5

Understanding and Improving Time Usage in Software Development

Dewayne E. Perry[†]**, Nancy A. Staudenmayer**[§] **and Lawrence G. Votta, Jr.**[†]

[†]*AT&T Bell Laboratories, USA*
[§]*Massachusetts Institute of Technology Sloan School of Management, USA*

ABSTRACT

Time and motion studies are a proven means toward understanding and improving any engineering enterprise. We believe that the engineering of software processes is no different in this respect; however, the fact that software development yields an intellectual — as opposed to physical — product calls for new and creative measurement techniques.

In attempting to answer the question "Where does time go in software development?" we have been experimenting with several forms of data collection. We have found that two techniques in particular, time diaries and direct observation, are feasible and yield useful information about time utilization. The major source of discrepancy between them is granularity: most software developers can not retrospectively report with a high degree of accuracy the large number of interruptions and unplanned transitory events that typically characterize their working day.

Drawing upon experimental techniques from the behavioral sciences, we used three alternative methods to gather data on engineers developing software in a large organization. We describe the design of our studies, discuss some of the critical issues in creating our designs, and indicate their strengths and weaknesses in order to encourage interested readers to conduct similar studies. We hope other researchers and software developers will thus add to the experimental data on time usage, and will also employ our techniques for calibrating and validating their own metrics.

Process-Centered Environments, Edited by Fuggetta and Wolf

5.1 INTRODUCTION

Time is an extremely important factor in the production of software. It is a significant contributor to costs as well as a significant factor in the ability of an organization to satisfy *time to market* pressures. As a result, a large amount of effort and attention have recently been devoted to decreasing market response time in software production.

Yet, surprisingly, there has been little research on time related behavior at the individual level (the micro level) and on the connection between individual actions and an organization's ability *to act* — that is, the relationships between the micro and macro levels. One possible reason for this deficiency is that researchers lack a basic understanding of how to gather time-related information at the micro level.

We have therefore chosen to focus our research on both the empirical and methodological issues surrounding how time is spent in organizations. Cultural anthropologists such as Hall [13] have argued that the way time is used is a function of culture. This is no less true in an organization. How time is partitioned, scheduled and used across different tasks at the micro level and the role this plays in determining output are a function of the technology, process and organizational culture in which they are embedded. As a result, the allocation of time can greatly influence the performance of an organization and the people in it [23, 29].

The literature yields a history of provocative (and troubling) anecdotes on time utilization during software development. Most visible are the *outcomes* of poor time usage; overshooting project deadlines by a factor of 2 or more has become almost a cliche (or even derigeur). Several prominent authors have written that a significant proportion of project effort is devoted to non-programming activities, with perhaps as much as 50% of the work week absorbed by machine downtime, meetings, paperwork, company business, sick and personal days [4, 7, 22]. A close examination of the references, however, shows that most of these claims are based on one unpublished 1964 dissertation by E.F. Bairdain on how software developers spend their time [22]. Details of the sample and methods used to generate these findings are not readily available.

The work that is most closely related to ours is that of Wolf and Rosenblum [33]. Using an event-based approach to modeling software processes, they employ a hybrid process capture technique (exploiting both manual and automatic methods) and develop a retrospective (statistical) technique to analyze the time intervals provided by the captured process data. They hypothesize, as we do, "... that process problems ultimately lead to wasted intervals of time" and that "... the causes of wasted time are best revealed by restrospective analysis of characteristic time intervals." While their manual process capture technique is both costly and labor intensive, they provide an objective view of the dynamic aspects of software processes.

Our study is therefore part of an on-going effort to understand what professional software developers *actually* do as opposed to what they say they do or are thought to do [32]. We focus on the use of *time* as a critical yet under-examined aspect of life in software organizations, and we explore alternative ways to gather that data.

Our strategy has been to do a series of experiments. The purposes of these experiments were to test prior findings, to calibrate our current instrumentation, to build an underlying theory, and to investigate newly generated hypotheses. In each case, we built on what we had learned and designed the subsequent experiment to substantiate our prior findings, to clarify what we did not understand, or to explore the limitations of our prior findings. At the end of our series of studies, we had a much more complete picture of time usage in this organization. We believe that such a series of experiments is essential to understand an issue fully. Certainly,

our recommendations for the management of software development in this firm would have been different (and probably detrimental) if we had based them only on information yielded by the first experiment.

We employed three complementary experimental methods to investigate time usage on software projects. In the first experiment we analyzed data on one software developer's daily activities over a 3 year period as recorded in a retrospective diary. We refer to this as an *archeological* study. [1] We then sought to validate our small sample findings (n=1 person, but many observations over time) by collecting daily activity information on a larger sample of subjects. In this second study, referred to as the *daily time diary* experiment, we designed a modified time card and asked software developers to record their daily activities. These two studies are described in full detail by Bradac, Perry, and Votta [5] and by Perry, Staudenmayer, and Votta [27, 26]. Finally, although periodic interviews and occasional unannounced visits had convinced us that no conscious misrepresentation occurred, we conducted a third study to check the reports of time usage via direct observation [26].

In the rest of this paper, we will describe these experiments in detail and show how they build on each other and interlock to build a credible (that is, a believable) story of micro time usage in a software development organization.

5.2 RESEARCH QUESTIONS

The initial, and most general, research questions that motivated our work are simply stated: How is time spent and why does software development take so long? [2] To address these questions and to gain initial data and understanding of the specific subject processes, we used the low cost and low resolution archeological approach of reconstructing the history of a specific developer's assignment [5]. We discovered that much time was spent either doing nothing at all or something else.

Our second set of research questions was aimed at determining the mechanisms by which these initial results were produced, and the validity of these results: were the results representative of other developer's assignments, and what are the limits of the archeological approach? Specifically, we had to answer two questions:

1. How much time is spent blocked (and why); how much time is spent working on something else (and why); and how is time divided between work and rework?
2. Do the patterns of progress we saw in the single developer's diary hold for multiple developers in multiple development projects?

To address these questions we performed a detailed cross-sectional study of multiple developments and multiple developers using daily time diaries with a finer degree of resolution than we used in the archeological approach. We discovered that the general pattern of mixed progress

[1] The analogy of assembling process enactment data from previous projects is important and in many ways similar to archeology.

[2] In the early 1990s, the executive management of this corporation identified time to market as one of the key metrics for success in their competitive marketplace [9]. This realization led to an effort throughout the software development organization to significantly improve their development interval. Among the various foci of interval reduction were the build process, the modification process, and determining where time was spent in the development cycle. As part of this improvement effort, development invited research to collaborate on various strands of complimentary investigations. Our work and that of Wolf and Rosenblum [33] began in response to the challenges offered by development's need to improve their development interval and to understand how time was spent in their organization.

that we had observed in the archeological study is reflected in a more general population. In particular, the ratio of development elapsed time to race time is the same, but the underlying reasons are different [27].

Our third set of questions assessed the validity of the time dairy approach and its strengths and weaknesses. The specific research questions we sought to address via direct observation were as follows:

1. To what extent are the daily retrospective self-reports an accurate representation of what actually happened during the developer's day? What are the sources and extent of bias? What are the strengths and weaknesses of the two methodologies?
2. What types of events are *not* being captured in the self-reports and how significant are they?

To address these questions, we performed detailed direct observations and correlated them with the self-reports. We discovered that the time diaries are accurate at their level of representation but do not reveal the degree and extent of short planned and unplanned events such as interruptions and casual communication.

Finally, a set of general research questions arise from our set of experiments:

1. How do we reconcile findings from archeology, diaries, and observation?
2. How do we arrive at a more holistic view of the process?

We expand on these questions, approaches and results in the subsequent sections.

5.3 EXPERIMENTAL DESIGN AND ANALYSIS

In this section, we start by providing some background on the setting and software system we were studying. Figure 5.1 depicts the relationship between the three different experiments we performed. A theme of this work is that we needed a series of experiments to understand what was really happening to time usage when this organization developed software.

We then proceed to describe each of the experiments in turn, emphasizing the advantages and disadvantages of each approach, the conclusions we drew from each, and how we designed the subsequent study to test our hypotheses.

5.3.1 Experimental Setting

The subjects for our time studies build software for a real time switching system [9]. They have at least an MS degree in either computer science or electrical engineering, more than 5 years industrial experience, and similar responsibilities and seniority. 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 about every 15 months.

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

The development organization responsible for product development consists of about 3,000 software developers and 500 hardware developers. This software development organization is currently registered to ISO 9001 standards [15] and has been assessed as an SEI level 2 development organization [14].

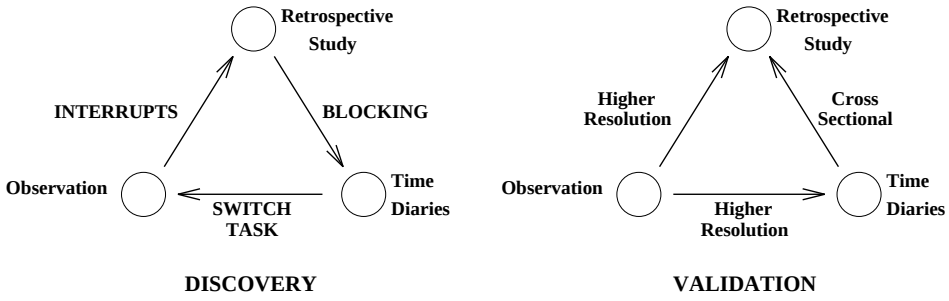


Figure 5.1 Summary of Time Study Experiments. The figure depicts the discovery and validation relationships between the three different experiments described in this paper. The discovery circuit begins with the identification of elapsed time added to the development interval due to blocking; this hypothesis is refined via the cross sectional study where we realized that the developer switches tasks when blocked, and finally confirming through the observation experiment. This experiment also uncovered the significant amount of communication interrupts that each developer must cope with. The validation circuit shows the two dimensions along which we refined our understanding of time usage: improving the time resolution and using both cross sectional and longitudinal designs.

5.3.2 The Archaeology Study

The initial motivation for this study was an earlier internal study focused on understanding the elapsed time intervals of different processes. This concern for process intervals led us to consider time and motion instruments as an experimental means to understand process intervals.

The archaeological study was a retrospective longitudinal study of one developer's time spent developing one feature. (See [5] for a full description of this study.) The data was reconstructed from personal diaries and the project notebooks, and consisted of the task that was done and what the status of that task was for each day of the 900 consecutive days of the feature development. The data depicted the most important task for each day for the lead developer on the feature development. It should be noted that the personal diary was not intended for study but was kept for personal use. This fact leads to several disadvantages: first, there is no control over the degree of granularity reported, and second, the definition of what was important was up to the developer.

On the basis of the diary and project notebooks, we were able to identify — at the relatively crude resolution of daily activity —how time was spent. Moreover, we learned that blocking was an important factor in determining why developments may take as long as they do in this project and organizational structure.

However, the diary and notebooks were not sufficient to develop a full understanding of how time was spent. First, the resolution of the data prevented us from understanding fully the reasons for the blocking and what the developer did when he was blocked. When the blocking lasted for the relatively long time of a week or more, the project and personal notes were sufficient to give a reasonable explanation. However, when the blocking only lasted for a day, as was typical for the tasks of design, coding, and testing, the notes were not sufficient to understand the phenomena. Second, since the data was focused on the most important work for each day, we did not know how the rest of the time was spent. Finally, only certain

types of individuals are meticulous enough to keep dairies at the required level of detail. This self-selection aspect raises the issue of how representative the study is, independent of the fact that it is a study based on only one person.

Despite the drawbacks of an archeological study such as this one, this approach constituted a useful starting point for time-studies. A wealth of data exists in organizations that is readily available for study (both non-intrusively and inexpensively) that researchers can use to explore new questions and to construct new hypotheses. An archeological study like this is a useful place to begin an iterative cycle of experiments such as the cycle reported in this paper [31].

5.3.3 Daily Time Diary Study

In response to the shortcomings of the archaeological study, we decided to investigate with finer resolution what was happening in the design, coding, and testing phases of several features being developed by several developers. Thus, for the second in this series of studies, we performed a cross-sectional, multi-feature, multi-developer time study using personal time dairies filled out at the end of each day. We designed the personal time diary instrument in cooperation with the subjects to improve recording accuracy while minimizing overhead to the subject. This led to an instrument accurate to about one-half hour per day, while requiring less than 5 minutes per day of the subject's time to record the information.

From this study, we learned that blocking is not wasted time for the developer, since developers typically maintain a two-feature workload and context switch from one to the other when blocked. However, when the most important feature is blocked, time elapses even if it is utilized productively on another feature. We determined that the ratio of race time to elapsed time is a very important metric [27] for the enactment of a feature development process, and that in this organizational and project structure the ratio is 2.5. One further important result is the amount of context switching that occurs, even when measured at this relatively gross grain [26].

This study still had certain shortcomings that were now focused more on the consistency of the subjects' recording of time. For example, a subject may unintentionally forget significant events when under the pressure of production or may characterize the same events in different ways (where they are considered important in one case but not in another). H. Russell Bernard, *et al.*, gives an excellent summary of informant accuracy research [2].

5.3.4 Direct Observation Study

Five software developers were chosen at random from the group participating in the self-reporting experiment. The remaining 10 developers from the self-reporting experiment served as one control group, allowing us to assess the impact of observing on self-reporting. We also had nine months of prior diary reports on these two groups, which could be used for comparison with post-observation entries. Two software developers who were *not* part of the self-reporting experiment were also included in the observation experiment as an additional form of control: comparison with the other subjects enabled us to assess the impact of self-reporting, given observation.

In summary, our sample consisted of one treatment and two control groups:

1. Treatment Group 1 contained subjects who had participated in the self-reporting experiments for 9 months. They continued to complete their time diaries and were simultaneously observed.

	Variable Y—(Observation Type)		
	Full Day Observing (Y = 1)	**One Hour With Questions (Y = 2)**	**One Hour Without Questions (Y = 3)**
Variable X (Observation Predictability)			
Scheduled (X = 1)	**X1Y1**	**none**	**none**
Unscheduled (X = 2)	**X2Y1**	**X2Y2**	**X2Y3**

Figure 5.2 2 × 3 Partial Factorial Design for Observing Software Developers. The figure displays the independent variables of the original experiment definition. This was later simplified by using only full day observations, for reasons described in the text. **X** represents the independent variable of *observation predictability*: scheduled and unscheduled by the subject. **Y** represents the *observation types*: a full day, snapshot with questions, and snapshot only. The original design is partial because no scheduled observations involving snapshots, either with or without questions, were performed.

2. Control Group 1 contained subjects who had participated in the self-reporting experiments for 9 months. They continued to fill out their diaries but were not observed.
3. Control Group 2 contained subjects who had not participated in the self-reporting experiments but were observed.

5.3.4.1 Case Study Design

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 developer's behavior. We combined elements from three standard behavioral science experimental designs: (a) a partial factorial design, (b) a repeated measure design, and (c) a replicated interrupted time series design [16]. Figure 5.2 depicts the 2 × 3 partial factorial design. The two independent variables are *observation predictability* and *observation type*. The design is partial because we collected data in only four of the six possible cells.

The intent behind the first variable (observation predictability) was twofold. First, we hoped that by observing at random we would hinder the study subjects from adjusting their schedules so as to favorably impress the researcher. Second, we sought to give the study subjects some sense of control over the experience. Prior research in psychology has confirmed the notion that having control over one's environment (or the perception thereof) produces physical and psychological well-being [20].

The use of multiple controls was deliberate and deserves further explanation. We were

particularly 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.[3] We therefore built in several alternative control mechanisms in order to assess the significance of these possible distortions. This included both standard hold-out samples (Control groups 1 and 2) as well as features of the experimental design. In particular, by observing our subjects more than once under each condition (repeated measurements), each subject served as his/her own control. The replicated interrupted time series aspect of the design further allowed us to compare the same subject over time and different subjects at the same time. This enabled us to assess potential threats to validity posed by maturation, history and testing. Finally, as noted earlier, we had extensive pre-observation self-reported data to compare with that which was reported after the observation experiment had begun.

The original design was quickly modified to only a full day observation type (collapsed into a 2×3 design), for three reasons: (1) the subjects were in three different locations, creating excessive travel costs for the observer; (2) the arrival of an observer in the middle of the day created too many awkward warm up periods and thus unduly interfered with on-going work; and (3) it was often necessary to observe an entire day to properly interpret a single event and the subject's response to it.

Each subject was observed a total of five full days (9-10 hours per day, on average, for a total of 315 to 350 hours of observation over a 12 week period). Two of the five days were chosen and scheduled by the subject.[4] The remaining three days were assigned by random draw without replacement, and the subjects were not informed of when they would occur.[5]

Alternative methods do exist, and we considered using an observer with a video camera instead of an observer taking notes. Although there are precedents for using video cameras [11, 12], we felt it would be inappropriate. Our study population was not used to being observed, and although the subjects were receptive to the notion of participating in an experiment, and quickly became comfortable with the observer, we felt the introduction of video equipment would have distorted their behavior (not to mention that of their peers and overall work progress). Also over 300 hours of video tape would have to be watched and interpreted, significantly increasing the cost and duration of the experiment.

Since some people might be uncomfortable with the prospect of being observed, we spent considerable time beforehand explaining the purpose of the study to the subjects. We introduced the observer as a student — there to learn how the subject spends his/her time when doing software development. The subjects 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).

The observer tried to function as a "human camera," recording everything with as few initial preconceptions about relative importance as possible. She was also required to 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 [16].

We used continuous real time recording for non-verbal behavior and for interpersonal

[3] See Parson's article for a modern interpretation of the "Hawthorne Effect". The author discusses the continuing misinterpretation of the Hawthorne experiments in organizational studies [25].

[4] 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.

[5] 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 in the event that a developer did not come in to work.

Factor	Effect on Experiment		
	Retrospective Study	Time Diaries	Observation
Cost	inexpensive	moderate	expensive
Impact on Measured System	none	moderate	great
Dominant Source of Bias	informer	subject	observer

Table 5.1 Comparison of Different Experimental Techniques This table compares our 3 experimental techniques along 3 dimensions. One clear tradeoff is the higher the resolution (as reflected by moving to the right across the table), the greater the potential for disturbing the system. A similar tradeoff exists between resolution and cost. We sought to insure that we were not changing what we were measuring while we were measuring it while simultaneously staying within reasonable boundaries in terms of cost and accuracy.

interactions. 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, one for each subject.

Such an inductive method, involving the joint collection, coding, and analysis of data, has been cited as one of the best ways to discover theory from data [10].

5.3.4.2 Miscellaneous Remarks About Observing

Almost without exception, the first response to this final study was "You just want to observe me? There's nothing to see" (from subjects) or "How boring" (from fellow researchers). In fact, nothing could be further from the truth. As described below, a software developer's day was filled with events and activities that varied considerably across individuals and time. The observer accompanied the subjects nearly everywhere (e.g., to group and department meetings, affirmative action presentations, tutorials, lab sessions) and witnessed a broad spectrum of events (promotions, mental and physical exhaustion, celebrations, etc.). In addition to observing, there were plenty of opportunities to talk with the developers and their colleagues about life in the organization, and we found them to be remarkably willing to do so.

5.3.5 Engineering Considerations for Three Different Time Studies

Each of the three time studies performed as part of this research can be characterized along four dimensions: cost, resolution, the potential intrusiveness of the measurement process and the biases associated with those measurements. Table 5.1 compares the studies along these dimensions and highlights the tradeoffs within a given study. In particular, note that the higher the resolution, the greater the cost and potential intrusiveness.

5.4 CALIBRATION ANALYSIS

In any experiment, the researcher must understand the sources of variance in the data in order to build a probability model of the experiment and answer the question, "What is the probability of being wrong?" This enables other researchers to reproduce the experiments and to interpret

DEVELOPER'S SUMMARY		OBSERVER'S SUMMARY NOTES	
0800 - 1800	High Level Design Work	0800 - 0900	Administration
		0900 - 1010	Analyze high level design
		1010 - 1021	Break
		1021 - 1135	Code experiment with peer
		1135 - 1226	Write high level design
		1226 - 1314	Lunch in cafeteria
		1314 - 1330	Answer document question
		1330 - 1349	Answer growth question
		1349 - 1406	Reading Results of Survey
		1406 - 1500	Code experiment with peer
		1500 - 1626	Search and read paper
		1626 - 1701	Code experiment with peer
		1701 - 1705	Administration

Table 5.2 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; about 55 minutes. The diary contains one entry for this 9-10 hour day "High Level Design Work." The observer had 13 entries of which about 5 hours corresponded to activities associated with high level design.

agreements or disagreements. To make the software development process more efficient, we need to gather data on how long it takes developers to perform given tasks, and we need to know how reliable this data is. The sources and properties of variance are also important in their own right [6].

5.4.1 Description of Data

To calibrate software developers' assessment of their own usage of time, we compared two sets of data — the self-reported data and the observer's notes. Table 5.2 displays one comparison sheet, with a software developer's report on the left (see example in Appendix B) and our summarized observations on the right. The raw data recorded by the observer contained an impressive amount of micro-level detail, often down to three minute intervals.[6] To make an intelligible comparison, we summarized that detail into major blocks of activities.[7] In doing so, we ignored many transitory, short duration events which the developers had to initiate and respond to during the course of a working day. Those types of interactions are analyzed separately in Section 5.5.

[6] Wolf and Rosenblum note that an appropriate level of granularity is very important and have designed and used an alternative direct observation method that is optimized for capturing and correlating short duration events with the actions of people [33].

[7] We 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 as represented by the Cronbach alpha statistic.

5.4.2 Calibrating Observations with Self-Reports

In the calibration analysis we considered the accuracy of the self-reports in terms of (1) *time elapsed* and (2) *what actually happened during that time.* We then proceeded to build a correction factor for the individual self-reports.

Our error model for the self-reports has two components. The first we call **normalization**. Figure 5.3 plots the total working time (in minutes) each software developer reported versus that observed. The diagonal line represents perfect agreement between observer and subject; those points falling above this line represent observations in which a subject actually worked more hours than he/she reported (under-reporting). Those points below the line mark instances where a developer actually worked less time than he/she reported (over-reporting). As seen in figure 5.3 , the majority of the observations are clustered around 500 to 550 minutes or 8.3 to 9.2 working hours.[8] An obvious exception is subject 2A, who twice worked more than 11 hours. He is an acknowledged local expert often called upon to help solve critical issues in the lab. Note that most subjects are consistent in the accuracy of their reporting. Subject 2B, for example, tends to over-represent total working time by about one hour whereas subjects 1B and 1C are remarkably accurate.

The average amount of over-reported working time — the amount by which the subject's reported work time exceeded that recorded by the observer — was 2.8%.

The second component of the error model is the **fidelity** of the self-reports. If we were to reconcile one of the subject's self-reported list of activities on a given day with that actually observed, to what extent would the the two viewpoints agree? We obtained such a measure by dividing the number of minutes that a subject and the observer agreed about what the developer was actually doing by the total number of minutes worked that day (as recorded by the observer).

Figure 5.4 displays that fraction plotted as a function of the date the subject was observed. The vertical line segments represent the confidence bounds around each ratio.[9] The agreement between the observer and subjects ranged from 0.95 to 0.58 for subjects 1B and 2B respectively. The clusters further indicate that the variation between subjects is greater than the variation within any one subject's set of observations.

Analysis of variance in the agreement rate between the subject's time diary and observer's entries produced the results shown in Table 5.3. The observation index has no explanatory power; therefore, the data appears to be independent of observation sequence. However, there is a significant difference between subjects. Each subject on different days is much more consistent in his/her fidelity rate than are two different subjects. This result is consistent with the observations of many researchers that there always seems to be large variation between software developers in performing any specific tasks.

Of course, some days are more eventful than others. But on average, as displayed in Figure 5.5, a developer's day can be segmented into about 12 different activity blocks (from the observer's perspective). The developers, however, varied in the level of detail they applied to their diaries. Subject 2B, for example, always noted one major activity for the entire day

[8] These calculations merely compare begin/end times; we have not subtracted out breaks or lunches from the totals. In general, there was a strong correlation in the reliability of all such measures; i.e., a developer who accurately reported total time and activity also tended to accurately record the number and duration of his breaks.

[9] Two independent researchers compared each of the activity blocks noted by the observer with the subject's diary entries that day and assigned each such block to one of three categories: agree, disagree, or maybe. The length of the line segment is the amount of time in the day where we could not definitively decide whether there was agreement or disagreement between the two reports.

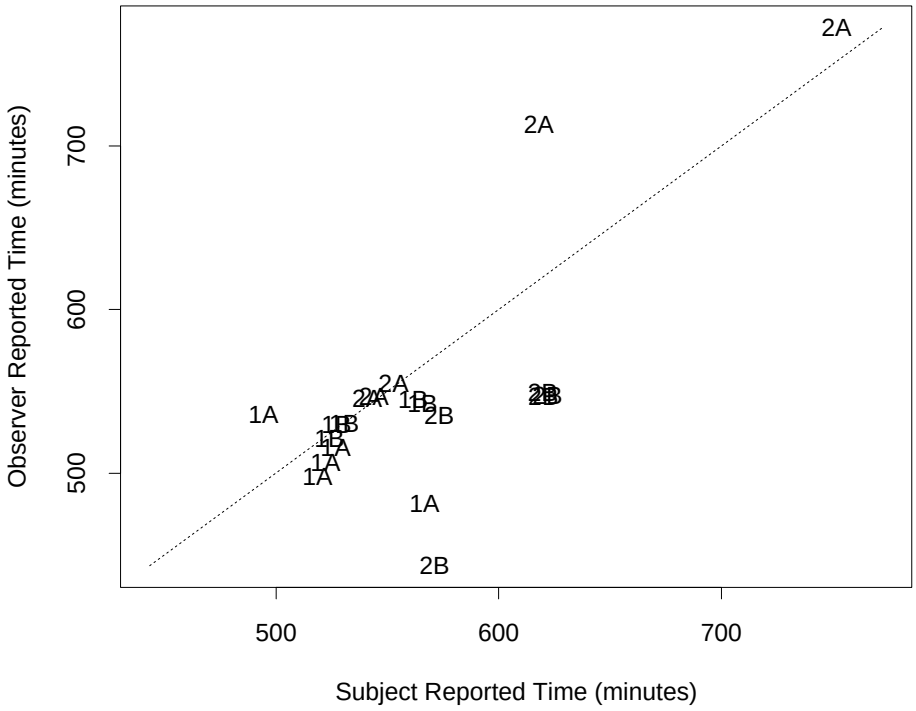


Figure 5.3 Comparison of Reports of Total Working Time. This plot compares each software developer's self-reported total time at work (in minutes) with that actually observed. Each developer is identified by a code (e.g., **2A**), and 5 data points (observation days) are plotted per individual. The diagonal line indicates perfect agreement between observer and subject. Subjects are said to under (over) report when their reported times are less (greater) than that observed, and the corresponding data points fall above (below) the line. For example, on two occasions subject **2A** under reported his work time. The average amount of over-represented working time was 2.8%.

whereas 1B's and 1C's level of detail approximated that of the observer. Interestingly, the time questionnaires also indicated that 1B and 1C were the most monochronic[10] of the study subjects.

We found that the more diary entries the subject made per day, the greater the agreement between the observer and subject. Figure 5.5 compares the number of entries made by a given subject with the number of daily activity blocks extracted from the observation notes for each day of observation.

When we examined the content of what developers were or were not reporting, the major source of variation was the large number of unexpected events that occurred during the course of a developer's day. (See, for example, the two afternoon entries on the right hand side of Table 5.2). Although most developers did not record them in their retrospective reports,

[10] A person is considered to be monochronic if they prefer to do only one thing at a time [29].

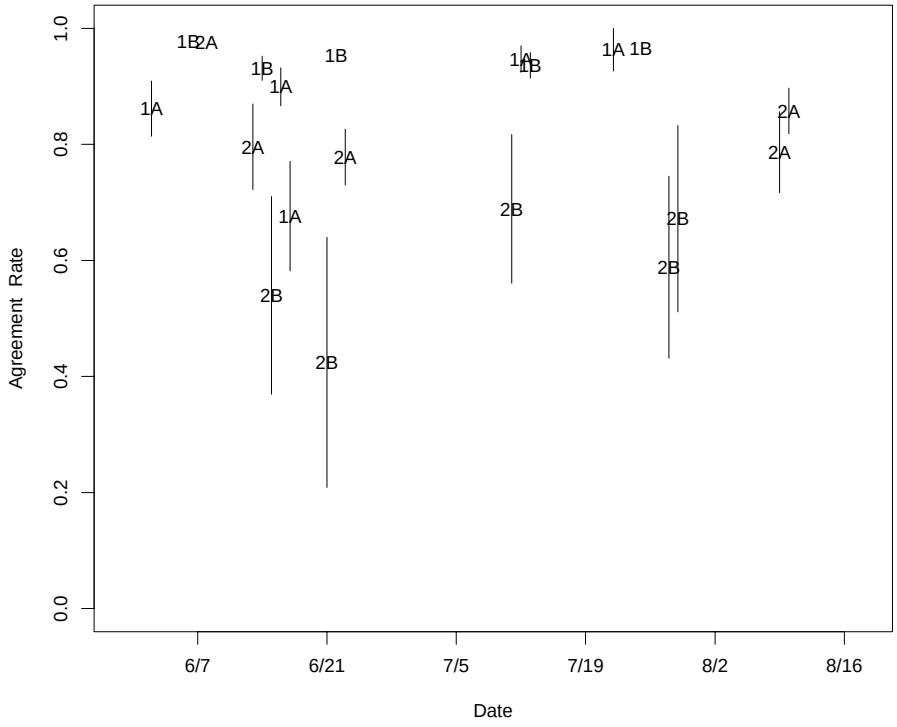


Figure 5.4 Rate of Agreement Between Observer and Subjects. The fidelity rate, calculated as the number of minutes that the subject and observer agree about what the subject was actually doing divided by the total number of minutes worked that day, is plotted by the date of observation. The vertical lines represent the confidence bounds around each estimate. The fidelity varies between 0.95 and 0.58, and the between-subject variance is clearly greater than the within-subject variance.

such interruptions were an ubiquitous part of the development process from the observers' perspective. In the next section, we use the observation data to quantify this qualitative impression that developers are frequently interrupted.

5.5 COMMUNICATION

Why did the observer and subject disagree? Figure 5.5 suggests one source of disagreement — the relationship between the fidelity rate and the number of observer and subject entries. As indicated above, one type of activity that subjects frequently failed to record was short unplanned interruptions. In this section we outline the analysis that led us to conclude that communication often requires interruption especially when a schedule deadline is imminent.

source of variation	sum of squares	degrees of freedom	ratio	proba-bility
average	13.20	1	NA	NA
subject	.39	3	15.67	$< .01$
obs index	.03	4	.88	$< .5$
residuals	.10	12	NA	N
total	13.72	20	NA	N

Table 5.3 Analysis of Variance Table. The table shows the analysis of variance for observation index treatments and subjects associated as blocks. The probability of <.01 suggest that for less than 1% of the time we would expect to see a ratio of 15.67 or greater, if no dependence exists. We can therefore reject the null hypothesis and say that the fidelity rates vary more between subjects than they do between days for the same subjects.

5.5.1 Description of Data

Figure 5.6 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).[11] Drawing on the methodology of Wolf and Rosenblum [33] we have categorized each interaction according to whether it was sent or received by the study subject.

The form and content of the interactions recorded here are easily recognized by anyone who has worked in a large corporate setting. Perhaps best described as "on-the-fly" exchanges [19], they usually involve little formal preparation and little reliance on pre-written documentation diagrams or notes.[12] For example, a developer often received a call from the lab about a testing problem that needed immediate attention, or he 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 about 2 months earlier received, on average, one call a day from his former group.) Finally, there existed much unplanned interaction with colleagues: requests to informally review code, questions about a particular tool, or general problem solving and debriefing.

5.5.2 Observations

How many unique people do these programmers interact with during a typical working day? Figure 5.7 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.

[11] 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.

[12] Note that we did not include contacts made in (scheduled) meetings or in the system test laboratory; nor did we include purely social exchanges (e.g., a call to a wife/husband, lunch partners). The unique count does not include contact with anonymous administrators.

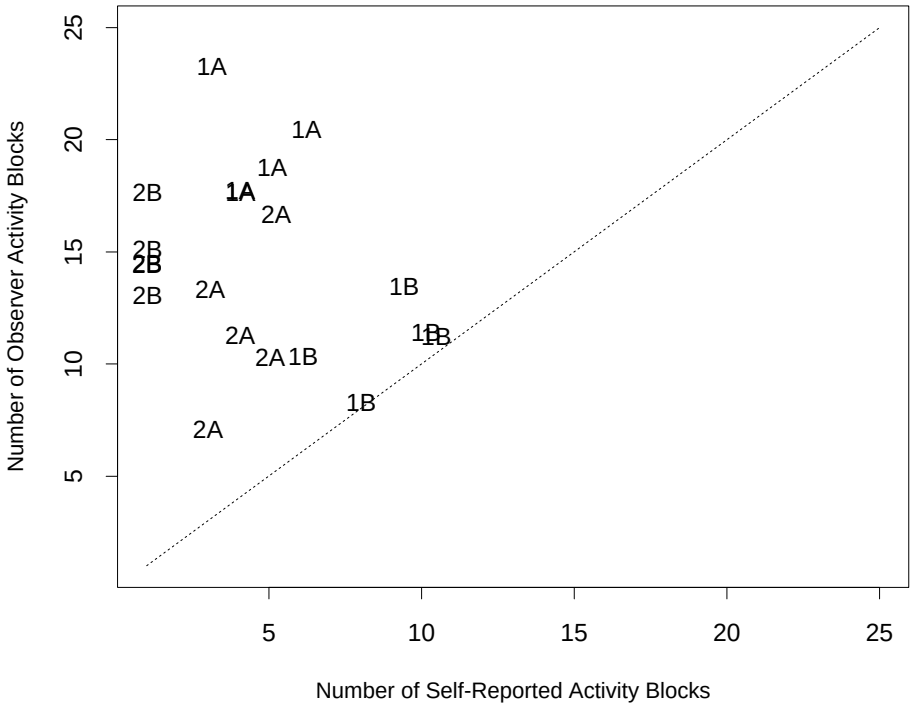


Figure 5.5 Comparison of Number of Entries. This plot compares the number of major activity blocks observed versus the number of diary entries recorded by each developer. We found that the closer the ratio of the two numbers was to 1 (represented by the diagonal line), the higher the *fidelity* of the self-reporting.

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 [8]. As depicted by the far right boxplot, the median number of unique contacts, across all study subjects, was seven per day. The median number of different people interacted with per day is already straining what is widely believed to be people's cognitive limitation [24].

Subject 2D stands out in contrast to the rest of the sample with a median number of 11 unique daily contacts. We attribute this to office layout (he shared space with two other people

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 5.6 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.

and their local coffee machine drew a lot of traffic).[13] This subject's Meyers-Briggs [17] personality profile also indicated a strong preference for social interaction.

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 contacts 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 (a) calls to a help desk for passwords or information about a particular release of the software; (b) calls to the lab requesting available time slots for testing; and (c) exchanges with peers about process procedures in general. Note that these contacts seldom involved requests for technical help. That is, asking the solution to a problem was usually not the reason for the call. Rather, the developers needed help *implementing* the solution.

We next examined the number of messages being sent and received each day across the different media channels. That is, how many total interactions the unique contacts described above generated. (See Figure 5.8.) 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.[14] 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 common form

[13] What, in general, is the impact of having an office mate? Three of our subjects shared an office with a single colleague in the same work group, three subjects had private offices and the remaining individual shared an office with two peers. Prior studies have cited an 8-11% productivity gain with private offices [3], and although we cannot assess such a claim here, the presence of a close peer is definitely an important driver of interruptions. However, the presence of a room-mate often gives quick access to sanity checks, or, in the case of a new employee, the more experienced room-mate often acts as a mentor.

[14] We do not here explicitly track communication threads (a group of related communication events all devoted to the discussion of a single problem), but that would be another interesting metric to calculate [33].

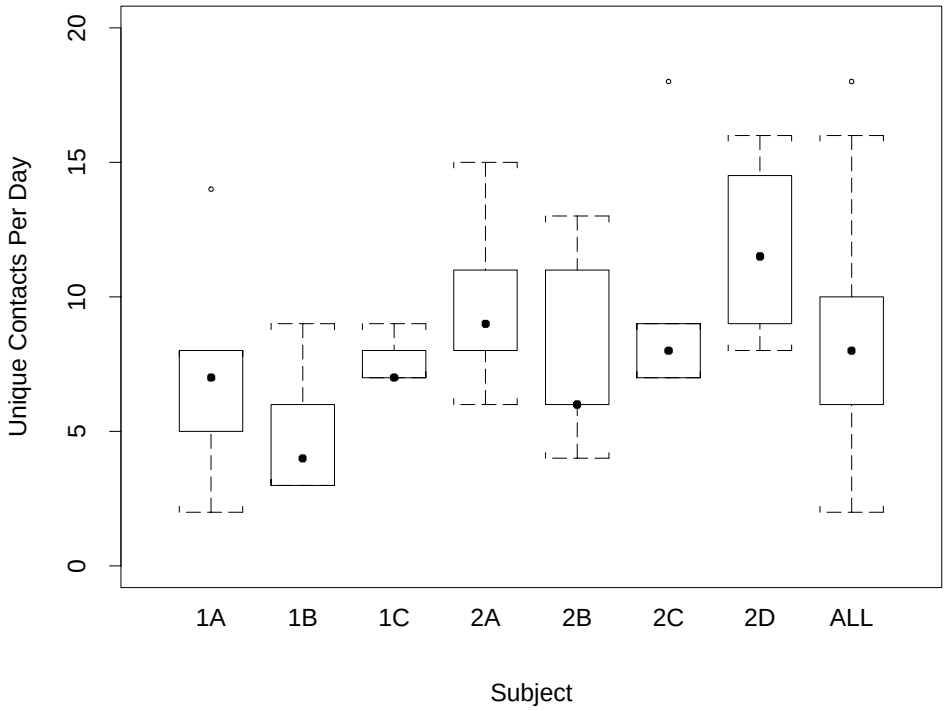


Figure 5.7 Number of Unique Contacts Per Subject Per Day. Dots indicate the median number of unique person contacts for each subject per day. This count reflects interactions across four media channels (voice mail, email, phone and in-person visits) but does not include contacts made during meetings or lab testing. Nor does it include social exchanges. The median over all these subjects is 7 (last boxplot). The outliers primarily reflect days in which a subject was working on modification to existing code.

of contact in this work environment was personal visits. They occurred two to three times as often as communication via the other channels.

One of the most surprising results concerned electronic mail. Given the computer intensive nature of this organization we fully expected, to see a large amount of e-mail 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 e-mail messages was rarely technical. Most of the traffic was devoted to organizational news (announcements of upcoming talks, recent product sales, celebration lunches, and congratulatory messages on a "job well done") or process related information (mostly announcements of process changes!)

We attribute this to several factors. First, it is often difficult and time consuming to draft a clear technical question or response. As noted by one developer "E-mail 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." Second, the ambiguity of software technology often necessitates a type of

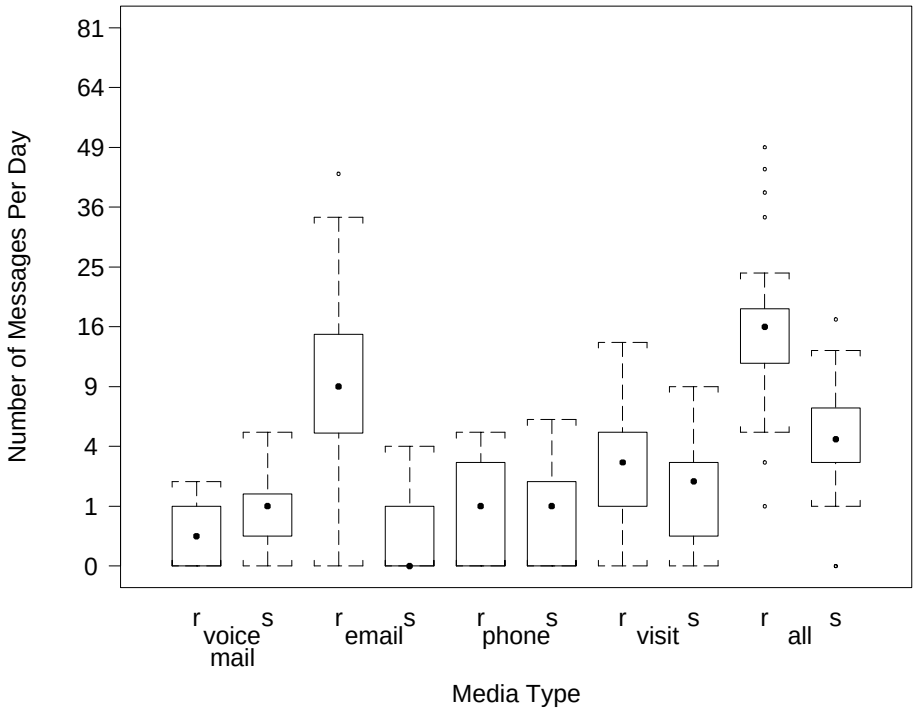


Figure 5.8 Number of Messages Being Sent and Received Across 4 Media Channels. Boxplots show the number of messages, organized by media channel and by whether they were received or initiated by the study subject. We have applied a square root transformation in order to stabilize the variance. Each box contains data on all 7 study subjects across 5 days of observation per individual.

iterative problem-solving that is ill-suited to e-mail [28, 30]. Our subjects may also have been reluctant to release a written recommendation or opinion without having control over its final distribution. Finally, electronic mail *in this development organization* appears to have evolved into something of a broadcast medium. It is the most efficient way to distribute information to everyone, but the fact that such general messages have flooded the system makes developers reluctant to use it for pressing technical issues.

Figure 5.9 plots the duration of messages in each medium. Across all forms of communication, about 68% lasted less than 5 minutes. This agrees with research done in the early 1980's at Xerox Parc [1]. It also confirms anecdotal evidence supplied by independent studies of this population [18]. The difference in the median duration of sent versus received personal visits (about 6 versus 3 minutes) is attributed to travel time. Similarly, the fact that sent e-mail messages were of longer duration than those received undoubtedly reflects composition time. 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; visits of close to one hour and phone calls of 30 minutes are not uncommon. This is

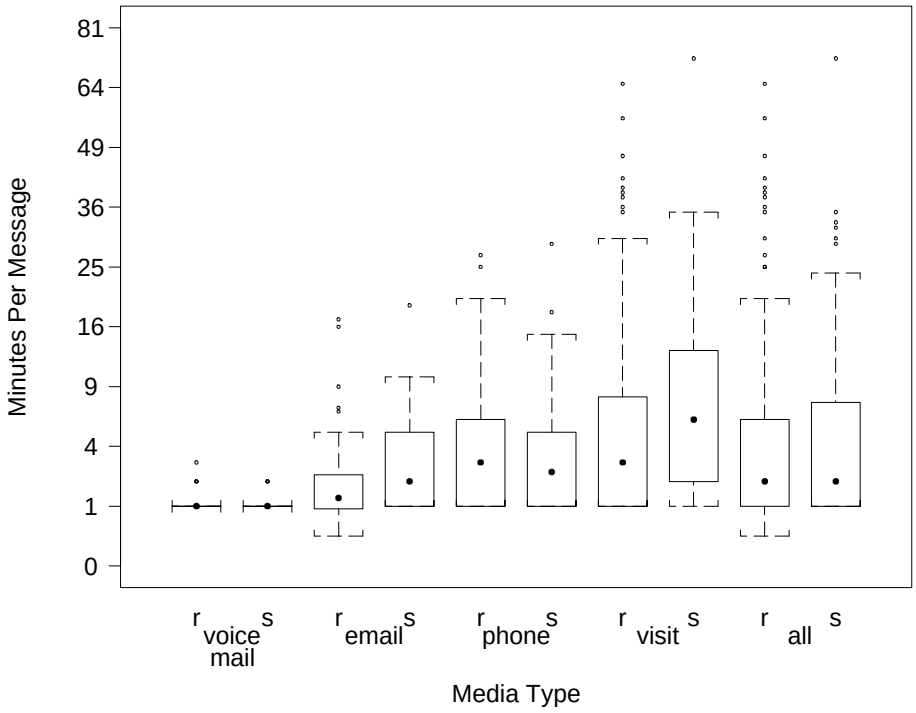


Figure 5.9 Duration Per Contact By Medium. Boxplot shows the duration per contact by medium and by whether the message was received or sent by the study subject. We have applied a square root transformation in order to stabilize the variance. Each box contains data on all 7 study subjects across 5 days of observation per individual.

a particularly non-intuitive result in that these are all unplanned and unanticipated forms of interaction.

Finally in Figure 5.10 , we have plotted the total time spent communicating. The median total time spent using all four communication vehicles is about 75 minutes per day (50 minutes receiving and 25 minutes sending). Of that total, about 35 minutes is occupied by face-to-face interactions. Note, however, that the variances associated with in-person visits and electronic mail are especially large. We attribute this result to the fact that in-person technical discussions often digress to include social topics.[15] Similarly, reading e-mail seems to be a form of communication that easily expands to fill the time allocated to it; something a developer might do at the end of the day or to kill time before lunch.

[15] Although we did not observe a tremendous amount of socializing, exchanges often combined technical and personal communication.

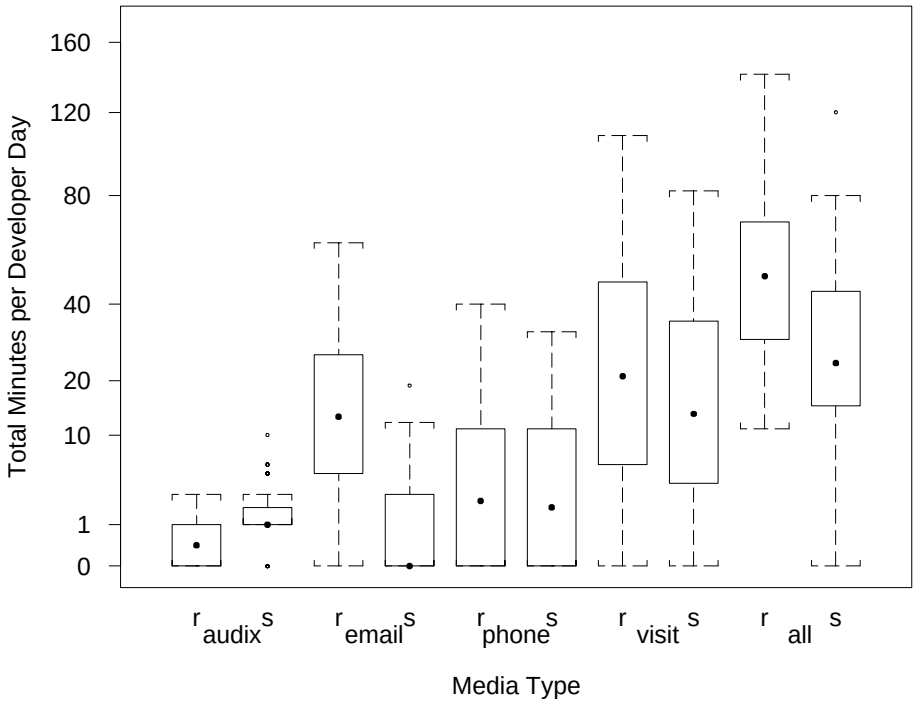


Figure 5.10 Total Time Devoted to Each Medium Boxplots show the number of minutes per day spent using each medium. Again, we have applied a square root transformation in order to stabilize the variance. The total time spent in communication each day is about 75 minutes. Of particular significance are the imbalance in total time spent receiving versus sending e-mail and the large variances in time spent using e-mail and visiting.

5.6 CONCLUSIONS

The primary motivation behind this research was to answer the question: "How is time used during the development of software products?" During the course of this investigation, we experimented with several methods of data collection: archeology, time diaries and direct observation. We concluded that all three methods of research are feasible and useful, depending on the particular questions one wishes to address and the resources available.

One limitation of all three approaches, however, is that they are not capable of assessing a subject's level of concentration or engagement with a task. This is an important consideration in determining efficiency levels but is beyond the scope of this stage of our research.

In addition to exploring new methodology, we sought to investigate certain other poorly understood factors affecting software production and time utilization. It is our belief that the social environment (both organizational structure and culture), process and technology all need to be examined in order to obtain a complete picture of software development efforts.

Indeed, we found that some elements of organizational character were at least equally important, and perhaps of greater significance, than the given technology in terms of explaining time utilization. In particular, the data we collected on the number of inter-personal contacts a software engineer needed to make during a typical working day strongly suggests that solving technical problems is not the only major consumer of development time. Rather, the software developers we studied must apply just as much effort and attention to determining and making inter-personal contacts. Most importantly, we were able to quantify what had previously been predominately qualitative impressions about life in this firm.

5.7 FUTURE RESEARCH

As noted by the organizational theorists March and Olsen, the empirical focus on time allocation is "deceptively mundane" [21]. It involves not only collecting and aggregating descriptive data and statistics but also developing some explanation of the underlying processes. In this research, we sought to extend knowledge in both directions. First, we explored how and what information to collect on time utilization. In addition, we attempted to reconcile and interpret those results within the context of an organization.

In subsequent research we plan to continue to identify some of the structural determinants of the allocation of time. What factors drive blocking or waiting periods? What are the primary causes of a consistent pattern of interruptions within an organization? An additional important area for further work concerns the individual and organizational costs and benefits of such patterns. For example, interruptions obviously affect a person's ability to concentrate, but they may also play important educational and motivational roles within a company. Finally, data on the sequence of activities and the time spent on each can be combined with cost information. A model of the development process can then be assembled that enables an organization to reduce cost and production time by more effectively focusing its technology and resources. Such a model can also alert managers to problems, scheduling bottlenecks or instances when the process is being circumvented.

There are also opportunities for further methodological research. As technology progresses, new forms of instrumentation are rapidly becoming available. Some advances are enabling researchers to capture information in very fine detail, while others can ease the manual burden associated with empirical studies. Yet we have seen very little experimentation with these new capabilities; most field researchers still rely on interviews or traditional paper surveys. We believe other approaches, of varying levels of technical sophistication, are called for, particularly in the case of software development. For example, we have considered giving developers a device similar to a Sharp Wizard and asking them to record their daily activities on it (a more interactive type of time diary). Similarly, one could prompt subjects to record their current activity by randomly paging them throughout the day. (A combination of the above two methods, perhaps targeted by personality, would probably be most efficient.) Such experiments are likely to not only improve our existing knowledge of software development but also offer the exciting prospect of access to previously unexplored research questions.

ACKNOWLEDGMENTS

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 approved the financial support for this work. Prof. Marcie Tyre of MIT was particularly helpful, providing extensive input along the way and pointing us in the direction of relevant literature on organizational theory. 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

[1] M. J. Abel. Experiences in an exploratory distributed organization. In Jolene Galegher, Robert E. Kraut, and Carmen Egido, editors, *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, pages 489–510. Lawrence Erlbaum Associates, Hillside, New Jersey, 1990.

[2] H. R. Bernard, P. D. Killworth, and L. Seiler. Informant accuracy in social network data. *Social Networks*, 2(3):191–218, 1980.

[3] B. W. Boehm. The hardware/software cost ratio: Is it a myth? *Computer*, 16:78–80, March 1983.

[4] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Trans Software Eng.*, SE-14(10):1462–1477, October 1988.

[5] M.G. Bradac, D. E. Perry, and L. G. Votta. Prototyping a process monitoring experiment. *IEEE Trans Software Eng.*, 20(10):774–784, May 1994.

[6] R. E. Brooks. Studying programmer behavior experimentally: The problems of proper methodology. *Communications of the ACM*, 23(4):207–213, April 1980.

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

[8] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Duxbury Press, Boston, MA., 1983.

[9] J. S. Colson Jr. and E. M. Prell. Total quality management for a large software project. *AT&T Technical Journal*, 73(3):48–56, May/June 1992.

[10] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory*. Aldine Publishing Company, Chicago, IL., 1967.

[11] R. Guindon. Designing the design process: Exploiting opportunistic thoughts. *Human-Computer Interaction*, 5:305–344, 1990.

[12] R. Guindon. Knowledge exploited by experts during software system design. *International Journal Man-Machine Studies*, 33:279–304, 1990.

[13] E. T. Hall. *The Other Dimension of Time*. Anchor Press/Doubleday, New York, New York, 1983.

[14] W. S. Humphery. *Managing the Software Process*. Addison-Wesley Publishing Co., 1989. Reading, Massachusetts.

[15] International Organization for Standardization. *Quality Systems — Model for Quality Assurance in Design/Development, Production, Installation, and Servicing*, international standard iso 9001: 1987(e) edition, 1987.

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

[17] D. Keirsey and M. Bates. *Please Understand Me: Character and Temperment Types*. Prometheus Nemesis, Del Mar, CA., fifth edition, 1984.

[18] R. Kelley and J. Caplan. How bell labs creates star performers. *Harvard Business Review*, pages 128 – 139, July - August 1993.

[19] R. E. Kraut and J. Galegher. Patterns of contact and communication in scientific research collaborations. In Jolene Galegher, Robert E. Kraut, and Carmen Egido, editors, *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, pages 149–171. Lawrence Erlbaum

Associates, Hillside, New Jersey, 1990.
[20] E. Langer. *Mindfulness*. Addison-Wesley, New York, New York, 1989.
[21] J. G. March and J. P. Olsen. *Ambiguity and Choice in Organizations*. Bergen, 1976.
[22] D. B. Mayer and A. W. Stalnaker. Selection and evaluation of computer personnel-the research history of sig/cpr. In *Proceedings 1968 ACM National Conference*, pages 657–670, 1968.
[23] J. E. McGrath. Time matters in groups. In Jolene Galegher, Robert E. Kraut, and Carmen Egido, editors, *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, pages 23–61. Lawrence Erlbaum Associates, Hillside, New Jersey, 1990.
[24] G. A. Miller. The magic number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.
[25] H. M. Parson. What happened at hawthorne? *Science*, 183:922–932, March 1974.
[26] D. E. Perry, N. A. Staudenmayer, and L. G. Votta. People, organizations, and process improvement. *IEEE Software*, pages 36–45, July 1994.
[27] D. E. Perry and L. G. Votta. The planning number 2.5 +,- .5. Technical Report BL0112650-930930-28TM, AT&T Bell Laboratories, Murray, NJ, December 1993.
[28] H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing on-line and off-line programming performance. *cacm*, 11(1):3–11, January 1968.
[29] J. B. Schriber and B. A. Gutek. Some time dimensions of work: Measurement of an underlying aspect of organizational culture. *Journal of Applied Psychology*, 72(4):642–650, 1987.
[30] L. S. Tsai. Overt and covert problem solving: Reversing the direction of a swimming fish. *Perceptual and Motor Skills*, 65:740–742, 1987.
[31] L. G. Votta. By the way, has anyone studied any real programmers, yet? In *Proceedings of the Ninth International Software Process Workshop, Airlie, Virginia*, pages 93–95, October 1994.
[32] G. M. Weinberg. *The Psychology of Computer Programming*. Van Nostrand Reinhold, New York, New York, 1971.
[33] A. L. Wolf and D. S. Rosenblum. A study in software process data capture and analysis. In *Proceedings of the Second International Conference on Software Process*, pages 115–124, February 1993.

Appendix A: Software Developer Self-Report Time Form

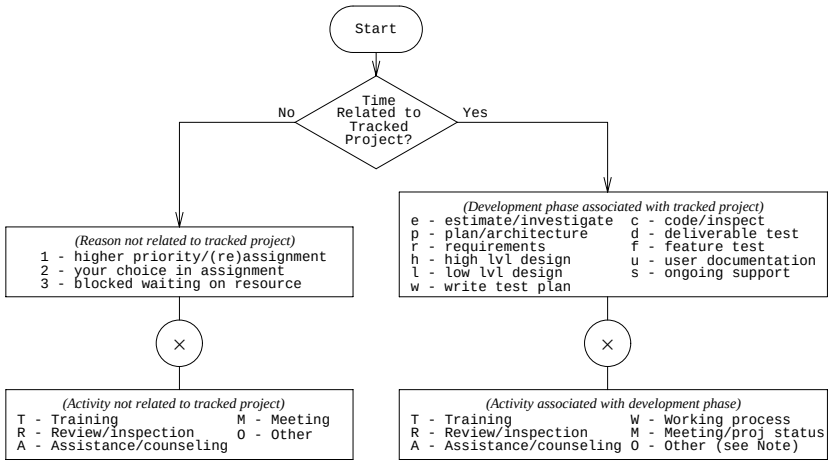
Diagnostic Development Process Measurement Record

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"*.

Appendix B: Completed Software Developer Time Form

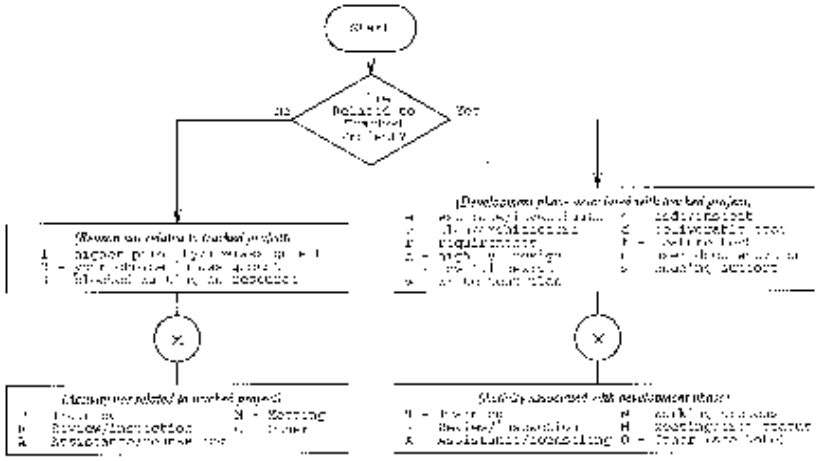
Diagnostic Development Process Measurement Record

Tracked Project: ABCDR

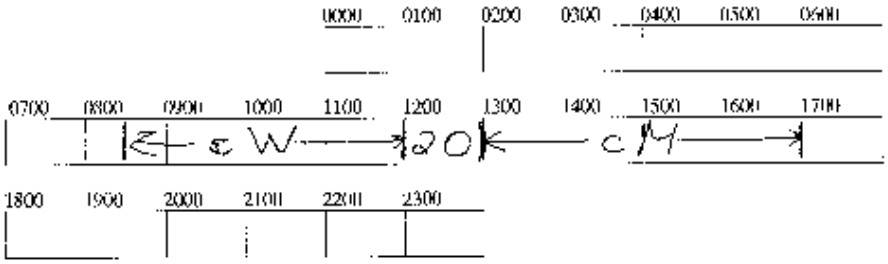
Developer: A. B. Smith

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:



Comments: 1200 - Lunch

Refer questions/comments to:

Mark Bradac: (708)979-3757, ihlpb!mbradac
Larry Votta: (708)713-1612, research!votta

Note: Comments are required for selection of "0 - Other".