Communication Metrics for Software Development

Allen H. Dutoit, *Member*, *IEEE*, and Bernd Bruegge, *Member*, *IEEE Computer Society*

Abstract—We present empirical evidence that metrics on communication artifacts generated by groupware tools can be used to gain significant insight into the development process that produced them. We describe a test-bed for developing and testing communication metrics, a senior level software engineering project course at Carnegie Mellon University, in which we conducted several studies and experiments from 1991–1996 with more than 400 participants. Such a test-bed is an ideal environment for empirical software engineering, providing sufficient realism while allowing for controlled observation of important project parameters. We describe three proof-of-concept experiments to illustrate the value of communication metrics in software development projects. Finally, we propose a statistical framework based on structural equations for validating these communication metrics.

Index Terms—Empirical software engineering, communication, statistics, structural equations.

1 Introduction

Metrics applied to software artifacts have proven to be useful in measuring the impact of a tool or method in the context of a software project. In some cases, they enabled specific problems in the engineering processes to be identified, thus demonstrating the value of metrics and other instrumentation for process improvement. However, software code is only one of the many artifacts produced during software development. Moreover, it is often available only late in the process. The infrastructure and the resources needed to collect metrics on code are also non-negligible, and often prevents their use for identifying problems as they occur.

Communication artifacts, such as electronic mail, memoranda, or records generated by groupware tools, represent a different perspective on the development process. They are available throughout the project, they capture information about a more comprehensive set of artifacts (e.g., code, process, organization, politics, morale), and their form is independent of implementation technology, development infrastructure, or even the existence of a product. Metrics applied to such communication artifacts can, therefore, provide significant insight into the development process that produced them.

In this paper, we discuss the design and evaluation of a set of communication metrics for software development. Our goal is to develop metrics that enable the assessment of a tool or a method in the context of a project. Our long-term goal is to provide metrics that help identifying problems as they occur. In Section 2, we illustrate possible uses of these metrics with an example.

Next, in Section 3, we present a test-bed, a senior level software engineering project course at Carnegie Mellon University, in which we conducted several studies and experiments. Such a test-bed represents an ideal environment for empirical software engineering, providing sufficient realism while allowing for controlled observation of important project parameters, such as development methodology, communication infrastructure, and implementation language.

Next, in Section 4, we present three proof-of-concept experiments to illustrate the value of communication metrics and their possible use for examining the correlation between various tools or methods and the development process. Measures are defined on the electronic messages exchanged between teams. Different patterns in the traffic are analyzed and associated with specific crises and differences in outcome.

While Section 4 provides strong anecdotal evidence that communication metrics can be used to investigate the variables of interest, Section 5 provides statistical evidence that this is the case. We build a statistical framework based on structural equations to measure the impact of a number of interesting variables (e.g., development methods and tools) on both communication (e.g., Bboard traffic) and work products (e.g., code, documents). We instantiate this framework using data from the test-bed. This empirical evidence strongly suggests that:

- in the set of projects studied the electronic traffic was representative enough of the overall communication
- communication metrics provide additional insight into the development process.

We conclude this paper by outlining future work that is necessary for the empirical validation of communication metrics.

2 An Illustrative Example

2.1 Scope and Approach

The case study presented in this section was conducted during the preliminary stages of this research [12]. Its goal was to illustrate the relationship between communication and outcome in large projects. It was conducted in the con-

• *A.H. Dutoit and B. Bruegge are with the Institut fuer Informatik / H1, Technische Universitaet Muenchen, D-80290 Munich, Germany. E-mail: {dutoit, bruegge}@in.tum.de.*

Manuscript received 22 May 1997; revised 1 Sept. 1997.
Recommended for acceptance by A. Fuggetta and R. Taylor.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 106770.

text of a senior level, software engineering project class at Carnegie Mellon University ("Software Engineering," 15-413, Fall'93). The class included 45 students divided into 10 teams, and lasted four months. The project successfully ended in the delivery of a system of about 30,000 lines of code and 200 pages of documentation.

The approach taken for this study was the use quantitative measures of electronic bulletin boards (called Bboards in this paper) traffic as a surrogate for interteam and intrateam communication measures. The findings of this study were confirmed against direct observations by the instructors of the class, project documents, and data collected from a student survey.

The quantitative measures on the Bboard traffic included the number of messages sent by each team per week, the number of noun phrases contained in those messages, and the number of unique noun phrases. The rationale behind counting noun phrases was to estimate the size of the vocabulary used by students to discuss development issues. It was assumed that a broad vocabulary indicated a lack of standards, from a both terminology and conceptual point of view. The size of this vocabulary would then decrease as developers negotiated standard terms and as the development of the system progressed towards delivery.

The number of unique noun phrases was computed using a set of natural language processing (NLP) tools [7]. Messages were stripped of their headers, signatures, special characters, and other encoded enclosures before being analyzed with the NLP tools.

Note that, given the preliminary nature and the scope of this case study, its findings and conclusions have an anecdotal value. We present it in this section only as a motivation for this research. Note also, however, that such anecdotal evidence support findings of previous empirical studies [9], [22] which went some way to show that communication critically impacts the process and its outcome.

2.2 Global Traffic

During the course of the semester, 2,502 messages were posted on Bboards by students, teaching assistants, and instructors. Most Bboards followed a weekly pattern: Bboard activity increased immediately before team weekly meetings, and decreased at the beginning of the week (Sunday night and Monday morning). This motivated the sampling of observations on a weekly basis.

Fig. 1 shows the communication activity on the Bboards for each week of the project. The bottom curve (solid black) shows the total number of messages sent per week. The top curve (dotted black) displays the total number of unique noun phrases used in those messages for each week. The middle curve (dashed black) displays the number of new noun phrases used in those messages for each week (i.e., noun phrases that were not used in any previous week). The horizontal axis represents time in weeks where the first week is numbered zero. The vertical axis represents the number of messages for the black curve, and the number of terms divided by 10 for the dotted and dashed curves.

The number of messages posted peaked during requirements (week #3) and the start of system integration (week #11). The drop at weeks #6 and #7 indicates a drop in

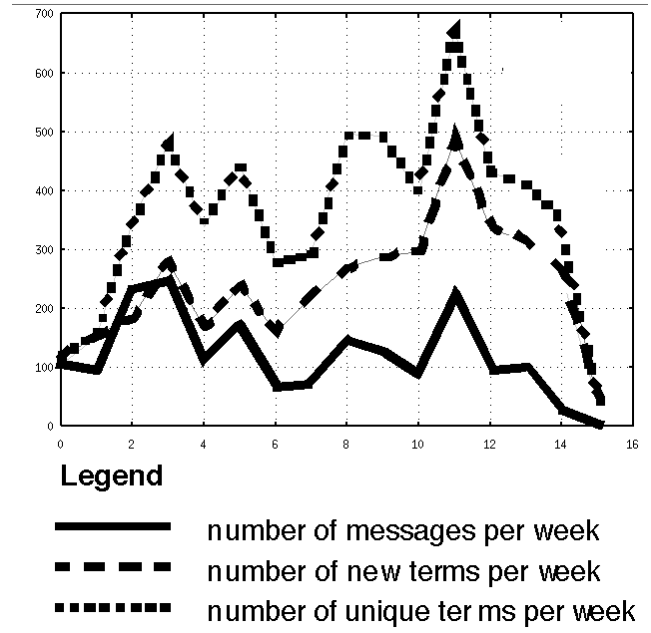

Fig. 1. Global Bboard traffic. The solid curve shows the number of Bboard messages which were sent each week. The dotted curve shows the number of unique noun phrases contained in these messages. The dashed curve shows the number of these noun phrases that were not used in previous weeks.

the activity of the class waiting for the integration team to release the first merged database of models (delay occurred due to problems with the modeling tool and with the set up time of the integration team).

An examination of the gray and dashed curves led to similar observations. However, the peak during integration is more pronounced than the requirement peaks, indicating a richer vocabulary, and possibly a broader range of issues being discussed.

2.3 Intrateam Communication

Fig. 2 displays the same curves as Fig. 1 for the messages posted on two selected team Bboards. The top graph represents the number of messages, the total number of unique terms and the number of new terms per week for the User Interface team. The bottom graph represents the same information for the Hazmat team (whose assignment was to build the database and query system for hazardous materials). The motivation for selecting these teams was that the User Interface team performed well (according to class standards, e.g., number of implemented user functions, level of integration with other subsystems) whereas the Hazmat team performed badly.

The User Interface team shows peaks in the number of messages similar to the global metric. However, the dotted and dashed curves peak during the requirements phase, thus indicating a richer vocabulary during problem definition. This is because the User Interface team was composed of representatives of all the teams. Moreover, this team had a better shared understanding of the overall system. This pattern was also observed for other teams who performed an integration function (e.g., the team charged with code integration, not displayed here, also demonstrated a richer vocabulary during requirements). In contrast, the Hazmat

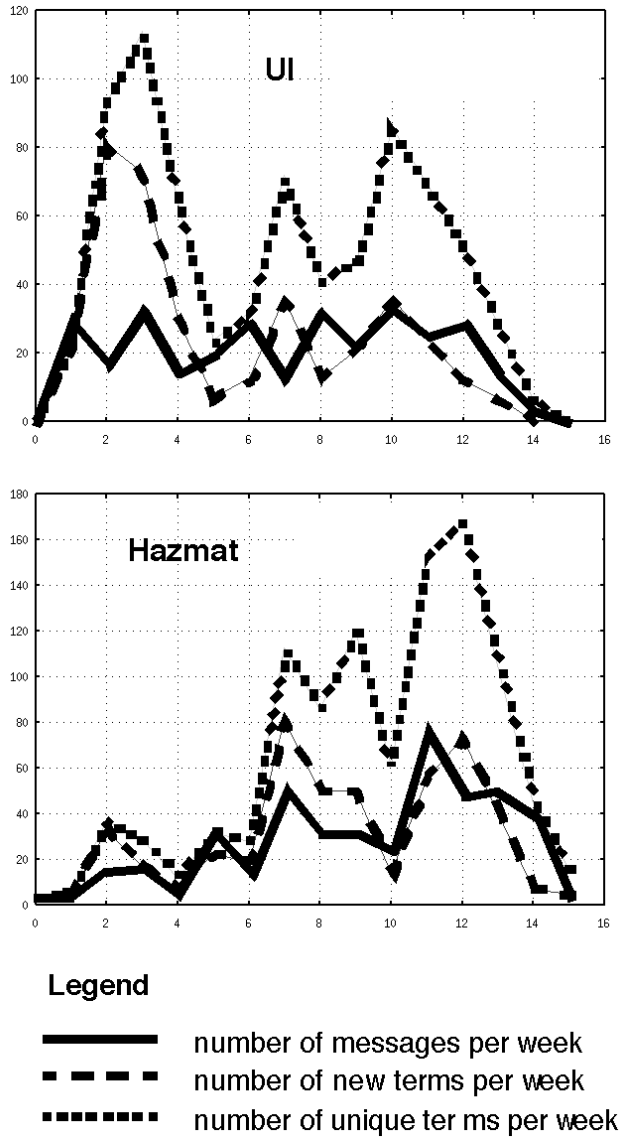

Fig. 2. Bboard traffic for UI and Hazmat teams. The top graph represents the number of messages, the number of unique noun phrases, and the number of new noun phrases for the User Interface team (who performed well). The bottom graph displays the same information for the Hazmat team (who performed badly). The User Interface team introduced many more terms during requirements analysis.

team displayed on the right, who did not have much understanding of the rest of the system before integration, shows a profile similar to the global profile (i.e., lesser volume during requirements and peaks in later phases near integration). This profile is representative of teams which were not assigned an integration function.

These observations were correlated with the observation that the teams displaying a better understanding of the overall system performed better according to the standards set in the course (fewer deadlines missed, functionally rich subsystem demonstrated, integration with the whole system achieved early).

2.4 Interteam Communication

The observations in the previous sections were further confirmed by the examination of the interteam communication for the User Interface and Hazmat teams. In Fig. 3, the x axis represents time in weeks. The y axis represents the teams of the project. In the top graph, the z axis represents the number of messages exchanged by the User Interface team and a given team. The bottom graph displays the same information for the Hazmat team. The User Interface team interacted with most teams at a relatively constant rate, with moderate peaks during integration. In the case of Hazmat, the interteam interaction was weaker while peaks during integration were more pronounced. Hazmat also heavily interacted with management. Hazmat relied more heavily on their teaching assistant for interteam coordination and did not often attempt to directly synchronize with other teams.

This case study illustrated some of the problems associated with communication in team-based projects. Moreover, it indicated that communication measures could be used to identify more problems associated with the organization and the development process.

The next section describes issues in measurement in software engineering, along with solutions and practices described in the literature. We then define communication metrics more formally and present a test-bed for their design and experimentation.

3 A Test-Bed for Communication Metrics

The need for software measures has been raised and solutions proposed for more than two decades. The measurement issue includes the evaluation of not only the outcome of a project, in terms of quality, appropriateness, or any user-oriented measure of goodness, but also of its intermediate work products. The focus on work products enables the estimation of project progress, the impact of various methods, organizations, and tools, and the identification of specific problem areas.

The difficulty in measuring software lies in its number of attributes and their variation across developers, organizations, application domains, and implementation technology. A software product could be measured in terms of its length (e.g., lines of code, number of operands), the complexity of its control flow (e.g., number of decision points, number of loops), number of known defects, mean time to failure, and many other measures. However, no single attribute captures completely the state of the software or can predict the future state of the software as a function of time. For this reason, many different measures have been proposed.

3.1 Metrics on Work Products

An early and popular software metric has been the number of source lines of code, due to its ease of collection. However, given the high variation of lines of code across developers, languages and domains, other measures have been proposed, such as the Halstead length and volume metrics [15], the cyclomatic number [21], and, more recently, Function Points [1]. It has been shown that the Halstead length and volume, cyclomatic number and lines of code do not differ substantially in their effectiveness when used for estimation [2]. Although the Function Point method leads to

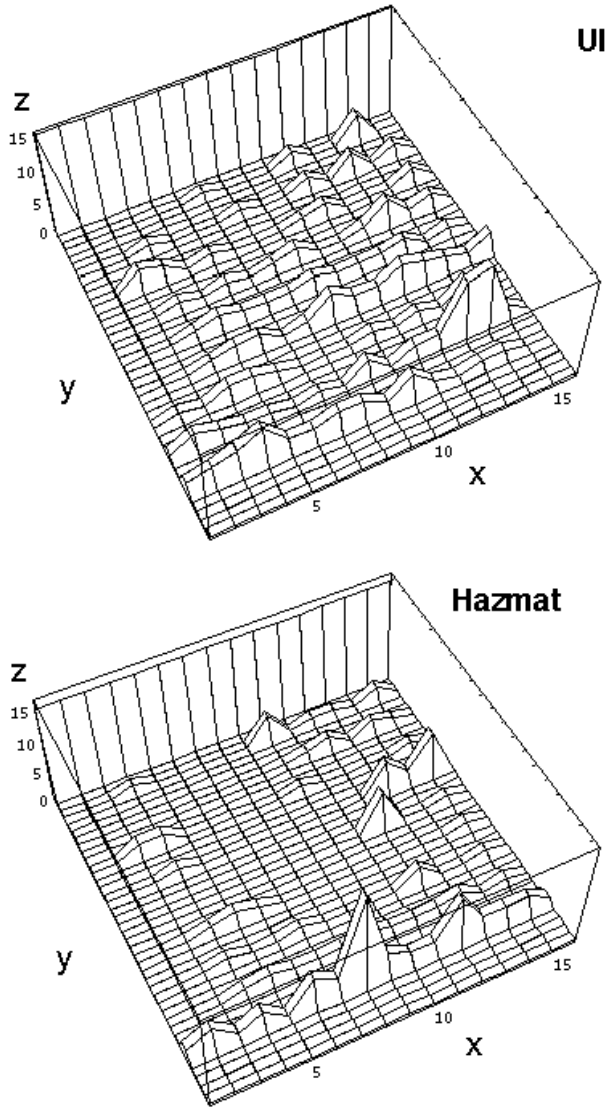

Fig. 3. Bboard interteam traffic for User Interface and Hazmat teams. The x axis represents time in weeks, the y axis, the teams in the project. In the top graph, the z axis represents the number of messages exchanged by the User Interface team with the other teams. In the bottom graph, the z axis represents the number of messages exchanged by the Hazmat team with the other teams. The User Interface team had much more interteam communication than the Hazmat team.

more accurate estimates [17], its application is still complex and has a steep learning curve [14]. The Bang method, which was proposed by DeMarco [10], addresses some of the issues related to Function Points. Bang and Function Points also have the advantage of not depending on the existence of code.

The practice of software metrics has been moving from academia to industry. An increasing number of case studies and success stories have been documented. For example, Grady and Caswell describe human and organizational issues which arose when transferring measurement practices into Hewlett Packard [14]. DeMarco provides a handbook describing the management issues associated with measurement and estimation [10].

The main obstacles encountered when transitioning software metrics into practice include the difficulty in instrumenting the software artifact under construction, the lack of tools, and the dependency of measures on a large number of variables [13].

3.2 Metrics on Communication

We use the terms *communication metrics* to denote measures derived from communication records. Communication metrics fall in the class of process measures [13], in the sense that they measure the number of incidents of a specified type within a process. Communication metrics have been used for studying the information flow within a software project [26], [27]. In general, we are interested in computing an approximation of the volume and complexity of the information exchanged among participants, not only to study the information flow among participants, but also to relate it to the outcome of the project. We have experimented with simple measures such as counting the number of messages exchanged among teams, to focusing only on meeting announcements and records, to extracting noun phrases from each electronic message to determining how many unique terms were used during certain period of times.

Communication metrics have interesting properties: they are available earlier and are easier to collect than outcome metrics. Communication records cover more aspects of the project than technical and refer to actual events, instead of planned events. Communication metrics also provide finer granularity at the participant level (e.g., electronic messages have explicit authors). On a document or software artifact, the individual contribution is usually more difficult to establish. Finally, communication metrics also provide finer granularity at the calendar level given that project participants are likely to communicate on a daily basis.

Our motivation behind investigating communication metrics is also based on the observation that misunderstandings and omissions in the communication process can result in costly defects in the resulting software. This is especially true as the number of participants and the diversity of their background grow [6].

3.3 Software Engineering Test-Bed

To investigate the value of communication metrics, we have used as a test-bed the senior level software engineering project courses at Carnegie Mellon University (15-413 and 15-499). In these project courses, students are taught software engineering by working on a real project, with real clients and real deadlines. They are organized into teams that need to cooperate for the project to succeed. Such a test-bed represents an ideal environment for experimental software engineering, providing sufficient realism while allowing for controlled observation of important project parameters. In particular, this test-bed has the following features.

Real Applications. A real client from industry or government defines the problem. Column 2 in Table 1 shows the range of applications developed in the test-bed between 1991 and 1996. Clients included individuals and organizations, such as a city planner, a chief of police, the Environmental Protection Agency and Daimler Benz Corporation.

TABLE 1
CARNEGIE MELLON UNIVERSITY SOFTWARE ENGINEERING TEST-BED PROJECTS FROM 1991 TO 1996

Project		Methodology Project		Management		Communication			Metrics	
Year	Application Domain	Methodology	Process	Participants	Teams	Project Wide Bboards	Bboards per Team	Tools	LOC (reuse not included)	Messages
Spring'91 (basic)	City planning (interactive maps)	SA/SD	Greenfield w/ waterfall model (1 prototype)	15	4	2	11	E-mail Bboards	17,000(C)	339
Fall'91 (basic)	City planning (interactive Pittsburgh)	OMT	Greenfield w/ waterfall model (1 prototype)	33	5	2	12	E-mail Bboards	9,000(C) 20,000(C++)	933
Spring'92 (advanced)	Air quality modeling (GEMS)	OMT	Re-engineering w/ conc. engineering (3 prototypes)	10	5	1	00	E-mail Bboards	20,000(C++)	401
Fall'92 (basic)	Accident management (FRIEND)	OMT	Greenfield w/ waterfall model (1 prototype)	46	6	3	1-3	E-mail Bboards	8,000(C) 31,000(C++)	2382
Fall'92 (basic)	Accident management (FRIEND III)	OOSE	Greenfield w/ saw-tooth model (2 prototypes)	55	9	4	1	E-mail Bboards	21,000(C++) 11,000(Tcl)	3056
Spring'94 (advanced)	Accident management (FRIEND IV)	OOSE	Re-engineering w/ conc. engineering (1 prototype)	16	7	5	1	E-mail Bboards	50,000(C++)	13909
Fall'94 (basic)	Environmental pollution modeling (JEWEL)	OMT+ Use case	Re-engineering w/ sawtooth model (3 prototypes)	59	6	4	1	E-mail Bboards	57,000(C++)	3613
Spring'95 (advanced)	Environmental pollution modeling (JEWEL II)	OMT+ Use case	Re-engineering w/ sawtooth model (4 prototypes)	29	6	5	1-4	E-mail Bboards	27,300(C++)	2792
Fall'95 (basic)	Predictive maintenance (DIAMOND)	OMT+ Use case	Re-engineering w/ sawtooth model (2 prototypes)	44	6	7	2-4	Lotus Notes 3.3 (E-mail data-bases)	20,000(C++)	4284
Spring'96 (advanced)	Predictive maintenance (DIAMOND II)	OMT+ Use case	Re-engineering w/ conc. engineering (3 prototypes)	19	3	5	2	Lotus Notes 3.3 (E-mail data-bases)	6,100(Java) 4,000(C++) 1440(HTML)	1624

Real Deadlines. The development has to be done within 17-18 weeks (duration of one semester).

Partial Distribution. Project members work at different times and different locations: Each of the project members is a student taking 4-5 other classes with overlapping schedules, therefore allowing only a limited number of face-to-face communication.

Multidisciplinarity. Project members come from different disciplines such computer science, technical writing, electrical engineering, mathematics and design.

State of the Art Tools. Each project uses state of the art tools, often tools that have just been released. For example, we worked with evaluation copies of StP and Objectory, alpha releases of OMTool and the Interviews user interface builder kit, and beta releases of Netscape's product family and Java.

Software Process. Depending on the project and on the technical expertise of the participants, we use a waterfall model, a sawtooth model (a refined V model described by Rowen [23]), or concurrent engineering, in which we allow each of the phases of the development to overlap significantly. The development either is from scratch (Greenfield engineering) or starts with an existing system that needs to be rebuilt (Re-engineering) or interfaced to (Interface Engineering). Each test-bed project produces one or more prototypes.

Project-Based Organization. The team structure is derived from the top level design of the system formulated after the requirements engineering phase. The functional requirements and the available team members form the input for the organization chart shown in Fig. 4. Subsystem teams implement a set of use cases identified during requirements engineering. Cross-functional teams, made out of representatives from the subsystem teams, address support tasks such as configuration management, technical writing, system integration and management of the system architecture. Team membership is a role, that is, a person can be a member of several teams. The total number of project members is shown in column "Participants" in Table 1. Teams in basic projects (See column "Year") are recruited with beginners who have never participated in a test-bed project. The majority of the team members in the advanced projects have participated in the basic course of the previous semester.

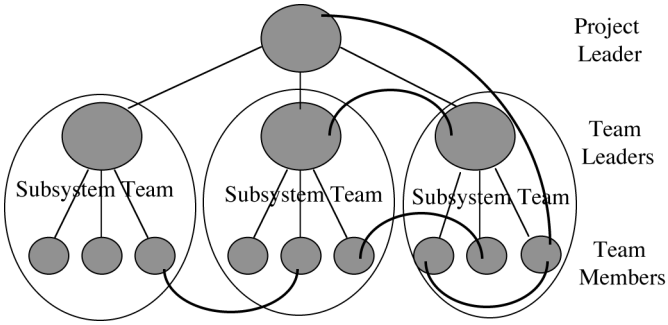

Fig. 4. Project-based organization. The project organization was driven by the decomposition of the system into subsystems. Each team was assigned a subsystem to analyze, design, and implement. Cross functional aspects were handled by cross functional teams, formed by representatives of each subsystem team.

As outcome metric, we used nonblank lines of code shown in column "LOC." This metric applies to the size of the last prototype delivered in the project. Communication among members of the same team is defined as *intrateam communication;* communication among members from different teams is defined as *interteam communication.* The main medium of communication in the test-bed projects under study has been a hierarchical structure of Bboards. Each team is assigned a set of Bboards that are primarily used for their internal communication. The communication structure is non-hierarchical as shown in Fig. 4. Members of any team can talk to members of any other team by sending e-mail or posting on the respective team Bboard. They are also strongly encouraged to read the Bboards of the other teams, which frequently leads to team Bboards also being used for team to team communication. In addition to team Bboards, we provide several Bboards for project-wide discussion and course announcements. The sum of interteam and intrateam BBoard communication is the archived project communication shown in column "BBoard Messages" in Table 1.

Other communication media include informal interactions, personal e-mail, weekly meetings, liaisons, and informal interactions. Personal e-mail has typically been used to discuss project issues that were not of public interest. More generally, e-mail is often selected over Bboard traffic for issues requiring privacy. Each team meets formally every week to report on status and plan for the next week. In addition, there is a weekly project meeting. At the beginning of the project, this meeting is used for training. As the project progresses project meetings are used for internal and client reviews.

Given the partial distribution of the project, the weekly team and project meetings are very critical components in the project communication. Meetings became even more critical once the role of a liaison was introduced into the project organization as a response to specific communication needs between teams. A liaison is a team member whose task is to gather information from other teams. For example, the user interface team, which depends on all of the functional teams, sends liaisons to the weekly meetings of all functional teams.

The fact that the team structure is derived from the problem at hand and targeted for the specific project, means that team and communication structure are changing with each test-bed project. A test-bed project has to cope with other changes. In fact, our test-bed projects can be characterized by constant change occurring in the following areas:

- Change in requirements (intraproject change)
- Change in development environment (mostly interproject change, sometimes even intraproject)
- Change in communication tools (interproject change)
- Change in organization: Each project starts with a customized team structured developed specifically for the project (interproject change)

Each of these changes makes repeatability of experiments and comparison of results across test-bed projects a problem.

One way to provide comparability of results from different test-bed projects despite these changes is to provide a controlled setting of the experimental parameters and not change parameters except for those under investigation. Designing an experiment such that each of the alternative technique or methods under study can be examined in a controlled environment allows differences between the methods (if present) to be isolated and statistically measured [3].

This *analytical approach* is preferred by many scientists, but hard to use in experimental software engineering given the state of the art in methods for software development. Currently each real project provides a set of lessons learned that are hard to ignore in the next project. If a CASE tool turns out to be unusable in one project, we cannot seriously ask developers to use it another time, just to keep the parameters of the experiment in a controlled setting. This would end up as an academic exercise with no relevance to practitioners, an exercise in which we are not interested. In fact, our experience has shown that developers vehemently reject such an experiment, and consequently work against it, for example, by not using the CASE tool, thereby invalidating the results of the "controlled experiment" anyway.

An alternative is the *summative approach*. In this approach [28], we accept that the result of one experiment as well as changes in the environment influences the formulation of the next experiment. The experimental methodology used in this paper is based on the summative approach. However, we are still able to use statistical tools by treating each project as a quasi-experiment instead of a classical experiment. We will discuss the implications of this approach on observable test-bed parameters in more detail in Section 5. First, we describe in detail the three case studies selected for this paper.

4 Case Studies

In this section, we describe three case studies in which we designed and applied communication metrics to investigate three assumptions:

- Object-oriented methods provide better benefits than structural analysis and design methods in medium to large scale projects.
- Emphasizing requirements early in the project reduces the number of integration problems.
- Improving communication improves project outcome.

We previously reported on the case studies as single project studies using qualitative observations and results

from a participants survey and preliminary data [5], [8], [11]. In this paper, we treat these projects as a multiproject variation study across a set of teams [3] using metrics data on the Bboard traffic. The goal is to confirm these previous results and illustrate the importance of communication. Although Bboard messages represent only a fraction of the overall communication, we believe that the number of messages constitutes a representative enough sample of the communication to allow meaningful interpretation. This belief is reinforced by the observation that most participants posted messages daily.

The next section, Section 4.1, describes a study in which we compared projects using two different development methods: Structural Analysis and Structured Design (SA/SD), and the Object Modeling Technique (OMT). Section 4.2 reports on a similar study comparing the use of OMT and Object-Oriented Software Engineering (OOSE). Finally, Section 4.3 reports on a study that assessed the correlation between an issue-based model and the quality of meetings and communication.

4.1 Improving Methodology: SA/SD and OMT

4.1.1 Assumptions

As the number of participants of the project increases, the choice of a software development methodology becomes a critical issue. The development methodology is critical in three aspects of the project: the development environment, communication, and management areas. First, given the lack of experience of most participants with medium to large scale projects, we needed to train developers and adopt common project standards and procedures. Second, the development methodology enables the participants to construct a model of the application domain (often unknown to them at the beginning of the project) which can then be used as a reference during interactions with the client and among the participants. Moreover, the development methodology enables the participants to construct a model of the system under development, thus allowing the exchange of accurate information about component interfaces and other shared knowledge. Finally, the development methodology enables management to divide the system in parts, assign each part to teams, and track the progress of the development on a finer granularity. In this study, we were interested to assess the effectiveness of SA/SD and OMT in addressing these issues.

4.1.2 Selection of Development Methodologies

The Spring'91 project, *Interactive Maps,* used the formal specification language Larch and a CASE tool, Software Through Pictures from Interactive Development Environments. The requirements analysis and design were described with a mixture of two design approaches, data flow-oriented design [31] with the SA/SD tools available in StP, and object-based design using abstract data types [20]. Data flow-oriented design has been successfully applied to strictly sequential information systems, but has not been as useful for the design of interactive systems. For example, project participants were not able to express the user interface for *Interactive Maps* in a structured design notation. Another problem with SA/SD development was its inability to express the results of the analysis and design phase with a uniform notation. We found it difficult and time-consuming to convince the participants that it was necessary to change not only the terminology but also the model when moving from analysis to design.

We, therefore, started looking for a methodology that could be used for consistent modeling across the project. Object-oriented methodologies looked especially promising and we selected OMT because of its strong support for analysis and design [24] and a tool (OMTool). We introduced OMT in the Fall'91 project, *Interactive Pittsburgh.*

4.1.3 Experimental Approach

In this study, we counted the number of interteam and intrateam messages per phase for the Spring'91 and Fall'91 projects. We defined an intrateam message as a message that was sent from a team member to the team's Bboard. We defined all other messages as interteam messages (e.g., a message from a member to another team's Bboard). Messages posted on multiple Bboards were only counted once. We defined a phase as the interval of time between any two deliverables. For example, we treated the time between the start of the project and the delivery of the first version of the requirements document as the requirements analysis phase. Although the requirements document was revised several times afterwards, often as late as in the integration phase, most of the requirements effort occurred during this phase.

4.1.4 Results and Interpretation

We mapped the communication data to four project phases named Requirements Analysis, Design (System Design and Object Design), Unit Coding (Unit Testing and Implementation), and Integration (System Testing and Delivery). For visualization of the data, we used a Kiviat graph because it is an ideal tool for visualizing imbalance. Fig. 5 visualizes the number of intrateam (black lines) and interteam messages (gray lines) with two Kiviat graphs for Spring '91 *(Interactive Maps)* and Fall'91 *(Interactive Pittsburgh),* respectively. Each Kiviat graph has four axes showing the amount of communication for each of the project phases.

As a good system design attempts to reduce coupling between subsystems and to increase the cohesiveness within subsystems, a good organization should require minimal interteam communication and encourage intrateam communication. This would result into a Kiviat graph where the interteam communication curve is inside the intrateam communication. One would also expect that information should flow freely within the organization instead of being driven by crises. This would result into more information exchanged during requirements (the definition of the project) and unit coding (the refinement of the interfaces) than during the two other phases.

The Spring'91 data show a quite different communication pattern, a pattern perhaps all- too familiar to those who have experience with large development projects using traditional development techniques. While the requirements analysis phase shows relatively little interteam communication when compared to intrateam communication, the integration phase shows a 4:1 ratio in favor of interteam communication.

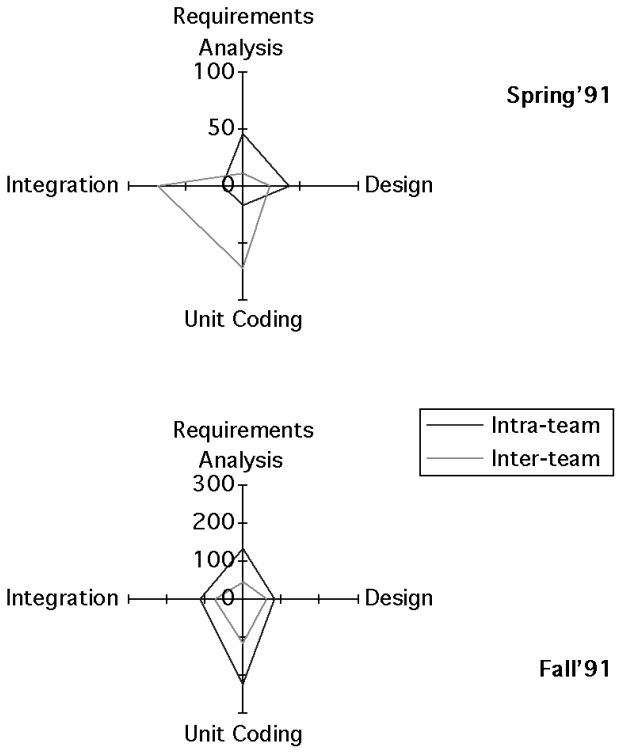

Fig. 5. Spring'91 vs. Fall'91: Interteam and intrateam Bboard communication. The top graph represents inter and intratean communication for the Interactive Maps project. The bottom graph represents the same information for the Interactive Pittsburgh project. Interactive Pittsburgh resulted into a better system and more balanced communication.

The Fall'91 data, on the other hand, show relatively even interteam communication during each phase, making the project very well balanced with respect to interteam communication. The similarity between the total number of interteam messages made in the last two phases for both *Interactive Maps* and *Interactive Pittsburgh* (74 and 76 Bboard Messages) is striking, given that *Interactive Pittsburgh* involved more than twice as many participants.

Our data suggest that object-oriented design enhanced communication, with benefits most noticeable during the latter stages of development. Other than the use of different design methodologies, the main differences between the *Interactive Maps* and *Interactive Pittsburgh* developments were the scope of the projects and the size of the development teams. With a more ambitious project and twice the number of developers we might have expected that the communications picture would have been much worse for *Interactive Pittsburgh* than it was for *Interactive Maps*, particularly during system integration. That the opposite was true seems to indicate that integration was much smoother for *Interactive Pittsburgh*, even with twice the number of developers. Our interpretation is that:

- the use of OMT minimized the dependencies across modules, and thus, across teams, therefore reducing the cost of interteam communication, and
- the use of OMT encouraged the early negotiation of hard issues during requirements and unit coding, in anticipation of crises during integration.

4.2 Improving Requirements: OMT and OOSE

4.2.1 Assumptions

For the Fall'93 project, we were interested in improving the requirements phase. As the number of project members increased, it became difficult to maintain a shared perspective of the system under development. The coding and integration phase witnessed discrepancies among the participants' view of the system (e.g., two teams understanding the same term or the same function differently), leading to late requirements changes (e.g., redefinition of a function, or removal of a function from the system), and an increased cost. These considerations lead us to consider OOSE [18], an object-oriented methodology whose focus is primarily on requirements engineering, in place of OMT, whose focus is on primarily on requirements analysis and design.[1] We assumed that providing better methodological support during the front end of the process would improve the quality of requirements, and thus, of the outcome.

4.2.2 Selection of Development Methods

OMT assumes a specification of the system has already been developed with the customer. The central focus of OMT is on constructing and refining the object models. In Fall'92, it was supported by a modeling tool, called OMTool, which provided functionality for mechanically translating object models into C++ templates.

OOSE, unlike OMT, supports requirements elicitation. OOSE is centered on *use cases*, which describe the functionality of the system from a user's perspective. A use case is a textual description of the interactions between the user and the system during a specific transaction. Use cases are written during requirements, used during analysis and design for the identification and classification of objects, and used during design for distributing behavior across objects and identifying their dependencies. Although OOSE leads to the construction and delivery of object models as in the case of OMT, the primary focus of the developer remains on user-level transactions throughout the process.

4.2.3 Experimental Approach

In this study, we compared the Fall'92 and Fall'93 projects. Both projects had to deliver the same software for the same client. No code or design was reused across these two projects. Except for the development methodology, both projects used the same tools (e.g., CVS for revision control, C++ as development language, FrameMaker for documentation, and Bboards for communication).

Unlike the first study, we counted the total number of Bboard messages sent during each phase. To take into account differences across both projects, we used the number of days in each phase to normalize the data.

4.2.4 Results and Interpretation

Fig. 6 displays the number of messages sent per day for each phase in the project. Note that the number of messages

1. After the introduction and wide-spread use of OOSE, Rumbaugh demonstrated how OMT could be extended to include requirements elicitation with use cases [23]. In fact, as a result of our own experiments, we started using a hybrid "OMT + Use Cases" since Fall'94 (see Table 1, Column Methodology). Note that OOSE, OMT, and Booch's notations have been merged into UML (Unified Modeling Language) and is supported by a single tool [4]. This change of methodology is an example of our summative approach.

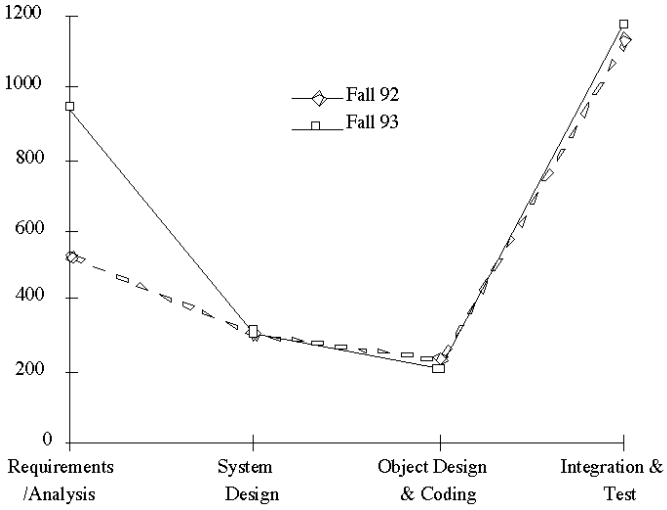

Fig. 6. Fall'92 vs. Fall'93: Total number of messages per day and per phase. The dashed curve represent the number of messages exchanged during each phase of the Fall'92 project. The solid curve represents the same information for the Fall'93 project. The latter project resulted in much better requirements and implemented functionality.

for the system design, object design and unit coding, and integration and test phases are very similar for both projects. However, the number of Bboard messages per day during the requirements analysis phase during Fall'93 was much higher than during Fall'92. Direct observation and survey also confirmed that the rate of interaction during the beginning of the project was much higher.

Given that Fall'93 resulted in a better system (e.g., more delivered functionality, more consistent user interface) than Fall'92, and given that the only major variation across both projects was the use of OOSE instead of OMT, we believe that OOSE supported the development process better than OMT. The communication metrics analysis provides us with a quantitative insight of the difference: the use of OOSE leads to a higher interaction rate during the front end of the process, and thus facilitates a faster convergence of project participants on a shared view of the system.

4.3 itIWEB: Improving Meeting Procedures

4.3.1 Assumptions

In Spring'95, we were interested in improving communication among and across teams of the project. In projects before Spring'95, we noticed that weekly meetings were critical communication and negotiation points due to the partial distribution of the project. Our assumption was that, by improving the planning, execution, and capture of meetings, the outcome of the project could be improved.

4.3.2 Improving Meeting Procedures

In the projects before to Spring'95, team leaders posted agendas prior to the meeting on the team Bboard. The agenda was then reviewed at the beginning of the meeting. A designated meeting participant was then responsible for capturing the meeting and posting minutes to the team Bboard shortly after the meeting. Little emphasis was put on the form of agendas; minutes were recorded in chronological form. Consequently, we observed a broad variance in the quality and effectiveness of the weekly meetings.

In Spring'95, we adopted an issue-based model, called *indented text Information Web* (itIWEB),[2] for capturing meetings. Team leaders prepared the meeting by collecting a list of outstanding issues into an agenda that was then posted on the team Bboard for review by the team members prior to the meeting. During the meeting, the team leader used the agenda to focus the discussion on specific issues and obtain resolution and consensus from the team members. In addition, the minute taker would record the discussion as a set of issues, proposals, arguments, and resolutions. Each discussion item was then organized according to its context instead of chronologically. The minutes were then posted to the team Bboard shortly after the meeting, as before.

4.3.3 Experimental Approach

In this study, we compared the Spring'94 and Spring'95 projects. Although both projects delivered different systems to different clients, the functionality and implementation was similar. FRIEND (Spring'94) is a distributed information system for emergency response, while JEWEL is a distributed information system for emissions modeling.

The communication metrics we developed for this study were centered on the messages pertaining to a meeting. We extracted from the Bboard traffic agendas, minutes, and replies to agendas and minutes. We then removed any e-mail header information, signatures and enclosures.

4.3.4 Results and Interpretation

Scanning through the agendas and minutes, we observed that meeting minutes were more comprehensive and more readable than in previous courses. In order to confirm these observations, we compared the word size of the meeting documents between Spring'94 and Spring'95, and the number of replies to each agenda and minutes.

We observed that the size of the agendas and the minutes was significantly larger when itIWEB was used. The relative size difference was larger for agendas than minutes, indicating a better meeting preparation (see Fig. 7). A careful reading of messages in both courses revealed that the information content was indeed larger in the agendas and minutes of Spring'95, and that this increase in size was not due to redundancies or verbosity.

In an attempt to quantify readability, we counted the number of replies for each agenda and minutes. Fig. 8 displays the average number of replies for agendas and minutes for the Spring'94 and Spring'95 classes. We observed that the number of replies for agendas doubled. A closer examination to the replies to the agendas, we realized that the subject of the replies also changed: in Spring'94, all of the replies to agendas were organizational in content, (e.g. students indicating they could not attend the meeting, notices informing of new location of the meeting). In Spring'95, replies also included reaction to the content of the agenda (e.g. update on the status of an action item, additions to the agenda, etc.). Moreover, replies to agendas in Spring'95 often included the relevant excerpt (in the form of an itIWEB action item or issue) the message was responding to. This reinforces our previous observation that meetings were better prepared in Spring'95.

2. The itIWEB model has been inspired by the itIBIS issue-based model [30].

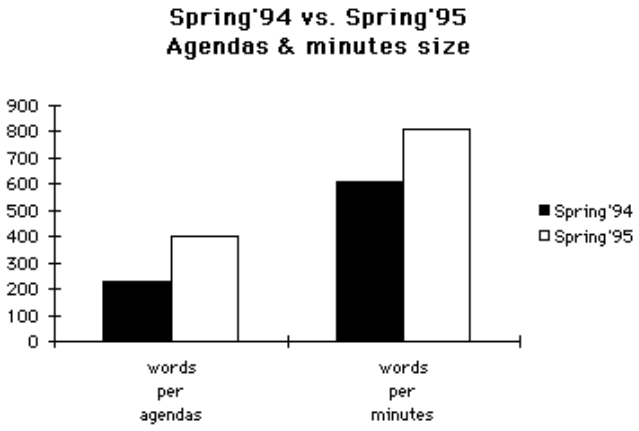

Fig. 7. Spring'94 vs. Spring'95: average word size of meeting documents. The meeting documents (i.e., agendas and minutes) were larger on average when using an issue-based approach (Spring'95).

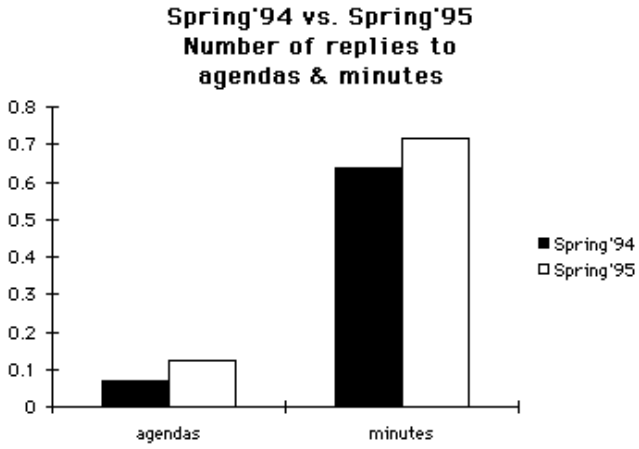

Fig. 8. Spring'94 vs. Spring'95: average number of replies per meeting document. The traffic generated by the meeting documents was larger when using an issue-based approach (Spring'95).

We made similar observations for minutes. However, the increase in the number of replies was not as drastic for minutes as for agendas. After analyzing the replies for both projects, we concluded that the deeper and broader structure of the Bboards in Spring'95 discouraged students to follow other teams' Bboards, thus decreasing the responses to minutes from other teams.[3] This assumption was confirmed by a survey of the Spring'95 project in which seven students (out of seven who raised the issue) admitted not following other team Bboards regularly. Overall, the number of replies to minutes from team members increased significantly.

The use of an issue-based model increased the amount and structure of the captured information. The survey confirmed that this had a positive impact on the project from the participants' point of view. By examining the number of replies to meeting related messages, we also confirmed that structured agendas and minutes increased the readability of the information captured.

3. In Spring'94 there was only one Bboard per team, whereas in Spring'95, each team had a Bboard for minutes, a Bboard for discussion and a Bboard for announcements.

5 Toward an Empirical Framework

Section 4 provides strong anecdotal evidence that communication metrics can be used to investigate the variables of interest. This section provides statistical evidence that this is the case. We build a statistical framework based on structural equations to measure the impact of a number of interesting variables (e.g., development methods and tools) on both communication (e.g., Bboard traffic) and work products (e.g., code, documents). We instantiate this framework using data from the test-bed.

A priori, building a statistical framework for the software engineering test-bed described in Section 3 is a challenge. The test-bed represents a compromise between a controlled experiment and a field study. On the one hand, the projects under study are representative of real projects: The complexity and duration of the project, the number of participants, and the application domain are such that any single participant cannot completely understand the delivered system on his or her own. On the other hand, because these projects all occur in the same context, variables such as the management style, participants' experience, and cultural and human issues do not need to be taken into account explicitly assuming that they did not vary significantly across projects. This reduces the measuring error associated with the variables of interest, such as the methods and tools used. Moreover, most sources of variations may be known to the researcher, given a greater access to the project and to the nonproprietary nature of the data set. Although some variation in the projects may not be directly controlled a priori, they can be modeled a posteriori, and thus, be taken into account.

Note that the empirical evidence resulting from the framework is statistically valid only for the ranges of variables we explored. For example, we investigated projects comprising of 20 to 60 participants. We can extrapolate to larger projects based on these results, however, these conclusions would have to be supported by additional experiments and insights.

In this section, we describe two structural equation models for testing hypotheses using communication and outcome metrics. Data from six projects were used to estimate these models and test hypotheses about the tools and methods used in these projects.

5.1 Regression and Structural Equation Models

Regression models (see Fig. 9) have been used in software engineering [10], [13] for explaining the variation of a single dependent variable Y as a linear combination of independent variables. The dependent variable usually models a measured attribute of the project (e.g., lines of code, number of function points). The independent variables X_i represent possible sources of variations (e.g., number of developers, number of teams, development methodology). Parameters β_i measuring the contribution of each independent variable to the variation of the dependent variable are estimated using test-bed data.

An important issue in modeling software engineering projects is the difficulty in measuring the goodness of a project using a single dependent variable. Often, goodness is measured along several dimensions such as size, complexity, and

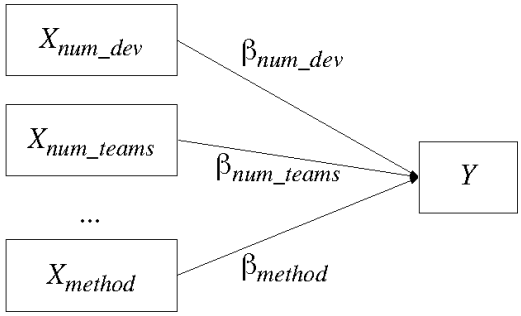

Fig. 9. Regression model.

quality. In addition, concepts such as quality are also often in turn measured in terms of a number of metrics [10]. Structural models [19], a generalization of regression and path analysis, addresses both issues as shown in Fig. 10. Latent variables (i.e., unobserved dependent variables, such as quality) can be modeled explicitly and measured as a combination of metrics (e.g., number of reported defects, deviations from schedule, deviations from requirements). Latent variables are measured as a linear function of the set of dependent variables. For example, in Fig. 10, the latent variable η is measured as a combination of the dependent variables Y_{doc} and Y_{code}

Moreover, interactions between dependent variables can also be taken into account, thus enabling the estimation of indirect effects (e.g., "How does a given method correlate with communication which in turn correlates with outcome?").

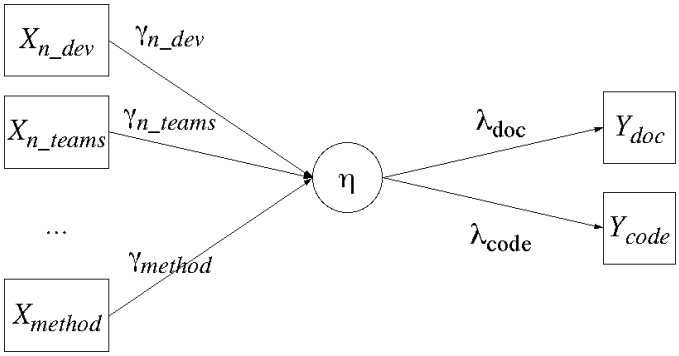

Fig. 10. Structural equation model with two independent variables and one latent variable.

5.2 Organizing Data into Observations

Given a data set such as the test-bed projects deliverables and communication records, it is highly desirable to extract a maximum amount of information to increase the significance of the statistical model. In order words, we want to organize the data such that we maximize the number of observations per independent variable. We have done so by considering each phase and each team in the project as an observation. This was possible since each phase was punctuated by a deliverable for each team (e.g., a section in a document, a code module). This lead to a large enough number of observations could be maximized (~200) for the number of independent variables taken into consideration (~20).

5.3 Models

We have estimated two structural equation models for explaining the outcome size as well as the outcome complexity. For both models, we used data from six test-bed projects (Fall'91, Fall'92, Fall'93, and Fall'94, Spring'94 and Spring'95: The Spring'91, Fall'95, and Spring'96 projects have not yet been included in the models). Both models are based on the general model shown in Fig. 11.

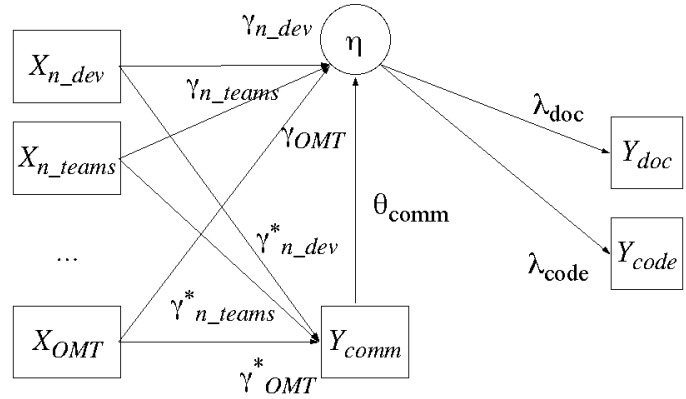

Fig. 11. General structural equation model with indirect communication effect.

The *outcome size* was modeled as a latent variable η measured by two dependent variables, source code size Y_{code} and document size Y_{doc}. The communication size Y_{comm} was also modeled as a dependent variable that contributed to the outcome size η. All effects that could have influenced the size of communication or outcome or both were modeled as independent variables, for example, the number of developers X_{n_dev}, the number of teams X_{nteams}, the use of the development methodology X_{OMT}. The parameters γ_i represent the contribution of each independent variable X_i to the outcome. The parameter θ_{comm} represents the contribution of the Bboard communication to the outcome. The parameters γ_i represent the contribution of each independent variable to the Bboard communication. They are referred to as indirect effects.

The second model was built for explaining *outcome complexity*. In this case, the dependent variables Y_{doc} and Y_{code} represent the complexity of documents and source code, respectively; the latent variable η represents the outcome complexity; Y_{comm} represents the complexity of Bboard communication. The estimation of the complexity model results in estimates for γ_i, γ^*_i, and θ_{comm}.

Note that many possible sources of variation (i.e., X_i variables) were considered, ranging from the effects we were interested in measuring (e.g., the change from OMT to Objectory) to effects we did not control (e.g., the number of participants) to other parameters which changed as a result of our summative approach (e.g., the change in organizational structure or the change in communication infrastructure).

The size and complexity of source code was measured using Halstead length and volume metrics [15]. The size and complexity of documents and Bboard messages were measured in terms of number of words and number of

unique noun phrases. Noun phrases were counted using off-the-shelf natural language processing tools [7].

For the two models we considered 14 potential sources of variation (conceptual variables) and modeled them with 27 independent variables. Some sources of variation are modeled as multiple independent variables, e.g., the development methodology used in the test-bed project was modeled with the three binary independent variables X_{OMT}, X_{OOSE}, and $X_{OMT+Usecases}$.

5.4 Significant Variables

In the size model, using a 10 percent level of significance, 10 conceptual variables were significant: four had a statistically significant correlation with outcome, three with Bboard communication, and three with both. In the complexity model, 11 variables were significant: four had a statistically significant correlation with outcome, three with communication, and four with both. Both Bboard communication size and complexity had a statistically significant correlation with outcome. Of particular interest were variables associated with the development infrastructure and project size: Bboard metrics did not vary significantly across development languages, platforms, and off-the shelf components, although outcome variables varied significantly. However, the Bboard communication varied significantly with the number of developers per team, whereas the outcome measures did not. Otherwise, both communication and outcome metrics varied significantly with the development method, type of code (e.g., database vs. user interface), and the use of itIWEB and communication infrastructure variables (e.g., number of Bboards).

5.5 Interpretation of Estimates

Given that space does not allow a thorough discussion of the estimation of these models, we present only a selected subset of our results. The reader is referred to [12] for a complete presentation of the methods and results used to estimate these structural models.

In the following discussion, we focus on four independent variables of interest: X_{OMT} (whether the OMT method was used or not), X_{itIWEB} (whether itIWEB was used or not), $X_{n\_liaisons}$ (number of liaisons per team), X_{n_dev} (number of developers per team) and $X_{square\_num\_developers}$ (square number of developers per team). Y_{comm} represents the Bboard communication measure (i.e., number of words per phase for the size measure, and number of unique noun phrases per phase for complexity measure).

Table 2 displays the estimates of the direct effects of selected independent variables for the size and complexity models. OMT represented by the parameter γ_{OMT} correlated strongly (3.7656) with the size of the deliverables modeled by η, which can be explained in part by the code generation facilities available in OMTool. OMT correlated also with an increase in deliverable complexity (0.4798). We interpret this correlation as an increase in productivity and in the amount of redundancy in deliverables. itIWEB represented by the parameter γ_{itIWEB} also contributed (0.7346) to an increase in size of the deliverables while not increasing the complexity of deliverables. We interpret this result as an improvement in the productivity of students. The number of liaisons per team represented by $\gamma_{n\_liaisons}$ reduced the complexity of deliverables, which can be interpreted as a better communication structure. That is, better communication leads to a faster convergence of participants on common concepts and terms, thus reducing complexity. Finally, the last row in Table 2 represents θ_{comm}, the contribution of Bboard communication on deliverable size and complexity. As we can see, an increase in communication results in larger and less complex deliverables.

TABLE 2
SELECTED DIRECT EFFECTS FOR STRUCTURAL EQUATION OF SIZE AND COMPLEXITY MODELS

Parameter	Size Model (Estimate)	Complexity Model (Estimate)
γ_{OMT}	3.7656	0.4798
γ_{itIWEB}	0.7346	~0
$\gamma_{n\_liaisons}$	0.1064	–0.6871
θ_{comm}	0.2503	–0.0867

Table 3 displays the estimates of the indirect effects of selected independent variables through Bboard communications. OMT reduced the size and complexity of Bboard communication, which is due to an increase communication of developers through an alternate channel, OMTool. The reduction of the size and complexity of Bboard communication due to itIWEB is interpreted by us as a decrease in spontaneous Bboard communication (i.e., the data presented in the previous section indicated that the traffic associated with agendas and minutes actually increased). We interpret this as an increase of communication associated with the meeting, thus reducing the need for spontaneous, crisis-driven interactions. Finally, note that Bboard communication is a function of the number of developers and the square number of developers, as expected.

TABLE 3
SELECTED INDIRECT EFFECTS FOR STRUCTURAL EQUATION SIZE AND COMPLEXITY MODELS

Parameter	Size Model (Estimate)	Complexity Model (Estimate)
γ^*_{OMT}	–0.3469	–0.3246
γ^*_{itIWEB}	–0.8051	–0.7266
$\gamma^*_{n\_dev}$	0.5711	0.5529
$\gamma^*_{square\_num\_developers}$	–0.0357	–0.0334

Note that the approach we adopted in building this statistical framework is that of quasiexperimental design. Although we do not have the full control over the experimental context that would enable us a true or classical experiment, the use of statistical procedures is still valuable for modeling sources of variations at hand. This is also consistent with our summative approach, in which we feedback into the test-bed lessons learned during case studies and statistical analysis.

6 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we proposed and established the value of communication metrics in the context of software development. We developed and refined several communication

metrics using a real test-bed, the software engineering course at Carnegie Mellon University. We considered this test-bed as an ideal compromise between a completely controlled environment and an industrial project. We presented empirical evidence that communication metrics can provide more insight into the development process than the use of outcome metrics alone. The return on investment is substantial, given that the additional effort entailed by the use of communication metrics is minimal.

The statistical analysis presented in Section 5 showed that communication variables (e.g., number of liaisons, number of Bboards, Bboard measures) had a statistically significant correlation with outcome. Also, communication and outcome vary differently, thus capturing different aspects of the development process. However, they both depend on a common set of independent variables. We postulate that, once the relation between communication and outcome is better understood, communication measures could be used in place of outcome metrics. This would decrease the cost of instrumentation and enable the use of metrics earlier in the process.

Several issues related to communication metrics need to be investigated further. Most importantly, further analysis and validation in the form of repeated studies and experiments need to be performed to refine the use of communication metrics for project control. In particular, we believe that digital communication provides sufficiently rapid variation in software development to allow the use of communication metrics during projects (vs. only for postmortem studies), as illustrated in Section 2.

All the communication metrics presented in this paper are measures of posted Bboard messages. As such, they measure the amount and complexity of information that is available as opposed to information that is shared. In a future experiment, we plan to count the accesses to each message.

Finally, we are planning to compare different communication mechanisms (e.g., Lotus Notes, the World Wide Web, information modeling environments such as *n*-dim [29]) to investigate the correlation between communication infrastructures and project outcome.

ACKNOWLEDGMENTS

This research was done while the authors were affiliated with the Engineering Design Research Center and the Computer Science Department of Carnegie Mellon University. This research was sponsored, in part, by the Division of Undergraduate Education, National Science Foundation under Grant Nos. USE-9251836 and DUE-9451375; a university equipment grant from Hewlett-Packard; and by the Engineering Design Research Center, a National Science Foundation Engineering Research Center. We thank the 400+ students of the 15-413 and 15-499 courses at Carnegie Mellon University for developing very exciting front wave systems while providing us with the data for this paper. We also thank the clients of these projects, including Chief Mike Bookser, Daimler-Benz, the EPA, and Ted Russel. We are grateful to Robert Coyne, Suresh Konda, Daniel P. Siewiorek, and Eswaran "Sub" Subrahmanian for their advice and guidance during this work.

REFERENCES

[1] A.J. Albrecht and J.E. Gaffney Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Eng.*, vol. 9, pp. 639–648, 1983.

[2] V.R. Basili and D. Hutchens, "An Empirical Study of a Syntactic Complexity Family," *IEEE Trans. Software Eng.*, vol. 9, pp. 664–672, 1983.

[3] V.R. Basili, R.W. Selby, and D.H. Hutchens, "Experimentation in Software Engineering," *IEEE Trans. Software Eng.*, vol. 12, pp. 733–743, 1986.

[4] G. Booch, "The Unified Modeling Language," *Unix Review*, vol. 14, no. 13, pp. 41–44, 46, 48, 1996.

[5] B. Bruegge, J. Blythe, J. Jackson, and J. Shufelt, "Object-Oriented System Modeling with OMT," *Proc. Object-Oriented Programming Systems, Languages, and Applications, OOPSLA'92*, Vancouver, pp. 359–376, 1992.

[6] L. Bucciarelli, "Reflective Practices in Engineering Design," *Design Studies*, vol. 5, no. 3, pp. 185–190, 1984.

[7] N. Coulter, I. Monarch, S. Konda, and M. Carr, "An Evolutionary Perspective of Software Engineering Research Through Coword Analysis," CMU/SEI–95–TR–019, Software Engineering Inst., Carnegie Mellon Univ., Pittsburgh, 1995.

[8] R. Coyne, A. Dutoit, B. Bruegge, and D. Rothenberger, "Teaching More Comprehensive Model-Based Software Engineering: Experience With Objectory's Use Case Approach," *Proc. Eighth Conf. Software Eng. Education, CSEE'95*, New Orleans, Lecture Notes in Computer Science 895, pp. 339–374, Springer-Verlag, 1995.

[9] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Comm. ACM*, vol. 31, no. 11, pp. 1,268–1,287, 1988.

[10] T. DeMarco, *Controlling Software Projects: Management, Measurement, and Estimation*. New York: Yourdon Press, 1982.

[11] A.H. Dutoit, B. Bruegge, and R. Coyne, "Using an Issue-Based Model in a Team-Based Software Engineering Course," *Proc. Software Eng.: Education and Practice, SEEP'96*, Dunedin, New Zealand, pp. 130–137, IEEE CS Press, 1996.

[12] A.H. Dutoit, "The Rôle of Communication in Team-Based Software Engineering Projects," PhD thesis, Dept. of Electrical and Computer Engineering, Pittsburgh: Carnegie Mellon Univ., 1996.

[13] N.E. Fenton, *Software Metrics, A Rigorous Approach*. Chapman & Hall, 1991.

[14] R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*. Englewood Cliffs, N.J.: Prentice Hall, 1987.

[15] M.H. Halstead, *Elements of Software Science*. North-Holland, 1977.

[16] W.C. Hetzel, "The Sorry State of Software Practice Measurement and Evaluation," *J. Systems Software*, no. 31, pp. 171–179. 1995.

[17] M. Itakura and A. Takayanagi, "A Model for Estimating Program Size and Its Evaluation," *Proc. Sixth Int'l Conf. Software Engineering*, Tokyo, pp. 104–109, 1982.

[18] I. Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard, *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[19] K.G. Jöreskog, "A General Method for Estimating a Linear Structural Equation System," *Structural Equation Models in the Social Sciences*, A.S. Goldberger and O.D. Ducan, eds. New York: Seminar Press, 1973.

[20] B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*. MIT Press, 1986.

[21] T.J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. 12, pp. 308–320, 1976.

[22] D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement." *IEEE Software*, vol. 11, no. 4, pp. 36–45, 1994.

[23] R.B. Rowen, "Software Project Management Under Incomplete and Ambiguous Specification," *IEEE Trans. Eng. Management*, vol. 37, 1990.

[24] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[25] J. Rumbaugh, "Getting Started: Using Use Cases to Capture Requirements," *J. for Object Oriented Programming*, vol. Sept., pp. 8–10, 23, Sept. 1994.

[26] C.B. Seaman and V.R. Basili, "An Empirical Study of Communication in Code Inspections," *Proc. 19th Int'l Conf. Software Eng.*, Boston, May 1997.

[27] M. Saeki, "Communication, Collaboration, and Cooperation in Software Development—How Should We Support Group Work in Software Development?" *Proc. Asia-Pacific Software Eng. Conf.* Brisbane, Australia, 1995.

[28] M. Scriven, "The Methodology of Evaluation," R. Tyler, R.M. Gagne, and M. Scriven, eds., *Perspectives of Curriculum Evaluation, AERA Monograph Series on Curriculum Evaluation,* no. 1. Chicago: Rand McNally, 1967.

[29] E. Subrahmanian, S.L. Konda, I.A. Monarch, Y. Reich, and A.W. Westerberg, "Computational Support for Shared Memory in Design," *Automation Based Creative Design: Current Issues in Computers and Architecture,* A. Tzonis and I. White, eds., Amsterdam: Elsevier Science, 1993.

[30] B. Yakemovic and J. Conklin, "Report on a Development Project Use of an Issue-Based Information System," *Proc. Conf. Computer-Supported Cooperative Work, CSCW,* Los Angeles, pp. 105–118, 1990.

[31] E. Yourdon and L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* Englewood Cliffs, N.J.: Prentice Hall, 1979.

Allen H. Dutoit received a diploma in computer science from the Swiss Federal Institute of Technology in Lausanne, and MS and PhD degrees in computer engineering from Carnegie Mellon University. Dr. Dutoit is a scientific assistant in the Computer Science Department at the Technische Universitaet Muenchen. His research interests include empirical software engineering, design rationale capture, prototype-based systems, and communication infrastructures for distributed projects.

Bernd Bruegge received a diploma in computer science at Universitaet Hamburg, Germany, and MS and PhD degrees in computer science from Carnegie Mellon University. Dr. Bruegge is a university professor in the Computer Science Department at Technische Universitaet Muenchen and an adjunct professor in the Computer Science Department at Carnegie Mellon University. His research interests include empirical software engineering, model-based software engineering, reusable architectures, diagnostic tools, and multiproject management.