Software Metrics for Distributed Development

Bernd Bruegge and Allen H. Dutoit[†]

November 1996
CMU-CS-96-190

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

[†]Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

In this paper, we present empirical evidence that metrics based on communication artifacts generated and recorded by groupware tools can be used to gain significant insight into the distributed development process that generated them. Moreover, we demonstrate that such communication metrics can provide better insights in the quality of processes and products than metrics based on outcome (e.g., function points, lines of code, words of documents). Most important, they can be uniformly applied throughout the process, independently of implementation technology, development infrastructure, or the existence of a product.

We discuss our approach in developing and testing the use of communication metrics for distributed developments. We present a testbed for distributed development, a senior level software engineering project course at Carnegie Mellon University, in which we conducted several studies and experiments from 1991 to 1996 with more than 400 participants. Such a testbed represents an ideal environment for empirical software engineering, providing sufficient realism while allowing for controlled observation of important project parameters. We present three proof-of-concept experiments to illustrate the value of communication metrics in distributed and evolving development projects. Finally, we propose a statistical framework based on structural equations for testing hypotheses in an environment characterized by constant change.

This research was sponsored in part by the Division of Undergraduate Education, National Science Foundation under grant numbers USE-9251836 & DUE-9451375, a university equipment grant from Hewlett Packard, and by the Engineering Design Research Center, a National Science Foundation Engineering Research Center.
The views and conclusions contained in this document are solely those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government.

Keywords: metrics, empirical software engineering, distributed development.

1. Introduction

Distributed development is rapidly emerging as an important development mode. In distributed development, the team members are distributed over space and time, only rarely meeting at the same time and location. Distributed development provides many benefits. Network connectivity advances the team's learning curve by being able to connect outside knowledge. If team members are separated by different time zones, one member can work on a problem effectively while the other one is sleeping. The distributed team member can use the network to span the globe in search of information to advance any of the phases.

Distributed development is not very well explored. It requires a new look at communication tools and management methods and a review of existing development methodologies and processes. Some of the questions that need to be answered are unique to distributed development: What tools are needed to support remote communication? Which is the appropriate tool for conducting remote discussions? How do project participants converge towards a common perspective despite the distributed nature of the project [1]? What is the right development methodology? What set of products should be developed? We believe that distributed development provides the experimental basis for answering these questions.

An important aspect of distributed development is that it requires a more disciplined approach. Formal agreements must be reached on many issues that could be treated nonchalantly in centralized developments but that impede the quality of the produced software: Scheduling of meetings, structure of agendas, contents of meeting minutes, error handling, documentation, reviews and software error tracking.

Most important for providing a better experimental setting is another difference: Because most of the communication is digitally based, a distributed development leaves an elaborate electronic trace of the project history. Centralized development tends to use synchronous communication which almost always occurs in a volatile manner, that is, no record of the communication itself is available. In a distributed development, more of the communication is asynchronous and as a result these communications are usually archivable. Of course, only if every communication is archived, would we have a complete picture of what happened. For many reasons this is not desired or even feasible. However, we argue that even a limited archival communication record of the project is a good basis for experimental software engineering.

In this paper, we describe how communication records can be used to answer empirically the questions such as, "Given two types of software development methodologies, which one is the better one?" or "Given alternative ways to conduct a meeting, which one is preferable? Structured vs. Unstructured?"

We believe it is unrealistic to attempt to answer these questions in a controlled setting, using a relatively small example or toy system. These questions are better answered with real-world systems, because some aspects of distribution such as team size, software process and communication infrastructure, are only relevant in large complex developments. However, this introduces the duration of the experiment as a factor. This is quite important, given the current speed of technology developments, where changes happen almost every few months.

The rest of this paper is organized into four sections. Section 2 motivates the use of communication metrics in distributed developments and proposes a testbed, the senior software engineering project course at Carnegie Mellon, for their evaluation. Section 3 presents three case studies conducted in our testbed illustrating the use and potential value of communication metrics. Section 4 proposes a statistical framework based on structural equation models to validate systematically communication metrics and other hypotheses. We provide and discuss preliminary model parameter estimations which confirm that communication metrics can provide substantial insight into distributed developments. Section 5 summarizes our results and outlines future directions.

2. A Testbed 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 products. The focus on intermediate products enable 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 be used to predict the future state of the software as a function of time. For this reason, many different measures have been proposed and many different estimation models constructed in the attempt to address this issue.

2.1. Outcome Metrics

An early and popular measure of software has been the number of source lines of code. 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 [2], the cyclomatic number [3], and, more recently, Function Points[4]. It has been shown [5] that the Halstead length and volume, the cyclomatic number and lines of code do not differ substantially in their effectiveness when used for estimation. Although the Function Point method leads to more accurate estimates [6], its application is still complex and requires a steep learning curve [7].

Nevertheless, the practice of software metrics has been moving from academia to industry. An increasing number of case studies and success stories were recently documented. For example, Grady and Caswell, members of a large effort to implement and evaluate software measurement approaches at Hewlett Packard, address human and organization issues which arose when transferring measurement practices into the organization [7]. Tom DeMarco provides a short handbook describing the management issues associated with measurement and estimation [8]. Popular methods of estimation based on software metrics, COCOMO [9] and SoftCost [10], make use of source lines of code for estimation. The general approach used for reducing the inaccuracy of this metric was, on the one hand, to restrict the application of the method to very similar projects (i.e. taking into account variations due to different organizations and products), and, on the other hand, to introduce a large number of parameters to model differences due to a variety of factors (e.g. required reliability of the target product, experience of developers, etc.).

To our knowledge, none of these models have modeled communication factors explicitly. It is generally known that communication complexity increases as a quadratic function of the number of developers (or the number of communicating entities), however, the type of communication medium used (e.g. paper, electronic mail, documents, hallway discussion), the communication structure (e.g. inter-team liaison roles) is not modeled as a factor.

2.2. Communication Metrics

Our hypothesis is that metrics applied to communication artifacts provide more insight into a distributed software development than outcome metrics. Moreover, communication metrics are more reliable and easier to implement. In addition, communication metrics provide accurate information generated during the project. Communication metrics can be used from the beginning of the project to the end. Outcome metrics can only be used when a work product or deliverable has been produced. In addition, different types of outcome metrics are necessary for measuring different products, making it more difficult to collect and validate data. Communication metrics also provide finer granularity on the individual level: During development, most participants communicate everyday. On a document, the individual contribution is usually not recognizable. There is also a difference in the frequency of change that can be captured by the metrics: Documents change a lot but not very often, whereas the communication history changes almost every day. This allows to capture the status of the project at a much finer level of granularity if we use communication metrics instead of outcome metrics.

2.3. Software Engineering Testbed

To investigate these hypotheses, we have used a testbed at Carnegie Mellon University for experimental software engineering that 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 testbed 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.

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

Distributed Development. 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. As process model we use either the waterfall model, or rapid prototyping with a sawtooth model (a refined V model described by Rowen [11]) or concurrent engineering, in which we allow each of the phases of the development to overlap significantly. The development is either from scratch (Greenfield engineering) or starts with an existing system that needs to be rebuilt (Reengineering) or interfaced to (Interface Engineering). Each testbed project produces one or more prototypes (The final delivery is counted as a prototype).

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 Figure 1. Each of the teams has the purpose of developing a set of use cases identified in the requirements engineering phase (functional teams) or to address support tasks such as configuration management, technical writing, system integration and management of the system architecture teams (cross-functional teams). 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 Figure 1. Teams in basic projects (See column "Year") are recruited with beginners who have never participated in a testbed project. The majority of the team members in the advanced projects have participated in the basic course of the previous semester.

Project		Methodology/Process		Management		Communication			Metrics	
Year	Application Domain	Methodology	Process	Participants	Teams	Project Wide Bboards	BBoard per Team	Tools	LOC (reuse not included)	Bboard Messages
Spring '91 (Basic)	City Planning (Interactive Maps)	SA/SD	Greenfield w/ Waterfall model (1 prototype)	15	4	2	1	E-Mail, Bboards	17,000(C)	339
Fall '91 (Basic)	City Planning (Interactive Pittsburgh)	OMT	Greenfield w/ Waterfall model (1 prototype)	33	5	2	1	E-Mail, Bboards	9,000(C) 18,000 (C++)	933
Spring '92 (Advanced)	Air Quality Modeling (GEMS)	OMT	Reengineering w/ Conc. Engineering (3 prototypes)	10	5	1	0	E-Mail, Bboards	20,000 (C++)	401
Fall '92 (Basic)	Accident Management (FRIEND)	OMT	Greenfield w/ Waterfall model (1 prototype)	46	6	3	1 to 3	E-Mail, Bboards	8,000 (C) 31,000 (C++)	2382
Fall '93 (Basic)	Accident Management (FRIEND III)	OOSE	Greenfield w/ Sawtooth 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	Reengineering w/ Conc. Engineering (1 prototype)	16	7	5	1	E-Mail, Bboards	50,000 (C++)	1309
Fall '94 (Basic)	Environmental Pollution Modeling (JEWEL)	OMT + Usecases	Reengineering 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 + Usecases	Reengineering w/ Sawtooth Model (4 Prototypes)	29	6	5	1 to 4	E-Mail, Bboards	27,300 (C++)	2792
Fall '95 (Basic)	Predictive Maintenance (DIAMOND)	OMT + Usecases	Interface Engineering w/ Sawtooth model (2 Prototypes)	44	6	7	2 to 4	Lotus Notes 3.3 (E-mail, databases)	20,000 (C++)	4284
Spring '96 (Advanced)	Predictive Maintenance (DIAMOND II)	OMT + Usecases	Interface Engineering w/ Conc. Engineering (3 prototypes)	19	3	5	2	Lotus Notes 3.3 (E-mail, databases)	16,100 (Java) 4,000 (C++) 1440 (HTML)	1624

Table 1. Carnegie Mellon University Software Engineering Testbed Projects from 1991 to 1996

As outcome metric we used non-blank lines of code (including comments) 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 *intra-team communication,* communication among members from different teams is defined as *inter-team communication.* The main medium of communication in the testbed projects under study has been a hierarchical structure of electronic bulletin boards (called Bboards in the following). Each team is assigned a set of Bboards which are primarily used for their internal communication. The communication structure is non-hierarchical as shown in Figure 1 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 inter-team and intra-team BBoard communication is the archived project communication shown in column "BBoard Messages" in Table 1.

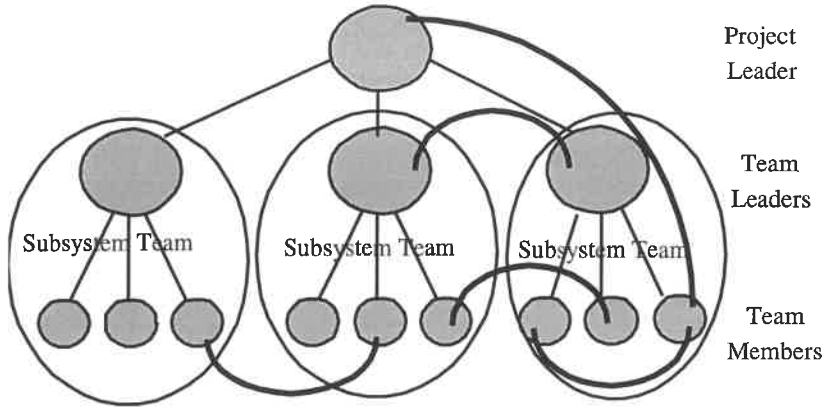

Figure 1. Project-based organization

Other communication media include informal interactions, personal e-mail, weekly meetings, liaisons, and informal interactions. Personal e-mail has typically been used to discuss the same issues as on the Bboard 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 distributed nature of the project, the weekly team and project meetings are very critical component in the project communication. Meetings became even more critical once the role of 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 each testbed project has a different team and communication structure. There are other changes a testbed project has to cope with as well. In fact, our testbed projects can be characterized by constant change occurring in the following areas:

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

Each of these changes make repeatability of experiments and comparison of results across testbed projects a problem.

One way to provide comparability of results from different testbed 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 [12].

This *analytical approach* is preferred by many scientists, but very 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 we are not interested in. In fact, our experience has shown, that developers vehemently reject such an experiment, and as a result work against it, for example, by not using the CASE tool, thereby invalidating the results of the "controlled experiment" anyway.

An alternative is the *formative approach.* In this approach, we accept that the result of one experiment as well as changes in the environment have impact on the formulation of the next experiment. The experimental methodology used in this paper is based on the formative 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 testbed parameters in more detail in the Section entitled Section 4. First we describe in detail the three case studies selected for this paper.

3. Case Studies

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

- Object-oriented methods provide better benefits that structural analysis and design methods in mid to large scale project.
- 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 [13, 14, 15]. In this paper, we treat these projects as a multiproject variation study across a set of teams [12] using validated metrics data on the Bboard traffic. The goal is to confirm these previous results and illustrate the importance of communication. Although Bboard messages only represent 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 3.1, 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). The section entitled Section 3.2 reports on a similar study comparing the use of OMT and Object-Oriented Software Engineering (OOSE). Finally, the section entitled Section 3.3 reports on a study which assessed the impact of an issue-based model, called indented text Information Web (itIWEB), on the quality of meetings and communication.

3.1. Improving Methodology: SA/SD and OMT

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 mid 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.

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 [16] with the SA/SD tools available in StP, and object-based design using abstract data types [17]. Data flow-oriented design has been successfully applied to strictly sequential informations 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 were 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 [18] and a tool (OMTool). We introduced OMT in the Fall'91 project, *Interactive Pittsburgh*.

Experimental approach

In this study, we counted the number of inter-team and intra-team messages per phase for the Spring'91 and Fall'91 projects. We defined an intra-team message as a message which was sent from a team member to the team's Bboard. We defined all other messages as inter-team 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. Even though 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.

Results and interpretation

We mapped the communication data to four project phases named Requirements Analysis, Design (System Design and Object or Detailed 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. Figure 2 visualizes the number of intra-team (black lines) and inter-team 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.

Ideally, one would like each project phase to produce the same amount of communication, which would be a symptom of good management. A corresponding Kiviat graph would be a symmetric diamond with all sides being of equal length. The Spring '91 data show a quite different communication pattern, a pattern which is 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 inter-team communication when compared to intra-team communication, the integration phase shows a 4-to-1 ratio in favor of inter-team communication.

The Fall '91 data, on the other hand, show relatively even inter-team communication during each phase, making the project very well balanced with respect to inter-team communication. The similarity between the total number of inter-team 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

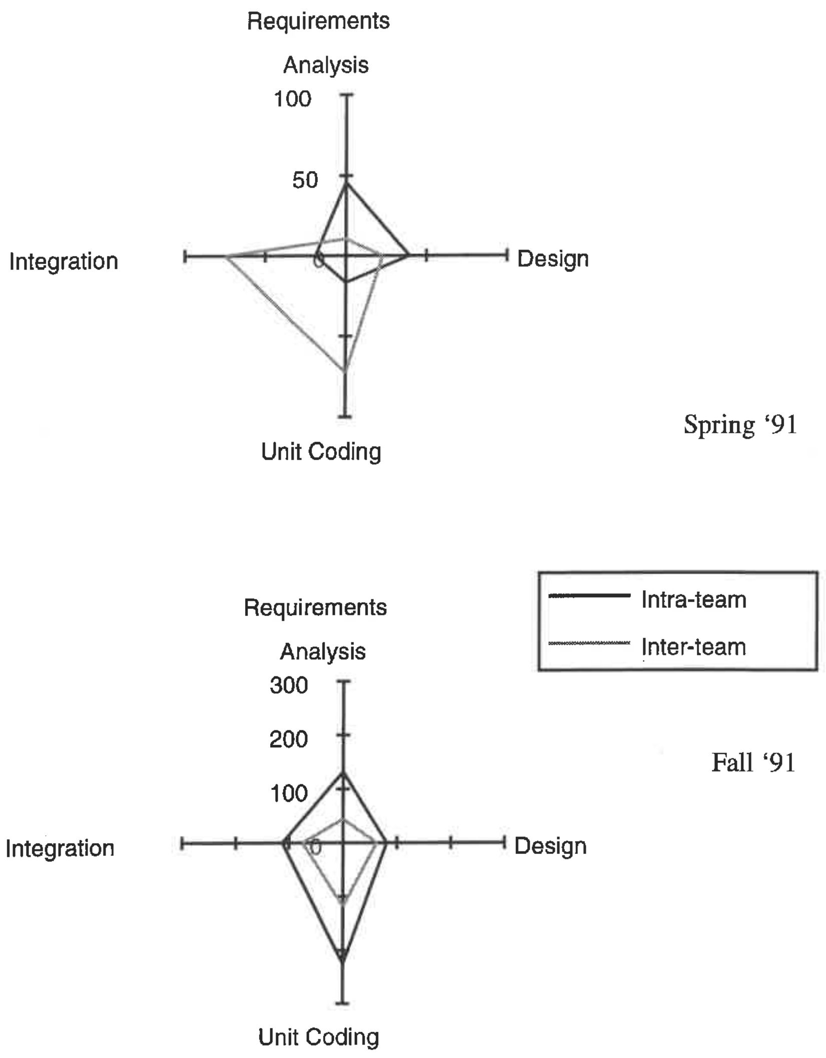

Figure 2. Spring'91 vs. Fall'91: Inter-team and intra-team Bboard communication

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 inter-team communication.

3.2. Improving Requirements: OMT and OOSE

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 [19], 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.

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 around *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 also 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.

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.

Results and interpretation

Figure 3 displays the number of messages sent per day for each phase in the project. Note that the number of messages 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.

1. It should be noted that, after the introduction and wide-spread use of OOSE, OMT was extended to include requirements methods under the form of scenarios[20]. In fact, as a result of our experiments, we started using a hybrid "OMT + Use Cases" since Fall '94 (see Figure 1, Column Methodology). This change of methodology is an example of our formative approach.

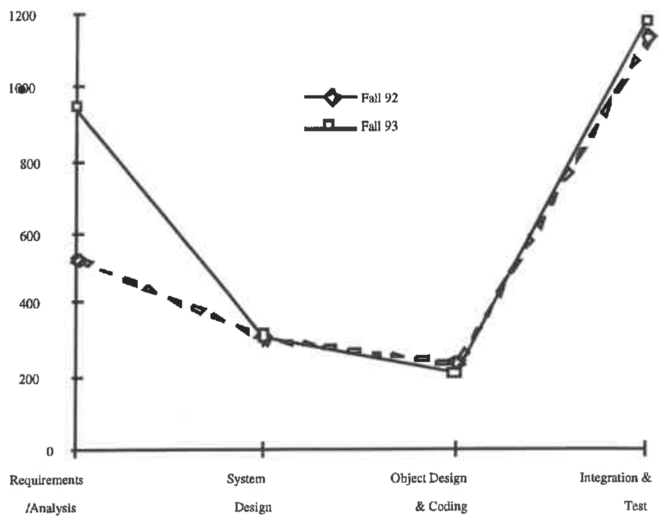

Figure 3. Fall'92 vs. Fall'93: Total number of messages per day and per phase

3.3. itIWEB: Improving Meeting Procedures

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 distributed nature of the project. Our assumption was that, by improving the planning, execution, and capture of meetings, the outcome of the project could be improved.

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)[1], for capturing meetings. Team leaders prepared the meeting by collecting a list of outstanding issues into an agenda., which 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. Also, 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.

1. itIWEB and its acronym have been inspired from the itIBIS issue-based model [21].

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 around 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.

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 sizes 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 Figure 4). 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.

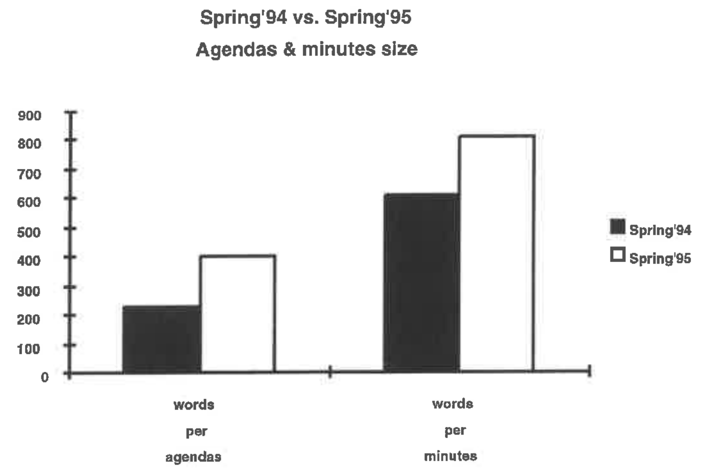

Figure 4. Spring'94 vs. Spring'95: average word size of meeting documents

In an attempt to quantify readability, we counted the number of replies for each agenda and minutes. Figure 5 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.

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.[1] This assumption was confirmed by a survey of the Spring'95 project in which 7 students (out of 7 who raised the issue) admitted not following other team Bboards regularly. Overall, the number of replies to minutes from team members increased significantly.

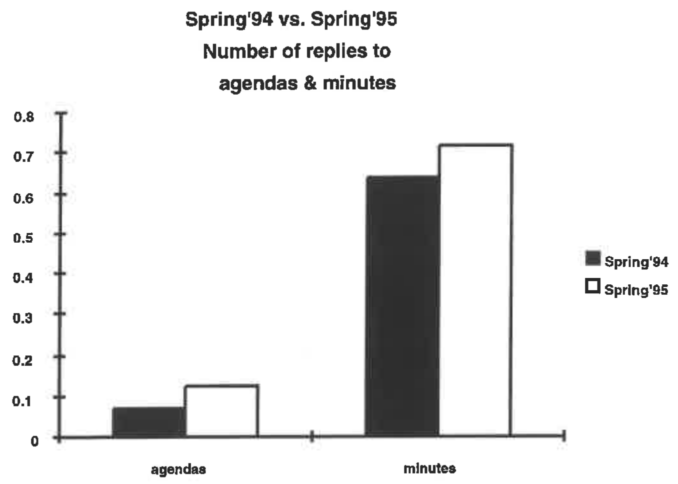

Figure 5. Spring'94 vs. Spring'95: average number of replies per meeting document

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.

1. 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.

4. Towards an Empirical Framework

Experimental design applied to software engineering faces a trade-off. On the one hand, software engineering is a young discipline in which the nature of the effects impacting the project outcome are not well understood. Moreover, the relationships between these effects and the project outcome are complex, if not unknown. This motivates the need for a large number of controlled experiments allowing statistical testing of hypotheses. On the other hand, experiments representative of real projects are expensive to conduct, and often, are not repeatable. Also, when industrial projects are the object of studies, the findings are often not completely documented because of the proprietary nature of some of the data. This has led to laboratory experiments which often include only a handful of developers working on toy projects. Moreover, findings from such controlled experiments are difficult to motivate and to generalize into well accepted practices.

The software engineering testbed described earlier in this paper represents a compromise between these two extremes. 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 own. 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 method and tools used. Also, most sources of variations may be known to the researcher, given a greater access to the project and to the non-proprietary nature of the data set. Even though some variation in the projects may not be directly controlled a priori, they can be modeled a posteriori, and thus, be taken into account.

In this section, we describe two structural equation models we developed for testing hypotheses using communication metrics. Since the deliverables of each project are available, we can validate communication metrics against traditional outcome metrics, in particular size and complexity.

4.1. Regression and Structural Equation Models

Regression models (Figure 6) have been used in software engineering [8] 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) which could have impacted the dependent variable Y. Parameters β_i measuring the contribution of each variable to the variation of the dependent variables are estimated using testbed data.

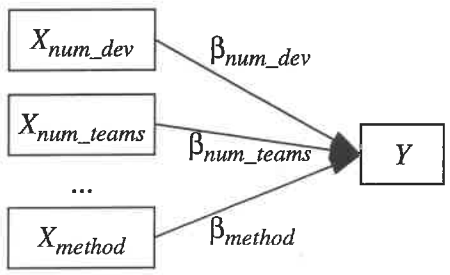

Figure 6. Regression model

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 quality. In addition, concepts such as quality are also often in turn measured in terms of a number of metrics [8]. Structural models [22], a generalization of regression and path analysis, addresses both issues as shown in Figure 7. 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 function of the set of dependent variables. For example, in Figure 7, the latent variable η is measured as a combination of the dependent variables Y_{doc} and Y_{code}.

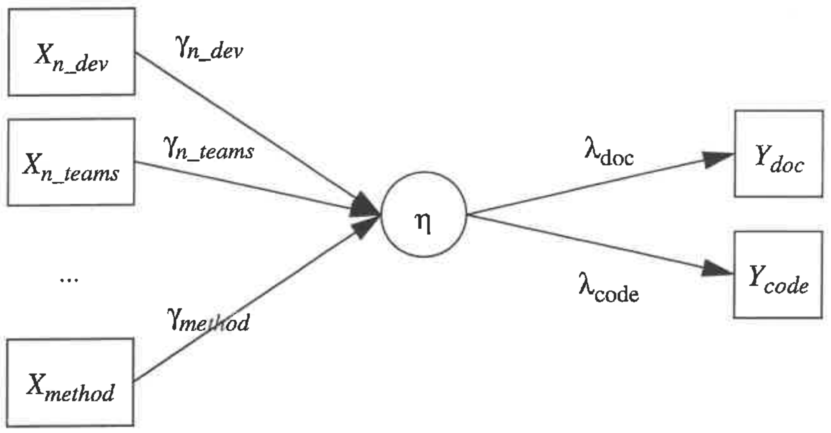

Figure 7. Structural equation model with two independent variables and one latent variable

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 affect communication which in turn affects outcome?").

4.2. Organizing Data into Observations

Given a data set such as the testbed 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).

4.3. Results

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 6 testbed projects (Fall'91, '92,'93, and '94, Spring'94 and '95; the Spring'91, Fall '95, and Spring '96 projects have not yet been included in the model). Both models are based on the general model shown in Figure 8.

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 an dependent variable which contributed to the outcome size η. All effects which 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_{n_teams}, 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 and 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}.

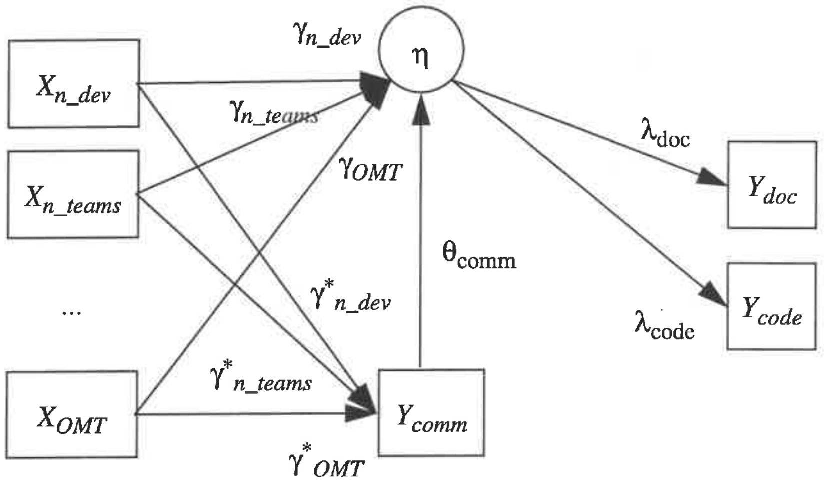

Figure 8. General structural equation model with indirect communication effect

The size and complexity of source code was measured using Halstead volume and size metrics [2]. 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 [23].

For the two models we considered 14 potential sources of variation 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 testbed project was modeled with the 3 independent variables X_{OMT}, X_{OOSE}, and X_{Hybrid}.

In the size model, 6 independent variables had a statistically significant impact on outcome, while 9 had an impact on communication (indirect effects). The communication size variable had also a significant impact on outcome. In the complexity model, 10 independent variables had an impact on outcome, while 11 independent variables had an impact on communication. Given that space does not allow a thorough discussion of the estimation of these models, we only present here a brief summary of these results. The reader is referred to [24] for a complete presentation of the methods and results of these structural models.

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} had a strong impact (3.7656) on the size of the deliverables modeled by η, which can be explained in part by the code generation facilities available in OMTool. OMT was unfortunately also responsible for an increase in deliverable complexity (0.4798). We interpret that result 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.

Parameter	Estimate in Size Model	Estimate in Complexity Model
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 2. Selected direct effects for structural equation of size and complexity models

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 due to 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.

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

Table 3. Selected indirect effects for structural equation size and complexity models

Note that the approach we adopted in building this statistical framework is that of quasi-experimental 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 formative approach, in which we feedback into the testbed lessons learned during case studies and statistical analysis.

5. Conclusions and Future Directions

In this paper, we proposed and established the value of communication metrics in the context of distributed development. We presented empirical evidence that they can provide more and deeper insight into a development process than outcome metrics. Once metrics based on communication artifact are accepted as a valid instrument for investigation, the cost of studying projects decreases significantly. Since participants in a distributed development communicate often daily, communication metrics, unlike outcome metrics, are available from the beginning of the projects. Moreover, they can be uniformly applied throughout the development. Finally, the cost of computing communication metrics is comparable than source lines of code, reducing the overall cost of data collection.

We developed and refined several communication metrics using a real testbed, the software engineering course at Carnegie Mellon. We considered this testbed as an ideal compromise between a completely controlled environment and an industrial project. We also described a statistical framework for validating communication metrics against outcome metrics, as well as studying variations in our testbed. As an illustration with preliminary estimations of the statistical framework, we showed that communication has a significant impact on outcome, that the choice development methodologies and communication infrastructure has a significant impact on both communication and outcome, and most important, that the size and structure of the organization also impacts the communication and outcome.

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 distributed developments to allow the use of communication metrics during projects (vs. only for post mortem studies). Preliminary data indicate that, when sampled on a weekly basis, communication patterns are changing substantially when teams are handling crises [24].

We are planning to experiment with the use of different communication infrastructures (e.g., threaded discussions on the World Wide Web, information modeling environments such as *n*-dim [1]), different methods (e.g., evolutionary prototyping), and tools (e.g., prototype-based object-oriented languages). Finally, in the attempt to address the issue of repeatability and lack of public domain data sets [25], we intend to distribute the testbed raw data in the form of World Wide Web pages.

6. References

[1] 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,* Edited by A. Tzonis and I. White, Elsevier Publishers, Amsterdam, 1993.

[2] M. H. Halstead, *Elements of Software Science*: North-Holland Publishing Company, 1977.

[3] T. J. McCabe, "A Complexity Measure," *IEEE Trans. in Software Engineering*, vol. 12, pp. 308-320, 1976.

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

[5] V. R. Basili and D. Hutchens, "An Empirical study of a Syntactic Complexity Family," *IEEE Trans. on Software Engineering*, vol. 9, pp. 664-672, 1983.

[6] M. Itakura and A. Takayanagi, "A Model for Estimating Program Size and Its Evaluation," Proceedings of the Sixth International Conference on Software Engineering, Tokyo, Japan, pp. 104-109, 1982.

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

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

[9] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[10]R. Tausworthe, "Software Specifications Document, DSN Software Cost Model," Jet Propulsion Laboratory, Pasadena, California JPL Publication 81-7, 1981.

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

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

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

[14]R. Coyne, A. Dutoit, B. Bruegge, and D. Rothenberger, "Teaching More Comprehensive Model-Based Software Engineering: Experience With Objectory's Use Case Approach," 8th Conference on Software Engineering Education (CSEE'95), New Orleans, Lecture Notes in Computer Science #895, Springer Verlag, pp. 339-374, 1995.

[15]A. Dutoit, B. Bruegge, and R. Coyne, "Using an Issue-Based Model in a Team-Based Software Engineering Course," Proceedings of Software Engineering: Education and Practice (SEEP'96), Dunedin, New Zealand, IEEE Computer Society Press, pp. 130-137, 1996.

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

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

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

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

[20]J. Rumbaugh, “Getting started: Using use cases to capture requirements,” *Journal for Object Oriented Programming*, vol. September, pp. 8-10, 23, 1994.

[21]B. Yakemovic and J. Conklin, “Report on a development project use of an issue-based information system,” Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW), Los Angeles, CA, pp. 105-118, 1990.

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

[23]N. Coulter, I. Monarch, S. Konda, and M. Carr, “An evolutionary perspective of software engineering research through coword analysis,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh CMU/SEI-95-TR-019, 1995.

[24]A. H. Dutoit, “The rôle of communication in team-based software engineering projects,” Ph.D. thesis, *Department of Electrical and Computer Engineering*. Pittsburgh: Carnegie Mellon University, 1996.

[25]W. C. Hetzel, “The Sorry State of Software Practice Measurement and Evaluation,” *Journal of Systems Software,* no. 31, pp. 171-79. 1995.