

Software Failure Analysis for High-Return Process Improvement Decisions

Software failure analysis and root-cause analysis have become valuable tools in enabling organizations to determine the weaknesses in their development processes and decide what changes they need to make and where.

by Robert B. Grady

When I was growing up, my father was fond of using sayings to encourage me to remember important lessons. One of his favorites was “Do you know the difference between a wise man and a fool?” He would then go on to say that a wise man makes a mistake only once. A fool makes the same mistake over and over again.

Applying my father’s saying to software defects, it sometimes seems as if there are many “fools” among software developers. However, there aren’t. Individually, they learn from their mistakes. What’s missing is organizational learning about our software mistakes. I guess that many organizations have earned my dad’s “fool” label.

One useful way to evaluate software defects is to transfer process learning from individuals to organizations. It includes not only analyzing software defects but also brainstorming the root causes of those defects and incorporating what we learn into training and process changes so that the defects won’t occur again. There are five steps:

1. Extend defect data collection to include root-cause information. Start shifting from reactive responses to defects toward proactive responses.
2. Do failure analysis on representative organization-wide defect data. *Failure analysis is the evaluation of defect patterns to learn process or product weaknesses.*
3. Do root-cause analysis to help decide what changes must be made. *Root-cause analysis is a group reasoning process applied to defect information to develop organizational understanding of the causes of a particular class of defects.*
4. Apply what is learned to train people and to change development and maintenance processes.
5. Evolve failure analysis and root-cause analysis to an effective continuous process improvement process.

How do these steps differ from other popular methods for analyzing processes? One popular method is process assessments, for example, SEI (Software Engineering Institute) process assessments.¹ Most assessments document peoples’ answers to subjective questions that are designed around somebody’s model of ideal software development practices. If such models are accurate and if peoples’ answers reflect reality, the models provide a good picture of an organization’s status. Thus, the results may or may not be timely, representative, or motivational.

The combination of failure analysis and root-cause analysis is potentially more valuable than subjective assessments, because it quantifies defect costs for a specific organization. The key point to remember is that software defect data is your most important available management information source for software process improvement decisions. Furthermore, subsequent data will provide a measurable way of seeing results and evaluating how methods can be further adapted when a specific set of changes is done.

Reactive Use of Defect Data (A Common Starting Point)

After initial analysis, everyone reacts to defects either by fixing them or by ignoring them. Customer dissatisfaction is minimized when we react quickly to fix problems that affect a customer’s business. This is often done with fast response to issues and by following up with patches or workarounds, when appropriate. Some Hewlett-Packard divisions track the resolution of “hot sites.” Fig. 1 shows an example.² Such a chart is a valuable way to track responsiveness, but it does little to prevent future defects. Furthermore, hot sites and patch management are very expensive.

Cumulative defects for long-lived software products are also tracked. For example, Fig. 2 shows the incoming service requests or discrepancy reports, the closed service requests or discrepancy reports, and the net progress for one NASA software project.³ Some HP divisions also track progress like this,² although HP’s progress measure subtracts incoming defects from closed defects so that positive progress represents a net reduction in defects. NASA appears to do the reverse.

Fig. 1. Tracking the number of hot sites during any particular week. For example, for the week indicated there were $M - N$ hot sites that had been hot for a long time and N hot sites that had been hot for a short time. © 1992 Prentice-Hall used with permission.

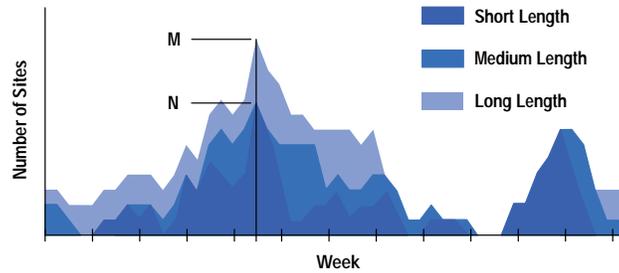
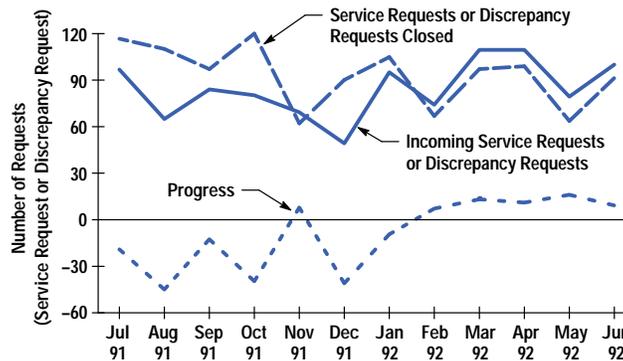


Fig. 2. Incoming maintenance requests, closed maintenance requests, and net progress for one NASA project. This figure is reprinted by permission of the publisher from "A Software Metric Set for Program Maintenance Management," by G. Stark, G.L. Kern, and C. Vowell, *Journal of Systems and Software*, Vol 24, p. 243. © 1994 by Elsevier Science Inc.



Both the hot site graph and the defect closure progress graph show reactive uses of defect data. In the examples, the respective organizations were using the data to try to improve their immediate customer situations. The alternative is to ignore the data or to react much more slowly.

Ignoring defect data can lead to serious consequences for an organization's business. For example, the division producing one HP software system decided to release its product despite a continuing incoming defect trend during system test. The result was a very costly update shortly after release, a continued steady need for defect repairs, and a product with a bad quality reputation. This is the kind of mistake that can cause an entire product line's downfall. A recent article described how one company learned this lesson the hard way.⁴

Responses should not be limited only to reaction. Besides endangering customer satisfaction and increasing costs, here are some other dangers that could occur if reactive processes aren't complemented with proactive steps to eliminate defect sources:

1. People can get in the habit of emphasizing reactive thinking. This, in turn, suggests that management finds shipping defective products acceptable.
2. Managers get in the habit of primarily rewarding reactive behavior. This further reinforces fixing defects late in development or after release. Late fixes are both costly and disruptive.
3. People place blame too easily in highly reactive environments because of accompanying pressure or stress. This is demoralizing, since the root causes of most defects are poor training, documentation, or processes, not individual incompetence.

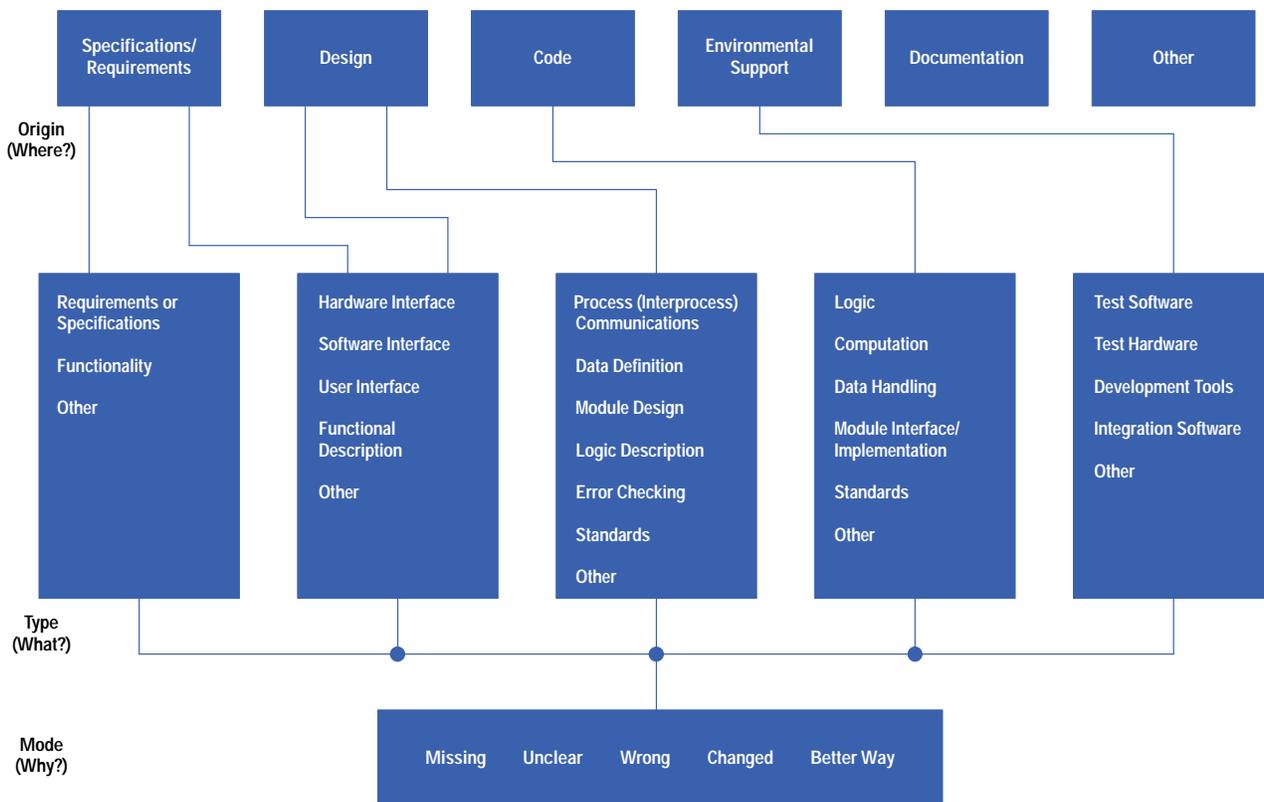
Remember that effectively reacting to defects is an important part of successfully producing software products. However, because business conditions change rapidly, many organizations can't seem to find the time to break old habits of using defect data reactively without considering ways of eliminating similar future problems. The elimination of the causes of potential future defects must be included in any successful long-term business strategy.

Failure Analysis (Changing Your Mental Frame of Reference)

The proactive use of defect data to eliminate the root causes of software defects starts with a change in mental frame of reference. The reactive frame generally focuses on single defects and asks "How much do they hurt?" It also considers how important it is to fix particular defects compared with others and asks "When must they be fixed?" The proactive frame asks, "What caused those defects in the first place? Which ones cause the greatest resource drain? How can we avoid them next time?"

Various reports have described successful efforts to analyze defects, their causes, and proposed solutions. But the terminology among them has differed, and the definitions could mean different things to different people. In the fall of 1986, the HP Software Metrics Council addressed the definition of standard categories of defect causes. Our goal was to provide standard defect terminology that different HP projects and labs could use to report, analyze, and focus efforts to eliminate defects and their root causes. Fig. 3 is the model that has evolved from our original definitions.²

Fig. 3. Categorization of software defects. © 1992 Prentice-Hall used with permission.



The model is used by selecting one descriptor each from origin, type, and mode for each defect report as it is resolved. For example, a defect might be a design defect in which part of the user interface described in the internal specification was missing. Another defect might be a coding defect in which some logic was wrong.

Fig. 4 gives some idea of how defects vary from one entity to another.⁵ The different shadings reflect the origin part of Fig. 3. The pie wedges come from the middle layer of Fig. 3, the defect types. The eight largest sources of defects for different HP divisions are shown in each pie. All four results profile defects found only during system and integration tests.

We can immediately see from Fig. 4 that the sources of defects vary greatly across the organizations. No two pie charts are alike. These differences are not surprising. If everyone developed the same way and experienced the same problems, then we would have fixed those problems by now. Instead, there are many different environments. While many proposed solutions to our problems apply to different situations, they don't necessarily apply equally well to all problems or all environments.

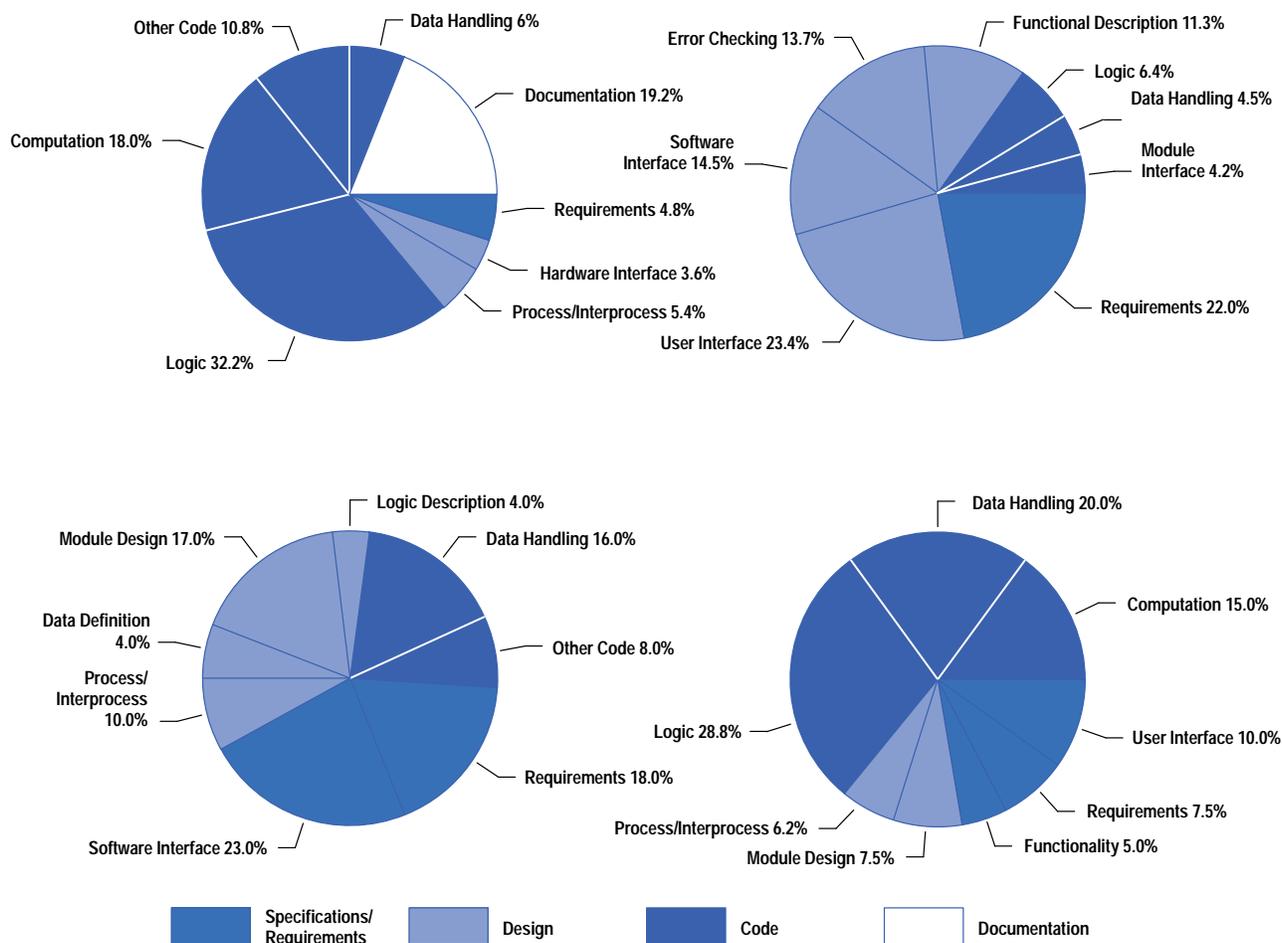
Some of the differences are because of inconsistencies in peoples' use of the origin and type definitions. Because the definitions are just a means to focus process improvement efforts on the costliest rework areas, groups resolve inconsistencies when they define root causes to problems and brainstorm potential fixes. It is the triggering of these discussions that makes the data in Fig. 4 so important. Discussing root causes is a way to instill a process improvement attitude in an organization. Defect data will provide a measurable basis for decisions that must be made. By continuing to track defect data, an organization can also measure how successful its solutions are.

Acting on Causal Data

Collecting defect source data is only the first step. Persuasive as the data might be, improvements won't happen automatically. Both managers and engineers must agree on what the data means and the importance of acting on it. One of the best ways to help ensure that this happens is to tie proposed improvements to stated business goals. This also keeps improvement priorities high enough to help ensure sustained management support.

Besides management support, some first-line managers and engineers affected by a proposed change must be motivated to do something and be assigned responsibility to plan and do the necessary changes. Finally, as for any effective project, there must be a way of monitoring progress and gauging success.

Fig. 4. Sources of defects found during testing in four HP divisions.



As a group, software developers now have several decades of software development experience. It is time to break out of our pressure-driven reactive habits and use our accumulated knowledge to drive lasting improvements. Failure analysis changes the way managers and developers look at software defects. This finally opens the way to a proactive frame of reference.

Root-Cause Analysis Processes

There are many possible ways to analyze root-cause data. Any successful way must be sensitive to project pressures and personnel motivation. HP has used several approaches in different organizations. For this discussion, I will label three that seem to evolve naturally from each other as *one-shot root-cause analysis*, *post-project root-cause analysis*, and *continuous process improvement cycle*. These three approaches include many common steps. Since the first is an introductory process, the most detailed explanation is saved for the post-project root-cause analysis.

One-Shot Root-Cause Analysis

A good starting approach for organizations that have not previously categorized their defect data by root causes is a one-shot root-cause analysis. This approach minimizes the amount of organizational effort invested by using someone from outside the organization to facilitate the process. At HP most divisions have defect tracking systems with complete enough information to extract such data.

The one-shot process has six steps.

1. Introduce a group of engineers and managers to the failure-analysis model (Fig. 3) and the root-cause analysis process. (About one hour.) Make it clear that the goals of the one-shot process are to:
 - Create a rough picture of divisional defect patterns.
 - Identify some potential improvement opportunities.
2. Select 50 to 75 defects from the defect tracking system using a random process. Make sure that the team thinks the defects have enough information to enable them to extract the necessary causal information. (About two hours sometime before the meeting.)

3. Have the people in the group classify one defect per person and discuss the findings as a group. Then have them classify enough defects so that you have about 50 total. Draw a pie chart of the top eight defect types. (About two hours.)
4. Pick two defect types to focus on. Create fishbone diagrams from the combined root causes and additional comments. A fishbone diagram is a brainstorming tool used to combine and organize group thoughts.^{2,6} (About half an hour.)
5. Develop some recommendations for improvements. (About half an hour)
6. Present the results and recommendations to management. Make assignments to do initial changes. (About one hour)

Participants in this process have been generally surprised and excited that they could learn so much in a very short time. They have also been uniformly interested in adopting the analysis process permanently. How quickly they have followed through has varied, depending on many business variables such as immediate product commitments, other in-progress changes, or a tight economic climate.

Post-Project Root-Cause Analysis

The major difference between this process and the one-shot process is that organizations that start with the one-shot process have not previously collected causal data. Organizations that already collect failure-analysis data and have an understanding of their past defect patterns analyze their data and act on their results more efficiently. The steps in this approach follow the meeting outline shown in Fig. 5. Note that the times shown in Fig. 5 are intended to force the meeting to keep moving. It is best to schedule a full two hours, since all that time will be needed. The example used here to illustrate this process came from a root-cause analysis meeting done at an HP division shortly after a team at that division released a new product.

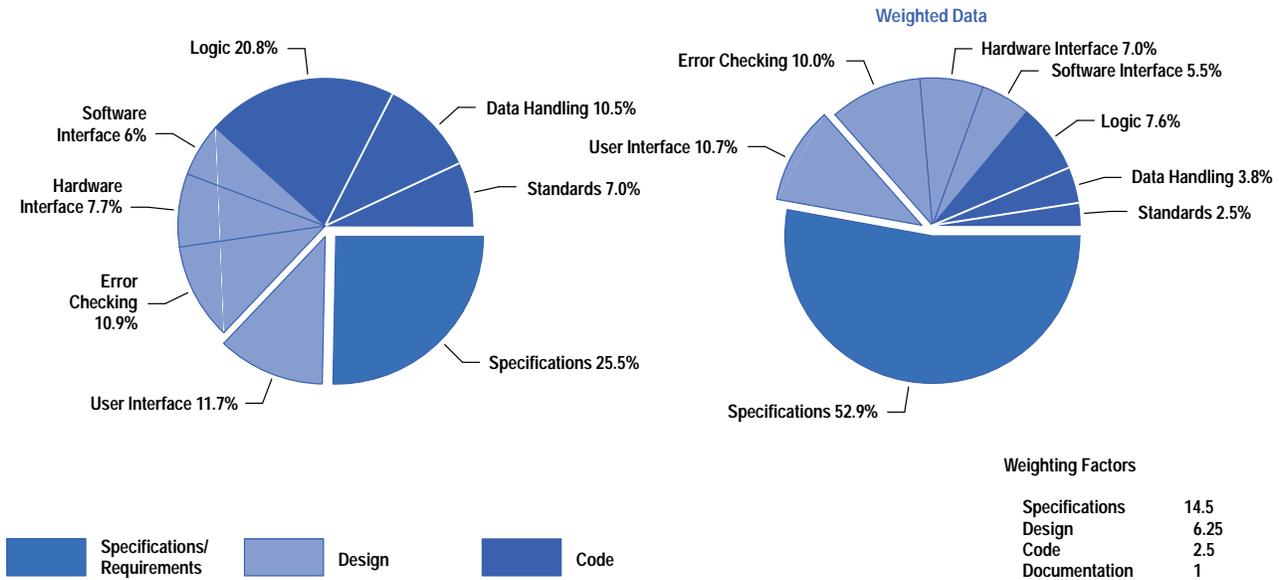
Fig. 5. Root-cause analysis meeting outline.

- Premeeting**
- Identify the division's primary business goal.
 - Have the division champion and root-cause facilitator analyze data.
 - Have the champion send out the meeting announcement and instructions to engineers.
 - Pick two defects from their code that have been chosen from the defect categories.
 - Think of ways to prevent or find defects sooner.
- Meeting**
- State the meeting's goal (use insights gained from failure analysis data to improve development and support practices).
 - Perform issues selection (10 minutes).
 - Review the defects brought to the meeting (15 minutes).
 - Perform analysis (15 minutes).
 - Take a break (10 minutes).
 - Brainstorm solutions (10 minutes).
 - Test for commitment (10 minutes).
 - Plan for change (10 minutes).
- Postmeeting**
- Have the division champion and root-cause facilitator review meeting process.
 - Have the division champion capture software development process baseline data.

Premeeting:

- Identify the organization's primary business goal. This goal is an important input when prioritizing which high-level defect causes should be addressed first. It also helps to frame management presentations to ensure sustained management support. Typical business goals might be framed around maximizing a particular customer group's satisfaction, evolving a product line to some future state, or controlling costs or schedule to get new customers.
- The division champion and root-cause facilitator analyze the data. The champion is a person who promotes a process or improvement activity, removes obstacles, enthusiastically supports implementers and users, and leads through active involvement. The root-cause facilitator is a person who runs the root-cause analysis meeting. The champion and the facilitator need to be skilled at meeting processes and dynamics and be familiar with software development and common defect types. One simple data-analysis approach is to enter the data into an electronic spreadsheet. Draw pie charts of the top eight defect types by quantity and by find and fix effort (either actual or estimated). Fig. 6 shows the system-test data for four projects at one HP division. The shading represents defect origin information, and the pie wedges are defect types. The left pie chart shows the eight most frequently recorded causes of defects. The right pie chart shows the data adjusted to reflect that design and specification defects found during system test cost much more to fix than coding defects do. Since the HP division that provided this data did not collect their defect-fix times, the weighting factors are based on six industry studies summarized in reference 2. The right pie chart in Fig. 6 was prepared by multiplying the left pie chart wedge percentages (or counts) by the appropriate weighting factor and then converting back to 100%.

Fig. 6. Top eight causes of defects for one division.



- Select two defect types to brainstorm based on the best estimate of the organization's concern or readiness to implement solutions. The two defect types selected for this meeting were user-interface defects and specifications defects. The specifications defect type was picked because it was the largest division category (64 out of 476 defects were classified as specifications defect types for this project team). User-interface defects were picked because they were the largest category (110 defects) that the particular brainstorming team had experienced. Both categories represented significant divisional improvement opportunities.
- Send out instructions to engineers. The organization champion should have each engineer bring hard-copy information on two defects from their code, based on the chosen types. Tell invitees to think back to the most likely root cause for each defect and to propose at least one way to prevent or find each defect sooner.

Meeting:

- State the meeting's goal (use insights gained from failure-analysis data to improve the organization's development and maintenance practices). Present the defect categorization model, show typical patterns for other organizations, and show your organization's pattern. Set a positive tone for the meeting. Remind participants that they will be looking at process flaws, and that they must avoid even joking comments that might belittle the data or solutions discussed.
- Issues selection. Reiterate the reasons for selecting this meeting's particular defect types. Let people make initial comments. Address concerns about potential data inaccuracies (if they come up at this point) by emphasizing the solution-oriented nature of the brainstorming process. Suggest that inaccuracies matter less when combining pie wedges to consider solutions. For example, for the sample division meeting, some engineers had a hard time calling some defects "user interface" as opposed to "specifications." We simply used both labels for such defects during the meeting instead of getting sidetracked on resolving the differences. You want to get people ready to share their defects by discussing a future time (like their next major product release) when they will have done something in their process to eliminate the reasons for the defects.
- Review the defects brought to the meeting. Have engineers read their own defects, root causes, and solutions. The major reason to do this is to get attendees involved in the meeting in a nonthreatening way. Thus, don't criticize those who did not prepare, rather encourage them to contribute in real time. Unlike inspections, root-cause analysis meetings require very little preparation time for attendees. After their first meeting, attendees will realize this, and it will be easier to get them to review their defects before the next meeting.

Get in a creative, brainstorming mood by showing the engineers that all their inputs are right, and begin to form a shared understanding of terminology and definitions, and an acceptable level of ambiguity. This section also gives you some idea whether there is some enthusiasm for any particular defect types. You can use such energy later to motivate action.

The following two examples are from the root-cause meeting held by the example HP division. There were 12 engineers and managers at this meeting.

1. User-interface defect: There was a way to select (data) peaks by hand for another part of the product, but not for the part being analyzed.

Cause: Features added late; unanticipated use.

Proposed way to avoid or detect sooner: Walkthrough or review by people other than the local design team.

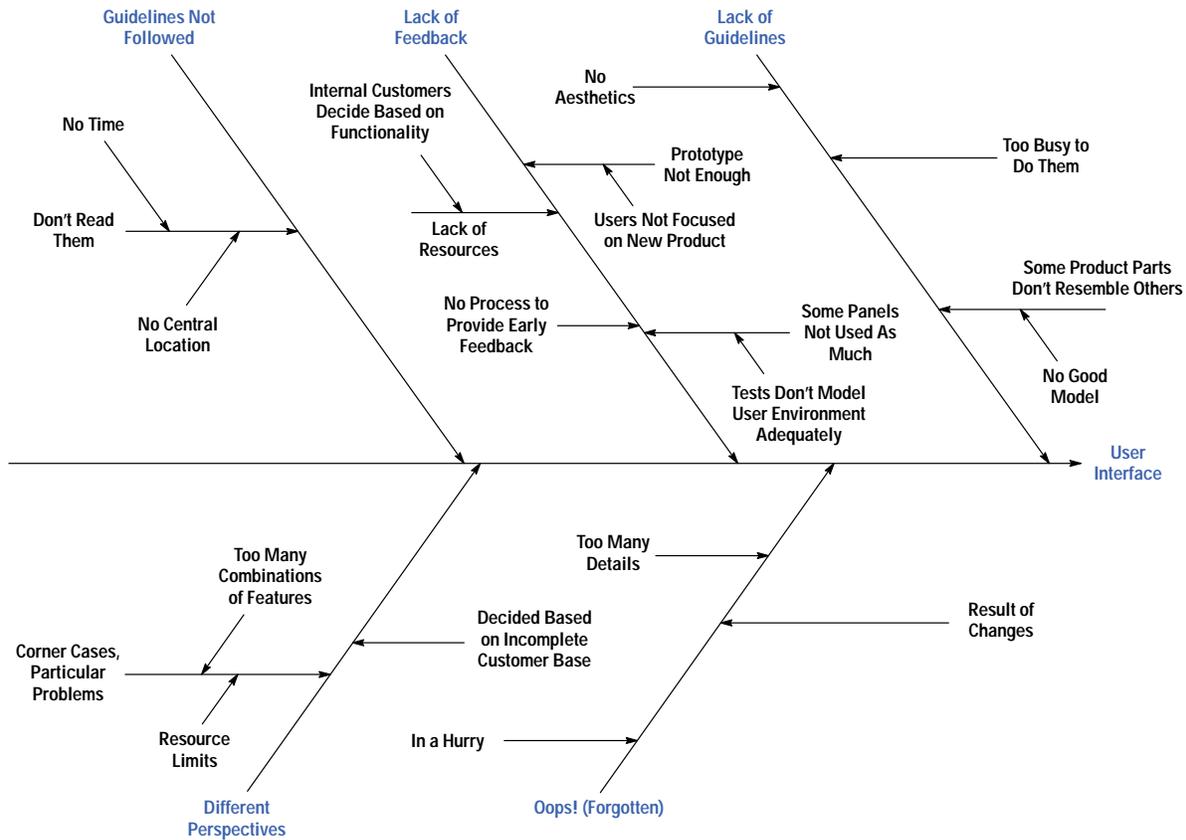
2. Specifications defect: Clip function doesn't copy sets of objects.

Cause: Inherited code, neither code nor error message existed. Highly useful feature, added, liked, but never found its way back into specifications or designs.

Proposal to avoid or detect sooner: Do written specifications and control creeping features.

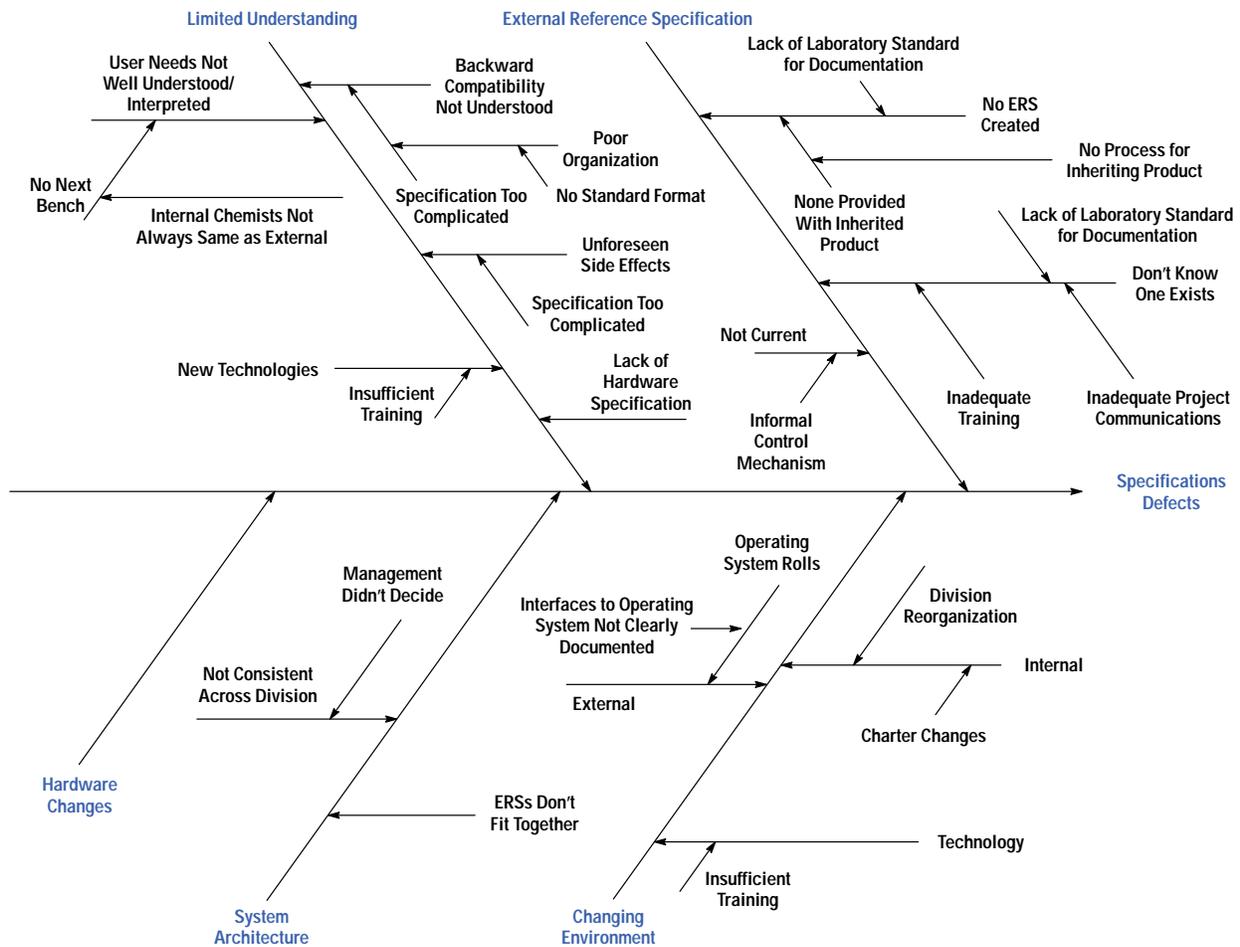
- Perform analysis. Create fishbone diagrams^{2,6} from combined root causes and additional comments. Use this discussion to bring the group from their individual premeeting biases regarding defects to a group consensus state. A useful technique for grouping the defects is to write the suggested causes on movable pieces of paper. Then have the group silently move the papers into groupings of related areas. If some of the papers move back and forth between two groups, duplicate them. The resulting groupings are called an *affinity diagram*.⁷ These are major bones of the fishbone that the group must name. Don't expect the fishbone to be perfect here or even complete. The next session will potentially contribute more. Also, don't get concerned about form. Let the group know that a fishbone is just a means to an end, that it will be cleaned up after the meeting, and that it is likely to change even after that point. The fishbone diagrams in Figs. 7 and 8 are from analyzing the two defect types mentioned above.
- Take a break. This type of meeting takes a lot of energy and focus. It's hard to sustain that for two full hours.
- Brainstorm solutions. Use this time as an orthogonal approach to analyzing the issues at hand. This is also the transition from analysis to action planning for change. Think about what group energy can be turned into solution planning.

Fig. 7. Fishbone diagram for the causes of user-interface defects.



For our sample team, there was a lot of group interest in both defect types. Because a task force already was working on specification defects as a result of the previous root-cause analysis meetings, planning focused on user-interface defects. In the solution list they created, some of the solutions may seem vague. Remember that the brainstorm list is only an intermediate step toward defining action steps. Just be sure that the group understands what it means by the solutions. If members seem to understand the solutions, there is no need to slow down the brainstorming process for more precise definitions. These can be added later.

Fig. 8. Fishbone diagram for the causes of specifications defects.



The solution list they created is as follows:

1. Learn from past experience—track user interfaces, particularly when changes occur.
2. When new functionality is thought of or added, always design and specify user-interface implications.
3. Evaluate other applications.
4. Use a checklist when designing panels.
5. Use the Caseworks design tool.
6. Complete an entire feature when you do it.
7. Give a new feature to someone else to use right away.
8. Solicit thoughtful feedback. Create guidelines for feedback and watch users use interfaces.
9. Perform usability walkthroughs and training.
10. Use standard modules (e.g., common dialog boxes).
 - Test for commitment. Normally there is no need for this section, but some organizations that are more tightly controlled than others may not feel empowered to implement solutions. In these organizations, solutions should be directed toward doing what the group feels it is empowered to do. When those solutions are successful, they can be more broadly or completely applied. You may need to test to identify the roadblocks to change (e.g., time, schedule, etc.).

Our example HP division seemed very committed. This was reinforced in the next step when several people volunteered to initiate specific changes.

- Plan for change. Discuss which defects can be eliminated with the proposed solution. Create an action plan with responsibilities and dates. A model action plan might contain the following steps:

1. Establish working group	10/8
2. Meet and define outputs	10/15
3. Present objectives and gather inputs	11/1
4. Create a change process and artifacts	12/1
5. Inspect and fix process and artifacts	12/15
6. Celebrate	
7. Use and measure results.	2/1

Our example division team decided to create guidelines for user interface designs that addressed many of its fishbone-diagram branches. The division's action plan consisted of the following steps.

1. Patty will create a checklist for designing panels. (First pass by 12/17)
2. The project manager will set expectations that all new functionality will be accompanied by design and specification implications. (Consider using new specification formats.)
3. Art will give the project team a presentation on Caseworks.
4. Follow up the project presentation with a discussion on the use of prototyping.

Remember to end the meeting with a clear understanding of ownership and responsibility. Use standard project-management techniques to plan and schedule follow-up.

Postmeeting:

- Review meeting process. The organization champion and root-cause facilitator review the process and develop changes to meeting format, data collection, analysis, and responsibilities. They should redo the fishbone diagram, being careful not to change it so much that participants no longer feel that it is theirs. Promptly send out meeting notes that include the fishbone diagram, responsibilities and action items, and schedule dates.
- Capture process baseline data. As part of structuring a process improvement project for success, someone (the organization champion) should record a minimum amount of process information before and after the project.² It is particularly important to document the basic divisional processes so that when the improvement is done, the group can better understand other influences besides the particular changes that were made. In this example, the team didn't do this step.

Results from Eliminating Defect Root Causes

The team from the example division did their checklist and used it during their next project. It had 30 items to watch out for, based on their previous experience and their defects. Fig. 9 shows an excerpt from their checklist. Over 20 percent of the defects on their previous project had been user-interface defects (though the division-wide average was lower). The results of their changes were impressive.

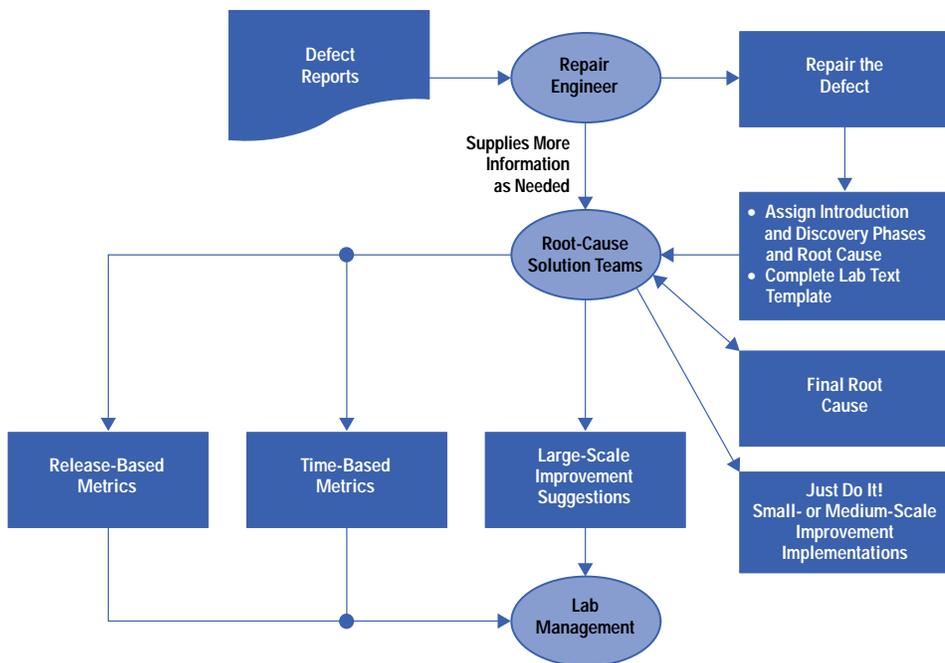
Fig. 9. A checklist of things to look for while developing dialog boxes.

-
-
-
- 7. Are fields case sensitive or not? What implications are there?
- 8. Are abbreviations kept to a minimum?
- 9. Are there any spelling mistakes on the panel?
- 10. Does the panel have a title that matches the action of the panel?
- 11. Is the screen too crowded? For data entry, less than 50 percent of the panel should be writing. Controls should "fill" the panel without cluttering it.
- 12. Is help available to the user? Is there a help line to aid the user in understanding the field?
- 13. Has the help writer been updated with information on the new panel?
- 14. Are the units for edit fields given when appropriate?
-
-
-

- They reduced the percentage of user-interface defects in test for their new year-long project to roughly five percent of their total system test defects.
- Even though the project produced 34 percent more code, they spent 27 percent less time in test.

Of course, other improvement efforts also contributed to their success. But the clear user interface defect reduction showed them that their new guidelines and the attention they paid to their interfaces were major contributors.⁸ Finally, the best news is that customers were very pleased with the user interface, and initial product sales were very good.

Fig. 10. Root-cause analysis process.



Two other project teams finished their projects recently, and their results were equally impressive. Both projects used new standard divisional specification templates created to eliminate many of the root causes shown in Fig. 8. A cross-project team task force had created two two-page specification templates (one for user-interface-oriented routines, one for software-interface-oriented ones) that they felt would help. Both teams substantially reduced specification defects compared with their previous project levels. While the reason for one team's reduction could possibly be that the project was second-generation, the other project wasn't.

While the action steps discussed here follow those of successful improvement projects at one HP division, they can also be applied in organizations with different defect patterns and business needs. One of the division people who worked with all three project teams summarized their results:

"... We must conclude that the root-cause approach is an effective mechanism to identify and introduce change into our software development process."⁹

Continuous Process Improvement Cycle

Some organizations have felt that root-cause analysis is so beneficial that they now use it to pursue continuous process improvement. It appears to be a natural evolution from post-process root-cause analysis successes. This approach extends the supporting infrastructure and requires an ongoing management commitment.

The first step that an organization generally takes is to widely adopt root-cause information logging by engineers. Causal information is then included as a normal part of the defect-handling process. Analysis is triggered in a variety of ways, often by a product or system release. Sometimes it is triggered by the end of a development phase or a series of inspections. It can also be triggered by an arbitrary time period. Fig. 10 shows how one HP division runs its process. Root-cause solution teams are empowered by management to initiate smaller process improvements.¹⁰ More far-reaching improvements still require lab management approval.

Knowing which defects occur most often in test or later helps to focus improvement efforts. We saw two examples of this in the post-project root-cause analysis discussion. The continuous process improvement cycle encourages examination of similar data throughout the development process. Take the HP division whose test data was shown as the lower-right pie chart in Fig. 4. It also captured data for specifications, design, and code inspections. All this data is shown in Fig. 11. Some caution should be used in interpreting this specific data, since it was not uniformly collected. For example, there may have been a higher percentage of design work products than code work products, but still less than there was code tested. Nevertheless, this figure suggests some interesting questions and reveals possible insights.

The bars above the centerline show counts for different defects that were found in the same phase in which they were created. Tall bars represent good opportunities to reduce these defect sources significantly. For example, the large number of module design defects suggests that a different design technique might be needed to replace or complement existing methods.

The bars below the line show counts for defects found in phases after the ones in which they were created. The later defects are found, the more expensive they are to fix. Therefore, the tall bars are sources of both better prevention and earlier

detection opportunities. For example, the requirements, functionality, and functional description defects combine to suggest that designs may be changing because of inadequate early product definition. It might be useful to use prototypes to reduce such changes.

It is clear that this type of data can contribute to more informed management decisions. It also provides a way of evaluating the results of changes with better precision than in the past. The amount of effort required to sustain a continuous process improvement cycle will vary, depending largely on the cost of implementing the changes suggested by analyses. Which changes are chosen for implementation will depend on other business aspects besides the projected costs and benefits. Just remember that the cost to sustain failure-analysis practice and modest improvements is small, and the returns have proven to far outweigh those costs.^{2,5,8}

Conclusion

Process improvement projects are started in many ways, for many reasons. In the software field especially, processes are changing and adapting daily, and software products and businesses are also rapidly evolving. One of the most effective ways to both motivate and evaluate the success of net improvements is to look at defect trends and patterns. This paper has shown how software defect data is a powerful management information source. Using it effectively will help achieve an optimal balance between reacting to defect information and proactively taking steps toward preventing future defects. HP divisions have used several successful approaches to handling defect causal data. The three root-cause analysis processes described in this paper are positioned against a suggested five-level maturity model shown in Fig. 12.

Like many other best practices, failure analysis can be applied with increasing levels of maturity that lead to different possible paybacks. HP's experience says that the biggest benefits of driving to higher maturity levels are:

- Increased likelihood of success when implementing process changes, particularly major ones
- Accelerated spread of already-proven best practices
- Increased potential returns because necessary infrastructure components are in place.

Our successful results from three failure-analysis approaches are very encouraging. While the time it takes to progress to higher maturity levels will vary among groups, our experience suggests that failure analysis starts providing returns almost immediately, particularly in visualizing progress.

Ironically, the main limiter to failure-analysis success is that many managers still believe that they can quickly reduce total effort or schedules by 50 percent or more. As a result, they won't invest in more modest process improvements. This prevents them from gaining 50 percent improvements through a series of smaller gains. Because it takes time to get any improvement adopted organization-wide, these managers will continue to be disappointed.

It has not been difficult to initiate use of the Fig. 3 defect model and the root-cause analysis process. The resulting data has led to effective, sometimes rapid, improvements. There are few other available sources of information that are as useful in identifying key process weaknesses specific to an organization. This information will help to drive process improvement decisions and commitment in an organization.

Fig. 11. A defect profile, an interesting way of analyzing defect data during the continuous process improvement phase.

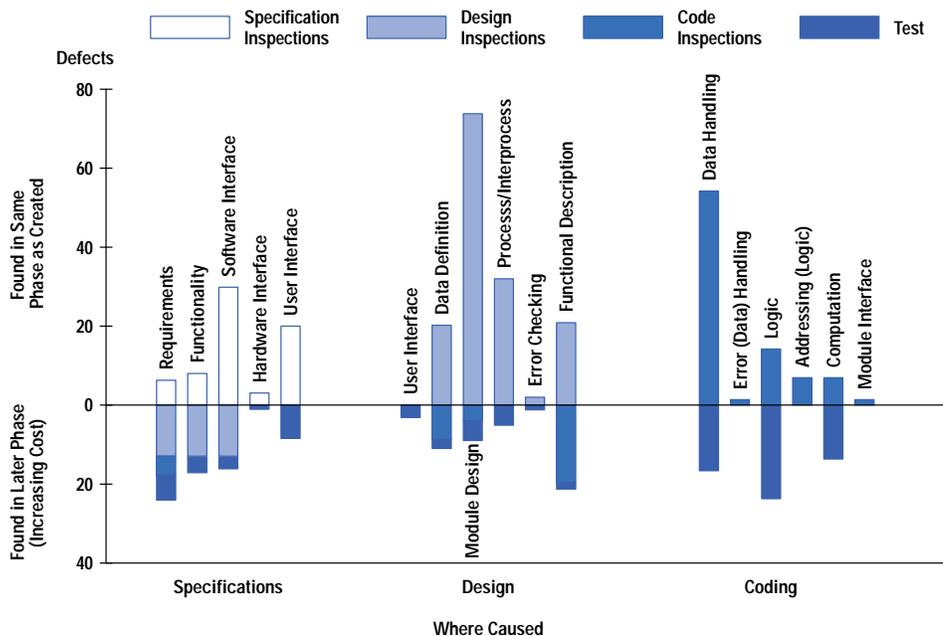


Fig. 12. A five-level software failure-analysis maturity model.

Software Failure Analysis Maturity Model

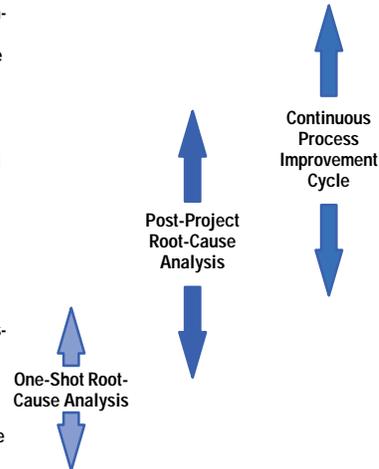
Level 5: Optimizing: Divisional goals set to achieve competitive advantage via specific software capabilities. People given primary responsibilities that include process improvement through root-cause analysis.

Level 4: Managed: Root-cause analysis meetings are a regular part of development process. There may be people responsible for improvements. Not all root-cause analysis meetings result in action items, but management reviews data.

Level 3: Defined: Defect source information uniformly collected, root-cause analysis meetings held, but not as a standard part of process. Data validating subsequent improvements is mostly anecdotal.

Level 2: Emerging: Defect source information collected, but not necessarily uniformly and not necessarily validated. General agreement on what requirements, design, and coding are.

Level 1: Initial/Ad hoc: Defect source information not regularly collected. No recognized divisional defect source patterns. Incomplete R&D process descriptions.



Acknowledgments

I'd like to thank Jan Grady, Debbie Caswell, Cate Meyers, Barbara Zimmer, Dell Fields, Tom Van Slack, and Jean MacLeod for their helpful suggestions in the development of this article. Finally, thanks to Brad Yackle, Marc Tischler, and others at HP's Scientific Instrument Division for sharing their failure-analysis results.

References

1. M. Paulk, B. Curtis, M. Chrissis, and C. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, July 1993, pp. 18-27.
2. R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Inc., 1992, pp. 37, 79, 129, 130, 137-157.
3. G. Stark, G., L. Kern, and C. Vowell, "A Software Metric Set for Program Maintenance Management," *Journal of Systems and Software* 24, 1994, pp. 239-249.
4. D. Clark, "Change of Heart at Oracle Corp.," *San Francisco Chronicle*, July 2, 1992, pp. B1 and B4.
5. R. Grady, "Practical Results from Measuring Software Quality," *Proceedings of the ACM*, Vol. 36, no. 11, November 1993, pp. 62-68.
6. K. Ishikawa, *A Guide to Quality Control*, Tokyo: Asian Productivity Organization, 1976.
7. M. Brassard, *The Memory Jogger Plus+*, GOAL/QPC, 1989.
8. R. Grady, "Successfully Applying Software Metrics," *IEEE Computer*, September 1994, pp. 18-25.
9. M. Tischler, e-mail message, Aug. 10, 1994.
10. D. Blanchard, "Rework Awareness Seminar: Root-Cause Analysis," March 12, 1992.

-
- ▶ [Go to Article 3](#)
 - ▶ [Go to Table of Contents](#)
 - ▶ [Go to HP Journal Home Page](#)