A Case Study of Software Process Improvement During Development

Inderpal Bhandari, Michael Halliday, Eric Tarver, David Brown, Jarir Chaar, and Ram Chillarege

Abstract— **We present a case study of the use of a software process improvement method which is based on the analysis of defect data. The first step of the method is the classification of software defects using attributes which relate defects to specific process activities. Such classification captures the semantics of the defects in a fashion which is useful for process correction. The second step utilizes a machine-assisted approach to data exploration which allows a project team to discover such knowledge from defect data as is useful for process correction. We show that such analysis of defect data can readily lead a project team to improve their process during development.**

Index Terms—**Date exploration, defect-based process improvement, in-process metrics, knowledge discovery.**

I. Introduction

THE software process [1] provides the framework for the development of software systems. Deficiencies in the definition or execution of the activities which comprise the process affect the quality and cycle time of the system under production adversely. Hence, it is important to understand how the errant activities may be corrected *in-process,* i.e., during the course of development.

Several approaches to in-process improvement have been developed over the years. Noteworthy examples include the Software Engineering Institute (SEI) approach [1], the Software Engineering Laboratory (NASA/SEL) approach [2], [3], the process cycle approach [4], etc. These approaches address the entire scope of software development and its improvement. The scope of this paper is narrower, and is restricted to defect-based process improvement.

Defects found during production may be viewed as a manifestation of process deficiencies. Hence, it makes sense that defect data be analyzed to make in-process improvements. This paper present a case study of the use of a defect-based method for such improvement. It addresses a practical question often asked by software developers: I know what I must produce; I know the process I must follow; and I have started development. How do I detect if things are going wrong, and what do I do to get back on track? In other words, we are interested in the real-time improvements a developer can make to his process. We address this issue by presenting a real-life experience which shows how a project team can readily improve their process during development. There have been several, illuminating defect-based studies in the past (such as [5]) but they do not appear to have explored real-time process improvement in detail. Therefore, we believe this paper makes a useful empirical contribution by presenting an experience based on the development of a major software product.

Historically speaking, defect-based process improvement has utilized two kinds of procedures:

1) Analyze every defect to determine their root causes. Modify the process to eliminate those causes. We will refer to this as the *causal analysis* approach. It has its roots in the defect prevention process [6].

2) Track and capture data on defects. Compare that data against specified goals for successful development. Take appropriate action if the goals are not met. We will refer to this as the *goal-oriented* approach, since it has its roots in the goal-question-metric paradigm [7].

We use the case study to demonstrate the contribution of the novel concepts which underlie our method by differentiating our approach from the two historical approaches at appropriate points in the paper.

The organization of the paper reflects the above objectives. It has three main sections which collectively describe the events related to the use of a process improvement method that occurred in the development of a software production project from its inception to the present time. First, the scope, goals, and process activities of the project are presented, followed by an explanation of why causal analysis and goal-oriented methods for defect-based feedback are not completely effective in that context. Using that explanation as the motivation for developing a new defect-based approach, we then describe our method and its integration with the process of the project team (see Section II). Second, the use of the method to make and validate process corrections and the corroboration of such use are described in Section III. The cost and benefit of using the method are also presented. Third, we describe related work in Section V. The material in the preceding sections is used to show that our method can complement current practice in defect-based feedback. Section VI summarizes what we have learned from this work.

II. Process Improvement Method

The case study is about the experience of a project team. The team was developing an operating systems product and constituted 25 people. Their development process included the use of computer-aided software engineering (CASE) technol-

Manuscript received November 1, 1992; revised August 1, 1993. Recommended by N. H. Madhavji and M. H. Pendeo.

I. Bhandari, M. Halliday, J. Chaar, and R. Chillarege are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

E. Tarver and D. Brown are with the IBM Mid-Hudson Valley Programming Laboratory, Wappingers Falls, NY 12590.

IEEE Log Number 9213500.

0098-5589/93$03.00 © 1993 IEEE

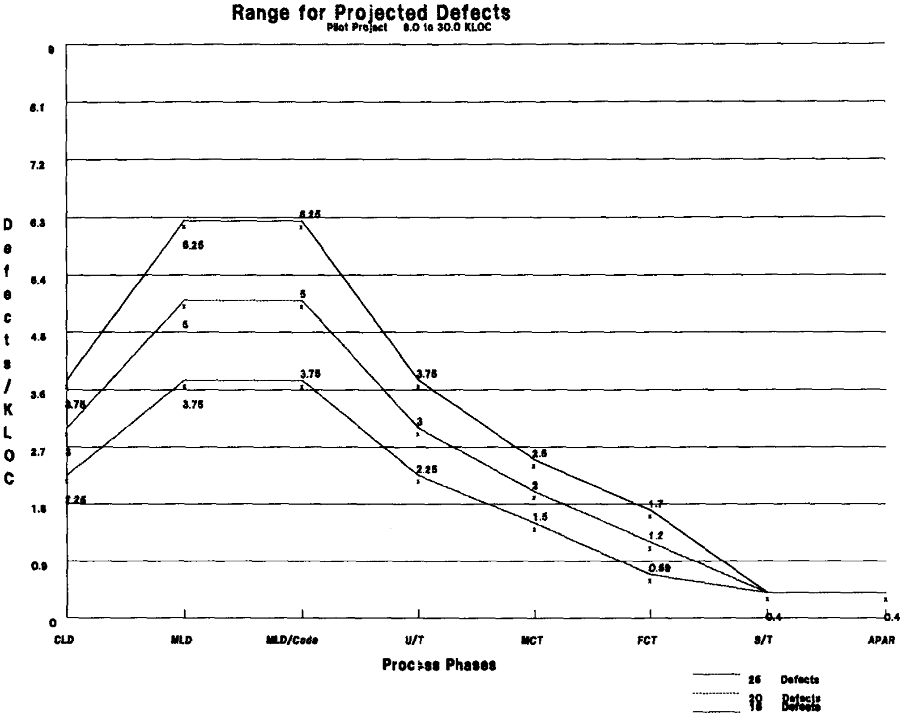


Fig. 1. Initial projections for pilot projects.

ogy and very high-level design tools. Since the team was not experienced in the use of such technology, their process was split into two stages. A prototype project consisting of requirements, component-level design (CLDP), module-level design (MLDP), and modified component test (MCTP) was to be conducted in which less than 5000 lines of code (or 5 KLOC) would be produced, to be followed by the main project consisting of component-level design (CLD), module-level design (MLD), unit test, modified component test, system test, in which the remainder of the code of a mid-sized product (8–30 KLOC) would be produced. The lessons learned from the prototype project were to be used to improve the process for the main project. Other than such improvements, the processes for the prototype project and the main project were to be identical.

Let us understand why there was a need for a new process improvement method by considering the limitations of goal-oriented approaches and causal analysis from the point of view of the project team. We begin with an example of goal-oriented measurement.

The project team used metrics and goals to measure the goodness of their process definition and process execution. The goals were set based on the benchmarks and accomplishments established by other projects. One such goal was to achieve a field defect rate of 0.4 Defects per 1000 lines of code (referred to as D/KLOC in the remainder of this document), to attain the level of quality which was required for the release of the component. As per their quality process, the team had to track whether they were meeting this objective.

To do so a model for defect rate was developed as indicated below. The team looked at data from other projects that ranged in size from 8 KLOC to 270 KLOC, and were from various functional areas of a large operating system. After considering the data of the other projects, and factors such as project complexity, and their projected field defect rate, a defect model was created for a midsized effort such as the pilot project (8–30 KLOC), based on a 20 D/KLOC injection rate (see Fig. 1). The middle line specifies the D/KLOC that the team expected to detect at each of the phases specified in the horizontal axis. The other two lines specify the upper and lower limits of the D/KLOC for such detection. Some of the phases that are listed are not relevant for this paper. For the sake of completeness, we include a brief description of those phases. *U/T, FCT, S/T* denote the test phases *unit test, function component test, and system test,* respectively. *APAR* is an acronym used within IBM to refer to a defect found in the field.

Having developed the defect model, the team could, in principle use a goal-oriented approach to improve their process: Compare the actual defect rate against the model. If it does not fall within the limits specified by the model (Fig. 1), take appropriate action. However, the team was uncomfortable about the approach for the following reasons.

Process Feedback: What should be done if the actual defect data did not follow the defect model? In other words, the defect model only provides the expected profile for defects. While it may be useful to spot deviations from that profile, those deviations do not help one to identify process inadequacies and corrections. They merely serve to point out that a process inadequacy exists.

Defect Model Corroboration: What should be done to verify that the project was on the right track if the actual defect data did follow the defect model? In other words, the defect model is an expectation in which the team did not have confidence. This was not surprising. The model was based on historical data and human judgement. However, as has been mentioned before, the project team was following a process that was based on state of the art tools and hence, quite a departure from the traditional process which was followed in their laboratory.

Clearly, some other kind of in-process feedback mechanism was required, namely, one that did not rely on an historical expectation. We will have more to say on this subject later. For now, let us examine the suitability of causal analysis as a feedback mechanism for the project.

As part of their quality process, the team had to do a *causal analysis* [6] of defects. From past experience, it was known that causal analysis, being a cost intensive and human intensive process, was practical only if used to study a small sample of a large population of defects and therefore, was only a partial feedback mechanism. In other words, there was a need to complement causal analysis by an in-process improvement method which was based on an analysis of the entire population of defects. Given the experience with causal analysis, this restriction ruled out manual analysis of defects. However, as we saw above, that analysis could not be based on historical data, since such data were not available. This restriction ruled out the use of traditional, goal-oriented automatic feedback techniques such as those presented in [8]. However, it appeared that both restrictions could be satisfied by providing feedback based on a particular analysis of classified data as described below. Hence, the team decided to use that method, while continuing to use causal analysis and goal-oriented metrics. The adoption of the method meant that its two principal activities, defect classification and analysis of classified data, be done. Those activities are described below.

A. Defect Classification

In the first step of the process improvement method, defects are classified using attributes whose values partition a set of the activities of the process. In other words, the set of possible values for an attribute distinguish between certain process activities or certain parts of the product. Every defect is classified by choosing a value for every attribute in the classification scheme, thus relating a defect to a subset of process activities which may require improvement. Clearly, such a relationship is essential if the classified defect data are to be used to correct the process. A brief description of the attributes which were used in the case study is given below. The values of the attributes which we will use in later sections are also described. Some of the attributes are the same as, or refinements of, attributes that have appeared in the literature on error classification schemes (e.g., [5]).

The classification scheme used for the pilot project had two kinds of attributes. First, there are the attributes *defect type, missing/incorrect, trigger, source, impact* from orthogonal defect classification (ODC) [9], [10]. These attributes are devised to partition the activities of the software development process. Such partitioning has been described in detail in [9]. *Defect type, missing/incorrect* capture information about the type of activity which was undertaken to fix the defect. For instance, a defect is classified *missing* if it had to be fixed by adding something new to a design document, and classified *incorrect* if it could be fixed by making an in-place correction in the document. Thus, one may say that the *missing/incorrect* attribute partitions the activity of fixing defects into two parts. Similarly, *defect type* also partitions the activity of fixing defects into different types of fixes. For instance, the *type* of a defect is *function* if it had to be fixed by correcting major product functionality. *Source* partitions the product in terms of its development history over releases and identifies the partition in which the defect is located, i.e., it captures whether the defect occurred on account of errant activities in previous releases, the current release, or both. For instance, a defect is classified *new function* if it was found in that part of the product which consisted of new code, and classified *rewritten* if instead, it was found in that part of the product which consisted of code that was part of an old release of the product but was being modified for the present release. *Trigger* captures information about the specific inspection focus or test strategy which 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 which 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.

The second kind of attributes which are used to classify defects are those which are not among the ODC attributes, but which are commonly used to relate a defect to a set of process activities. 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 *CLD* if the defect was introduced during component-level design, else, if the defect was introduced during module-level design it is classified *MLD. Component* identifies the software component in which the defect was located. Clearly, those attributes also relate 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.

The project team members classified all defects found during the reviews of all deliverables produced (i.e., final programming specification document, design structures document,

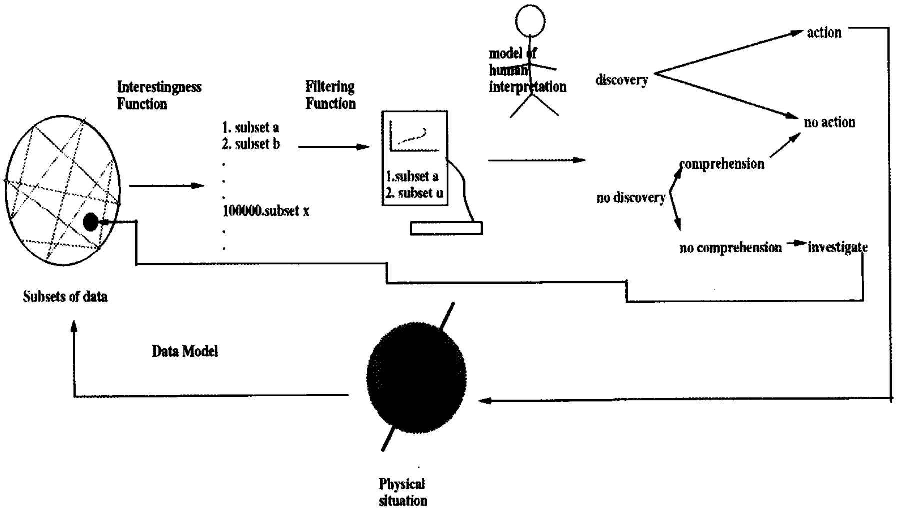


Fig. 2. The attribute focusing approach.

logic and code) as well as during the test phases. Having no prior experience, the defect analysis step was integrated into the process of the project in the most obvious fashion. The classified defects were analyzed after every phase of the process as follows. The analysis was done after the component-level design, module-level design, and modified component test phases of the prototype project. Hence, three analyses were done. For the main project, the analysis was done after both the component-level design and module-level design phases.[1] The process was adjusted to reflect the results of every analysis before proceeding to the next phase. Hence, by measuring and analyzing defects at different points in the process, and by using the results of the analyses to make process adjustments, the process of the project team was converted from what was orginally specified as a standard waterfall process to one which may be periodically adjusted during its execution. The analysis is described below.

B. Defect Analysis and Feedback

The second step of the method 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 [11]. Other experiences based on the application of attribute focusing (AF) to software development have been reported. The use of AF to analyze survey data to suggest improvements *post-process,* i.e., after the end of development, is described in [12]. In [10], the use of AF to assess the effectiveness of inspections and testing methods was described. Those works did not address real-time process improvement in depth, which is the subject of this paper.

The approach is summarized below, followed by a description of its use by the project team. 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 layman instead of the data analyst, a goal that distinguishes it from the usual data exploration system (see, for example, systems described in [13]). The typical software developer or tester fits the above description of a domain specialist.

The key aspects of the approach are illustrated in Fig. 2. 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 Section II-A along with a written description of the defect. The classified data are processed automatically to produce a set of interesting charts which are then interpreted by the project team in a specific manner. We present the mechanical and manual procedures of attribute focusing, exemplify each procedure using data from the pilot project, and point out differences from related approaches.

1) Interestingness and Filtering Functions: First, the classified data are processed automatically by a procedure called an *interestingness function* which orders attribute-values to reflect their potential interestingness for a human analyst.[2] Attribute-values correspond to a set of defects, namely, the defects which were classified using those attribute-values. Hence, Fig. 2 depicts the ordering of attribute values by an ordering of subsets. The data of the project team was 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 [13], [15].

Second, another automatic procedure called a *filtering fuction* processes the orderings produced by an interestingness function and presents it in a manner suitable for human

[1] These were the phases that had been completed at the time of writing this paper.

[2] We are indebted to Ashwin Ram for coining the term "interestingness" [14].

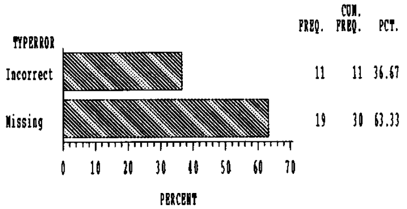
consumption. It makes use of knowledge of human processing of attribue-valued data to do this. The use of such a filtering function is a novel idea, although it is in keeping with the recent emphasis on interactive approaches for data exploration [16].

Let us understand the above concepts in the context of the pilot project. For that purpose, a filtering function was used to produce bar charts which show the spread of values for an attribute, such as the chart in Fig. 3, or which show the cross-product of two attributes such as the chart in Fig. 4. The charts have a legend which presents the values of the attributes in decreasing order of their interestingness. As we shall see, that legend is used to focus the project team on certain trends in the data. The legend does not have more than eight items, which is in keeping with the well-known fact that people find it difficult to retain more than seven plus or minus two items in short-term memory [17]. 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 shown that the number of charts produced should be restricted to less than or equal to 20 [11] (to limit the duration of an interpretation session to about 2 hours.) We observed that restriction in our implementation. Once the project team had classified all the data for a phase, 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.

Next, let us understand the layout of the AF charts to gain a better understanding of the interestingness and filtering functions. Much of the data for the pilot project was generated by classifying defects found during the inspections [18] of the component-level design and module-level design documents of the prototype project and the main project.

The AF chart in Fig. 3 was among the twenty charts generated for the data from the component-level design inspections of the prototype project. This is indicated by the CLDP which appears in the title. The chart shows the distribution of the attribute *missing/incorrect.* The numbers to the right of the chart specify the frequency, cumulative frequency, and percentage of defects for every bar. The legend in the bottom of the chart is a table which indicates the interestingness of the attribute values based on a degree of magnitude. Let us understand the first row of this table. The column *Obs* indicates that 37% of the defects were classified *incorrect.* The column *Expec* shows what the percentage of such defects would have been had the values of the attribute been equally likely. Since *missing/incorrect* has two values, that percentage is 50%. The column *Diff* is the difference between 37% and 50%, or −13%. The Interestingness function computes the difference column to order attribute values. The Filtering function uses the difference to decide which charts should be brought to the attention of the analyst.

The AF chart in Fig. 4 which shows the cross product of the attributes *defect type* and *phase introduced.* As indicated by the MLDP in the title, the chart was among those generated by processing the data from the module-level design inspections of the prototype project. The legend in the bottom of the chart is a table which indicates the interestingness of the attribute values based on a degree of association. Let us understand the first row of the table. Column *Obs1* indicates that 14% of the defects had a *defect type* of *document.* Column *Obs2* indicates that 16% of the defects were introduced in *CLD* of the prototype phase. Column *Obs12* indicates that 5% of the defects were introduced in *CLD* and were of type *document.* 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%, of the defects had been classified *document* and *CLD,* as is indicated in the column *Expec12.* The difference $(5\% - 2\% = 3\%)$ is indicated in the column *Diff,* and serves as a measure of the degree of association between the attribute values in the first row.



Missing/Incorrect	Obs	Expec	Diff
1. Incorrect	37%	50%	-13%
2. Missing	63%	50%	13%

Fig. 3. CLDP feedback—missing/incorrect.

2) Interpretation of AF Charts: Let us continue with the description of the attribute focusing approach. As indicated in Fig. 2, AF charts are interpreted by a domain specialist, called the *analyst,* by using a *model of interpretation* [11]. While a complete discussion of the model interpretation is beyond the scope of this paper, its use will be adequately illustrated later in this section. Essentially, the specialist attempts to explain the interestingness (magnitude or association) of the attribute values that appear in the legend of 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.

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

- *Understand cause of interestingness:*

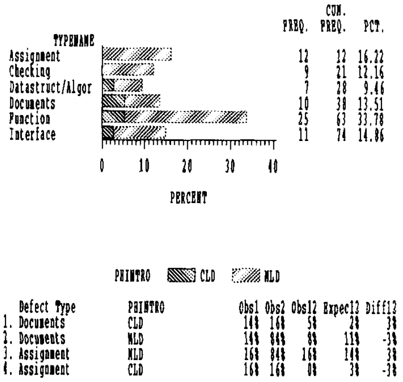
— *a* occurs frequently with *b* but infrequently with *w.* Why? — What event could have lead to such occurrence?

- *Understand 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 the analyst to think directly about the events



	Defect Type	PHINTRO	Obs1	Obs2	Obs12	Expec12	Diff12
1.	Documents	CLD	14%	16%	5%	2%	3%
2.	Documents	MLD	14%	84%	8%	11%	-3%
3.	Assignment	MLD	16%	84%	16%	14%	3%
4.	Assignment	CLD	16%	16%	0%	3%	-3%

Fig. 4. MLDP feedback—defect type versus phase introduced.

in their domain. As shown in Fig. 2, that exercise results in the following.

Discovery of Knowledge: 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.

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 as follows. 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 which 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 the pilot project. To interpret the chart in Fig. 3, 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 such correction. Details are given below.

Cause: Lapses in communication between designers who were working in different subgroups, i.e., designers in a subgroup were making assumptions about what other subgroups were designing without verifying those assumptions with the designers in the other subgroups. Consequently, parts of their design which were based on such assumptions were often incorrect. These lapses in communication were manifested by functionality that was missing in the design, and which was discovered during inspections. Repeated inspections of the design were then done until the team felt confident that the design was complete. Thus, the process problem was identified: A lack of communication between different subgroups.

Implication: Inspections would have to be repeated in the main project as well unless some other corrective action was taken.

Corrective Action: No further corrective action was planned for the CLDP phase. The team felt that by repeating inspections they had already corrected the problem for the prototype project. During the main project, 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, and also be more cost effective than repeating inspections. The fact that the process problem did indeed exist was corroborated by soliciting (and verifying) information from the team which was independent of the data but used by the team to make their decision.

Corroboration: • Consensus among team members that they had not communicated.

• The nature of the *missing* defects as described in their written descriptions.

• The fact that the defects were found in inspections which involved members across different subteams.

Such corroboration was sought for all the process problems reported in this paper. Not only did this exercise help establish that a problem had indeed been found by using attribute focusing, but also allowed us to appreciate the importance of placing the control and decision making in the hands of the project team.

As is clear from the interpretation of the AF chart above, the team made use of knowledge which was not captured by the data to reach a decision. For instance, 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 was not part of the project team will not have that knowledge and hence, quite possibly misinterpret the AF chart.

On a related note, let us consider the detection of the above problem using a goal-oriented approach. Since goals are typically established prior to the start of development, they represent generalizations of previous experiences. In general, it is not unusual for earlier phases such as component-level design inspections to find more *missing* defects (refer to figures on pp. 417–418 in [19]). Fig. 3 is consistent with that general knowledge and hence, may not lead to the detection of the process problem when compared against a goal based on that knowledge. This line of reasoning suggests that *once a project is underway,* it may be important to relate the data not only to previous experience of development as represented by prespecified goals, but also to the immediate experience of development as done by the use of attribute focusing. The reason being that if a particular aspect of that immediate experience is sufficiently different from past experience, a goal-oriented approach may not be effective.

Let us turn to the interpretation of the AF chart in Fig. 4. As indicated by the MLDP in the title, the chart was based on the data from the module-level design of the prototype stage.

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 that led to the identification of a process problem, the team had to come up with a corrective action. While the associations shown in the table were easily explained by the project team, the absence of a strong association between the defect type *function* and phase introduced *CLD* led to the identification of a process problem. Such an association was expected since functional aspects of the design are addressed during CLD. The reason for the absence of that association is clear from the insets in the bar chart. A large number of *function* defects were introduced in *MLD.*

Cause: CLD work was still being done after the beginning of the MLD phase. That work was attributed to an inadequacy in the requirements process. A missing item in the requirements document was responsible for an incomplete high-level

Corroboration:

- Missing item in the requirements document.
- 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 be only 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 document was reinspected for completeness. Again we see that the informed judgement of the team is a key input to the method. The AF chart and its legend serve to focus the team on the right information, which they may not otherwise see owing to the combinatorial complexity of the information in the data. There are simply too many possible two-way charts and too many possible items to focus on even for one such chart.

On a related note, if using a goal-oriented approach, one would compare the data to goals that one knew were relevant based on previous experience. However, as the above examples indicate, once a project is underway, the immediate experience is also important. The patterns in the data which reveal an immediate problem may not be known if they are unique to the present experience. Such patterns will not revealed by a goal-oriented method. Instead, it may be better to use a method such a attribute focusing. Perhaps this concept is best understood by noting that attribute focusing may suggest the analysis of completely different sets of charts for two different projects even if their data are classified using the same attributes. Such behavior reflects the data oriented (as opposed to goal-oriented) nature of the technique, and is necessary to search the vast space of data patterns that may be potentially interesting but that one may not know what to look for.

The Validation Step: Let us return to the description of attribute focusing. There is a *validation* step, which is represented by the cycle in Fig. 2. [11] describes how attribute focusing may be used to validate that corrective actions are having the desired effect. Since the corrective actions are based on the interestingness of attribute values, one can compare the charts that are produced from data before and after the corrective actions. If the interestingness persists, the corrective action is not having the desired effect. Similar monitoring has been described before in the context of process management and control [20, ch. 12]. Hence, we will not discuss it in general, but instead, proceed directly to its use by the project team.

Attribute Focusing was applied as follows to observe the effect of corrective actions in the case study. The classified defect data for every phase were analyzed at the end of that phase. Charts to validate corrective actions were generated by applying attribute focusing to the combined data of the phase and its preceding phases. As will become clear in later sections, charts that show the association between the attribute *phase found* and other attributes highlight the differences of those attributes across phases. Thus, they can be used to validate corrective actions which were undertaken on the basis of those attributes.

Corrective actions were also validated by comparing the charts related to those actions for the prototype project and the main project. That comparison showed whether the lessons learned from the prototype project had been successfully applied in the main project.

III. Results

The results of the case study show that the interpretation of attribute focusing charts allowed the project team to identify and correct process problems, as well as to validate the effectiveness of corrective actions. We discuss the two kinds of results in turn.

A. Problem Identification and Correction

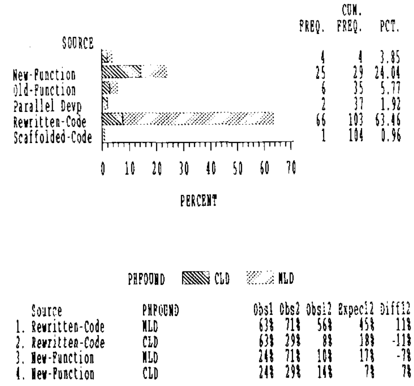
Table I summarizes the process problems which were identified and the corrective actions taken over the course of the project as a direct result of Attribute Focusing based feedback sessions. The column *Trend* indicates the AF chart and the pertinent items (or their absence) in its legend which led to the identification of the problem. The titles of the specified charts, indicate the phase, namely, CLDP (component-level design—prototype project), MLDP (module-level design—prototype project), MCTP (modified-component test—prototype project), CLD (component-level design—main project), or MLD (module-level design—main project), in which the problem was discovered. The problem and the corrective action are also specified in the table. While Section II-B-2 also involved determining the cause, implication, and corroboration for a problem, we omit that information in the interest of brevity.

Problems 1 and 3 were explained in detail in Section II-B-2. Two actions are specified for Problem 1 because different actions were taken in the prototype and main projects. While most of the other entries in Table I are easily understood by reviewing the specified AF charts, some items require explanation.

Problem 4 occurred because the definitions of the values of *source* were ambiguous in the context of the product being developed. For the most part, the functionality of the product

TABLE I
PROBLEM IDENTIFICATION AND CORRECTION

#	Trend	Process Problem	Corrective Action
1	Too many missing defects, too few incorrect; Fig. 3	Lack of communication between subteams	Repeat inspections; teach-the-team sessions during main project
2	Too many capability, too few performance and usability defects; Fig. 11	Nonfunctional areas of design not adequately addressed	Emphasize nonfunctional areas in succeeding phases
3	Function, CLDP lack of association; Fig. 4	Component-level design done at MLDP on discovery of missing requirement	Reinspect requirements document before proceeding further
4	Defects in new function found mainly in CLDP versus rewritten in MLDP; Fig. 5	Inconsistent classification of source by project team	Consistent definition of source attribute values must be used in the future
5	Function, missing association; Fig. 6	Design of recovery may be incomplete	Future inspections should emphasize finding problems in recovery
6	Component India, backward compatibility disassociation; Fig. 12	Design of component India did not address backward compatibility adequately	Complete design for component before proceeding further
7	Component Alpha, function association; Fig. 7	Unplanned CLD iteration for Alpha during MLD on account of vague requirements	Future inspections should emphasize finding problems in Alpha



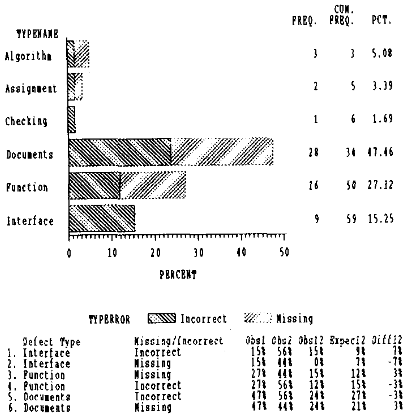
```
          CUM.
   FREQ.  FREQ.   PCT.
      4      4   3.85
     25     29  24.04
      6     35   5.77
      2     37   1.92
     66    103  63.46
      1    104   0.96
```

```
   Source              PHFOUND        Obs1 Obs2 Obs12 Expec12 Diff12
1. Rewritten-Code      MLD             63%  71%   56%     45%    11%
2. Rewritten-Code      CLD             63%  29%    8%     18%   -11%
3. New-Function        MLD             24%  71%   10%     17%    -7%
4. New-Function        CLD             24%  29%   14%      7%     7%
```

Fig. 5. MLDP validation feedback—source versus phase found.

was identical to that of a previous release. However, the code for that part was being completely rewritten from scratch. It did not use the code for the old release as a starting point. Hence, it was not clear whether that part of code should be considered *new function* or *rewritten code.*

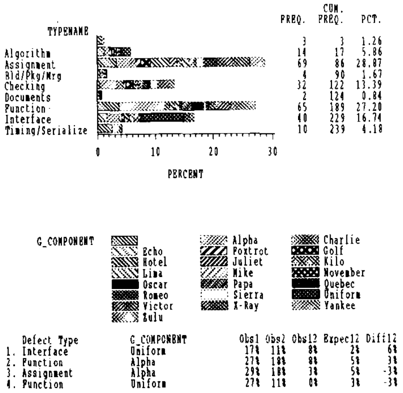
Problem 5 was discovered as follows. The specified trend could not be explained by the project team at the feedback session. Hence, they had to investigate the issue by studying the defects which were classified both *function* and *missing* (as per the model in Section II-B-2). That investigation revealed that all such defects pertained to a specific functionality of the product, namely, its ability to recover from an erroneous state.



```
          CUM.
   FREQ.  FREQ.   PCT.
      3      3   5.08
      2      5   3.39
      1      6   1.69
     28     34  47.46
     16     50  27.12
      9     59  15.25
```

```
   Defect Type         Missing/Incorrect   Obs1 Obs2 Obs12 Expec12 Diff12
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. Documents           Incorrect            47%  56%   24%     27%    -3%
6. Documents           Missing              47%  44%   24%     21%     3%
```

Fig. 6. Main CLD feedback—defect type versus missing/incorrect.



```
          CUM.
   FREQ.  FREQ.   PCT.
      3      3   1.26
     14     17   5.86
     69     86  28.87
      4     90   1.67
     32    122  13.39
      2    124   0.84
     65    189  27.20
     40    229  16.74
     10    239   4.18
```

```
   Defect Type         G_COMPONENT        Obs1 Obs2 Obs12 Expec12 Diff12
1. Interface           Uniform             17%  11%    8%      2%     6%
2. Function            Alpha               27%  18%    8%      5%     3%
3. Assignment          Alpha               29%  18%    3%      5%    -3%
4. Function            Uniform             27%  11%    0%      3%    -3%
```

Fig. 7. MLD feedback—defect type versus component.

Also, in the case of Problem 7, the team felt that the component-level design iteration for Alpha was unavoidable because component Alpha implemented complex functionality and some of the requirements for that functionality were vague and incomplete. Hence, it was only possible to finalize the component-level design for Alpha by exploring some direction to the level of a module-level design.

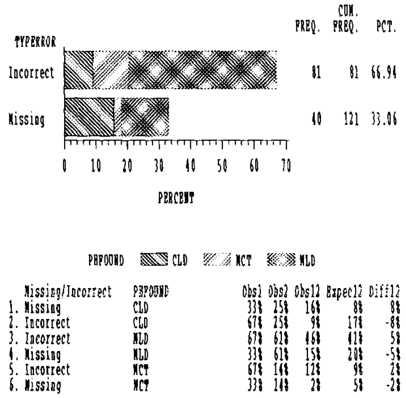
B. Validation of Corrective Actions

The AF charts which show the cross products of relevant attributes with the attribute *phase found* are useful to validate corrective actions. Table II summarizes the results of using those charts. For brevity, we will only present those validations which originated from process problems identified in the Prototype project. This will allow us to determine which problems were truly resolved at the Prototype project and the experience carried over to the Main project.

The column *Problem/Initial Trend* in Table II specifies the chart which triggered the discovery of a process problem. The column *Later Trends* specifies two AF charts, to validate the effectiveness of the corrective action for the prototype and main projects, respectively. The titles of the charts suggest the data which was used to generate the charts. For instance, if the title has "MLD Validation" as part of its title, then it is based

TABLE II
PROBLEM TRENDS

#	Problem/Initial Trend	Later Trends	Was problem Resolved?
1	Lack of subteam communication; Fig. 3	Fig. 8, Fig. 9	Yes, Yes
2	Nonfunctional areas of design; Fig. 11	Fig. 13, Fig. 14	No, No
3	Incomplete and vague requirements; Fig. 4	Fig. 15, Omitted	Yes, No
4	Inconsistent classification; Fig. 5	Omitted, Fig. 16	Yes, Yes



Missing/Incorrect	PHFOUND	Obs1	Obs2	Obs12	Expec12	Diff12
1. Missing	CLD	33%	25%	16%	8%	8%
2. Incorrect	CLD	67%	25%	9%	17%	-8%
3. Incorrect	MLD	67%	61%	46%	41%	5%
4. Missing	MLD	33%	61%	15%	20%	-5%
5. Incorrect	MCT	67%	14%	12%	9%	2%
6. Missing	MCT	33%	14%	2%	5%	-2%

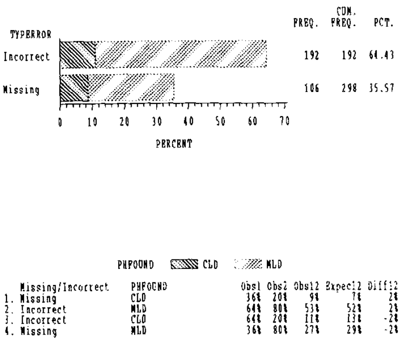
Fig. 8. MCTP validation feedback—missing/incorrect versus phase found.

on the combination of data for the component-level design and the module-level design for the main project. However, if it has "MLDP Validation" as part of the title, it is based on the combination of component-level design and the module-level design for the prototype project.

The last column of the table indicates whether AF charts suggest that a process problem has or has not been resolved. There are two entries per problem corresponding to the prototype project and the main project, respectively.

In order to validate the corrective actions, we must determine the trend we expect to see based on the initial trend which triggered the detections of the associated process problem and the nature of the corrective action (see Table I). Let us consider Process problem 1. The trend which triggered detection of the problem was the large proportion of *missing* defects in Fig. 3. Insofar as the prototype project was concerned, inspections were repeated to ensure the component-level design was complete. If the inspections were successful, we would expect the proportion of missing defects to decrease in later phases, or that *incorrect* would be associated with those phases. From Fig. 8, it is clear that *incorrect* defects are associated with *MLD, MCT* while *missing* defects are associated with *CLD.* The chart shows that the proportion of *missing* defects continues to decrease over the later phases, suggesting that the problem has been corrected.

Next, let us turn to the main project. The corrective action involved the use of teach-the-team sessions. If those sessions were successful, we would expect mainly *incorrect* defects to be found in the main project. From Fig. 9, we see that to be true. Furthermore, it is clear from the insets in the figure, the proportion of *missing* defects has become less than that of the *incorrect* defects in component-level phase of the main project. In other words, the initial trend which triggered the discovery of the problem has been reversed, suggesting that the corrective action has resolved the problem. Thus, the AF charts indicate that the problem of poor communication between subteams had been resolved successfully for both the prototype project and the main project.



Missing/Incorrect	PHFOUND	Obs1	Obs2	Obs12	Expec12	Diff12
1. Missing	CLD	36%	20%	9%	7%	2%
2. Incorrect	MLD	64%	80%	53%	52%	2%
3. Incorrect	CLD	64%	20%	11%	13%	-2%
4. Missing	MLD	36%	80%	27%	29%	-2%

Fig. 9. MLD validation feedback—missing/incorrect versus phase found.

This finding was corroborated by the following information.

- Consensus among team members that there were no communication problems.
- Nature of *missing* defects found during main project suggested that they were caused by poorly understood requirements and not by poor communication.

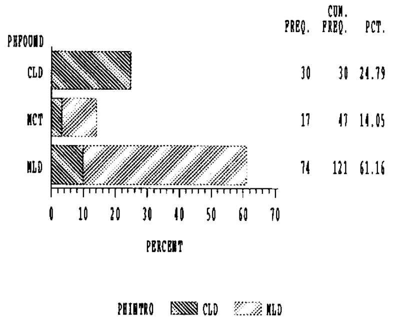
Note the close relationship of the above information to that in "Corroboration" in Section II-B-2. That relationship illustrates how the corroboratory information which was used by the team to determine the existance of a process problem may be used to suggest what should be monitored to see if the problem has been resolved.

The other entries in the last column of Table II were determined in similar fashion, i.e., by using the initial trends, corrective actions, later trends and corroboratory evidence. The initial and later trends specified in Table II in conjunction with the information in Table I may be used to understand the entries in the third column of Table II. We omit the details but do provide some guidance for each case in the Appendix.

C. Lessons from the Case Study

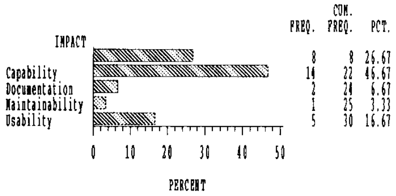
The major lesson from the case study was that the process improvement method works. It can be used by the project team to correct problems in-process and as early in the development cycle as component-level design, and to validate the effectiveness of their in-process corrections. However, in hindsight, it is clear that we could have improved the manner attribute focusing was integrated as a feedback mechanism as discussed below.

Attribute focusing produced the AF chart in Fig. 10, showing the cross product between *phase introduced* and *phase found,* using the combined data for all phases of the prototype project. The figure and its associations indicate that for the most part, defects are detected in the phases where they are introduced; and that the number of defects which had escaped



```
   PHFOUND          PHINTRO          Obs1 Obs2 Obs12 Expec12 Diff12
1. CLD              CLD               25%  38%   25%      9%    15%
2. CLD              MLD               25%  62%    0%     15%   -15%
3. MLD              MLD               61%  62%   51%     38%    13%
4. MLD              CLD               61%  38%   10%     23%   -13%
5. MCT              MLD               14%  62%   11%      9%     2%
6. MCT              CLD               14%  38%    3%      5%    -2%
```

Fig. 10. MCTP Validation Feedback—Phase Introduced vs Phase Found



```
   Impact                 Obs Expec  Diff
1. Capability             47%    8%   38%
2. Installability          0%    8%   -8%
3. Integrity/Security      0%    8%   -8%
4. Reliability             0%    8%   -8%
5. Performance             0%    8%   -8%
6. Migration               0%    8%   -8%
7. Availability            0%    8%   -8%
8. Serviceability          0%    8%   -8%
   --- MORE ---  Use command ANNOT to view more singles.
```

Fig. 11. CLDP feedback—impact.

detection at CLDP and MLDP to be found at MCTP is small. This AF chart is interesting since it made us appreciate a new aspect of the process improvement method. The AF chart suggests that the detection of defects in the part of the product that was produced during the prototype phase is at the point of diminishing returns, a sign of a good quality product. Also, as has been shown above, all relevant corrective actions have been validated, which is another sign of a good quality product. Furthermore, no new process problems were discovered by analyzing the MCTP data. All evidence seems to indicate a good product. This suggests that the method may be useful not only to correct process problems, but also to obtain evidence that one is indeed converging to a good product. This is a promising direction for future work.

We also realized that defects should be analyzed separately for each high-level functionality of the product. As was evident from items 3 and 7 in Table I, design of complex functionality will often result in design iteration. Since the interpretation of AF charts relies on the use of the meanings of the attribute values, and *phase found* is an important attribute-value for interpretation, data from different phases should not be mixed and labeled as being found in a particular phase. But that is exactly what will happen if the defects from different functionalities are not separated and unplanned activity iterations occur for some of the functionalities.

Table II summarizes instances of process corrections which resolved a process problem and those which did not resolve it. The effectiveness of the process correction is determined by following a validation procedure which can span a number of phases. Clearly, it is desirable to learn as soon as possible whether a process change is going to be effective. One possibility is to assess the potential effectiveness of process corrections based on the nature of the correction. On reviewing Tables I and II, it is seen that the corrections which did not resolve the problem did not suggest concrete process change. Instead, they relied on the human ability to perform a little better when focused on a specific problem. In contrast, the corrections which did resolve the problems specified processes which could be implemented and enforced, such as the "teach the team" sessions proposed to improve communication between subteams. This suggests that one dimension to assess the effectiveness of corrections is whether they specify actual mechanisms to solve the problem or simply leave it to team members to find their own solutions. Had such as assessment been done in the Prototype project, we would have known that the project team was struggling to clarify requirements. Alternative corrections, such as the addition of an expert on customer requirements to the team, could have been pursued in conjunction with what was already being done. Similarly, action could have been taken to help the project team define a process for addressing nonfunctional areas of design such as usability and performance (Table I). This observation suggests the following amendment be made to the model of interpretation when attribute focusing is used for process feedback.

Specify that two types of knowledge will be discovered, namely, problem identification and process corrections. In the context of process correction, include a step to assess the effectiveness of a correction based on whether it specifies an actual mechanism to correct the problem. Corrections that do not specify a mechanism may indicate that the project team is struggling to solve a problem, and that while the correction may improve the situation, it may not solve the problem. The search for an alternative correction should be pursued in conjunction with what is already being done.

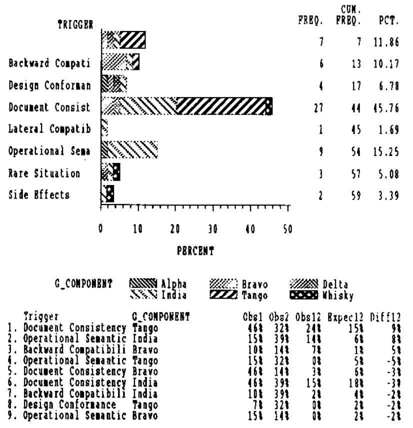
IV. Cost

The cost of using our method is best assessed from the viewpoint of the project team. The project team had identified areas in their process that warranted individualized ownership or focus. Data collection, metrics, and goals was one of these areas. A team member had the responsibility of ensuring that the data were collected and analyzed, and scheduling the feedback sessions. These activities required approximately 10–20% of the individual's time.

The cost of classifying the defects is less than 4 person minutes per defect [9]. Thus, a thousand defects may be classified in fewer than 3 person days. For the pilot project, about 400 defects were classified, for which the estimated cost is bounded by 27 person hours.

The generation of the AF charts from the data is done automatically , and takes less than five minutes of wall clock time for each data set. The cost to the project team is negligible.

Finally, there is the cost of the feedback session itself, which lasts about 2 hours, and must include key members of the project team. The cost in person hours for the entire project is simply the product number of phases, the average number of attendees, times 2. For the pilot project, this worked out to (2*8*5 = 80) person hours. Hence, the total cost for the project is 117 person hours plus 10-20% of one individual's time. Given the time over which the case study was done, we estimate the total cost to be no more than 25% of one individual's time.



```
   Trigger                  G_COMPONENT           Obs1  Obs2  Obs12  Expec12  Diff12
1. Document Consistency Tango                      46%   32%    24%      15%      9%
2. Operational Semantic India                      15%   39%    14%       6%      8%
3. Backward Compatibili Bravo                      10%   14%     7%       1%      5%
4. Operational Semantic 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 Compatibili India                      10%   39%     2%       4%     -2%
8. Design Conformance   Tango                       7%   32%     0%       2%     -2%
9. Operational Semantic Bravo                      15%   14%     0%       2%     -2%
```

Fig. 12. CLD feedback—component versus trigger.

V. Related Work

The main thesis of this paper—that attribute focusing based analysis of classified defect data can readily lead a project team to correct their process in real time—has been established as evidenced by the results in Section III and the computation of cost in Section IV. In this section, let us understand how our method complements other approaches to in-process improvement.

We start with causal analysis. A detailed causal analysis of a single defect can often engage a team of four for as much as fifteen minutes, a cost of a person hour per defect. In the case study, around 400 defects were analyzed. If every defect underwent causal analysis, the total cost is 400 person hours, which is about 10 person weeks. To cut the cost, only a sample of the defects in a project undergo causal analysis. It is worth noting that the total analysis cost to the project team in the case study was 80 person hours, or 2 person weeks (Section IV), a five-fold improvement.

As was mentioned in Section II, the project team also used causal analysis. Thus, the case study demonstrates that the interpretation of AF charts can lead to the correction of process problems over and beyond those problems corrected by causal analysis. This assessment is also supported by the details of the case study which suggest that the two methodologies are complementary and can be used in synergy.

- Causal analysis feedback, being labor intensive, is usually based on an in-depth study limited to a subset of the total defects. It was thus used by the project team. Therefore, process problems that are associated with the defects that are not considered may remain uncorrected.
- Since causal analysis feedback is based on an in-depth study of individual defects, process problems that are manifested by trends in the defect population may remain uncorrected. Problem 6 in Table I is a good example of such a problem. A causal analyst would have found it difficult to determine that compatibility had not been addressed in the design of component India as this cannot be discerned by studying defects found in component India individually. The problem could only surface by observing that while compatibility had been addressed in other components, they were not addressed in component India. To make such an observation, one must study the AF chart in Fig. 12.
- Since the classification scheme generalizes the information that is present in a defect, not all the information in a defect is processed by attribute focusing. Therefore, problems that can only be uncovered by reading the written description of the defect, will not be found by using the process improvement method. These problems can be found by causal analysis.
- AF charts list attribute-values, every entry refers to a subset of the defects that have the corresponding attribute-value. If the interestingness of an entry cannot be explained easily, the relevant defects are investigated by studying their written descriptions. This may be viewed as a causal analysis of a subset of defects that are identified by an AF chart. Hence, the process improvement method helps focus causal analysis. Process Problem 5 in Table I is a good example.

Next, let us consider goal-oriented approaches. As was mentioned in Section II, the project team also used goal-oriented analysis. Thus, the case study demonstrates that the interpretation of AF charts can lead to the correction of process problems over and beyond those problems corrected by such analysis. This assessment is also supported by the details of the case study which suggest that the two methodologies are complementary and can be used in synergy.

- Our approach does not depend on the availability of historical information to be effective. We believe such an approach is a significant development. For instance, our experience has been that the development of a statistical baseline for as diversified an exercise as software production is often difficult. Given the continual advances in software engineering tools and techniques, we think that this condition will persist. This expectation was borne out in the case study. The project team was using a state of the art process and hence, usable baselines were not available for the project (Section II).
- It is evident that the use of attribute focusing complements goal-oriented methodology. Clearly, the interestingness functions cannot substitute for goals based on hard-won experience in instances where such experience can be applied. On

the other hand, as exemplified in Section II-B-2, goal-oriented methods may find it difficult to detect problems that are unique to the current experience of the project. Those problems may be detected by using attribute focusing.

- One may argue that, *after a project team has started development* it is these uncommon problems that are the real troublespots, since the goal-oriented methods that are part of their process will neutralize the adverse effects of other problems. Hence, it is important to investigate how the attribute focusing approach could be used in conjunction with goal-oriented methods. An obvious way is to use a slightly modified interestingness function, one which removes the patterns that are checked for by goal-oriented methods from the output of ordered attribute-values. The remainder of the procedure is unchanged. However, as per the example on *missing* defects in Section II-B, one may not necessarily want to remove all goal-oriented patterns. A better solution would be for the interestingness functions to tag some of the patterns checked for by goal-oriented methods and then, one could interpret those patterns along both goal-oriented and data-oriented lines. Once uncommon patterns are discovered, they may be used to refine the goal-oriented part of the process for the next project. We have begun work in this direction.

Next, let us understand what the case study tells us about our method when viewed in the broader context of project management and control. Clearly, a goal of the management of the project team must have been to identify and correct such problems as in Table I. But those problems remained unidentified and uncorrected until the attribute focusing feedback sessions. What does this tell us? From a scientific viewpoint, not very much, since we have not discussed the details of the procedures used to manage and control the project team other than those that were defect based. On the other hand, from a practical viewpoint, the case study is quite revealing. Note that our study is not a classroom study or an experiment in a controlled environment. It is the study of a team involved in the actual production of a large operating system. We can expect that the team and its management did the best they could do, and therefore, that their effort is representative of current practices in project management and control. Therefore, one may compare our method to *that* effort to conclude that it can complement current practice in project management control

On a related point, note that the project-related information other than defect data may also contain information which is unique to the current experience of the team, and hence, amenable to the attribute focusing approach (perhaps, using different interestingness and filtering functions than those in the implementation described here). We have begun work in this direction.

Our approach complements the use of statistical defect modeling approaches which predict the reliability of a software product ([21]–[23]). Such approaches cannot be used for detailed process correction as has been shown in this case study. For instance, they cannot suggest that communication between subteams needs to be improved. (see Table I, Problem I). On the other hand, we do not presently understand how AF charts can be used to quantify reliability in the manner of the software reliability approaches, although, as shown in Section III-C, there is evidence that AF charts can suggest that one is converging to a good quality product. More work needs to be done in that direction.

The attribute focusing validation step also represents a natural means of growing an "experience factory" [2], [7]. The comparison of trends in Section III-B illustrates how the experience of the prototype project was used to advantage in the main project. In like fashion, the trends that have been highlighted for a project can be checked for and corrected in similar projects. However, in general, our method finds trends which reflect the particular experience of a project team, and more analysis needs to be done to determine how the trends should be interpreted in the context of another project. We have begun work in these directions.

In summary, we conclude that attribute focusing based feedback can complement current practice of in-process improvement and offers many promising directions for further investigation. Those directions suggest that this case is only a modest first step and much work remains to be done.

VI. Conclusion

We have presented a detailed case study of the use of an in-process improvement method by a midsized project team which was developing an operating systems product. We have described how the method was integrated in the process of the project and used to make real-time process adjustments, as well as used to validate those adjustments. Limitations and scope of the method have been specified where appropriate, and the approach evolved to reflect the lessons learned from the case study. We have discussed the differences between the approach and the related work in Section V.

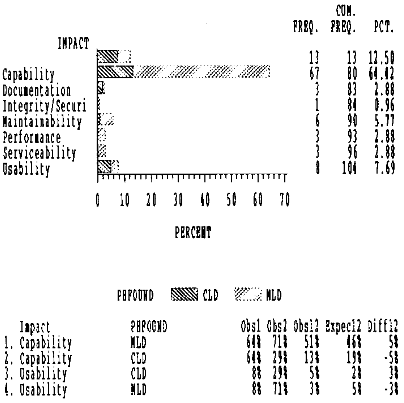
In conclusion, we summarize the major lessons from our case study.

- Machine-assisted data exploration of classified defect data can readily lead a project team to improve their process during development.
- Such analysis focuses the team on the immediate experience of development and helps them correct process problems as well as to validate whether those problems have indeed been corrected.
- Such analysis can complement current practices of in-process improvement.

Appendix

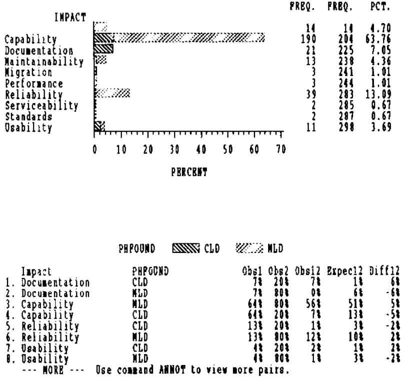
We now discuss the remaining entries of Table II, starting with Problem 2. Problem 2 was not resolved. Given the initial trend and the corrective action (Table I), the proportion of *capability* defects should have reduced while the proportion of *usability* and *performance* defects, should have increased during later phases. Those changes did not occur. (Figs. 13 and 14).

Problem 3 was resolved for the Prototype phase. No further defects were found in the part of the requirements document which correspond to the prototype project. Fig. 15 shows that only a few *function* defects were found during Modified Component Test. The main project was a different story. As evidenced by Problem 7 in Table I, a requirements-based defect did surface during that project.



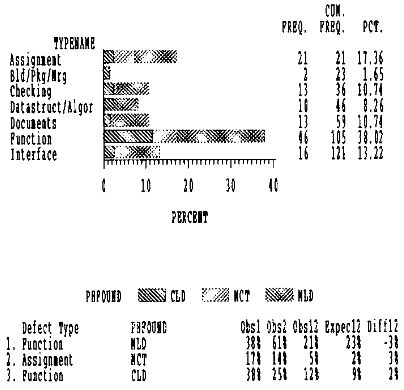
```
   Impact              PHFOUND              Obs1 Obs2 Obs12 Expec12 Diff12
1. Capability          MLD                   64%  71%   51%     46%     5%
2. Capability          CLD                   64%  29%   13%     19%    -5%
3. Usability           CLD                    8%  29%    5%      2%     3%
4. Usability           MLD                    8%  71%    3%      5%    -3%
```

Fig. 13. MLDP validation feedback—impact versus phase found.



```
   Impact              PHFOUND              Obs1 Obs2 Obs12 Expec12 Diff12
1. Documentation       CLD                    7%  20%    7%      1%     6%
2. Documentation       MLD                    7%  80%    0%      6%    -6%
3. Capability          MLD                   64%  80%   56%     51%     5%
4. Capability          CLD                   64%  20%    7%     13%    -5%
5. Reliability         CLD                   13%  20%    1%      3%    -2%
6. Reliability         MLD                   13%  80%   12%     10%     2%
7. Usability           CLD                    4%  20%    2%      1%     2%
8. Usability           MLD                    4%  80%    1%      3%    -2%
   --- MORE ---  Use command ANNOT to view more pairs.
```

Fig. 14. MLD validation feedback—impact versus phase found.



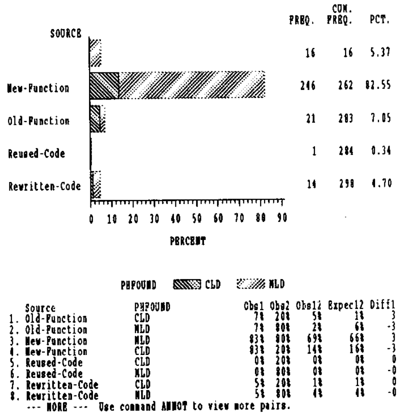
```
   Defect Type         PHFOUND              Obs1 Obs2 Obs12 Expec12 Diff12
1. Function            MLD                   38%  61%   21%     23%    -3%
2. Assignment          MCT                   17%  14%    5%      2%     3%
3. Function            CLD                   38%  25%   12%      9%     2%
```

Fig. 15. MCTP validation feedback—defect type versus phase found.

Fig. 16 shows that Problem 4 was resolved for the main project. Most defects are classified *new function* in keeping with the corresponding corrective action in Table I. A similar chart was obtained after the Modified Component Test phase of the prototype project (omitted).

ACKNOWLEDGMENT

The authors would like to acknowledge the key role played by the following individuals to transform our process improvement method from concept to reality: D. Alpert, D. Butler, C. Coppola, R. Dyer, S. Horowitz, J. Kong, M. Koury, H. Morgenstern, Y. Tan, and C. Vignola, for participating in the study; N. Roth, for the many astute observations that have enhanced the process improvement method; A. Dooley, for helpful suggestions and leads. Finally, we thank the reviewers for doing a tremendous job. There is a world of difference between the final and initial drafts of this paper and they are primarily responsible for that improvement.



```
   Source              PHFOUND              Obs1 Obs2 Obs12 Expec12 Diff12
1. Old-Function        CLD                    7%  20%    5%      1%     3%
2. Old-Function        MLD                    7%  80%    2%      6%    -3%
3. New-Function        MLD                   83%  80%   69%     66%     3%
4. New-Function        CLD                   83%  20%   14%     16%    -3%
5. Reused-Code         CLD                    0%  20%    0%      0%     0%
6. Reused-Code         MLD                    0%  80%    0%      0%    -0%
7. Rewritten-Code      CLD                    5%  20%    1%      1%     0%
8. Rewritten-Code      MLD                    5%  80%    4%      4%    -0%
   --- MORE ---  Use command ANNOT to view more pairs.
```

Fig. 16. MLD validation fedback—source versus phase found.

REFERENCES

[1] W. Humphrey, *Managing the Software Process.* Reading, MA: Addison-Wesley, 1989, pp. 301–334.

[2] V. R. Basili and H. D. Rombach, "The TAME project: Towards improvement-oriented software environments," *IEEE Trans. Software Eng.*, vol. 14, pp. 758–772, June 1988.

[3] H. Rombach, G. Ulery, and J. Valett, "Toward full life cycle control: Adding maintenance measurement to the SEL," *J. Syst. Software,* vol. 18, pp. 125–138, 1992.

[4] N. H. Madhavji, "The process cycle," *IEEE/BCS Software Eng.*, vol. 6, no. 5, pp. 234–242, Sept. 1991.

[5] V. R. Basili and H. D. Rombach, "Tailoring the software process to project goals and environments," in *Proc. Int. Conf. Software Engineering.* New York: IEEE Computer Society Press, Apr. 1987, pp. 345–357.

[6] R. Mays, C. Jones, G. Holloway, and D. Stidinski, "Experiences with defect prevention," *IBM Syst. J.*, vol. 29, no. 1, 1990.

[7] V. R. Basili, "Software development: A paradigm for the future," in *Proc. Compsac.* New York: IEEE Computer Society Press, 1989, pp. 471–485.

[8] V. R. Basili and R. W. Selby, "Paradigms for experimentation and empirical studies in software engineering," *Rel. Eng. Syst. Safety,* vol. 32, pp. 171–191, 1991.

[9] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Y. Wong, "Orthogonal defect classification—A concept for in-process measurement," *IEEE Trans. Software Eng.*, vol. 18, pp. 943–956, Nov. 1992.

[10] J. Chaar, M. Halliday, I. Bhandari, and R. Chillarege, "In-process metrics for software inspection and test evaluations," *IEEE Trans. Software Eng.*, accepted for publication, 1993; also available as IBM Res. Rep. RC (log 80725), 1992.

[11] I. Bhandari, "Attribute focusing: Machine-assisted knowledge discovery applied to software production process control," in *Proc. Workshop Knowledge Discovery in Databases,* AAAI Tech. Rep. Series, Rep. No. WF–93–02, July 1993; also in *Knowledge Acquisition*, accepted for publication.

[12] I. Bhandari and N. Roth, "Post-process feedback with and without attribute focusing: A comparative evaluation," in *Proc. Int. Conf. Software Engineering,* May 1993, pp. 89–98.

[13] G. Piatetsky-Shapiro and W. Frawley, *Knowledge Discovery in Databases.* Menlo Park, CA: AAAI Press/MIT Press, 1991.

[14] A. Ram, "Knowledge goals: A theory of interestingness," in *Proc. 12th Annu. Conf. Cognitive Science Soc.*, Aug. 1990.

[15] J. Mingers, "An empirical comparison of selection measures for decision-tree induction," *Machine Learning,* vol. 3, 1989.

[16] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus, "Knowledge discovery in databases: An overview," in *Knowledge Discovery in Databases,* G. Piatetsky-Shapiro and W. Frawley, Eds. Menlo Park, CA: AAAI Press/MIT Press, 1991.

[17] G. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psychol. Rev.,* vol. 63, 1956.

[18] M Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.,* vol. 15, no. 3, 1976.

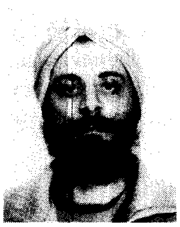
[19] T. Gilb and S. Finzi, *Principles of Software Engineering Management.* Reading, MA: Addison-Wesley, 1988.

[20] A. Mayerhauser, *Software Engineering—Methods and Management.* San Diego, CA: Academic, 1990.

[21] J. Musa, A. Iannino, and K. Okumoto, *Software Reliability—Measurement, Prediction, Application.* New York: McGraw-Hill, 1987.

[22] A. L. Goel, "Software reliability models: Assumptions, limitations," *IEEE Trans. Software Eng.,* vol. SE-11, no. 12, pp. 1411–1423, 1985.

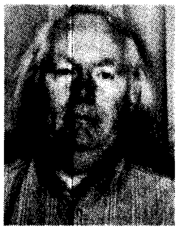
[23] B. Ray, I. Bhandari, and R. Chillarege, "Reliability growth for typed defects," in *Proc. IEEE Reliability and Maintainability Symp.,* 1991.



Inderpal Bhandari received the B.Eng. degree from the Birla Institute of Technology and Science, Pilani, India, the M.S. degree from the University of Massachusetts at Amherst, and the Ph.D. degree from Carnegie Mellon University.

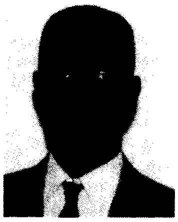
He is a member of the Research Staff at the IBM Thomas J. Watson Research Center. His primary research areas are software engineering (process, metrics, and feedback), and artificial intelligence (knowledge discovery and diagnosis).

Dr. Bhandari is a member of the IEEE Computer Society.



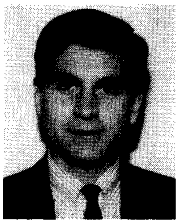
Michael J. Halliday graduated from the University of Nottingham, England.

He is a Senior Programmer at the IBM Thomas 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.



Eric D. Tarver received the B.S. degree in computer science from Jackson State University.

He is an Associate Programmer at the IBM Mid-Hudson Programming Laboratory. He joined IBM four years ago 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.



David D. Brown received the B.A. degree in mathematics from the University of Utah in 1973 and the Ph.D. degree in history from the University of Hawaii in 1981.

He is an Advisory Programmer at the IBM Mid-Hudson Valley Programming Laboratory. Since joining IBM in 1983, he has been involved in technical competitive analysis, and in the design and development of the MVS/ESA operating system. He has five years management experience in projects relating to large virtual memory and highly parallel system services. His current work focuses on the use of metrics to improve software process maturity, and effective transfer of object-oriented technology into large development environments.

Dr. Brown was a Fulbright Fellow in Japan and a Post-Doctoral Fellow at Harvard University.

Jarir Chaar received the B.E. (with distinction) degree from the American University of Beirut, Lebanon, in 1981, and the M.S. degree in electrical and computer engineering and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor.

He is a Research Staff member at the IBM Thomas J. Watson Research Center. Prior to joining IBM, he worked as a Senior Software Engineer at Deneb Robotics Inc. He also held the positions of Instructor with the Department of Electrical Engineering at the American University of Beirut (1983–1985) and Manager and Systems Engineer at Computeknix Microcomputers (1982–1984).

Ram Chillarege received the B.Sc. degree in mathematics and physics from the University of Mysore, the B.E. (with distinction) in electrical engineering and the M.E. (with distinction) degrees in automation from the Indian Institute of Science, and the Ph.D. degree in electrical and computer engineering from the University of Illinois in 1986.

He is currently Manager of Continuous Availability and Quality at IBM Thomas 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 workload 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 an Associate Editor of the IEEE TRANSACTIONS ON RELIABILITY.