# A Methodology for Cost-Effective Analysis of In-Place Software Processes

Jonathan E. Cook†, Lawrence G. Votta‡, and Alexander L. Wolf†

†Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
{jcook,alw}@cs.colorado.edu

‡Systems and Software Research Laboratory
Bell Laboratories
Naperville, IL 60566 USA
votta@bell-labs.com

University of Colorado
Department of Computer Science
Technical Report CU-CS-825-97 January 1997

A version of this report under second review for
IEEE Transaction on Software Engineering

## ABSTRACT

*Process studies and improvement efforts typically call for new instrumentation on the process in order to collect the data they have deemed necessary. This can be intrusive and expensive, and resistance to the extra workload often foils the study before it begins. The result is neither interesting new knowledge nor an improved process. In many organizations, however, extensive historical process and product data already exist. Can these existing data be used to empirically explore what process factors might be affecting the outcome of the process? If they can, organizations would have a cost-effective method for quantitatively, if not causally, understanding their process and its relationship to the product.*

*We present a case study that analyzes an in-place industrial process and takes advantage of existing data sources. In doing this, we also illustrate and propose a methodology for such exploratory empirical studies. The case study makes use of several readily available repositories of process data in the industrial organization. Our results show that readily available data can be used to correlate both simple aggregate metrics and complex process metrics with defects in the product. Through the case study, we give evidence supporting the claim that exploratory empirical studies can provide significant results and benefits while being cost effective in their demands on the organization.*

The work of J.E. Cook and A.L. Wolf was supported in part by the National Science Foundation under grant CCR-93-02739 and by the Air Force Material Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

## 1 Introduction

Analyzing a software engineering process to determine how the process impacts the resulting product is an important task in producing quality software. Current methods for this analysis first define what is to be learned, and then instrument the process for data collection and metrics calculation to support that learning. This approach is embodied in the Goal-Question-Metric paradigm and its descendants [1, 2, 4, 10], and reflects an orientation toward experiments that are both controlled and confirmatory. While it has succeeded in substantially advancing our understanding of general software engineering methods, typified by the studies of Cleanroom Software Engineering [9, 11], the approach can be both expensive and intrusive when applied to in-place, specialized industrial software processes that are seeking rapid improvements. Furthermore, it often ignores the past history of a process, only viewing the process from a point in time after the instrumentation has been established.

What is overlooked is that in-place industrial processes generally have existing, routinely collected repositories of readily available data that can be mined for information useful in empirical process analysis. The repositories derive from the use of such facilities as source code control systems, problem tracking systems, time and activity reporting forms, and the like. If one could take appropriate advantage of these data sources, then a quantitative empirical framework becomes available to explore the current process and assess its performance, without forcing additional instrumentation.

Not only does such an approach provide a cost-effective means for analyzing an existing process, it gives one a much stronger foundation from which to advocate for subsequent instrumentation of the process. This approach brings the people involved in the process on board by showing early results of what may be learned from empirical process analysis, and smooths the way for further process improvement efforts.

In this paper we advocate a methodology of using a historical, exploratory approach to analyze in-place, specialized software processes. Because exploratory studies will necessarily vary in their specifics due to the inherent differences in subject processes, we present the methodology largely "by example" through a case study of a real-world industrial process. What we hope to do is give the reader new ideas for finding, combining, and using existing data repositories to learn about and improve a process, as well as give an awareness of the issues that arise in such a study. We view this case study and its results as first evidence of the viability of our approach.

In the next section we give a brief overview of the methodology, the case study, and our results. The two sections following that provide the details: Section 3 describes the process under study, the sources of data, and the analysis methods used; Section 4 presents the results of our analyses. We conclude in Section 5 with some general observations on the importance of exploratory studies and the significance of our results. Appendices A and B provide details of the data used in the case study.

## 2 Overview

Presented here is an outline and discussion of the methodology of exploratory study that we advocate, followed by an overview of our specific case study as it relates to the methodology.

### 2.1 The Methodology

The methodology we propose for the exploratory analysis of a project guides a study through the following steps:

1. *Commence understanding the organization, project, and process.* This involves talking with developers, reading project and process documentation, and learning the organizational culture. One need not completely understand all facets of these domains before proceeding to subsequent steps. Indeed, there is a danger in lingering in this step too long—that the organization will begin to lose interest in cooperating with the study.

2. *Identify possible data sources.* The sources may be widely varied, such as version control systems (for documents as well as source code), problem tracking systems, tool logs, project effort databases, inspection and review databases, and communication logs. Furthermore, it may not always be obvious how to integrate these sources. For example, version control comments may have keys hidden in them that identify a record in the problem tracking system.

3. *Identify success metrics of the project and process.* The goal of exploratory analysis is to find aspects of the project and process that correlate with some measure of goodness or success of the project. Thus, having success metrics is important. These might take the form of the outcome of an integration test for a change, the speed or efficiency of the process, the defect density per subsystem, customer satisfaction per feature, and other similar metrics. A success metric acts as a dependent variable.

4. *Identify metrics computable from data sources.* With the data sources in hand, and knowledge of how they can be integrated, metrics can be identified that are both computable from the data sources and potentially informative in content. Common examples include size metrics (e.g., number of source lines of a change), time metrics (e.g., total time for a change, delay between process steps), and effort metrics. Further, with understanding gained in Step 1, hypotheses connecting these metrics to the dependent variable can often be stated. These hypotheses help to identify possible mechanisms. These metrics are independent variables that are then evaluated for correlation with the dependent variable (the success metric).

5. *Extract the data and perform analyses.* Data extraction usually will involve writing scripts that can access the data sources and format and integrate the data. Analysis will involve statistical and plotting tools, and perhaps other tools as determined by the chosen analyses.

6. *Interpret the results.* Any significant results (and sometimes surprising insignificant results) should be interpreted in the context of the project and organization. This will involve taking results back to the organization, and seeing if they can provide insight and understanding for the specific results obtained.

Thus, in general, the idea is to interview the team to discover the data sources, learn how to extract information from those sources, determine a method by which the separate data can be integrated, and establish a procedure for analyzing the data.

By looking at historical data, we are limited in two respects. First, we can only investigate process features for which we have data; some features that may be desirable to examine might not have any historical data available to describe them. Moreover, the data themselves may not exactly represent the process feature that we are interpreting from it, so questions of construct validity (*"Does the metric represent the real-world aspect you think it does?"*) arise as well. The second limitation is that we can only present evidence, in the form of a correlation, that a particular process feature may be having an impact on the product. We cannot confirm through direct manipulation of the independent variable that a causal relationship exists. Nevertheless, given the high cost of experimentation, the correlation is useful in suggesting places to focus further investigation or even experimentation.

Even with these limitations, the use of historical, readily available data allows one to explore a process in detail and in a cost-effective manner. No extra data collection is necessary and there is little intrusion into the developers' activities. Compared to new instrumentation on a process, this effort is minimal. A similar kind of historical data analysis was successfully employed in a recent study conducted by Votta and Zajac [14], in which they looked at process waiver data (data indicating when a process was exempted from its prescribed activities) and correlated this with the outcome of the product.

We call a study done under this methodology a case study because it looks at a specific, contemporary development organization and project, although it borders on what Yin ([16]) would call a historical study because it uses existing data. Case studies generally identify correlations rather than confirm causality, and our exploratory methodology does the same. A common feature of case studies is that they have preconceived hypotheses, and then collect data to test those hypotheses. In our historical data analysis, we do not generally have preconceived quantitative hypotheses. Rather, we explore the available data for interesting relationships—our qualitative hypothesis is that the available data contains significant relationships that will tell us something about our process. In finding data sources and understanding the project and process, however, specific hypotheses may be formulated and then tested against the data.

We as researchers and scientists might prefer that every study be experimentally controlled and to be able to confirm that suspected causal relationships do indeed hold. But the fact remains that for any specific organization's process to be improved, controlled confirmatory experiments are usually prohibitive in cost. An organization is interested in improving their process in the most cost-effective manner possible. We have found that exploratory studies based on readily available data can be an effective aid in process improvement.

## 2.2 An Application of the Methodology

The case study we performed examined historical data gathered from a repetitive software update process. We correlated process metrics with a measure of the success of each process execution to show how a process can be quantitatively understood in terms of the process features that might be significantly affecting the resulting product. The metrics we look at encompass both aggregate features of the process, such as the time it takes to complete the process, and measures of how well the process executions themselves correspond to a formal model used to specify how the organization expects the project to behave.

Our study proceeded as follows, with the numbers in parentheses denoting the methodology step that the activity encompasses.

- We started by talking to a member of the process improvement team about the organization and its process (1), and about what data sources might be available (2).
- Parallel with this we began collecting and studying the process documentation to understand what the organization viewed as the "ideal" process (1).
- Once data sources were identified, the process team member directed us to various people in charge of the different data sources. They assisted us in locating relevant data, and pointed out how to extract data from in-house specialized databases (2). In the case of the code inspection reports, this meant which stack of binders to wade through!
- In the data sources we had available, the success metric for each process execution was defined as whether the fix was accepted or rejected by the customer (3).
- Metrics were identified that were computable from the available data sources (4). These were a combination of product-based (e.g., number of source lines changed, number of source files changed), process-based (e.g., total time for the process execution, internal delay time, developer who was involved), and process-behavior-based (e.g., how closely the process execution followed a process model) metrics. Hypotheses were also formed with these metrics (4), such as "the more source lines changed for a fix, the more likely the fix is to fail (be rejected)", and "the closer a process follows the process model, the more likely the fix is to succeed (be accepted)".
- At this point, the effort turned to writing scripts that would extract data from the various sources and merge the extracted data with data from other sources (5). Decisions on data selection (we chose to restrict ourselves to the changes made to one system version) and inspections for data consistency were also done at this time.
- Once the data collection was complete, scripts, analysis tools, and statistical tools were applied to perform the actual analyses of the data (5).
- Interpretation of the results (6) was done by describing the study in a report, making this accessible to the organization, and providing some qualitative understanding of the results.

The cost to the development organization of our study was virtually negligible compared to the size of the quality group's annual improvement budget ($\ll$1%).

As a first part of our case study, simple aggregate metrics, such as the number of source lines changed and the elapsed time of the process, were examined for correlation with the process success metric. In the second part of the study, process validation tools [5] were used to measure the correspondence between the process as executed and the process as prescribed by a formal process model.

We found two aggregate metrics that correlated with the defect metric (customer rejecting the fix): the *delay* between the appearance of a customer problem report and the beginning of the activity to make the fix, and the *developer* who performed the fix. We also found significant differences between how the successful and unsuccessful processes followed the prescribed process, with the unsuccessful ones deviating from the process model more than the successful ones.

It is important to understand these results in the context for which they were being sought. In particular, our goal in the case study was primarily to understand whether the methodology held promise, and secondarily to offer suggestions to the organization for improvement. Thus, if we could show that our method uncovers facts that are well known from established methods to be generally true, then a certain degree of adequacy has been demonstrated. This is clearly the case for our finding of the significant role that the individual developer plays in the process. If we can go farther and reveal correlations of interest in their own right, then a certain degree of utility has been demonstrated. This is the case for our finding of the correlation between process deviations and process success or failure. To our knowledge, this is the first empirical evidence of such a correlation, although belief in this relationship has been the cornerstone of process research for years.

Our results have already led to changes in the process employed by the organization we studied. Those changes have yielded a reduction in unsuccessful processes by approximately 50%, and this was accomplished with changes based only on one of the results, namely by implementing a trigger based on the delay.

The next two sections expand on this brief overview.

## 3 Foundations for the Case Study

This section details how we conducted the case study demonstrating our methodology. In particular, we describe the software process under study, our methods for selecting, collecting, and combining data from the various sources, and our methods for analyzing the data. Although this exploratory study is not an experiment (it is a case study), the credibility of the results and the ability to interpret them is greatly improved using a rigorous empirical framework [12]. Therefore, we describe our study below similar to the way we would describe an experiment.

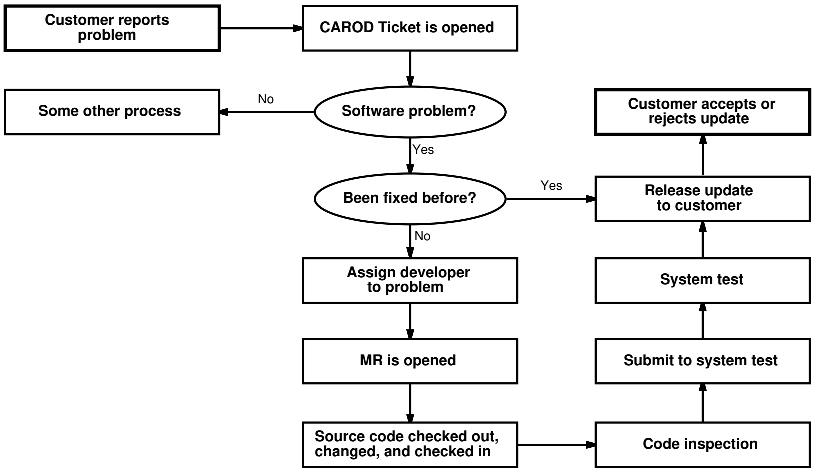



**Figure 1: Basic structure of the process under study.**

### 3.1 Overview of the Subject Process

The process we studied was a customer-initiated software update process for a large telecommunications software product. It is a repetitive process responsible for identifying and solving customer-reported software problems. The prescribed steps in the process are depicted informally and at a high level in Figure 1. We are interested only in those instances of the process that involved making actual changes to the software. Data about the other instances were ignored.

Any problem in the field that causes the customer to call for assistance is recorded in a customer assistance database and identified by a so-called CAROD ticket.[1] Most reports are not specifically software problems, so they can be resolved by performing some other simple process, not part of our study, such as supplying the customer with the documentation they need to solve their own problem or helping them with some confusion about the configuration of their system.

Some number of customer reports, however, are identified as problems in the software. If there already exists a fix for the problem, then it is released to the customer as a software update. If not, then the problem is assigned to a specific developer who assumes responsibility for generating a fix. We use this assignment to indicate an instance of the process to study. Performing the fix involves opening a modification request (MR) for that fix, employing the source code control system to gain access to the code, and subjecting the changed code to various levels of quality assurance.

Once a fix is completed, it is released to the customer as a software update. When a customer applies the fix, they may find that the fix does not in fact solve their original problem. We consider this to be a failure in the process; some unknown mechanism was at work to introduce a defect into

[1]CAROD is the name of the customer request database, and each entry is called a *ticket*.

the software. For our purposes, the accept/reject judgment by the customer terminates an instance of the process. Of course, the organization will act to resolve the problem, and in the end most such rejected fixes are eventually corrected and accepted. But what is important for our study is that the first attempt to fix the problem was not successful.

### 3.2 Sources of Data

As mentioned above, the approach taken in this study was one of analyzing process data previously collected by the organization. Therefore, we could not specify the data we wanted; rather, we could only view the process through the data that existed. The danger in such an approach is that the data only reflect what the organization felt was important. Parts of the process for which no data are collected will be invisible to our analyses. On the other hand, no additional data collection costs are required of the organization and we can examine large amounts of data in a non-intrusive way. While certainly not true of all processes, in our subject process we were able to find enough data of high quality for valid statistical analysis.

There are many kinds of data we could examine, but we chose to look at *event* data [3, 15] because they neatly characterize the dynamic behavior of the process in terms of the sequencing of its major activities. The event data come from several sources. The customer database gives us events concerning the interaction with the customer, including the opening of a CAROD ticket, each update to the status of the ticket, and when the problem was solved, the fix delivered, and the ticket closed. The source code control system gives us every check-in event associated with changes to individual source files. Each process instance is tracked by an MR number, and this tracking database gives us the events of opening an MR, assigning an MR to a developer, accepting an MR by the developer, generating a test plan, submitting an MR solution to system test, having an MR accepted by system test, and eventually closing an MR. Lastly, a database of inspection information gives us the date of the code inspection and its result. All but the last source of event data involved automatic collection through tools; inspection dates and results were recorded manually. It is our experience that there is often at least one valuable data source that exists only on paper.

By merging the events for a particular instance of the process from each of the data sources, we were able to create an *event stream* that represented the sequence of activities occurring in a process execution. We collected 159 such event streams. The event streams were then separated into two populations: one for which the customer accepted the fix (141) and one for which the customer rejected the fix (18). Because there is no source of data that directly records the presence of defects in a released software update, this partitioning of the fixes had to serve as our defect metric.

### 3.3 Threats to Validity

As with any empirical study, validity concerns must be explicitly addressed. Here we discuss threats to the construct, internal, and external validity of our results. We use the definitions of validity given by Judd, Smith, and Kidder [8].

Construct validity is concerned with how well the metrics used in the study faithfully and successfully reflect real-world attributes and values. In this study, we are using the customer's acceptance or rejection of a fix as the metric for the success or failure of the process; it is on this basis that we separate the populations. But one could imagine that there would be other reasons for a customer to reject a fix, not necessarily related to whether or not the developer fixed the problem as they understood it, such as simply deciding their need for the fix did not warrant upgrading their system. This metric, however, was the closest we could come to a direct measure of success and failure.

In contrast to the success/failure metric, most of the other metrics that we used measure attributes directly, such as the number of source lines or the elapsed time of portions of the process. These direct measures are only threatened by the possibility of false or inaccurate data. As mentioned above, most of the data were automatically collected, so inaccuracies are unlikely. During our assembly of the data, and in interacting with those providing the data, there was no indication that people were engaged in purposely falsifying the data; since the data were not particularly used before our study, there was not even a motivation for them to do so.

A threat to construct validity is the danger of inferring some process feature from a direct measure, where the feature may not quite be truly represented. For example, the elapsed time of the process is measured directly, but this does not necessarily mean that more effort was spent in the longer process executions. Any significant results we obtain can only suggest some set of process features that may be involved, but one must keep in mind the limitations of the metrics.

The remaining set of metrics are those for measuring how closely the process models are followed, using the validation metrics discussed in Section 4.2.1. The application of these metrics are to date largely untested, and this study is part of an evaluation of whether they do measure something useful. However, they are based on widely-used methods for measuring differences in similar types of data, so there is good reason to expect that the measurements are accurate.

Internal validity is concerned with how well an experimental design allows for conclusions of causality among the constructs under study. Conclusions about causality come from being able to control the experimental setting and randomizing independent variable assignments. Since ours is a historical study examining processes that already occurred, we cannot randomize variables and so cannot conclude causality from any statistically significant measures that we might obtain. This does not mean, however, that we cannot learn anything from the results.

External validity is concerned with how well the study's results can be generalized. Even though our goal is not to present generalizable results, but rather an example of a methodology, it is always useful to evaluate this aspect of any study. On the negative side, the process we studied is more of a maintenance process than a development process, so the results may be biased towards maintenance kinds of processes. It is also fairly small, although the software it manages is very large. On the positive side, this is a real-world industrial process that is repeatable in its execution. It is also a process that is ubiquitous in industry; almost all organizations have a problem reporting and fault fixing process. Thus, while the results probably do not generalize to all processes, they are likely to shed light on many processes that are in use and important to industry.

### 3.4 Methods of Analysis

With the separation of populations based on the acceptance or rejection of a fix, we then performed analyses on a variety of metrics in order to discover process characteristics that correlate with our separation metric. Our analyses centered on performing statistical significance tests for each metric that was calculated.

Most of the analyses were performed using metrics whose values are numeric. For those metrics, we used the Mann-Whitney significance test [7, Chapter 15], which does not assume an underlying distribution of the data but is still nearly as powerful as standard significance tests that do assume a distribution.[2] The premise behind this test is that if there is no difference between the two populations (the null hypothesis), then when the data values from the two populations are merged and sorted, each population should be distributed approximately the same in the merged ranking. For this reason, it is also called the Wilcoxon Rank-Sum test.

Other metrics we use are two-valued, such as *"an event in the event stream is either* matched *or* deleted *when it is validated with respect to a model"*. These metrics produce binomially distributed populations, and imply a test over the population proportions, with the null hypothesis being that true proportions in the populations are the same. The standard significance test is

$$Z = \frac{p_1 - p_2}{\sqrt{pq(\frac{1}{m} + \frac{1}{n})}}$$

where $m$ and $p_1$ are the size and proportion of one population, and $n$ and $p_2$ are the size and proportion of the other population. $p$ and $q$ are defined as

$$p = p_1 \frac{m}{m+n} + p_2 \frac{n}{m+n} \quad , \quad q = 1 - p$$

that is, $p$ is the weighted sum of the two proportions and $q$ is its inverse.

For each application of the Wilcoxon Rank-Sum (W) and the proportional (Z) significance tests, the two-tailed p-value is calculated. A p-value is a standard value that directly represents the chance that the null hypothesis is rejected when in fact it is true—that is, one mistakenly concludes that there is a significant relationship when there is not one. Thus, one does not need to know what values of Z and W provide a given level of significance, one only need look at the derived p-value.

An issue that comes up when a set of statistical tests are done is that of obtaining "significant" results purely by chance—that is, the result is not really significant, but happens to lie in that small range indicated by the p-value as the likelihood of rejecting the null hypothesis when it actually is true. This likelihood increases as the number of tests increase. For example, if one performed five independent tests at the 95% confidence level, the chance of at least one of those tests showing significance purely by chance would be

$$1 - 0.95^5 = 0.226$$

[2]Plotting the distribution of data is an effective way of quickly determining whether one can assume a type of distribution. Each metric we used has a distribution plot shown in Appendix B.

or about 23%.[3]

A statistical result called the Bonferroni inequality allows us to compute a conservative upper bound on the overall significance level of a set of tests [13]. This inequality deduces that the joint significance level of $N$ tests, each performed at an $\alpha$ significance level, is at least $N\alpha$. Thus, to ensure a significance level $p$ for some set of $N$ tests, each test should be evaluated at the $p/N$ significance level. We will call $p$ the *set-p-value* and $p/N$ the *test-p-value*.

Since we are exploring data for results and not trying to establish extremely confident experimental relations, and since the above relation can be very conservative, we will accept set-p-values of 0.15 as indicating a probable relation, and even look at set-p-values of 0.2 as likely indicating a relation.

Finally, one metric we use is a ratio, but the independent variable is nominal, so a significance equation cannot be used. For this metric, we estimate the standard deviation for each ratio using $\sqrt{p(1-p)/t}$, where $p$ is the proportion, and $t$ is the total. Plotting this data then, reveals non-overlapping error bars, which are significant at about the 0.1 level.[4]

Metrics that significantly differentiate the populations were then examined and interpreted. In addition to presenting the p-values in our results, we also present the means and standard deviations, or proportions, of these metrics. This, along with the plot of each metric's distribution in Appendix B, provides an understanding of the general range and makeup of the data, and thus the process being studied.

## 4 Results

We now present the results of our analyses on the subject data. We separate the case study into two parts: the first part analyzes fairly simple aggregate metrics, while the second part uses tools to measure how the behavior of the process executions correspond to a process model representing the prescribed process.

### 4.1 Part One: Simple Aggregate Metrics

We calculated several simple aggregate metrics and measured their statistical significance in separating the accepted fix population from the rejected fix population. Definitions for the metrics are given in Table 1. For each of the metrics, a statistical test was performed to determine if the populations were significantly different for that metric. The results for the first six metrics are shown in Table 2. The results for *developer*, which uses a nominal scale, are shown in Figure 2. For the set of six tests in the table, at a set-p-value of 0.15, a test-p-value would have to be 0.0214. Five

[3]The value $0.95^5$ is the probability that all 5 tests fall in the 95% region. Subtracting this from 1 gives us the probability of at least one test falling outside of this region. This assumes that all tests are completely independent.

[4]Since the error bars visually represent the standard deviations, then if the error bars are exactly touching ends, the two proportions are two standard deviations apart. With ideal conditions (equal numbers of observations and equal standard deviations), this reduces to a Z test statistic of $\sqrt{2}$, which has a single-tailed significance of 0.08. Thus 0.1 is a good rule of thumb for real data.

| | |
|---|---|
| *ncsl* | Number of source lines of the fix, including new, changed, and removed lines. This is calculated directly from the source code control system. |
| *nfiles* | Number of source files modified for the fix. This is calculated from the source code control system. |
| *nevents* | Total number of events for each process execution. This represents a simplistic count of the number of steps executed in a particular execution of the process. |
| *ctime* | Total time, in days, from the customer ticket open to close. This is the total elapsed time of the process execution. |
| *dtime* | Delay time, in days, from customer ticket open to MR open. This is the interval between the time a problem is reported and the time a developer begins to fix the problem. |
| *mtime* | Total time, in days, from the MR open to close. This is the total elapsed time of the development subprocess. |
| *developer* | The developer who performed the fix, as recorded in the MR database. |

**Table 1: Definitions of the simple aggregate metrics.**

| | P-value | Sig Test | Accept Pop.(N=141) | | Reject Pop.(N=18) | |
|---|---|---|---|---|---|---|
| Measure | (2-tailed) | (W) | Mean | Std Dev | Mean | Std Dev |
| ncsl | 0.230 | 1.20 | 217.22 | 554.31 | 166.22 | 275.62 |
| nfiles | 0.317 | 1.00 | 2.75 | 4.00 | 2.94 | 2.67 |
| nevents | 0.250 | 1.15 | 30.34 | 18.55 | 35.22 | 16.28 |
| ctime | 0.187 | 1.32 | 165.94 | 141.10 | 180.83 | 89.91 |
| dtime | 0.002 | 3.13 | 18.29 | 31.22 | 68.17 | 75.41 |
| mtime | 0.719 | -0.36 | 96.89 | 110.76 | 87.94 | 85.56 |

**Table 2: Values of the simple aggregate metrics.**

of the metrics were not statistically significant in separating the populations. We now examine the two metrics that were significant.

The first significant metric is *dtime*, the delay between the time a customer reports a problem and the time a developer starts working on the problem. This correlation is quite interesting, and could have several explanations. One explanation might be that the understanding of the problem degrades during the delay, leading to a wrong or vague problem definition being handed to the developer, and thus the developer encounters a harder time fixing the problem. Another might be that it takes longer for customer support, working with the customer, to understand the problem in enough detail to relay it the developer. This could be due to the fact that the problem is simply a difficult one to fix, and thus less likely to be fixed correctly. In either case, this result would warrant a closer examination of the relationship.

The other significant metric is *developer*, the developer working on the fix. For each developer, we calculated the ratio of rejected fixes to total fixes as a measure of that developer's failure

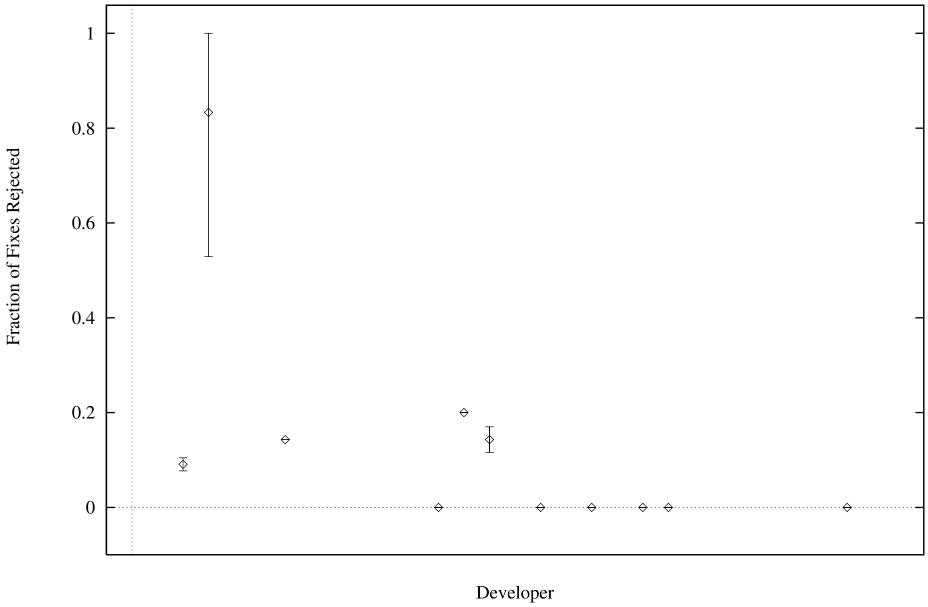



**Figure 2: Fraction of rejected fixes per developer. Only those developers who performed four or more fixes are shown. Non-overlapping error bars are significant at approximately the 0.1 level.**

rate. Figure 2 shows these ratios, with error bars estimated according to the method described in Section 3.4. Since non-overlapping error bars are significant at about the 0.1 level, *developer* is clearly a significant metric.

That the success of a change depends on the person making the change is already a well-known result. Duplicating that result here gives us evidence to believe that our data and methods are producing meaningful results, but it is not surprising or enlightening by itself.

One possible explanation for why certain developers have more fixes rejected than others is that the fixes are assigned to developers who have an area of expertise that matches the suspected problem. Some developers may be working on rather simple peripheral code, while others may be engrossed in the internals of a large subsystem that is difficult to change.

A further examination of the event streams is shown in Table 3, where each event type is counted for the number of times it occurs in each process execution.[5] In this table, there are 22 event types that have significance tests, so an individual test would need a p-value of 0.0068 to indicate significance at the 0.15 level for the whole set, or 0.0091 for an overall 0.2 significance level. One can see that two event types have significantly different average counts between the

[5]Definitions for the event types in the subject process are given in Appendix A.

| Measure | P-value (2-tailed) | Sig Test (W) | Accept Pop. N | Accept Pop. Mean | Accept Pop. Std Dev | Reject Pop. N | Reject Pop. Mean | Reject Pop. Std Dev |
|---|---|---|---|---|---|---|---|---|
| carod-dupdated | 0.0052 | 2.79 | 141 | 3.04 | 3.42 | 18 | 6.00 | 5.38 |
| carod-abstract | 0.0074 | 2.67 | 141 | 1.18 | 0.58 | 18 | 1.56 | 0.92 |
| carod-custdue | 0.0098 | 2.58 | 141 | 1.16 | 0.61 | 18 | 1.56 | 0.92 |
| carod-create | 0.0106 | 2.55 | 141 | 1.18 | 0.58 | 18 | 1.56 | 0.92 |
| carod-inhouse | 0.0106 | 2.55 | 141 | 1.18 | 0.58 | 18 | 1.56 | 0.92 |
| carod-response | 0.0106 | 2.55 | 141 | 1.18 | 0.58 | 18 | 1.56 | 0.92 |
| carod-update | 0.0106 | 2.55 | 141 | 1.18 | 0.58 | 18 | 1.56 | 0.92 |
| carod-dcreated | 0.080 | 1.75 | 141 | 1.26 | 0.63 | 18 | 1.61 | 0.98 |
| mr-acptsys | 0.099 | -1.65 | 141 | 0.86 | 0.59 | 18 | 0.61 | 0.50 |
| mr-acceptmr | 0.13 | 1.53 | 141 | 0.96 | 0.51 | 18 | 1.11 | 0.32 |
| mr-smitsys | 0.17 | 1.37 | 141 | 1.20 | 0.74 | 18 | 1.39 | 0.61 |
| mr-createmymr | 0.18 | 1.35 | 141 | 0.74 | 0.44 | 18 | 0.89 | 0.32 |
| carod-closed | 0.23 | -1.21 | 141 | 0.80 | 0.76 | 18 | 0.78 | 1.22 |
| mr-asnmymr | 0.24 | 1.17 | 141 | 1.01 | 0.71 | 18 | 1.11 | 0.32 |
| mr-smit | 0.29 | 1.06 | 141 | 1.23 | 0.76 | 18 | 1.39 | 0.61 |
| carod-delivered | 0.39 | -0.87 | 141 | 0.81 | 0.76 | 18 | 0.83 | 1.20 |
| code-checkin | 0.45 | 0.75 | 141 | 5.96 | 15.63 | 18 | 4.44 | 5.64 |
| mr-assign | 0.52 | 0.65 | 141 | 1.11 | 0.77 | 18 | 1.11 | 0.32 |
| code-inspect | 0.61 | -0.51 | 141 | 0.93 | 0.31 | 18 | 0.89 | 0.32 |
| carod-solved | 0.85 | -0.19 | 141 | 0.91 | 0.78 | 18 | 1.06 | 1.21 |
| mr-test-plan | 0.89 | 0.13 | 141 | 0.94 | 0.48 | 18 | 1.00 | 0.59 |
| mr-bwmbuilt | 0.91 | -0.12 | 141 | 0.82 | 0.56 | 18 | 0.83 | 0.79 |
| mr-rejectbwm | – | – | 141 | 0.05 | 0.22 | 18 | 0.22 | 0.43 |
| mr-asndue | – | – | 141 | 0.00 | 0.00 | 18 | 0.06 | 0.24 |
| mr-rejectmr | – | – | 141 | 0.04 | 0.20 | 18 | 0.11 | 0.32 |
| mr-create | – | – | 141 | 0.21 | 0.41 | 18 | 0.11 | 0.32 |
| mr-asnprior | – | – | 141 | 0.01 | 0.12 | 18 | 0.00 | 0.00 |
| mr-featchg | – | – | 141 | 0.09 | 0.30 | 18 | 0.11 | 0.32 |
| mr-killmdmr | – | – | 141 | 0.10 | 0.36 | 18 | 0.11 | 0.32 |
| mr-asntarg | – | – | 141 | 0.01 | 0.17 | 18 | 0.00 | 0.00 |
| mr-defer | – | – | 141 | 0.01 | 0.08 | 18 | 0.00 | 0.00 |
| mr-integrate | – | – | 141 | 0.01 | 0.08 | 18 | 0.00 | 0.00 |
| mr-killmrnm | – | – | 141 | 0.01 | 0.08 | 18 | 0.00 | 0.00 |
| mr-killok | – | – | 141 | 0.01 | 0.08 | 18 | 0.00 | 0.00 |
| mr-killmr | – | – | 141 | 0.06 | 0.32 | 18 | 0.06 | 0.24 |
| mr-killmymr | – | – | 141 | 0.06 | 0.27 | 18 | 0.06 | 0.24 |

**Table 3: Event type counts per event stream. All events above the first horizontal line are significant, and those between the lines are possibly significant. Those without P-values do not have enough occurrences to reliably calculate statistics.**

populations, while several others are close enough to warrant consideration.

One of the most significant is *carod-dupdated*, which is an instance of someone updating the description of the customer's problem. The greater value of the rejected fix population would seem to imply that it takes more effort to figure out the problem. This could be due to the problem being more difficult, or that communication with the customer is breaking down and requires more iterations.

The other significant CAROD event type (and those that are almost significant) indicate that there are slightly more CAROD records per MR in the rejected fix population, as can be seen by the mean values of the event counts. This occurs when more than one customer reports the problem while it is open and being fixed. If the fix was already performed for the first customer, then it would be a simple software update for the next; if the fix is not yet completed, and another customer reports the same problem, then both of the customer CAROD tickets are associated with the same MR number and thus the same fix process. This could cause more customer rejections because, for example, the fix might be directed more towards the first customer, and not quite fix the problem reported by the second customer. This could also happen if the association was erroneous—that is, the person who associated the two problems with the same fix may have been wrong. A third interpretation could be that more CAROD tickets for a given problem result from the code being exercised more in the field, leading more customers to encounter the same problem in the code.

In general, the data reveal the presence of one or more defect-producing mechanisms at work, and that the mechanisms are somehow associated with the delay in beginning a fix and the developer performing the fix. The results do not identify the mechanisms themselves, but rather point the organization in possible directions to look for improvements.

## 4.2 Part Two: Measuring Process Behavior

Our second use of the data we collected was in measuring how well the process executions followed the prescribed process. This organization had spent much effort in producing a process description. Showing that it was relevant, and relating specific activities to the product outcome, were both desirable. Below we present the tools and metrics we used to do this, and then present the results that were derived.

### 4.2.1 Comparing Executing and Prescribed Processes

Process validation measures the behavioral correspondence between an executing process and a formal model of that process, by cataloging both the missed activities and the extra activities A full description of the metrics that we review here, and of the theory behind them, can be found elsewhere [5, 6].

Behavioral correspondence is a critical measurement for our case study because it can help to relate process successes and failures to deviations from the prescribed process. For example, it could be that developers must violate the organization's established practices in order to successfully solve a problem in a timely manner. Conversely, it could be that strict adherence to the prescribed process is the most reliable way to achieve a successful resolution of a customer problem. Also interesting would be the result that there is no correlation between deviations and success or failure; in that case it would seem that the documented process might not be capturing the critical aspects of the organization's activities.

The process validation method we employ uses a finite-state machine representation of the process model together with the event stream representation of a process execution. The machine specifies the "language" of event streams that it will accept—that is, the set of valid sequences of events, while the event stream can be viewed as a candidate string in that language. The measurement techniques are based on string difference metrics, which count the number of insertions and deletions of "characters" (i.e., events) needed to transform the event stream string into a string acceptable to the machine. Event insertion operations are interpreted as activities that were missed in the executing process, while event deletion operations are interpreted as extra activities performed by the process that were not called for by the model.

From the count of insertion and deletion operations, several metrics are calculated. The two primary metrics are *SSD*, the Simple Stream Distance metric, and *NSD*, the Non-linear Stream Distance metric. SSD calculates a simple count of the number of insertions and deletions, then normalizes this value to the length of the execution event stream. NSD enhances SSD to take into account *blocks* of insertions and deletions. A block represents a single, longer deviation between the executing process and the process model, so rather than just counting single event deviations, NSD calculates its measurement by applying a parameterized exponential function to the block lengths. Thus, runs of deviating events carry more weight than single event deviations.

In addition to the SSD and NSD metrics, we can look at per-event-type and per-model-state information. For each type of event (e.g., a check-in event of the source code control system), we record the total number of matches, insertions, and deletions for each process execution. This allows us to calculate, for each event type, the number of events of that type that occurred correctly according to the model (matches), the number that were missed (insertions), and the number that were extra or at the wrong time (deletions). Similarly, for each state in the model, we also record the total number of event matches, insertions, and deletions that occur at that state in the model. These counts are then combined into meaningful metrics, as follows.

1. *matches/(matches+deletions)* gives, for each event type, the proportion of event occurrences in the event stream that are matched by the model;

2. *matches/(matches+insertions)* gives, for each event type, the proportion of events predicted by the model that are matched in the event stream;

3. *matches/(matches+deletions)* gives, for each state in the model, the proportion of event occurrences in the event stream that are matched by the model; and

4. *matches/(matches+insertions)* gives, for each state of the model, the proportion of events predicted by the model that are matched in the event stream.

In effect, the first and third metrics represent the proportion of matches from the perspective of the event stream, while the second and fourth metrics represent the proportion of matches from the perspective of the model.

| Measure | P-value (2-tailed) | Sig Test (W) | Accept Pop.(N=141) Mean | Accept Pop.(N=141) Std Dev | Reject Pop.(N=18) Mean | Reject Pop.(N=18) Std Dev |
|---|---|---|---|---|---|---|
| SSD | 0.19 | 1.32 | 0.65 | 0.24 | 0.72 | 0.21 |
| NSD | 0.0064 | 2.72 | 29.75 | 72.59 | 88.58 | 111.67 |
| Matches | 0.62 | -0.50 | 17.93 | 15.78 | 16.11 | 5.28 |
| Insertions | 0.80 | -0.26 | 6.96 | 2.86 | 7.33 | 3.38 |
| Deletions | 0.09 | 1.72 | 10.77 | 7.32 | 14.56 | 8.99 |
| Insertion blocks | 0.22 | -1.24 | 3.30 | 1.24 | 3.17 | 0.99 |
| Deletion blocks | 0.45 | 0.75 | 4.88 | 2.12 | 5.22 | 1.96 |

**Table 4: Correspondence metrics for the model of Figure 3.**

Further, the first and second metrics provide a measure of on what event types the process and model may be differing, while the third and fourth metrics measure where in the model the process and model may be differing.

With these four metrics, we can understand deviations (missed or extra activities) both from the perspective of the model and of the process execution, and can localize the deviations both to areas in the model and to specific types of activities.

### 4.2.2 Correspondence Results

Using the validation methods described above, we measure the correspondence of the process executions, as represented by event streams, to the organization's prescribed process, as represented by a finite-state machine model of the process.

The model, shown in Figure 3, is based on the organization's paper documentation of the process, interviews with several members of the organization, and minimal inspection of the data. This model is meant to represent the idealized view of the prescribed process. The paper documentation gave a high-level structure, and specified MR event orderings and code check-in and inspection orderings. Talking with members of the customer assistance team, and looking at the CAROD data both contributed to the areas in the model where CAROD events are occurring, with the data inspection only used to resolve some event sequencing that was due to how the collection and merging of the data was done.

Using this model and the 159 event streams, we calculated the SSD and NSD metrics. We also took simple counts of the number of matches, insertions, and deletions of events, as well as of insertion and deletion blocks. Table 4 gives the results of our metric calculations for both the accepted fix and rejected fix populations. For these seven tests, the test-p-value is 0.0214 for a set-p-value of 0.15.

The first thing to notice is that, based on the number of event matches and the average number of events in an event stream (from Table 3), only about 45-55% of the total events are matched by the model. Evidently, the model in Figure 3 is not good at describing the actual behavior of the

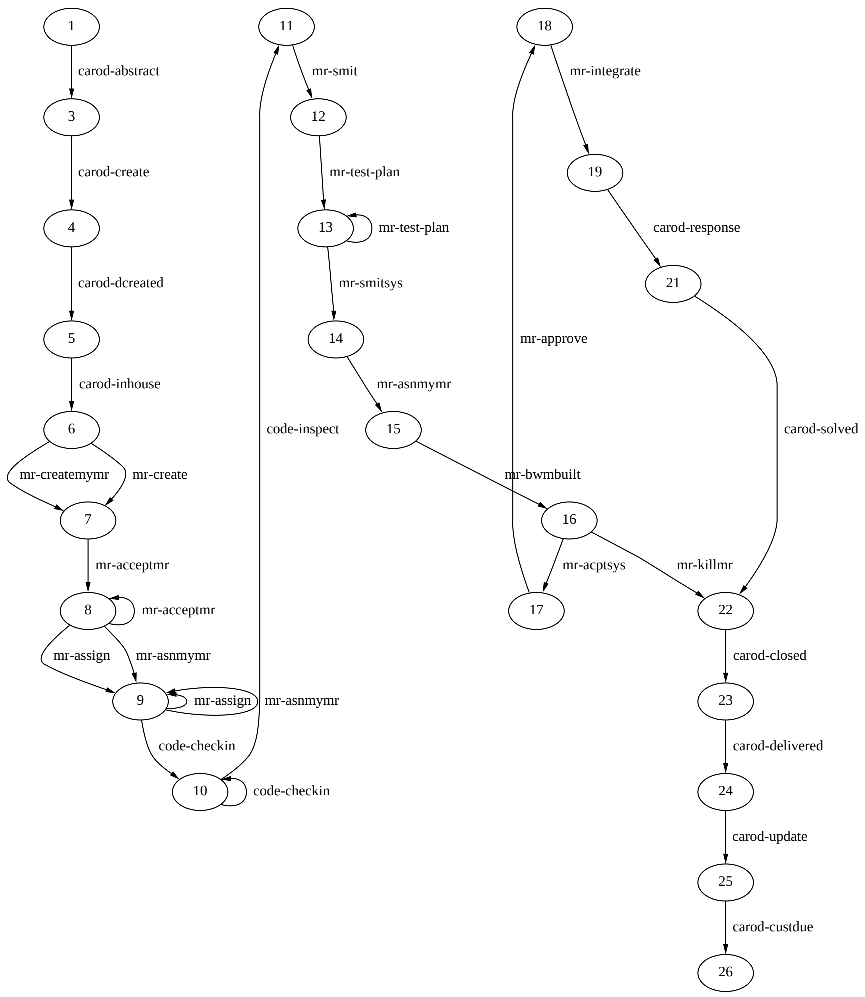



**Figure 3: Finite-state machine model of the subject process.**

| Event Type | P-value (2-tailed) | Sig Test (Z) | Accept Pop. # of Events Occurring | Accept Pop. Proportion Matched | Reject Pop. # of Events Occurring | Reject Pop. Proportion Matched |
|---|---|---|---|---|---|---|
| code-checkin | 0.0106 | 2.55 | 833 | 0.87 | 80 | 0.76 |

**Table 5: Per-event-type metrics for the model of Figure 3. Only those event types that had significant measures are shown. No event types had significantly different predicted vs. matched proportions.**

| Model State | P-value (2-tailed) | Sig Test (Z) | Accept Pop. # of Events Occurring | Accept Pop. Proportion Matched | Reject Pop. # of Events Occurring | Reject Pop. Proportion Matched |
|---|---|---|---|---|---|---|
| 16 | 0.0008 | 3.34 | 176 | 0.54 | 32 | 0.22 |
| 24 | 0.0116 | 2.52 | 183 | 0.63 | 39 | 0.41 |

| Model State | P-value (2-tailed) | Sig Test (Z) | Accept Pop. # of Events Predicted | Accept Pop. Proportion Matched | Reject Pop. # of Events Predicted | Reject Pop. Proportion Matched |
|---|---|---|---|---|---|---|
| 16 | 0.0046 | 2.82 | 132 | 0.72 | 18 | 0.39 |

**Table 6: Per-model-state metrics for the model of Figure 3. Only those model states that had significant measures are shown.**

processes. This is also reflected in the SSD and NSD metrics, which have quite large values.

Next, notice that while the SSD metric fails to differentiate the populations, the NSD metric succeeds. In particular, the deviation from the prescribed process is significantly greater for the rejected fix population than it is for the accepted fix population. If we look at the constituents of the NSD metric, none of them show a significant difference by themselves. The NSD metric focuses attention on blocks of insertions and deletions, so the fact that the NSD metric detects a strongly significant difference in the populations indicates that there were larger areas of deviation from the process model in the rejected fix population than in the accepted fix population.

At this level, then, we see that the model describes only about half of the behavior, and that there is some significant difference in how the populations relate to the model. We can see the difference in more depth by looking at the proportions of matched events, as described in Section 4.2.1). They are shown per event type and per model state in tables 5 and 6, respectively. In these tables, only the event types and model states that did in fact have significant differences are shown. For all of these tables, the number of tests that make up a single set for computing p-values for the whole set is 18. Thus, for a set-p-value of 0.2, the test-p-value is 0.011. For a set-p-value of 0.15, the test-p-value is 0.008.

The first significant difference is the check-in event, which the model accepts near state 10. The tables show that the check-in event is matched more often in the accepted fix population, so we can conclude that check-in occurs with less regularity in the rejected fix population. Because the process documentation explicitly requests that code be checked in before it is inspected, this result suggests both that rejected fixes may not be following this rule as rigorously as they should be and that the rule is good—it correlates with successful instances of the process.

Another result shows state 16 having significantly fewer matches of predicted events in the rejected fix population, and also having fewer matches of events that occurred. This may point to problems in controlling the system test or the delivery of the fix.

Finally, two results that might be indicating something significant is the carod-closed event and state 24. They are both somewhat weak results, but together possibly indicate some differences between the populations near the end of the model. This could simply be that the rejected fix processes are already shutting down at this point, and do not follow the model through to the end.

Overall, the validation results show that the rejected fix processes do not follow the model as closely as the accepted fix processes, and that specific areas in the model had significant behavior differences between the accepted and rejected fix populations. Thus, the model seems to capture some important, and good, process behavior. On the other hand, the mean values of the SSD and NSD metrics for both populations indicate that there is much behavior that is not accounted for in the model. Perhaps the model is out of date with how the process behaves; or the process is simply highly adaptable and variable, and the model too abstract to capture these variations and "exceptions". In either case, it would seem that with some effort, a better model that more accurately captured ideal process behavior could be constructed.

### 4.3 Summary of Results

Our analysis of the data has revealed several interesting things about the subject organization and its process.

- The documented model of the process does not adequately capture what the members of the organization record that they do to successfully carry out the process.
- There is a large variance in the structure of activities among individual process executions.
- The success of the process is highly correlated with the person responsible for making the required fix.
- A long delay in beginning a fix is correlated with a likely failure in the process.
- Missed or irregular code check-in is correlated with a likely failure in the process.

While we cannot point to specific defect-producing mechanisms from these results, the organization can use this information to focus their process improvement efforts. For example, they can better document their process in order to communicate good practices and potential pitfalls to new members; they can establish an alarm system that monitors the delay between accepting a problem report and beginning the corrective action, and have managers automatically notified when the delay exceeds some significant threshold; and they can consider better ways to structure access to code so that members of the organization are motivated to use the check-in mechanism. In fact, the organization has already begun to make improvements based on our study.

## 5 Conclusion

Empirical process improvement relies on having data collected not only from the product but from the process as well. In many cases, existing repositories of data can contribute greatly to an empirical understanding of the process. This paper presented a methodology for exploratory empirical studies that takes advantage of such readily available data.

We gave evidence of the adequacy and utility of the methodology by presenting a case study that showed that specific process features could be related to product outcome, leading to a better understanding of the process in enough detail that specific recommendations for improvement could be made. Indeed, the development organization has already instituted changes based on this study, including tracking the delay time of a CAROD ticket, and using historical data on other repetitive processes. In fact, they now post a chart in the building entrance tracking quality metrics over the projects' lifetimes. On this specific project, their reject rate has been reduced by approximately 50% from the time of our study.

The success of this study shows that it is feasible to perform a process study by using historical data that are readily available—that is, already collected manually or automatically during the regular course of the process for purposes other than the study at hand—without the addition of new and specific instrumentation.

The main disadvantage of this approach to data gathering is that the study effectively becomes a naturally occurring experiment whose subject cannot be statistically controlled or varied in order to test hypotheses. Thus, substantive conclusions about causality are limited.

Nevertheless, there are two benefits of this approach that should not be undervalued. First, large amounts of data can be collected without disturbing the process and the people involved. This is particularly important for researchers who are interested in studying real-world organizations. Second, using previously collected, readily available data is much less costly than instrumenting to collect new data. The results from a study such as this can then be used to focus subsequent instrumentation, providing a cost-effective path to data-driven process improvement. This two-step approach overcomes the problem of convincing an organization to implement a potentially costly, general instrumentation of their process, and provides a solid foundation from which to argue its benefits.

A unique aspect of this study is its use of a formal process model in conjunction with process data to help analyze the process. Demonstrating that process models are useful is critical to the acceptance of process research results in industry, which to date has been slower than some have expected. Our study explored the relationships among process models, executions, and outcomes for a real-world industrial process. Using both simple aggregation metrics and sophisticated process data analysis tools, we were able to show correlations between deviations from the prescribed process and the presence of defects in the product. Moreover, the results were gained at a sufficiently detailed level to point to relatively specific places to improve the process. We hope that these results, and further studies, will encourage the application of formal process models to everyday software process improvement.

**Acknowledgements** The work described here was performed in part while J.E. Cook was a visiting researcher at AT&T Bell Laboratories. We would like to thank the Autoplex organization of Lucent Technologies (formerly AT&T) for access to their projects and data, and David Christensen, Michael Carper, Carol Ferguson, and the many others who were patient with our questions. We also thank Jim Herbsleb and the reviewers for the many comments that improved this paper.

## REFERENCES


[1] V.R. Basili and D.M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, 10(6):728–737, 1984.

[2] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, and R. Chillarege. A case study of software process improvement during development. *IEEE Transactions on Software Engineering*, 19(12):1157–1170, December 1993.

[3] M.G. Bradac, D.E. Perry, and L.G. Votta. Prototyping a process monitoring experiment. *IEEE Transactions on Software Engineering*, pages 774–784, October 1994.

[4] L.J. Chmura, A.F. Norcio, and T.J. Wicinski. Evaluating software design process by analyzing change data over time. *IEEE Transactions on Software Engineering*, 16(7):729–739, July 1990.

[5] J.E. Cook and A.L. Wolf. Toward Metrics for Process Validation. In *Proceedings of the Third International Conference on the Software Process*, pages 33–44. IEEE Computer Society, October 1994.

[6] J.E. Cook and A.L. Wolf. Process discovery and validation through event-data analysis. Technical Report CU-CS-817-96, Department of Computer Science, University of Colorado, November 1996.

[7] J.L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, Pacific Grove, California, 3rd edition, 1991.

[8] C.M. Judd, E.R. Smith, and L.H. Kidder. *Research Methods in Social Relations*. Holt, Rinehart and Winston, Inc., Fort Worth, sixth edition, 1991.

[9] A. Kouchakdjian, S. Green, and V.R. Basili. Evaluation of the Cleanroom methodology in the Software Engineering Laboratory. In *Proc. Fourteenth Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, 1989.

[10] R.W. Selby, A.A. Porter, D.C. Schmidt, and J. Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of the 13th International Conference on Software Engineering*, pages 288–298. IEEE Computer Society Press, May 1991.

[11] S.W. Sherer, A. Kouchakdjian, and P.G. Arnold. Experience using Cleanroom software engineering. *IEEE Software*, 13(3), May 1996.

[12] Allan S.Lee. A Scientific Methodology for MIS Case Studies. *MIS Quarterly*, pages 33–50, March 1989.

[13] G.W. Snedecor and W.G. Cochran, editors. *Statistical Methods*. Iowa State University Press, 8th edition, 1989.

[14] L.G. Votta and M.L. Zajac. Design process improvement case study using process waiver data. In *Proceedings of the Fifth European Software Engineering Conference (ESEC'95)*, pages 44–58. Springer-Verlag, September 1995.

[15] A.L. Wolf and D.S. Rosenblum. A Study in Software Process Data Capture and Analysis. In *Proceedings of the Second International Conference on the Software Process*, pages 115–124. IEEE Computer Society, February 1993.

[16] R. Yin. *Case Study Research: Design and Methods*. SAGE Publications, Thousand Oaks, California, 2nd edition, 1994.

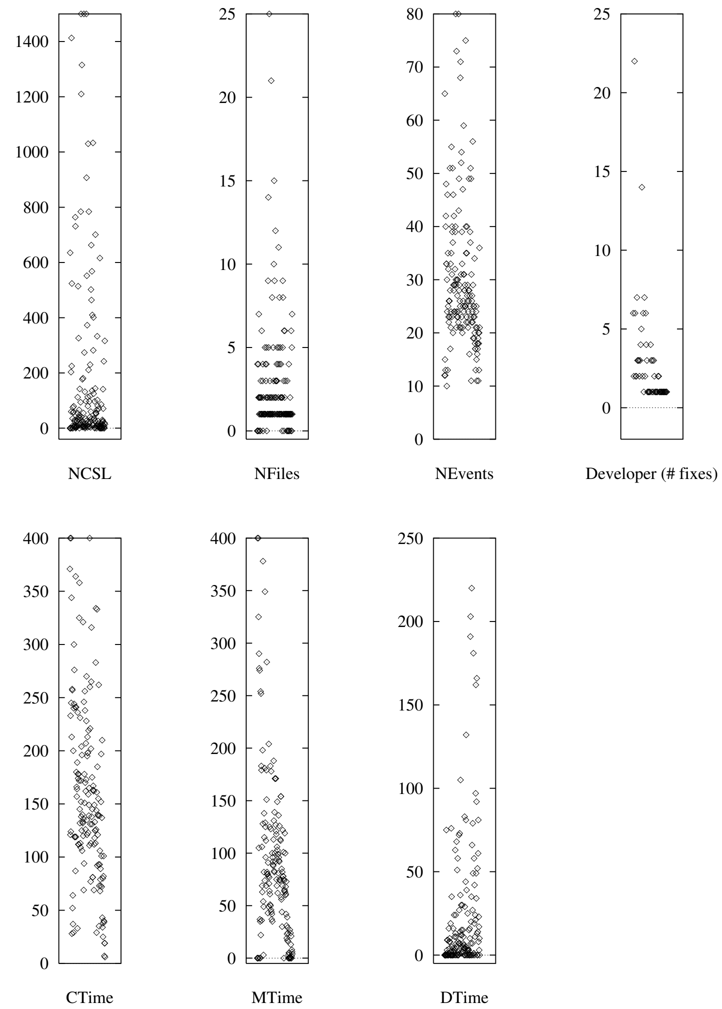
## A Event Type Definitions

The following table lists the event types and their definitions for the subject process.

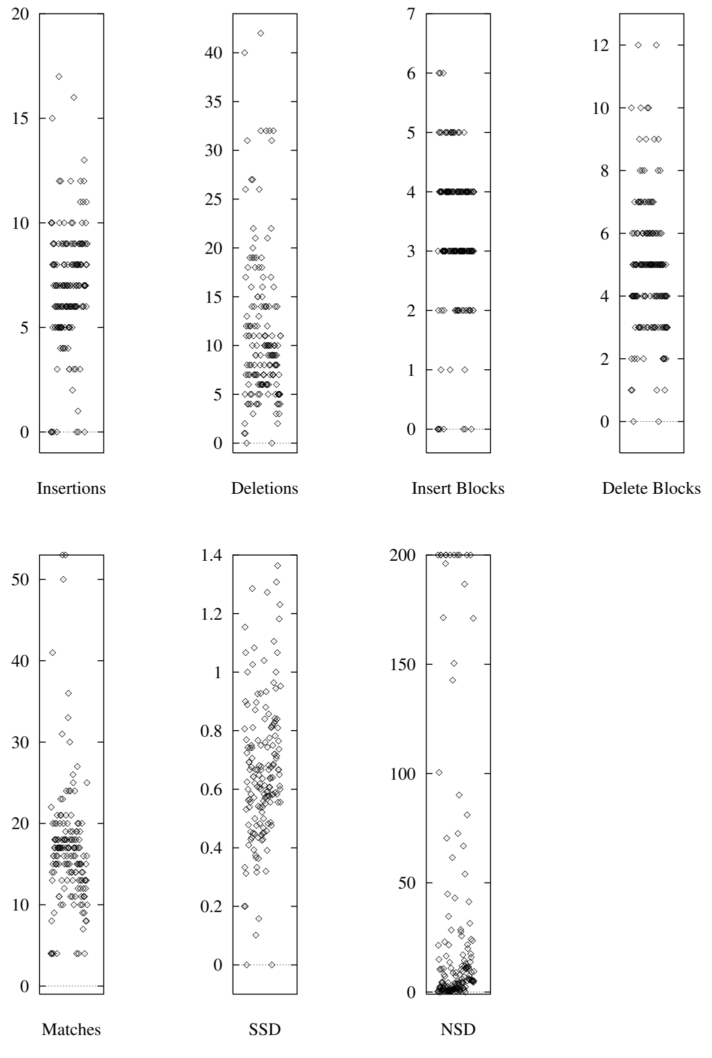
| Event | Definition |
|---|---|
| carod-abstract | create an abstract of the customer's problem |
| carod-closed | close the CAROD ticket (i.e., mark as complete) |
| carod-create | create a CAROD ticket for a customer problem |
| carod-custdue | promised due date for a customer solution |
| carod-dcreated | description of ticket created |
| carod-delivered | customer solution is delivered |
| carod-dupdated | update problem description |
| carod-inhouse | problem brought in-house |
| carod-response | initial response to the customer |
| carod-solved | customer problem has been solved |
| carod-update | update of CAROD ticket |
| code-checkin | source code module check-in |
| code-inspect | source code inspection occurred |
| mr-acceptmr | developer accepts MR assignment |
| mr-acptsys | system test accepts MR for a test build |
| mr-asndue | assign due date to MR |
| mr-asnmymr | assign my MR (to myself) |
| mr-asnprior | assign a priority to an MR |
| mr-asntarg | assign target to MR |
| mr-assign | assign MR to developer |
| mr-bwmbuilt | test build has been built for MR |
| mr-create | create an MR |
| mr-createmymr | create an MR (for myself) |
| mr-defer | defer the MR |
| mr-featchg | MR is a feature change |
| mr-integrate | MR is integrated (into a build) |
| mr-killmr* | kill an MR (various subtypes) |
| mr-rejectbwm | MR is rejected from system build |
| mr-rejectmr | MR is rejected |
| mr-smit | submit MR to system test |
| mr-smitsys | submit MR to system test |
| mr-test-plan | test plan written for MR |

## B Data Characterization

In this appendix we present a distribution plot for each of the metrics. These are provided to give the reader a more intuitive understanding of each metric under test. Each point on a plot is a single data item, with both the accepted fix and rejected fix populations being combined. All of the metrics and their units are explained in Section 4.1.




**Distribution plots of the aggregate metric data.**




**Distribution plots of the correspondence metric data for the model of Figure 3.**