A process-integrated approach to defect prevention

by C. L. Jones

Recent efforts to improve quality in software have concentrated on defect detection. This paper presents a programming process methodology for using causal analysis and feedback as a means for achieving quality improvements and ultimately defect prevention. The methodology emphasizes effective utilization of all error data to prevent the recurrence of defects.

Historically, formalized programming development processes have been defined in terms of defect detection.[1] In general, a stage of the process was considered to be complete when all inspection steps or tests were done. At that point, the next stage of the process could officially begin. As these processes evolved, they became more comprehensive through the introduction of entry and exit criteria and clearly documented definitions of each stage of the process.[2] However, the emphasis for quality has centered, for the most part, on the *defect detection* process.

With the advent of software engineering, *defect prevention*, rather than defect detection, has become a concept of interest. Although the quality concepts which advocate that we should "do it right the first time" have been introduced to programmers, the concepts have not been incorporated adequately into our processes.

Some defect prevention already exists in the way we do business today. The very fact that we *have* documented and clearly defined processes is, in itself, a prevention technique. The existence of documented internal standards and methodologies allows us to enhance our techniques over time rather than forcing us to invent new methods with each iteration. Education classes, tools, and quality improvement teams are also examples of prevention techniques. Any mechanism that helps to prepare the programmer or that automates error-prone tasks is a defect preventer. But even though we already do much for defect prevention, more can be done. A vast amount of information available to us is not used.

Some analysis that takes advantage of our data by trying to determine error rate trends has been done.[3] Statistics are being used to point out error-prone modules, components, or process stages. These analyses are very useful; however, they must be augmented with an in-depth study of the errors themselves and what causes them if we are to take full advantage of our data. Field problems, test problems, inspection errors, and design changes can all be evaluated in a more comprehensive way to allow us both to understand the causes of errors and to put action plans in place to remove error-causing situations. The purpose of this paper is to present a methodology for performing causal analysis as part of the programming process.

The paper presents the philosophy behind this methodology, followed by specific enhancements which

should be made to the existing process. Details are included about ways to implement causal analysis. Finally, some remarks are offered on the cost as well as the results of the new methodology.

Goals in defining the methodology

The quality improvement teams (QITs)[4] that were widely established several years ago basically operated in a subjective mode. That is, the team would brainstorm about what problems existed and would then try to resolve them. The approach worked for a while but gradually died out in many areas. There seemed to be two main reasons why this occurred. First, the suggested actions very often were not implemented because of resource constraints. Second, people had a limited set of opinions and quickly ran out of discussion items. Subjectively identifying problems did not take full advantage of the data at our disposal. In contrast, one of the goals for causal analysis is to provide for *a systematic way to identify problem areas.*

When problem analyses take place, recommendations result.

History also demonstrates that if problem analysis is done on an *ad hoc* basis, and if resources are not specifically allocated for analysis, the analysis work will be done in a haphazard way or it will fade away altogether. Yet, when an analysis is part of the process itself, it tends to continue on a regular basis regardless of the specific personnel involved. Therefore, *data should be analyzed as part of the process* and not as a stand-alone activity.

When problem analyses take place, recommendations usually result. Often, however, the recommendations do not receive attention and are not enacted. A popular sentiment among developers seems to be "There are lots of good ideas floating around, but we cannot seem to keep up with them all if our main mission is to produce products." But it is also an ineffective way of operating. *A mechanism for collecting suggestions is necessary* to prevent ideas from becoming lost.

If a collection mechanism for suggestions is provided, an effective way of implementing those suggestions must also be provided. As mentioned earlier, QITs often fail because resources are not available to implement the ideas. *An implementation plan is critical* to a successful quality approach.

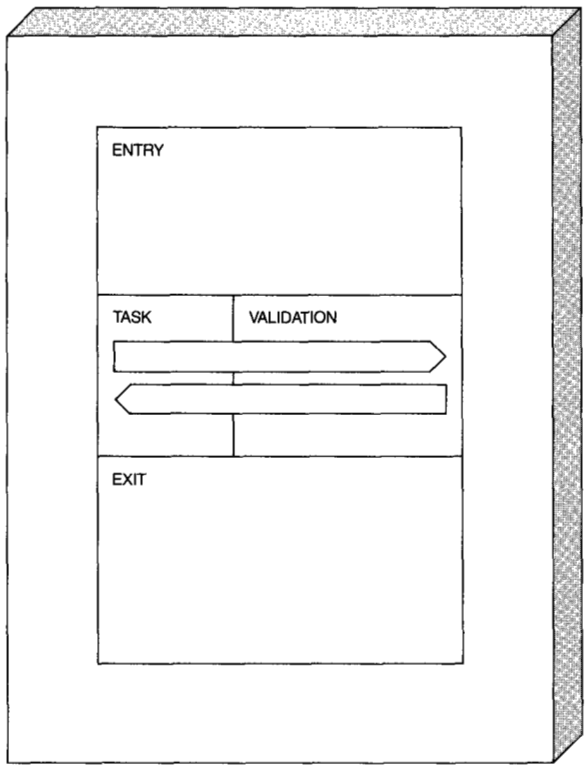
Finally, the feedback mechanism itself needs to be defined in detail. Questions arise as to who should do each of the activities, in what order, and when. The goal is to *clearly define the feedback process.*

Basic concepts

The defect prevention methodology is based on three concepts:

1. *Programmers should evaluate their own errors.* Programmers themselves should be involved in the analysis of errors because
   - The best person to determine the cause of error is the person who made it. Having those people involved provides for a more accurate assessment of what really happened in creating the error. There are subtle reasons for making some errors, and what may appear to an observer to be the cause may not be remotely related to the real cause.
   - There is also a difference in the interest that people take in their own errors. It is a natural tendency for people, when seeing someone else's error, to assume that they would never make that error themselves, and therefore they do not take such an error seriously. But when it is their own error, they become much more interested in discussing how to prevent it. They are more likely to suggest meaningful ideas because they understand the error more thoroughly.
   - People also benefit from the immediacy of the feedback. Instead of seeing data months after the fact, they see it right after they have made the error.
   - If one person is assigned to do the causal analysis for an entire project or organization, he or she will soon suffer burnout. Moreover, an individual can bring only a single perspective to the problems, thus limiting creativity in determining corrective actions.
2. *Causal analysis should be part of the process.* As mentioned earlier, if the activities for problem

Figure 1 A normal process stage



analysis are not established as part of the process, they are subject to haphazard implementation. Making causal analysis part of the process spreads the work, involves the appropriate people, and creates an environment that emphasizes prevention of errors as a daily activity. As a result, the programmers realize that quality is everyone's day-to-day responsibility.

3. *Feedback should be part of the process.* Even though programming processes have been used for quite some time, detected errors and statistics have seldom been used as feedback to improve those processes. A "looping" effect for the detection and correction of errors has been defined, but an equivalent feedback of data for improvements to the process or for the education of programmers has not been fully exploited. Errors and statistics should be made visible to the programmers, thus allowing them to refine their techniques and to learn from their own mistakes. "Feedback" is a key consideration in the methodology described here. The goal is to change the process from one where a single programmer learns from only his or her own mistakes to one where all programmers learn from every mistake.

Enhancements to the process for defect prevention

Defect prevention is established both by implementing a feedback mechanism which encourages fine-tuning of the process and by augmenting the activities that occur in each stage of the process.

Although the process stages themselves remain the same with the new methodology, additional activities are needed to facilitate defect prevention, such as kickoff meetings, causal analysis meetings, and Action Team follow-up. Details of these meetings and activities of the Action Team are given in the appendices. The process enhancements are discussed in this section.

Figure 1 illustrates a typical stage of the development process, such as module level design or code.[2] In the ENTRY substage, the input to the stage is evaluated for completeness. The TASK substage represents the work being done for the stage, VALIDATION includes verification of the quality of the work product, and EXIT ensures that all activities have been completed before work may begin on the next stage. The arrangement of the TASK and VALIDATION substages illustrates the looping between rework and verification until all detected errors have been corrected.

ENTRY substage—Add kickoff meeting. The defect prevention methodology adds a formal kickoff meeting to the ENTRY substage. At the beginning of a stage, the team assigned to the unit of work meets together with the following objectives:

- *Review what is available as input.* The input to the stage is evaluated to make sure that all members of the team understand what is available and whether or not everything from the previous stage is complete.

- *Review process/methodology guidelines.* The team discusses the output requirements for the stage, going over examples, guidelines, and explicit expectations of the team leader(s). Examples are extremely useful in eliminating contradictory interpretations of guidelines or descriptions of what the output should be. Sample outputs help to clarify exactly what is expected. Guidelines and standardized kickoff packages should be available for each stage to ensure consistency between teams

and to provide documentation that can be incrementally improved rather than constantly reinvented.

- *Review error checklists.* The most common error causes for a particular stage are reviewed from a checklist. This review increases awareness, which is critical to defect prevention. If people can be reminded of the most common causes for errors made in a task just before they start to perform that task, they are less likely to repeat the errors. Checklists should be available for each stage and they should be updated constantly.

- *Set team goals.* Normally, quality goals will be set for a product for each stage of the process and will be documented in a quality plan. These goals will generally take the form of expected error detection rates. There is an inherent conflict between the desire to detect and remove a high number of errors and the desire to prevent their creation, thus resulting in a low number of errors. However, an expected rate for the release which takes both perspectives into consideration and provides a realistic estimate for the release given the process being used should be established.

  At the kickoff meeting, the team should establish its own team goals, thereby increasing the psychological commitment of the individual members to the quality of the team's work. The team goals are never published. They are simply kept informally. After each person on the team is polled for an opinion, a group discussion is held to project the number of errors for the stage. This discussion helps to emphasize quality rather than schedules or other pressures.

VALIDATION substage—Add preliminary causal analysis. Normally inspections are conducted as part of the VALIDATION substage to ensure that all work has been done properly. The inspection data is entered into a data base.

The enhancement to this substage is to have the defect description added to the data base, and as rework occurs, to add defect resolutions as well as preliminary causal analysis. Since rework is usually done by the person who created the error, he or she can provide some additional insight about the real cause for the error. He or she can also determine the category of error, the originating stage, and some suggestions for prevention. This helps the causal analysis team later when they are trying to determine specific preventive actions.

EXIT substage—Add causal analysis meeting. The EXIT substage is normally concerned with making sure that all required work is complete. The defect prevention methodology incorporates in the EXIT substage a formal causal analysis meeting to do the following:

- *Analyze defects.* The causal analysis meeting itself is a brainstorming session to produce not only creative actions for individual problems but also comprehensive actions for recurring groups of problems. During the sessions, all team members learn from the errors of other team members. In this way, the entire team can create an action plan and feel a real closure to the stage.
- *Evaluate results versus team goals.* The results of the errors actually detected are compared in this meeting with the goals set in the ENTRY substage. This enables the team to understand how it did in relation to the product or release goals and to its own goal. The comparison triggers discussions on how to improve future work.
- *Process stage evaluation.* After all causal analysis is complete, the team should discuss general process improvements, such as how the inspections could be improved, what tools would be useful, or what positive actions were taken during the stage that should be added to the process or kickoff packages.

Questions for each defect

Answers to the following questions are collected for each error:

- What stage originated the error?
- Category of cause of error?
  - Communications
  - Education
    - New functions
    - Old functions
    - Other
  - Oversight
  - Transcription
- How was the error introduced? What caused the error?
- How could it have been avoided?
- What corrective actions are recommended?

The purpose of these questions is to collect only the essential information. Too many questions become a burden to the causal analysis team; too few undermine later evaluation and follow-up to action plans.

Questions about how the process could be improved to *detect* the error are intentionally delayed until the end of the causal analysis meeting. During causal analysis meetings programmers tend to focus on how

The causal analysis meeting should follow an agenda.

an error should have been found during the inspection or test, rather than dealing with the real cause of the problem. Causal analysis should emphasize *prevention* of errors rather than *detection*. Therefore, the causal analysis meeting should follow an agenda such as the example in Appendix B. In this agenda, the meeting consists of a causal analysis portion followed by a stage evaluation portion. The analysis portion helps to emphasize preventive actions. The evaluation portion helps to identify early detection issues.

Following is the rationale behind each of the questions:

- *Originating stage:* The stage where the error originated is noted in order to allow errors to be directed to the appropriate groups for analysis. For errors originating in previous stages, the present team may not be able to determine adequately the actual cause or an appropriate preventive action. In this case, the error is routed to the team that actually created it.
- *Category of cause of error:* Many lists are available for categorizing errors.[3] Their purpose should be to aid in determining how to prevent the error. If, as in this methodology, each error is being evaluated individually and corrective actions occur in direct response to the error, there is no reason to have an elaborate list of categories or types of problems. Categories should be chosen to help the causal analysis team to think of corrective actions. For example, if the programmer classifies his/her error as a communications problem, the team has a better insight into the real cause of the error and can better recommend corrective actions. Four categories describe any error:
  1. *Communications* errors result from a breakdown in communication between groups or among team members. For example, a high-level designer may state a concept that is misinterpreted by a low-level designer.
  2. *Education* errors occur when a team member's failure to understand something causes the error. Education errors are further divided into the following:
     - New function—the programmer does not understand the new function and therefore makes an error (e.g., misunderstands the use of a bit).
     - Old function—the base code or function is not understood, and when new function is added to it, the placement or implementation causes problems (e.g., the programmer does not understand the base code well enough to know that the placement of new function causes regression problems).
     - Other—the programmer needs education in a subject other than the function being developed (e.g., compiler knowledge).
  3. An *oversight* arises when all of the possible cases or conditions are not considered or handled (e.g., an error condition is missed).
  4. A *transcription* error occurs when the programmer knows what to do and thoroughly understands the item, but for some reason simply makes a mistake (e.g., types in the wrong label).
- *What caused the error:* The purpose of this question is to identify the exact cause of the error before trying to put an action in place. Causes vary enormously from error to error. Causes such as inadequate schedule time, too many interruptions, or too little tools support often occur when least expected. Because causes are not obvious from the error, it is very important to have the creator of the error do the causal analysis, if at all possible.
- *How to avoid the error:* This item calls for a suggestion that would prevent the error from occurring in the future. For example, in a situation where an education session is needed, the avoidance suggestion might be to establish a class to address the topic that is confusing.
- *Recommended corrective action:* Whereas the preceding question asks for a general suggestion, this question requests specific plans to correct the problem. For example, the corrective action to establish a needed class would be to assign the task to a specific person to determine who should teach the class, determine who should attend, establish a location, send out notices, and handle all other

administrative details. This is quite different from the general suggestion that a class is needed. The action question focuses on *how* to get the solution implemented.

Action Team

When programmers make suggestions but nothing happens to implement them, they are less inclined to continue making the suggestions. Without someone assigned to follow through on ideas, many suggestions fail to be implemented. This failure is not only discouraging to the programmers but is also an ineffective way to operate. Therefore, this methodology suggests the formation of an Action Team to respond to suggestions for improvements. The size of the team depends on the number of recommended actions deemed appropriate for implementation. In addition, a manager should be responsible for the team, to prioritize work and to provide management focus. It is his or her responsibility to make sure that the work gets done and is visible.

An Action Team which will respond to suggestions for improvements is recommended.

The Action Team should consist of one or more persons who can handle suggestions in the areas of process, education, and tools. These three areas correspond to the normal support functions that must occur for any program development group. Actions that need to be implemented in areas other than process, education, or tools can be directed to other people in the development organization. The final resolution of the action, however, is still the responsibility of the Action Team.

The Action Team, then, has the following responsibilities:

- Prioritization of all action items.
- Status tracking of all action items.
- Implementation of all action items.
- Dissemination of feedback. Feedback can take many forms, such as classes, seminars, process document updates, process management technique memos, newsletters, tools, guidelines for product-specific internals, kickoff packages, error lists, etc.
- Data base administration. A tool and an associated data base are needed to assist in the collection and manipulation of the defect prevention data. The tool should handle data on both defects and actions, since data for the two are logically associated.
- Generic analysis. Causal analysis in this methodology emphasizes the investigation of every error. When causal analysis teams focus on errors, they may not be exposed to a global range of problems across an entire organization. It is the responsibility of the Action Team to evaluate the defect data periodically to see if trends may indicate problems which need to be addressed and which the individual analysis teams have not noticed.
- Visibility of success stories, recognition. The Action Team should make success stories visible to the entire organization and should identify for recognition all effective solutions.

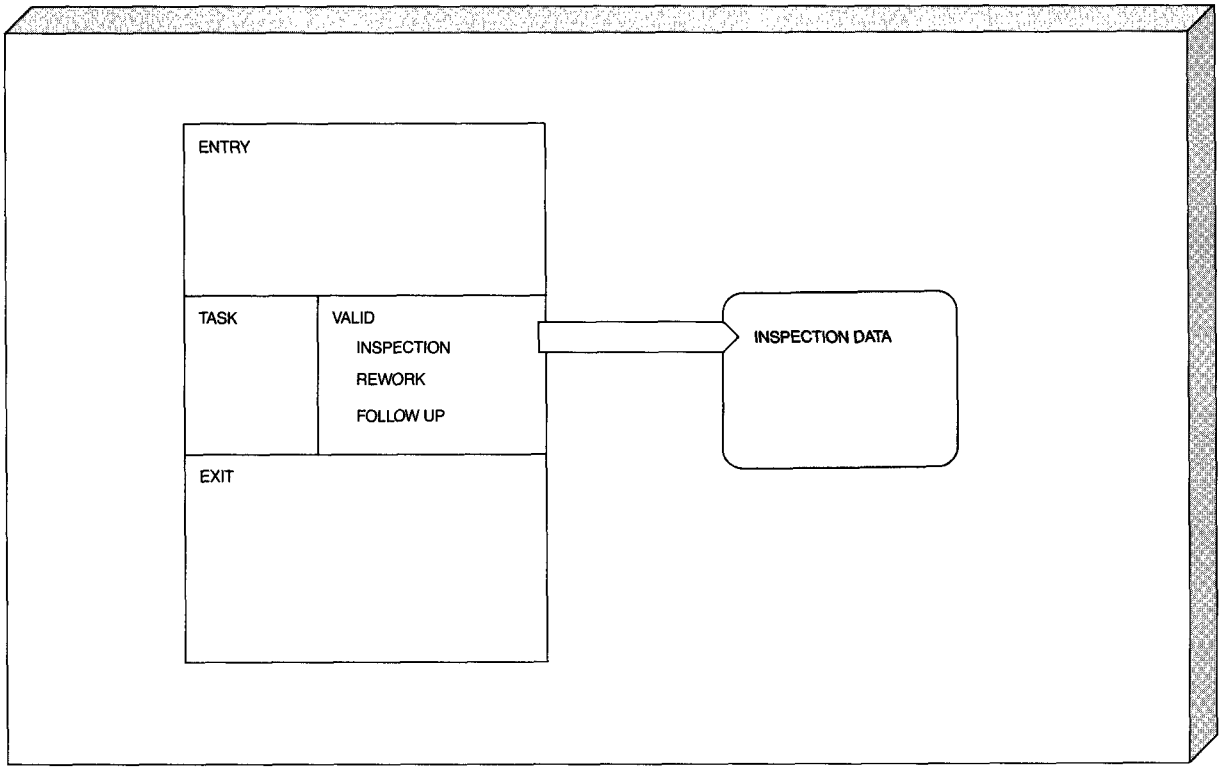
Defect prevention methodology

Figures 2 through 5 help describe the flow through the process stages. Those stages which utilize inspections are discussed first, with additional stages introduced later in this series of charts.

Figure 2 shows the process stage before the addition of the defect prevention enhancements described in this paper. Inspection data are collected and entered into a data base following the inspection meeting. Rework and Follow-up are tasks that resolve the problems found in the inspection and verify that resolutions are correct.

Figure 3 shows the addition of a kickoff meeting during the ENTRY substage. Inspections are held, as usual, after the TASK substage work is complete. Following the inspection meeting(s), inspection data and defect data are entered into a data base. As rework progresses, the errors are analyzed by the programmers on an individual basis as each error is corrected. The cause of the error and the programmer's suggestion of how to avoid the error are added to the data base. Following rework, the team holds a Causal Analysis Review meeting. At this meeting the team completes the causal analysis of all errors

Figure 2 A process stage without defect prevention



and produces a set of actions aimed at preventing errors in the future. These actions are then entered into an action data base. At this point, the stage is considered complete.

In Figure 4, the Action Team has been added. The Action Team holds regular meetings to set priorities, make new assignments, and check status. The team is aware of new actions entered into the data base and responds by assigning each action to an Action Team member for resolution. As actions are implemented, the Action Team closes the entries in the data base and provides feedback to the rest of the development organization via the most effective feedback technique for that action.

There are additional data that can be analyzed and that do not result from an inspection: specifically test problems, field problems, design changes, and QIT and miscellaneous suggestions. Ordinarily, test data and design changes are handled the same as inspection data and are analyzed at the end of the normal test stage. Special causal analysis meetings may, however, be required for some data, as shown in Figure 5. For example, a Functional Verification Test team may evaluate their operational errors and those errors which were not detected by their test but were found in a later test. This type of error analysis is done to determine how the test might be run more effectively or how test cases could be written better.

Field errors are received constantly through the entire development cycle for inclusion into the latest release of the product. Since there is a lag time involved with these problems, the person who created the problem may not be available, and the current owner of the code may be the best person to determine causes and suggest preventions. A special causal analysis meeting is held periodically to evaluate these errors.

There are many causal analysis sessions that can be performed in addition to those in the main path development cycle. Whenever possible, the errors should be reviewed during an EXIT causal analysis

session, but some additional work may be necessary for data received at irregular intervals.

Cost of implementing this methodology

Time required. The cost of implementing this methodology is not as high as one might assume at first glance. Since it has been integrated into the process, the impacts to the development team are minimized. Listed below are the various time and resource requirements.

- Kickoff sessions—A kickoff session normally lasts approximately one to one and a half hours. Generally this is enough time to review all materials and discuss quality and team goals.
- Entry of data during rework—The additional overhead of having programmers enter error data into a system and do preliminary causal analysis depends on the number of errors to be processed. However, this overhead, when spread across an entire team, does not create a problem. Generally, having the data on the system reduces the control work required by a team leader.
- Causal analysis meetings—Generally a causal analysis meeting will last about one and a half hours. The time, of course, is dependent on the number of problems to be evaluated. However, if a large number of errors need to be analyzed, more than one session should be planned, since one and a half to two hours is about the maximum amount of time that people can concentrate on errors. This session is creative and therefore should be limited in time.
- Action Team meetings—Since the Action Team meeting is really just a status, prioritization, and

Figure 3 Addition of causal analysis

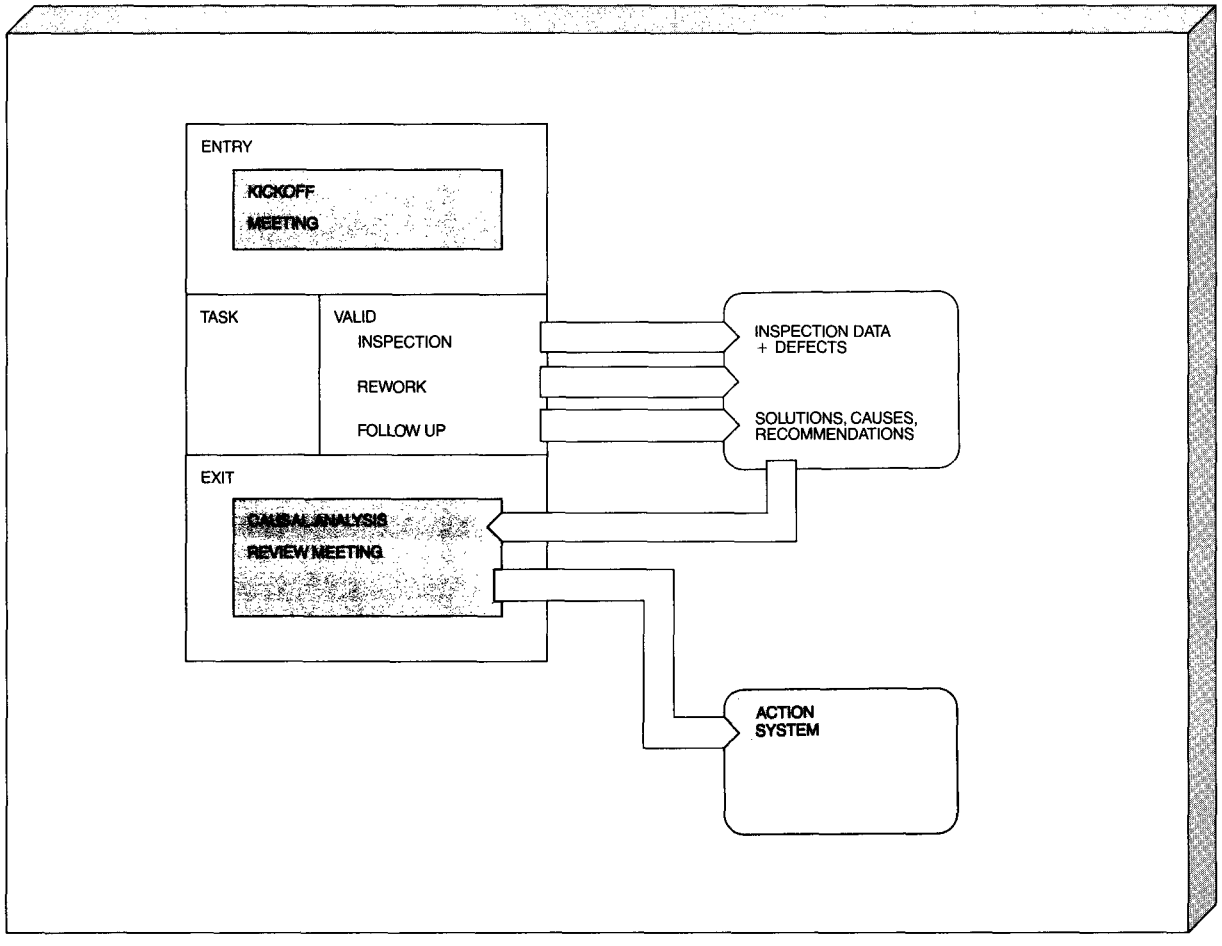
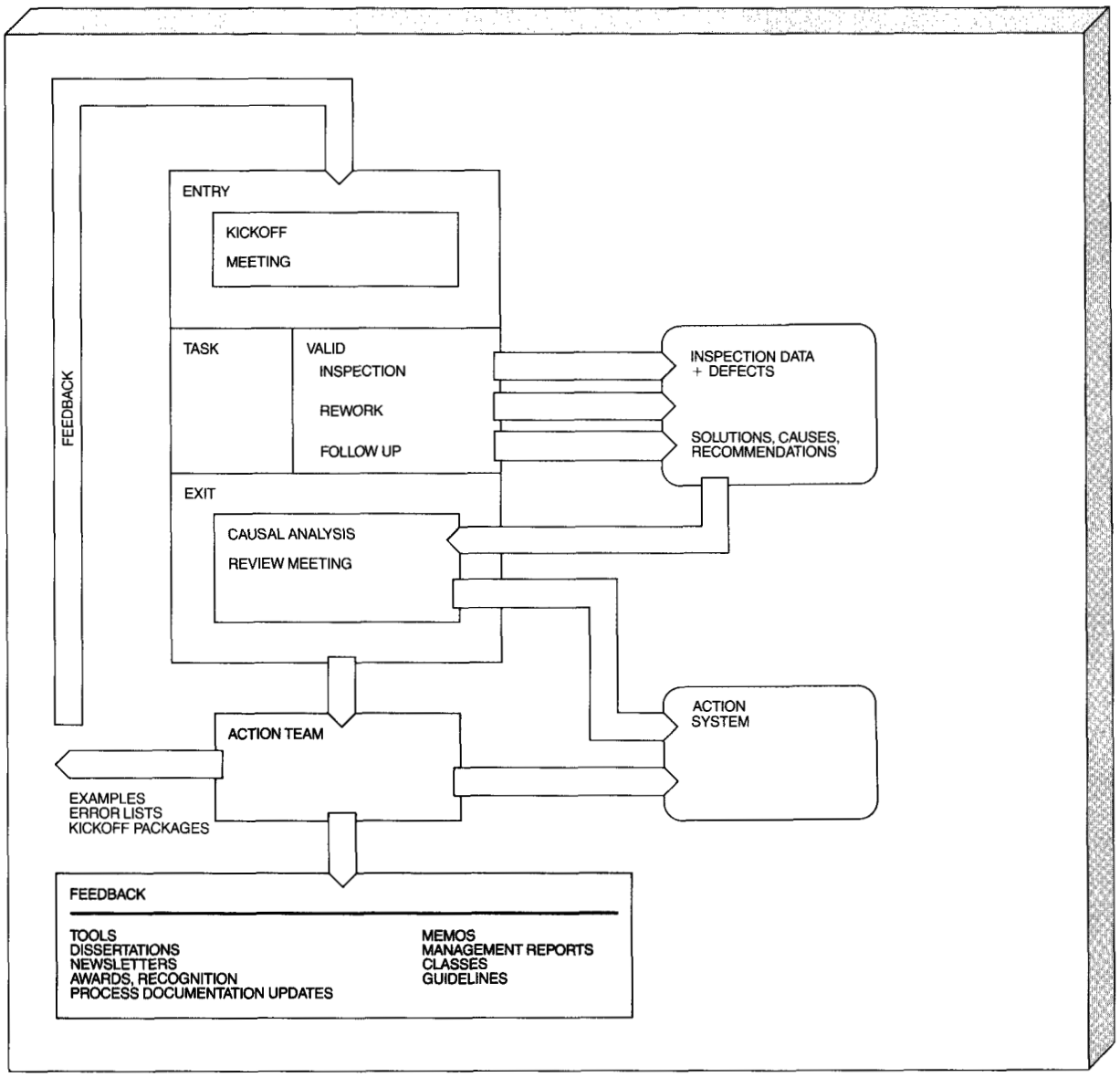


Figure 4 Addition of the Action Team and feedback



assignment meeting, it generally goes fairly fast. One-half to one hour is usually the maximum, again depending on the volume of work to be reviewed.

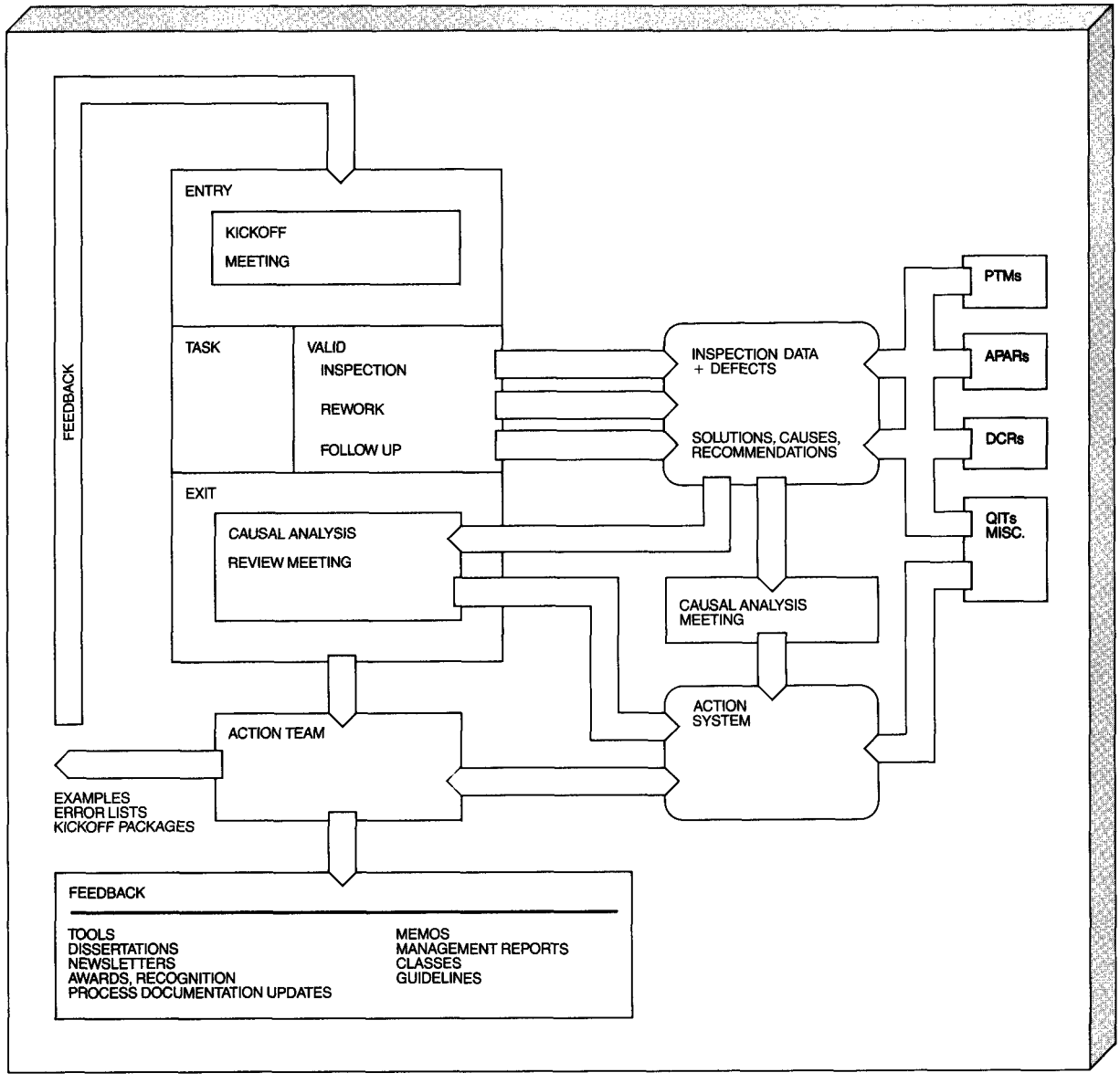
Staffing required

- Action Team—The Action Team will vary in size based on the number of actions being generated by the organization. For a small group, the team may be one part-time person plus management involvement. For a development organization of 100 to 150 people, the team may be as large as four or five full-time members with a dedicated manager.

Additional requirements

- A data base and tool are needed for tracking both defects and actions.

Figure 5 Addition of noninspection data



- Training is required for the Action Team and causal analysis teams.
- Process documentation should be provided for people entering the organization and for reference.

Offsetting benefits

The cost of each error is high, as has been demonstrated many times. The cost of field problems alone can be estimated in the tens of thousands of dollars. Internal test problems are very expensive if one considers the cost of screening the problem, debugging it, developing and applying the fix, and verifying its correctness. This normally costs a minimum of nine hours for the most simple problem when all factors are considered.

Obviously, at these prices, the prevention of errors will pay for the additional cost of implementing this methodology in a short time.

Samples of defects and actions

Samples of actual defects and actions are shown here to illustrate the types of errors and subsequent actions which can occur.

Error: WXTRN was coded when an EXTRN was needed.
Categ.: Education—base code/other.
Cause: EXTRN statement not understood.
Action: Create an entry for the project notebook on EXTRN statements.
Action: Add this item to the common errors list.

Error: Several program error conditions were overlooked.
Categ.: Oversight.
Cause: Last-minute additions caused multiple errors.
Action: Add item to common error list to warn people that late changes are more prone to error.

Error: Program abnormally ended.
Categ.: Transcription.
Cause: Wrong register label typed in. (Assembler program.)
Action: Write a tool to trace registers during unit test.

Error: Program functions were coded in incorrect order.
Categ.: Communications.
Cause: Communications failed between high-level and low-level designers.
Action: Change the process to include regularly scheduled communication sessions to be attended by the designers.

Error: Function would not work as designed in the low-level stage.
Categ.: Education—new function.
Cause: Low-level designers were not aware of a high-level design implication. (The low-level designers and high-level designers are in separate groups.)
Action: Institute education sessions involving both designer groups.
Action: Make sure that low-level designers are assigned in time to attend all education sessions.

Conclusions

The major benefit of this program is that higher product quality is achieved through *reduced errors.* Preliminary results of applying defect prevention in two product areas indicate that fewer errors are injected at each development stage, and there is a corresponding reduction in errors during the test stages by as much as 50 percent. Improvements are also noted in the areas of communications, quality awareness, education, and use of methodologies and tools.

By establishing a feedback mechanism, the process can be improved incrementally by taking advantage of the defect information at our disposal. The process can evolve in a concrete way based on actual data rather than purely from perceptions. Quality takes an active role in the process, and fine-tuning can occur on a day-to-day basis with everyone involved.

The defect prevention methodology described in this paper has the following recommendations:

1. The development team must be directly involved in the causal analysis of defects.
2. The causal analysis of all defects must be integrated into the process.
3. The ENTRY substage must be enhanced to accommodate kickoff sessions at which input, process guidelines, and common errors are reviewed and team goals are set.
4. The EXIT substage must be enhanced to accommodate causal analysis sessions consisting of causal analysis of defects and evaluation of team results.
5. Defect and action data must be collected and managed. A tool and a data base are required.
6. An Action Team should be established to manage the defect prevention data, implement actions to improve the development process, and provide feedback to the developers. A manager should be responsible for the Action Team, to give the team's work management focus.

Several actions can be taken to prevent defects. No matter what methodology is used, however, there must be an aggressive effort to learn from the available data. Without analyzing statistics, defects, or trends, we fail to utilize the most valuable information we have at our disposal for fine-tuning the development processes.

Acknowledgments

The author wishes to acknowledge Robert G. Mays for his insightful contributions to this paper as well as for his active involvement in the development of

this methodology, Gerald J. Holloway and Charles E. Brabec for their initial work on causal analysis, and John F. Osterhoudt and Judith E. Dearlove for their invaluable editorial work.

Appendix A. Guidelines for conducting a kickoff session

Preparing for the meeting. A team leader, a process leader, an Action Team member, or other trained leader can run the meeting. The leader must plan and prepare for the meeting.

The kickoff sessions utilize a standard kickoff package which consists of an agenda, process stage definition, product guidelines for the stage, examples of expected output, and a common errors list for the stage. The kickoff leader simply prints out the package for everyone and brings the package to the meeting. There is no preparation required of the participants for these meetings, so the package need not be distributed ahead of time.

The input to the stage is distributed to all members of the team in advance. In addition, the entry criteria have been verified earlier. Issues regarding input to the stage are resolved prior to the meeting, but if members wish to discuss special problems with the input to the stage, the kickoff meeting is an excellent place for such topics.

The only other material needed by the leader for the meeting is the release quality plan, which contains the expected error rates for the stage. The team will need to know the release quality values in order to determine what they plan to achieve as their team goal.

Conducting the meeting. The following topics should be addressed in the meeting:

- Input to the stage. A brief review of the input is conducted and the team evaluates whether the input is understood by everyone and whether it is complete.

- Process stage definition. This includes entry and exit criteria as well as the intent of the stage and the tasks to be performed.

- Product-specific guidelines. Each product usually has its own set of internal guidelines for each stage; for example, methodologies to be used, status tracking techniques, specifics about what is required as output, design languages to be used, etc. These guidelines need to be reviewed so that all team members understand exactly what is expected from them as a result of this stage.

- Examples of output. Examples are specifically identified here because they are an excellent way of showing exactly what is meant by a guideline. For example, levels of detail are often interpreted differently by each team member, and examples help to clarify what is expected.

- Common errors list. Each stage should have a list of the most commonly experienced errors. New errors can be added and old errors removed as new data become available. The purpose of reviewing an error list is to heighten the awareness of the team members immediately before they start working on an item. If the leader can elicit discussions while reviewing the error lists, the value of the kickoff increases.

- Team goals. The release quality goal for the stage should be presented by the leader, followed by a discussion by the team of what they believe their team goals should be. Can they do any better than the release goals? What does each person think the goal should be, and what is the plan for achieving it? What can be done to raise the release quality?

  The leader should record the team goal for the exit substage. This goal is not recorded in any data bases or made public to anyone but the team. When the team meets for the exit stage, the actual results will be compared to the team goal, and discussion can ensue about what worked and what did not work in the methodologies used. Often, the comparison of team goals versus actual results can aid in discussions about process improvements which do not show up as a result of the causal analysis of specific errors.

Appendix B. Guidelines for conducting a causal analysis session

Preparing for the meeting. A team leader, process leader, or Action Team member can run the meeting, but he or she should be aware of what is to be expected from the meeting as well as what can be expected during the meeting. For specifics, see the following subsection on conducting the meeting.

A description of all errors being evaluated is made available to each team member. The format will vary

depending on the type of error. Causal analysis forms (see Appendix F) are helpful in collecting the right information if the error report form does not have all appropriate fields on it. The causal analysis form has been designed to provide a convenient way to record the results of the discussions for later input into a data base system.

For a causal analysis exit meeting, the release goal, team goals, and the corresponding actual results are needed.

Conducting the meeting. The purpose of the meeting is to use all available data to recommend actions for improvement.

The session is run in three parts. First, the release and team defect goals are compared to the actual results. This comparison provides feedback to the team members in terms of how well they did compared to their initial estimates. Second, all errors are reviewed and an action list is created. Third, the stage is evaluated for improvements. A sample agenda is given below. The analysis portions are divided because it is important to focus on the preventive suggestions based solely on the defects first. After all errors have been evaluated and actions have been created, the team has a good understanding of what happened in the stage and can then have a meaningful evaluation of that stage.

During the analysis portion of the meeting, it is best to review all errors before trying to create an action list. This approach of reviewing all errors first allows the team to group similar errors together before determining suggested actions. It also creates an environment where more comprehensive suggestions can be discussed. Trying to formulate actions based on single errors becomes meaningless at times; however, when the scope includes all errors in the session, the team will expand its discussions beyond the immediate errors.

Sample agenda

1. Compare release goals, team goals, and actual results.
2. Causal analysis
   a. Evaluate all errors. That is, read through each error, discussing the defect and the cause. Make sure that all team members understand why the error occurred.
   b. Create an action list after all errors have been discussed by quickly reviewing the errors and placing their preventive actions in a list. Several defects may contribute to the same action and several actions may be required for a given defect.
3. Process stage evaluation—When all causal analysis is complete, do the following:
   a. Ask the following questions:
      (1) What was done by this team that worked well and should be recommended or pointed out to the next team going through this stage?
      (2) What could be done to improve the process?
      (3) What tools could help detect the errors while still in this stage?
   b. Create additional action items based on discussions in the process evaluation part of the session.

Clues for conducting the meeting

Be sensitive to defensiveness. When inspections were first instituted, there was a tendency on the part of the authors to be defensive about errors. With causal analysis, this problem once again becomes an issue. The team investigates, in depth, why a team member made a mistake, putting the team member in a vulnerable position. The leader should be sensitive to this issue. There are several techniques that can be used:

- Do not attack. Do not focus on who is at fault or who is to blame.
- Keep the discussion focused on preventing a similar error in the future. Team members must feel free to discuss their own mistakes. Members should be reminded that the error that they made has been made in the past and may well be common. What can be learned from the mistake?
- If necessary, assure the team that only the name of the team leader will be entered into the data base so that errors are not traceable to individuals.
- Keep the meeting informal. Humor frequently helps.

Allow errors to trigger general discussions. Each error can be used as a springboard to more general solutions. The specific error will usually remind the team of a series of errors that are similar. The *type* of error then becomes the topic of discussion.

Consider error categories. Generally, communications- and education-type errors are easier to resolve. They tend to result in either process or education actions.

Preventive techniques for oversight- or transcription-type errors are more difficult to determine and require more creativity. Techniques that improve the working environment, methodologies for coverage completion, or tools are often good ideas for these errors.

Do not stop after one action. Just because the team has identified an action for a defect does not mean that it should stop the discussion and move on to the next defect. Often, several other suggestions will surface if given a chance.

Consider all errors. Make sure that all errors are considered when making the action list.

Get real actions—not suggestions. Remember that someone on the Action Team must actually do a specific task to implement the suggestion. When defining the action, the team should be very precise about what must be done in the way of actions or tasks. The determination of the actions should not be left to the Action Team. Be precise. Define not only *what* is to be done but also *how* to do it.

Following the meeting. After the meeting is over, the leader enters the actions into a data base. The actions are linked to the defects at this time. Action suggestions formulated during the process evaluation part of the session are also entered into the system but without linked defects.

The leader generates a report of the actions and their associated defects for distribution to all team members. This report summarizes the causal analysis session and identifies the actions that have been turned over to the Action Team. The action system should be available for anyone to browse so that interested team members can determine the status of an action at any time.

Appendix C. Action meeting

The Action Team meets regularly, every week or every two weeks. The Action Team meeting is short and emphasizes status and actions that have not been assigned. A member of the Action Team is responsible for bringing the appropriate reports to the meeting. This member also enters the updates to the data base reflecting new assignments. Each individual Action Team member can update action entries when his or her status changes.

Periodically, the Action Team performs generic analyses of errors and evaluation of feedback techniques. The team makes recommendations to management for recognition of development teams that have made effective suggestions.

Appendix D. Defect records

Automated support of a defect prevention methodology requires keeping data on each error reported and each suggested action. This information is stored in defect and action records. Many defects may correspond to a single action, and one defect may relate to many actions. Therefore, it is recommended that there be some ability to cross-reference defects to actions and vice versa.

A defect record must track certain metrics. Those listed below are the fields which, at a minimum, should be considered.

DEFECT ENTRY #	A unique number to identify the entry
PRODUCT	Name of the product
RELEASE	Release identifier
DRIVER	Identifier of driver (a small subset of the release)
LINEITEM	Functional item being developed
STAGE DETECTED IN	Process stage where error was detected
STAGE CREATED IN	Process stage where error was created
QIT DEPARTMENT	For QIT suggestions, the department of the QIT
PROBLEM REPORT #	For miscellaneous problem suggestions, the number of the problem (example, test problem number)
CREATE DATE	Date the defect was entered into the system
ANALYST	Programmer who should be contacted for questions
TYPE OF ANALYSIS	Inspection, test problem, field problem, QIT, miscellaneous
TYPE OF INSPECTION	Group, peer, team leader, other
CHECKPOINT DATA	Current status of the entry—attentioned, screened, being investigated, closed

CLOSE DATA	Closing/reason codes, answer text, programmer identification, and date
CATEGORY OF CAUSE	Communications breakdown Education—New function Education—Base function Education—Other Oversight—Did not consider all cases Transcription error—Mistake
ABSTRACT OF DEFECT/ PROBLEM	A short description of the defect
ABSTRACT OF CAUSE OF DEFECT	A short description of the defect cause
ASSIGNED ACTIONS (a list of associated actions)	Action numbers assigned to prevent this defect
PROBLEM DESCRIPTION and SUGGESTED ACTIONS	A full description of the problem and actions
LOG OF ACTIVITIES AGAINST THE DATA BASE	A track record of all activities checkpointed against this defect

Appendix E. Action records

The purpose of the action record is to track each action that is recommended. Each record should contain at least the following:

ACTION #	A unique number to identify the action
PRODUCT	The product name or identifier
PROGRAMMER	Programmer submitting the action
CREATE DATE	Date action item was entered into the system
PRIORITY	1–4
AREA CODE	Where the implementation occurred (process, tools, etc.)
LINEITEM	Specific item within the area
CHECKPOINT INFO	Current status of the entry—attentioned, screened, being investigated, closed
COST ESTIMATE	Number of days expected for implementation
TARGET DATE	Date of expected completion
CLOSE DATA	Closing/reason codes, programmer identification, date
FINAL COST	Number of days implementation actually took
ABSTRACT OF ACTION	A short description of the action
ACTION DESCRIPTION	A full description of the action
ASSOCIATED DEFECTS	List of all defects linked to this action
ANSWER TEXT	A full description of the action that took place
LOG OF ACTIVITIES AGAINST THE DATA BASE	A track record of all activities checkpointed against the action

Appendix F. Causal analysis forms

Three forms are shown here:

- *Problem Report Form:* This form is used during the inspection to collect data on the problem and during rework to record resolutions and causal analysis data.
- *Defect Analysis Report:* This is a general-purpose form used in any causal analysis meeting for recording analyses of defects.
- *Action Plan Analysis:* This form, also used in the causal analysis meeting, is used to determine recommended actions.

Cited references

1. M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal* **15**, No. 3, 182–211 (1976).
2. R. A. Radice, N. K. Roth, A. C. O'Hara, Jr., and W. A. Ciarfella, "A programming process architecture," *IBM Systems Journal* **24**, No. 2, 79–90 (1985, this issue).
3. A. Endres, "An analysis of errors and their causes in system programs," *IEEE Transactions on Software Engineering* **SE-1**, 140–149 (1975).
4. P. B. Crosby, *Quality Is Free*, McGraw-Hill Book Co., Inc., New York (1979).

General references

J. D. Aron, "The programming development process: Part II: The programming team," *The System Development Life Cycle*, Addison-Wesley Publishing Co., Reading, MA (1983), pp. 163–195.

P. W. Metzger, *Managing a Programming Project*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973).

PROBLEM REPORT FORM

Page ___ of ___

circle one: PRODUCT COMPONENT MODULE CODE OTHER:_____

circle one: INSPECTION PEER REVIEW TEAM LEADER REVIEW

Product	Release	Driver	Lineitem
Component	Module/Macro	Developer	Reviewer
Date Submitted for Review	Date Reviewed	Hours	Number of Major Errors

Person raising issue:	Problem Description:

Type: 1 2 circle one:	Answer Date:	Answered by:	Ans. Review Date:	Reviewed by:
Solution:				

Abstract of defect (50 char max):

PHASE where error was CREATED (check one):

__ PRODUCT
__ COMPONENT
__ MODULE
__ CODE
__ UNIT TEST
__ FUNCTIONAL TEST
__ PRODUCT TEST
__ SYSTEM TEST
__ PACKAGING TEST
__ POST-SHIP

Cause Category (check one):

___ Education: Didn't understand new
___ Education: Didn't understand base
___ Education: Other
___ Communications.
___ Oversight. Didn't consider all cases
___ Transcription. Simply made a mistake

Abstract of cause for error (50 char max):

How can error be avoided the next time?

What action is needed?

DEFECT ANALYSIS REPORT:

PRODUCT: _____

ANALYST NAME: _____

TYPE OF ANALYSIS: INSPECTION TEST-PROBLEM FIELD-PROBLEM DESIGN-CHANGE QIT MISC POST-MORTEM

If an Inspection:	If a Test Problem:	If a Field-Problem / Design Change:
Release:_____ (XX.X) Driver: _____ (XX) Lineitem: _____ (XXX)	Release: _____ (XX.X) Lineitem:_____ (XXX) Prob. #: _____ (7-chars)	Number:_____ (7-chars)
Stage where error was detected:	Stage where error was detected:	If from a QIT:
___ PRODUCT ___ FUNCTIONAL TEST ___ COMPONENT ___ PRODUCT TEST ___ MODULE ___ SYSTEM TEST ___ CODE ___ FINAL PACKAGE TEST ___ UNIT TEST ___ AFTER SHIPMENT ___ OTHER	___ UNIT TEST ___ PACKAGE TEST ___ FUNC. TEST ___ AFTER SHIP ___ PROD. TEST ___ OTHER ___ SYSTEM TEST	Dept:_____ (3-chars)
		If a Post-Mortem:
		Release:_____ XX.X

For Inspection,Test-Prob.,Field-Prob.,Design-Change:	For QIT, Post-Mortem, or Misc. Entry:
Abstract of Defect (50 char max):	Abstract of Problem (50 char max):
Process Stage where created: (check one) ___ PRODUCT ___ UNIT TEST ___ PACKAGE TEST ___ COMPONENT ___ FUNC. TEST ___ AFTER SHIPMENT ___ MODULE ___ PROD. TEST ___ OTHER ___ CODE ___ SYSTEM TEST	Abstract of Cause of Problem (50 char max):
Category of Cause (check one): __ Education: Didn't understand new __ Education: Didn't understand base __ Education: Other __ Communications Breakdown __ Oversight. Didn't consider all cases. __ Transcription error. Simply made a mistake.	
Abstract of Cause of Error (50 char max):	

How can error be avoided next time:

ACTION PLAN ANALYSIS: Page ___ of ___

PRODUCT: _____

PROGRAMMER NAME: _____

PRIORITY (circle one): 1 2 3 4

COMMITTED TARGET DATE: (YYMMDD)

PROJECTED COST (in person-days): (XXXX.X)

ABSTRACT OF ACTION ITEM (50 char max):

DETAILED DESCRIPTION OF ACTION ITEM:

LIST OF DEFECT ANALYSIS ENTRY IDS (7 chars each):

Carole L. Jones *IBM Communication Products Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Ms. Jones received her B.S. in mathematics from Youngstown State University in 1966. She joined IBM in DOS development at Endicott, New York, working in both the development and advanced testing areas. She transferred to Research Triangle Park (Raleigh) in 1970 and began work in Engineering and Administrative Software Support for the Raleigh site. She then transferred to the TPNS project, with emphasis in both development and test. Ms. Jones joined the Network Control Program (NCP) development area in 1975 and worked as a development leader on several releases until 1980, when she assumed the responsibility of process coordinator for the NCP products. In that capacity, she focused on process definitions, methodologies, and all aspects of quality and productivity as they apply to process management. For the past eighteen months, she has been actively developing and applying the concepts of defect prevention to the NCP products.

Reprint Order No. G321-5245.