Understanding Software Development Processes, Organizations, and Technologies

Dewayne E. Perry[1], Nancy A. Staudenmayer[2], and Lawrence G. Votta, Jr.[3]

[1]AT&T Bell Laboratories, Murray Hill, New Jersey 07974
[2]Sloan School of Management, MIT, Cambridge, Massachusetts 02139
[3]AT&T Bell Laboratories, Naperville, Illinois 60566

ABSTRACT

Our primary goal is to understand what people do when they develop software and how long it takes them to do it. To get a proper perspective on software development processes we must study them in their context — that is, in their organizational and technological context. An extremely important means of gaining the needed understanding and perspective is to measure what goes on.

Time and motion studies constitute a proven approach to understanding and improving any engineering processes. We believe software processes are no different in this respect; however, the fact that software development yields a collaborative intellectual, as opposed to physical, output calls for careful and creative measurement techniques.

In attempting to answer the question "what do people do in software development?" we have experimented with two novel forms of data collection in the software development field: time diaries and direct observation. We found both methods to be feasible and to yield useful information about time utilization. In effect, we have quantified the effect of these social processes using the observational data.

Among the insights gained from our time diary experiment are 1) developers switch between developments to minimize blocking and maximize overall throughput, and 2) there is a high degree of dynamic reassignment in response to changing project and organizational priorities.

Among the insights gained from our direct observation experiment are 1) time diaries are a valid and accurate instrument with respect to their level of resolution, 2) unplanned interruptions constitute a significant time factor, and 3) the amount and kinds of communication are significant time and social factors.

1. Introduction and Motivation

To understand the processes by which we build large software systems, we must consider the larger context within which these systems are developed: namely, the organizational and social contexts as well as the technological. It is our claim that entirely too much attention has been paid to the technological aspects of software development at the expense of social and organizational issues. One reason for this divergence is the inherent difficulty of obtaining quantitative measurements of "people" factors. We disagree with this reasoning. It *is* possible to quantify what are often assumed to be qualitative factors. Furthermore, we believe that a more holistic measurement-based approach (encompassing all these aspects of process technology and organization) is necessary first to understand truly development processes, second to improve these processes and third to assess and to justify the improvements.

In the remainder of this introduction, we expand on the importance of organizational and social contexts and the need for new measurements. We also address the iterative nature of understanding and measuring in their role as necessary preconditions to effective improvement. In Section 2 we discuss some general measurement and methodological issues confronted when studying organizations. In Sections 3 and 4, we describe in detail the studies we conducted in a software development organization—why and how we took measurements on the existing process and the insights about these processes. We present our conclusions in Section 6.

1.1 Organizational and Social Context

In 1975, Frederick P. Brooks, Jr. described software construction as inherently a systems effort, "an exercise in complex (human) inter-relationships" [Brooks75]. Yet now, almost twenty years later, most improvement exercises still focus on the technological aspects of building software systems — the tools, techniques and languages people use to actually write software. Relatively little systematic study has been done about the associated people issues—how to ensure accurate and effective communication about a product no one can see, how to maintain project motivation and focus in the face of blockage and distraction. While many articles have addressed the *importance* of such issues, few have conducted *systematic* investigations about their *operation* in software development.

We offer three reasons for a greater emphasis on the organizational and social context of software processes. First, even in the most tool-intensive parts of software development such as the system build process the human element is critical and dominant. While the efficiency and appropriateness of the tools are obviously important, the crucial job of tracking down the sources of inconsistency and negotiating their resolution is performed by people.

Second, numerous process studies have indicated a large amount of unexplained variance in performance [Boehm88; Brooks80; Cusum91;Vos84] suggesting that significant aspects of the process are independent of the technological context. In fact, one might argue that the continual introduction of more and more sophisticated tools is responsible for a "tool mastery burden" [Brooks75] and thus augments the unexplained variance. Moreover, all too often tools are designed in isolation and subsequently fail to achieve their promised potential once "in use." Genuine advances in tools and languages must be accompanied by a consideration as to how the technology will be incorporated into the existing social and organizational infrastructure.

Finally, several prominent authors have noted that a significant proportion of project effort is devoted to non-programming activities, with some estimates indicating as much as 50% of a work week typically absorbed by machine downtime, meetings, paperwork, company business, sick and personal days [Boehm88; Brooks75]. If less than half the time is spent programming and if new technical advances are yielding decreasing marginal benefits, perhaps we should look elsewhere for sources of process improvement leverage.

1.2 The Need for New Measurements

The fast pace of technological and market changes, together with constantly shifting organizational structures, require us to reevaluate our assumptions and understanding of development processes on a regular basis. On the one hand, a changing environment may render old assumptions invalid. For example, one unfortunate result of a narrow technical focus is that the few studies that have investigated programmers have generated a large number of myths that need to be challenged [Weinb71]. We will have

some comments to make about such accepted truisms as (1) developers don't like to be (and, hence, cannot be) observed; (2) software programming is an isolated type of activity.

On the other hand, there may be new problems that are not being addressed adequately. Most studies that investigate the human aspects of programming rely primarily on student programmers or artificial tasks in laboratory settings [Curtis86]. Although these studies are informative and useful, we question their relevance to large scale software development. How representative are the samples and tasks? What kinds of problems, unique to organizational environments, are being ignored by focusing on these small and artificial domains?

We need to explore new ways to measure and understand our processes in their actual context. We have to stop quoting "facts" that are twenty years old and start aggressively questioning our assumptions about how work actually gets done in the development of software.

1.3 The Iteration of Understanding and Measurement

Answering the question of what to measure is an iterative exercise that both depends on our understanding of the process and helps to transform that understanding. We (as performers and observers of the processes) have some intuition about where problems lie. For example, if it is our understanding that progress is often blocked, the obvious things to measure are the blocking factors. If meetings are seen as a significant impedance to progress, then we should measure the number, duration, and effectiveness of meetings to gain an understanding of their effect on our processes and performance. We use our understanding to determine what measures to take and then use the results of those measures to confirm or deny our hypotheses and perhaps to transform our understanding. We iterate between hypotheses based on our understanding and gathering data using various measures. Note that one must be careful not to let preconceptions interfere with insights to be gained from the measurement process.

1.4 Measurement and Understanding as the Necessary Prelude to Improvement

Many of our process improvement claims are based on anecdotal evidence or reasonably plausible arguments. While these may give us some comfort, they do not constitute a quantifiable basis for *claiming* improvement. Measurement-based understanding of the current processes does provide this necessary quantifiable basis. It enables us to accurately benchmark existing processes and quantify the value of subsequent improvement efforts.

An important and significant precedent for such an approach is the work done in the early 1960's in Japanese software factories in which the Japanese gathered data on existing processes before changing or improving them [Cusum91]. More recently, Wolf and Rosenblum made the same point, noting that "in order to improve processes and design new ones, it is necessary to obtain concise, accurate and meaningful information about existing processes" [Wolf93]. That is, by understanding how and why programmers work the way they do, we will be better positioned to identify tools and methods that enable them to perform tasks better and/or in less time [Leves92; Shneid86].

2. Methodological and Measurement Issues in Studying Organizations

To gain a better understanding of software development intervals, we performed three experiments: a prototype experiment to explore issues in using time diaries [Bradac93], the time diary experiment itself, and a calibration study to determine the accuracy and validity of time diaries as an experimental instrument.

Before describing our experiments we address four important issues which arise when one adopts the approach of studying software development processes in action: protecting the anonymity of study participants and minimizing the interference with their on-going work, documenting the dimensions of the study site, selecting the study sample, choosing the instrumentation and levels of resolution. In each case, we discuss the general problem faced by the researcher and how we chose to resolve it.

2.1 Experimenting with People Who Work

Several unique issues arise when one studies individuals in organizations. People are often understandably reluctant to reveal things at work. They have concerns about how the information will be used and who will see it. The researcher must therefore always be mindful of the study's impact on individuals. Thus, anonymity and confidentiality are of the utmost importance — after all, careers are

potentially at stake! In addition, interference with normal work progress must be minimized as much as possible.

Because we were aware that some people might be uncomfortable about participating in our experiments, we spent considerable time beforehand explaining the purpose of the studies to the subjects. They were reminded that there are no right or wrong ways to work (i.e., our purpose was not to judge but to understand behavior within a given environment).

All data was entered under an ID code known only to the researchers. Each subject was also given a list of his/her rights: the right to halt or to discontinue participation at any time or to withdraw from the study altogether; the right to examine the research notes; and the right to ask us *not* to record something. None of these situations occurred.

2.2 The Organizational Site

Software development organizations come in many shapes and sizes. They have distinct cultures and build widely different products. To facilitate cross-study comparisons, researchers must get in the habit of clearly delineating the principle dimensions of the organization, the process and the supporting technology they are measuring.

The subjects for our studies build software for a real time switching system. The system is a successful product with over 10 million noncommentary source lines (NCSL) of C code, divided into 41 different subsystems. New hardware and software functionality are added to the system approximately every 15 months.

A unit of functionality that a customer will pay for is called a feature and is the fundamental unit tracked by project management. These features vary in size from a few NCSL with no new hardware required to 50,000 NCSL with many complex hardware circuits developed specifically for that feature. Most of the software is built using a real time operating system.

The development organization responsible for product development consists of approximately 3,000 software developers and 500 hardware developers. This software development organization is currently registered to ISO 9001 standard and has been assessed as an SEI level 2 development organization.

We make no sweeping claims as to the generalizability of our study results to *all* software organizations. On the contrary, the relative size, complexity and maturity of this particular software system are inextricably associated with the process. We document these dimensions in order to bound the results we present later.

2.3 Sample Selection

A tradeoff always exists between minimizing the possible variance and maximizing the generalizability or external validity of a study's findings. On the one hand, we would like our results to be applicable to a large number of settings and samples. Yet we simultaneously need to control for random errors in order to improve the internal validity of our results. The researcher must therefore decide which extraneous variables are most important to control for. There is no right answer to such a question. Rather, one relies on judgement, prior research and resource availability. One way of avoiding these control issues *a priori* is to collect as much information on the sample subjects as possible and then control for various effects in an *a posteriori* covariance analysis.

We applied a purposive sampling scheme, selecting subjects at random yet stratifying along those dimensions we felt would be most significant. The two major factors we felt were most important to control for were (1) the project factors of organization, project phase and project type and (2) the personal factors of age, gender, race, individual personality and years of experience. Our goal was not to conduct a comparative study but rather to obtain a broad base of observations and to decrease the likelihood of idiosyncratic findings.

We are aware that our sample sizes are small and probably inadequate for statistical validity but, following the logic of [Brooks88], we believe "any data is better than none." This work falls within the second category of nested results Brooks cites as necessary and desirable for progress in software development: reports of facts of real user behavior even though observed in under-controlled, limited sample experiences.

2.4 Instrumentation

Finally, we come to the question of instrumentation — that is, how do we get the data and at what level of resolution?

The studies that we describe in sections 3 and 4 are part of a series of planned experiments whose goal is to understand the structure of software development processes in order to reduce the process development interval. The underlying theoretical model views such processes as complex queuing networks. In most software projects, there exists a large discrepancy between the race time and the elapsed time. The former is that time spent in actual work. It expands to the actual elapsed time in the presence of interruptions and blocking and waiting periods. The purpose of our studies was to document what kind of factors inhibited progress and their impact on overall development time. We considered several alternative methods for data collection before settling on time diaries and direct observation.

A standard methodology in behavioral science is a one-time, retrospective survey questionnaire. The drawback of this is that it provides a relatively flat and static view of the development process whereas we were interested primarily in the dynamic behavior of people performing highly interdependent tasks. Hence we chose to design a modified time card and asked software developers to record their daily activities over a period of 12 months.

To calibrate the accuracy of the time diaries, however, we needed an instrument with finer resolution. One option was to use video cameras. There are experimental precedents for this, but we felt it would be inappropriate for a number of reasons. First, our study population was not used to such intrusiveness. While the subjects were fairly receptive to the notion of participating in experiments, the introduction of video equipment would have distorted their behavior (not to mention that of their peers and overall work progress). Second, over 300 hours of video tape would have to be watched and interpreted, significantly increasing the cost and duration of the experiment. Finally, we were interested in obtaining information about *why* developers used their time the way they did — why they made certain choices and how they decided among competing demands on their time. Use of video would have precluded our asking questions about the subject's choices. Given these drawbacks, we decided to use direct observation of a sample of the participants in the time diary experiment.

2.5 Resolution

Fundamental to any decision to measure processes is the necessity of determining the cost/benefit tradeoffs. An important factor in that tradeoff is the resolution of the measurements required by the experiment — that is, the level of analysis and the frequency of the sampling. No single approach will cover all the different levels and frequency.

Figure 1 illustrates the contrasting resolution of the time diary and the calibrating direct observation. It is important to recognize that the observer recorded data contains an impressive amount of micro-level detail, often down to three minute intervals. In order to form an effective comparison, we therefore summarized that detail into major blocks of activities. We then verified the reliability of the summary process by randomly comparing reports prepared by independent researchers. The level of comparability was well within accepted research standards.

The first result of the calibration experiment is the validation of the time diary instrument as a low-cost, effective means of obtaining relatively large resolution data about the software process.

3. The Time Diary Study

In an initial pilot study [Bradac93], we drew upon a single programmer's personal log to construct a prototype time diary instrument. The log enabled us to identify the principle activities and working states as well as to formulate several hypotheses to test in the subsequent experiment.

3.1 The Time Diary Experiment

Over the one year life of the experiment we had a total of 13 people from 4 different software development departments fill out the time diary on a daily basis. During the course of the study, we revised the instrumentation several times as a result of both positive and negative feedback from the subjects. The result of these adjustments was an instrument that is easier to use than our initial prototype (most subjects

DIARY		OBSERVER	
0800 - 1800	Working High Level Design	0800 - 0900	Administration
		0900 - 1010	High level design analysis
		1010 - 1021	Break
		1021 - 1135	Code experiment with peer
		1135 - 1226	High level design document writing
		1226 - 1314	Lunch in cafeteria
		1314 - 1330	Answer document question (responsible person out)
		1330 - 1349	Answer growth question
		1349 - 1406	Reading results of Business Unit Survey
		1406 - 1500	Code experiment with peer
		1500 - 1626	Searching for paper and reading
		1626 - 1701	Code experiment with peer
		1701 - 1705	Administration

Figure 1: Comparison Sheet Example
Report form comparing a software developer's self-reported time diary with the observer's summarized notes. This sheet is typical of the calibration. Note the difference in end time between the diary and the observer's notes; ≈55 minutes. The diary contains one entry for this 9-10 hour day "Working High Level Design." The observer had 13 entries of which about 5 hours corresponded to activities associated with high level design.

spent 5-10 minutes per day filling it in) and yet which still managed to capture the basic data we sought to obtain about the development processes.

Figure 2 illustrates the final form of the time diary instrument. The resolution of reported time was one hour segments — a relatively large granularity, but one that is appropriate to the goals of this experiment.

The basic question for each reported time segment is whether the developer is working the assigned development project or not. If the developer is working, then it is a matter of reporting the appropriate task within the process and the appropriate activity within that task. (The task steps are extracted from the defined development processes; the activities partition the possible ways in which a developer may be performing that task.)

If the developer is not working the assignment, we differentiate three basic reasons: reassigned to a higher priority project, blocked waiting on resources, or personal choice to work on another activity.

3.2 Insights

Figure 3 presents the distribution of time spent over various tasks in the development processes. It is interesting to note that even though the phase of the development cycle was primarily coding, there is a reasonable distribution of time spent on other types of tasks. In fact, roughly half the time is occupied by non-coding tasks. This indicates rather clearly that not only does the waterfall model not reflect what actually goes on (which every developer already knows), but the accepted wisdom of an iterative and cyclical model of development is also inadequate. In a large project such as this one, both the product and the process are in multiple states at once. What we have is a large number of iterative, evolutionary development processes being performed concurrently.

In Figure 4 we present the distribution of time over the various process states (working the process, blocked waiting for resources and not working the process).[1] The ratio of elapsed time to race time is

1. Weekends are included in the not worked category because it is often the case that weekends were worked, especially in times of crises.

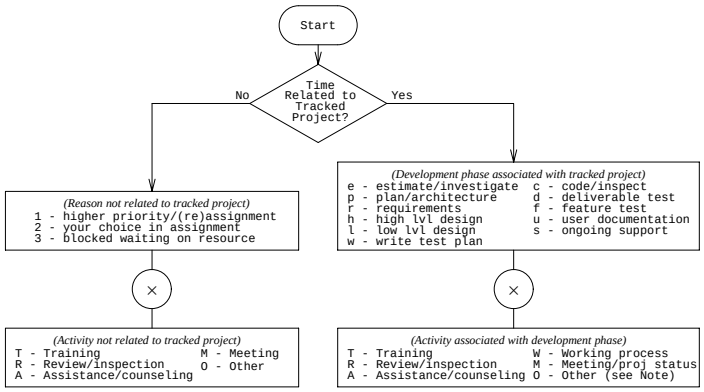
Diagnostic Development Process Measurement Record

Tracked Project: ABCDE

Developer: A. B. Smith

Date: Monday, August 9, 1993

Use the following flow chart to determine the number/letter or letter combination that best describes your activity on and off this project:



Use the number/letter or letter combination determined above to fill in the following time chart:

0000	0100	0200	0300	0400	0500	0600

0700	0800	0900	1000	1100	1200	1300	1400	1500	1600	1700

1800	1900	2000	2100	2200	2300

Comments: _____

Refer questions/comments to:

Mark Bradac: (708)979-3757, ihlpb!mbradac
Larry Votta: (708)713-4612, research!votta

Note: Comments are required for selection of: *"O - Other"*.

Figure 2: Software Developer Self-Report Time Form

roughly 2.5. In other words, these developers effectively worked on a particular development only 40% of the time. The remainder of the time was either spent waiting on resources or doing other work.

Most of the subjects happened to be in the coding phase of their assigned project. According to the data uncovered in our pilot study, the amount of blocking varied throughout the development cycle, and coding often exhibited the least amount of blocking, probably reflecting the fact that dependencies on outside organizations, resources and experts were at a minimum during this phase. This result was corroborated in the subsequent experiment. We also discovered that most of the developers in this experiment worked concurrently on two development projects. It is our conjecture that blocking was masked by opportunistic switching as a means of optimizing progress. This represents an organizational solution to the problem of

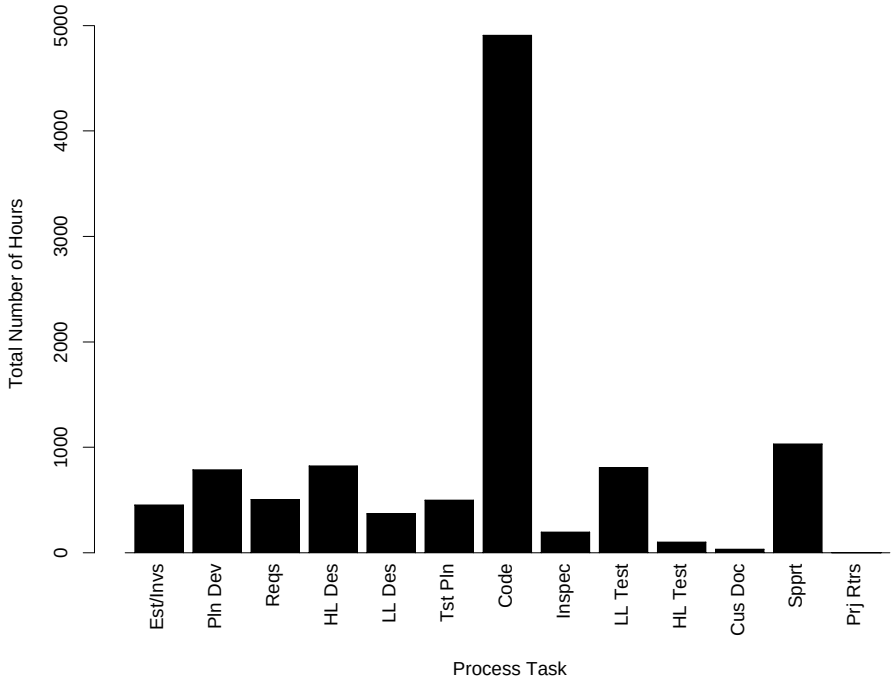


Figure 3: Self-Reported Total Time By Task

The histogram shows the number of self reported time diary hours reported for each process task for all subjects. The development organization is functionally organized and are participants for this cross sectional study were drawn from four different software development groups. Only about half the hours are reported as coding.

blocking.

The amount of rework done in our sample developments was about 20% — that is, roughly one fifth of the working time was devoted to fixing problems from previous work.

The time spent not working on each reported development is clearly dominated by reassignment to other projects. This reassignment emphasizes two important aspects of large scale software development. First, project organization is extremely dynamic bcause of changing priorities and evolving requirements. Second, reassignment arises because this is a realtime system that is being simultaneously used and modified. Besides fielding occasional critical customer problems, developers must also customize new features for specific customers.

Figure 5 presents a histogram of the duration of time intervals across all study subjects. Note that they tend to be weighted towards the low end of the scale. This indicates that developers rarely have lengthy uninterrupted working time periods, contributing to the inefficiency of the overall development process. The significant number of four hour working segments reflects the relatively mundane artifact of a day broken in half by lunch. More interesting are the frequency of two and eight hour segments. Two hour intervals arise from the organizational mandate limiting review meetings to less than or equal to two hours. The significant number of eight hour segments is due to the fact that the test laboratory is scheduled for either four or eight hour intervals depending on the complexity of the lab setup.

4. The Direct Observation Study

Although periodic interviews and occasional unannounced visits had convinced us that no conscious misrepresentation was occurring, we sought to check the reports of time usage submitted by software

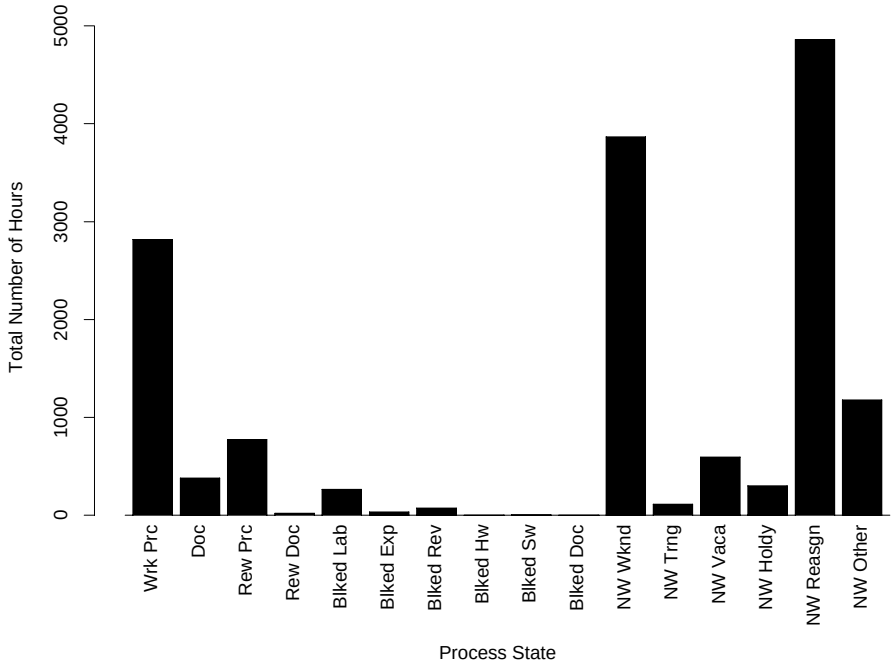


Figure 4: Self-Reported Time Diary Total Time By State

The histogram shows the number of self reported time diary hours reported for each process state for all subjects. Note the maximum amount of time is reported in the *NW Reasgn* state. This reflects the dynamic nature of priorities of software development feature work in this enterprise.

developers in a second experiment using direct observation. Although direct observation and ethnographic studies are fairly common in social science research, they are highly unusual in studies of software development. A common rationale is that "software developers don't like to be (and therefore cannot be) observed." The truth is that *no one* likes to be observed, but a well-designed experiment can do much to alleviate people's trepidation. This approach is not entirely without precedent. Hitachi conducted extensive observations of its programmers in the 1960's prior to the design of its software factory [Cusum91]. This section starts by describing the conduct of our observation experiment, followed by a description of some of the unexpected insights its measurements revealed.

4.1 The Direct Observation Calibration Experiment

Five software developers were chosen at random from the group participating in the time diary experiment. Two software developers who were *not* part of the self-reporting experiment were also included for observation; comparison with the other subjects enabled us to assess the impact of self-reporting, given observation. Somewhat surprisingly, no one who was asked refused to participate.

We applied a social experimental design that allowed us to efficiently control many factors with as few observations as possible and to determine whether the presence of an observer significantly changed the developers' behavior. In particular, by observing our subjects more than once under various conditions (repeated measurements), each subject served as his/her own control. The replicated interrupted time series design further allowed us to compare the same subject over time and different subjects at the same time. This enabled us to efficiently assess potential threats to validity posed by maturation, history and testing.

We were also cognizant of the so-called "Hawthorne Effect," the notion that the mere fact of having subjects self-report and/or be observed might alter their behavior and distort our conclusions. We therefore

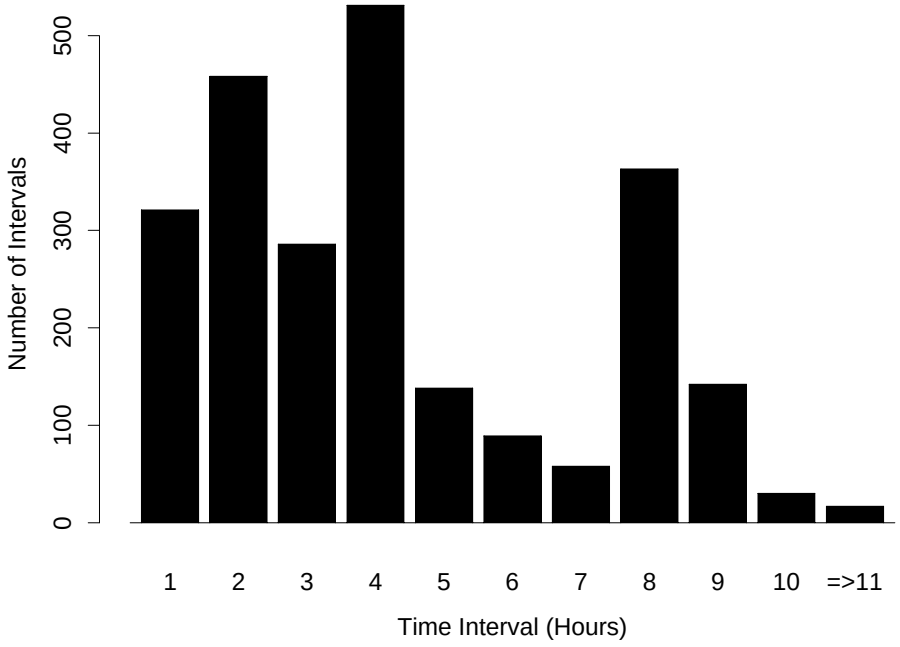


Figure 5: Self-Reported Time Diary Intervals

The histogram shows the number of self reported time diary entries by all subjects. For instance, there were over 600 diary entries of duration 4 to 5 hours. The histogram demonstrates that even in a relatively crude measurement as a self reported time diary, there are recognized breaks in their use of time. The spikes at 2, 4, and 8 hours are part of the culture of the development organization. The 2 hour peak occurs because of the maximum review meeting time of 2 hours. The 4 hour peak is the morning—afternoon break caused by lunch. The 8 hour break are the time diaries where the subject either worked through lunch or only had one entry per day.

built in several alternative control mechanisms in order to assess the significance of these possible mitigating factors. This included both standard hold-out samples as well as features of the experimental design. For example, we varied the predictability of the observations in the hope that by observing at random we would hinder the study subjects from adjusting their schedule so as to favorably impress the observer.

Each subject was observed a total of five full days (9-10 hours per day, on average, for 315 to 350 total hours of observation over a 12 week period). Two of the five days were chosen and scheduled by each subject.[2] The remaining three days were assigned by random draw without replacement, and the subjects were not informed of when they would occur.[3]

2. Interestingly, subjects often forgot when they had scheduled such sessions and were subsequently surprised to see the researcher in the morning. This reassures us that subjects were not too intimidated by the prospect of being observed.

3. The logistics behind this were not trivial. For example, vacations had to be blocked out in advance, and the observer had to adjust her schedule to accommodate subjects who worked flexible hours. Many lab sessions were also conducted off-hours, and a procedure was established for what to do in the event that a developer did not come into work.

To insure that information about the software developers was uniform, we created a checklist for data collection. It consisted of demographic information (age, gender, race, family obligations outside of work); educational and professional experience; current organization and project status; work habits; job security level; and overall job satisfaction. These statistics were collected in a one-hour interview. We also administered three survey instruments. Finally, we copied each subject's desk calendar for two months and noted all scheduled meetings, classes and vacation days. This data was used to corroborate and validate the observations.

The observer endeavored to function as a "human camera," recording everything with as few initial preconceptions about relative importance as possible. She frequently read a half page list of reminders designed to keep the observation procedure consistent. Because a single observer was used for all seven subjects, issues of inter-observer variability were avoided.

We used continuous real time recording for non-verbal behavior and interpersonal interactions. During those interims when a developer was working at the terminal, we used a time sampled approach: asking the developer at regular intervals "what are you doing now?" Daily observations were recorded in small spiral notebooks, unique to each subject. Each evening, the raw notebook observations were converted to standard computer files. This allowed us to readily fill in observations while still fresh and served as the basis for interim summary and analysis sheets described below. As data came in, it was added to a loose-leaf notebook, with separate sections for each subject. This helped us stay organized over time and facilitated communication among the researchers.

4.2 Insights

Gerald Weinberg once posed the provocative question, "Does it matter how many people a software developer runs into during the day?" [Weinb71] He argued that although the task of writing code is usually assigned to an individual, the end product will inevitably reflect the input of others. Indeed, one of the most salient impressions conveyed by observation was the sheer amount of time each developer spent in informal communication (we measured a total of, on average, 75 minutes per day of unplanned interpersonal interaction although this was scattered into episodes of widely differing duration).

Organizational theorists have long acknowledged the fact that information flow is a critical factor in organizational success [Allen77]. Most prior communication studies, however, address a very narrow range of interactions that occur in the course of a collaborative work effort. For example, they are typically restricted to only one media channel or focus on exchanges that are planned-in-advance and of relatively long duration. Moreover, the empirical data often consists of asking subjects who they talk to the most and thus risks confounding frequency with duration or impact. Our observational study offers a unique opportunity to address some of these deficiences in that it tracked all communication activity, at the individual level, across multiple media channels.

Figure 6 presents a sample of the communication summary sheet we prepared on each subject, based on the daily observations of their interactions across four major channels (voice mail, electronic mail, phone and in-person visits).[4] Drawing on the methodology of [Wolf93] we have broken each interaction down in terms of whether it was sent or received by the study subject. The form and content of the interactions recorded here are readily recognizable by anyone who has worked in a large corporate setting. Perhaps best described as "on-the-fly" exchanges [Galegh90], they usually involved little formal preparation and little reliance on pre-written documentation, diagrams or notes.[5] For example, a developer often received a call from the lab about a testing problem that needed immediate attention or had to respond to requests for authorization to change code that he was responsible for. Several of our developers had worked in other departments and therefore had to field questions from their former colleagues (this declines over time, but one of our subjects who had transferred departments approximately 2 months earlier received, on average,

4. Paper documentation is practically non-existent in this organization. This is partly due to the firm's interpretation of ISO requirements: if all documentation is on-line, the possibility of people using out-dated versions is decreased.

5. Note that we did not include contacts made in (scheduled) meetings or in the laboratory; nor did we include purely social exchanges (e.g., a call to a wife/husband, lunch partners). The unique count does not include faceless administrators.

SUBJECT ID									
		Voice Mail		Email		Phone		Visit	
	Uniq	Sent	Recv'd	Sent	Recv'd	Sent	Recv'd	Sent	Recv'd
Day1									
Day2									
Day3									
Day4									
Day5									

Figure 6 Communication Summary Sheet Example

This interim sheet summarizes the communication messages a software developer sent and received across four media channels during five days of observation. The table entries contain the total number of unique daily contacts ("Uniq") and the duration and time of day of a particular exchange.

one call a day from his former group). Finally, there existed a large amount of unplanned interaction with colleagues: requests to informally review code, questions about a particular tool, or general problem solving and debriefing sessions.

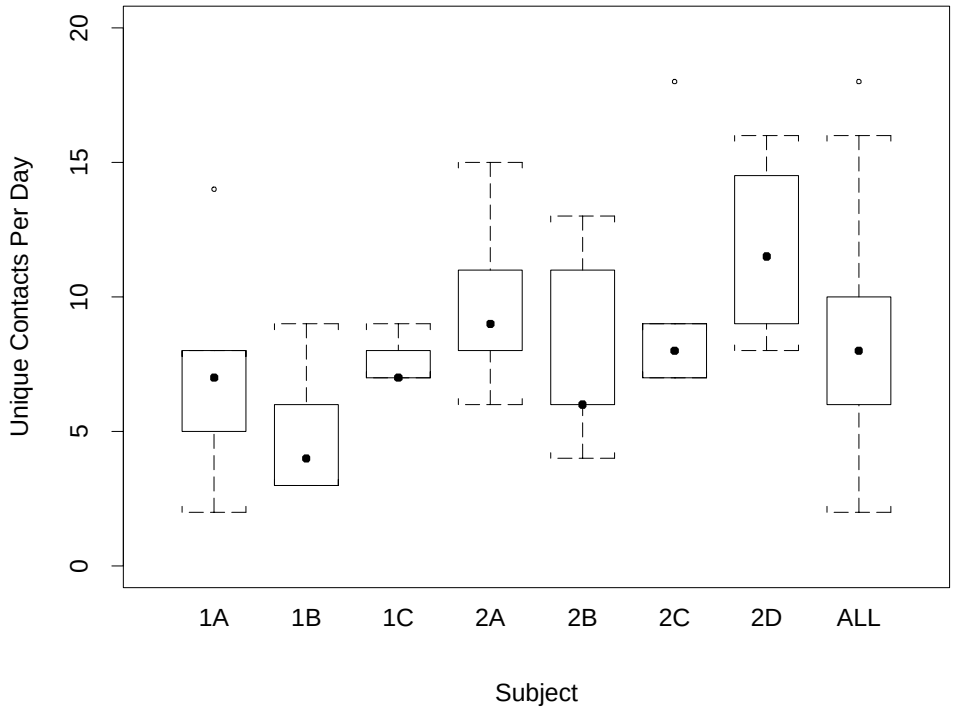


Figure 7 Number of Unique Contacts Per Subject Per Day

The number of unique person contacts for each subject per day. This count reflects interactions across four media channels (voice mail, email, phone and in-person visits) but does not include contacts made during meetings or lab testing. Nor does it include social exchanges. The median over all subjects is 7 (last boxplot). The outliers primarily reflect days in which a subject was working on modification to existing code.

How many people do these programmers interact with during a typical working day? Figure 7 presents a boxplot diagram depicting the number of unique daily contacts over 5 days of observation for each of our study subjects. A boxplot serves as an excellent and efficient means to convey certain prominent features of a distribution. Each set of data is represented by a box, the height of which corresponds to the spread of the bulk of the data (the central 50%), with the upper and lower ends of the box being the upper and lower quartiles. The data median is denoted by a bold point within the box. The lengths of the vertical dashed lines relative to the box indicate how stretched the tails of the distribution are; they extend to the standard range of the data, defined as 1.5 times the inter-quartile range. The detached points are "outliers" lying beyond this range. As depicted by the far right boxplot, the median number of unique contacts, across all study subjects, was seven per day.

The outliers in the boxplot diagram are particularly interesting. The highest point (17 unique contacts) represents a day in which developer 2C started to work on a code modification motivated by a customer field request. The other outliers also correspond to modifications of existing code, and in each case, the number of unique interfaces approximately doubled from the baseline of 7. The majority of these contacts were requests for authorization to change code owned by another developer. Just slightly less frequent were calls to a help desk for passwords or information about a particular release of the software; calls to the lab requesting available time slots for testing; and exchanges with peers about process procedures in general. Note that these contacts were not technically related per se. That is, the solution was usually not the motivating issue driving this behavior. Rather, the developers needed help *implementing* the solution.

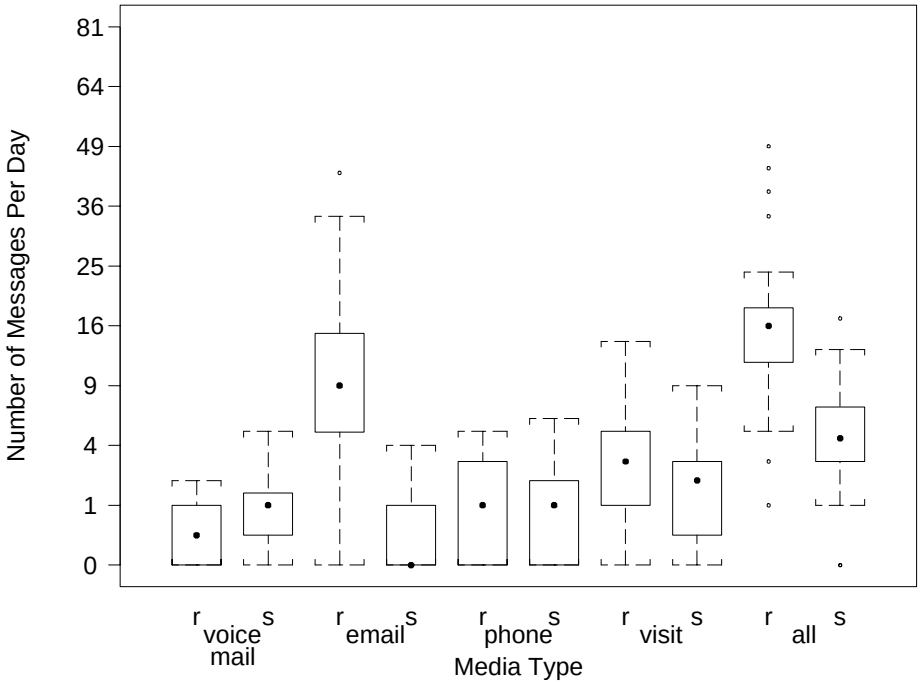


Figure 8 Number of Messages Being Sent and Received Across 4 Media Channels
The number of messages being sent and received, broken down by media channel and whether they were received or initiated by the study subject. We have applied a square root transformation in order to stabilize the variance. Each box contains data on all 7 study subjects across 5 days of observation per individual.

We next examined the number of messages being sent and received each day across the different media channels (Figure 8). Note that the distributions of sent and received visits and phone messages are both approximately normal, reassuring us that the sample is not significantly skewed and also suggesting the presence of reciprocal interactions.[6] As noted in the far right set of boxes, a developer typically received a

total of 16 messages and sent a total of 6 messages during a working day. Ignoring email for the moment, the most ubiquitous form of contact in this work environment was in-person visits. They were used approximately two to three times as often as the other channels.

One of the most surprising results concerned the usage of electronic mail. Many corporations are starting to implement this new form of communication, and we fully expected, given the computer intensive nature of this organization, to see a large amount of email traffic. Although our study subjects *received* many such messages (a median of 9 per day), they sent very few (a median of 0 per day). What's more, the content of these email messages was rarely technical. Most of the traffic was devoted to organizational news (announcements of upcoming talks, recent product sales, or celebratory lunches; congratulatory messages on a "job well done") or process related information (mostly announcements of process changes).

We attribute this phenomenon to several factors. First, it is difficult and time consuming to coherently draft a complex technical question or response. As noted by one developer "Email is too slow; by the time I type out a coherent description of the problem, I could have called or walked over and gotten the answer." Secondly, the ambiguity of software technology may necessitate a type of iterative problem-solving that is ill-suited to the email venue. Our subjects may also have been somewhat reluctant to release a written recommendation or opinion without having control over its final distribution. Finally, the relative maturity of this email system was undoubtedly a factor (it has been in existence for over 10 years). That is, electronic mail *in this development organization* appears to have evolved into something of a broadcast medium. It is the most efficient way for the organization to distribute information to the technical population, but the very fact that such messages have flooded the system makes developers reluctant to use it for pressing technical issues.

Figure 9 plots the duration of messages in each media channel. Looking across all forms of communication, approximately 68% of the interactions are of less than 5 minutes in duration. This agrees with research done in the early 1980's at Xerox Parc [Galegh90]. It also confirms anecdotal evidence supplied by independent studies of this population. The difference in the median duration of a sent versus received visit (approximately 6 versus 3 minutes) is attributed to travel time. Similarly, the fact that sent email messages are of relatively longer duration than those received undoubtedly reflects compositional factors. Not surprisingly, voice mail messages are very brief (1 minute), but phone messages are also unexpectedly short (2-3 minutes). Finally, note the existence of significant outliers in all the media channels; visits of close to one hour and phone calls of 30 minutes are not uncommon. This is a particularly non-intuitive result in that these are all unplanned and unanticipated forms of interaction.

---

6. We do not explicitly track communication threads (a group single problem) here [Wolf93].

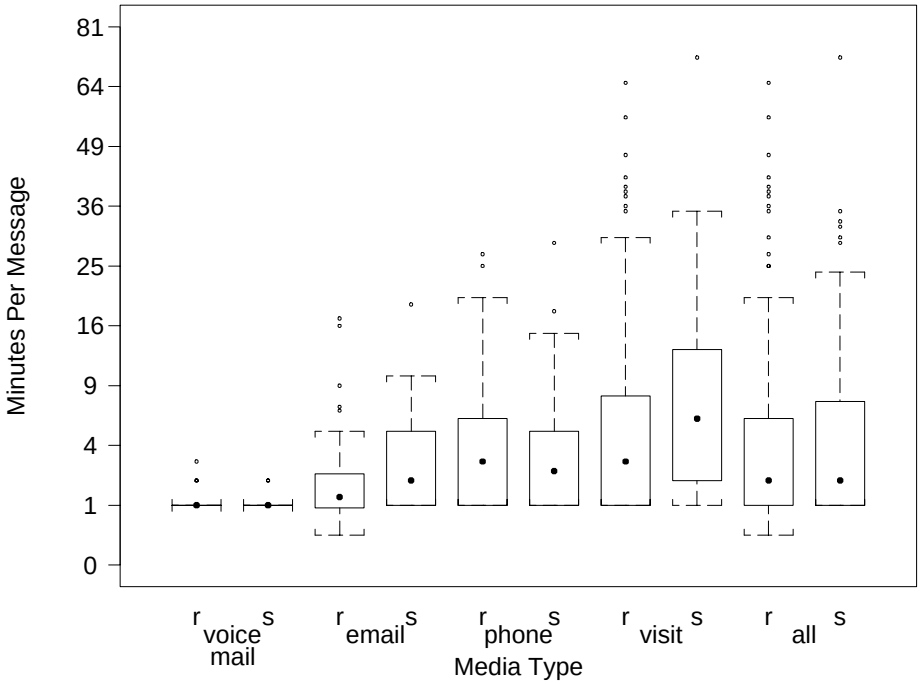


Figure 9 Duration Per Contact By Media Channel

The duration per contact broken down by media channel and whether the message was received or initiated by the study subject. We have applied a square root transformation in order to stabilize the variance. Each box contains data on all 7 study subjects across 5 days of observation per individual.

5. Conclusions

The primary motivation behind this study was to measure and understand various aspects of process intervals in software development. In so doing, we experimented with two novel forms of data collection in the software development field: time diaries and direct observation. We concluded that both methods of research are feasible and useful, depending on the questions one wishes to address.

In addition to exploring new methodology, however, we also sought to investigate under-developed arenas in software research: the social structure, environment and culture of a real organization of software developers. It is our belief that all three elements (organization, process and technology) need to be addressed in order to obtain a complete picture of the development process.

Indeed, we found that elements of the organization were as important as technology, if not more so. In particular, 1) a large percentage of the time was devoted to non-technological aspects of the process cycle, and 2) the data on the number of inter-personal contacts a software developer needs to make during a typical working day strongly suggests that technical problems are not the real issue in this organization. Rather, these software developers need to apply just as much effort and attention to determine *who* to contact within their organization in order to get their work done. Most importantly, we were able to *quantify* what had previously been predominately *qualitative* impressions about life as a software developer in this firm.

The insights that we have gained on the organization that we have been measuring are as follows.

- People are willing to be observed and measured, provided the proper precautions are taken. The subjects of our studies considered it essential to understanding their processes and to improving them.

- Software development is not an isolated type of activity. Even in a most individual-intensive activity such as coding two results belie this myth. First, over half our subjects' time was spent in activities other than coding which are much more interactive. Second, a significant part of their day was spent interacting in various ways with co-workers.
- Progress on a particular development is often impeded for a variety of reasons: reassignment to a higher priority task, waiting for resources, and context switching to maximize individual thoughput.
- There is a high degree of context switching evident in the time diaries. On average, work was performed in 2 hour chunks.
- Time diaries are adequate for their intended level of resolution. What is missing, however, is data on unplanned, transitory events. Direct observation uncovered the fact that about 75 minutes per day is spent in unplanned interpersonal interactions.
- The unique number of personal contacts averages around 7 per day representing continuing interactions; this number can double for certain kinds of activities.
- Direct interpersonal communications are the dominant means of interaction. Electronic mail tended to be used as a broadcast medium in this organization rather than as a means of exchanging technical ideas or acheiving social consensus. This fact is particularly important as much of cooperative work technology presupposes email as the central basis for cooperation.

Acknowledgements

We would like to recognize the extremely hard work of our collaborators who made all these experiments possible: M.G. Bradac, D.O. Knudson, and D. Loge. Professor Tom Allen at MIT brought us together, while Peter Weinberger, Eric Sumner and Gerry Ramage provided the financial support for this work. Professor Marcie Tyre of MIT was particularly helpful, providing extensive input along the way and pointing us in the direction of relevant organizational theory literature. W.J. Infosino's early suggestions regarding experimental design issues and A. Caso's editing of the paper are also much appreciated. Finally, we would like to acknowledge the cooperation, work, and honesty of the study subjects and their colleagues and management for supporting our work with their participation.

References:

[Allen77] T.J. Allen. *Managing the Flow of Technology.* MIT Press, Cambridge, MA, 1977.

[Boehm88] B.W. Boehm and P.N. Papaccio, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering, Vol 14, No 10*, October 1988, pp. 1462 - 1477.

[Bradac93] M.G. Bradac, D.E. Perry and L.G. Votta, "Prototyping a Process Monitoring Experiment," *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, MD, 1993.

[Brooks75] F.P. Brooks, Jr. *The Mythical Man-Month.* Addison-Wesley Publishing Company, Menlo Park, CA, 1975.

[Brooks88] F.P. Brooks, Jr. "Plenary Address: Grasping Reality Through Illusion," *Proceedings of CHI'88,*, May 1988. March 1988.

[Brooks80] R.E. Brooks. "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," *Communications of the ACM, Vol 23, No 4*, April 1980, pp. 207 - 213.

[Curtis86] W. Curtis, "By the Way, Did Anyone Study Any Real Programmers?" *Empirical Studies of Programmers*, E. Soloway and S. Iyengar (eds.) Ablex Publishing Corp., 1986, pp. 256 - 262.

[Cusum91] M.A. Cusumano. *Japan's Software Factories*. Oxford University Press, NY, 1991.

[Galegh90] *Intellectual Teamwork*, J. Galegher, R. E. Kraut and C. Egido (eds.), Erlbaum Publishing Company, NJ, 1990.

[Leves92] N.G. Leveson. "High Pressure Steam Engines and Computer Software," *ACM*, May 15, 1992.

[Shneid86] B. Shneiderman, "Empirical Studies of Programmers: The Territory, Paths and Destinations," *Keynote Address in Soloway and Iyengar*, 1986, pp. 1 - 12.

[Vos84] J. Vosburgh, B. Curtis, R. Wolverton, B Albert, H. Malec, S. Hoben and Y. Liu, "Productivity Factors and Programming Environments," *Proceedings of the 7th International Conference on Software Engineering*, Washington D.C. IEEE Computer Society, 1984, pp.143 - 152.

[Weinb71] G.M. Weinberg. *The Psychology of Computer Programming.* Van Nostrand Reinhold Co, NY, 1971.

[Wolf93] A.L. Wolf and D.S. Rosenblum, "A Study in Software Process Data Capture and Analysis," *Proceedings of the 2nd International Conference on the Software Process (ICSP2)*, Berlin, Germany, February 25 - 26, 1993.