 1
    SEARCH SCHOOL-CODE
        AT END
            MOVE SPLAT TO ERR-SCHOOL
        WHEN SCHOOL = SCHOOL-CODE (I-SCHOOL)
            NEXT SENTENCE.

305-RECORD-TYPE-CHECK.
    SET I-RT TO 1
    SEARCH RECORD-TYPE-CODE
        AT END
            MOVE SPLAT TO ERR-RECORD-TYPE
        WHEN RECORD-TYPE = RECORD-TYPE-CODE (I-RT)
            NEXT SENTENCE.

306-LOC-AND-BLDG-CHECK.
    IF LOCATION = 'K'
        PERFORM 400-K-BLDG-CHECK
    ELSE IF LOCATION = 'X'
        PERFORM 401-X-LOC-CHECK
    ELSE IF LOCATION = 'Z'
        PERFORM 402-Z-LOC-CHECK
    ELSE
        PERFORM 403-AN-BLDG-CHECK.
```

Figure 18a. Checker program (6 of 13)

```
400-K-BLDG-CHECK.
    SET I-BLDG TO 1
    SEARCH KGN-BLDG
        AT END
            MOVE SPLAT TO ERR-BLDG
        WHEN BLDG = KGN-BLDG (I-BLDG)
            NEXT SENTENCE.

401-X-LOC-CHECK.
    SET I-XLOC TO 1
    SEARCH X-LOCATION-CODE
        AT END
            MOVE SPLAT TO ERR-SCHOOL
        WHEN OTHER-LOCATION = X-LOCATION-CODE (I-XLOC)
            NEXT SENTENCE.

402-Z-LOC-CHECK.
    SET I-ZLOC TO 1
    SEARCH Z-LOCATION-CODE
        AT END
            MOVE SPLAT TO ERR-SCHOOL
        WHEN OTHER-LOCATION = Z-LOCATION-CODE (I-ZLOC)
            NEXT SENTENCE.

403-AN-BLDG-CHECK.
    IF BLDG IS ALPHABETIC  OR  BLDG IS NUMERIC
        NEXT SENTENCE
    ELSE
        MOVE SPLAT TO ERR-BLDG.
```

Figure 18a. Checker program (7 of 13)

```
202-MAIN-PROCESS-CONTD.
    SET I-LOC TO 1
    SEARCH LOCATION-CODE
        AT END
            MOVE SPLAT TO ERR-LOCATION
        WHEN LOCATION = LOCATION-CODE (I-LOC)
            NEXT SENTENCE.
    SEARCH DIVISION-CODE
        AT END
            MOVE SPLAT TO ERR-DIVISION
        WHEN DIV = DIVISION-CODE (I-DIV)
            NEXT SENTENCE.
    EXAMINE TITLE TALLYING LEADING SPACES
    IF TALLY IS NOT ZERO
        COMPUTE  J  =  TALLY  +  34
        PERFORM 300-POINTER-INSERT VARYING I FROM 1 BY 1
                UNTIL I > J.
    IF COURSE-NO (1) IS NUMERIC
        NEXT SENTENCE
    ELSE IF COURSE-NO (1) = 'S'  OR  'F'  OR  'D'
        NEXT SENTENCE
    ELSE
        MOVE SPLAT TO ERR (54).
    PERFORM 307-COURSE-NO-CHECK VARYING I FROM 2 BY 1
            UNTIL I > 4.

307-COURSE-NO-CHECK.
    IF COURSE-NO (I) IS NOT NUMERIC
        COMPUTE  J  =  I  +  53
        MOVE SPLAT TO ERR (J).
    IF RECORD-TYPE = '47'
        IF PROJECT = 'V30'  OR  'V31'
            NEXT SENTENCE
        ELSE
            MOVE SPLAT TO ERR-PROJECT
    ELSE
        IF RECORD-TYPE = '10'
            IF PROJECT = 'GWS'
                NEXT SENTENCE
            ELSE
                MOVE SPLAT TO ERR-PROJECT
        ELSE
            IF PROJECT = 'V32'  OR  'V33'
                NEXT SENTENCE
            ELSE
                MOVE SPLAT TO ERR-PROJECT.
    PERFORM 404-COURSE-HOUR-CHECK VARYING I FROM 1 BY 1
            UNTIL I > 4.

404-COURSE-HOUR-CHECK.
    IF COURSE-HOUR-DIGIT (I) IS NOT NUMERIC
        COMPUTE  J  =  I  +  60
        MOVE SPLAT TO ERR (J).
```

Figure 18a. Checker program (8 of 13)

```
 203-MAIN-PROCESS-CONTD.
**   COURSE-HOUR-VALUE-CHECK
     IF COURSE-HOUR-VALUE > 230
         MOVE SPLAT TO ERR-COURSE-HOURS.
**   GRADE-CHECK-SWITCH
     IF GRADE-CHECK-SWITCH = 'F'
         PERFORM 308-COMPLETION-GRADE-CHECK
     ELSE
**       ENROLLMENT-GRADE-CHECK
         IF GRADE NOT = SPACE
             MOVE SPLAT TO ERR (65).
**   DATE CHECK
     PERFORM 309-DATE-DIGIT-CHECK VARYING I FROM 1 BY 1
             UNTIL I > 4
**   MONTH-CHECK
     IF MONTH-COMPLETED > '12'
         MOVE SPLAT TO ERR-MONTH-COMPLETED.
     IF TEACHING-LOCATION NOT = 'K'
         MOVE SPLAT TO ERR (70).
     IF TEACHING-LOCATION NOT = 'L'
         MOVE SPLAT TO ERR (71).
     IF RECORD-TYPE = '10'
         IF CREDITS IS NOT NUMERIC
             MOVE SPLAT TO ERR (72)
         ELSE
             NEXT SENTENCE
     ELSE
         IF CREDITS IS NOT NUMERIC AND CREDITS NOT = SPACE
             MOVE SPLAT TO ERR (72).
     IF FLAG IS = ' '  OR  '1'  OR  '2'
         NEXT SENTENCE
     ELSE
         MOVE SPLAT TO ERR (73).
     IF PRESENTATION-MODE IS NOT NUMERIC
             OR  PRESENTATION-MODE = ZERO
         MOVE SPLAT TO ERR (74).
     IF BILL-OR-FREE IS = ' '  OR  'B'  OR  'F'
         NEXT SENTENCE
     ELSE
         MOVE SPLAT TO ERR (75).

 308-COMPLETION-GRADE-CHECK.
     SET I-GRADE TO 1
     SEARCH GRADE-CODE
         AT END
             MOVE SPLAT TO ERR (65)
         WHEN GRADE = GRADE-CODE (I-GRADE)
             NEXT SENTENCE.

 309-DATE-DIGIT-CHECK.
     IF DATE-COMPLETED-DIGIT (I) IS NOT NUMERIC
         COMPUTE  J  =  I  +  65
         MOVE SPLAT TO ERR (J).
```

Figure 18a. Checker program (9 of 13)

```
204-MAIN-PROCESS-CONTD.
    IF DATE-BEGUN = SPACE
        NEXT SENTENCE
    ELSE
        PERFORM 310-DATE-BEGUN-CHECK.
    IF ACD = ' '  OR  'C'  OR  'D'  OR  'A'
        NEXT SENTENCE
    ELSE
        MOVE SPLAT TO ERR (80).

310-DATE-BEGUN-CHECK.
    PERFORM 405-DATE-BEGUN-DIGIT-CHECK VARYING I FROM 1 BY 1
            UNTIL I > 4
    IF MONTH-BEGUN > '12'
        MOVE SPLAT TO ERR-MONTH-BEGUN.
    IF YEAR-BEGUN > YEAR-COMPLETED
        MOVE SPLAT TO ERR-YEAR-BEGUN.

405-DATE-BEGUN-DIGIT-CHECK.
    IF DATE-BEGUN-DIGIT (I) IS NOT NUMERIC
        COMPUTE  J  =  I  +  75
        MOVE SPLAT TO ERR (J).
```

Figure 18a. Checker program (10 of 13)

```
                                 CRCSS-REFERENCE DICTICNARY

DATA NAMES                      CEFN      REFERENCE

ACC                             000091    000430
BILL-OR-FREE                    000086    000407
BLCG                            000064    000291    000312
CARC                            000053    000156    000187    000199
CARC-IN                         000022    000156    000199
CARRIAGE-CCNTRCL                000145    000194    000196
COUNTER                         000032    000169    000179    000181    000189    000198
COURSE-HCUR-DIGIT               000073    000367
COURSE-HOUR-VALUE               000072    000373
COURSE-NO                       000070    000334    000336    000344
CREDITS                         000083    000393    000398
DATA-FIELDS                     000146    000187
DATA-LINE                       000144    000188    000195
DATE-BEGUN                      000087    000426
DATE-BEGUN-DIGIT                000090    000444
DATE-COMPLETED                  000076
DATE-COMPLETED-DIGIT            000079    000421
DEPT-CHARACTER                  000060    000243
DIV                             000067    000326
DIVISION-CCDE                   000100    000326
DIVISION-CCDES                  000099
ENRCLLMENTS                     000014    000152    000156    000159    000199
EOF-IND                         000030    000154    000157    000158    000200
ERR                             000142    000172    000202    000209    000221    000227    000235    000240    000248    000339    000346
                                          000369    000381    000389    000391    000394    000399    000403    000406    000410    000416
                                          000423    000433    000446
ERR-BLDG                        000126    000291    000315
ERR-CONTRCL                     000121
ERR-COURSE-HCURS                000134    000374
ERR-CEPT                        000124
ERR-CIVISICN                    000130    000326
ERR-FLCOR                       000127    000259
ERR-LOCATICN                    000129    000321
ERR-MONTH-BEGUN                 000139    000439
ERR-MONTH-CCMPLETEC             000136    000387
ERR-PROJECT                     000133    000351    000357    000362
ERR-RECORC-TYPE                 000128    000273
ERR-SCHOCL                      000125    000265    000299    000307
ERR-TITLE                       000131
ERR-YEAR-BEGUN                  000140    000441
ERR-YEAR-COMPLETED              000137
ERRCR-FIELDS                    000122
ERRCR-LINE                      000120    000190    000197
FLAG                            000084    000400
FLCCR                           000065    000256
GRACE                           000075    000380    000416
GRADE-CHECK-SWITCH              000031    000155    000165    000376
GRADE-CCDE                      000097    000416
GRACE-CODES                     000096
HEACER                          000038    000182
HEACER2                         000041    000183
HEACER3                         000044    000184    000185
HEACER4                         000047    000166    000168
HEACER5                         000049    000167
I                               000034    000191    000202    000203    000205    000212    000213    000215    000222    000224    000230
                                          000234    000238    000239    000243    000247    000332    000340    000344    000345    000363
                                          000367    000368    000383    000421    000422    000436    000444    000445
I-BLDG                          000118    000288    000291
I-CIV                           000100    000326
I-GRADE                         000097    000413    000416
I-LCC                           000094    000318    000321
I-RT                            000103    000270    000273
I-SCHOOL                        000106    000262    000265
I-XLCC                          000110    000296    000299
I-ZLCC                          000113    000304    000307
INIT1                           000054    000164    000171    000173
```

Figure 18a. Checker program (11 of 13)

```
INIT2                       000055    000208
IT                          000036    000217    000218    000230    000231    000243    000244
J                           000035    000212    000213    000234    000235    000239    000240    000247    000248    000331    000332
                                      000345    000346    000368    000369    000422    000423    000445    000446
KGN-BLDG                    000118    000291
KGN-BLDGS                   000115
LINE-CHARACTER              000147    000203    000205
LINE-OUT                    000027    000166    000167    000168    000182    000183    000184    000185    000188    000195    000197
LISTING                     000015    000153    000160    000166    000167    000168    000182    000183    000184    000185    000188
                                      000195    000197
LOCATION                    000066    000278    000280    000282    000321
LOCATION-CODE               000094    000321
LOCATION-CODES              000093
MONTH-BEGUN                 000088    000438
MONTH-COMPLETED             000077    000386
NAME                        000056    000210
NAME-CHARACTER              000057    000230
OTHER-LOCATION              000062    000299    000307
PRESENTATION-MODE           000085    000404
PRCJECT                     000071    000348    000354    000359
RECORD-TYPE                 000068    000251    000273    000347    000353    000392
RECCRD-TYPE-CODE            000103    000273
RECCRD-TYPE-CODES           000102
SCHCOL                      000061    000265
SCHCOL-CODE                 000106    000265
SCHCOL-CODES                000105
SERIAL-NO                   000058
SERIAL-NO-CHARACTER         000059    000217    000238
SPLAT                       000037    000172    000209    000221    000227    000235    000240    000248    000259    000265    000273
                                      000291    000299    000307    000315    000321    000326    000339    000346    000351    000357
                                      000362    000369    000374    000381    000387    000389    000391    000394    000399    000403
                                      000406    000410    000416    000423    000433    000439    000441    000446
TEACHING-DIVISION           000082
TEACHING-LCCATICN           000081    000388    000390
THIS-YEAR                   000033    000150
TITLE                       000069    000329
X-LCCATICN-CCDE             000110    000299
X-LCCATION-CODES            000108
YEAR-BEGUN                  000089    000440
YEAR-COMPLETED              000078    000440
Z-LCCATICN-CCDE             000113    000307
Z-LCCATION-CODES            000112
```

Figure 18a. Checker program (12 of 13)

```
PROCEDURE NAMES                    DEFN      REFERENCE


100-MAIN-PROCESS                   000163    000158
200-MAIN-PROCESS-CONTD             000207    000174
201-MAIN-PROCESS-CONTD             000250    000175
202-MAIN-PROCESS-CONTD             000317    000176
203-MAIN-PROCESS-CONTD             000371    000177
204-MAIN-PROCESS-CONTD             000425    000178
205-BUILD-UNDERSCORE-LINE          000201    000191
300-POINTER-INSERT                 000226    000213    000332
301-NAME-CHECK                     000229    000215
302-SERIAL-NO-CHECK                000237    000222
303-DEPT-CHECK                     000242    000224
304-SCHOOL-CHECK                   000261    000252
305-RECORD-TYPE-CHECK              000269    000254
306-LOC-AND-BLDG-CHECK             000277    000255
307-COURSE-NO-CHECK                000343    000340
308-COMPLETION-GRADE-CHECK         000412    000377
309-DATE-DIGIT-CHECK               000420    000383
310-DATE-BEGUN-CHECK               000435    000429
400-K-BLDG-CHECK                   000287    000279
401-X-LOC-CHECK                    000295    000281
402-Z-LOC-CHECK                    000303    000283
403-AN-BLDG-CHECK                  000311    000285
404-COURSE-HOUR-CHECK              000366    000363
405-DATE-BEGUN-DIGIT-CHECK         000443    000436
```

Figure 18a. Checker program (13 of 13)

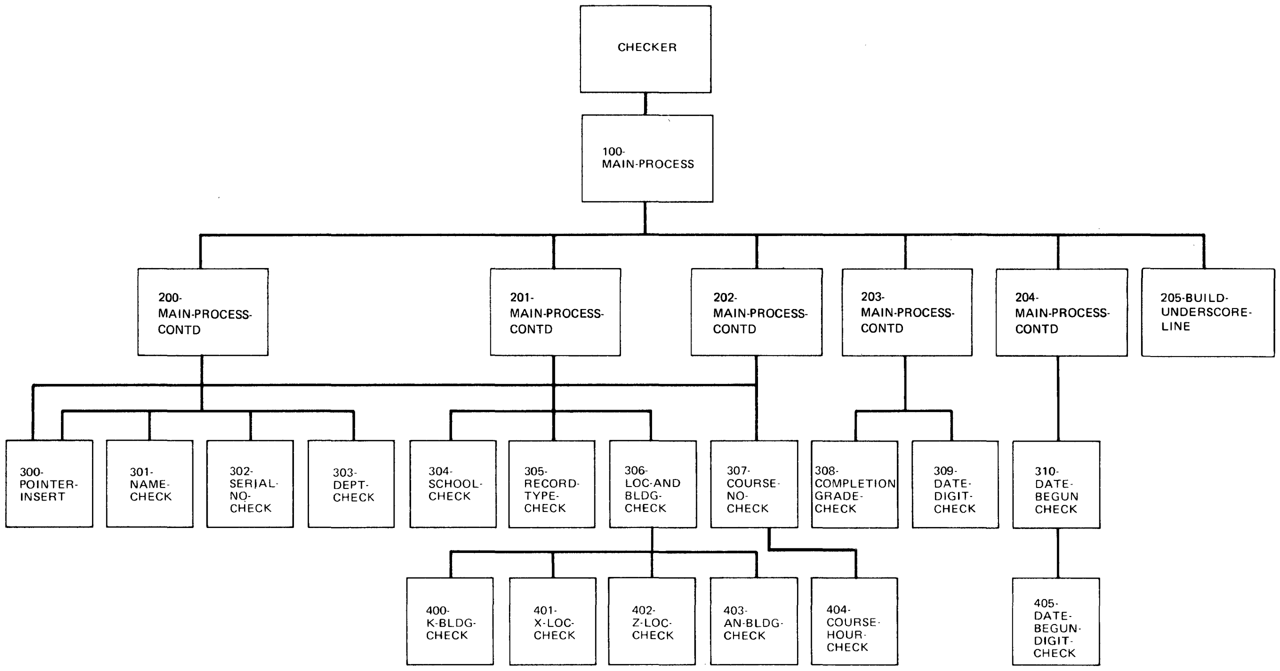


Figure 18b. Checker program structure chart corresponding to COBOL procedure names cross-reference dictionary

This page intentionally left blank

Appendix B: Case Study Solution

This appendix contains the error list (Figure 19) and detail and summary inspection reports (Figure 20 and 21). Note that the error list (the average inspection team finds 13 or 14 of these errors) could be expanded to include violations of installation standards. For example, the following might be an additional error that could be listed:

MN/W/MIN – Line 3: SPLAT is not a meaningful label.

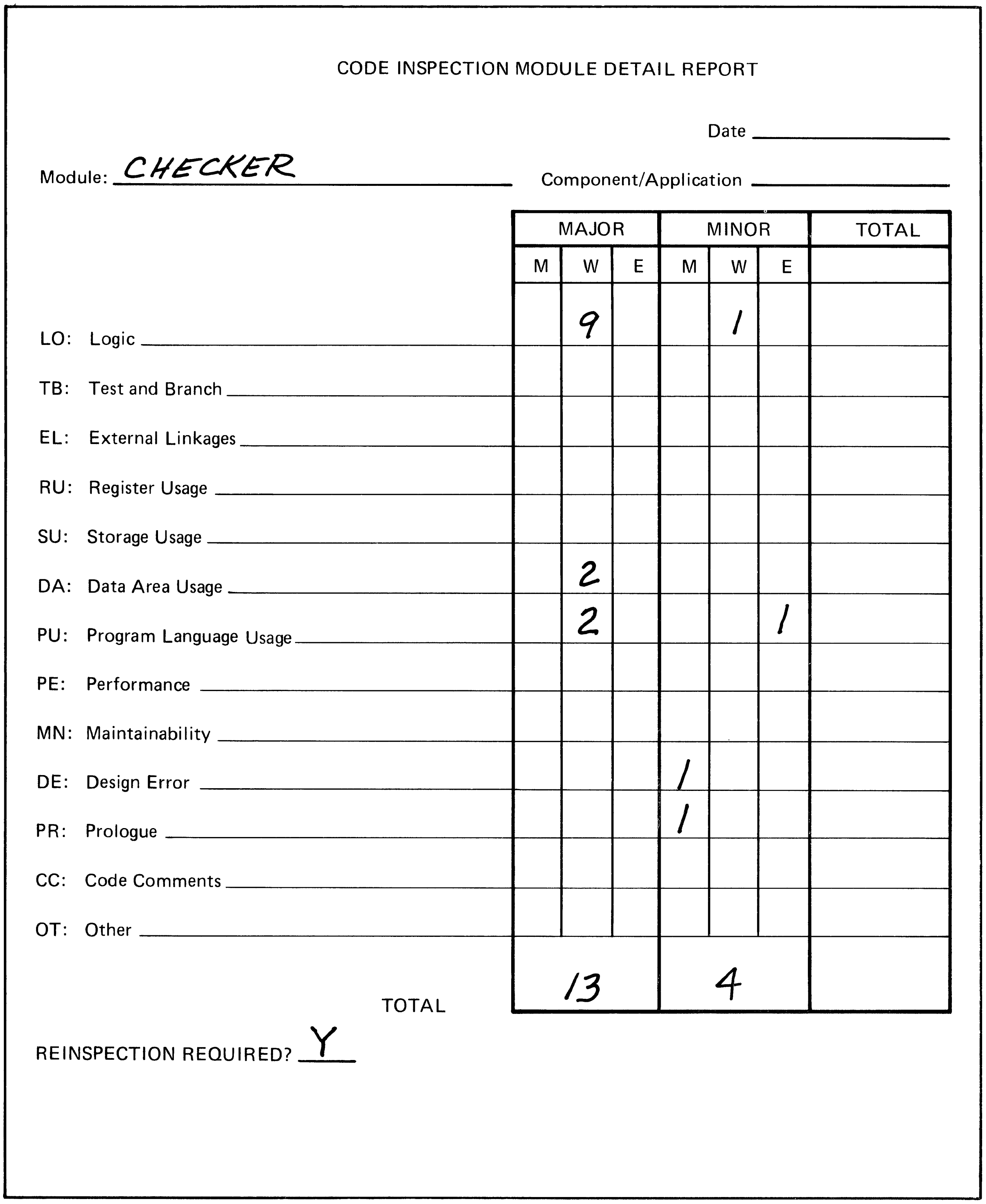
---

1. PR/M/MIN – Line 3: The statement of the prologue in the REMARKS section needs expansion.
2. DA/W/MAJ – Line 129: ERR-RECORD-TYPE is out of sequence.
3. PU/W/MAJ – Line 151: The wrong bytes of an eight-byte field (current date) are moved into the two-byte field (this year).
4. LO/M/MAJ – Line 179: COUNTER is not initialized at the start of the program. Thus, the processing at line 179 is not guaranteed to be correct.
5. LO/M/MAJ – Line 198: ERROR-LINE is never housekept.
6. DA/W/MAJ – Line 205: The underscore symbol should be used instead of the vertical bar.
7. LO/W/MAJ – Line 212: While counting the number of leading spaces in NAME, the wrong variable (I) is used to calculate "J".
8. LO/W/MAJ – Line 215: NAME-CHECK is PERFORMED one less time than required.
9. PU/E/MIN – Line 231: In NAME-CHECK, the check for SPACE is redundant.
10. DE/W/MAJ – Line 231: The design should allow for the occurrence of a period in a last name.
11. LO/W/MAJ – Line 239: In SERIAL-NO-CHECK, the error flag is set into the wrong column.
12: LO/W/MIN – Line 256: This logic is executed for locations "X" and "Z". It should not be.
13: LO/W/MAJ – Line 312: The code does not match the specifications. Any combination of alpha, blanks, and numbers should be allowed.
14. LO/W/MAJ – Line 332: If there are leading spaces, one too many columns are error-flagged.
15. LO/M/MAJ – Line 353: Record type 46 logic is missing.
16. LO/W/MAJ – Lines 347 through 364 should follow line 341.

---

17. PU/W/MAJ – Line 373: The conversion of the constant by COBOL does not occur as the user expects.

Total rework time is estimated to be 8.0 hours.



CODE INSPECTION MODULE DETAIL REPORT

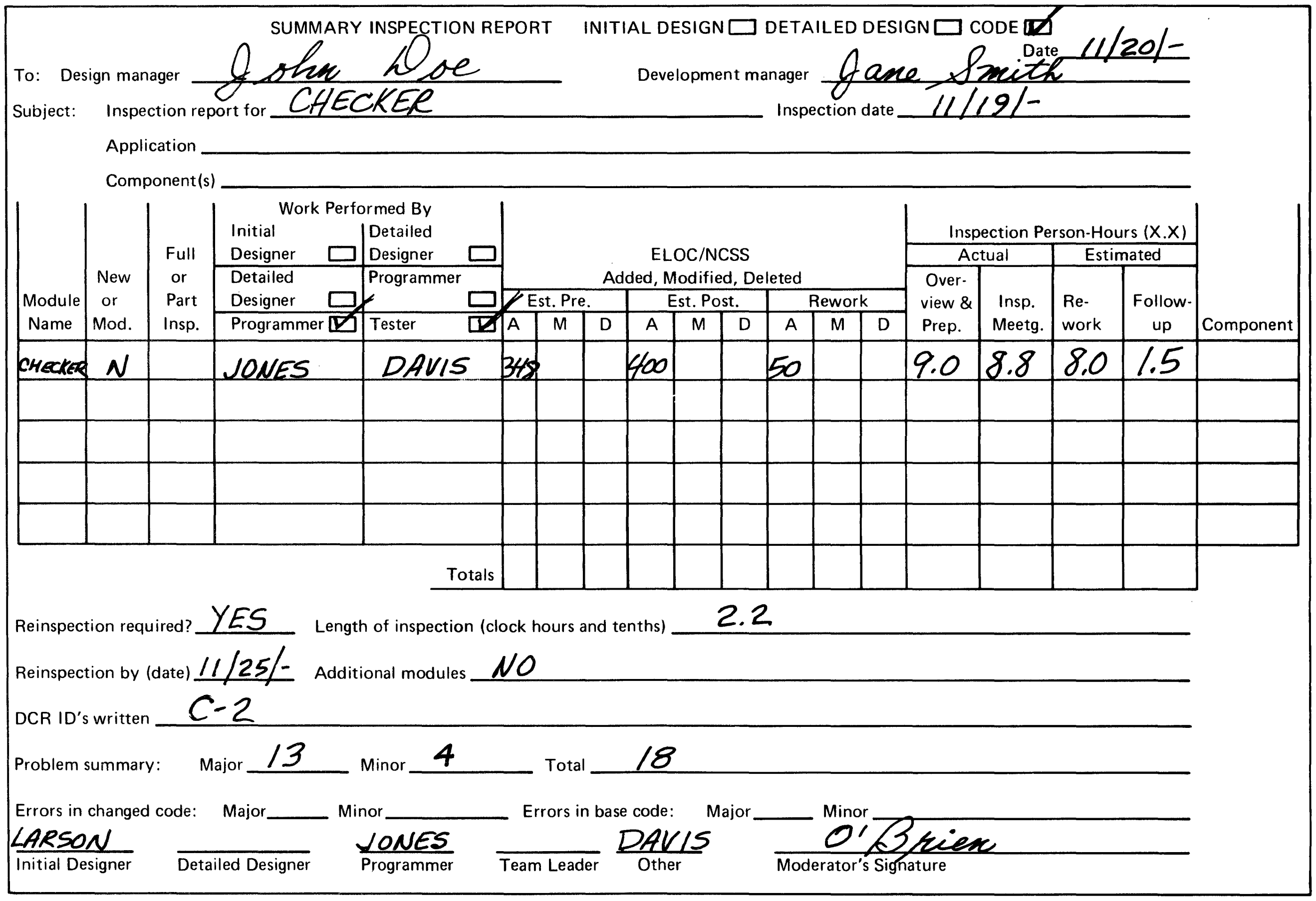
Date ______

Module: CHECKER   Component/Application ______

	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
LO: Logic		9			1		
TB: Test and Branch							
EL: External Linkages							
RU: Register Usage							
SU: Storage Usage							
DA: Data Area Usage		2					
PU: Program Language Usage		2				1	
PE: Performance							
MN: Maintainability							
DE: Design Error				1			
PR: Prologue				1			
CC: Code Comments							
OT: Other							
TOTAL	13			4			

REINSPECTION REQUIRED? Y

Figure 20. Checker program code inspection module detail report



SUMMARY INSPECTION REPORT INITIAL DESIGN ☐ DETAILED DESIGN ☐ CODE ☑

Date 11/20/-

To: Design manager John Doe Development manager Jane Smith

Subject: Inspection report for CHECKER Inspection date 11/19/-

Application

Component(s)

			Work Performed By		ELOC/NCSS Added, Modified, Deleted									Inspection Person-Hours (X.X)				
														Actual		Estimated		
Module Name	New or Mod.	Full or Part Insp.	Initial Designer ☐ / Detailed Designer ☐ / Programmer ☑	Detailed Designer ☐ / Programmer ☐ / Tester ☑	Est. Pre.			Est. Post.			Rework			Over-view & Prep.	Insp. Meetg.	Re-work	Follow-up	Component
					A	M	D	A	M	D	A	M	D					
CHECKER	N		JONES	DAVIS	348			400			50			9.0	8.8	8.0	1.5	
				Totals														

Reinspection required? YES Length of inspection (clock hours and tenths) 2.2

Reinspection by (date) 11/25/- Additional modules NO

DCR ID's written C-2

Problem summary: Major 13 Minor 4 Total 18

Errors in changed code: Major___ Minor___ Errors in base code: Major___ Minor___

LARSON		JONES		DAVIS	O'Brien
Initial Designer	Detailed Designer	Programmer	Team Leader	Other	Moderator's Signature

Figure 21. Checker program summary inspection report

Appendix C: Exit Criteria

Exit criteria are requirements to be met before a phase is considered complete. One objective of inspections is to determine whether these installation-established requirements have, in fact, been met. The initial design, detailed design, and code inspection exit criteria that follow are samples developed within IBM for a system control programming environment. Each installation should develop its own criteria to meet the needs of its specific application development environment.

Initial Design Exit Criteria

Initial design specifications are divided into two sections:

1. Specifications that describe the new functions as it applies to all affected areas of a component and system (Group I specifications)
2. Individual specifications for each module affected by the new function (Group II specifications)

The exit criteria for Group I and Group II specifications follow.

Group I Specifications

1. Completed external specifications, that is, those that define a function from a user or outside viewpoint.
2. Internal specifications containing the following for all new or changed functions for each affected component.
   a. *Component Description*
   An overall description of the new function – *what* is to be done.
   b. *Design Rationale*
   Statement of design requirements – *why* it is to be done. Include performance and storage requirements.
   c. *Control Flow*
   Diagram showing transfer of control between divisions of the component.
   d. *Dependencies*
   • *Internal*
   Dependencies on other components within the system.
   • *External*
   Dependencies outside the system, such as hardware.
   e. *Data Areas*
   Names of data areas that are required by this function.
   f. *Internal Macros*
   Names of macros contained within the system that are required by this function.
   g. *Linkages*
   • *Intercomponent*
   List of linkages between components required by this function.
   • *Intracomponent*
   List of linkages between major divisions of a component required by this function.
3. Internal specifications containing the following for all new or changed functions for each affected major division of a component.
   a. *Major Division Description*
   An overall description of the function as it affects each major division.
   b. *Module List*
   A list of all new or changed modules that are affected by the new function.
   c. *Control Flow*
   A graphic representation of the control flow (linkages) between the modules within the division that are affected by the new function.
   d. *Packaging Requirements*
   Any paging or link-editing required by the new function must be defined and listed.

Group II Specifications

The Group II portion of the initial design specifications is a module description. Its purpose is to provide a functional description design statement that will be used as the source for detailed design. The Group II specifications are not intended for use as coding specifications. The following are required for each module affected by a new function:

1. Functional description statements must be to a level of detail in the approximate range of 15-25 source language lines of code per design statement. (A design statement can be expressed in any design language, for example, HIPO, flowcharts, pseudo code, etc.)
2. If a specific sequence of processing or control flow in a module is required, it must be stated.
3. All required data area changes/additions must be specified to the field description level (field names and storage attributes will be defined during detailed design).
4. All required linkage changes/additions must be defined:
   a. For macros internal to the component, definition must be to the macro description

level. (Specific macro keywords and parameters will be defined during detailed design.)

b. For control flow between modules, all linkage information requirements must be defined. (Specific return codes, parameter formats, etc., are not required; they will be defined during detailed design.)

5. All new/changed module attributes must be specified:
   a. Reenterable, reusable
   b. Protection key
   c. Fixed or pageable
   d. Supervisor or problem state
   e. Enabled or disabled state
6. All new/changed performance and storage criteria must be specified. These include storage size and path length requirements.

Detailed Design Exit Criteria

1. Completed specifications represent design to the approximate range of three to ten source language lines of code per design statement. (A design statement can be expressed in any design language, for example, HIPO, flowcharts, pseudo code, etc.)
2. Design specifications must be structured (all applicable structured programming rules are to be followed).
3. For design specifications covering modified modules, all new/changed statements should be flagged with release identification.
4. References to data areas should be by field name.
5. Macro invocation should specify all required parameters and associated values. Those parameters that are not specified are assumed to be default values.
6. Linkages to external routines/modules must be defined. Parameter values passed and return codes expected are specified.
7. Messages issued by modules must be defined. Identification numbers and message texts are specified. Entry and exit points from modules must be defined.
8. All new/changed data areas must be documented in respective design documentation.
9. Any design changes to initial design since initial design exit must be included in the current design specifications; that is, initial and detailed design materials must match.

Code Inspection Exit Criteria

1. Complete and up-to-date module prologue.
2. Error-free compile (that is, no higher-level messages than warning messages).
3. Program listing containing:
   a. Compile phase output with cross-reference listing
   b. Assembler language (BAL) phase output with cross-reference listing
   c. Data area maps
   d. New/changed lines of code flagged with programmer identification code
4. Any design changes since detailed design exit must be included in current design specifications; that is, design and code materials must match.

Appendix D: Checklists

Checklists provide the inspection team with a set of prompters to help uncover high-cost, high-occurrence errors in the work product being inspected, and are based on an analysis of errors found during prior inspections. The inspectors review the appropriate checklists during the preparation period and also use them during the inspection meeting to help them focus on the error types listed.

The initial design, detailed design, and code inspection checklists that follow in this appendix are samples developed in a system control programming environment. Each installation will develop its own checklists to reflect its own error statistics and requirements. Note that the errors specified in the sample checklists are associated with a two-character alphabetic error type code that corresponds to the codes used on the inspection report forms (see Appendix E).

Initial Design Inspection Checklists

The Group I and Group II checklists are used with Group I and Group II initial design specifications, respectively (see Appendix C).

Group I Checklist

1. Is the design consistent with the program objectives or requirements? ES
2. Is the external specification for all new/changed externals correct and in sufficient detail? ES
3. Does the external specification properly address human factors considerations (for example, eliminate redundant user options, create meaningful and *usable* user macros, keywords, etc.)? ST
4. Do the Group I specifications give an accurate and complete: SC
   a. Overall description of the new function?
   b. Graphic description of the new function, showing control flow between linking modules, both within and outside the function (component and system)?
   c. List of external dependencies?
   d. List of packaging requirements (modules that must be link-edited or paged together)?
   e. List of control blocks and a description of new functions for which they are used?
   f. List of component macros required by new function?

Group II Checklist

1. Do the Group II specifications give a complete and accurate description of the overall function of the module?
   a. Is all new/changed design specified at the functional description level? SC
   b. Is the design understandable as stated? Can detailed design be implemented from this level of detail? SC
   c. Does the functional description cover all known possible cases (for example, no missing logic)? Watch in particular for exception cases. LO
   d. Does the functional description cover abnormal conditions (for example, what would happen if there is an ABEND)? ST
   e. For design being implemented in more than one system, does the new function fit correctly into all systems (each system may present unique situations)? LR
   f. For changed functions, does the new logic mesh with the old? LO
2. Are all required data definition changes and additions specified and defined to the field description level (field names and storage attributes will be defined during detailed design)? DA
   a. Is the field described correctly?
   b. Have any field definitions been omitted?
3. Are all required external linkage changes/additions specified and defined correctly? LR
   a. Should an external routine be used rather than performing the function internally?
   b. Does the processing module set up (for output) and process (on input) all required passed parameters? Correctly?
   c. Is all new/changed information flow across external linkages correctly described? Specific return codes, parameter formats, etc., are not required; they will be documented in detailed design.
4. Are new/changed module attributes specified? MA
   a. Reenterable, reusable?
   b. Protection key?
   c. Fixed or pageable?
   d. Supervisor or problem?
   e. Enabled or disabled?
5. Are new/changed performance and storage criteria specified? PE
   a. Path lengths?
   b. Storage size?
   c. Is the function designed optimally for performance and storage?

d. Does the function use existing facilities (logic, control blocks, etc.) where possible? Creation of new facilities should be avoided where existing ones are available.
e. Will execution of the function as designed cause minimum (optimal) paging?

Detailed Design Inspection Checklists

Logic (LO)
- Are all constants defined?
- Are all unique values explicitly tested on input parameters?
- Are values stored after they are calculated?
- Are all defaults checked explicitly; for example, blanks in an input stream?
- If character strings are created, are they complete? Are all delimiters shown?
- If a keyword has many values, are they all checked?
- Are all keywords tested in a macro?
- Are all keyword-related parameters tested in a service routine?
- Are all increment counts properly initialized (0 or 1)?
- After processing a table entry, should any value be decremented or incremented?
- Is provision made for possible processing at logical checkpoints in the program (end-of-file, end-of-volume, etc.)?
- Is all I/O performed on opened files?
- Are routine error conditions adequately provided for (INVALID KEY, ON SIZE ERROR, etc.)?
- Are literals shown where there should be constant data names?
- On comparison of group items, should all fields be compared?
- Is the value of a data item used before the item is initialized?
- Are all data areas shown in design necessary or are some extraneous?

Data Area Usage (DA)
- If design is dependent on building/creating/deleting various data areas, are all designated?
- Should a called macro provide any INCLUDEs for any data areas that the macro expanded code may depend on?
- Does design show explicitly which area to use in a data area, that is, if there are multiple save areas?
- If the program stores into a data area, does it store into the correct field?
- If a value is fetched from a data area, is the correct field fetched?
- Should the data area be boundary-aligned?
- Does a save area have multiple uses? Can conflicts arise?

Test and Branch (TB)
- Are all three conditions tested, that is, greater than, equal to, and less than zero?
- After a linkage, should a return code be tested?
- Is a SORT or a MERGE operation tested for successful completion?
- Are branch legs correct, that is, should YES be NO and NO be YES?

Return Codes/Messages (RM)
- Are messages issued for all error conditions?
- On exits, should a return code be set or a message issued?
- Does the message say what it means?
- Could more information be supplied in the message?
- Do return codes in the design for particular situations match the global definition of the return code as documented?

Register Usage (RU)
- If a specific register is required, is it specified?
- Does any macro expansion use a register already in use without saving the data?
- Is the integrity of all input registers maintained?

More Detail (MD)
- Does the design specify a process ambiguously, or does the process require more than ten instructions?

External Linkages (EL)
- Should a standard linkage be used rather than coding a subroutine inline?
- Is the designated linkage the right one for the function to be performed?
- Is the data area mapped as the receiving module expects it to be?

Standards (ST)
- Are any programming standards for the project in jeopardy of compromise because of the design?

Initial Design Documentation (HL)
- Does the detailed design match the initial design? If not, the initial design documentation could be in error.

Performance (PE)
- Does the design impair the performance of this module to any significant degree?

Code Inspection Checklists

COBOL Checklist

Identification Division

- Does the prose in the REMARKS paragraph function as a complete prologue for the program? PR

Environment Division

- Does each SELECT sentence explicitly define the external (system-dependent) specifications for the file? SU

Data Division – File Section

- Are the file definitions (FDs) in the same order as their respective SELECT sentences in the environment division? DA
- Do the record and data item names conform to their usage? DA
- Does each FD contain comments regarding: DA
  File usage (recording mode), block size, record length, imbedded keys, etc.)?
  Amount of activity (updated how often, used every time program is run, etc.)?
  Interaction with other data items. (Do its records contain objects of OCCURS ... DEPENDING ON clauses? Is the length of its records dependent on such an object elsewhere in the program, etc.?)
- Is the file sorted or merged? EL
- Are statistics kept on file activity in a given run or series of runs? EL
- Is the correct balance struck between specifying complete file attributes in the program and specifying some of them dynamically (such as block size, maximum record length); that is, if a file is designed to be flexible in the given program, is it defined as flexibly as needed? DA

Data Division – Working Storage and Linkage Sections

- Do the data item names conform to their usage? DA
- Does each data item (except for elementary items of obvious usage – subscripts, etc.) contain comments regarding: DA
  Characteristics (fixed- or variable-length, maximum allowable length, etc.)?
  Interaction with other data items? (Does this data item contain or depend on objects of OCCURS ... DEPENDING ON, etc.?)
  Area of use in program? (Is it used only in a certain section, or during a range of paragraphs, etc.?)
- Are all data items with any kind of unifying quality placed together according to a particular scheme? DA
  Usage (arithmetic work areas, work areas for file records, etc.)?
  Commonality of purpose (everything used to process a particular file, etc.)?
  Attributes (message texts, constants, etc.)?
- Are all working storage items that are used as constants designated as such? DA
- Are data items that are required to be in a particular order sequenced correctly? DA
- Is the use of REDEFINE/RENAME in a data description justified and documented in terms of a simplification of data references, rather than reliance on the normal hierarchy of level numbers? SU

Procedure Division

- Are block comments included for major functional areas (for example, paragraph, section, segment)? CC
- Is the module commented on in sufficient detail? CC
- Are comments accurate and meaningful? CC
- Does the code essentially correspond to the outline of the module documented in the remarks paragraph? LO
- Does each paragraph, section, or segment have a homogeneous purpose which justifies and/or necessitates placing all the code together under such a grouping? MN
- Does each performed paragraph or section document the function it accomplishes and the part of the overall logic it represents? CC
- In a segmented program, is it clear why segmentation is necessary? MN
- Does each segment stand alone, or is there heavy dependence on other segments? MN

Format

- Are IFTHENELSE and DO groups aligned properly? MN
- Are nested IFs indented properly? MN
- Are comments accurate and meaningful? MN
- Are meaningful labels used? MN
- Are the clauses of complex verbs (for example, SORT/MERGE and OPEN/CLOSE) indented properly and clearly under the verb? MN
- Does all use of GO TO conform to installation standards? MN

External Linkages

- Are initial entry and final exit correct? EL
- Is each entry point defined correctly? EL

- Is each parameter referenced in an ENTRY statement a 77 or 01 item in the linkage section? EL
- Is the usage of STOP,RUN/GOBACK/EXIT PROGRAM verbs correct? EL
- For each external call to another module: EL
  Are all required parameters passed to each called module?
  Are the parameter values passed set correctly?
  Upon final exit from this module, are all files closed?

Logic
- Has all design been implemented? LO
- Does the code do what the design specified? LO
- Is the design correct and complete? LO
- Are the proper number of characters within a field tested or set? LO
- Is each loop executed and the correct number of times? LO

Program Language Usage
- Is the optimal verb or set of verbs used? PU
- Is the installation-defined restricted subset of COBOL used throughout the module? PU
- Is attention given to normal "housekeeping" requirements in COBOL (for example, setting the length of a variable-length target field before a MOVE to that field is executed)? PU

Storage Usage
- Is each field initialized properly before its first use? SU
- Is the correct field specified? SU
- If a storage area is set and used recursively, is its housekeeping performed properly? SU
- Is the field initialized statically (that is, by means of the VALUE clause on its definition), when it should be dynamically (by assignment), or vice versa? SU
- Is the use of the REDEFINES clause in the data item's definition compatible with all uses of the data item in the code? SU
- If the CORRESPONDING option of the MOVE and arithmetic verbs is used, is it absolutely clear from the data definitions which target fields will be affected (and, equally important, which will not)? SU,MN

Test and Branch
- Is the correct condition tested (IF X=ON vs IF X=OFF)? TB
- Is the correct variable used for the test (IF X=ON vs IF Y=ON)? TB
- Is each condition name, used as a test of a data item, defined as an 88-level under that data item? TB
- Is each branch target of a simple GO TO or GO TO ... DEPENDING ON statement, correct and exercised at least once? TB
- Is the most frequently exercised test leg of an IF statement the THEN clause? TB

Performance
- Is logic coded optimally (that is, in the fewest and most efficient statements)? PE
- Has subscripting been used where indexing logic would be more effective and appropriate, or vice versa? PE
- Have ERROR DECLARATIVEs been coded for files likely to have recoverable I/O errors? PE
- Are normal error/exception routines provided for: PE
  ON SIZE ERROR – for arithmetic statements?
  INVALID KEY – for start/read/write/rewrite statements?
  AT END – for search/release/sequential READ?
  ON OVERFLOW – for STRING/UNSTRING?

Maintainability
- Are listing controls utilized to enhance readability (for example, EJECT, SKIPx)? MN
- Are paragraph and SECTION names consistent with the logical significance of the code? MN
- Is each PERFORMed paragraph terminated with an EXIT paragraph? MN
- Is the use of the ALTER statement completely justified, as opposed to some sort of switch/conditional branch flow of control? MN
- Are null ELSEs included as appropriate? MN

Copy Facility Usage
- Is every data item definition and processing paragraph, standardized for the installation, generated in the module via the COPY facility? OT
- Is there a sound reason why the REPLACE option of the COPY statement is utilized to change any of the names of data items in the COPY'd code? OT

PL/I Checklist

Code Documentation Prologue
- Are there discrepancies between the code and the prologue? PR

Procedure Statement (external procedure)
- Is the procedure name the same as the module name? MN
- Are the options present consistent with module design; that is, both the OPTIONS sublist (reentrant, etc.) and the options for the PROC statement (recursive, etc.)? OT
- Are any required options missing? OT

Initial DECLAREs and INCLUDEs, Storage DECLAREs
- Do the names of the data items correspond to their usage? MN
- Does each data item (except for items of obvious usage – constants, subscripts, etc.) contain comments regarding: CC
  Characteristics (fixed- or variable-length, maximum, allowable length, customary (arithmetic) precision, etc.)?
  Interaction with other data items? (Does this data item determine the length of another data item? Is this data item used to control a major loop, etc.?)
  Area of use in the program? (Is it used only in a certain section, or during a particular internal procedure, etc.?)
- Are all related data items placed together according to a particular scheme?
  Usage (arithmetic work areas, work areas for file records, etc.)? MN
  Commonality of purpose (everything used to process a particular file, arguments passed to the same subroutine, etc.)? MN
  Attributes (message texts, constants, etc.)? MN
- Are all data items used as constants designated as such? MN
- Are data items that are required to be in a particular order sequenced correctly? MN
- Do the INITIAL values conform to the design? LO
- Are the correct data attributes used, that is, fixed, character, bit, decimal, etc.? DA
- Is the correct storage attribute used (static, automatic, etc.)? DA
- Are addresses defined correctly (use of BASED, ADDR function, etc.)? DA
- Is the use of the DEFINED attribute in a data description justified by the simplification of data references? DA

File DECLAREs
- Do the names of the files, and the associated data items, correspond with their usage? MN
- Does each file declaration contain comments regarding: CC
  File usage (record type, block size, record length, imbedded keys, etc.), provided such information is not clearly indicated elsewhere, such as the ENVIRONMENT attribute?
  Amount of activity (updated how often, updated every time program is run, etc.)?
  Interaction with other data items? (Do its records contain data items which determine the length of other data items? Is the length of its records determined by a data item elsewhere in the program, etc.?)
- Are statistics kept on file activity in a given run or series of runs? PE
- Are the correct file attributes used (STREAM/RECORD, SEQUENTIAL/DIRECT/TRANSIENT, etc.)? OT
- Is there a valid reason for placing file attributes on an OPEN statement, rather than in the file declaration itself? MN
- Is the correct balance struck between specifying complete file attributes in the program, and specifying some of them dynamically at execution time (for instance, block size, maximum record length, etc.)? MN

INCLUDEs
- Are all data areas required by the module INCLUDEd correctly? DA
- Is the INCLUDE feature (macro or option) used to define identical or nearly identical data areas (such as work areas and file areas)? DA
- Is every data item definition and processing procedure standardized for the installation generated in the module via the INCLUDE facility? OT
- Is there a sound reason why any of the labels or names of the data items are changed in an INCLUDEd portion of text? OT

Format
- Are block comments included for major functional areas, such as the internal procedure section? CC
- Are the module's comments sufficiently detailed? CC
- Are comments organized and formatted so that they do not interfere with the readability of the program? CC
- Are comments accurate and meaningful? CC

- Does the code essentially correspond to the outline of the module documented in the prologue? PR
- Does each internal procedure or subroutine have a consistent purpose which justifies and/or necessitates placing all the code together under such a grouping? CC
- Does each such internal procedure or subroutine document the function it accomplishes and the part of the overall logic it represents? CC
- Are meaningful labels used? MN
- Are IFTHENELSE and do-groups aligned properly? MN
- Are nested IF's and do-groups indented properly? MN
- Are block comments and remarks effectively positioned? MN
- Are the clauses of complex statements indented properly under the verb? MN
  Complex arithmetic and assignment?
  File names and options on OPEN/CLOSE?
  ALLOCATE/FREE?
  CALL/ENTRY?
  FROM/INTO/KEYFROM/KEYTO on I/O statements?
  Complex DO?
  GET/PUT?
- Does all use of GO TO conform to installation standards? MN

Entry and Exit Linkage (EL)
- Are initial entry and final exit correct?
- Is each parameter referenced in a PROCEDURE or ENTRY statement defined with the proper attribute (that is, compared to arguments in calling procedure)?
- Are subroutines (internal PROCEDUREs) entered and exited properly?
- Are options on internal PROCEDURE statements consistent with module design?

Logic (LO)
- Has all design been implemented?
- Does the code do what the design specified; that is, is the design translated correctly?
- Is the design correct and complete?
- Are the appropriate number of characters within a field tested or set?
- Is each loop executed the correct number of times?

Program Language Usage (PU)
- Is the optimal verb or set of verbs used?
- Is the installation-defined restricted subset of PL/I used throughout the module?
- Are PL/I built-ins and functions used appropriately in lieu of equivalent ordinary statements (for example, LENGTH, MAX, MIN, etc.)?
- Is special attention paid to conversion rules?
- Is an arithmetic built-in used instead of arithmetic assignment when difficult cases of precision arise (such as multiply, divide, etc.)?
- Is attention given to normal "housekeeping" requirements in PL/I (such as setting the length of a variable-length target field before an assignment is made to that field)?
- Are the proper scope rules of PL/I followed (for example, defining the same variable in procedure and in contained BEGIN block, etc.)?
- Is the use of the LABEL variable completely justified, as opposed to some sort of switch/conditional branch flow of control?

Storage Usage (SU)
- Is each field to be initialized set correctly?
- Before the first use of any field, has it been initialized properly?
- Is the correct field specified?
- If storage is set and used recursively, is it "housekept" properly, depending on storage attributes (controlled, static, automatic), on a procedure basis, at least?
- Is the field initialized statically (that is, by means of the INITIAL attribute in its DCL), when it should be initialized dynamically (by assignment), or vice versa?
- Is the use of the DEFINED attribute in the data item's definition compatible with all uses of the data item in the code?
- If a BY NAME assignment is done, is it absolutely clear from the structure definitions which target fields will be affected (and, equally important, which will not)?

Test and Branch (TB)
- Is the correct condition tested (IF X=ON vs IF X=OFF)?
- Is the correct variable used for the test (IF X=ON vs IF Y=ON)?
- Are null THENs/ELSEs included as appropriate?
- Is each branch target correct and exercised at least once?
- Is the most frequently exercised test leg of an IF statement the THEN clause?

Performance (PE)
- Is logic coded optimally (that is, in the fewest and most efficient statements)?
- Have ON statements been coded for files likely to

have recoverable I/O errors (TRANSMIT, UNDEFINEDFILE, KEY, etc.)?
- For array assignments, would a more efficient assignment operation result from a subscripted do-loop, or vice versa? Similarly for structures?
- Are normal error/exceptional conditions provided for via ON (condition) statements?
  - ENDFILE/KEY/ERROR?
  - CONVERSION/OVERFLOW/SIZE?

Maintainability (MN)
- Are listing controls utilized to enhance readability?
  - %PAGE, %SKIP in macros/preprocessor?
  - Control characters/blank cards in regular source?
- Are labels and PROCEDURE names consistent with the logical significance of the code?
- Are paired DO and END labels used?
- Is proper formatting (such as indenting) provided either by the programmer or the compiler?

External Linkages (EL)
- For each linkage call to either a macro or another module:
  - Are all required parameters passed to each linkage?
  - Are the parameter values passed set correctly?
  - If the linkage is a macro:
    - Does the inline expansion contain all required code?
    - Are there storage conflicts between macro and calling module?
- If the linkage returns, are all returned parameters processed correctly?
- Upon final exit from this module, are all files closed and CONTROLLED storage FREEd?

Indirect Addressing (OT)
- Is each pointer notation that is used correct?
- If pointer/offset notation is used, is it required (that is, can a more direct method of addressing be used)? Similarly for the I-sub feature of arrays?

FORTRAN Checklist

Comments
- Are comment lines used to group logically related statements? MN

Data Declaration
- If COMMON contains many entries, check that repetitions in other subroutines match in length and order of listing. DA
- If EQUIVALENCE is used, check shared data storage for problems of unintended use. DA
- Check that FORMAT statements match READ, WRITE lists and that the intended conversion of data is specified. DA
- Are FORMAT statements grouped at the beginning of the program? MN
- Are meaningful names used? MN

Control
- Check nesting of DO loops. TB
- If extended range of DO is used, check carefully. LO
- Check whether any DO variable is to be used upon exit of the DO loop. LO
- Does any use of GO TO conform to installation standards? LO

Computation
- Check all mixed mode expressions in assignment statements. DA
- Examine all usage of complex numbers. LO

Data Handling
- Check that indexing out of the range of an array does not inadvertently destroy constants or data areas. SU

Format
- Are installation indention standards followed? MN

Call Usage
- Watch for the use of call by value, call by name. EL
- Match parameter list in caller-called program. PU

Subroutine Usage
- Check that any needed local variables have not been destroyed by consecutive invocations. LO

Variable Types
- Check that the correct variable types (integer, real, complex, logical) are declared. PU

Object Time Execution
- Check job control language. OT
- Check the use of associated variables in direct access statements. LO
- Check for the destruction of constant values upon return of a subroutine. EL

  For example
```
  CALL SUB(2.,x,y)
  .
  .
  .
  SUBROUTINE SUB(a,b,c)
  A=4.
  RETURN
```
  in which 2. is destroyed.

FORTRAN Conventions
- Check the use of FORTRAN default conventions. PU

  Examples:
```
A. 1)    X=Y**12
   2)    X=Y**12.
```
  These produce different answers. In case 1, multiplication is performed, while in case 2, logical expression routines are used.
```
B. 1)    X=X+2     I=I*2
   2)    X=X+2.    I=I*2.
```
  In case 1, the constant 2 must be floated, while in case 2, the number must be fixed.

Error Analysis
- Check that arithmetic expressions will be evaluated as intended (use of parentheses). PU
- Check whether the proper length of precision has been selected for calculation and whether constants match in type. PU

Assembler Checklist

Register Usage (RU)

- Check that base registers defined by USINGs are all loaded at the appropriate time, that is, before first attempted use.
- Check that all temporary base registers are dropped when no longer needed.
- Check to ensure that base registers cannot be destroyed during execution, particularly via calls to subroutines or across CSECT boundaries.
- Check that all intended entry points are defined by ENTRY statements. Use the External Symbol Dictionary to verify their external status.

Program Language Usage(PU)

- Check for operation code misspellings that will nevertheless be accepted by the assembler because the misspelling is another valid assembler operation code for which the operands have the same format as the intended instruction. Examples are:

	Instruction			
	As Intended		*As Misspelled*	
a.	SR	SUBTRACT REGISTER	S	SUBTRACT
b.	LA	LOAD ADDRESS	L	LOAD
c.	STH	STORE HALFWORD	SH	SUBTRACT HALFWORD
d.	SRL	SHIFT RIGHT LOGICAL	SLR	SUBTRACT LOGICAL REGISTER

a. Omitting the "R" in the instruction: SR 5,8 will yield the valid instruction: S 5,8
b. Omitting the "A" in the instruction: LA 5,8 will yield the valid instruction: L 5,8
c. Omitting the "T" in the instruction: STH 5, XYZNUM will yield the valid instruction:
SH 5, XYZNUM
d. The error in this case can go either way, that is, from SLR to SRL or from SRL to SLR. For example, both instructions are equally plausible: SLR 5,8 and SRL 5,8.

- Check that Load Multiple (LM) picks up the desired sequence of fullwords and that they are placed into the expected registers.
- Ensure that CLI is not used when TM is really required, that is, check that bit-switches are not confused with byte-switches.
- Check that register 2 has not been unwittingly destroyed by a TR (Translate) or TRT (Translate and Test) instruction.
- Check that expressions representing lengths are specified correctly:
For example,
```
MVC 0(LABEL2-LABEL1,R6),0(R3)
vs
MVC 0(LABEL2-LABEL1+1,R6),0(R3)
or
vs
MVC 0(LABEL2-LABEL1-1),R6),0(R3)
```
- Check that all possible cases of conditional assembly parameters are generating the code expected. An assembly should be produced for all major cases and the logic of each compared with a card image printout of the source statements.
- Check that use of unconditional branches conforms to installation standards.
- Check that a save area exists, if required, and is set up according to the prevailing operating system conventions (for example, forward/backward pointers, etc.). If available, a system macro should be used to establish save area linkages (for example, the OS SAVE macro).
- Check that register usage conforms to the prevailing standards applicable to the project, if any. If no special standards are in use, operating system standards should be applied (for example, for OS, R13 is the save area pointer; R14, the return address; R15, the entry point address; R1, the parameter list pointer; R10 and R11, parameter registers).
- Check that EQUATEs are all meaningfully defined; in particular, check that register EQUATEs such as R5 EQU 5 are not redefined as a shortcut method of introducing changes, as would be the case if the foregoing example were changed to R5 EQU 6 to free up register 5, assigning its current use to register 6.
- Check system macro calls to ensure that keyword parameters are not specified as positional parameters, and vice versa. For macros accepting mixed format (both positional and keyword parameters), a keyword parameter written in positional form might be accepted as meaning something else than intended.

Data Area Usage (DA)

- Check that DSECTs correspond in format to the data which they represent.
- If modifications have been made to a data structure, for example, addition of fields within the structure (control block), check that required alignments are still preserved. Use particular care in the case of control blocks iteratively generated via conditional assembly logic. (Even if the first

block is right, subsequent blocks may not start on the same type of boundary, causing program failure only when operating on blocks other than the first.)

Maintainability (MN)

- Ensure that extended mnemonics are used whenever possible rather than hand-coded condition-code masks.

Code Comments (CC)

- Check that instruction-level documentation adds meaning to the code, for example, in the instruction SR R5,R5 ZERO R5, the comment ZERO R5 adds nothing to the content of the instruction, while SR R5,R5 ASSUME NO REQUESTS PENDING does add meaning.

Logic (LO)

- Is each loop executed the correct number of times?

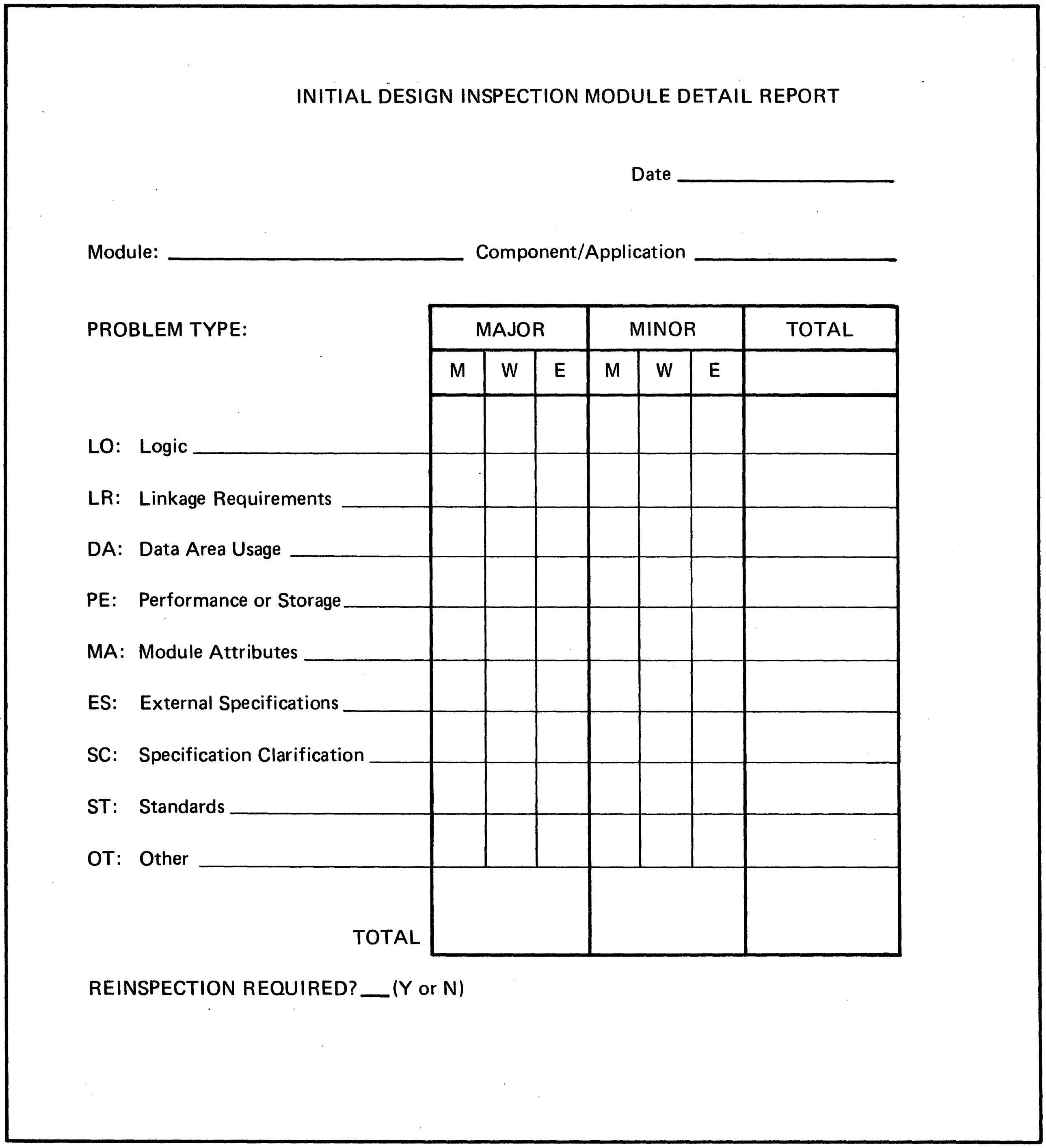
Appendix E: Reporting Forms

Module Detail Reports

The module detail report is completed for each module in which valid errors are discovered during an inspection. The following describes the use of each field of the three reports – initial design inspection module detail report (Figure 22), detail design inspection module detail report (Figure 23), and code inspection module detail report (Figure 24):

1. MODULE: The module name.
2. COMPONENT/APPLICATION: The associated component/application name.
3. PROBLEM TYPE: Summarize the number of problems by type (logic, etc.), by severity (major/minor – a major error is one that would cause the program to malfunction), and by category (missing, wrong, extra). For modified modules, problems in the changed portion and in the base program can be shown as follows: 3(2), where 3 is the number of problems in the changed portion, and 2 is the number of problems in the base.
4. REINSPECTION REQUIRED?: Indicate whether the module requires a reinspection. All valid errors found in the inspection are listed and attached to the report. A brief description of each problem, its error type, and the estimated rework time to correct it is included, as shown below.

LO/M/MAJ The VARY side of the close DEB linkage design is not given (rework = 3.0 hours)



INITIAL DESIGN INSPECTION MODULE DETAIL REPORT

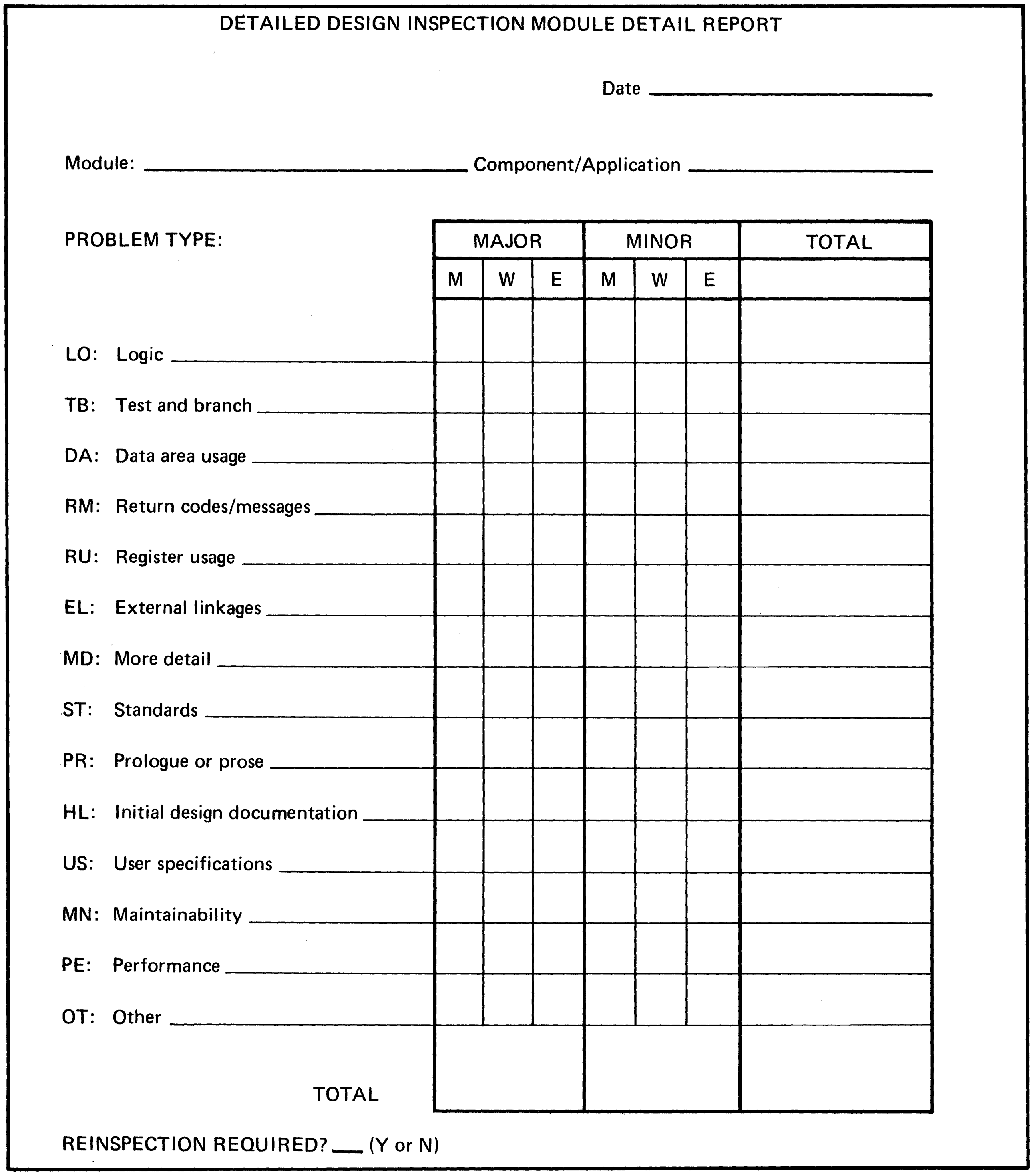
Date ______________

Module: ______________________ Component/Application ______________

PROBLEM TYPE:	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
LO: Logic							
LR: Linkage Requirements							
DA: Data Area Usage							
PE: Performance or Storage							
MA: Module Attributes							
ES: External Specifications							
SC: Specification Clarification							
ST: Standards							
OT: Other							
TOTAL							

REINSPECTION REQUIRED?___(Y or N)

Figure 22. Initial design inspection module detail report



DETAILED DESIGN INSPECTION MODULE DETAIL REPORT

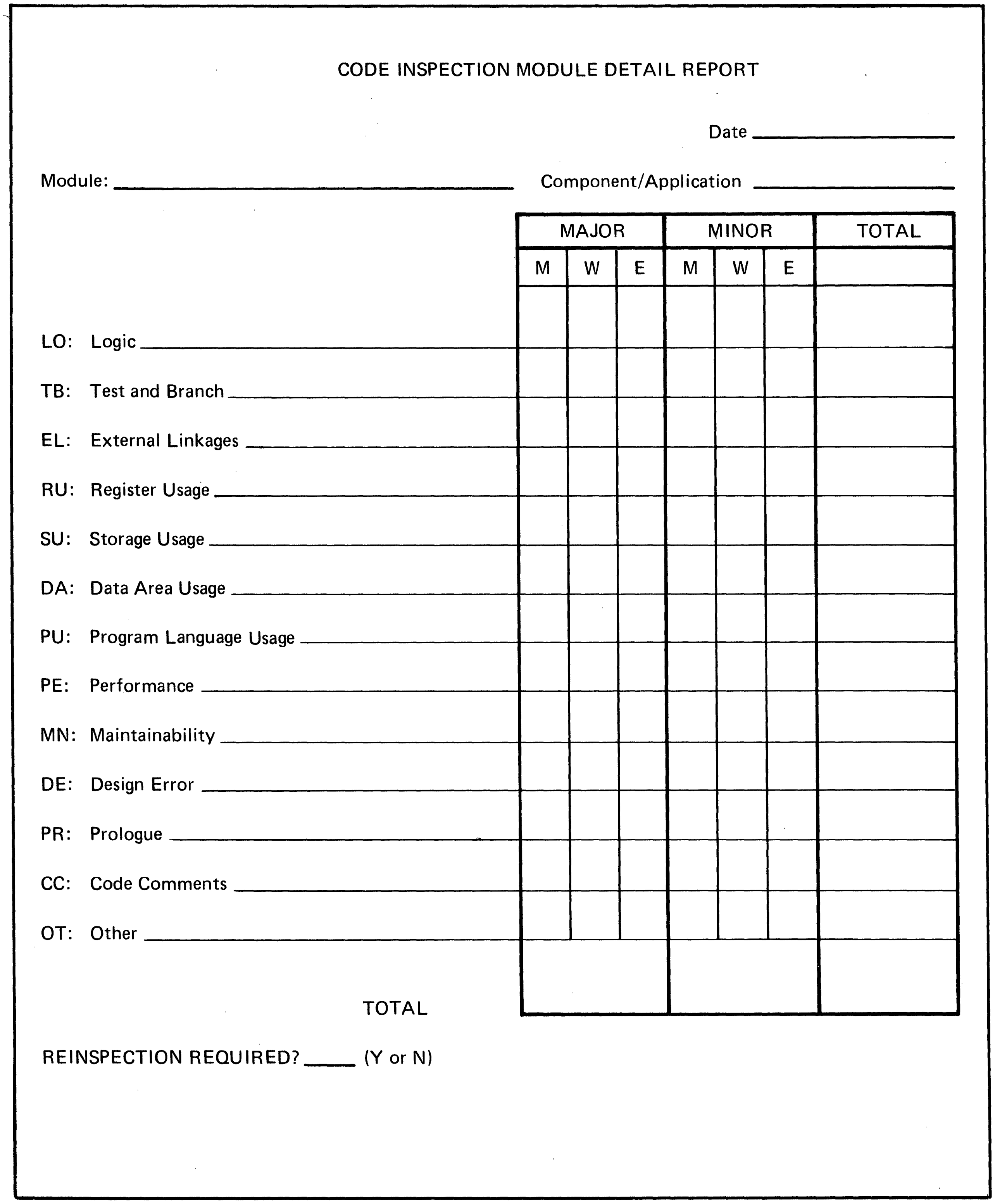
Date ____________

Module: ____________ Component/Application ____________

PROBLEM TYPE:	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
LO: Logic							
TB: Test and branch							
DA: Data area usage							
RM: Return codes/messages							
RU: Register usage							
EL: External linkages							
MD: More detail							
ST: Standards							
PR: Prologue or prose							
HL: Initial design documentation							
US: User specifications							
MN: Maintainability							
PE: Performance							
OT: Other							
TOTAL							

REINSPECTION REQUIRED? ___ (Y or N)

Figure 23. Detailed design inspection module detail report



CODE INSPECTION MODULE DETAIL REPORT

Date ____________

Module: ______________________ Component/Application ____________

	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
LO: Logic							
TB: Test and Branch							
EL: External Linkages							
RU: Register Usage							
SU: Storage Usage							
DA: Data Area Usage							
PU: Program Language Usage							
PE: Performance							
MN: Maintainability							
DE: Design Error							
PR: Prologue							
CC: Code Comments							
OT: Other							
TOTAL							

REINSPECTION REQUIRED? _____ (Y or N)

Figure 24. Code inspection module detail report

Summary Inspection Report

The summary inspection report (see Figure 25) contains the results of the inspections of several modules (usually those forming a component of an application) and is distributed to design and development management. The following describes how each section of the report is used.

REPORT NAME: The box is checked corresponding to the type of inspections summarized.

SUBJECT: The unit inspected is identified.

MODULE NAME: The name of each module as it resides on the source library.

NEW OR MOD.: "N" if the module is new; "M" if the module is "modified".

FULL OR PART INSP.: For a modified module, "F" if the module was fully inspected; "P" if the module was partially inspected.

WORK PERFORMED BY: The correct categories are checked and the individuals' names specified.

EST. PRE. ELOC/NCSS: ELOC is the estimated executable lines of source code made before a design inspection by the designer. NCSS is a count of the lines of noncommentary source statements made before a code inspection by the programmer.

EST. POST. ELOC/NCSS: The estimate or count made after the inspection.

REWORK ELOC/NCSS: The estimated executable lines of source code in rework as a result of the inspection.

OVERVIEW AND PREP.: The number of person hours (in tenths of hours) actually spent preparing for the overview, in the overview meeting itself, and preparing for the inspection meeting.

INSP. MEETG.: The number of person hours actually spent on the inspection meeting.

REWORK: The estimated number of people hours spent to correct the problems found during the inspection.

FOLLOW-UP: The estimated number of people hours spent by the moderator (and others, if necessary) in verifying the correctness of changes made by the author as a result of the inspection.

COMPONENT: The component of which the module is a part.

REINSPECTION REQUIRED?: Yes or no.

LENGTH OF INSPECTION: Clock hours spent in the inspection meeting.

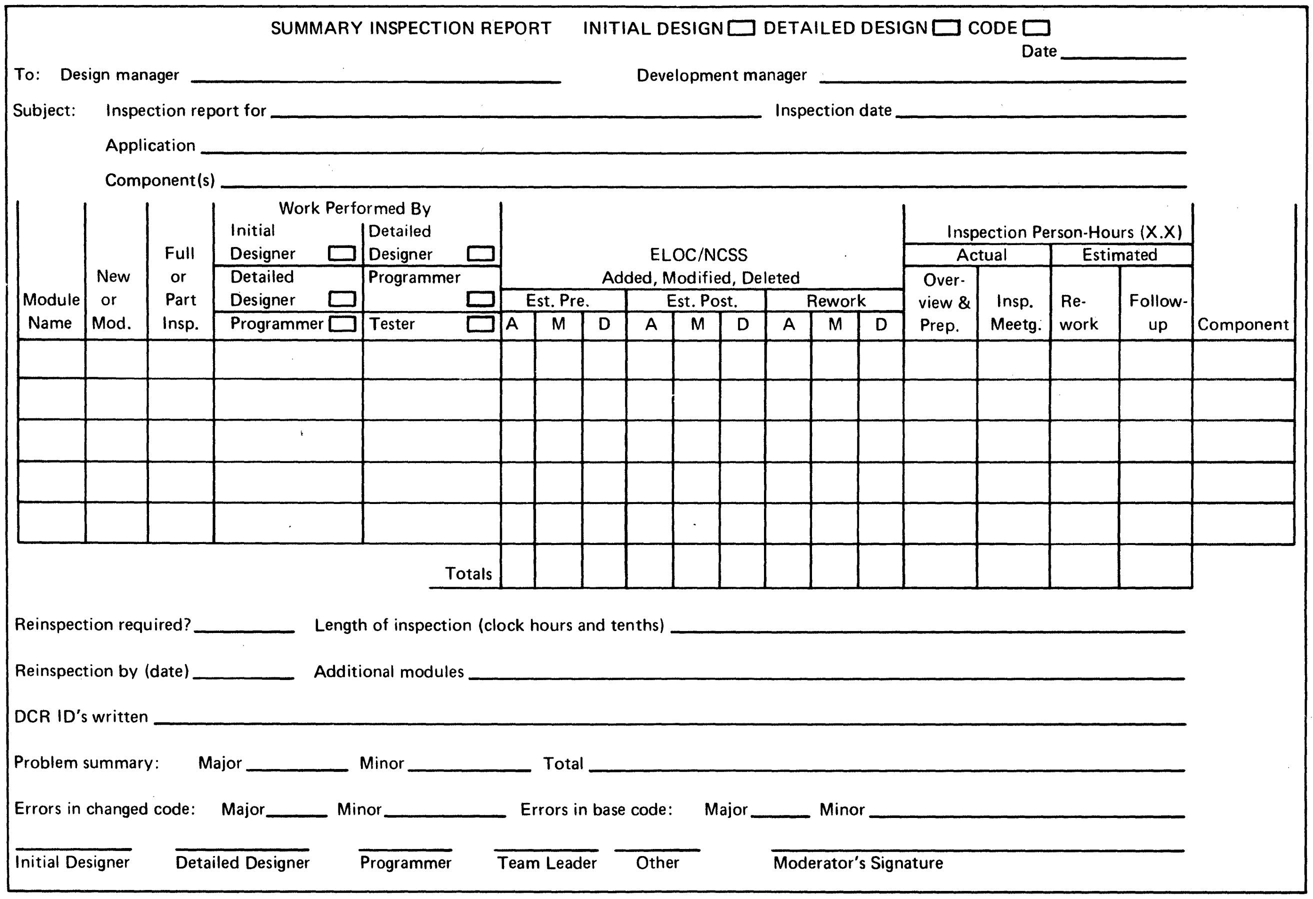
REINSPECTION BY (DATE): Latest acceptable date for reinspection.

ADDITIONAL MODULES?: For these components, are additional modules yet to be inspected?

DCR's ID's WRITTEN: The identification of design change requests written to cover problems in rework.

PROBLEM SUMMARY: Totals taken from module detail form(s).

INITIAL DESIGNER DETAILED DESIGNER, etc.: Identification of members of the inspection team.



SUMMARY INSPECTION REPORT INITIAL DESIGN ☐ DETAILED DESIGN ☐ CODE ☐

Date ________

To: Design manager ________________ Development manager ________________

Subject: Inspection report for ________________ Inspection date ________________

Application ________________

Component(s) ________________

			Work Performed By		ELOC/NCSS Added, Modified, Deleted									Inspection Person-Hours (X.X)				
					Est. Pre.			Est. Post.			Rework			Actual		Estimated		
Module Name	New or Mod.	Full or Part Insp.	Initial Designer ☐ Detailed Designer ☐ Programmer ☐	Detailed Designer ☐ Programmer ☐ Tester ☐	A	M	D	A	M	D	A	M	D	Over-view & Prep.	Insp. Meetg.	Re-work	Follow-up	Component
				Totals														

Reinspection required? ________ Length of inspection (clock hours and tenths) ________________

Reinspection by (date) ________ Additional modules ________________

DCR ID's written ________________

Problem summary: Major ________ Minor ________ Total ________________

Errors in changed code: Major ______ Minor ________ Errors in base code: Major ______ Minor ________

Initial Designer Detailed Designer Programmer Team Leader Other Moderator's Signature

Figure 25. Summary inspection report

Appendix F: Test Plan and Test Case Inspections

This appendix discusses two additional types of inspections – test plan and test case – as shown in Figure 26. Since the inspection process is basically the same as for other inspections, this appendix discusses in detail only those items that differ from the other inspection types.

Test plan and test case inspections are designed to inspect the plan made for testing new and/or changed functions of a component and to inspect the test cases developed from that plan with the objectives of ensuring more complete and smoothly functioning testing with fewer testing resources. An indication of the productivity possibilities of test plan and test case inspections was provided by an IBM test in which four new functions approximating 20K lines of source code were subjected to these inspection types in addition to the design and code inspections. It was estimated that personnel hours spent in testing and in test plan and test case inspections were 85% less than the personnel hours that would have been spent in function testing without the test plan and test case inspections. (Note that these test plan and test case inspection results involved only one study. Since the results depend on many factors, they cannot be considered representative of every situation.)

Test Plan Inspections

Objectives

The primary objectives of a test plan inspection are to verify that function testing will provide assurance that the function operates correctly within its intended environment and the previously tested related functions still execute properly. A major part of a test plan is the identification and description of each test case.

Process and Participants

The test plan inspection process consists of the same steps as other inspections: planning, preparation, inspection meeting, rework, and follow-up.

In addition to the moderator, the participants include:

- Functional tester – the author of the test plan.
- Functional designer – the designer of the function to be tested. The designer is the key inspector because it is assumed that a function's designer has the most knowledge of the function being tested and can verify that the function will be comprehensively tested in an appropriate environment.

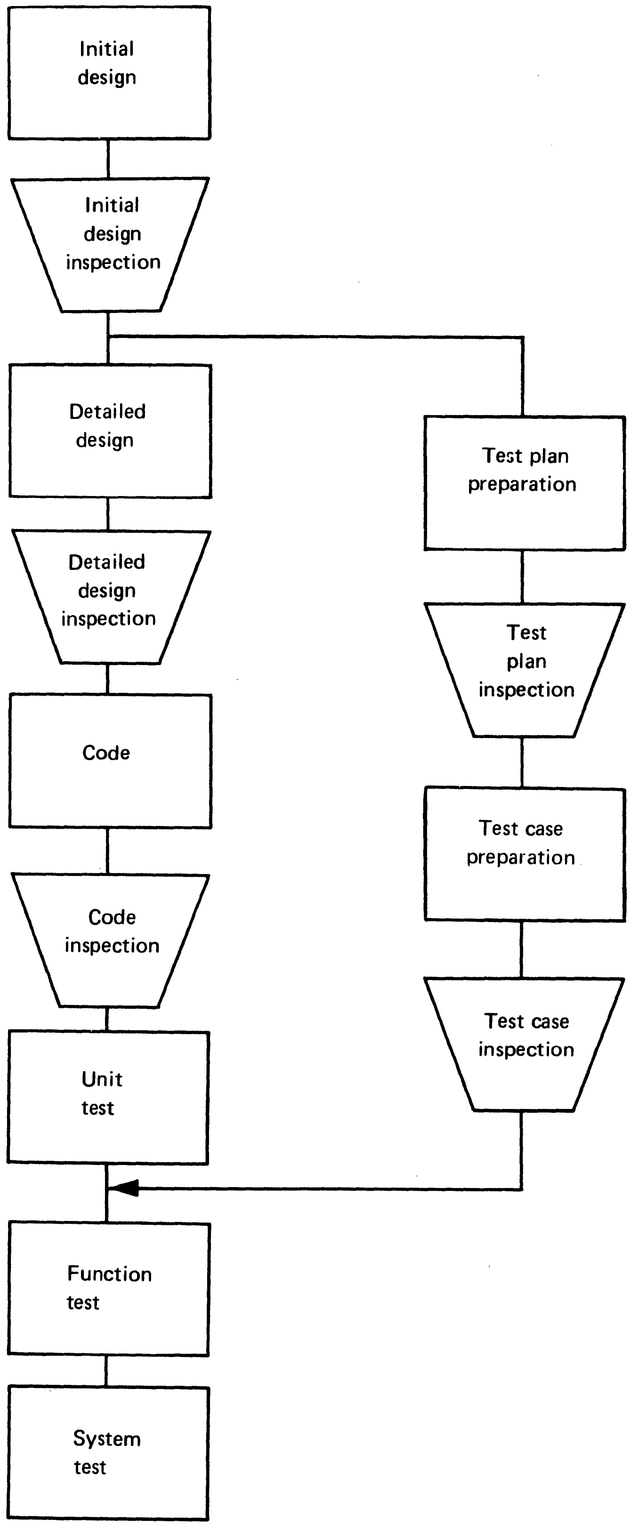


Figure 26. Inspection types

- Other inspectors – these inspectors, from design and development groups, are usually responsible for incorporating the function into a particular program or application and can verify that the significant coding changes made to their modules (particularly linkages) will be tested. Even if one test plan includes several functions, there may be one inspection for each function. In such cases, it is worthwhile for one of the participants to attend all the inspections to provide continuity. Frequently, test cases designed to test one function can be slightly modified to test others.

Preparation
Each participant examines the test plan and the initial design documents to verify that the function will be properly tested. It is assumed that the test plan, at a minimum, includes the following information:

- A general test philosophy or strategy
- A description of the function to be tested
- A representation of functional coverage (that is, matrix, cause and effect graph, family tree, etc.)
- A description of the conditions each test case will test and how this will be accomplished
- Testing dependencies (hardware/simulator needs, etc.)
- Entrance and exit criteria

Participants in the inspection meeting review the planned testing activity as documented in the test plan and compare the plan to the initial design materials with the objective of trying to find errors in the test plan. As with other inspection types, a checklist can assist in focusing attention on possible errors. The checklist below is a sample of the items that could be included; it should be modified to meet the needs and standards of each installation.

- Is the description of the function being tested, as documented in the test plan, complete and accurate?
- Are the mainline and alternate paths listed sufficiently to provide confidence that the function being tested operates correctly?
- Is the testing approach feasible?
- Are all the new and/or changed user linkages exercised?
- Is a sufficient number of defaults exercised?
- Are messages verified?
- Are error paths exercised?
- Are return codes generated?
- Are sufficient and proper tests identified to reverify previously tested related functions (regression test cases)?
- Are there simulator and hardware dependencies that are not addressed?
- Are test plan entrance and exit criteria realistic?
- Are there any outstanding design changes to be made that will invalidate the completeness of the test plan?

Rework and Follow-up
The same considerations apply to test plan inspection steps as to the corresponding steps of other inspection types.

Reporting Forms
Test plan inspection detail and summary report forms as shown in Figures 27 and 28. It is expected that installations will tailor them to their own needs.

Test Case Inspections

Objectives
The primary objectives of a test case inspection are to review the test cases, now in executable form, to verify that:

- The test cases cause those conditions specified in the test plan to be executed.
- Each test case prologue provides a complete and accurate description of its purpose and expected results, and explicit instructions for its execution.

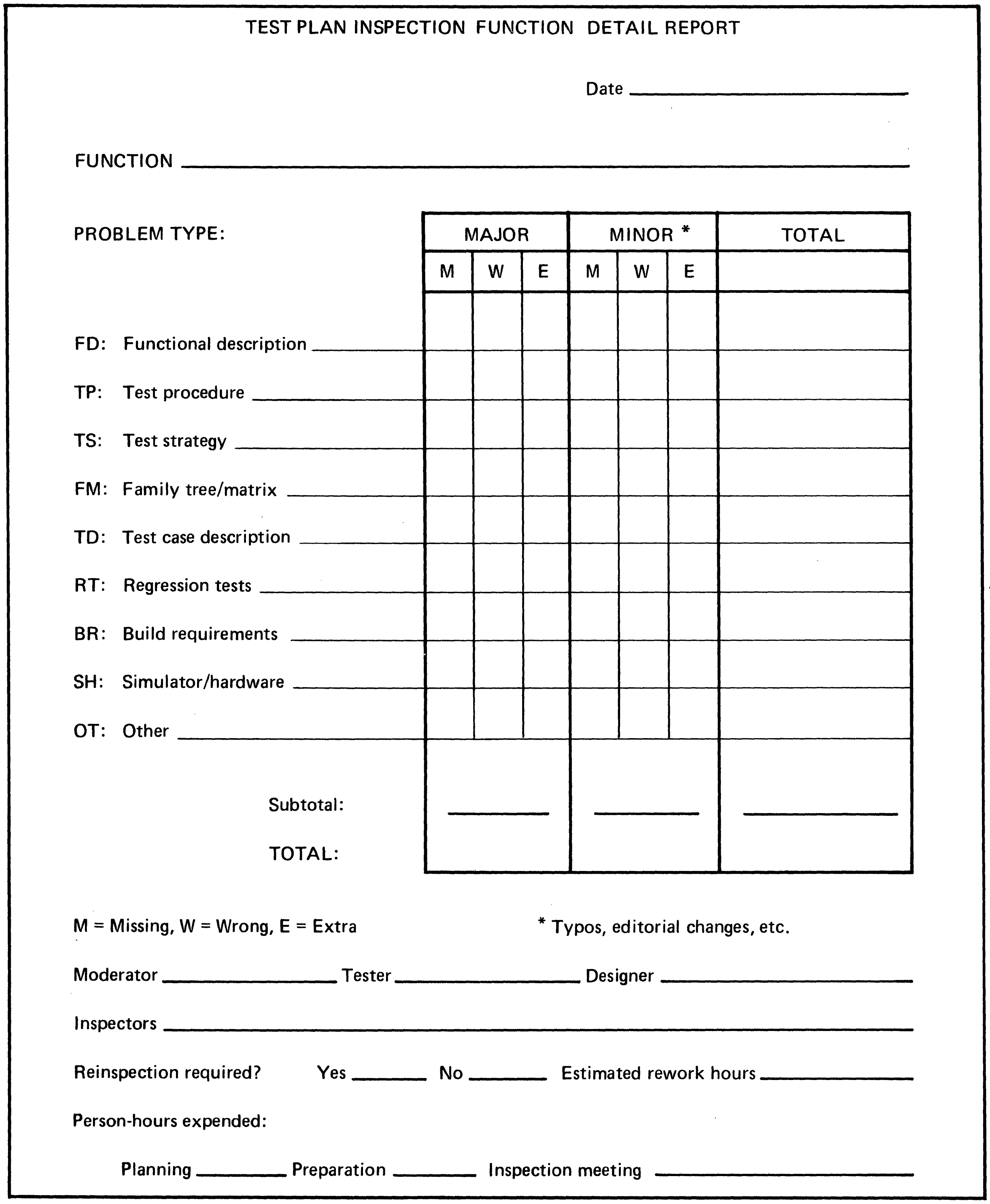
Process and Participants
The test case inspection process consists of the same steps as other inspections: planning, preparation, inspection meeting, rework, and follow-up.

In addition to the moderator, the participants include:

- The person responsible for the test cases being inspected.
- An inspector experienced in running test cases. This inspector can determine whether the prologue identifies test case dependencies and provides all the information needed by the operator.
- An inspector experienced in developing test cases for this application.

Preparation
Each participant examines the test case data, comparing each test case with its description in the test plan. The test case inspection materials include:

1. Test case data
2. Test case prologue containing the following information:
   - Description and purpose of the test case
   - Setup requirements



TEST PLAN INSPECTION FUNCTION DETAIL REPORT

Date ____________

FUNCTION ____________

PROBLEM TYPE:	MAJOR			MINOR *			TOTAL
	M	W	E	M	W	E	
FD: Functional description							
TP: Test procedure							
TS: Test strategy							
FM: Family tree/matrix							
TD: Test case description							
RT: Regression tests							
BR: Build requirements							
SH: Simulator/hardware							
OT: Other							
Subtotal:	______			______			______
TOTAL:							

M = Missing, W = Wrong, E = Extra

* Typos, editorial changes, etc.

Moderator ____________ Tester ____________ Designer ____________

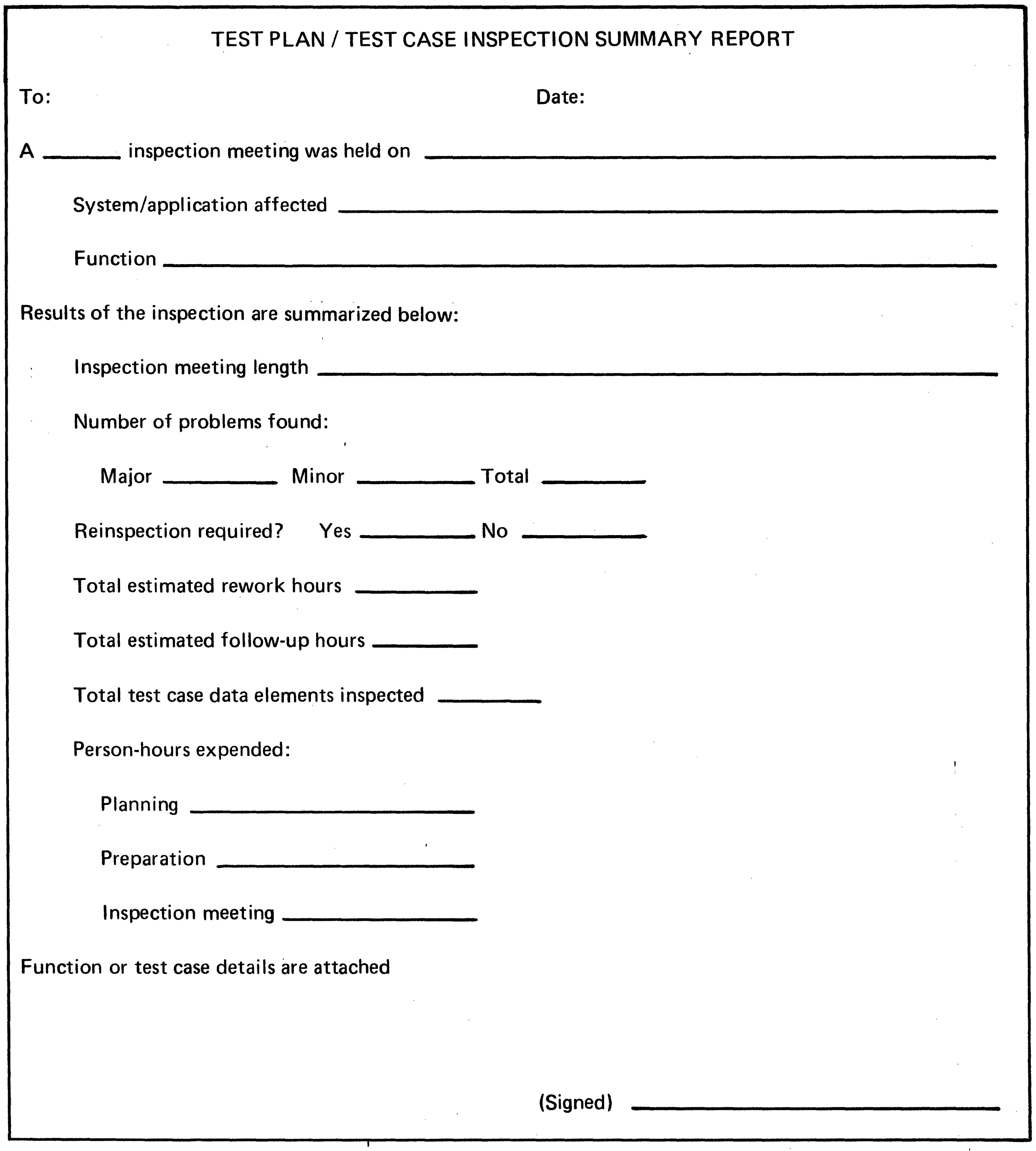
Inspectors ____________

Reinspection required? Yes ______ No ______ Estimated rework hours ____________

Person-hours expended:

Planning ______ Preparation ______ Inspection meeting ____________

Figure 27. Test plan inspection function detail report


TEST PLAN / TEST CASE INSPECTION SUMMARY REPORT

To: Date:

A ______ inspection meeting was held on ______________________

System/application affected ______________________

Function ______________________

Results of the inspection are summarized below:

Inspection meeting length ______________________

Number of problems found:

Major ________ Minor ________ Total ________

Reinspection required? Yes ________ No ________

Total estimated rework hours ________

Total estimated follow-up hours ________

Total test case data elements inspected ________

Person-hours expended:

Planning ______________

Preparation ______________

Inspection meeting ______________

Function or test case details are attached

(Signed) ______________________

Figure 28. Test plan/test case summary inspection report

- Operator instructions for running the test case
- Normal and abnormal completion messages
- Dependencies required by this test case, such as simulator, hardware, or test macros
- Name of owner of the test case

3. Sections of the test plan necessary to define this test case
4. A copy of the initial design documents

Participants in the inspection meeting review the test cases with the objective of finding errors in them. As with other inspection types, a checklist can assist in focusing attention on possible errors. The checklist below is a sample of the items that could be included; it should be modified to meet the needs and standards of each installation.

Prologue:

- Is the description of the purpose of this test case complete and accurate?
- Are the operator instructions explicit and clear to ease test case execution?
- Are all dependencies identified?
- Are all normal and abnormal completion messages identified?
- Are setup requirements explicit and complete?
- Is the owner of the test case identified?
- Are there "progress" messages identified that will notify the operator when significant parts of the test case are being executed?

Test Data:

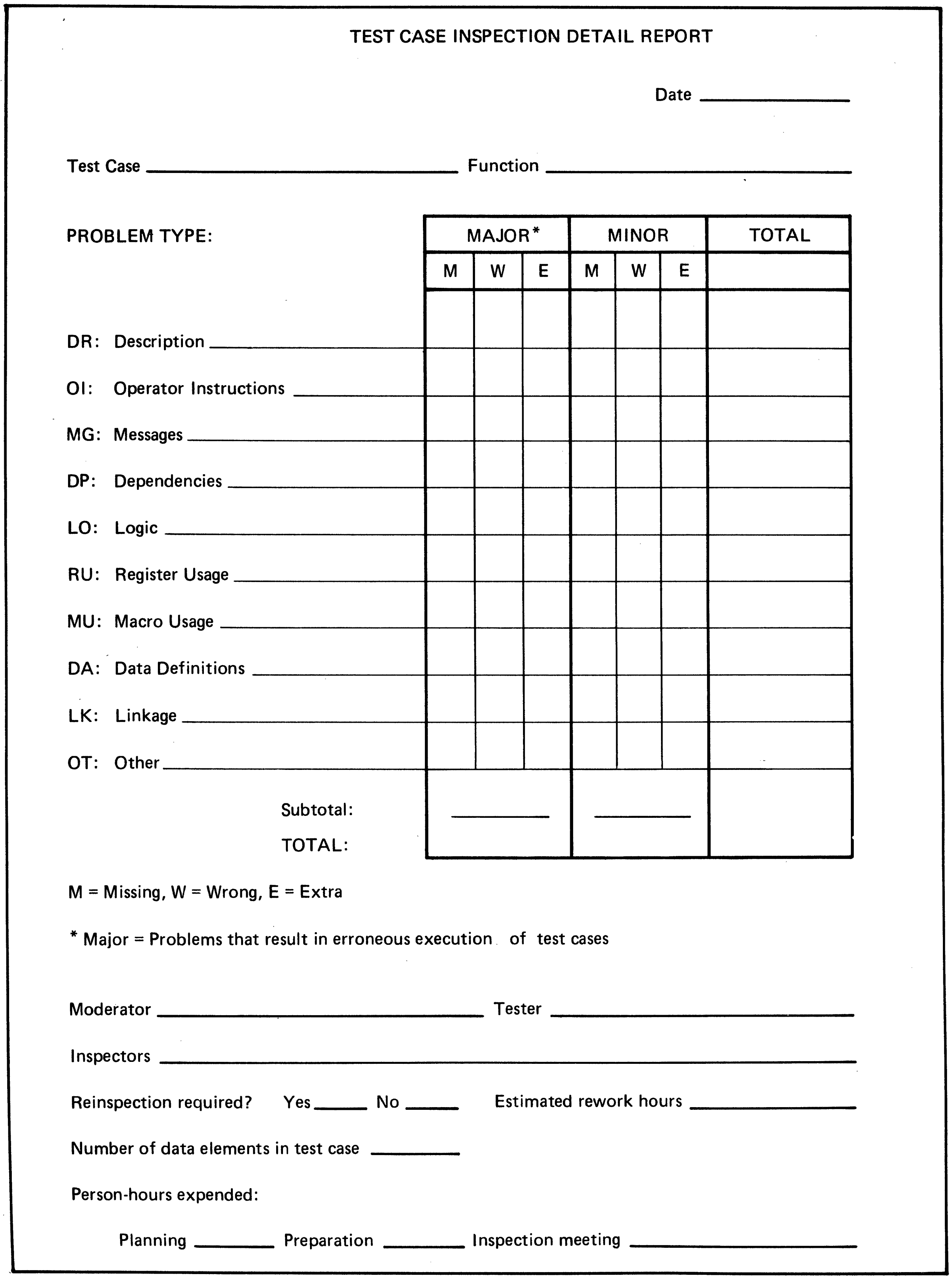
- Does the test data follow its description as stated in the test plan?
- Are the appropriate conditions established to test the intended variations?
- Is the test for successful completion correct?
- Are initial declares, respecify, and includes complete and correct?
- Are entry and exit linkages correct?
- Are macros issued properly?
- Are appropriate return and feedback codes properly verified?
- Are the messages identified in the prologue issued during the test case?

Rework and Follow-up

The same considerations apply to test case inspection steps as to the corresponding steps of other inspection types.

Reporting Forms

The test case inspection detail report form is shown in Figure 29. The summary form is the same as that used for test plan inspections (Figure 28). It is expected that installations will tailor the forms to their own needs.



TEST CASE INSPECTION DETAIL REPORT

Date ________

Test Case ________ Function ________

PROBLEM TYPE:	MAJOR*			MINOR			TOTAL
	M	W	E	M	W	E	
DR: Description							
OI: Operator Instructions							
MG: Messages							
DP: Dependencies							
LO: Logic							
RU: Register Usage							
MU: Macro Usage							
DA: Data Definitions							
LK: Linkage							
OT: Other							
Subtotal:	________			________			
TOTAL:							

M = Missing, W = Wrong, E = Extra

* Major = Problems that result in erroneous execution of test cases

Moderator ________ Tester ________

Inspectors ________

Reinspection required? Yes ____ No ____ Estimated rework hours ________

Number of data elements in test case ________

Person-hours expended:

Planning ________ Preparation ________ Inspection meeting ________

Figure 29. Test case inspection detail report

Appendix G: A Report on the Use of Inspections for an Applications Development Project within IBM

This report describes the use of inspections in an applications development project in IBM's Information Services Ltd., England and was prepared by individuals in close contact with the project. The objectives of the pilot project were to check as to whether inspections did increase productivity and program quality. Quality was to be measured by the APAR (reportable error rate), with an objective of one valid APAR per thousand lines of code in the first year of production. As a more immediate indication of quality, the system test error rates would be compared with a project that used structured walk-throughs. Since productivity increases are much more difficult to measure because of the lack of an accurate estimating method, this area was to be treated more subjectively.

Another objective was to determine the acceptability of the process to the team members and the project managers, with emphasis on the impact on the workload and the schedule; and also to judge the effect on the individual of the open examination of an individual's work, with the consequent disclosure of errors.

ProjectDescription

The inspections pilot project was the IBM World Trade Parts Returns System (PRS), a system to track and report on parts taken out or returned by CE's. PRS is IMS-based, primarily data base, but with one data communications component. The original estimate was 5000 lines of PL/I code in nine separately identified programs. The final count was 6271 lines of code.

The team used most of the improved programming technologies (structured programming, top-down development, development support libraries, Hierarchy plus Input-Process-Output (HIPO), and chief programmer teams) during system development. However, they used inspections rather than walk-throughs, used top-down design starting with the detailed design level, and documented the design with a combination of HIPO charts and prose description. Although structured programming techniques were used, strict structuring was not always apparent. TSO 3270 Display Support and Structured Programming Facility (TSO/SPF), program product 5740-XT8, was used for library management.

The team consisted of five people – a project leader and four programmer/analysts, one of whom was the chief programmer. The project leader devoted the greater part of the time to project control and took only an indirect part in the design and coding of the system. However, the leader made a point of attending almost all of the inspection meetings. The rest of the design/code responsibilities were shared by the four programmer/analysts. Average DP experience of team members at the beginning of the project was about 4 1/2 years, with a minimum of 2 1/2 years and a maximum of 6 1/2.

ImplementationApproach

To allay the fears that individual performance would be judged on the basis of inspection error lists, it was decided that all statistics would be kept within the group for the first month, at which point a meeting of all the interested parties would be held to resolve issues, such as use of statistics, and to review progress. It was agreed that at any sign of a major adverse effect on team morale inspections would be discontinued.

Before any inspections started, a presentation describing the process and its objectives was given to the team and its managers. Shortly afterward, lists of exit criteria and error checklists for the detailed design and code inspections were distributed to all team members. It was decided to implement inspections at the current position without any backtracking. This meant that inspections for overviews of system components were not held, nor were some initial design inspections.

It was apparent early that a flexible approach would have to be adopted, because the ground rules laid down by Fagan* were derived from a systems software environment, an environment that differs markedly in some respects from the applications environment. An example of the sort of adjustments that had to be made was the transfer of test plan inspections to code inspection time because it was found that examining a series of test cases with supporting data was more easily done in parallel with the inspection of logic.

*M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Volume 15, Number 3, 1976. Reprints are available from IBM under order number G321-5033.

Later, it was decided to inspect test situations (without the data) at detailed design inspection time as well when the situations and the design specs acted as a mutual check on the completeness of one another. This approach proved successful and was adopted for the rest of the system.

The selection of attendees at the inspections was the responsibility of the team leader. The whole team attended the first design inspection for each program for education purposes: an initial design inspection where possible, or where this had been missed, the detailed design inspection. Thereafter, usually three people from the team attended – typically, designer, coder, and team leader, with the moderator bringing the number to four. However, this arrangement was changed whenever necessary; for instance, for the most important part of the system, or where an interface involving more people was being inspected. No problems with the size of inspection meetings were encountered, although six people should be an upper limit while four is an ideal number, promoting maximum interaction with minimum communication problems.

Scheduling of inspections was also done by negotiation with the team leader, who was in the best position to know when work would be ready. When a program required several inspections, major branches of the hierarchical structure were chosen as material for each meeting, at first using the working rates given as guides by Fagan and later drawing upon experience. The moderator usually arranged for the meeting room.

The conduct of the inspections quickly settled down into a routine with all participants knowing what was expected of them. The meeting would start with the moderator recording the times spent by individuals in preparation, and then the reader would start paraphrasing the material to the rest of the team, while they followed from the documentation. As errors came to light, the moderator recorded these briefly, while the rest of the team proceeded. These brief notes were read back at the end of the meeting for the agreement of the team, and were expanded and classified on a formal error list afterwards. It took several meetings for the mechanics of the error discovery process to become familiar to the team members; but once learned, the meetings proceeded more quickly and smoothly, and it was noticeable that the lessons learned from the error discovery process were being applied as new material was submitted – in fact, a learning process by feedback was taking place.

Sometimes team members would, in inspection meetings, disagree with various aspects of the project. The resulting discussion either clarified the situation for everyone or showed that further investigation was necessary. However, not all such discussions were as immediately productive, because some time was wasted over differing interpretations of standards. The project standards manual, while perfectly adequate for the normal working environment, was still not as complete or as formally defined as necessary for development using the inspections process.

The team in general accepted inspections quite readily, although accepted work practices were somewhat changed by it. What this meant in practice was that peak project loading was moved from the end of the development phase to the early part.

On a personal level, the inspections environment demands an attitude to one's work radically different from that of the traditional programmer. Whereas programmers have in the past been secretive about their work and defensive of their style, inspections expose their work to the critical scrutiny of their colleagues. Although one person on the project reacted defensively to the identification of errors in that person's work, not only did this team member's work come successfully through the inspection process but the individual became more comfortable with the process.

This experience demonstrates that while not everybody is necessarily ideally suited by temperament to the inspection process, this need not preclude the use of the technique. Tact and patience is required of the team, whoever's work is being inspected, and it is probably true to say that increased team spirit and cooperation was the result in this group.

Results

The results of the pilot project will be discussed in relation to the stated objectives: to verify the benefits, and the acceptability of inspections to the project personnel. In addition, some statistics will be presented that quantify basic parameters of the inspection process in the project environment, and could be useful to anyone implementing inspections for the first time. The benefits to be verified were quality and productivity, and increased professionalism of team members. Quality was further subdivided into reliability, modifiability, and predictability (a measure of how the project meets its budget, its schedule, and its expectations).

Professionalism

Within the inspections environment there is a positive feedback mechanism for both authors and inspectors. To demonstrate this, errors for each designer and programmer, found during detailed design and code inspections and normalized to error rates per thousand lines of code (errors/KLOC), were plotted against the calendar date of the relevant meeting (see Figure G1). Then curves were drawn through the plotted design and code points for each individual. Unconnected points on Figure G1 represent the error rate for individuals who submitted only one piece of design or code.

The curves shown represent a downward trend in the error rate made in both design and coding operations. Because a uniform postinspection standard was achieved for all programs, this trend represents an increase in technical competence of the individual; an increased awareness of the errors one is prone to make results in a positive improvement in programming efficiency. No attempt is made to quantify this increase because of the varying complexities of programs and experience of programmers, but the trend shown over a period of six weeks during the coding phase is very encouraging for such a short period, and suggests that inspections provide a powerful learning mechanism.

This effect is apparently not confined to authors, because individuals who submitted only one piece of work near the end of the coding phase (represented by the unconnected points on the graph) had results that followed the trend indicated by the curves, which suggests that by acting as inspectors they benefited vicariously from the inspection results of those who had gone before.

When this subject was discussed with the team members, they all felt they had benefited from the learning mechanism, but the more experienced people less than the others. Some people also felt that the discipline inherent in the inspections process forced them to concentrate more on detail checking, to the benefit of their work. All team members felt that better communications throughout the team were a very real benefit, with everyone conversant with all parts of the system. This means that backup and maintenance problems are eased.

On the more personal level of job satisfaction derived from this method of working, it was apparent that the involvement in other inspections meant more variety of work.

Reliability

Reliability in software is a deceptively simple quality that is not only a function of the number of errors found in system tests, or in the field, but also has connotations of trust and confidence. Thus any discussion of reliability should not be confined to error statistics, but should also include the more personal or abstract considerations of those responsible for the work. A measure of reliability, therefore, results not only from the removal of errors, but from a feeling of assurance that all errors have been removed.

Figure G2 shows the error rates experienced, and the hours of effort required to find them. It is interesting to note that during the design phase the majority of errors are of the "missing" category, while the majority made during coding are in the "wrong" category. It is encouraging that missing function is detected at the design stage, because missing function is traditionally the most difficult to fix after coding is complete.

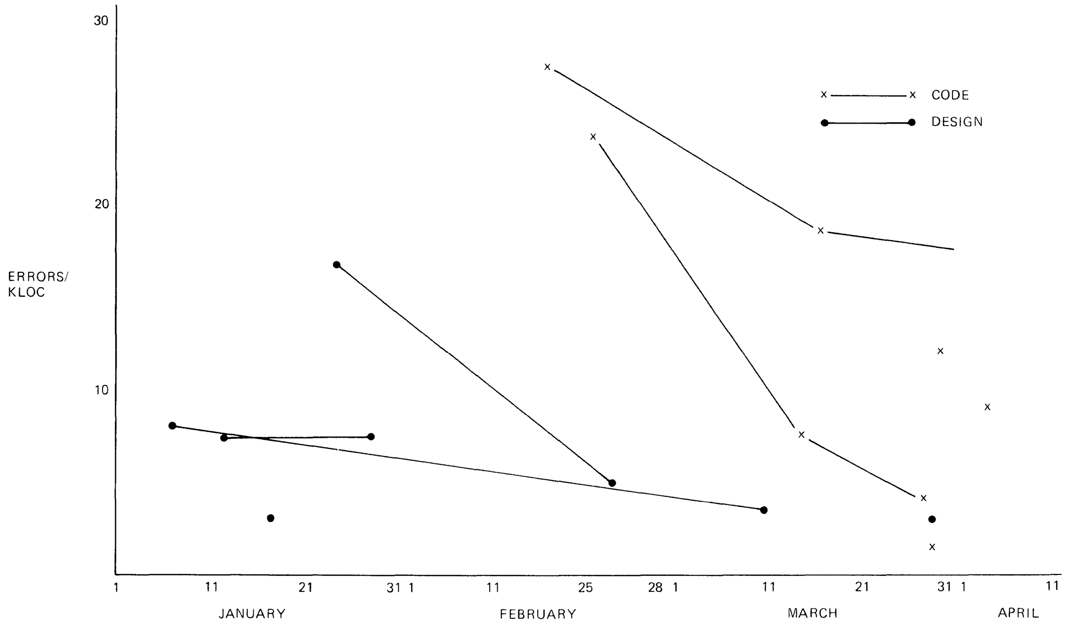


Figure G1. Individual error rates during development for designers and programmers

OPERATION	ERRORS						TIME				LOC.	SUMMARY					
	Major			Minor			Team		Mod			Errors		Time		Error detection effort	
	M	W	E	M	W	E	Prep.	Insp.	Prep.	Insp.		Maj.	Tot.	Team	Total	Hrs./maj. errs.	Hrs./all errs.
Initial design	0	2	0	5	11	0	9½	23	5¼	5¾	2387	2	18	32½	43½	21.75	2.4
Detailed design	28	11	1	82	53	15	62¼	87	23¾	20¾	5630	40	190	149¼	193¾	4.8	1.0
Code	40	50	4	39	99	9	140¼	96¼	43¼	25¼	5711	94	241	236½	305	3.25	1.25
Error Classification:	M = Missing W = Wrong E = Extra																

Figure G2. Error rates and detection effort

The greater error rate at the code stage is probably due to the differing characteristics of the design and code targets. Design is read by a person, while code is read by a compiler – the compiler has no imagination while a person can guess intent. The human ability to interpolate, combined with the expressiveness and flexibility of the English language, means that detail and function are fairly easily omitted from the design. On the other hand, the inflexibility of a compiler and its rigid syntax rules mean that it is much easier to make a mistake in expressing an intent.

Consistency in reliability is an important factor, on the principle that a chain is only as strong as its weakest link. The inspections process aims for uniformly high quality. To test the uniformity of the quality, the number of errors found in each program during unit/integration test was recorded and plotted on a graph against time. The plot was of an error rate, normalized to errors per thousand lines of code (KLOC), against the calendar date of the last code inspection for that program. The results are shown in Figure G3, with a "least squares fit" relation plotted. The slight downward trend of this curve indicates that no more errors were detected during the later stages of unit test than in the earlier stages. The conclusion drawn from this is that, similarly, no more errors remained at the end of the inspection process for the later programs than the earlier ones and that, at least, the process was consistent in the quality of output, and possibly with a slight increase in quality with time.

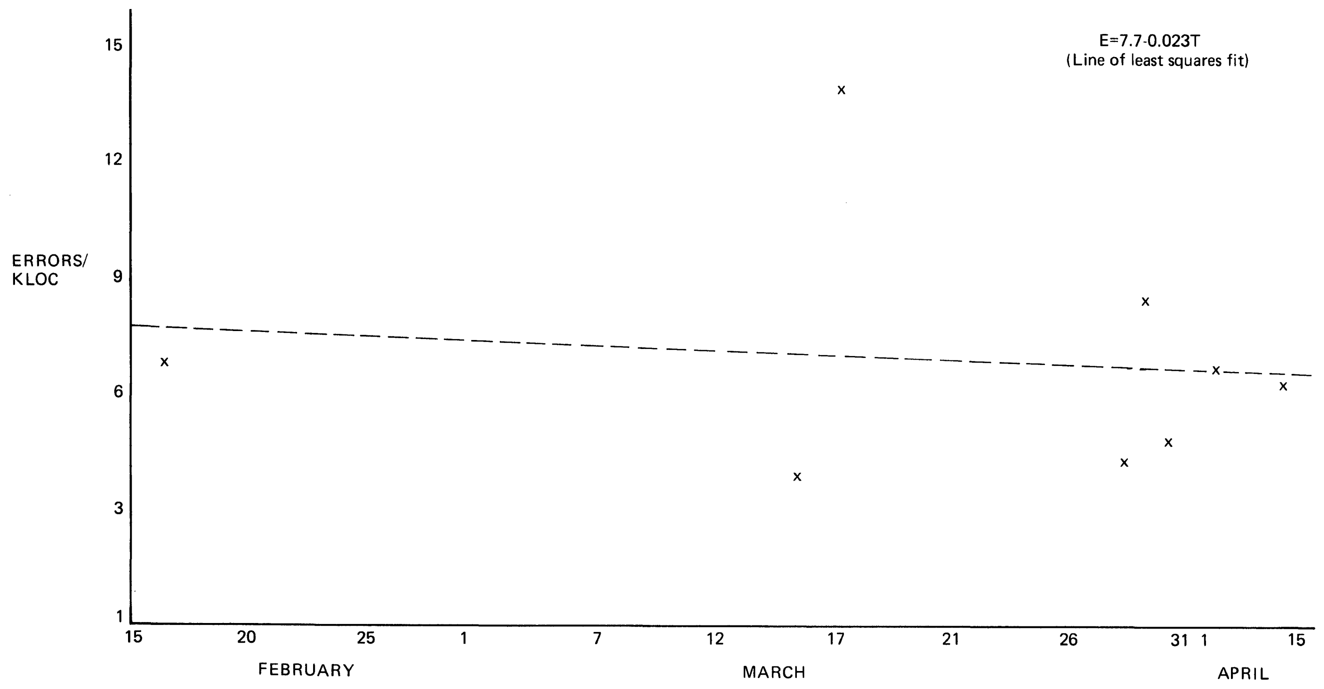


Figure G3. Errors remaining after code inspection and found during unit test

The team felt a little doubtful that all parts of the system would prove equally reliable, but believed they knew where problems might arise. This was a reaction based on their knowledge of the complexities of the programs.

Modifiability and Predictability
The ease with which program function can be changed or added to is the quality of modifiability and is largely a subjective matter (though if the principles of structured design* are strictly followed, it is possible to derive a quantitative measure). Very few changes involving more than a line or two of code were made at any time during development, but where this had to be done the programmers involved found that the affected areas were easily isolated, and the changes could be quickly made. Although this was not a direct consequence of the inspection process, it was one of the aims of the detailed design inspection meetings to achieve a "good" design. Where this objective could be said to have been missed was in the size of modules: in several cases, modules of up to 200 statements were recorded. The contrast between those and the more usual one-page modules was very evident during code inspection meetings and provided convincing support for the "small module" argument.

Predictability, like modifiability, is largely subjective. The managers directly involved were worried by the steadily rising overtime figures in the middle of the development phase. However, during the latter half of detailed design, the overtime figures steadily declined to zero (instead of, as is usual, rising to a peak at system test). The hump was due to a transition from design to code (when both activities were being carried out concurrently) and the absence of the moderator for a week, which tended to compress activities on either side. The programs all met systems test start date, and were fully unit/integration tested (except for one program not on the critical path) in an unstressed atmosphere, so the project can be said to have met its schedule. The manual project tracking system used by the team leader employed exits from inspections as milestones.

Comparison with Another Project
A similar project was selected for the purpose of comparing reliability by means of system test results. The project (project X) used improved programming technology techniques throughout, apart from the chief programmer team concept. Except that project X used walk-throughs instead of inspections, implementation was very similar to PRS. As can be seen from Figure G4, which summarizes the two projects, there is a significant improvement in the test error rate for the project using inspections, while the coding rate is similar. The coding rate is for the total development process (design, code, and test), and the PRS figure includes the moderator's time, approximately 10%. It would appear that an error frequency of only 20% of that of previous products can be expected for no increase in manpower. Project X is a highly regarded system, remarkable for rigorous development and testing. The fact that PRS was able to improve on its quality is an indication of the effectiveness of the inspections technique.

	PROJECT X	PRS
Improved Programming Technologies	Yes	Yes
Reviews	Walk-throughs	Inspections
Number of statements	10,000	6,250
Total detail design, code, and test personnel	64 person months	41 person months
Duration	14 months	7 months
System test errors	51	11
Pilot installation errors	26	0 (also 0 errors in first 6 months of operation)
	—	—
	87	11
Coding rate (LOC/ person months)	155	153
Test error rate (errors/KLOC)	8.7	1.76

Figure G4. Summarizing project X and PRS

Working Rates
Figure G5 shows the working rates achieved by the PRS team for the two basic types of inspections: detailed design and code inspections. As will be seen, code inspections proceeded much more slowly than design, especially in preparation. This possibly reflects the relatively large size of some modules; it is likely that rates would improve if module size were restricted to a page of printout.

*W. P. Stevens, G. J. Myers, and L. C. Constantine, "Structured Design", IBM Systems Journal, Volume 13, Number 2, 1974.

Phase	Lines of code/hour	
	Preparation	Inspection
Design	275	275
Code	125	225

Figure G5. Working rates experienced by PRS team during detailed design and code inspections

Productivity

Although the moderator represents an additional overhead to the system development process, probably in the order of 10% in a fully established inspections environment, Fagan claims that there is a *net* productivity gain, and this seems to be borne out by the PRS project experience. However, it is likely that the ultimate justification will be in the reduced maintenance overhead from shipping a higher quality product. In practical terms, this means that making the effort earlier makes things easier later.

The following is an estimate of the productivity gain derived from the use of the inspections technique in the PRS project. It will be seen that much of the gain resulted from errors being detected in an earlier stage of development rather than during unit, system, or field test. The savings estimates are based on the following assumptions:

1. PRS will be released to the field with one valid APAR per 1000 lines of code (a total of 6 APARS since program size is approximately 6000 lines of code).
2. To compare the errors found in the various PRS inspections with errors in a project in which inspections were not used, assume that the errors in the noninspections project would be found in the unit, system, and field test in the same proportion as they were found during the PRS project. With this assumption, the error figures break down to:

Implementation	In-spec-tion	Unit test	Sys test	Fld. test	Tot.
Using Inspections	136	41	11	6	194
Without Inspections	-	138	37*	19	194

*This figure agrees fairly well with what would be expected from project X experience (see Figure G4). The project X figures scaled to PRS size would lead one to expect in the order of 32 system test errors.

3. Approximate average error correction times are:
   - Inspection errors (I) – 1 hour
   - Unit test errors (U) – 3 hours
   - System test errors (S) – 7 hours
   - Field test errors (F) – 20 hours

If the average time to fix errors at each stage are I, U, S, F respectively, then the total saving due to inspections is:

```
  138U   +  37S   +  19F
— 41U    +  11S   +   6F    + 136I
  97U    +  26S   +  13F    - 136I
  97(3)  +  26(7) +  13(20) - 136(1)
```

Thus, the total estimated saving is 597 person-hours, or 17 person-weeks, assuming a 36-hour work week. Since the project required 41 person-months overall, this 17 person-weeks saving translates into a net savings of 9% due to inspections.

Conclusions

The disciplined team approach to verifying quality brought about by inspections has resulted in the following benefits for the PRS team.

- A high standard of quality in the finished product that compares favorably with comparable projects using improved programming technology techniques including walk-throughs.
- A satisfying working environment for team members.
- A net productivity rate comparable to similar projects. The overhead (10% maximum) added by the moderator is more than compensated for either by increased productivity or by reduced maintenance.

AdditionalInformation

Figures G6, G7, and G8 show code inspection and design inspection criteria used by the project as well as a sample inspection error list.

1. Module prologue must be complete and up to date.
2. Code must be structured, and formatted to reflect the structure.
3. Code must be sufficiently commented upon.
4. All project standards and conventions must be followed. The compile must be error-free, with no messages with a severity level greater than "warning".
5. The listing should include cross-reference listing and aggregate list.
6. The code must reflect the current design specification.

Figure G6. Code inspection exit criteria

1. Completed specifications must represent design to the approximate range of 3 - 10 source language lines of code per design process statement.
2. Design specifications must be structured (all applicable structured programming rules are to be followed).
3. For design specifications covering modified modules, all new/changed statements should be flagged with release identification.
4. References to data areas should be by field name. Specific values tested or set should be specified.
5. Linkages to external routines/modules must be defined. Parameter values passed and return codes are specified.
6. Messages issued by modules must be defined, with text and identification numbers.
7. Any design changes to high-level design since high-level design exit must be included in the current design specifications (high-level and low-level design materials must match).
8. All design documents submitted for inspection should conform to local or project standards.

Figure G7. Detailed design inspection exit criteria

FUNCTION PAXX-12(#1)	DATE 5/4/77

Module	Classification			Error Description
	Type	M/W/E	Min/Maj	
PAXX	DA	W	Maj	PCB address parameters in wrong sequence.
	MN	W	Min	I/O areas should be declared before the structures based on them.
	DA	W	Min	HEADSH(5) should contain CEN02 to match LINED; which should also refer to CEN02 instead of CEN01.
	DA	W	Min	Security lines should have ASA char. of 'c'.
	MN	W	Min	Blocks of DCL's should be separated by comments.
	DA	W	Maj	Base PICCONV on CHARCONV.
	LO	W	Maj	TRAIL(6) is written to the wrong file.
PAXX201	–	–	–	Comments on 1st occurrences of CALL's.
	TP	W	Min	Message in case A should read ". . . will not . . .".
PAXX302	DA	W	Min	Message does not refer to correct card type, when C07 card is not present.
	LO	W	Maj	A recycle # of '0X' is treated incorrectly as valid. Should be included in test plan.
PAXX202	LO	W	Maj	By-name assignment into KEYN will not always work for PIC fields.
	MN	W	Min	'R03' should be qualified by FPAOB not A.
	DA	W	Min	Positioning wrong when SUBSTRinging into MSG (III)

Figure G8. Sample error list

Technical Newsletter

This Newsletter No.	GN20-3814
Date	August 15, 1978
Base Publication No.	GC20-2000-0
File No.	
Previous Newsletters	None

Inspections in Application Development
Introduction and Implementation
Guidelines
Installation Management Manual

This Technical Newsletter provides replacement pages for the subject manual to substitute a structured program for the original one, and to add a report on the use of inspection in an application development project within IBM. Pages to be inserted and/or removed are listed below.

Inside Front Cover, Preface
iii, iv
23 - 36 (33.1 - 33.3 added)
61 - 68
Reader's Comments Form (added)

A vertical rule in the left margin indicates a change. Absence of a vertical rule on a page bearing a 'revised' notice means only that existing copy has been moved or that a minor typographical error has been corrected.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Technical Publications, Dept. 824, 1133 Westchester Ave., White Plains, N.Y. 10604

Printed in U.S.A.

Inspections in Application Development Introduction and Implementation Guidelines Installation Management Manual

GC20-2000-0

READER'S COMMENT FORM

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

What is your occupation?____

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

GC20-2000-0

Reader's Comment Form

Cut or Fold Along Line

Fold and tape   **Please Do Not Staple**   Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 824
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape   **Please Do Not Staple**   Fold and tape

Inspections in Application Development Introduction and Implementation Guidelines Installation Management Manual Printed in U.S.A. GC20-2000-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

GC20-2000-0

Installation Management Manual Inspections in Application Development – Introduction and Implementation Guidelines GC20-2000-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601