From the Proc. of the 2nd Inter. Conf. on the Software Process, Berlin, Germany, February 25–26, 1993

A Study in Software Process Data Capture and Analysis

Alexander L. Wolf

Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
(alw@cs.colorado.edu)

David S. Rosenblum

Advanced Software Technology Department
AT&T Bell Laboratories
Murray Hill, NJ 07974 USA
(dsr@research.att.com)

Abstract

Process data capture is the activity of obtaining information about an existing software process. Process analysis is the manipulation of that information for purposes of problem identification. Capture and analysis are key elements in any strategy for software process improvement. We have developed a model of the software process that is based on a notion of events characterizing identifiable, instantaneous milestones in a process. We have also developed capture and analysis techniques suited to that model. This paper reports on a study that was undertaken to gain experience with both the model and the capture and analysis techniques. In that study, we captured event data on several actual enactments of the build process of a large, complex software project within AT&T. We entered the captured event data into a database and ran several queries against the data. The queries implement a variety of analyses on the event data by examining relationships among events, such as dependencies and time intervals. The output of the queries are statistical data that can be used to guide the design of process improvements. While the data we collected in the study are incomplete, our initial results demonstrate the viability of this approach to capture and analysis.

1 Introduction

Organizations are paying more and more attention to the processes they use to carry out software development projects. One way to attempt to improve a process is to improve the underlying technology used in that process, such as through faster computing hardware and more sophisticated development tools. Ultimately, however, it becomes necessary to study and improve the dynamic interactions of the process itself, such as the steps that are carried out in the process, the roles and responsibilities assigned to project personnel, the manner and frequency of communication in the process, and so on. In order to improve processes, or even to design new ones, it is necessary to obtain concise, accurate and meaningful information about existing processes that can be used to identify and eliminate problems and to develop and validate improvements. We refer to the activity of obtaining the necessary information as *process data capture*, and we refer to the manipulation of the captured information for problem identification as *process analysis*.

We distinguish two kinds of process analysis, namely *deductive analysis* and *retrospective analysis*. Deductive analysis is concerned with analyzing an abstract specification of a process in some formal logic, with the goal of discovering inconsistencies or other anomalies that would be present in enactments of the process. The choice of formal notation governs what kinds of deductive analysis techniques can be applied to the specified process. A number of formal notations and associated deductive analysis techniques have been designed specifically for process specification and analysis, including APPL/A [16], Interact/Intermediate [10], Marvel/MSL [2] and FUNSOFT nets [8]. In addition, a number of formal notations and associated deductive analysis techniques for describing software systems have been adapted to process specification and analysis, including Petri Nets [1], CSP [6], LOTOS [13] and Statecharts [9].

Retrospective analysis, on the other hand, is concerned with analyzing empirically gathered data from several enactments of a process, with the goal of discovering patterns of anomalous behavior that can be eliminated in future enactments. A "post-mortem" is a typical kind of informal retrospective analysis that is usually performed on a single data point (e.g., the most recently completed process enactment) in hopes of finding egregious problems that can be easily eliminated in the future. We seek a more formal framework within which multiple data points can be analyzed and compared through more powerful and objective analytical techniques.

We hypothesize that process problems ultimately lead to wasted intervals of time. We further hypothesize that the causes of wasted time are best revealed by retrospective analysis of characteristic time intervals. For instance, a long period of idleness between the time a meeting is scheduled and the time it takes place may reveal poor planning for activities that require a long preparation time. Bradac, Perry and Votta have also begun to explore these hypotheses [5].

In order to analyze a process's characteristic time intervals, it is first necessary to capture the relevant data about the significant events of the process, including the times at which those events occur. In deciding what information to capture, it must be remembered that the level of detail at which the process is modeled and captured determines the level of specificity with which process improvements can be prescribed.

Capture techniques typically have been either purely *manual* or purely *automated*. Basili and Weiss describe a methodology for manual, forms-based collection of data for use in evaluating and comparing software development methodologies [4]. Amadeus is a system for automated collection and analysis of process metrics within the Arcadia process-centered environment [14]. Amadeus contains an event-action component for automating the generation and feedback of metrics into the process in response to process events. Yeast is a general event-action system that is being used to automate event capture in a variety of processes in the UNIX® environment [12]. We feel that a *hybrid* approach to capture is necessary because purely automated approaches are inherently biased towards the computerized aspects of processes, while purely manual approaches are inefficient for high volumes of data. (For related views, see Basili and Rombach [3], and Sutton [15].)

In this paper we describe an event-based software process model and techniques for data capture and retrospective analysis that are suited to the model. We also describe our initial results from a study in which we captured and analyzed the build process of a large, complex software project within AT&T. Our event-based model is a general model that can be used in conjunction with a variety of different capture and analysis techniques. The capture technique we developed for our study is a hybrid of manual and automated approaches, while our analysis technique is based on statistical analysis of the time intervals contained in the captured process data. We entered the captured event data into a relational database system, allowing us to implement our analysis objectives as database queries. While the database aids us in carrying out formal analysis, we have also found the data to be suitable for more informal analyses, in particular graphical display for *process visualization*.

In Section 2 we describe in detail our event-based process model. In Section 3 we describe the build process that is the subject of our study. In Section 4 we describe how this process is characterized within our event-based model, and we describe our work in capturing event data from actual enactments of the process. In Section 5 we describe two of the formal analysis queries we developed and the results of those queries. In Section 6 we conclude with a discussion of our plans for future work on capture and analysis techniques for event-based process models.

®UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/OPEN corporation.

2 The Event-Based Model

As mentioned in Section 1, our model of software processes is based on a simple notion of events that occur at specific points in time. In this section we define what an event is in more detail, and we describe a general taxonomy of process events.

2.1 Events and Event Intervals

Figure 1 depicts the fundamental notions of events and event intervals, and their correspondence with process activities. The figure depicts three different activities, which are carried out in parallel by one or more persons or machines as part of a software process. The heavy, solid bars denote the periods of time during which separate instances of the activities occur.

An *event* is an instantaneous happenstance that occurs during an activity at a specific, identifiable point in time; events are depicted in Figure 1 as solid circles. An *event interval* is the period of time between a pair of events. Any arbitrary pair of events can potentially define an interesting event interval. To date, we have found four kinds of event intervals to be particularly useful in process analysis.

The first of these involves the events occurring within a single instance of a particular activity. Two special kinds of events are the beginning and ending

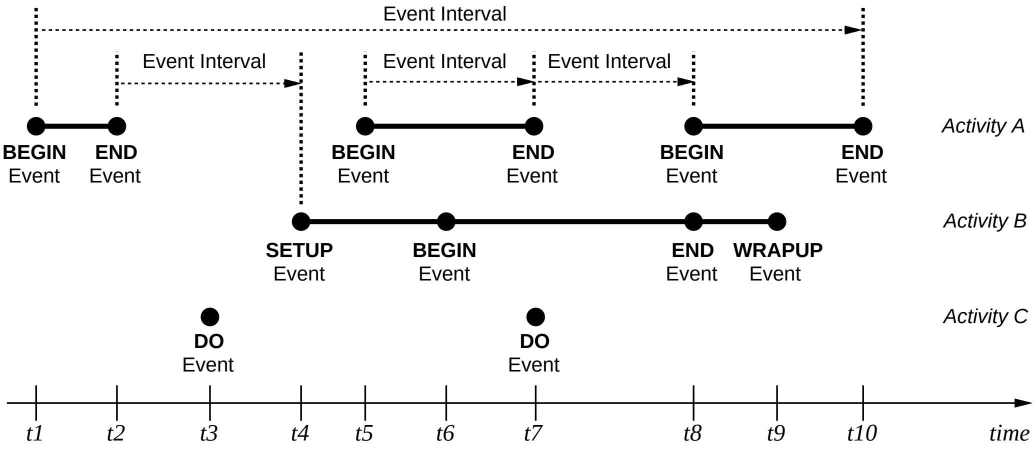

Figure 1: Events and Event Intervals.

events, which we refer to as the BEGIN and END events, respectively. Figure 1 illustrates this kind of interval for the instance of Activity A carried out between the BEGIN and END events at times *t5* and *t7*. The BEGIN/END interval is intended to indicate the period of time during which the main purpose of the activity is being carried out. For some activities, however, this main period may be preceded by an initial setup period and/or followed by a final wrapup period. It is often desirable to keep such setup and wrapup intervals as short as possible. Consider, for example, a meeting. The time between the scheduling of the meeting and the actual beginning of the meeting is the setup period. After the end of the meeting, a record of the meeting is generated and distributed during the wrapup period. In a sense, the meeting activity spans these setup, main, and wrapup time periods. The instance of Activity B in Figure 1 illustrates this kind of activity, with the event interval associated with that activity extending from the SETUP event at time *t4* to the WRAPUP event at time *t9*.

The second kind of event interval is the complement of the event interval characterizing an instance of an activity. In particular, it is the period of time between the last event of one instance of the activity and the first event of the next instance of that same activity. Such event intervals are often indications of idle, and possibly wasted, time. Figure 1 illustrates this kind of interval between the END event of the second instance of Activity A at time *t7* and the BEGIN event of the third instance of Activity A at time *t8*.

The third kind of event interval corresponds to the period of time throughout a sequence of instances of an activity. For instance, an activity that must be repeated until some goal has been met can be characterized by the interval between the first event of the first instance of the activity and the last event of the last instance of that activity; such an interval characterizes the amount of time needed to achieve the goal associated with the activity. Figure 1 illustrates this kind of interval between the BEGIN event of the first instance of Activity A at time *t1* and the END event of the third instance of Activity A at time *t10*.

The fourth kind of event interval is an interval between events of different activities. This kind of interval typically corresponds to the time taken between sequential steps in a process. Figure 1 illustrates this kind of interval between the END event of the first instance of Activity A at time *t2* and the SETUP event of Activity B at time *t4*.

While in our experience most activities involve multiple events, there are some activities that are considered to occur instantaneously and thus have a single event associated with their instances. We refer to the event of an instantaneously occurring activity as a DO event, as illustrated in the figure by the instances of Activity C at times *t3* and *t7*.

2.2 Event Taxonomy

In order to provide a more insightful event-based characterization of some process of interest, and to provide a richer level of detail in the information that is captured about that process, we have designed a general taxonomy of events characterizing the different activities that are carried out in a software process. The taxonomy comprises six categories of events:

1. communication events;
2. automation events;
3. analysis events;
4. work events;
5. workday events; and
6. decision events.

An instance of this taxonomy for a particular process will contain a number of event kinds within each category characterizing the particular activities of that process. Section 4 describes the instantiation of this taxonomy for the build process we studied.

The first category of events is *communication events*. Communication plays a central role in large processes, for it is primarily through coordination of a sizable and sometimes geographically dispersed group of people that large software systems are built. An instance of a communication activity can be characterized by the period of time between its initiation (the BEGIN event) and termination (the END event). Several kinds of communication may take place before actual communication between humans can occur; for instance, it may be necessary to leave voice-mail messages when some intended parties to a communication are unavailable. For clarity, we further identify the BEGIN events of communication intervals as either BEGIN-Send events or BEGIN-Receive events.

The second category of events is *automation events*, which demarcate intervals of activity in which a computer performs some task, such as a compilation. In some cases, an automation activity is initiated by a setup procedure to prepare the computer for the automation task. For instance, job requests may be placed in a queue in order to allocate computer resources more efficiently. In this case, the instances of the automation activity would be characterized by an interval starting with a SETUP event (denoting the beginning of the period during which the job is in the queue), followed by a BEGIN event (denoting the beginning of the requested task), and followed finally by an END event (denoting the completion of the task). Some automation activities may be terminated before they would complete normally. Thus, we further identify the END events of automation intervals as either END-Normal events or END-Abort events.

The next three categories of events are straightforward characterizations of process activities carried out over intervals between corresponding pairs of BEGIN and END events. *Analysis events* identify intervals in which the results of a previous process chore are analyzed for validation or problem resolution; typical analysis event intervals correspond to code inspections, debugging sessions, and the like. *Work events* identify intervals in which a person is performing tasks such as fixing code, writing documentation, mounting a tape, and so on. *Workday events* correspond to times at which a person stops or resumes work in the process; typical workday events include arriving at work, going to lunch, starting work on some other assignment not related to the process, and so on.

The final category of events is *decision events*. Decision events correspond to a person synthesizing the results of a number of previously completed activities and using information about those results to select from a set of possible new activities to begin. Decision events are extremely sensitive to their inputs (i.e., the results of previous activities) and can thus provide a great deal of information about the efficiency of a process. For example, a decision may be made based on information gleaned from previously completed analysis intervals and in anticipation of an imminent workday event. Retrospective analysis can reveal where additional information could have led to a better decision. Decisions are considered to be made instantaneously and are thus characterized by a single DO event rather than by an event interval.

3 Overview of the Subject Process

The initial subject of our study was a software build process employed in a large project under development at AT&T. This process is an ideal one to examine because it is regularly repeated with little change in its basic, day-to-day activities. The group responsible for official builds enacts the several-day process once every few weeks. The software that they build during one of these cycles consists of several million lines of source code partitioned into several dozen *subsystems*. Three major tools are used to build the software; we refer to them below simply as `Build1`, `Build2`, and `Build3`. At the conclusion of the process, a small set of executable *products* have been built from the subsystems and installed in system labs or test execution environments. Each product is targeted to a different hardware component of the system.

The build process involves several roles. The *build owner* coordinates the process, tracks down build problems, and communicates with *developers* located around the world. The *build administrator*, at the direction of the build owner, sets up the actual builds for execution according to a written guidebook. The build owner also typically has several *build assistants* to whom problem-tracking chores can be delegated.

4 Capture of the Subject Process

A critical aspect of this study was to determine how best to capture, in some practical way, the full range of events in the taxonomy described in Section 2.2. We began by interviewing members of the group responsible for the build process under study and observing some of their activities during an actual enactment of the process. This allowed us to to become familiar with the personnel involved (and they with us) as well as with the details of their activities and interactions. From these early experiences we learned several important things:

- It is impossible to capture all events automatically, as desirable as that may be. Moreover, it would be inadequate to capture only those events that could be captured automatically.
- It is inappropriate to capture events by asking the members of the group themselves to either record the events as they occur or to log the events sometime later, such as at the end of the workday.
- It is important to decide upon an appropriate level of granularity for the captured events. At too fine a grain, it would be extremely difficult, if not impossible, to capture all necessary information about the events as they occurred. At too coarse a grain, we would likely miss important information that would contribute to our understanding of the process.

Based on these early experiences, we designed a technique for process capture that relies upon independent, direct observation to record those events whose capture could not be automated, and we developed tools to automatically derive event data from the log files generated by the build tools.

We decided to place an observer alongside the build owner, since the build owner is clearly the focus of activity during the process. To avoid overloading the observer, we chose a granularity based on the subsystem. Thus, for example, events of the same kind occurring on multiple files of a particular subsystem would be recorded with the event kind and the name of the subsystem, but would not be distinguished by, say, the names of the affected files. This appears consistent with the way that the build owners themselves view their activities. We designed a simple log sheet, shown in Figure 2, to allow the observer to easily capture the important information about an event. In particular, the observer is expected to record each event on a single line of the log sheet, giving a unique numeric identifier to the event (UIE) and noting the kind of the event, the date and time the event occurred, the names of the subsystems and products associated with the event, and the people involved in or contributing to the event. In addition, the observer is expected to relate this event to any other relevant, preceding events (in the field "Other UIE(s)") and to indicate any other information contributing to the understanding of the event (in the field "Comments"). To make the task of assigning event kinds easier, we predefined a specific set of event kinds that we found relevant to the process under study; this set is an instantiation of the taxonomy of Section 2.2. The event kinds are presented in Table 1, along with the codes used to identify them in the log sheets. On the whole, we found the log sheet to be highly useful both for making the recorded data informative and for keeping it concise.

The observer was responsible for recording events of all kinds except automation events. The automation events were instead automatically derived from the log files produced by the build tools. This derivation is performed "off-line", which poses the problem of how to relate the automation events to the other, manually captured events. The solution we adopted was to assume that each automation event interval for a given subsystem is related to the analysis event interval for that subsystem beginning soonest in time after the automation interval. We record this assumed relationship in the "Other UIE(s)" field of the analysis events.

5 Analysis of the Subject Process

There are many kinds of properties one might want to discover about or enforce within a software process and, therefore, many kinds of analysis one might perform. Some properties, such as satisfaction of safety or liveness requirements (e.g., freedom from deadlock), are difficult or impossible to detect through retrospective analysis; deductive analysis is better suited to discovering the existence or non-existence of such properties. The kinds of analysis best suited to empirically gathered data include analysis of fairness in resource allocation, real-time performance of process activities, resource utilization, and degree of concurrency. Many such analysis tasks are naturally characterized in terms of relationships among process events.

The basic approach in our analysis of the subject process has been to enter the captured event information into a relational database and to perform queries

UIE	Date	Time	Event	Product	Subsystem	Contacts(s)	Other UIE(s)	Comments
00								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
1								
2								
3								
4								
5								
6								
7								
8								
9								
20								
1								
2								
3								
4								
5								
6								
7								
8								
9								
30								
1								
2								
3								
4								
5								
6								
7								
8								
9								
40								
1								
2								
3								
4								
5								
6								
7								
8								
9								

Figure 2: Log Sheet Used for Capture of the Build Process.

COMMUNICATION		AUTOMATION		ANALYSIS	
C1	BEGIN-Send-Call	B1	SETUP-Build1	A1	BEGIN-Analysis
C2	BEGIN-Send-VoiceMail	B2	BEGIN-Build1	A2	END-Analysis
C3	BEGIN-Send-EMail	B3	SETUP-Build2	**WORK**	
C4	BEGIN-Send-Fax	B4	BEGIN-Build2		
C5	BEGIN-Send-Visit	B5	SETUP-Build3	W1	BEGIN-Work
C6	BEGIN-Send-Hangup	B6	BEGIN-Build3	W2	END-Work
C7	BEGIN-Send-VisitNotIn	B7	END-Build-Normal	**WORKDAY**	
C8	BEGIN-Receive-Call	B8	END-Build-Abort		
C9	BEGIN-Receive-VoiceMail			O1	BEGIN-Other
C10	BEGIN-Receive-EMail			O2	END-Other
C11	BEGIN-Receive-Fax			**DECISION**	
C12	BEGIN-Receive-Visit				
C13	END-Communication			D1	DO-Decision

Table 1: Event Kinds for the Build Process.

on the database that analyze the relationships among the events to reveal interesting characteristics of the process. The database system we use for our analysis experiments is DataShare [7], whose query language, Cymbal, provides powerful facilities for analyzing relationships among data records.

By way of example, this section describes two analysis queries we have developed. The analysis queries are based on the following hypothesis: There are some subsystems that are continual sources of build problems from enactment to enactment. Such problems may arise from a corruption of the original architecture of the subsystem, or from a large amount of special-case code required to tailor system features to different configurations. Such problems produce faults that may not be discovered until build time, resulting in two classes of undesirable phenomena at build time:

1. There can be an inordinately large number rebuilds of the same subsystem. This could be due to a large number of faults discovered at build time, or a large number of unsuccessful attempts to eliminate a fault.

2. There can be inordinately large amounts of time required to eliminate faults. This could be due to the inherit difficulty of analyzing the code of the subsystem for fault isolation and elimination, or the lack of sufficient resources allocated to the subsystem for problem resolution.

Such "problem subsystems" can be identified by analyzing the event data from at least two perspectives:

1. From an automation-oriented perspective, problem subsystems are revealed by large numbers of SETUP-Build or BEGIN-Build events for the subsystem (corresponding to phenomenon (1) described above) and/or long intervals of time between builds of the subsystem (corresponding to phenomenon (2) described above).

2. From a communication-oriented perspective, problem subsystems are revealed by long problem-solving communication intervals during the intervals between builds (again corresponding to phenomenon (2) described above).

If the event data revealed such phenomena in the same subsystem across several enactments, that subsystem would be a good candidate for an improvement of some kind. For instance, the subsystem could be given a higher priority during builds in subsequent enactments, or the development organization responsible for the subsystem could undertake root-cause analysis, an architecture redesign, or some other activity aimed at process improvement.

The sections below describe the two queries we developed to detect these phenomena. The queries analyze the process event data from the two complementary perspectives of automation and communication.

5.1 Sample Analysis 1: Time Between Builds

We developed the query *SubsysInterbld* to compute the average time between instances of each kind of build (`Build1`, `Build2` and `Build3`), for each subsystem; the query presents these averages for the three kinds of builds for all subsystems.

To compute the averages, the query only analyzes the relationships between related groups of automation events. In particular, an inter-build interval is identified by an END event for the particular kind of build on a particular subsystem, followed by a subsequent corresponding SETUP event, or a subsequent corresponding BEGIN event if the next build had no SETUP event. The results of this query for one enactment of the process are presented in Table 2.

The statistics in Table 2 can be somewhat difficult to analyze because of an inherent interdependency among the build tools. Scanning through the `Build1` averages for the subsystems, we note that most subsystems required at least six complete build cycles starting with `Build1`, all possibly due to a number of fixes with widespread impact across subsystems. Thus, the `Build2` averages for those subsystems are skewed somewhat by the fact that they involve pairs of `Build2` intervals containing intervening `Build1` intervals. By analyzing the statistical correlation between different kinds of builds on the same subsystem, and between builds on different subsystems, one could more accurately characterize the effect of the interdependency among the build tools.

Despite this interdependency, we see that the five subsystems with the greatest number of `Build2`s—`s3`, `s4`, `s10`, `s32` and `s37`—all required roughly the same relatively short average time between `Build2`s, even though they required different numbers of, and widely varying times between, their `Build1`s. In contrast, the subsystems with the fewest number of `Build2`s also had large average inter-build times for `Build2`. Furthermore, the number of `Build2`s for these latter subsystems nearly matched the corresponding number of `Build1`s, suggesting a number of possible explanations. For example, it may be that `Build2`s for these latter subsystems were performed more to account for changes to global interfaces than as a direct result of fixes being made to those subsystems. Correlation analysis would reveal the proper explanations

Subsystem	Build1		Build2		Build3	
s1	0d 00h 00m	(0)	0d 00h 00m	(0)	0d 07h 14m	(16)
s2	2d 19h 25m	(6)	0d 15h 53m	(22)	0d 00h 00m	(0)
s3	1d 23h 55m	(8)	0d 09h 05m	(38)	0d 00h 00m	(0)
s4	1d 17h 59m	(9)	0d 10h 07m	(32)	0d 00h 00m	(0)
s5	1d 23h 45m	(8)	0d 12h 53m	(25)	0d 00h 00m	(0)
s6	2d 08h 07m	(7)	0d 17h 43m	(20)	0d 00h 00m	(0)
s7	2d 07h 42m	(7)	1d 01h 22m	(14)	0d 00h 00m	(0)
s8	1d 10h 50m	(12)	0d 22h 02m	(16)	0d 00h 00m	(0)
s9	0d 00h 00m	(0)	1d 05h 59m	(12)	0d 00h 00m	(0)
s10	1d 17h 32m	(9)	0d 11h 10m	(30)	0d 00h 00m	(0)
s11	2d 07h 27m	(7)	0d 18h 52m	(19)	0d 00h 00m	(0)
s12	0d 00h 00m	(0)	2d 13h 40m	(6)	0d 00h 00m	(0)
s13	2d 07h 22m	(7)	0d 23h 48m	(15)	0d 00h 00m	(0)
s14	2d 07h 18m	(7)	1d 02h 57m	(12)	0d 00h 00m	(0)
s15	0d 00h 00m	(0)	1d 20h 11m	(9)	0d 00h 00m	(0)
s16	2d 18h 58m	(6)	0d 00h 00m	(0)	0d 00h 00m	(0)
s17	1d 17h 37m	(9)	1d 10h 39m	(10)	0d 00h 00m	(0)
s18	2d 08h 01m	(7)	2d 15h 36m	(6)	0d 00h 00m	(0)
s19	0d 00h 00m	(0)	0d 00h 00m	(0)	1d 05h 50m	(5)
s20	1d 23h 42m	(8)	0d 15h 38m	(21)	0d 00h 00m	(0)
s21	2d 07h 37m	(7)	1d 04h 28m	(12)	0d 00h 00m	(0)
s22	1d 17h 43m	(9)	1d 14h 30m	(9)	0d 00h 00m	(0)
s23	1d 12h 38m	(11)	1d 05h 02m	(15)	0d 00h 00m	(0)
s24	2d 18h 22m	(6)	0d 18h 21m	(19)	0d 00h 00m	(0)
s25	2d 07h 55m	(7)	0d 21h 00m	(16)	0d 00h 00m	(0)
s26	1d 09h 37m	(11)	1d 13h 15m	(10)	0d 00h 00m	(0)
s27	1d 23h 44m	(8)	0d 19h 11m	(18)	0d 00h 00m	(0)
s28	1d 23h 47m	(8)	0d 22h 21m	(15)	0d 00h 00m	(0)
s29	1d 14h 20m	(9)	0d 14h 02m	(23)	0d 00h 00m	(0)
s30	2d 00h 02m	(8)	1d 11h 41m	(10)	0d 00h 00m	(0)
s31	2d 07h 51m	(7)	0d 18h 32m	(19)	0d 00h 00m	(0)
s32	0d 00h 00m	(0)	0d 08h 12m	(42)	0d 00h 00m	(0)
s33	1d 23h 48m	(8)	0d 17h 07m	(19)	0d 00h 00m	(0)
s34	1d 23h 38m	(8)	0d 23h 15m	(15)	0d 00h 00m	(0)
s35	1d 13h 18m	(9)	0d 15h 01m	(20)	0d 00h 00m	(0)
s36	2d 07h 29m	(7)	1d 00h 09m	(14)	0d 00h 00m	(0)
s37	1d 17h 48m	(9)	0d 11h 46m	(29)	0d 00h 00m	(0)
s38	2d 08h 03m	(7)	0d 22h 21m	(16)	0d 03h 08m	(37)
s39	2d 07h 56m	(7)	2d 07h 17m	(7)	0d 00h 00m	(0)
s40	1d 23h 45m	(8)	0d 15h 39m	(22)	0d 00h 00m	(0)
s41	2d 07h 53m	(7)	2d 11h 47m	(6)	0d 00h 00m	(0)
s42	0d 00h 00m	(0)	2d 03h 26m	(7)	0d 00h 00m	(0)
s43	0d 00h 00m	(0)	1d 05h 02m	(15)	0d 00h 00m	(0)
s44	1d 18h 01m	(9)	0d 11h 20m	(27)	0d 00h 00m	(0)
s45	1d 23h 54m	(8)	1d 08h 42m	(10)	0d 00h 00m	(0)
s46	0d 00h 00m	(0)	2d 13h 27m	(6)	0d 00h 00m	(0)
s47	1d 17h 59m	(9)	0d 20h 49m	(16)	0d 00h 00m	(0)
s48	2d 07h 57m	(7)	0d 18h 28m	(18)	0d 00h 00m	(0)
s49	2d 07h 53m	(7)	1d 04h 27m	(12)	0d 00h 00m	(0)
s50	2d 00h 09m	(8)	1d 12h 26m	(10)	0d 00h 00m	(0)

Table 2: Average Time Between Builds (and Number of Builds) for One Enactment of the Build Process.

for these phenomena.

Based on these preliminary observations, we conclude that the subsystems with the greatest number of `Build2`s are the subsystems that required the greatest amount of problem resolution in this version of the software. If these observations were correlated across multiple enactments for the same subsystems, they would be objective evidence of the need for some kind of process improvement targeted to those subsystems.

5.2 Sample Analysis 2: Duration of Communication Threads

For purposes of our second analysis, we define a *communication thread* as a group of related communication events all devoted to discussion of a single problem. For instance, consider the following scenario, each item of which would be identified by a BEGIN-Send/END-Communication or a BEGIN-Receive/END-Communication event interval:

1. The build owner sends e-mail to developer X, who is responsible for the portion of code in which the build owner found a problem.

2. After waiting an hour for a reply to the e-mail, the build owner places a phone call to developer X; the developer does not answer, so the build owner leaves a voice-mail message.

3. Some time later, developer X returns the build owner's phone call, during which time the build owner describes the problem to the developer.

4. The developer telephones the build owner again to declare the problem solved.

We identify this communication thread by the interval between the BEGIN-Send-EMail event of event interval (1) and the END-Communication event of event interval (4).

We developed the query *SubsysComm* to compute the average duration of a communication thread for each subsystem. The query presents these averages for all subsystems on which communication took place, along with the number of communication threads for those subsystems.

Table 3 presents the results of this query for two days of an enactment of the build process. Communication threads that did not involve a specific subsystem are combined into a single item called *NONE*. The subsystems in Table 3 fall into two classes—subsystems that required communication threads of less than an hour to resolve problems, and subsystems that required communication threads of more than an hour to resolve problems. These latter subsystems—`s1`, `s10` and `s29`—are potential problem subsystems. If such statistics were revealed for the same subsystems in a larger sample and were correlated across multiple enactments, they would be objective evidence of the need for some kind of process improvement targeted to those subsystems.

Subsystem	Average Duration	
NONE	0d 00h 32m	(5)
s1	0d 01h 53m	(3)
s5	0d 00h 05m	(1)
s6	0d 00h 40m	(1)
s10	0d 01h 26m	(10)
s13	0d 00h 40m	(1)
s17	0d 00h 36m	(3)
s18	0d 00h 03m	(1)
s28	0d 00h 52m	(2)
s29	0d 02h 12m	(1)
s35	0d 00h 27m	(2)
s38	0d 00h 01m	(1)
s44	0d 00h 17m	(6)

Table 3: Average Duration (and Number) of Communication Threads for Two Days of an Enactment of the Build Process.

6 Conclusion

This paper has described an event-based software process model and associated process data capture and analysis techniques. The paper has also described a study in which we applied the capture and analysis techniques to the build process of a large software project within AT&T. Our experience has demonstrated the viability of this approach to process data capture and analysis. The greatest strengths of our approach are its objectivity and its focus on the dynamic aspects of process. However, we feel that the manual side of process data capture needs to be improved because it is currently a costly and labor-intensive undertaking. While all manual capture techniques suffer these inherent limitations, we feel that we have minimized their effect through the design of our log sheets and our streamlining of the captured data.

We are currently studying other build processes within AT&T. In the future we would also like to apply our model and techniques to other kinds of processes, such as testing and product-distribution processes. For the build process we studied, our event kinds and analysis queries were designed in a rather ad hoc manner. In the future we would like to use a system such as TAME [3, 11] to help make the design of the analysis queries more systematic.

Acknowledgments

We wish to thank the following people for their contributions to this study: Randee Fabrizius, Mary Caruso, Randy Hackbarth, Harry Harabedian, Steve Gryl, Malissia Williams, Rowena Johnson, Lori Ann Thorson, Rick Greer and Dave Belanger.

References

[1] Sergio Bandinelli, Carlo Ghezzi, and Angelo Morzenti. A multi-paradigm Petri net based approach to process description. In Ian Thomas, editor, *Proceedings of the 7th International Software Process Workshop*, October 1991.

[2] Naser S. Barghouti and Gail E. Kaiser. Scaling up rule-based development environments. In A. van Lamsweerde and A. Fuggetta, editors, *Proceedings of the 3rd European Software Engineering Conference*, number 550 in Lecture Notes in Computer Science, pages 380–395. Springer-Verlag, October 1991.

[3] Victor R. Basili and H. Dieter Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[4] Victor R. Basili and David M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, November 1984.

[5] Mark G. Bradac, Dewayne E. Perry, and Lawrence G. Votta. Prototyping a process monitoring experiment. Internal AT&T Bell Laboratories Memorandum; submitted for external publication, 1992.

[6] R. Mark Greenwood. Using CSP and system dynamics as process engineering tools. In J. C. Derniame, editor, *Proceedings of the Second European Workshop on Software Process Technology*, number 635 in Lecture Notes in Computer Science, pages 138–145. Springer-Veralag, September 1992.

[7] Richard Greer, April 1992. Internal AT&T Bell Laboratories Memorandum.

[8] Volker Gruhn and Rüdiger Jegelka. An evaluation of FUNSOFT nets. In J. C. Derniame, editor, *Proceedings of the Second European Workshop on Software Process Technology*, number 635 in Lecture Notes in Computer Science, pages 196–214. Springer-Veralag, September 1992.

[9] Mark I. Kellner. Software process modeling support for management planning and control. In Mark Dowson, editor, *Proceedings of the 1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 8–28. IEEE Computer Society, October 1991.

[10] Dewayne E. Perry. Policy-directed coordination and cooperation. In Ian Thomas, editor, *Proceedings of the 7th International Software Process Workshop*, October 1991.

[11] H. Dieter Rombach. Specification of software process measurement. In Dewayne E. Perry, editor, *Experience with Software Process Models: Proceedings of the 5th International Software Process Workshop*, pages 127–179. IEEE Computer Society, October 1989.

[12] David S. Rosenblum and Balachander Krishnamurthy. An event-based model of software configuration management. In Peter H. Feiler, editor, *Proceedings of the 3rd International Workshop on Software Configuration Management*, pages 94–97. ACM SIGSOFT, 1991.

[13] Motoshi Saeki, Tsuyoshi Kaneko, and Maskai Sakamoto. A method for software process modeling and description using LOTOS. In Mark Dowson, editor, *Proceedings of the 1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 90–104. IEEE Computer Society, October 1991.

[14] Richard W. Selby, Adam A. Porter, Doug C. Schmidt, and Jim 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, May 1991.

[15] Stanley M. Sutton, Jr. Accommodating manual activities in automated process programs. In Ian Thomas, editor, *Proceedings of the 7th International Software Process Workshop*, October 1991.

[16] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. Language constructs for managing change in process-centered environments. In Richard N. Taylor, editor, *Proceedings of the 4th Symposium on Software Development Environments*, pages 206–217. ACM SIGSOFT, December 1990.