In-process improvement through defect data interpretation

by I. Bhandari
M. J. Halliday
J. Chaar
R. Chillarege
K. Jones
J. S. Atkinson
C. Lepori-Costello
P. Y. Jasper
E. D. Tarver
C. C. Lewis
M. Yonezawa

An approach that involves both automatic and human interpretation to correct the software production process during development is becoming important in IBM as a means to improve quality and productivity. A key step of the approach is the interpretation of defect data by the project team. This paper uses examples of such correction to evaluate and evolve the approach, and to inform and teach those who will use the approach in software development. The methodology is shown to benefit different kinds of projects beyond what can be achieved by current practices, and the collection of examples discussed represents the experiences of using a model of correction.

The software process[1] provides a framework for the development of software systems. Deficiencies in the definition or execution of the activities that comprise the process result in poor quality products and large cycle times. Hence, it is important to understand how the errant activities may be corrected *in process*, i.e., during the course of development.

Recently, an approach to in-process correction that involves both machine and human interpretation of classified defect data[2] has been steadily gaining momentum in IBM, and considerable experience with its use is now available. A study of this experience is presented to understand the scope and value of the approach.

The scope of the approach may be determined by asking whether there are any restrictions on the software projects that use this approach. Applying the approach to a wide variety of software development efforts can help determine the answer. Reference 2 reports that the methodology was used successfully to correct the process problems of a specific project, and examples of process correction and corroborating evidence showed that the process had indeed been corrected. This paper shows that the methodology can be successfully used with a range of different projects, thereby suggesting that there are no imposed restrictions on the kinds of projects that may use the approach.

The value gained by using the approach is described in two ways. First, the process correction examples are presented in a fashion that expresses their immediate benefit to a software developer or manager. For instance, an example

The value is in the cost saved by correcting defects in development.

may show that a correction surfaced five defects that might otherwise have escaped to be discovered by the customer in the field. Thus, the value of the methodology can be expressed in terms of the cost saved by correcting defects during development, instead of correcting them at the customer site.[3]

Second, the examples are studied to establish whether other methods could have been successfully used to make similar corrections. This demonstrates the drawbacks associated with *not* using the approach.

The organization of this paper reflects the above objectives. First, background material is presented on the process correction methodology. Second, examples of process correction from different projects are shown in order to demonstrate the value of the methodology to project teams. The experiences are used to address the issues of *scope* and *value*. Third, the steps that must be taken to correct the process are refined based on the experiences. Finally, on the basis of the evidence that is presented, we conclude that the methodology is an important advance in software process correction.

This paper contains a set of real-life experiences that can be studied by software developers and managers to understand how they can make use of an evolving process correction methodology and what it can do for them beyond current practices. Hence, in addition to its technical contribution, the paper also has educational value.

The methodology

The details of the projects that used our process improvement methodology are given in Table 1, which lists the following characteristics of each such project:

- Hardware environment
- Software environment
- Project size
- Staffing, including programmers and testers
- Tracking tools used
- Process model—waterfall or a combination of waterfall and iterative
- Parallel development—whether some project components were following a different schedule

As can be seen from the table, the projects described cover wide ranges in terms of complexity, size, and environment, and all used our process improvement methodology. The two principal activities of the methodology, namely, defect classification and analysis, are next described using material from Reference 2 to the extent needed to make this paper self-contained. The experience with one specific project (Project A) forms the basis for introducing the methodology to the reader.

Orthogonal Defect Classification. The history of software engineering is populated with numerous examples of using metrics to better manage and improve the development process. The classification of defects to identify key components is also fairly common and a good exposition on it can be found in Reference 4. There is also a proposed draft of an IEEE Standard on classification.[5] However, although most developed classification systems are useful, they are quite ad hoc. The limitation that an ad hoc measurement imposes is that the measurements are hard to validate and are much harder to leverage toward more scientific analysis or the development of a baseline. We believe a significant contribution in this arena is a technique of measurement called Orthogonal Defect Classification (ODC).[6,7]

The ideas for ODC evolved from a finding that there is a link between the semantics of defects and the maturity of a product going through the development process.[8] This led to developing classification, where the values of an attribute, called defect type, were designed to recognize the maturity through a change in the distribution

Table 1 Project characteristics

Project Name	Hardware Environment	Software Environment	Amount of New and Changed Code	Programmers and Testers	Tools Used	Process Model	Parallel Development
A	mainframe	operating system	medium	15	test defect tracking, inspection tracking	waterfall combined with iterative	yes
B	mainframe, midrange, workstation	database	very large	85	test defect tracking, inspection tracking	waterfall combined with iterative	yes
C	mainframe	operating system	small	3	test defect tracking, test case tracking	waterfall	no
D	workstation	application	very large	80	test defect tracking	waterfall combined with iterative	yes
E	mainframe	database	medium	10	test defect tracking, inspection tracking	waterfall	no
F	midrange, workstation	compiler	very large	38	test defect tracking	waterfall combined with iterative	yes
G	mainframe	operating system	large	32	test defect tracking, inspection tracking	waterfall combined with iterative	no

of the attribute values. Thus, the distributions provide an instrument to measure progress. The defect type attribute was primarily designed to provide feedback on the maturity of the product in a development process. The key was to establish the values of the attribute so that changes in the distribution explain the maturity of the product through the process. ODC measurements may provide the capability of long-term value due to process and product invariance in the measurement.

The power of measurement can be multiplied by measuring several aspects of the process using multiple attributes. This leads to multidimensional data. The implementation of the ODC scheme in IBM, reported in this study, involved five key attributes: defect type, missing or incorrect, trigger, source, and impact. Among these, the defect type and the trigger have been developed and tested to follow the guidelines of ODC and collectively provide causal information. Referring to Table 2, the *impact*, as the name suggests, measures an effect. *Source* partitions the product in terms of the developmental history of the code, identifying subpopulations that may be interesting. Thus the categories collectively provide a measure of cause and effect, critical to the improvement of the process or product. These data over the project life cycle may provide long-term value in terms of baselines and sophisticated models.

For the purpose of this paper it is necessary to understand the values of the ODC attributes that are referenced. Apart from the engineering aspects of identifying the values there is a usability and human aspect to it that is equally important.

In particular, the choices have to be few so that it improves accuracy, and the education process must be sound. *Defect type* and *missing or incorrect* capture information about the type of activity that was undertaken to fix the defect. For instance, the type of a defect is *function* if it had to be fixed by correcting major product functionality. A defect is classified *missing* if it had to be fixed by adding something new, and classified *incorrect* if it could be fixed by making an in-place correction. *Source* identifies the partition in which the defect is located, i.e., it captures whether the defect occurred because of errant activities in previous releases or the current release. For instance, a defect is classified *new function* if it was found in that part of the product that consisted of new code, and classified *rewritten* if, instead, it was found in code that was part of an old release of the product but was being rewritten for the present release. *Trigger* captures information about the specific inspection focus or test strategy that caused the defect to surface. For instance, a defect found by thinking about the flow of logic of a design or implementation is classified *operational semantics*, while a defect found by thinking about the compatibility of the current release to previous releases is classified *backward compatibility*. *Impact* captures information about customer activities that would be affected should the defect have escaped into the field. For instance, the impact of a defect is classified *capability* if, had it escaped to the field, it would have affected the functionality of the product adversely; is classified *usability* if instead, it would have affected only the ease with which the customer could use the product; is classified *performance* if it would have affected only the performance of the product but not its capability.

In addition to these measurements, there has traditionally been a set of process- and product-specific measurements that are collected by defect tracking tools. The project team also classified defects by using the attributes phase found, phase introduced, and component. *Phase found* identifies the developmental phase at which a defect was found, while *phase introduced* identifies the phase at which it was introduced. For instance, the phase introduced is classified CLD if the defect was introduced during component-level design, or classified MLD if it was introduced during module-level design. *Component* identifies the software component in which the defect was located. Clearly, those attributes also relate a defect to a specific set of process activities. For instance, if a defect is located in Component A, we know that the activities used to develop Component A are responsible for the defect being introduced.

Table 2 Classification attributes

Attribute	Description
Defect type	Determines correction method
Missing or incorrect	Corrected by adding something new or making in-place change
Trigger	Captures information that caused the defect to surface
Source	Partitions developmental history of the code
Impact	Measures an effect
Where:	
Defect type	= Function, interface, assignment, checking, data structure, document, etc.
Trigger	= Operational semantics, language dependencies, concurrency, side effects, document consistency, lateral compatibility, backward compatibility, rare situation, design conformance, etc.
Source	= New function, rewritten code, etc.
Impact	= Capability, usability, performance, etc.

The project team members classified all defects found during the reviews of all deliverables produced (i.e., final programming specification document, design structures document, logic manuals, and code) as well as during the execution of the test phases. Having no prior experience, the analysis step was integrated into the process of a project in the most obvious fashion. The classified defects were analyzed after every phase of the process. Thus, the analysis was done after phases such as component-level design, module-level design, code, unit test, function test, and system test. The process of the project was adjusted to reflect the results of every analysis before proceeding to the next phase. The analysis step is described next.

Defect analysis and feedback. The second step of the methodology is the analysis of the defect data

using an approach to machine-assisted data exploration called attribute focusing. Details of the approach may be found in Reference 9. Other experiences based on the application of attribute focusing (AF) to software development have been reported. AF was used[10] to analyze defect survey data to suggest improvements *post-process*, i.e., after the end of development. AF was utilized[6] to assess the effectiveness of inspections and testing methods. AF was also used[2] to make in-process improvements to the process of a single project. Those works did not address the scope and value of using attribute focusing for in-process improvement, which is the subject of this paper.

The approach is discussed next. The goal of attribute focusing is to provide a systematic way for a domain specialist, who may not be skilled at data analysis, to analyze data that are classified across many different attributes. It targets the lay person instead of the data analyst, a goal that distinguishes it from the usual data exploration system (see, for example, systems described in Reference 11). The typical software developer or tester is considered to be a domain specialist in this context.

The key aspects of the approach are illustrated in Figure 1. Information is abstracted from a physical situation to create an attribute-valued data set. For our purpose, a record of the data set represents a defect found by the project team during the course of development and classified using the attributes described in the previous section, Orthogonal Defect Classification, along with a written description of the defect. A data set consists of data from all the defects available at the end of a phase or a stage within the phase of the project. A project can start using the methodology at any stage, and examples of projects that started as early as component-level design or as late as function test will be described.

Continuing with the description of attribute focusing (Figure 1), the classified defects in a data set are processed automatically to produce a set of interesting charts that are interpreted by the project team in a specific manner. We present the mechanical and manual procedures of attribute focusing, and exemplify each procedure using data from one specific project (Project A).

Interestingness and filtering functions. First, the classified data are processed automatically by a procedure called an *interestingness function* that orders attribute values to reflect their potential *interestingness* for a human analyst. (Ashwin Ram[12] is credited with coining the term "interestingness.") A set of attribute values corresponds to a set of defects, namely, the defects that were classified using those attribute values.

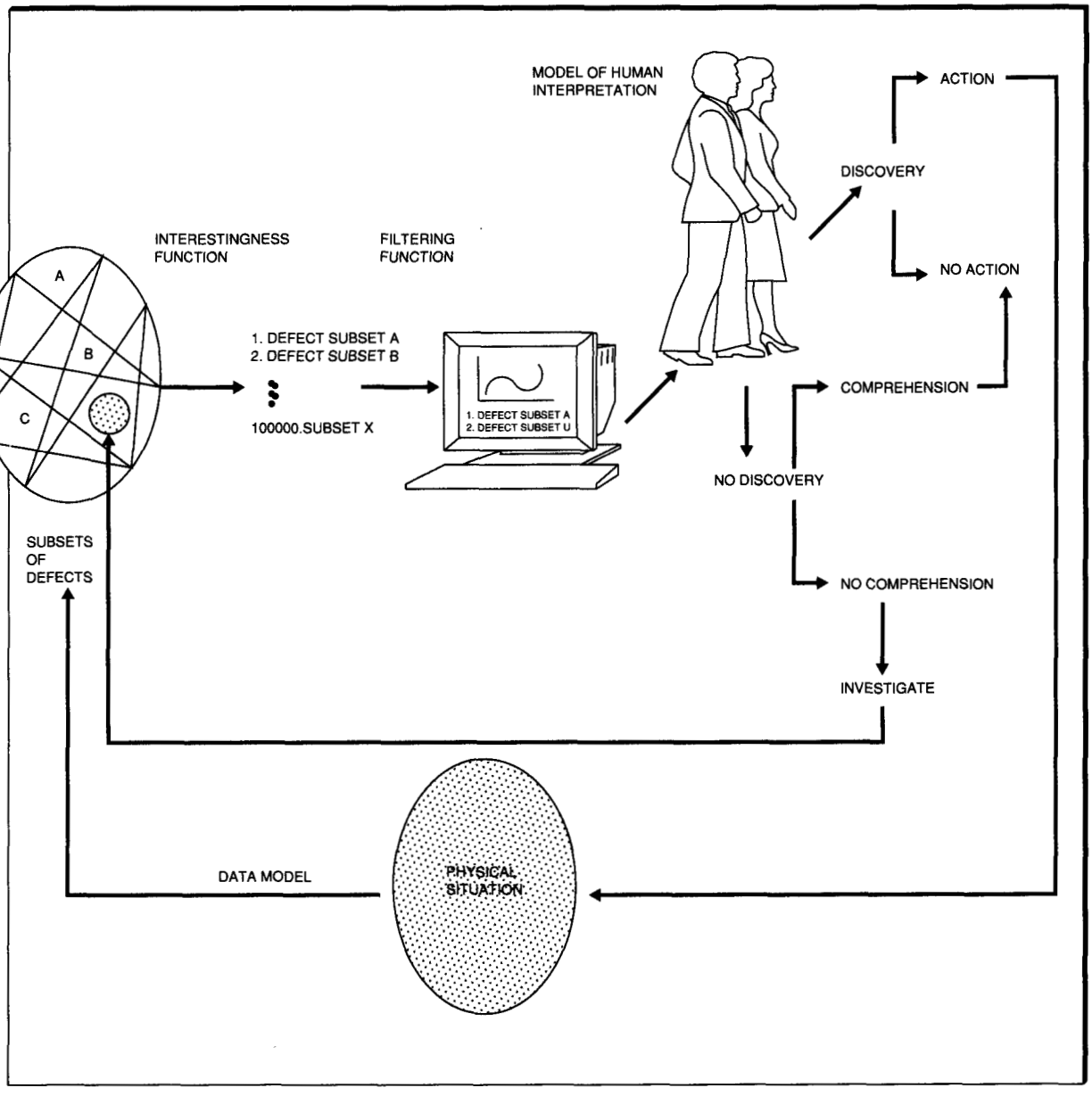
The interestingness function orders attribute values.

Hence, Figure 1 depicts the ordering of attribute values by ranking subsets. The data of the project teams were processed by using two interestingness functions to order attribute values based on a degree of magnitude, and pairs of attribute values based on a degree of association. The use of such functions is quite common in data exploration and in machine learning. Heuristics, which are commonly used to search for interestingness, include measures of magnitude, association, correlation, and informational entropy.[11,13]

Second, another automatic procedure called a *filtering function* processes the orderings produced by an interestingness function and presents it in a manner suitable for human consumption. It makes use of knowledge of human processing of attribute-valued data to do this. The use of such a filtering function is a novel idea, although it is in keeping with the recent emphasis on interactive approaches for data exploration.[14]

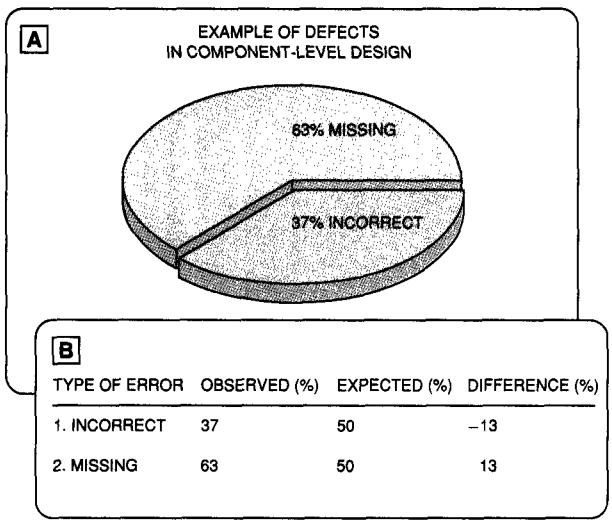
Concepts from Project A. Let us understand the above concepts in the context of Project A. A filtering function was used to produce charts that show the spread of values for an attribute, such as shown in Figure 2, or which show the cross-product of two attributes such as shown in Figure 3. The tables in the charts present the values of the attributes in decreasing order of their interestingness. As we shall see, that information is used to focus the project team on certain trends in the data. Usually, there are eight numbered items or less, which is in keeping with a published observation that people find it difficult to retain more

Figure 1 The attribute focusing approach



than seven plus or minus two items in short-term memory.[15] Furthermore, the charts are produced in decreasing order of interestingness, and only so many charts are produced as are reasonable for a person to interpret at one sitting. Based on knowledge of the limits of human processing and a calibration of the average time it takes a person to interpret one chart pertaining to defect data, it has been our experience that the number of charts produced should be restricted to less than or equal to 20 (to limit the duration of an interpretation session to about two hours).[9] We observed that restriction in our implementation. Once the project team had classified all the data for a phase,

Figure 2 Missing or incorrect type of error for Project A



TYPE OF ERROR	OBSERVED (%)	EXPECTED (%)	DIFFERENCE (%)
1. INCORRECT	37	50	−13
2. MISSING	63	50	13

our implementation took less than five minutes (of wall clock time) to generate the 20 charts (referred to as AF charts) to be used to analyze the data for that phase. That cost of chart generation is negligible and, hence, the overhead of using attribute focusing is about two hours per phase of the process, that being the duration of the feedback meeting. For this paper, the original charts generated by the authors were simplified and the method of display illustrates only the information pertinent to the discussion.

Next, let us understand the information in the AF charts to gain a better understanding of the interestingness and filtering functions. Much of the data for Project A was generated by classifying defects found during the inspections[16] of the component-level design and module-level design documents.

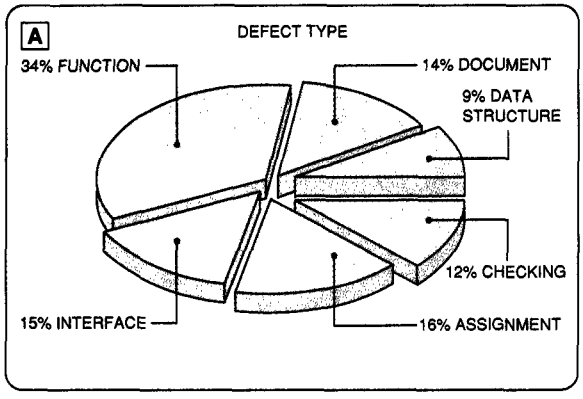
The AF chart in Figure 2 was among the 20 charts generated for the data from the early component-level design inspections of Project A. Figure 2A shows the distribution of the attribute *missing or incorrect*. The table in part B indicates the interestingness of the attribute values based on a degree of magnitude. Let us understand the first row of this table. The *observed* column indicates that 37 percent of the defects were classified *incorrect*. The *expected* column shows what the percentage of such defects would have been had the values of the attribute been equally likely. Since *missing or incorrect* has two values, that percentage is 50 percent. The *difference* column is the difference between 37 percent and 50 percent, or −13 percent.

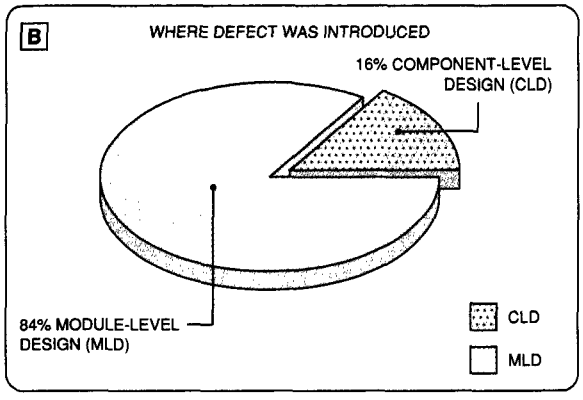
We can now get a better understanding of the relationship between the interestingness and filtering functions. The interestingness function computes the difference for every attribute value and then lists attribute values in decreasing order of the absolute value of the differences. The filtering function uses the ordering to decide which charts should be brought to the attention of the analyst. Starting from the top, the filtering function selects the attribute values and hence, the attributes to be charted. Once 20 charts have been selected, the procedure terminates.

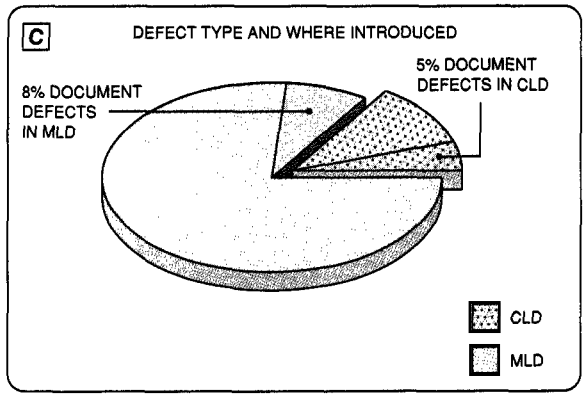
The AF chart in Figure 3 shows the cross product of the attributes *defect type* and *phase introduced*. The chart was among those generated by processing the data from the module-level design inspections of Project A. The table in part D indicates the interestingness of the attribute values based on a degree of association. Let us understand the first row of the table. The column *type* indicates that 14 percent of the defects had a *defect type* of *document* (see also Figure 3A). The *phase* column indicates that 16 percent of the defects were introduced in component-level design (CLD) (see also Figure 3B). Column *type and phase* indicates that 5 percent of the total defects were introduced in CLD and were of type *document* (see also Figure 3C). Had the classification of defects as *document* and CLD been statistically independent, we would have expected to find that $(0.14 * 0.16 = 0.2)$, or 2 percent, of the defects had been classified *document* and CLD, as indicated in the *expected* column. The difference (5% − 2% = 3%) is indicated in the *difference* column, and serves as a measure of the degree of association between the attribute values in the first row.

Interpretation of AF charts. Let us continue with the description of the attribute focusing approach. As indicated in Figure 1, the AF charts are interpreted by a domain specialist or analyst by using a *model of interpretation*.[9] While a complete discussion of the model of interpretation is beyond the scope of this paper, its use will be adequately illustrated later in this section. The specialist performs the following for every AF chart. There is an attempt to explain the interest-

Figure 3 Example of defect types and where they were introduced for Project A







D

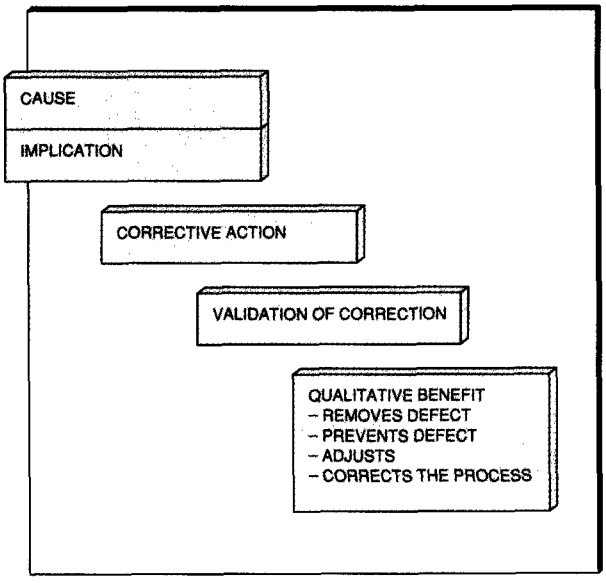
DEFECT TYPE	PHASE INTRODUCED	TYPE (%)	PHASE (%)	TYPE AND PHASE (%)	EXPECTED (%)	DIFFERENCE (%)
1. DOCUMENT	CLD	14	16	5	2	3
2. DOCUMENT	MLD	14	84	8	11	−3
3. ASSIGNMENT	MLD	16	84	16	14	3
4. ASSIGNMENT	CLD	16	16	0	3	−3

WHERE

TYPE = PERCENT OF A (COLUMN 1) DEFECT TYPE
PHASE = PERCENT OF TOTAL DEFECTS THAT WERE INTRODUCED IN A (COLUMN 2) PHASE
TYPE AND PHASE = PERCENT OF TOTAL DEFECTS OF A (COLUMN 1) TYPE AND INTRODUCED IN A (COLUMN 2) PHASE

ingness (magnitude or association) of the attribute values that appear in the chart by relating the interestingness to events in the domain. Specifically, the cause of the interestingness must be determined, and the implication of the interestingness considered in the context of the domain. Such consideration is given to every item in the table of the AF chart. Some items are easily understood by the specialist and do not lead to new insights. However, some items do lead the spe-

Figure 4 Model of interpretation



cialist to a better understanding of the domain and, consequently, to action that reflects the new realization.

The model of how the analyst gains such an understanding is exemplified below and in Figure 4 for a table that shows that a is associated with b but is disassociated with w, where a, b, w are attribute values.

- Understand the cause of interestingness
  - a occurs frequently with b but infrequently with w.
  - Why?
  - What event could have lead to such an occurrence?

- Understand the implication of interestingness
  - a occurs frequently with b but infrequently with w.
  - Is that desirable?
  - What will happen if no action is taken?

Recall that the analyst is a domain specialist who may not be skilled in data analysis. Note that the questions above do not make reference to data analysis terms and concepts. Instead, they encourage analysts to think directly about events in their domain. In fact, analysts are also encouraged to consider the *absence* of any table items, which their domain experience has taught them to expect. As shown in Figure 1, that exercise results in either the discovery or no discovery of knowledge. The *discovery of knowledge* is where the explanation of the interestingness leads to a new insight about the domain. In our context, this translates to the identification of a process problem and the implementation of a corrective action. Occasionally, there is no need to implement a corrective action because the problem has already been resolved. With *no discovery of knowledge* there are two possibilities. Either the explanation of the interestingness is something already known to the analyst, or the analyst cannot comprehend the interestingness. In the former case, no action is necessary. In the latter case, known as the *investigate* step, the interestingness must be investigated by the analyst at a later time. The interestingness is based on the magnitude of a particular attribute value or the association of a pair of attribute values. The analyst must study the detailed written descriptions of a sample of defects that were classified by choosing those attribute values in an attempt to explain the observed interestingness.

Let us exemplify the interpretation process using the AF charts for Project A. To interpret the chart in Figure 2, the project team played the role of analyst. They had to explain the magnitude of the attribute values in the table by determining the cause and considering the implication of the magnitude. If that led to the identification of a process problem, the team had to come up with a corrective action. The large proportion of *missing* defects in the chart led to the discovery and correction of a process problem in Project A.

We use the headings *cause*, *implication*, and *corrective action* to describe the results of the different steps in the model of interpretation given above.

- *Cause*—The large proportion of *missing* defects was attributed to lapses in communication between designers who were working in separate subteams. These lapses were manifested by functionality that was missing in the design, a fact that was discovered during inspections. Repeated inspections of the design were then done until the team felt confident that the design was complete. Thus, a process problem was identi-

fied: the lack of communication between different subteams.

- *Implication*—The product may be shipped with incomplete functionality.

- *Corrective action*—The use of "teach-the-team" sessions, wherein individual team members periodically present their work to the entire team, was planned. These sessions would prevent lapses in communication. To show that the methodology achieves what it is meant to do, namely in-process identification and correction of process problems, every experience is also evaluated along the dimensions of (1) corroboration of process problems identified by information that was not captured by the defect data but was known to the team and used to make their decision, and (2) validation of corrective actions undertaken by looking for appropriate trends in defect data collected after those actions were implemented.

  The relevant information for process corroboration of the process problem in Project A was consensus among team members that they had not communicated, the nature of the *missing* defects as described in their written descriptions, and the fact that the defects were found in inspections that involved members across different subteams.

- *Validation of correction*—The injection rate of missing defects dropped from 63 percent to 44 percent for CLD completed after the start of teach-the-team sessions. Since the reduced rate applied to about 80 percent of the product, the savings were significant. As we determined from the interpretation of the Figure 2 AF chart, the team made use of knowledge that was not captured by the data to reach a decision. For instance, what the data did not indicate was the feeling that communication between subteams was poor, the fact that defects were found when subteams did communicate via inspections, and the nature of the missing defects as evident from their written descriptions. Only a person who is very familiar with the project could have such knowledge and apply it in a few minutes to interpret the AF chart. A quality expert who is not part of the project team will not have that knowledge and hence, may quite possibly misinterpret the AF chart.

- *Benefit*—The dimensions just discussed help establish that an experience indeed represents the successful identification and correction of problems. To provide a better understanding of the value of such correction, it is shown that every experience results in one or more benefits, such as (1) removes defects, which were latent, (2) prevents defects from being injected, (3) adjusts the course of the project to avoid a problem or exposes a problem that the project team does not know how to avoid, which is a near-term benefit since it only affects the current release, and (4) corrects the process used by the project or exposes a process weakness that the project team does not know how to correct, which is a long-term benefit since it should also affect successive releases that use the same process.

  Therefore, in the Project A discussion (Figure 2), the benefit was that the corrective action prevents defects, and identifies and corrects a deficiency in the process used by the team. The teach-the-team sessions should be part of the process for parallel development, to improve communication.

Next, we describe the interpretation of the AF chart in Figure 3 using the dimensions described above. Recall that the chart was among those generated from data of the module-level design inspections of Project A. To interpret the chart, the project team had to explain the association of the attribute values in the table or the absence of an expected association of attribute values in the table, by determining the cause and considering the implication of the association. If this interpretation led to identifying a process problem, the team had to come up with a corrective action. The associations shown in the table were easily explained by the project team. For instance, Item 3 in Figure 3D, shows an association between defects that were fixed by correcting assignment statements and the module-level design (MLD) phase, while Item 4 shows a disassociation between such defects and component-level design (CLD). The team noted those associations were to be expected since at component-level design, the nature of the work done was at too high a level to include specification of assignment statements. That level of detail was addressed in MLD. However, the *absence* of a row which showed a strong association between the defect type *function* and phase introduced *CLD* led to the identification of

another process problem in Project A. The absence of a pair of attribute values indicates this pair is neither strongly associated nor strongly disassociated. However, a strong association was expected since functional aspects of the design are addressed during CLD. The reason for the absence of such association became clear when the data (not shown in the figure) showed that a large number of *function* defects were introduced in MLD.

- *Cause*—CLD work was still being done after the beginning of the MLD stage. The corroboration of the process problem showed that (1) a missing item in the requirements list was responsible for an incomplete component-level design and the defect was discovered and fixed in the module-level design stage, (2) a formal requirements document did not exist and individual team leaders maintained their own list of requirements, and (3) there was consensus among team members that, owing to the varied customer base of the product, some requirements were hard to determine.

- *Implication*—Module-level design should only be done after component-level design is complete, else the process is not being followed. Serious consequences can include missed schedules and poor quality.

- *Corrective action*—The requirements material and the final programming specification were reinspected for completeness.

- *Validation of correction*—No further missing requirements were found through test.

- *Benefit*—The corrective action removes defects and is an adjustment that prevents shipping a product with missing functionality. It also identifies a deficiency in the process: the product is too complex to avoid a formal requirements process. A suitable process change should be made for future releases.

Experiences from Projects B and C

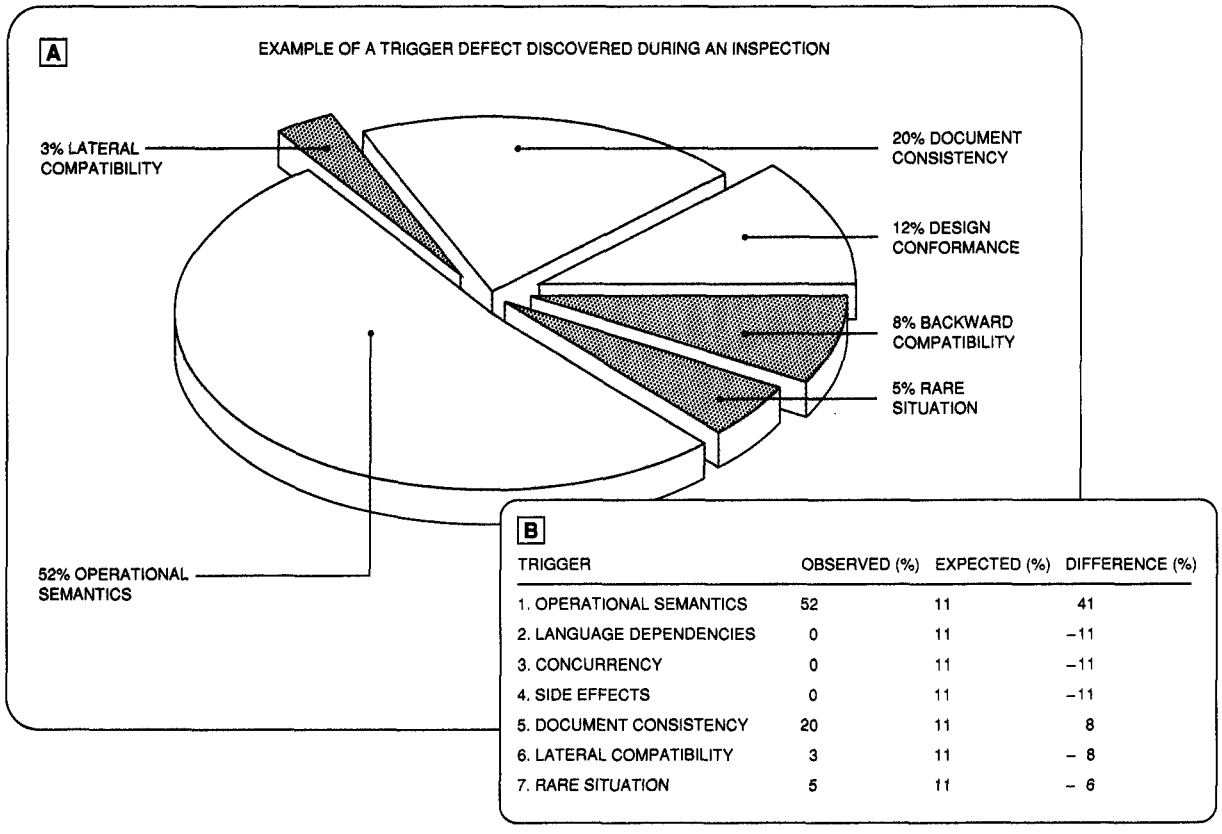
Having described our process improvement methodology, we now present the experiences from a number of different projects (see Table 1). The experiences are presented in two parts, separated by the section "Comparison with other feedback techniques." That separation allows the reader who is interested mainly in the principles underlying our methodology to skip the second set of additional examples. The additional examples do not introduce new principles but they do serve an auxiliary purpose. First, they serve as an educational aid for readers who wish to use and deploy our methodology. Second, they provide additional evidence to support the main result of this paper, namely, that our methodology works for different kinds of projects.

The descriptions of the project experiences use the same dimensions as were exemplified in the earlier subsection "Interpretation of AF charts" for Project A, and illustrate that the methodology provides benefits in four ways. It helps remove defects, it helps prevent defects, it helps identify corrective actions for the immediate project, and it helps identify problems with the underlying process being used.

While a quantitative expression of the value of all corrective actions will only be complete once field data exist, it is possible to quantify the net benefit of corrective actions that remove defects. For instance, if such a correction led to the detection of five defects that would otherwise have been found in the field, the cost of finding the defects in the field and the cost of the correction may be used to calculate the net benefit. Relevant experiences are presented by expressing the value of the corrective actions in both qualitative and quantitative terms. To determine the quantitative value, the defects removed as a consequence of a corrective action were studied by the project team to determine the number of defects that would have escaped to the field had the situation not been corrected. The basis for such determination is described and the total cost of dealing with those field defects computed by using 15 person days to be the cost of dealing with a single field defect (note that this number may underestimate the impact of such defects).[3] Similarly, the costs of finding defects in function test (FT) and system test (ST) are assumed to be one and three person days, respectively.

Experiences from Projects B and C are used to illustrate the principles learned about the methodology. Later in the paper (after the section on comparison with other techniques) more experiences from Project A and the experiences from Projects D, E, F, and G are used to only further support the ideas presented. Each subsequent

Figure 5 A trigger in component-level design for Project B



B

TRIGGER	OBSERVED (%)	EXPECTED (%)	DIFFERENCE (%)
1. OPERATIONAL SEMANTICS	52	11	41
2. LANGUAGE DEPENDENCIES	0	11	−11
3. CONCURRENCY	0	11	−11
4. SIDE EFFECTS	0	11	−11
5. DOCUMENT CONSISTENCY	20	11	8
6. LATERAL COMPATIBILITY	3	11	− 8
7. RARE SITUATION	5	11	− 6

experience is introduced in terms of the problem that was identified by the team.

Experience of reviewers not considered. Project B used a textual document called a functional programming specification to describe component-level design, and a team of inspectors read through the document to validate the design.

The AF chart in Figure 5 shows the distribution for the attribute *trigger*, which captures what the inspector was looking for when the defect was found. The relatively small magnitude of defects that were found by considering *lateral compatibility*, *rare situation*, and *backward compatibility* led to the discovery of a process problem.

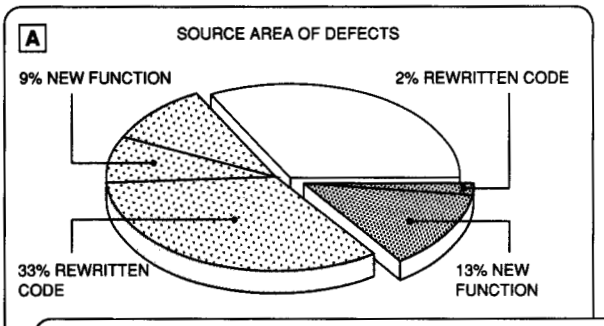
Cause. Review was limited by the experience of the reviewers. Compatibility issues and pathological scenarios were not considered adequately. Corroboration of the process problem indicated that (1) there was a consensus among reviewers that they were uncertain about compatibility issues, (2) reviewers did not have extensive experience with the product, and (3) the compatibility requirement for the product was not met.

Implication. Existing customer applications may fail when the new release is installed, and the new release may fail when used with other products.

Corrective action. An experienced person reviewed the document with focus on identifying missing external information, particularly in *backward compatibility* and *lateral compatibility*.

Validation of correction. An experienced reviewer found 38, 18, and 1 additional defects classified *backward compatibility*, *lateral compatibility*, and *rare situation*, respectively.

Figure 6 Source and trigger in function test for Project C



B

SOURCE	TRIGGER	SOURCE (%)	TRIGGER (%)	SOURCE AND TRIGGER (%)	EXPECTED (%)	DIFFERENCE (%)
1. NEW FUNCTION	TEST COVERAGE	37	52	9	19	−10
2. NEW FUNCTION	SIMPLE PATH COVERAGE	37	15	13	5	7
3. REWRITTEN CODE	TEST COVERAGE	52	52	33	27	6
4. REWRITTEN CODE	SIMPLE PATH COVERAGE	52	15	2	8	− 6

Qualitative benefit. The corrective action removes defects and is an adjustment that prevents shipping a product that is incompatible with previous releases and other products. It also identifies a weakness in the process used to select reviewers, namely, it does not adequately consider breadth and depth of experience of reviewers.

Not function testing all areas directly. In the Project C experience, all the problems were identified after the function test stage, and function test was restarted after all corrective actions had been implemented. This made it easy to assess the effectiveness of the corrective actions since the result could be compared using the same function test process before and after the corrective actions. Furthermore, the product had a main path of execution, which was to be the main focus of system test cases. Hence, by determining if they were on or off the main path, it was easy to establish if defects found by implementing corrective actions would have escaped to the customer. In the text below, such defects are referred to as *field defects*. Recall that the cost of correcting each defect detected in the field is assumed to be 15 person days.

Items 1 and 3 in the table of the AF chart in Figure 6 show that *new function* is disassociated with *test coverage*, while *rewritten code* is associated with *test coverage*. More defects were found in that part of the product that was being rewritten using a function test strategy called *test coverage* than were being found in the part of the product that was being newly developed. This trend led to the identification of a process problem.

Cause. Function test was doing a better job of testing the rewritten code than the new function. Corroboration of the process problem indicated that (1) the purpose of function test was to cover two areas: Area 1, consisting of old and rewritten code, and Area 2, a new function used by Area 1; and (2) for function testing, the product was configured as depicted in Figure 7, namely, to test Area 2 through Area 1, i.e., by using Area 1 to manipulate the function in Area 2. Since Area 2 is manipulated using Area 1, it is easier to test the functionality of Area 1 than Area 2.

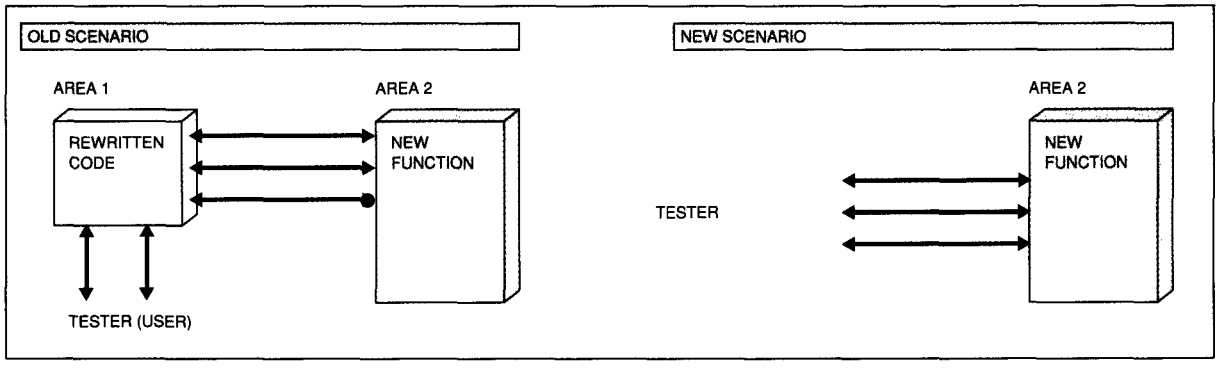
Implication. Defects may remain latent in Area 2.

Corrective action. Area 2 was tested directly as shown in the new scenario in Figure 7.

Validation of correction. Two field defects were found.

Qualitative benefit. The corrective action removes defects and is an adjustment to ensure

Figure 7 Correcting the test strategy for Project C



there are no latent defects in the Area 2. It also identifies a weakness in the function test process, namely, the use of a policy of not function testing any area that cannot be manipulated directly. The process should be corrected to consider situations such as this and determine if function test is even appropriate.

Quantitative benefit. The cost of corrective action was 25 person days to generate new test cases and execute them. The total benefit was two field defects found, potentially saving 30 person days, and the net benefit was five person days.

Design not re-evaluated. Item 1 in the table of the AF chart in Figure 8 shows that *interface* errors are associated with *test coverage*, indicating it was easier to find defects in interfaces in function test than other kinds of defects.

Cause. The new environment was not accounted for in the original software design. Corroboration of the process problem indicated that (1) function test variations in a new environment revealed 13 defects, and (2) design documents did not address the new environment.

Implication. Errors pertaining to the new environment may remain in the code after function test completion.

Corrective action. Function test was suspended while additional code inspections were completed that focused on eliminating errors in the new application environment.

Validation of correction. Only one new error was found relating to the new environment in the continuation of function test after the code inspections were completed, and the code reviews found eight field defects.

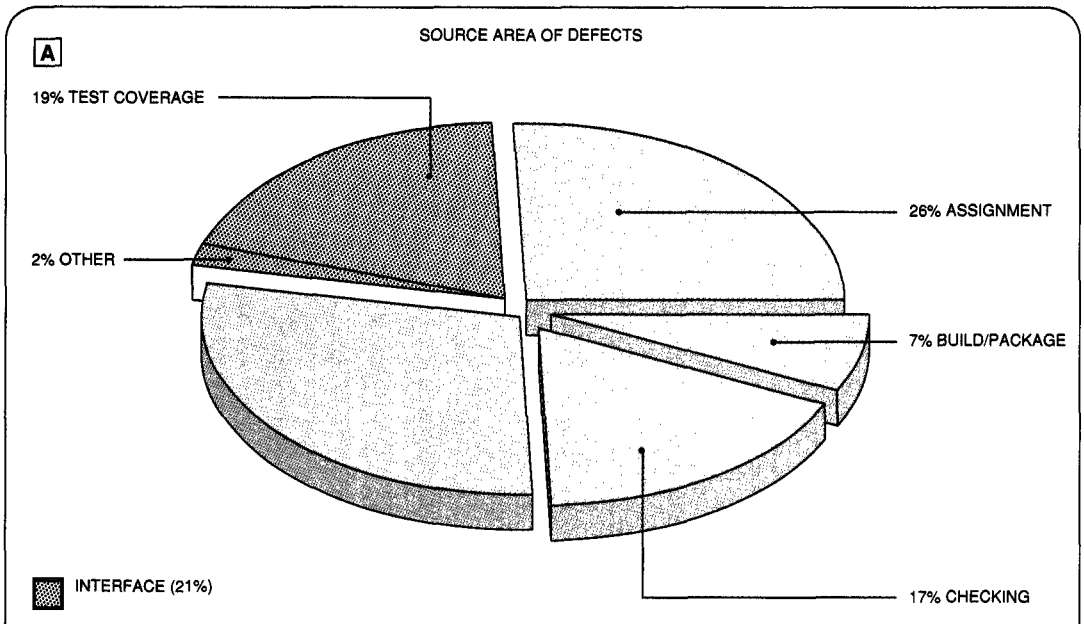
Qualitative benefit. The corrective action removes defects and is an adjustment to make sure there are no latent defects pertaining to the new environment. It also identifies a weakness in the planning process for the product. This problem was caused because the project was originally designed for an earlier version of the host operating system. When it was moved to a different product release, the design was not re-evaluated to consider the new operating environment. This can be corrected by requiring projects to re-evaluate design if the product release level running the function is changed.

Quantitative benefit. The cost of corrective action was 20 person days to complete code reviews. The total benefit was eight field defects found, potentially saving 120 person days, and the net benefit was 100 person days.

A weakness testing complex operations. Item 1 in the table of the AF chart in Figure 9 shows that *rewritten code* is associated with an impact to *usability*, suggesting that defects in the rewritten code are more likely to affect customer usability than other kinds of impact.

Cause. The team identified the programs that consisted of *rewritten code* and then considered

Figure 8 Defect type and trigger for Project C



B

DEFECT TYPE	TRIGGER	TYPE (%)	TRIGGER (%)	TYPE AND TRIGGER (%)	EXPECTED (%)	DIFFERENCE (%)
1. INTERFACE	TEST COVERAGE	21	52	19	11	8
2. CHECKING	SIMPLE PATH COVERAGE	17	15	9	2	7
3. BUILD/PACKAGE	TEST COVERAGE	7	52	0	4	−4
4. BUILD/PACKAGE	SIDE EFFECTS	7	26	6	2	4
5. INTERFACE	SIDE EFFECTS	21	26	2	5	−3
6. CHECKING	TEST COVERAGE	17	52	6	9	−3
7. INTERFACE	SIMPLE PATH COVERAGE	20	15	0	3	−3
8. ASSIGNMENT	SIDE EFFECTS	26	26	9	7	3

which of those would be most susceptible to *usability* problems. They concluded that the program that handles operations requiring more than one tape drive (referred to as multi-reel operations) was more error-prone because (1) it involves more complex and end-user interaction, (2) function test variations had surfaced six defects running multi-reel functions, and (3) unit test had not executed a multi-reel variation.

Implication. Multi-reel operations may not have been adequately designed or tested. Customers may identify additional problems running such operations.

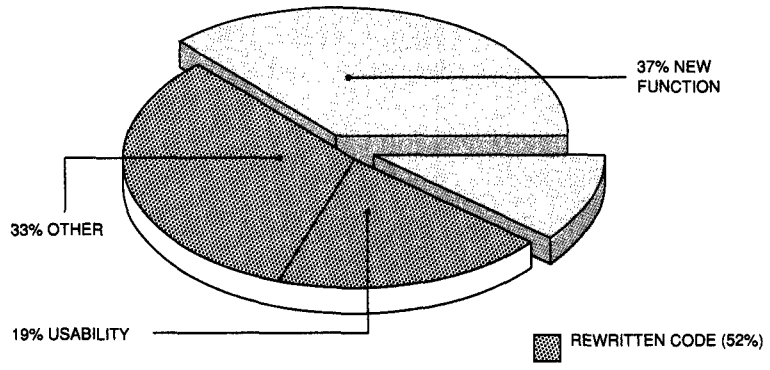
Corrective action. Additional code inspections were completed that focused on the error-prone program. This was followed by additional unit test variations using multi-reel operations.

Validation of corrective action. Four field defects were found, and no new errors were found in the error-prone program after the corrective actions were implemented.

Qualitative benefit. The corrective action removes defects and is an adjustment to make sure there are no latent defects in multi-reel operations. It also identifies a weakness in the devel-

Figure 9 Source and impact for Project C

A



B

SOURCE	IMPACT	SOURCE (%)	IMPACT (%)	SOURCE AND IMPACT (%)	EXPECTED (%)	DIFFERENCE (%)
1. REWRITTEN CODE	USABILITY	52	28	19	14	4
2. NEW FUNCTION	USABILITY	37	28	7	10	−3
3. NEW FUNCTION	INSTALLABILITY	37	4	4	1	2

opment and testing of complex operations. The experience suggests that the process for the product must distinguish between simple and complex operations. Hence, the project team suggested that a checklist of complex environments be maintained and considered during development of future releases.

Quantitative benefit. The cost of corrective action was 10 person days to complete reviews. The total benefit was four defects found, potentially saving 60 person days, and the net benefit was 50 person days.

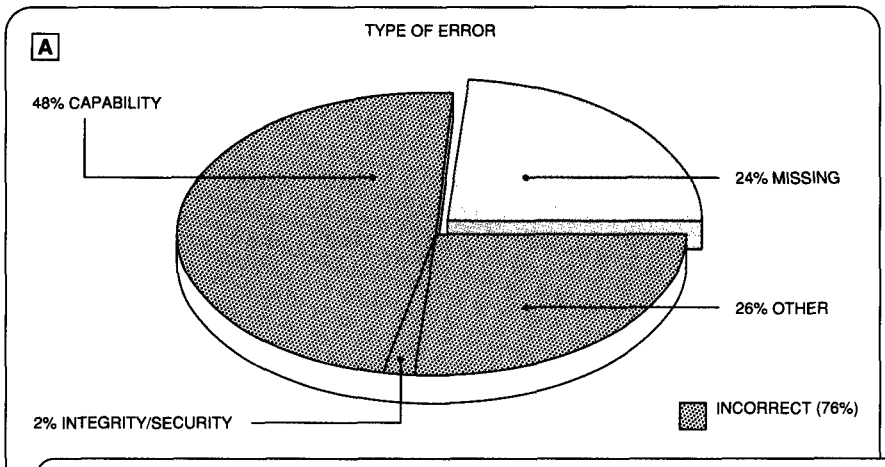
Not distinguish types of execution. The following trend lead to the identification of a process problem. Refer to Item 2 in the table of Figure 10. Errors that impact the *capability* of the component are associated with being *incorrect*. Item 1 of Figure 11 shows that *capability* defects are being found during function test using the more simplistic *test coverage* strategy. Item 3 of Figure 12 shows that *timing* and *serialization* problems are being found that impact the *capability* of the product.

The trend suggests that the product has its capability in place, but even function test cases that test the capability in obvious ways detect timing and serialization problems.

Cause. The product was unstable with regard to events that could occur in different sequences. Corroboration of the process problem indicated that (1) the component was a complex function with many different events that could occur at any time and in different sequences. Thus, while it was easy to implement the capability of the product corresponding to the obvious sequence of events (the main-line execution), it was much harder to cover the unexpected sequences that could occur, and (2) there was consensus among the team that such sequences had not received special emphasis.

Implication. An important part of the capability of the product was to allow the user to restore a corrupted database. Defects that affect this capability may still exist after function test is completed, suggesting that customers may not be able to recover lost data.

Figure 10 Impact of a missing or incorrect type of error for Project C



B

MISSING OR INCORRECT ERROR	IMPACT	TYPE (%)	IMPACT (%)	TYPE AND IMPACT (%)	EXPECTED (%)	DIFFERENCE (%)
1. MISSING	CAPABILITY	24	57	9	14	−5
2. INCORRECT	CAPABILITY	76	57	48	44	5
3. MISSING	INTEGRITY/SECURITY	24	7	6	2	4
4. INCORRECT	INTEGRITY/SECURITY	76	7	2	6	−4

Corrective action. Additional test variations that forced the component to execute events in different orders were executed. The focus of this change was in function test, where these types of defects should be surfaced, but unit test also attempted some of these variations.

Validation of correction. Four field defects were found.

Qualitative benefit. The corrective action removes defects and is an adjustment to make sure there are no latent defects that affect the restore operation. It also identifies a process weakness that is similar to the weakness identified in the previous process problem. While the product has at least two types of execution that have very different levels of development complexity, the process used does not distinguish between them. The problem was caused by the large number of event sequences that could be generated by the component. An automated tool that can randomly generate different event sequences will help in bringing more of these defects to the surface. The use of such a tool must be made part of the process for future releases.

Quantitative benefit. The cost of corrective action was 15 person days for additional testing. The total benefit was four defects found, potentially saving 60 person days, and the net benefit was 45 person days.

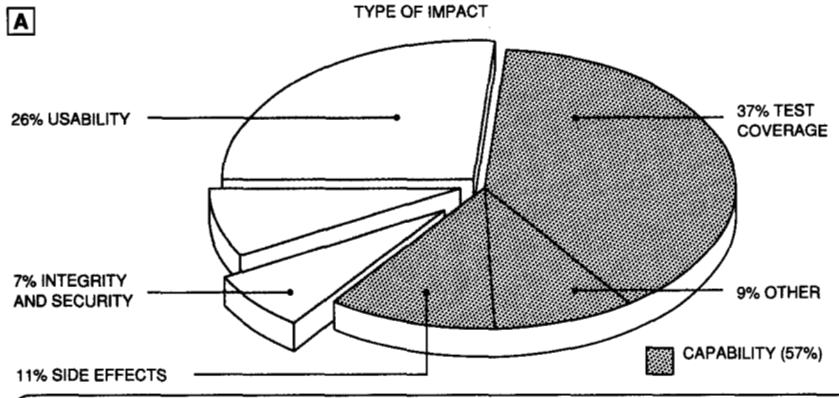
The total of the net benefits for the project was 200 person days. The additional cost of restarting the function test after the corrective actions were implemented was estimated by the lead tester to be 22 person days; hence, the net benefit for the project was 178 person days.

Comparison with other feedback techniques

The examples described thus far demonstrate the value of using our process improvement methodology. Next, we study these examples to establish if other methods could have been used successfully to make similar corrections. This allows the

Figure 11 Impact and trigger for Project C

A



B

IMPACT	TRIGGER	IMPACT (%)	TRIGGER (%)	IMPACT AND TRIGGER (%)	EXPECTED (%)	DIFFERENCE (%)
1. CAPABILITY	TEST COVERAGE	57	52	37	30	7
2. CAPABILITY	SIDE EFFECTS	57	26	11	15	−4
3. INTEGRITY/SECURITY	SIDE EFFECTS	7	26	6	2	4
4. CAPABILITY	SIMPLE PATH COVERAGE	57	15	6	9	−3

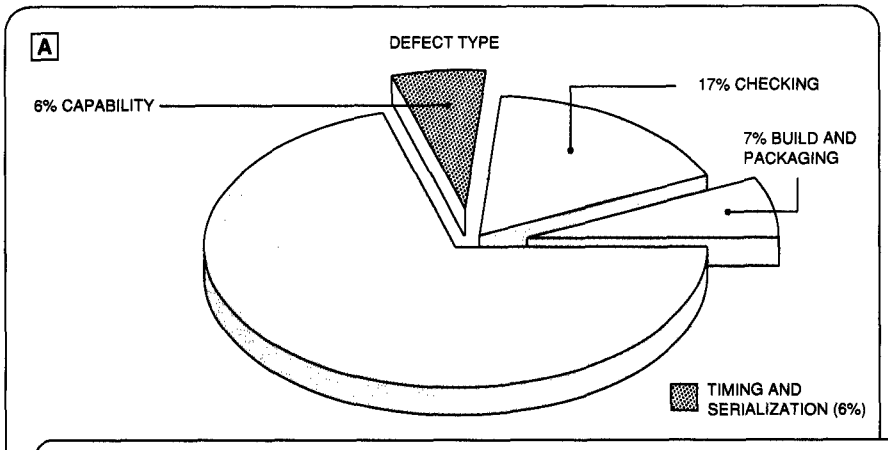
drawbacks associated with *not* using the approach to be understood.

Two other approaches for providing in-process feedback are considered: causal analysis and goal-oriented approaches. *Causal analysis* entails the detailed study of written descriptions of defects to correct the process, while the *goal-oriented* approaches utilize in-process metrics that measure whether specific process objectives (or goals) are being achieved. First, let us review the advantages of process feedback based on attribute focusing, which are known already. References 2 and 10 show that the use of attribute focusing results in several advantages over the other techniques:

- Efficiency. Causal analysis of a large number of defects is a manual and time-consuming process. Hence, if a large number of defects are detected, only a subset of defects will be analyzed. In contrast, the overhead of using attribute focusing is two hours per set of defects, independent of the number of defects. Hence, the entire population of defects can be analyzed.
- Applicability. Goal-oriented approaches often rely on the availability of historical measures to define the goal. For instance, the goal may be that the number of defects per lines of code found in a component lie within a pre-specified range that is set based on historical data. In contrast, attribute focusing uses measures of interestingness that need not rely on historical expectations. Since in practice such expectations are seldom available, this is a major advantage.

In this paper the above advantages are not reemphasized. Instead a deeper understanding of the differences between the use of attribute focusing and the use of other techniques is gained. Note that the advantages above suggest that feedback based on attribute focusing is especially useful when the number of defects is large and historical expectations are not available, since the other techniques may not apply under these circumstances. Here, to get a deeper understanding of the difference, the circumstances under which causal analysis and goal-oriented approaches *do* apply are considered. Hence, it is assumed that all defects undergo causal analysis and reliable

Figure 12 Impact of a defect type for Project C



B

DEFECT TYPE	IMPACT	TYPE (%)	IMPACT (%)	TYPE AND IMPACT (%)	EXPECTED (%)	DIFFERENCE (%)
1. CHECKING	USABILITY	17	28	9	5	5
2. BUILD AND PACKAGING	INSTALLABILITY	7	4	4	0	3
3. TIMING/SERIALIZATION	CAPABILITY	6	57	6	3	2
4. CHECKING	CAPABILITY	17	57	7	10	−2
5. BUILD AND PACKAGING	USABILITY	7	28	0	2	−2

historical expectations are available, and the examples of process correction described in preceding sections are studied to establish if the other methods could have been used to make similar corrections. It is next shown that feedback based on attribute focusing complements the other techniques since it identifies and corrects problems that cannot be addressed by the other methods.

Causal analysis. During causal analysis[17,18] a select set of defects is studied one at a time. While the causal analysts will undoubtedly notice obvious trends in the defects, it is hard for people to spot the more complicated patterns with this manual process. For instance, consider the problem section, "Not distinguish types of execution." Three trends must be noticed before the problem can be identified. From the written descriptions of the defects, the analysts must note that:

1. Capability defects tend to require in-place correction.
2. Capability defects tend to be found by straightforward function test cases.
3. Timing and serialization problems tend to impact capability.

It is quite difficult for a causal analyst to notice even one such trend. Consider what must be done to notice Trend 1 in the list above. First, the description of a defect must be read and understood in terms of capability and fixes that are corrected in place. This is difficult since it requires that, of all the myriad ways that could have been chosen to understand the defect, it is understood particularly in these terms. As is clear from the many experiences presented in preceding sections, there are many different trends that can prove useful in detecting a process problem, and a causal analyst would have to check every defect for all these trends. Second, the first step must be repeated for the next such defect while remembering the prior pattern. This second step must be repeated for a set of defects before concluding there is a trend. Such a process is difficult for

people to apply. Therefore, it seems less likely that causal analysis will detect complicated trends such as those in the referenced prior section. In contrast, attribute focusing finds the trends automatically.

To be more specific, the above example does not indicate that causal analysis cannot easily find the process problem. The message is that the problem cannot be found easily in causal analysis by replicating the identification of trends found by attribute focusing. If the written descriptions of the defects could have suggested the problem without the need to notice trends, then there would be no difficulty in finding the problem in causal analysis. However, there is another class of process problems that is clearly hard to detect in causal analysis simply because there are no defects to analyze. Consider the experience in Project B in the problem section "Experience of reviewers not considered." A problem was identified because very few defects had *trigger* classified as *lateral compatibility*, indicating that few defects were found by thinking about compatibility. To understand why causal analysis may not find this process problem, let us consider a situation where no defects at all are classified *lateral compatibility*. Since there are no defects to consider, there are no written descriptions to study and, hence, the problem cannot be identified. The difficulty in finding such problems using causal analysis occurs because the method makes an implicit assumption that none or few defects reflect goodness, while many defects indicate a problem. However, as shown by the experience, none or few defects can also indicate a problem, namely, insufficient execution of a required activity. It follows that causal analysis may not find such problems. In contrast, an AF chart will highlight the case if very few or no defects occur in a category.

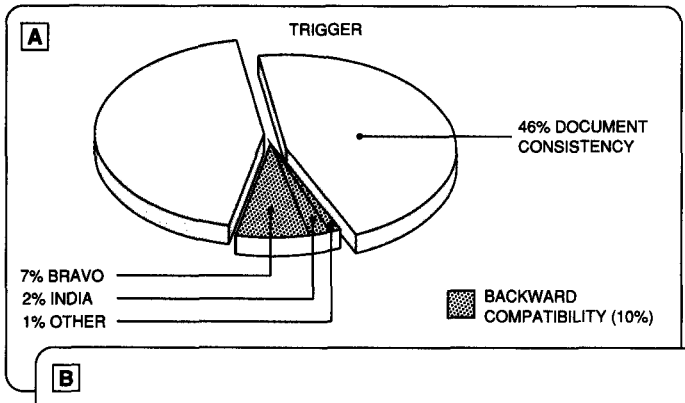
Feedback based on attribute focusing complements other techniques.

Goal-oriented approaches. Many different techniques have been proposed to manage software production. These include statistical defect modeling approaches that predict the reliability of a software product (for example, see References 19–21), and feedback techniques that compare in-process measurements with historical expectations to determine if the project is on the right track (for example, see References 22–24). While these techniques accomplish different useful purposes, they share a common philosophy—they are goal-oriented, i.e., they utilize specific in-process metrics to measure if specific process objectives (or goals) are being achieved. For example, Goal A is achieved if Metric X lies in the range Z, else it is not achieved. And therein lies their major difference with feedback based on attribute focusing which, while it uses in-process metrics, does not utilize them to measure whether predefined goals are being met. If a goal-oriented approach is being used, it must be precisely known which subset of a set of attributes is going to be used and exactly how it is going to be used to provide feedback. When the approach is applied to different projects, the same subset of attributes is used in similar fashion to determine if the relevant goals have been achieved.

In contrast, when using AF, a subset of the data attributes is automatically selected to generate the 20 AF charts that are used to provide feedback. So, even though two different projects may collect data using the same set of attributes, two different projects may receive feedback from AF charts based on different subsets of those attributes. That data-driven (as opposed to goal-driven) nature allows AF to complement goal-oriented approaches for reasons given next.

Process feedback from defect data begins with the observation of some pattern in attribute-valued data. In principle, if one could look for all possible patterns that can occur, one could provide perfect feedback. However, the combinatorial complexity of the space is much too large for such observation to be feasible. For example, in a data set with 10 attributes, each of which has 10 values, there are 45 distinct AF charts to show the cross product of pairs of attributes, and 100 possible trends that may be observed in a chart. Therefore, there are 4500 possible patterns that may be observed by simply considering pairs of attributes. Not surprisingly, goal-oriented approaches focus only on some predefined patterns

Figure 13 Trigger and component for Project A



B

TRIGGER	COMPONENT	TRIGGER (%)	COMPONENT (%)	TRIGGER AND COMPONENT (%)	EXPECTED (%)	DIFFERENCE (%)
1. DOCUMENT CONSISTENCY	TANGO	46	32	24	15	9
2. OPERATIONAL SEMANTICS	INDIA	15	39	14	6	8
3. BACKWARD COMPATIBILITY	BRAVO	10	14	7	1	5
4. OPERATIONAL SEMANTICS	TANGO	15	32	0	5	−5
5. DOCUMENT CONSISTENCY	BRAVO	46	14	3	6	−3
6. DOCUMENT CONSISTENCY	INDIA	46	39	15	18	−3
7. BACKWARD COMPATIBILITY	INDIA	10	39	2	4	−2
8. DESIGN CONFORMANCE	TANGO	7	32	0	2	−2
9. OPERATIONAL SEMANTICS	BRAVO	15	14	0	2	−2

that are known to be relevant. Hence, a set of goal-oriented approaches used to provide feedback to a project define the set of known patterns in the data to look for and act upon. On the other hand, attribute focusing is a mechanism to learn about and act upon unknown yet possibly important patterns. In this regard, the two approaches complement each other. The subtle nature of some of the process problems that have been described, e.g., in the sections describing Project C, support this viewpoint. After determining the relevance of such trends, they can form the basis for a goal-oriented feedback technique that can be used for future projects. However, such trends may not have been discovered by using a goal-oriented technique.

In summary, we argue that the causal analysis and the goal-oriented approaches may not address certain kinds of process problems that can be corrected by using attribute focusing. Hence, it is not surprising that many of the experiences of process correction in this paper were detected by attribute focusing in projects where causal analysis and goal-oriented approaches were also used for process feedback. That result is discussed in a later section on "Summarizing the experience." The reader who is interested in the principles underlying our methodology may skip ahead to that section. The intervening material consists of additional experiences and does not introduce new principles. It has mainly educational and evidential value.

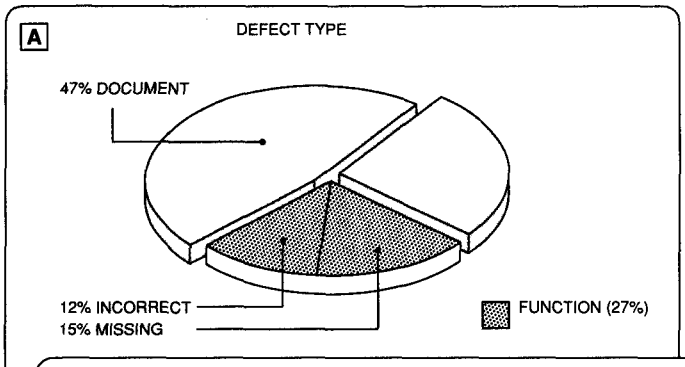
More experiences from Project A

In addition to the problems previously described for Project A, two other problems are discussed next.

Lack of formal requirements process. The AF chart in Figure 13 shows the cross product of the attributes *trigger*, which captures what the inspector was looking for when the defect was found, and *component*, which specifies the software component of the product in which the defect was found. The project team explained most of the associations in the table without discovering a

Figure 14 Defect type and missing or incorrect error for Project A

A



B

DEFECT TYPE	MISSING OR INCORRECT ERROR	TYPE (%)	ERROR (%)	TYPE AND ERROR (%)	EXPECTED (%)	DIFFERENCE (%)
1. INTERFACE	INCORRECT	15	56	15	9	7
2. INTERFACE	MISSING	15	44	0	7	−7
3. FUNCTION	MISSING	27	44	15	12	3
4. FUNCTION	INCORRECT	27	56	12	15	−3
5. DOCUMENT	INCORRECT	47	56	24	27	−3
6. DOCUMENT	MISSING	47	44	24	21	3

process problem. For example, the disassociation between component Bravo and *operational semantics* in table Item 9 was attributed to the behavior of component Bravo, which was meant to merely copy blocks of storage and therefore did not involve complex logic. Hence, most defects in component Bravo would not be found by thinking about the flow of logic in the operation of component Bravo, since that operation was very simple. However, the disassociation between component India and *backward compatibility* (table Item 7) led to the discovery of a process problem.

Cause. Component India must address compatibility issues to ensure that customer applications built upon previous releases of the product run correctly on this release. However, those issues were not addressed in component India. Hence, few defects that pertained to compatibility were found. Corroboration of the process problem indicated (1) consensus among designers, (2) a compatibility requirement for the product, and (3) the lack of compatibility features in the design of component India to date.

Implication. Existing customer applications may fail when the new release is installed.

Corrective action. Component India was designed for compatibility before proceeding to the next phase, and the redesigned component India was inspected for compatibility.

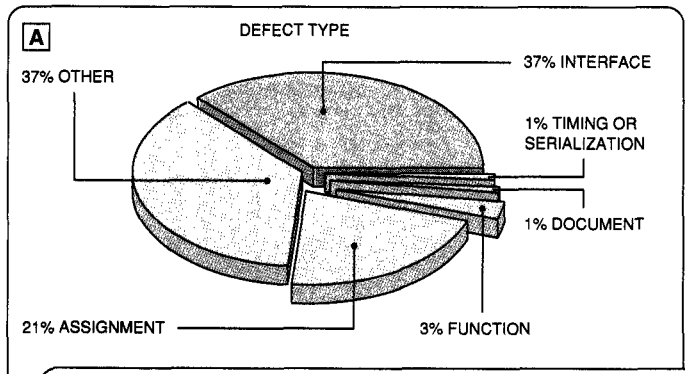
Validation of correction. Similar to the corroboration of the same process problem.

Benefit. The corrective action removes defects and is an adjustment that prevents shipping a product that is incompatible with previous releases. It also identifies a weakness in the process used to translate requirements to design. The project team attributed this weakness to the lack of a formal requirements process already noted for Project A and, perhaps, a lack of experience with the product on the part of some team members.

Component-level design is incomplete. The AF chart in Figure 14 shows the cross product of the attributes *defect type* and *missing or incorrect*. Table Item 3 shows that *function* defects are associated with *missing*.

Cause. This could not be determined by the team at the feedback session. As per the model of in-

Figure 15 Defect type during system test for Project D



B

DEFECT TYPE	OBSERVED (%)	EXPECTED (%)	DIFFERENCE (%)
1. INTERFACE	37	13	24
2. TIMING OR SERIALIZATION	1	13	-11
3. DOCUMENT	1	13	-11
4. FUNCTION	3	13	- 9
5. ASSIGNMENT	21	13	9

terpretation,[9] the team had to investigate that association at a later time. The results of the investigation are given below in "Investigate."

Implication. While no cause was found, the trend had a clear implication: The component-level design may be incomplete. The project should not exit the CLD phase before ensuring that all major functionality is covered.

Investigate. The project team read the detailed descriptions of the defects that had been classified *function* and *missing* to try and understand the cause of the association. They found those defects pertained to a specific functionality of the product, namely, its ability to recover from an erroneous state. This implied that the design of that function may have been incomplete.

Corrective action. Reinspect component-level design of the recovery function.

Validation of corrective action. Twelve defects were found.

Qualitative benefit. The corrective action removes defects, and is an adjustment to make sure that the component-level design of the recovery function is complete before proceeding to the module-level design stage.

Quantitative benefit. The cost of corrective action was 15 person days. The total benefit was that, based on historical data, most recovery defects that escape design are eventually found in the field; hence, assuming 10 of the 12 defects would have been found in the field, the total benefit is 150 person days. The net benefit was 135 person days.

Experiences from Projects D and E

Problems with the design, test, and inspection processes are uncovered in the examples from Projects D and E.

Flawed design and test processes. The AF chart in Figure 15 shows the distribution for the attribute *defect type*, which captures information about the nature of the error. A process problem was uncovered when the project team noticed the large magnitude of defects found at system test (ST) that were fixed by correcting interfaces, and the small magnitude of defects found at system test

that were fixed by correcting timing and serialization problems.

Cause. Function test did not adequately remove module or component interface defects in many components. Those defects escaped to system test, and were overwhelming system testers. Also, many interface defects were injected during design and escaped to later stages.

Corroboration of the process problem indicated that (1) each component team executed function test for its own components, hence function test could not adequately focus on interfaces between component and user interfaces, (2) a low-level design stage was not part of the formal process, and (3) component-level design documents contained little or no detail about component interfaces.

Implication. Interface defects, timing and serialization, and other defects may remain in the product after shipment since the time scheduled for system test is spent removing interface problems instead of executing true system test testcases.

Corrective action. System test was halted and function test was re-entered. Special emphasis was placed on function test of the components most associated with interface problems.

Validation of correction. An additional 424 interface defects were removed from the product.

Qualitative benefit. The corrective action removes defects and is an adjustment that prevents shipping a product which is poorly integrated. It also identifies a weakness in the design process and the test process. The process used for the current release did not have a formal low-level design stage, and the function test strategy focused only on component function, and not on the relationship between components, which was to be tested during system test. Evidently, that relationship is too complex for such a process. Future releases of the product should use a formal low-level design stage that specifies and inspects interfaces in detail and tests an integrated product during function test.

Quantitative benefit. The cost of corrective action was 424 person days at a cost of one person day per defect. The total benefit was 1272 person days, assuming that all 424 defects were found at system test at a cost of three person days per defect, and the net benefit was 848 person days.

It should be noted, however, that the team was aware of a project quality problem prior to using the methodology, but they had not agreed on the cause of the problem or the corrective action. Thus it is not clear that the methodology was solely responsible for all 424 discovered defects.

Inspections did not detect interface errors. The AF chart in Figure 16 shows an association between defect type *interface* and the development phase *unit test*, indicating that more such defects were found in unit test than other phases. In fact, there is a continued increase in defect type *interface* as the development progresses from the module-level design through code inspection to unit test. This led to the discovery of the following process problem.

Cause. The module-level design and code inspections were not effective at detecting *interface* defects.

Corroboration of the process problem indicated that (1) the teams had inspected pseudocode instead of inspecting the code itself. Thus, many defects detected in code inspections are problems with the quality of pseudocode being reported via defect type *document* (see table Item 3 in Figure 16), and (2) some teams had treated the pseudocode inspections as a combined module-level design and code inspection, omitting inspection of the actual code and low-level design.

Implication. Function test will have difficulty completing its scheduled test plan because of *interface* defects, which should have been removed earlier.

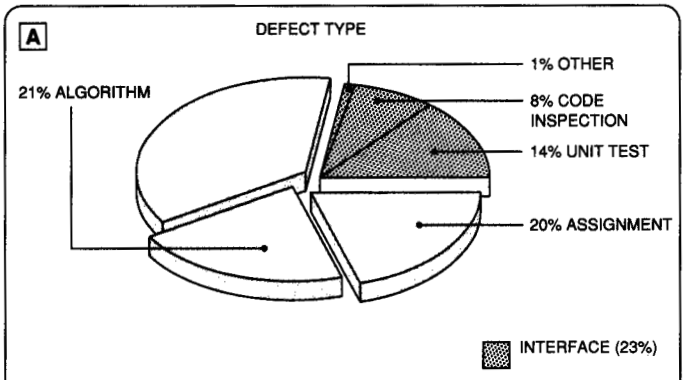
Corrective action. For subsequently scheduled components, the inspections for module-level design and code were decoupled, and the focus during the code inspection was completed code instead of pseudocode.

Validation of correction. Defects of type *interface* were expected to peak at code inspection and to diminish at unit test.

Benefit. The corrective action should remove defects. It is an adjustment that prevents an adverse impact to the function test schedule caused by the

Figure 16 Defect type and phase found for Project E

A



B

DEFECT TYPE	PHASE FOUND	TYPE (%)	PHASE (%)	TYPE AND PHASE (%)	EXPECTED (%)	DIFFERENCE (%)
1. DOCUMENT	UNIT TEST	10	42	0	4	−4
2. INTERFACE	UNIT TEST	23	42	14	10	4
3. DOCUMENT	CODE INSPECTION	10	44	8	4	4
4. ASSIGNMENT	UNIT TEST	20	42	11	9	3
5. INTERFACE	CODE INSPECTION	23	44	8	10	−2
6. ALGORITHM	CODE INSPECTION	21	44	11	9	2
7. FUNCTION	UNIT TEST	12	42	3	5	−2

escape of numerous *interface* defects into function test. It also identifies and corrects a problem with the inspection process, namely, the combining of module-level design and code inspections into a pseudocode inspection.

Experiences from Projects F and G

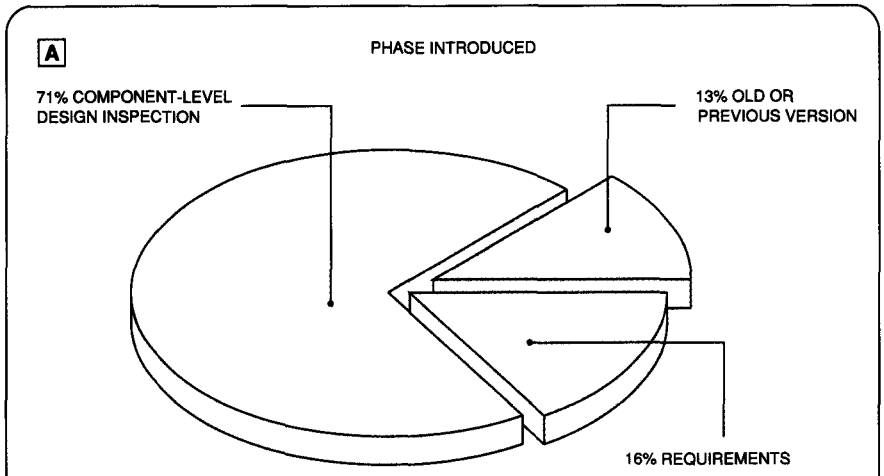
Problems with informational messages and the lack of complete requirements are discussed for Project F, and untested overlay structures are noted in Project G.

Existing compiler messages had errors. This project team was developing a compiler. The magnitude of defects in the AF chart in Figure 17 that were introduced in the *old* version (i.e., that are defects in a previous version of the compiler) led to the identification of a process problem.

Cause. Even though the relative magnitude of old defects was small, the developers were concerned that even a small number of latent defects would disrupt the current development schedule, which had not planned for such defects, so they decided to analyze them further. It was discovered that a majority of the defects were associated with the problem that developers were unfamiliar with compiler message development guidelines and review process. Hence, many such defects were injected into the compiler messages.

Corroboration of the process problem indicated that (1) apart from the defects in existing compiler messages, many defects were injected into compiler messages being developed for the current release, which supported the hypothesis that the process of message development was suspect (for example, 54 percent of the defects found in one of the components being currently developed were defects in compiler messages, while in another component the proportion was 60 percent), (2) most of the defects in the compiler messages were related to incorrect terminologies used in the messages and missing description of the error recovery actions of the compiler, (3) correct terminol-

Figure 17 Phase where error was introduced for Project F



B

PHASE INTRODUCED	OBSERVED (%)	EXPECTED (%)	DIFFERENCE (%)
1. COMPONENT-LEVEL DESIGN INSPECTION	71	33	38
2. OLD VERSION	13	33	−21
3. REQUIREMENTS	16	33	−17

ogies and necessary parts of compiler messages were specified in the compiler messages development guidelines, and (4) there was no consensus among developers when compiler messages should be reviewed during the development cycle.

Implication. The compiler messages may contain incorrect terminologies that customers do not understand, which affects the usability of the compiler. The structure of the compiler messages may be inconsistent, which affects the perceived quality of the compiler. Translators may find it difficult to translate the messages into a different national language, which will increase the translation cost of the compiler. And finally, information developers will need to spend more time reviewing the compiler messages.

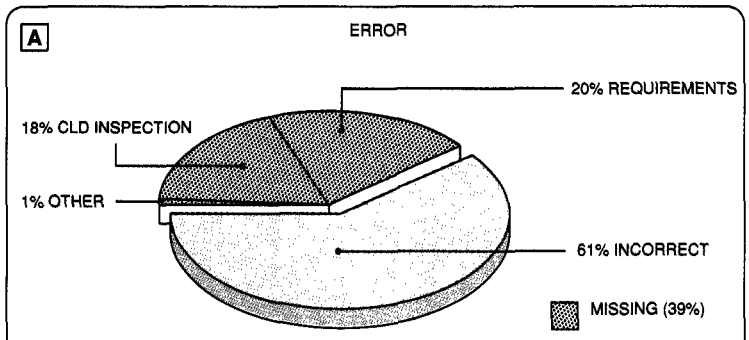
Corrective action. A compiler messages kickoff meeting was held, during which information developers explained the compiler messages development guidelines and review process to the entire development team.

Validation of correction. Information developers found fewer defects when inspecting compiler messages.

Benefit. The corrective action will prevent such defects from being injected and will improve customer usability and reduce language translation cost. It is a process adjustment as well as a process correction. It should improve the quality of the current product as well as improve the process of message development for subsequent releases.

Incomplete requirements. Refer to the AF chart in Figure 17, which shows data collected during early component-level design inspections. The magnitude of defects that were introduced during the requirements stage, and had therefore es-

Figure 18 Missing or incorrect error and where introduced for Project F



B

MISSING OR INCORRECT ERROR	PHASE INTRODUCED	ERROR (%)	PHASE (%)	ERROR AND PHASE (%)	EXPECTED (%)	DIFFERENCE (%)
1. INCORRECT	COMPONENT-LEVEL DESIGN INSPECTION	61	70	52	43	10
2. MISSING	COMPONENT-LEVEL DESIGN INSPECTION	39	70	18	28	−10
3. INCORRECT	REQUIREMENTS	61	27	8	17	− 9
4. MISSING	REQUIREMENTS	39	27	20	11	9

caped the requirements inspection, led to the identification of a process problem.

Cause. Even though the relative magnitude of requirements escapes was small, the team was concerned that the requirements developers and inspectors did not have the appropriate level of experience.

Corroboration of the process problem indicated that (1) the developers created the component-level designs based on the requirements without even recognizing defects in the requirements, and (2) consensus among developers who inspected the requirements was that they had difficulties in finding defects because the authors of the requirements were the more experienced members of the team, and there is a tendency to be less critical of the work of experienced colleagues.

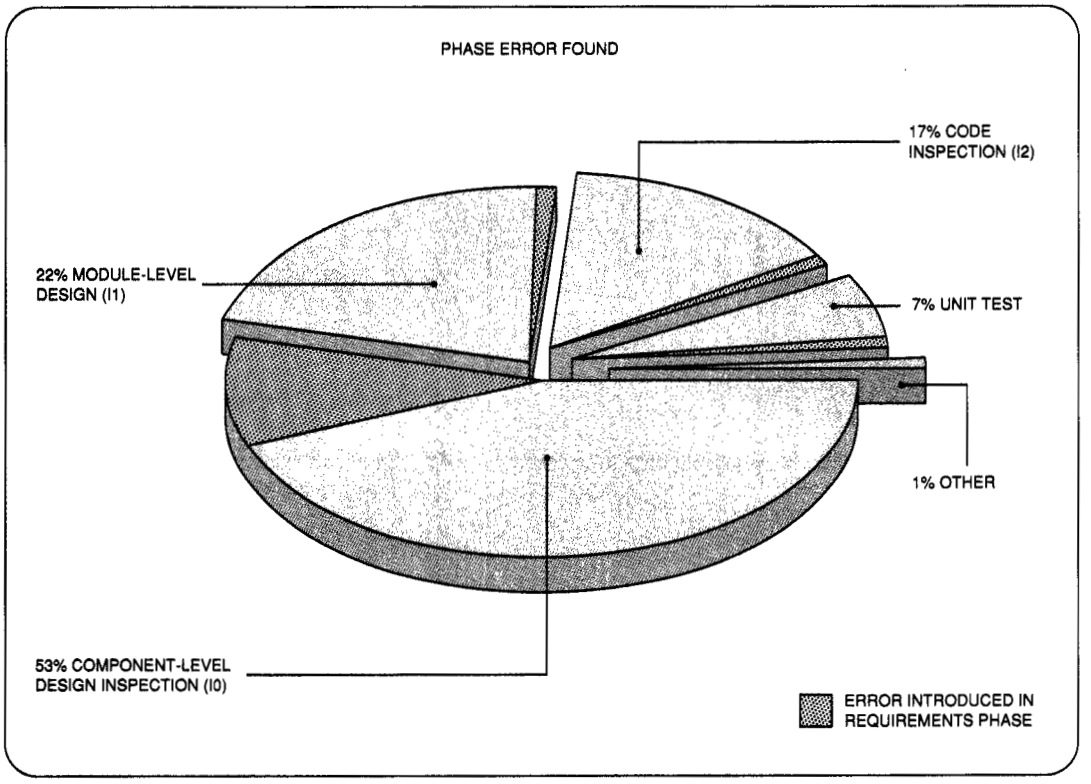
The first fact is also captured by the AF chart in Figure 18, which was generated at the midpoint of the component-level design inspection (I/O). Defects introduced in the requirements stage are associated with *missing* in table Item 4. Such defects are more likely to be fixed by introducing new material in the requirements document than correcting in place. One would think that the designers would have recognized that the requirements document was incomplete. Such was not the case. Defects in the requirements were not found until the component-level design inspections, which resulted in major rework of the designs.

Implication. The compiler will not provide the functionalities expected by the customers.

Corrective action. Developers worked with the authors of the requirements document to ensure the requirements were complete and correct, then reiterated component-level design for the affected areas. The work to ensure the requirements were complete and correct included reinspecting the appropriate sections of the requirements document, and holding meetings during which developers explained their interpretation of the requirements from a customer's perspective.

Validation of correction. Reduction in the number of escapes from requirements inspection to later phases was expected. Refer to the AF chart in Figure 19, showing that escapes from requirements were indeed few.

Figure 19 Phase error was introduced and found for Project F



Benefit. The second corrective action removes defects, whereas the first corrective action should prevent defects from being injected, since the less-experienced members will have a better understanding of the requirements. It is a process adjustment that reduces the risk of shipping a compiler with incomplete functionality. However, that risk has not been eliminated since Figure 19 shows that, while escapes from requirements have declined, they did occur all the way to unit test. A better adjustment would have been to add experienced staff to the team. This discussion also identifies a weakness in the staffing process for the product, namely, that it does not weight the experience of personnel adequately when deciding the composition of the team.

Untested overlay structures. The relatively large magnitude of defects classified *side effects* in unit test data (see AF chart in Figure 20) led to the identification of the following process problem.

Cause. The previous version of the software contained overlay structures within work areas, which the unit testers did not understand well.

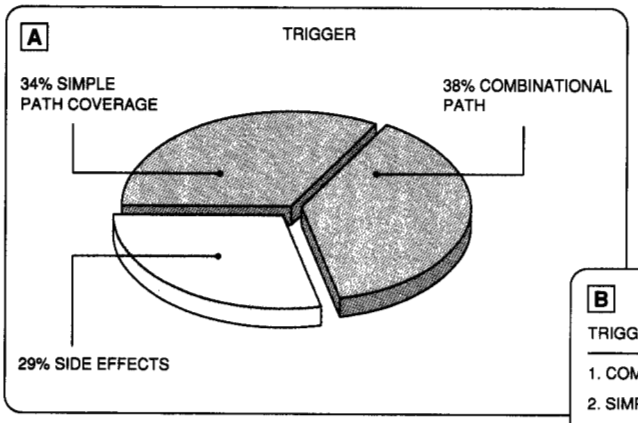
Corroboration. Several problems escaped unit testing due to the affected area not being examined. The overlay was detected with regression test cases for component areas that were not modified.

Implication. Latent problems may still exist.

Corrective action. Since the old code was difficult to understand and test, future projects should redesign, recode, and retest the overlay structures.

Benefit. Exposes a potential problem for the current project and identifies a process correction for future projects.

Figure 20 Trigger in unit test for Project G



B

TRIGGER	OBSERVED (%)	EXPECTED (%)	DIFFERENCE (%)
1. COMBINATIONAL PATH	38	14	23
2. SIMPLE PATH COVERAGE	34	14	20
3. SIDE EFFECTS	29	14	14
4. TEST VARIATION	0	14	−14
5. TEST COVERAGE	0	14	−14
6. TEST INTERACTION	0	14	−14
7. TEST SEQUENCING	0	14	−14

Table 3 Project interactions

Process Correction Methodology Started	Project	Number of Feedback Sessions	Causal Analysis	Number of Preventions	Number of Removals
Component-level design	A	2	yes	4	5
	B	1	yes	2	2
	E	2	yes	1	2
	F	3	yes	4	5
Unit test	G	2	yes	0	3
Function test	C	1	no	0	4
System test	D	5	yes	0	14

Summarizing the experience

Table 3 uses the following dimensions to summarize the cost and benefit of using attribute focusing for the different projects in Table 1.

- Phase started, or the phase the project was in when the team started using the process correction methodology
- Number of feedback sessions held, where each feedback session lasts about two hours
- Formal causal analysis used or not used (yes or no)
- Number of preventions, or the number of issues that led to defect prevention measures for the current release
- Number of removals, or the number of issues that led to defect removal measures for the current release

What the table indicates is summarized in the following four points.

1. The methodology identifies and corrects process problems in different kinds of projects. This is the main result discussed in this paper and is evident from Table 1 and the two rightmost columns of Table 3.
2. With regard to the current release, projects that start earlier benefit by the prevention and removal of defects, while projects that start late benefit only by the removal of defects (see Table 3). All projects benefit by the prevention and removal of defects in future releases as evidenced by the experiences in this paper (read the sections with the heading "Benefit" for each experience).
3. The use of other techniques does not appear to reduce the effectiveness of the methodology. See Table 1 (the tracking tools used are suggestive of the kinds of goal-oriented analyses used by the teams) and Table 3 (see the column "Causal Analysis"). This supports the argument in the section of this paper called "Goal-oriented approaches" that our methodology complements other defect-based approaches for in-process correction.
4. Let us understand what the above results tell us about our methodology in the general context of project management and control. Clearly, a goal of the management of the different projects described in Table 1 was to identify and correct process problems such as those presented in this paper. But those problems remained unidentified and uncorrected until the attribute focusing feedback sessions. All projects described in Table 1 were involved in the actual production of major software products. We can assume that the project teams and their management did the best they could do, and therefore conclude that their effort is representative of current practices in project management and control. That conclusion is also justified by the fact that the projects in Table 1 were drawn from six sites separated by considerable geographical distances and engaged in very different lines of business. Therefore, one may compare our methodology against the effort of the management of the different projects to conclude that it can complement current practice in project management and control.

Model of interpretation and correction

Finally, the lessons of experience are incorporated into the methodology to build a complete model of correction. AF, as described in Reference 9, is a general approach to knowledge discovery that can be applied to domains other than software engineering. As such, it does not specify the exact process one uses in the application domain to discover knowledge, or to implement an action in the physical domain once knowledge is discovered. It merely states that the analyst relates the items in the legends of the selected charts to the domain—which leads to the discovery of knowledge and the implementation of actions based on this knowledge. Based on the experience of using AF as a feedback mechanism, the following model of correction has evolved for defect analysis. The steps of the model correspond to the dimensions used to present the experiences in preceding sections, and were illustrated in Figure 4.

1. Identify a problem.
   - Discuss a trend—the presence or absence of certain items in the table of one or more charts.
     - Discuss the possible cause of the trend
     - Corroborate the cause with the team
     - Infer the result if a corroborated cause is not acted upon
   - Possible outcomes
     - Problem identified—continue
     - Determined cause is not a problem—stop
     - The cause could not be determined—sample defects that correspond to the trend and study written descriptions to determine cause
2. Act to correct the problem.
3. Validate a corrective action.
   - Did the corrective action have the desired effect?
     - Plan—what is the anticipated change in analysis of classified defect data? Other effects?
     - Observe the anticipated change
   - Outcome
     - Change observed—continue
     - Change not observed—go back to identify
4. Assess the corrective action in terms of:
   - Cost—labor, dollars
   - Benefit—additional defects discovered or prevented, cycle reduction
   - Nature—does it truly solve the problem or is it merely expedient?
   - Exposure—is the process correction expedient?

5. Report process corrections and exposures to appropriate parties:
   - Other teams on project
   - Lab-wide process team
   - Management

Step 1 is undertaken during the attribute focusing feedback session. The people who will carry out corrective actions or investigate problems are also identified during this session. The other steps are carried out after the session, and are usually coordinated by a member of the project team.

The use of the model is quite intuitive as one step leads naturally to the next. It is possible to get a feeling for this by reading the experiences in the section "Experiences from Projects B and C" in conjunction with the model description.

Conclusion

Experiences with a software process correction methodology that uses machine-assisted data exploration of classified defect data have been presented. The experiences were analyzed to understand the scope and value of the methodology. It was shown that the methodology has been used successfully by very different projects, thereby suggesting that it does not impose restrictions on the kinds of projects that may use the approach. The projects used the methodology to remove latent defects, to prevent the injection of defects, and to identify near-term process adjustments and long-term process corrections. It was shown that those benefits translate to substantial labor savings and quality improvement. Finally, the methodology was shown to address process problems that are not addressed by current practices. On the basis of the above evidence, we conclude that the software process correction methodology is an important advance that can have a major impact on software development in the near future.

Acknowledgments

We would like to acknowledge the key role played by the following individuals: Chris Byrne, Paul Merrill, Steve Mink, Nick Nomm, and Ron Peterson of the IBM Santa Teresa laboratory; Jean Beardsley, Nelson Fincher, Manny Valvidia, Fanny Chen, Mark McClintok, and Jean Chang of the IBM San Jose laboratory; David Brown, Chris Vignola, Harris Morgenstern, Dean Butler, Steve Horowitz, Mark Koury, Dawn Alpert, Yiwen Tan, John Kong, Ron Dyer, and Carolyn Coppola of the IBM Mid-Hudson Valley Programming Laboratory; Barry Goldberg, Rick Matela, and Mike DeCarlo of the IBM Danbury Programming Laboratory; Dennis Fick, Michael Fraenkel, and Larry Isely of the IBM Networking Systems Laboratory; and C. W. Cheung, Sieghard Flaser, Allan Jenoff, Dave Knott, and Priti Shah of the IBM Canada Development Laboratory.

Cited references

1. W. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Co., Reading, MA (1989).
2. I. S. Bhandari, M. J. Halliday, E. Tarver, D. Brown, J. K. Chaar, and R. Chillarege, *A Case Study of Process Evolution During Development*, Research Report RC 18592, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (December 1992). Also in *IEEE Transactions on Software Engineering* (December 1993).
3. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1981), pp. 39–41.
4. R. B. Grady and D. L. Caswell, *Software Metrics: Establishing a Company-wide Program*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1987), pp. 82–90.
5. P1044/D5—A Standard Classification of Software Anomalies, Technical Committee on Software Engineering of the IEEE Computer Society, Sponsor, draft (1993). IEEE Computer Society Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.
6. J. Chaar, M. Halliday, I. Bhandari, and R. Chillarege, "In-Process Metrics for Software Inspection and Test Evaluations," *IEEE Transactions on Software Engineering* (1993). Also available as Research Report RC 19063, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (July 1993).
7. R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. Moebus, B. Ray, and M-Y. Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurement," *IEEE Transactions on Software Engineering* **18**, No. 11, 943–956 (November 1992).
8. R. Chillarege, W. L. Kao, and R. G. Condit, "Defect Type and Its Impact on the Growth Curve," *Proceedings of the 13th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA 90720-1264 (1991), pp. 246–255.
9. I. Bhandari, "Attribute Focusing: Machine-Assisted Knowledge Discovery Applied to Software Production Process Control," *Proceedings of the Workshop on Knowledge Discovery in Databases*, G. Piatetsky-Shapiro, Editor, AAAI Press (July 1993). Also abbreviated from Research Report RC 18320, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (September 1992).
10. I. S. Bhandari and N. Roth, *Post-Process Feedback with and without Attribute Focusing: A Comparative Evaluation*, Research Report RC 18321, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (September 1992).
11. All articles numbered I–XXX in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. Frawley, Editors, AAAI Press/The MIT Press, Menlo Park, CA (1991).

12. A. Ram, "Knowledge Goals: A Theory of Interestingness," *Proceedings 12th Annual Conference of the Cognitive Science Society*, LEA (Lawrence Erlbaum Association), Hillsdale, NJ (August 1990), 206–214.
13. J. Mingers, "An Empirical Comparison of Selection Measures for Decision-Tree Induction," *Machine Learning* **3**, No. 3, 319–342 (1989).
14. W. Frawley, G. Piatetsky-Shapiro, and C. Matheus, "Knowledge Discovery in Databases: An Overview," *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. Frawley, Editors, AAAI Press/The MIT Press, Menlo Park, CA (1991).
15. G. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review* **63**, No. 2, 81–97 (March 1956).
16. M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15**, No. 3, 182–211 (1976).
17. C. Jones, "A Process-Integrated Approach to Defect Prevention," *IBM Systems Journal* **24**, No. 2, 150–167 (1986).
18. R. Mays, C. Jones, G. Holloway, and D. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal* **29**, No. 1, 4–32 (1990).
19. A. L. Goel, "Software Reliability Models: Assumptions, Limitations," *IEEE Transactions on Software Engineering* **11**, No. 12, 1411–1423 (1985).
20. J. Musa, A. Iannino, and K. Okumoto, *Software Reliability—Measurement, Prediction, Application*, McGraw-Hill Book Co., Inc., NY (1987).
21. B. Ray, I. S. Bhandari, and R. Chillarege, "Reliability Growth for Typed Defects," *Proceedings IEEE Reliability and Maintainability Symposium*, IEEE Computer Society, NY (1991), 327–336.
22. V. R. Basili and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety* **32**, 171–191 (1991).
23. W. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Co., Reading, MA (1989), pp. 301–334.
24. R. W. Selby and A. A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis," *IEEE Transactions on Software Engineering* **14**, No. 12, 1743–1757 (December 1988).

Accepted for publication August 20, 1993.

Inderpal Bhandari *IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: isb@watson.ibm.com).* Dr. Bhandari is a member of the research staff at the IBM T. J. Watson Research Center. He was educated at Carnegie Mellon University (Ph.D., 1990), the University of Massachusetts at Amherst (M.S.) and the Birla Institute of Technology and Science, Pilani, India (B. Engg.). He is a member of the IEEE and the IEEE Computer Society. His primary research areas are software engineering (process, metrics, and feedback) and artificial intelligence (knowledge discovery and diagnosis).

Michael J. Halliday *IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: flymike@watson.ibm.com).* Mr. Halliday is a senior programmer at the IBM T. J. Watson Research Center. He joined IBM in 1969 and has worked on the design and development of IBM's mainframe operating systems (MVS, VM, and TPF) for much of that time. He graduated from the University of Nottingham, UK.

Jarir Chaar *IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: jarir@watson.ibm.com).* Dr. Chaar is a research staff member at the IBM T. J. Watson Research Center. He received both his Ph.D. in CSE (1990) and his M.S. in ECE (1982) from the University of Michigan in Ann Arbor. He also received his B.E. in E.E. (1981, with distinction) from the American University of Beirut in Lebanon. Prior to joining IBM, he worked as a senior software engineer at Deneb Robotics Inc. He also held the positions of instructor in the electrical engineering department at the American University of Beirut (1983–1985) and manager and systems engineer at Computeknix Microcomputers (1982–1984). He is a member of the IEEE.

Ram Chillarege *IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: ramchill@watson.ibm.com).* Dr. Chillarege received his B.Sc. degree in mathematics and physics from the University of Mysore, and his B.E. (with distinction) in electrical engineering and M.E. (with distinction) in automation from the Indian Institute of Science, Bangalore. He worked as an independent hardware design consultant before returning to school. He received his Ph.D. in electrical and computer engineering from the University of Illinois in 1986. Dr. Chillarege is currently manager of continuous availability and quality at the IBM T. J. Watson Research Center. His work has predominantly been in the area of experimental evaluation of reliability and failure characteristics of machines. His work includes the first measurements of error latency under real work load and the concept of failure acceleration for fault-injection-based measurement of coverage. More recently, his work has focused on error and failure characteristics in software. Dr. Chillarege is a senior member of IEEE and an associate editor of the *IEEE Transactions on Reliability*.

Kevin Jones *IBM Large Scale Computing Division, 522 South Road, Poughkeepsie, New York 12601.* Mr. Jones is a staff programmer at the IBM/TPF (Transaction Processing Facility) Programming Laboratory in Poughkeepsie, New York. He joined IBM in 1988 and has worked on the function component testing of several major enhancements to the TPF operating system, including the TPF 4.1 release. He received a B.S. degree in computer science from the University of Connecticut.

Janette S. Atkinson *IBM Canada Development Laboratory, 1150 Eglinton Avenue East, North York, Ontario, Canada M3C 1H7.* Ms. Atkinson is a senior associate development analyst at the IBM Canada Development Laboratory in North York, Ontario. She joined IBM in 1989 and has worked in the AS/400® languages area. She graduated from the University of Toronto.

Clotilde Lepori-Costello *IBM Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161.* Ms. Lepori-Costello is site benchmarking coordinator at the IBM Santa Teresa laboratory in San Jose, California. She joined IBM in 1989 and is currently the Orthogonal Defect Classification (ODC)

team leader. Prior to joining IBM, Ms. Lepori-Costello worked at Varian Associates as a facilities engineer, and at Lockheed Missiles and Space Company as a manufacturing engineer for the Satellite Systems Division. While at Lockheed, she designed and developed an expert system to assist industrial engineers in the development of work measurement standards for manufacturing processes. Ms. Lepori-Costello holds bachelor's and master's degrees in industrial and systems engineering from San Jose State University. She is also pursuing a master's degree in computer science.

Pamela Y. Jasper *IBM Networking Systems, 200 Silicon Drive, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Ms. Jasper is currently the manager of the Experimental Systems department. She was previously the focal person for the Defect Prevention Process, MDQ, and Orthogonal Defect Classification for the Enterprise Workgroup Networking Software (EWNS) product area. Prior to joining EWNS, she worked on the design and development of IBM mainframes, mainframe applications, and workstation applications. She is a graduate of the University of Arkansas.

Eric D. Tarver *IBM Mid-Hudson Valley Programming Laboratory, P.O. Box 950, Poughkeepsie, New York 12602.* Mr. Tarver is an associate programmer at the IBM Mid-Hudson Programming Laboratory. He joined IBM in 1989 and works as a function component tester in the MVS Operation, Design, Development, and Test department. He has a special interest in the study, development, and implementation of data collection tools and quality metrics. He received a B.S. in computer science from Jackson State University.

Cecilia Carranza Lewis *IBM San Jose, 5600 Cottle Road, San Jose, California 95193.* Ms. Lewis is a staff programmer at the IBM Programming Laboratory in San Jose, California. She joined IBM in 1983 and has worked on the design and development of the Data Facility Storage Management Subsystem (DFSMS™) operating system software in the area of data management. She graduated from Santa Clara University.

Masato Yonezawa *IBM Japan, Ltd., Yamato Laboratory, 1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242, Japan.* Mr. Yonezawa is a project manager of workstation application products development, Asia Pacific Products, at the IBM Yamato Programming Laboratory in Japan. He joined IBM in 1982 and has worked on the software product development of document processing systems (MVS, VSI, VSE, VM) and word processors (DOS, OS/2®). He graduated from the University of Kyoto.

Reprint Order No. G321-5538.