12-1988

Programmer and End User Communication in Computer System Development

Theresa J. Hart
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1210

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
Many computer professionals agree that good communication between computer programmers or analysts and the people for whom a system is being created is essential for system success. This study researches possible reasons behind the communication problems that exist between programmers and end users and suggests ways to narrow the communication gap.

Many theories, including the theories of hemispheric brain dominance, thinking styles, and motivational factors may help explain the differences in communication styles between programmers and users. To further explore the communication problem, an exploratory survey was given to programmers and users to evaluate their perception of the communication that took place during system development.

Based on the available literature, the author's ten years of experience in the field, and the results of the exploratory survey, several methods are proposed to improve communication between programmers and end users in system development.
ACKNOWLEDGEMENTS

I would like to dedicate this thesis to my parents, who taught me how to communicate with and appreciate other people.

I would also like to express my deep appreciation to my thesis advisor, Dr. Donna Kaminski, and to my degree advisor, Dr. Dalia Motzkin, for all their help and patience.

Theresa J. Hart
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700  800/521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Programmer and end user communication in computer system development

Hart, Theresa J., M.S.
Western Michigan University, 1988
Table of Contents--Continued

CHAPTER

Tap Your Right Brain: Intuition......................... 38
Don't Overrun the User................................. 39
Fit the User's Concept of Reality...................... 41
Evolving Systems......................................... 43
User Participation......................................... 46
Feedback.................................................. 50
Close the Gap in the Coding Phase..................... 52
Prototype.................................................. 53
Let Joe Do It............................................. 54
Useful Interfaces......................................... 55
Close the Gap in the Test Phase......................... 56
Making the Right and Left Brains Fit..................... 58

V. CONCLUSION.............................................. 61

APPENDICES

A. Survey Questionnaire................................. 65
B. Survey Cover Letter................................... 72
C. Survey Database Listing.............................. 74
D. Human Subjects Institutional Review Board
   Approval Letter....................................... 76

BIBLIOGRAPHY ............................................. 78
# LIST OF TABLES

1. Perceived System Usefulness Compared to Perceived Amount of Communication......................... 20

2. Perceived System Usefulness Compared to Perceived Quality of Communication...................... 21

3. Perceived Quality of Communication Compared to Perceived Amount of Communication............. 22

4. Perceived System Usefulness Compared to Perceived Match on Hemispheric Preference.................. 22

5. Perceived System Usefulness Compared to Perceived Match on Thinking Style.......................... 23
CHAPTER I

THE PROBLEM

Background of the Problem

A couple years ago, I was asked to present a newly developed software system to the work unit for which it had been created. As it took over two years to develop, it was decided that I, a member of the Computer Information Center, rather than one of the developers, should give the demonstration in order to ease any ill feelings toward the system. I reviewed the system and prepared my presentation. It was, in my estimation, put together quite well and fairly "user friendly", and it was well received by the end users. What bothered me, though, were the many questions they asked regarding whether the system could still do certain things that the old system could do and whether the system had certain desired capabilities not present in the old system. In many cases, the answers to these questions were "no." I couldn't help thinking, as I listened to the dialog among the unit staff and the developers, "Didn't they talk to each other? Didn't they discuss these things?"

We have all heard of such situations. Lazarus and Tomeski (1975) describe the development of a computerized model of the supply and distribution area of a large petroleum company. After years of
development, the supply and distribution staff found the model to be too theoretical and not applicable to real-world problems. Just another example of how years of work can be wasted because of ineffective communication between programmers and the users for whom systems are developed.

Statement of the Problem

The previous scenario is often repeated in system development. The end user requests a system; the programmer gets the specifications and develops it; the new system is presented to the user, and, then, it isn't what the user really wanted. The programmer complains that the user doesn't know what he or she wants and the users complains that the programmer didn't give them what they asked for. Why doesn't it end up as expected? It is hypothesized that a large share of the problem lies in the differences in communication patterns between programmers and end users. Wessner (1987) poses the communication problem as follows:

There can never be too much communication between DP and the rest of the corporation....For the company as a whole to succeed, DP must communicate effectively with the corporation it supports. Let's look at the user side of the DP-corporate relationship. Each user has an implicit set of expectations regarding the types and levels of DP support. The user may expect a daily report by 10:00 every morning or a terminal response time for the on-line order entry system of less than two seconds. But unless the DP department knows of these expectations, and what drives them, it's highly unlikely that the corporate user will ever be entirely satisfied. On the other side, DP develops over time its own set of perceived user requirements (which may or may not be realistic), based on previous experience with each user. These perceived requirements are then used to develop DP's operating timetable and list of priorities. If DP doesn't meet user expectations, then DP is viewed as a problem rather than an asset. At the other extreme, this
lack of understanding may cause DP to spend unnecessary time, effort, and resources in providing over responsive levels of support in non-critical areas. In many companies, the failure to communicate means both situations apply--DP wastes resources in some areas and underachieves in others. (pp. 16-17)

Other authors (Blank, 1988; Lutz, 1986; Martin, 1984; Richardson, 1986) agree that good communication skills are essential to system development, and the lack of it is a major reason for problems in system development. They agree that no level of expertise in other areas can offset a lack of effective communication. As Gruenberger (1971) asserts, "a project organization implies a great deal of interpersonal interaction and managers distinguish themselves as much by the excellence of coordination and communication within the project as by the technological worth of the project" (p. 85).

In one unpublished study of programmers, Peres (cited in Gruenberger, 1971) asked each of approximately 200 programmers to write essays describing the "best programmer he/she had ever known." Communication skills was one of the five major categories identified. Communication skills questions can also be found in computer aptitude tests, given to test the ability of an individual to be a good programmer.

A study done of programmers at Bell Labs (Bradley, 1986) pointed out that 32% of their time was spent in job communications. That means that about a third of a programmer's productivity depends on his or her ability to communicate well with others.

Communication problems exist in all fields of work but those problems encountered in the field of computer system design and development have more severe effects because the systems that we
create today can only deal with concrete information. They cannot deal well with vague data. Programmers must know what a user wants and needs in order to add value to the user's work and to make that user more productive. While explaining the topic of this thesis to another programmer, the author was asked what communication had to do with computer science. That is a symptom of a fundamental problem in the human/computer environments. That programmer truly believed that communication skills were not valuable or necessary to system production, just good programming code. The author asserts that not only are communication skills important to the field of computer science; communication skills are perhaps the most important part of computer science. If programmers understand what the user wants and gives him or her only part of that, the system is successful at least in part. But if the programmer does not understand what the user wants and gives him or her an unusable system, all the good computer coding in the world and all the extra features are just a waste of time.

The Project

This paper is written for computer programmers and analysts. It first points out how they basically differ in communication style from many of the users they serve, and then proposes ways in which those differences can be overcome in system development. In the related reading and in the ten years of experience in the field, the author has never encountered an article or book strictly devoted to the subject of improving communication in the system development.
process, i.e., from the first conception of the system idea through the final testing of the product. This paper is an attempt to address that problem.

Relatively few people in the mainstream computer science community are currently concerned with the notion of applying psychology to system development or investigating the human computer interaction in that development process. However, there is a growing body of research and literature on such phenomenon that includes both psychologists (Deutsch & Springer, 1985; Donovan & Wonder, 1984; Masie, 1988) and computer scientists (Chomsky, 1968; Floyd, 1985; Schneiderman, 1987; Tognazzinni, 1986; Weizenbaum, 1976). Chapter II of this thesis explores that body of knowledge and the accompanying theories which explain the differences between the communication patterns of programmers and that of most of the rest of the population. Much work has been done in the areas of hemisphericity of the brain and personality types to try to explain the differences in the way people think and communicate. There is some evidence that programmers and analysts differ in those areas from the majority of the population. Studies have also shown that programmers are different from the general population in what motivates them. A review of those studies is provided in chapter II of this thesis. As Martin (1984) states, "When the traditional systems analyst and potential end users first come face-to-face, they come from widely different cultures. It is rather like a Victorian missionary first entering an African village" (p. 55).

Based on the literature cited, it is hypothesized that major
communication problems exist between programmers and end users in system development and that part of the reason for the problem could be that many programmers differ from users in their communication patterns. To investigate this further, a survey was conducted in a large midwest company. Thirty-three programmers and users of three different major systems developed in the last two years were sent surveys. They were asked to evaluate the usefulness of a particular system when it was first developed and what level of communication transpired between the programmer and the user. They were also asked to evaluate themselves and the person with whom they worked most closely as to his or her thinking styles. Chapter III of this thesis describes that study in more detail, discusses the limitations of the study and lists the results.

In Chapter IV of this thesis, various methods are proposed for improving the communication process in system development. The ideas and solutions are those of the author and those compiled from a number of authors and researchers in the computer field. It is the product of not just the review of the literature done for this thesis, but of ten years working in the field, programming and dealing with users; ten years of reading and listening to noted authorities and fellow programmers.

Chapter V summarizes the gravity of the problem and the proposed solutions. Lack of effective communication is a serious problem. Programmers can choose to ignore it and develop well coded systems that never get used or are difficult to use, or they can try to attack the problem and use the solutions suggested in this thesis to
narrow the communication gap. As Floyd (1985) states:

People act in open situations, on the basis of their needs and commitments, pursuing changing goals as they go along and revising their understanding in the event of new insights. We have to develop a new theory drawing largely on concepts that are useful for studying living systems, as they exist in nature and society, and language as it is used in communication between people. (p. 10)
CHAPTER II

THE THEORIES

Chomsky (1968) states, "as far as we know, possession of human language is associated with a specific type of mental organization, not simply a higher degree of intelligence" (p. 62). People think and learn differently, depending on various psychological factors. The way they communicate is influenced by their particular orientation. We can improve the system development process by "using psychological methods for improving the communication between groups of people enacting different roles" (Floyd, 1985, p. 10).

Three theories are identified that help explain the communication differences between programmers and end users: (1) the hemisphericity of the brain, i.e., the assumption that one hemisphere of the brain contributes more than another to the way an individual thinks and solves problems, (2) proposed theories for differences in thinking styles, and (3) motivational factors for programmers.

Right Brain, Left Brain

The first theory is that of hemisphericity of the brain—the idea that a given individual relies more on one mode or hemisphere than the other. As stated by Deutsch and Springer (1985):
This differential utilization is presumed to be reflected in the individual's cognitive style - the person's preferences and approach to problem solving. A tendency to use verbal or analytical approaches to problems is seen as evidence of left sided hemisphericity, while those who favor holistic or spatial ways of dealing with information are seen as right-hemisphere people. Hemisphericity has been claimed by different sources to extend not only to perception but to all kinds of intellectual and personality dimensions. (p. 239).

Several authors (Deutsch & Springer, 1985; Donovan & Wonder, 1984; Masie, 1988; Weizenbaum, 1976;) describe the differences between right brain and left brain orientation. They concur that the left hemisphere is skilled at sequential processing in general and is therefore the more analytic of the two hemispheres. This analytic mode of information processing is thought to apply to all incoming information, including speech. In contrast, areas of the right hemisphere become more adept at simultaneously processing the type of information required to perceive spatial patterns and relationships. The right hemisphere is the more holistic and synthetic of the two in handling all kinds of information. The left brain is the center for spoken communication and the right brain is the center of visual communication.

The left brain is verbal, using words to name, describe and define, whereas the right is nonverbal, with an awareness of things but minimal connection with words. The left is linear and analytic, thinks in terms of linked ideas with one thought following another and often leading to a convergent conclusion, that has figured things out step-by-step and part-by-part. On the other hand, the right brain is holistic and synthetic: it sees whole things at once, perceives the overall patterns and structures, and often comes to
divergent conclusions. The left brain is symbolic and abstract, using symbols to stand for things: it takes a small bit of information and uses it to represent the whole thing. The right brain is concrete and analogic; it relates to things as they are at the present moment and sees likenesses between things. The left brain is logical, drawing conclusions based on deduction, but the right makes insightful leaps, often based on incomplete patterns, hunches, feelings, or visual images. The left is rational, drawing conclusions based on reason and facts; the right is nonrational, not requiring a basis of reason or facts. The left is temporal, keeping track of time, sequencing one thing after another; the right is nontemporal, without a sense of time.

Some studies (Donovan & Wonder, 1984) show that individuals in occupations involving highly structured procedures, such as accountants and chemists, are clearly left dominants. Those in less structured work, such as athletes and painters, are right dominants. It was also found that there were differences within specific occupations. For example, corporate lawyers were more left brained than criminal lawyers. Classical musicians were left dominant, while rock musicians were right dominant.

On a whole, computer programmers and analysts tend to be left dominant (Tognazzinni, 1986). They are verbal and symbolic, describing and defining things in words or symbols. They and the computer languages they use to create applications and systems are linear and logical. Problem processing precedes in a step wise manner, conclusions are based on facts and reason, and all the
Information must be available before a decision is made.

Such a scenario is not like most real world situations where the available information is not clear, the situation is in a constant state of change, but the whole picture must be considered. The problems encountered in the field of artificial intelligence are due to just such situations. Most real life situations are best dealt with by the right brain, but our current computer systems deal best with the left brain.

Tognazzini (1986), in a paper presented at the Office Automation Conference, maintained that human styles of perception fell into two categories, as originally perceived by Carl Jung, with 80% of the population being sensory or concerned basically with concrete reality and the remainder being intuitives, the Einstein type thinkers. Comparatively, intuitives are left brain dominate and sensories are right brain dominant. He also pointed out that in one study of government programmers, it was discovered that there was a higher percent of intuitives (left brain dominants) than in the normal population. He referred to a study at Apple Computer which showed that 90% of their programmers were intuitives. On the other hand though, he claimed that over 90% of all businessmen were sensory (right brain dominants).

Other indicators and conclusions (Danziger & Greenwood, 1967; Demb, 1979; Gruenberger, 1971) also point to hemisphericity differences between programmers and the people for whom they develop systems. The Computer Programmer Aptitude Battery, developed by Jean Palormo (cited in Gruenberger, 1971), tests for competence or ability
in 5 major areas: (1) the ability to think logically, (2) the ability to translate ideas and operations from work problems into mathematical notations, (3) the ability to use abstract reasoning, (4) the ability to use numbers, and (5) the ability to analyze a problem and order the steps for solution in a logical sequence. Those are all functions of the left hemisphere of the brain. What Palormo suggests in this test is that left brain dominant individuals make the best programmers. The SET Guide, a manual prepared by the Selection, Education and Training Committee at Guide International in 1965 (cited in Danzinger & Greenwood, 1967) describes the main desirable characteristics of data processing personnel. The majority of the items are left brain characteristics.

Styles of Thinking

Besides the differences due to dominance of one hemisphere of the brain over another in an individual, some authorities (e.g., Masie, 1988; O. D. Resources, 1988) claim that there are also four different styles of thinking that contribute to both the hemispheric preference and to environmental factors:

1. The reflective thinker integrates experience with self, seeks meaning and clarity, and needs to be personally involved and learns by listening and sharing ideas.

2. The conceptual thinker forms theories and concepts, seeks facts and continuity, needs to know what the experts think, and values the sequential nature of thinking and detail.

3. The practical thinker practices and personalizes, seeks
usable information that has utility, needs to know how things work and edits reality.

4. The creative thinker seeks hidden responsibilities, responds to new and different, is adaptable to change, and learns by trial and error. (Masie, 1988)

In their class, Managing the Human Aspects of Technological Change, O. D. Resources (1988) refers to four thinking styles: (1) thinker, (2) feeler, (3) intuitor, and (4) sensor. The thinker is logical, avoids emotionalism, is skeptical of any solution until it is tested and analyzed, and prefers to think things through before making decisions. The feeler is dynamic, stimulating, sensitive to others' needs and wants, patient, a good listener, and acts on the basis of gut feeling or intuition to make decisions. The intuitor is a fast, deep thinker, who enjoys creating structure out of disorder and looks at the whole picture before making decisions. The sensor is a doer, thriving on a variety of projects at once, who learns best by doing and is direct, decisive and well organized. (pp. 31-34)

Many parallels can be seen in the two proposed thinking styles. The reflective thinker of the first theory and the feeler of the second match. The conceptual thinker of the first can be combined with the thinker of the second. Finally the practical thinker can be lined up with the intuitor and the creative with the sensor.

No evidence was found that programmers tend to be of one thinking style or another, although many would fit in the styles that show left brain tendencies. Thinkers have left brain tendencies and
feelers have right brain tendencies. In general, people are most receptive to others whose thinking style is similar to their own (O.D. Resources, 1988). Differences in the thinking styles of the programmer or analyst and the user of the system can lead to communication problems, just like they can lead to communication problems in any other business setting. If the programmer is aware of his/her own style and that of the user, he/she can adapt and avoid possible problems.

Motivational Factors

The final theory for exploration is that of motivational factors—what makes a person accomplish things. According to a famous survey by Hackman and Oldman (cited in Cougar & Zawicki, 1980), programmers are distinctly different in what motivates them from the rest of the population in other job categories in what motivates them. The results of the Job Diagnostic Survey (cited in Cougar & Zawicki) showed that data processing (DP) professionals have substantially higher Growth Need Strength scores than any of the five hundred job categories surveyed. This is true of analysts, programmer/analysts and programmers. On the other hand, that same group had the lowest Social Need scores among professionals, and it is significantly lower. The low social need and high growth need of DP personnel may not be surprising to the observers of the DP profession, but what is surprising is the degree to which DP personnel differ from their peers in other parts of a company.

Another factor identified by Demb (1979) influencing programmers
is that they tend to identify themselves by their profession rather than by their organization. That gives them a view of institutional authority as a form of control which insures the proper integration of individuals into a system. In that mode, they are more apt to resist bureaucratic authority than their colleagues in other professions.

If the programmer or analyst has a very low social need, then he or she will only communicate with the user when he or she feels it is absolutely necessary, and then as little information as possible is communicated. The programmer's high growth need motivates them to over produce--to develop systems with all sorts of extras that only tend to confuse the user.

These studies and tests offer some evidence that programmers basically differ in ways of thinking and, therefore, in their communication patterns from the majority of the people that for whom they design systems. And yet, the very key to the successful implementation of that system depends on the programmer's ability to understand and communicate effectively with those people.
CHAPTER III

THE SURVEY

As the theories and studies suggest, communication differences exist between programmer/analysts and the general user community. In order to explore this theory further, a small survey was conducted to determine perceived communication styles and the quality of communication during the system development process.

Survey Protocol

A questionnaire was developed that asked programmers and users of recently developed systems to rate the usefulness of the system when it was first released for use, and the amount and quality of communication between each person and his/her main contact person in the other group (programmers or users). In addition, each person was asked to choose one of two categories that best described his/her way of thinking and one of four categories that best described his/her type of thinking. The descriptive used in those questions reflected the general characteristics of the two different hemispheric preferences and the four different thinking styles. Then respondents were asked to rate their principal contact person in the same two areas. (See Appendix A for the full survey.) A cover letter was attached to the survey, explaining its purpose and the level of
confidentiality that would be used. (See Appendix B for a copy of the cover letter.)

In the questionnaire, each person was first asked to identify himself or herself with the understanding that his/her name, the system name and the company name would not be used in the thesis or in the computer database holding the survey results. Each was asked to rate, on a scale of 1 to 10, with 1 being very poor and 10 being excellent, the usefulness of the system when it was first developed. The second question asked each to rate the amount of communication between the respondent and either the end users or programmers when the system was being developed, with 1 the lowest estimate of communication and 10 the highest, i.e., a rating of 1 for "none" to 10 for "a lot." The respondents were also asked to rate the quality of that communication, using the same scale as that for the first question. Each participant was then asked to rate his/her way of thinking by choosing from two groups of descriptive, corresponding to right brain and left brain hemispheric preference indicators, drawn from the literature reviewed in the previous chapter. Each was also asked to rate his/her style of thinking, based on the four thinking styles indentified in this paper. Each was queried for the name of the person from the other group (programmers or end users) with whom he/she had the most contact during the system development and asked to rate that person's way of thinking and thinking style, using the same categories as he/she had used in their own evaluation. Finally, each participant was asked how he/she thought the communication process between programmers and
end users could have been improved in the development of the system.

Three major systems, developed within the last two years for one company division, were used for the survey. Management currently responsible for those systems was asked to identify all programmers and users who were involved in the initial version and were still employed in the division. Thirty-three employees were listed and received the survey. Contacted were: ten users and one programmer involved in the initial development of system #1, ten users and four programmers of system #2, and four users and three programmers of system #3. Of the total, four programmers and four users indicated that they were not a significant part of their system's first phase, and thus, were not used in the results. All the remaining people returned their surveys. That left a final survey population of eight users and one programmer for system #1; seven users and three programmers of system #2; and four users and one programmer for system #3.

After the surveys were collected, the information was entered into a computer database, with numbers instead of names used to identify individuals and systems. (See Appendix C for a complete database listing.) Although comments were not entered on the actual database, for reasons of confidentiality, they will be referred to in the survey results. Two additional variables were created: "match1" indicated whether the participant and his/her contact matched on perceived way of thinking and "match2" indicated whether they matched on thinking style.

Frequency tables were produced that compared the rating of the
usefulness of the system to the rating of the amount and quality of communication. The latter two were also compared to each other. Ratings were grouped into categories for easier analysis. For instance, for the question regarding system usefulness and for the question regarding quality of communication, a 1 to 3 rating was considered "poor," a 4 to 7 rating "ok" and a 8 to 10 rating "good." The comparisons were done for the whole group as well as for the data partitioned out by system and by occupation (programmer or user). The usefulness rating and the quality of communication rating were also compared to whether the survey participant rated himself or herself the same on the way or style of thinking as he or she rated his/her principle contact from the other group.

Survey Results

Several frequency tables were developed comparing the various variables and searching for any possible relationships in the results. Due to the small sample size, the tables which compared the various variables by system will not be discussed. Also, since only 5 of the 24 persons interviewed were programmers, the tables which show the programmers' ratings separated from the entire group will not be discussed either. Each table discussed will be that of the entire group together and any major deviations from the results, when the users are taken separately, will be mentioned.

When comparing the rating of perceived system usefulness against the rating for the perceived amount of communication that transpired in system development, everyone who rated a system "good" and most of
the persons surveyed who rated a system "ok," rated the amount of communication as "some" or "a lot" (see Table 1). Only one programmer and one user rated the system usefulness as "good." When the same table was created, splitting apart programmers and users, the user ratings were within a few percentage points of those on Table 1, except for the one "good" system rating.

Table 1
Perceived System Usefulness Compared to Perceived Amount of Communication

<table>
<thead>
<tr>
<th>Amount of Communication</th>
<th>System Usefulness</th>
<th>None or Little</th>
<th>Some</th>
<th>A Lot</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Poor</td>
<td>50.00%</td>
<td>20.00%</td>
<td>10.00%</td>
<td>20.83%</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>50.00%</td>
<td>70.00%</td>
<td>80.00%</td>
<td>70.83%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>0.00%</td>
<td>10.00%</td>
<td>10.00%</td>
<td>8.33%</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Number</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

The rating of perceived system usefulness compared to the perceived quality of communication shows that in both cases where the system was perceived "good," the quality of communication was perceived "good." In over half the cases where the system was perceived "ok," the quality of communication was perceived as "ok" or
A comparison of the perceived amount of communication in the system development and the quality of that communication showed some relationship between quality and quantity but not as much as in the previous tables discussed (see Table 3). The users taken separately show the same range within 0 to 7 percentage points, except for the rating of "good" communication quality against "a lot" of perceived communication. That rating was 60% of the total for all persons interviewed but 75% of all users interviewed.

Table 2
Perceived System Usefulness Compared to Perceived Quality of Communication

<table>
<thead>
<tr>
<th>Quality of Communication</th>
<th>System Usefulness</th>
<th>None or Little</th>
<th>Some</th>
<th>A Lot</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>40.00%</td>
<td>11.11%</td>
<td>22.22%</td>
<td>21.74%</td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td>60.00%</td>
<td>88.89%</td>
<td>55.56%</td>
<td>69.57%</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>0.00%</td>
<td>0.00%</td>
<td>22.22%</td>
<td>8.70%</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>23    (1 missing)</td>
<td></td>
</tr>
</tbody>
</table>
Table 3
Perceived Quality of Communication Compared to Perceived Amount of Communication

<table>
<thead>
<tr>
<th>Amount of Communication</th>
<th>None or Little</th>
<th>Some</th>
<th>A Lot</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>33.33%</td>
<td>30.00%</td>
<td>10.00%</td>
<td>21.74%</td>
</tr>
<tr>
<td>OK</td>
<td>33.33%</td>
<td>50.00%</td>
<td>30.00%</td>
<td>39.13%</td>
</tr>
<tr>
<td>Good</td>
<td>33.33%</td>
<td>20.00%</td>
<td>60.00%</td>
<td>39.13%</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Number</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>23</td>
</tr>
</tbody>
</table>

(1 missing)

Table 4
Perceived System Usefulness Compared to Perceived Match on Hemispheric Preference

<table>
<thead>
<tr>
<th>Match on Hemispheric Preference</th>
<th>No</th>
<th>Yes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>10.00%</td>
<td>28.57%</td>
<td>20.83%</td>
</tr>
<tr>
<td>OK</td>
<td>70.00%</td>
<td>71.43%</td>
<td>70.83%</td>
</tr>
<tr>
<td>Good</td>
<td>20.00%</td>
<td>0.00%</td>
<td>8.33%</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Number</td>
<td>10</td>
<td>14</td>
<td>24</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Finally, the usefulness of the system was rated against the interviewee's perception as to whether the principal person with whom they communicated with was like them in hemispheric preference and thinking style. In the first case, hemispheric preference, little relationship can be seen in comparing the two tables, (Table 4). Even less relationship can be seen when matching on thinking styles (Table 5).

Table 5
Perceived System Usefulness Compared to Perceived Match on Thinking Style

<table>
<thead>
<tr>
<th>Match on Thinking Style</th>
<th>No</th>
<th>Yes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Usefulness</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poor</td>
<td>15.79%</td>
<td>40.00%</td>
<td>20.83%</td>
</tr>
<tr>
<td>OK</td>
<td>73.68%</td>
<td>60.00%</td>
<td>70.83%</td>
</tr>
<tr>
<td>Good</td>
<td>10.53%</td>
<td>0.00%</td>
<td>8.33%</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Number</td>
<td>19</td>
<td>5</td>
<td>24</td>
</tr>
</tbody>
</table>

Many suggestions for improving communication were made in the survey comments. The participants suggested weekly meetings, user input on test data, periodic demonstrations of screens and output reports during development, and establishment of a user liaison person in daily contact with programmers. Others said that
programmers should approach the user with an open mind and "can do" attitude; that the system should serve the needs of the users, not what programming leadership perceives to be the user's needs. In general, users expressed a need for more communication and more participation on their part in the system design.

The few programmers surveyed suggested a class to teach them the "business" of the users and more meetings with all persons involved. There was also expressed dissatisfaction with the communication process established for them, such as only one programmer assigned to obtain end user problems, and too many management people involved in the development, thus intimidating the "real" end users.

Survey Discussion

The most significant finding of the survey was that, in every case where the usefulness of the system was rated poorly (6 or less), the participant made a written comment to the effect that the programmer of the system needed to have a better understanding of the user's needs and a better perception of how end users will make use of the system once it is online. That supports the premise that lack of good communication can lead to poor system perception. Table 1 and Table 2, comparing the relationship between the perceived rating of the usefulness of the system and the amount and quality of the communication that took place, also support that premise.

A major limitation was that the sample size of the survey conducted was too small to make any statistically significant
generalizations from the data collected. Without a larger population to work with, the unusual rating or one different from the norm cannot be easily identified. The author had originally intended to survey a system involving several hundred people, but the completion dates had been postponed until after this thesis was to be approved. However, the survey results from this exploratory study are suggestive that certain patterns exist between perception of system usefulness and amount and quality of communication.

Another limitation was that all three systems were developed long enough ago so that it might have been difficult for the people involved to remember accurately their perceptions at that time. Also, several programmers and users involved in development had moved out of the company since the time when each system was put into production. Management had even erroneously identified some of the participants involved in the first phase.

The survey questions which ask the programmer or user to categorize his/her problem solving and thinking styles are not of proper length or breadth to conclusively identify a person's hemisphericity preference or thinking style. In order to get an accurate evaluation, each person would have to be given a lengthy more formal questionnaire, like the Myers-Briggs Indicator (McCaulley & Myers, 1985). That was out of the question in the environment chosen, as the employees would not have been allowed to take that much time from their jobs to complete it. For this exploratory study, the author wanted to see if at least a brief categorization, like that used in the survey, would be indicative of some
relationships.

There is always a difficulty in people rating themselves or others, so it is hard to judge how accurate the findings are. People often rate someone to whom they relate well as more like themselves than someone to whom they do not. People also do not like to make judgments regarding themselves or others. As one programmer interviewed said, "I don't want to know how I solve problems. I don't want to think that deeply about it." Unfortunately, this is often a problem with social/psychological survey research.

If the work on this problem is to be expanded, the following alterations should be considered. If there could be fewer restraints on what could be asked, each person should be given a Myers-Briggs Type Indicator test (McCaulley & Myers, 1985) to determine the hemispheric preference of the participant. Under those circumstances there might be a stronger relationship between perceived system usefulness and how well the participant and his/her contact matched on the Myers-Briggs Type Indicator test scores. Also, each participant would not have to rate another and it could better be evaluated as to whether a match on thinking styles really exists. To allow for a more thorough statistical analysis, the sample size should be enlarged. Then the questions regarding who the person had most contact with, system usefulness, amount of communication that transpired in system development and quality of communication would be able to be more thoroughly investigated. To allow for the better assessment of the variables under study, the survey should be given within six months of the introduction of a new system.
CHAPTER IV

GUIDELINES

Overview

Based on the literature and theories reviewed in Chapter II, the survey results of Chapter III and the author's ten years of experience working in industry in the field of computer science, this final chapter proposes ideas for improving the communication process between programmers and end users in the system development process, and offers supportive literature for those ideas.

1. It is suggested that it is important for the programmer to understand the environment in which he or she is working. That includes knowing how he or she solves problems, views the world and finds motivation to get things done, as well as understanding the user in each of those areas, as much as is possible. The programmer also needs to know his or her computer, i.e. understand the limitations of the machine itself. The programmer must understand the business needs of the environment in which he or she works.

2. Various ways of closing the communication gap in the different stages of system development must be addressed. In the design phase, it is recommended that the programmer tap the intuition of his or her right brain and be careful not to overrun the user with
his or her left brain logic or language specific to his or her profession. The system should be developed to fit the user's concept of reality, not the programmers. It should evolve gradually, with frequent user participation and feedback.

3. In the coding phase, it is suggested that prototyping be employed, that the user possibly be used to develop part of the code, and that the human interface be well designed.

4. The testing phase should include as much and varied user participation as possible.

5. If communication is to improve, system development should attempt to make the right and left brains fit. Programmers should be matched to those phases of system development which best fit their orientation to problem solving. Communication skills can be developed. The universities should make that part of a computer science curriculum.

Understanding the Environment

Know Yourself and Your Users

The first task for any programmer should be to discover what his/her personal orientation is to the world, what hemisphere of the brain dominates his/her thinking and what style of thinking best describes him/her. As much as is possible, the same should be done regarding the users with whom he/she will communicate with during the system development. Schneiderman (1987) states that "many designers assume that they understand the users and their tasks. Successful designers are aware that other people learn, think and solve problems
in very different ways" (p. 34).

Donovan and Wonder (1984) suggest that there are always two forms of communication going on at the same time: the planned, conscious one from the left brain and the unintentional, unconscious one from the right brain. If the messages sent by the two hemispheres are contradictory, the receiver is more likely to rely on the subliminals given by the right brain—the tone of voice, gestures, and emphasis. If programmers understand the two forces involved and how they each contribute to the communication process, they will be able to improve that same process.

Various testing instruments are available to measure a person's orientation. Brain domination can be measured with the Kersey Sorter (Masie, 1988) or the Myers Briggs Type Indicator (McCaulley & Myers, 1985). Thinking styles can be measured with the Survey of Communicating Styles (O. D. Resources, 1988).

Programmers can't expect potential system users to take one of these tests, so they may have to rely on their own perception of the user to make a determination of what his or her thinking orientation is. A left brain dominant is task oriented and well organized, using direct and deliberate speech. He or she plans with practicality and purpose. A right brain is diffuse and emotional, using sensory, expansive and impulsive speech. The right brain dominant looks at things holistically and solicits consensus in planning. Much can be discovered by a person's communication pattern. On the one extreme, the left brain dominant is an articulate, formal speaker, who approaches a subject logically. His or her opinions are strong and
forcefully presented, with little gesture. As a listener, the left brain dominant constantly evaluates the speaker, editing the information and preparing a response, although he or she ignores body language. The predominant right brain speaks in more of a disorganized, diffused and wandering manner. He or she speaks best in emotional, colorful terms. As a listener, the right brain dominant doesn't like to hear messages that upset his or her feelings and biases (Donovan & Wonder, 1984). The scenarios just given point out the extreme cases of left or right brain dominance. Most people fall somewhere in between, and everyone moves back and forth, communicating from the dominant side to communicating from the nondominant side.

Many authorities would agree that by the very nature of their work, programmers are "left brain" (Tognazzini, 1986). Chapter II dealt with the studies and tests showing that tendency. When the Xerox Star was being created, the theory of hemisphericity was addressed and the interface which evolved, and later became the basis for the Apple MacIntosh, was based on the right brain orientation to the world (Tognazzini, 1986). Because most programmers are left brain in orientation, all interfaces up to that point were created to also be left brain oriented.

To identify a person's thinking styles, the programmer can observe the actions of the user in the work setting. In their course on managing the human aspects of technological change, O. D. Resources (1988) suggests the following clues to help the programmer identify the user's thinking pattern. The thinker is fairly
business-like on a telephone, speaking in an ordered, measured manner. He or she is often more effective in writing than speaking and is likely to communicate with logical letters and memos. His or her dress and office is likely to be conservative and unassuming. The "feeler" is more informal and humorous on the telephone, but prefers face to face communication over all other types. His or her dress is usually according to his or her mood rather than to suit others expectations. Feelers tend to personalize their offices, with an appearance of being messy. The "intuitive" is generally wordy and impersonal on the telephone. His or her letters and memos are usually idea and technically oriented. Offices tend to reflect the "intuitive's" imagination with futuristic or thinktank type furnishings. "Sensors" are more abrupt on the phone and need to control the conversation. Their written communication tends to be brief and action oriented. Their dress more often informal, simple and functional. (O. D. Resources, 1988)

If the programmer is aware of how he thinks and how he or she is motivated in contrast to the rest of the population, he or she can be more tolerant of differing orientations, and can strive to work on the areas which may cause problems. For instance, the programmer's left brain orientation to linear, sequential processing often gets in the way of his or her seeing the entire system picture. The programmer's low social need prevents him or her from communicating as often as he or she should in the system development process with the end user of that system. "Computer technicians too often lack practical understanding about the organization's objectives and
practices. They attempt to apply their technical knowledge in an assumed or idealistic world" (Lazarus & Tomeski, 1975, p. 10).

If the programmer understands how the user fits in comparison to his or her own style and way of thinking, he or she can better bridge the gap and fit the system communication to the user's mode of understanding. Programmers have to accept the fact that they are service providers and, in that capacity, have to understand the people they serve. It is not the responsibility of the user to understand the programmer. If you go to an electronics store to purchase a radio, you expect the salesman to explain the various models and features to you in a way you can understand, otherwise, you will walk out without a purchase. This is also true of the user. He or she expects the programmer to understand his or her needs and and fit the system to those needs.

Know Your Computer

Besides understanding his own orientation to the world, the computer professional needs to understand the orientation of his machine. He or she needs to accept the fact that the computer can't do everything. Weizenbaum (1976) asserts that "there are certain questions that can be asked for which it can be proved that no answers can be produced by any effective procedure whatever" (p. 65). He says that even if a programmer understands what the user wants, that understanding may not be able to be formalized.

We may, for example, be able to predict with great confidence what an animal will do under a large variety of circumstances. But our predictive power, great and reliable as it may be, may rest on intuitions that we are simply
unable to adequately explicate. Yet we may be driven to force our ideas into a formal mold anyway. A computer program based on a formal system so derived is certain to misbehave. The trouble then is not merely that the theory it represents contains certain errors in detail, but that that theory is grossly wrong in what it asserts about the matters it concerns. (p.65)

The field of artificial intelligence continues to try to find ways to get a computer to resemble more closely the operation of the human brain, especially the right brain, full of contradictions and incomplete data, and often motivated by hunches and intuition. Our current computer systems are left brain in orientation. VonNeumann (1959) states that

An efficiently organized large natural automation (like the human nervous system) will tend to pick up as many logical (or informational) items as possible simultaneously, while an efficiently organized large artificial automation (like a large modern computing machine) will be more likely to do things successively - one thing at a time, or at any rate not so many things at a time. That is, large and efficient natural automata are likely to be highly parallel, while large and efficient artificial automata will tend to be less so, and rather to be serial. (p. 51)

Left brain orientation thinks linear and right brain thinks holistically. VonNeumann is saying that natural automata (i.e. humans) tend to be holistic, not linear--the opposite of today's computers.

Waldrop (1987) and Naur (1985) agree that present computer systems do not operate wholly as humans do. Imposing strict rules of method may contribute to flaws in the system. Software development involves the programmer's dealing with complicated patterns of interconnected restrictions and concerns and deriving new, relevant conclusions from them, something current computer systems are not able to do well. Programmers need to be aware of the limitations of
the current machines they deal with, be honest with the user and themselves when a task cannot be handled as efficiently with the computer as it can with the current method of completing it, and inform users of the limitations of the system being developed for them.

Current computer systems can only mimic the functions of the left brain; they cannot yet mirror the functions of the right brain. Therefore, it might be said that a computer is only "half a man". Weizenbaum (1976) summarizes the problem well:

I shall argue that an organism is defined, in large part, by the problem it faces. Man faces problems no machine could possibly be made to face. Man is not a machine. I shall argue that, although man most certainly processes information, he does not necessarily process it the way computers do. Computers and men are not species of the same genus. (p. 203)

Know the Business and Its Needs

Programmers write program, but often they forget that the user of the system they develop doesn't want a program. The user wants a solution to his or her problem.

Fifty-five offices in 26 different organizations were included in a study of computer-mediated office work by Bikson and Gutek (1983). It was discovered that four summary dimensions underlie a user's satisfaction with a system: functionality, equipment performance, system interaction and office environment. In conclusion, they report that the most critical problems in implementing information systems are not inherently technological ones. Instead, they involve characteristics of the organization and
its implementation effort: how it structures work, how it approaches change, and how adequately it responds to employee needs in designing the user-computer interface.

Many authors (Arthur, 1983; Naur, 1985) state that systems have to be built to help people in their jobs, reducing costs or improving productivity and quality. Systems have to be developed with the purpose of supporting human beings in their activity.

In his classic work, The Computer and the Brain, John VonNeuman (1959) spoke of producing efficient code:

A code, which according to Turing's schema is supposed to make one machine behave as if it were another specific machine must do the following things. It must contain, in terms that the machine will understand and purposively obey, instructions that will cause the machine to examine every order it gets and determine whether this order has the structure appropriate to an order of the second machine. It must then contain, in terms of the order system of the first machine, sufficient orders to make the machine cause the actions to be taken that the second machine would have taken under the influence of the order in questions. (pp. 72-73)

Since people are also a type of machine, we can borrow VonNeumann's words and apply them to application generation in general: the application must contain, in terms that the computer will understand and purposively obey, instructions that will cause the computer to examine every order it gets and determine whether this order has the structure appropriate to an order of the user. It must then contain, in terms of the order system of the computer, sufficient orders to make the application cause the actions to be taken that the user would have taken under the influence of the order in question.

How do we know if a system is worth developing? Boone and Meyer (1987) assert that the primary determinant should be cost and benefit
to the user. Just as people don't change unless they want to, users will work to make a system succeed when doing so helps them succeed. The successful programmer seeks applications that are good both for the organization and for the people who must live with the system. We must let the user determine the applications that need to be developed and the value of that application to the business. The analyst, through a series of questions, can help the user to define improvements. But, once the improvement is clearly defined, the user and programmer must determine how it fits in the scope of the organization's profits or missions.

Considering the communication gap that exists between users and programmers, the only effective way that a programmer can understand a user's business need is to walk in his shoes. One acquaintance that works for an electronic company said that in one project at his company, they happened on that idea by mistake. They were asked to computerize the testing phase of certain electronic aviation equipment. Management, at the time, was knee deep in another project and, therefore, sent the programmers for the system to the site. For a couple days, those programmers sat next to the individuals conducting the testing, pushing the buttons, operating the equipment. They returned home with a good understanding of the current method of performing the task they were asked to computerize. The project was a complete success.

Many times, there are business needs or productivity improvements that could be met with computerization but the users do not recognize the tasks as candidates for that. Programmers or
analysts who lean toward right brain dominance or who can easily switch between the two should be chosen to act as what the author calls "end user liaisons". Each of those people would be assigned to an individual company unit. The liaison would reside with the unit and be responsible for becoming familiar with that unit's business operation and its personnel. The liaison would meet regularly with the unit's personnel to discuss and help identify computer needs, both hardware and systems, and, after developing plans, make the necessary arrangements and contacts to insure that those systems, identified as meeting a business need for that unit, were developed or obtained. The liaison would meet regularly with other end user liaisons to compare and share ideas generated. He or she would also be responsible for insuring that communication, participation and feedback were maintained for all systems being developed for that unit.

A major problem that was consistently identified in the survey conducted as part of this paper was the fact that the programmer did not understand what the user did, what the user really needed, or how the user would be able use the systems in his or her work. The ideas presented here would help solve those communication problems. First, establish end user liaisons to help identify systems needed to meet the business need. Second, put the programmers with the users for awhile before system development to give them a chance to experience the environment in which the user lives.
Close the Gap in the Design Phase

Tap Your Right Brain: Intuition

No individual is totally controlled by one hemisphere of the brain or the other. Left brain dominant programmers have right brain characteristics, which are not as pronounced as a person whose right brain is dominant, and shift back and forth between the two hemispheres. Therefore the programmer should tap into his or her right brain characteristics as much as possible when communicating with end users. He or she should try to think in terms of the whole system picture during the design phase, leaving the detail till later. When designing a system interface, they should remember that the right brain is nonverbal. The Xerox Star and very popular Macintosh Computer interfaces were designed for right brain dominant individuals.

The right brain is the source of intuition and creativity. The right brain is where the sudden insights or solutions come from. The left brain can then recast those solutions into a logical path for completion. The programmer needs to trust his or her intuition, their "gut" feelings. Naur (1985) goes so far as to say the following:

Immediate human apprehension, or intuition, is the basis on which all activities involving software development must be built....Part of our intuitive ability is to make sense of the impressions we receive through our senses....In our intuitive apprehensions we are perfectly capable of dealing with the world and its quality without basis in criteria or scales of value....Any kinds of well-recognizable categories, counts, and measures, in their application to the world depend entirely of our intuitive understanding. Any description of the world in terms of strict categories
and scale values is no better than our intuition will make it....Our intuitive knowledge is a connected whole of innumerable items of insight and knowhow, continually adjusting to the changing situation....The compatibility of descriptions used in developing a piece of software and the matters of the world that are supposed to be modeled by them remains a matter for human intuition in any case. (pp. 60-77)

Donovan and Wonder (1984) suggest ways to tap right brain power. They suggest trying to see whole situations and how each person and element is related. Programmers should try to be more aware of the colors, space, aromas, sounds and emotions around them and make eye contact when communicating with others. Other ideas include shifting the phone to the left ear (controlled by the right brain), drawing, singing, joking, and relaxing.

Why should programmers even bother to tap their right brain? If they learn how to move between the two hemispheres, they can use their right brain to communicate effectively and their left to develop the computer systems that result from that communication. As a side benefit, they will become more valuable employees and that should stroke their high growth need. Agor (1984) points out that most successful managers use right brain intuition to make decisions and claims that "any individuals aspiring to top levels of management will need to possess a greater degree of right brain skills, including intuition, than ever before" (p. 4).

Don't Overrun the User

Programmers must try to meet the user at his or her level of comprehension. The use of technical computer jargon should be avoided when communicating with the user. They need to remember,
too, that communication is not just telling, it is listening. Weinberg (1982) tells the following story to illustrate this point:

Mack, like so many computer programmers and analysts, had a very high IQ and knew many facts about computers and the rest of the world....Having a high IQ is like a CPU having a terrific computing speed. It's a great asset in problem solving—as long as the problem doesn't involve a lot of input or output. But when it's necessary to communicate with other people in order to convert the idea of a solution into an actual solution, that high internal speed often causes overrunning....If Mack were to encounter a computer system that was unbalanced in the direction of CPU power, he would instantly know precisely what to do to remedy the situation. He certainly wouldn't spend time trying to make the programs run even faster, but that's what he did when he was the system. What Mack needed, and what so many bright young computer people need, is to build up their input and output capabilities until they are in better balance with their environment. When writing or talking, they can reduce the quantity of output and consume some of their excess computing power on improving the quality....One way to trade computing power for improved output is to use a lot of that power processing the input. We usually call that listening. (pp. 125-126)

The programmer needs to be able to bend a little and realize that the system is built to accommodate a business need and the user knows best what that need is. Gruenberger (1971) points out that a manager can expect a more mission oriented and professional attitude from his programmers than he can from others, but, "by the very fact that they are highly skilled and talented and require less personal direction, they are frequently strong minded individuals. They have the courage of their convictions and are not easily dissuaded once an opinion is formed" (p. 85).

The Japanese have a word "rikutsupoi," which translates as "too logical." It is used to describe people who are very bright but excessively logical and pushy (Athos & Pascale, 1981). Programmers need to be very careful not to be "rikutsupoi" and turn people off

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
before they even begin. As Athos and Pascale (1981) ask, "How often are brute integrity and explicit communication worth the price of the listener's goodwill, openmindedness, and receptivity to change" (p.159). Programmers must realize that, if they fit the typical composite, their high growth need makes it possible to run over the user as their enthusiasm and drive take over a project. They have to check themselves and slow down, or risk putting together a marvelous piece of code that accomplishes nothing.

**Fit the User's Concept of Reality**

Programmers must face the fact that user's concept of reality is probably not like theirs. To begin, the user's reality is full of contradictions and incomplete information. Users do "not know exactly what they want and are unable to tell us what they do know" (Clements & Parras, 1985, p. 82). Other authors (Birnbaum, 1985; Leventhal, 1987; Olson & Schott, 1988) agree that the user's world is full of ambiguities and that his or her mental abstraction of the process taking place in the machine may differ from that of the programmer. Many systems are developed with a bias toward the systems professional's view of the world, not toward the view of the actual users.

Programmers often feel that a user cannot understand the system, that the user is not smart enough or technically competent enough to grasp it. What is missed is that there are two parts to a computer system. The first is the technical part, handled by the programmer, but the second, and of equal importance, is that of the user and his
view of the business need being served. Weizenbaum (1976) points out that the computer science professional tends to be arrogant because his or her knowledge is harder than that of his or her humanist colleagues. "But the hardness of the knowledge available to him is of no advantage at all. His knowledge is merely less ambiguous and therefore, like his computer languages, less expressive of reality" (p. 20). As Brooks (1986) states:

We cannot stand back and gripe that the user didn't know what he wanted. We must take it as a given that the user does not and cannot know what he wants about artifacts as complex as those we now build. The mind of man cannot imagine all the ramifications of such artifacts. There must be an iterative cycle in which the professional works with the user to define the requirements; demonstrates their consequences in human factors, cost and performance; then in a prototyping phase iterates with the user to develop a product that is, in fact, satisfactory. (p. 8)

Programmers that develop scientific applications can usually communicate well with the users, who also tend to be of left brain orientation. But those applications developed for the remaining population, and that is usually the majority, are for people generally of right brain orientation. Therefore their "business" systems don't always fit well into the programmer's view of the world. Gruenberger (1971) asserts that business computer problems are usually ill defined and the solutions to the problems are usually not defined at all, unlike the nice neat checks one has in solving equations. He further points out that business problems deal with data that has a large level of error, and that there are few standard algorithms for solutions of those type of problems.

It is crucial to successful system development that the program is written to solve a real world problem. As Olson and Schott (1988)
point out, "It is important for systems developers to understand the complete context in which the system's users function. Developers need to describe not only who the users are and what tasks they perform, but also where and how often the users perform those tasks, how they learn to do them, and what they do when things go wrong" (p. 70).

How can a programmer be sure that the system fits the user's reality? There is an old Indian saying, "Lord, grant that I might not criticize my neighbor until I have walked a mile in his moccasins." The author proposes that the programmer walk in the user's shoes, follow the user around in his environment for a day or a week. Because of the discrepancy in the way the programmer and users perceive the world, perhaps that is the only real way to acquire a good understanding the user's world.

**Evolving Systems**

Systems cannot be totally defined in the traditional design phase, they must evolve. That is contradictory to the current, traditional system development theories, which state that the users should first sign off on all specifications before coding is begun. But several authors (Demb, 1979; Floyd, 1985; Gunton, 1988; Martin, 1984; Naur, 1985) assert that may not be the best method. Dealing with reality as it evolves is a right brain feature. Reality is holistic not linear. Analysts cannot expect the user to identify all the specifications and sign off on those before the coding begins. The finished product would probably be useless. There are whole
classes of results and reactions to the program which the would be unprepared to deal with early in development. The processes of communication and creative corporation cannot, and should not be entirely formalized. One research team (Demb, 1979) found that there was no correlation between the existence of a set of measurable objectives at the start and the success of the system. In situations where there were no specific early goals, the specifications were developed as the project unfolded.

There has been a great deal of concern in America regarding the gains that Japanese corporations have made in the past few years. In the field of data processing, there also exist major differences. The 1988 Price Waterhouse IS Trend Survey of information system executives (cited in Grindley, 1988) showed a dramatic difference between what information system executives in the United States and those in Japan perceived to be the top three user issues in system development. Executives in the US cited meeting project deadlines as the most critical issue, followed by predicting information technology and integrating communications with data processing. On the other hand, Japanese executives did not see meeting project deadlines as a problem at all. Japan's major issues were program maintenance, go alone users and recruitment. When discussing this point with an associate who had been to Japan on engineering business, he stated that he felt the reason for this was because the Japanese always meet deadlines. Bell (1985) gives us a clue as to the reason for this discrepancy. He states that the Japanese approach is to work with the "natural pattern, starting with the
criterion of usefulness, rather than devising arbitrary, revolutionary, and perhaps useless architectures" (p. 25). That natural pattern better fits the user's concept of reality, and that natural pattern makes use of both sides of the brain.

On a whole, the Japanese business communicates differently than American business people do. Japanese communicate to the right hemisphere of the brain far more than Americans do. Several authors (Athos & Pascale, 1981; Donovan & Wonder, 1984; Weizenbaum, 1976) have observed various Japanese traits that point this out. Americans admire directness, whereas the Japanese tend to be more vague. They use many forms of nonverbal communication in contrast to the Americans who predominantly use verbal communication within a context of physical separateness. They are more means oriented, or process oriented, whereas Americans tend to focus more on the bottom line, on the ends. Americans are more Aristotelian. They feel if it is not white, by deduction it has to be black. The Japanese live comfortably with gray. The Japanese employ open discussions with generalities that leave room for movement and compromise. They have nineteen different ways of saying no. Their management techniques make far greater use of right-brain holistic talents than do most managers in the world. The Japanese are more group oriented and interdependent in their relations with others, whereas Americans are more individual oriented and independent. Perhaps, we can learn from the Japanese style of communication how to deal more effectively with users.
User Participation

The user must participate, as much as possible, in the design phase. User participation is one of the best ways to close the communication gap between programmers and users. As Arthur (1983) points out:

Active involvement of the end user serves both the user and DP. The programming staff benefits from gaining direct understanding of what is needed, so the system is built correctly the first time, eliminating the need for extensive system enhancement after conversion. Nothing is more frustrating than discovering the project you have devoted your time to has failed to meet the user's needs. (p. 93)

In a study conducted to measure associations between a worker's participation in a system's design and that worker's resulting attitudes, equipment usage and perception of system effectiveness (Herold et al., 1987) it was shown that participation was consistently associated with positive attitudes towards the finished product and consistently related to reported performance effectiveness both 6 and 18 months after the introduction of new equipment. Another study (Herold et al., 1988) showed that user participation had an 84% success rate even though it was only used in 17% of the cases studied.

User participation also cuts down on system errors. Studies at ITT Corporation., IBM, TRW, Inc. and Mitre Corporation (Rush, 1985) indicate that error removal constitutes up to 40% of the cost of the system - and that between 45% and 65% of these errors are made in the system design. Other authors (Arthur, 1983; Slusky, 1987) point out that user participation can quickly uncover deficient or erroneous design features. This is especially important because errors made in
design features. This is especially important because errors made in system design are often discovered in the testing phase and, therefore, very expensive to correct.

In order for the project to be successful, the user must feel that it is his or her system, not the programmers. If the user participates fully in the design and development of a system, he or she will experience a feeling of ownership. Some authors (Boone & Meyer, 1987) feel that a true people centered approach to implementation gives users control over their systems. Users should retain control of the project management process, problem selection, system design, justification, and measurement of benefits. The user is in the best position to identify the systems which will serve business needs and the culture in which those systems must reside. The users must also put a lot of effort into making a system work and is more apt to be supportive if they see the system as theirs.

The author was involved in a project tracking system planned to serve an entire division. As the scope of the project was quite large, several years were spent in planning and coding. Meanwhile, one employee felt the need for an interim system to track his activity. A simple screen application in an easy to use language was developed in a half a day. Changes were made and reports generated as needed. The user relied on that system and the reports it generated for over a year. When the major project planning project was finally put into place, the employee was very reluctant to give up "his" system for one that he did not perceive belonged to him. The user must feel to be in control of the system he uses.
How can the programmer best include the user in the system design phase? Because of the communication problems that exist between programmers and users, perhaps due in part to the differences in their thinking patterns, the communication path must follow a defined structure. The programmer must be forced to communicate on a regular basis and in an effective manner, as their motivational factors and thinking styles do not readily lend themselves to spontaneous communication techniques.

For example, Donaldson (1978) suggests that every project start with a survey stage, to study the nature of the problem to be solved. Starting at the top of the organizational unit under study, get the physical layout of the staff and equipment. Next, find out what each sub-unit does by creating an activity list, covering the functions performed by that unit, the approximate proportion of time on each and identify the functions which are difficult or problem areas. Eliminate work which is not time consuming, difficult or a problem area. Finally, get an understanding of the flow of paperwork between the sub-units, and the files which are maintained. Once the survey stage is complete, then the analysts should precede with fact gathering and pinpointing the business need. Too often we forget that a computer system is not an end in itself but exists only to support a business need.

Other techniques for formal design specification are addressed by Rush (1985). He points out four major techniques that have been developed since the late 1970s to address the information gathering problem. They are Joint Application Design (JAD), Consensus, Wisdom,
and The Method. Geared toward the front end of the system design life cycle, they help users define an application from its first conception through the complete design.

Unfortunately, the communication gap is usually too large to bridge effectively with just a structured method of verbal communication. What is needed is a computer system that gathers the business information needed from the right brain users and the technical information needed from the left brain programmers. Martin (1984) suggests just such a system that is a language used as formal input to a computerized design tool, with which requirements, specifications, and details can be expressed. The requirements statements are decomposed into greater detail and become the specifications. The specifications are decomposed into greater detail until sufficient detail is reached that code can be generated automatically. When changes are made at a lower level, these are automatically reflected upward.

Martin's proposal is a good way to solve the communication problem at the design level of a system. Due to the differences in communication patterns between programmers and many users, all the talking in the world will still result in problems, because each group interprets what is being said in terms of their own frame of reference. Weizenbaum (1976) puts in very well, as follows:

A person's belief structure is a product of his entire life experience. All people have some common formative experiences, e.g. they were born of mothers. There is consequently some basis of understanding between any two humans simply because they are human. But even humans living in the same culture will have difficulty in understanding one another whereas their respective lives differ radically. Since, in the last analysis, each of our
lives is unique, there is a limit to what we can bring another person to understand. There is an ultimate privacy about each of us that absolutely precludes full communication of any of our ideas to the universe outside ourselves and which isolates each one of us from every other neotic object in the world.

There can be no total understanding and no absolutely reliable test of understanding.

To know with certainty that a person understood what has been said to him is to perceive his entire belief structure and that is equivalent to sharing his entire life experience. (pp. 192-193)

If the language which Martin (1984) suggests has a right brain function, the would will be able to specify his initial specifications in that language. The left brain function would then gather the needed technical information from the left brain programmer. The system would then perform the "translation" process into the level of specification needed for code generation. It would become a right brain, left brain translator. When the science of computers is finally able, through artificial intelligence and parallel computers, to simulate the holistic view of the right brain, the translators will no longer be necessary. In addition, because the specifications will constantly change, the idea that the code be automatically generated from the specifications makes it easy to modify the system when the user changes specification.

Feedback

Constant feedback to the users is important. The programmer/analyst should always reflect back to the user what he or she thought they heard the user say. Because programmers have a low social need, they don't perceive a need for feedback. They must
always be aware that the users are different and probably have a high need for that feedback.

One excellent form of feedback is to tell the users what the system can do. Sell it. Sell it's capabilities. Help the user see how it adds value to his or her work situation. Arthur (1983) says, "These people have little chance to investigate the possibilities on their own, so inform them. Fewer cries of 'But we don't know what we want!' will be heard" (p. 92).

Cougar and Zawicki (1980) believe that because of the low social need of programmers, there is a need for more formalized feedback procedures. As covered in the section on user participation, it does not come naturally to most programmers to include the user in the design process or communicate with them on a regular basis. A formal approach, such as those suggested in the previous section, would serve best to tackle the problem. Mathews and Vogt (1987) suggest one method to formalize the process of feedback. The user and programmer should first set goals in order to establish a centralized focus on the problem. That would then be followed by extensive and frequent feedback throughout the development stages to clarify the problem as each person interprets it, and to make sure the goals that are set are, in fact, achieved. During that feedback process, the programmer must take many steps in order to clarify the problem for the user, so the user can better grasp and adjust to the new information. That process will help prevent the user from becoming resentful towards, or losing interest in, working with the programmer. Another excellent tool for feedback is prototyping the
system and will be covered later in this paper.

Close the Gap in the Coding Phase

Time and again one finds stories of a system being cut over after years of development effort and end users saying that it is not what they want, or trying it for awhile and then giving up. Frequently, after using a system, laboriously created, for a few weeks the users say they want something different.

A common reaction to this unfortunate situation is to say that the requirements were not specified sufficiently thoroughly. So more elaborate procedures have been devised for requirements specification, sometimes resulting in voluminous documentation. But still the system has been unsatisfactory.

The fact is that many of the most important potential users of DP do not know what they want until they experience using the system. When they first experience it, many changes are needed to make them comfortable with it and to meet their basic requirements. Once comfortable with it, their imaginations go to work and they think of all manner of different functions and variations on the theme that would be useful to them. And they want those changes immediately. (Martin, 1984, p. 41)

The system must evolve, and it must evolve in pace with the user's ability to inseminate it. The production phase should start long before the design phase is completed. All the ideas for improving communication in the design phase can be used in the production phase, for they should flow between each other. User participation and feedback are just as crucial in the coding phase as in the design phase. If the users are given a small part of the system to work with in the beginning, they can better articulate changes needed and future needs than they can if all the design is still on paper. End users tend to be right brain and nonverbal. If they can touch and use a system, they will better understand their own needs. Guton (1988) proposes that the system evolve at the same
traditional approach of system specifications closes off many of the
design options before end users have understood the implications. As
Martin (1984) states, "The act of providing what an end user says he
needs changes his perception of those needs. The mere act of
implementing a user-driven system changes the requirements for that
system. The solution to a problem changes the problem" (p. 43).

The method suggested above makes programming difficult because
change is considered to be part of the process. Therefore, we need
to develop new methods of system design, built on the concept that
change is inevitable and that it must be possible to precede with the
development with limited information. One proposed system was
discussed in the previous section.

Prototype

Programmers must develop the system in pieces, starting with a
basic core. In a seminar, Martin (1984) suggested that any piece
that takes more than four weeks from the start to when it is placed
in the user's hands should be broke down further. Let the user use
that piece, test it, work with it. Then that piece can be modified
to meet the user's needs and the next piece can be developed with the
new knowledge gained from the development of the first. Each
successive piece developed can benefit from those that preceded it.

The author was part of a project to develop a system to track
financial data. Because of the time constraints under which we
worked, it was necessary to first build a module to obtain unique
computer generated numbers. The use of that module by the users
taught us many things about the system that were then incorporated into the next section. Had it not been for the time problem, all modules would have probably been built in unison and all would have had similar problems that would have had to been changed.

Prototyping is a very successful tool that can be used by users to produce the system in pieces. Put very well by Nolan (1986): "Prototyping is results oriented rather than project oriented. Emphasis is placed not so much on the method employed as it is on the overwhelming need to bring the computer to the people who will be using it in a form they can recognize and then make the system do what is necessary to satisfy them" (p. 93).

A prototype does not have to be a program. It can take on a variety of forms. A straightforward paper-and-pencil simulation of a system can serve as a prototype, or a computerized slide show demonstrating the major features of the user interface. As Olson and Schott (1988) point out, "It does not even have to closely resemble or faithfully simulate the final product for usability testing to be helpful" (p. 74). Another prototyping technique that could be used is that of creating a storyboard, where system designers take potential users on a tour of the interface and work flow.

Let Joe Do It

One way to implement a system is to let the users create and/or modify their own applications or some parts of them. Or as a modification to that idea, Martin (1984) suggests having a system analyst create the application working at the terminal with the end
user much of the time. "There are now many case histories of end users having created systems with multiple applications which are by any standards spectacular. The moral of such case histories is that we have grossly underestimated the end users. Some end users are amazingly bright people. They need only the right tools and encouragement" (p. 50).

Some possible tools are the vast array of microcomputer software, spreadsheets and database packages. Those systems or easy to use mainframe languages could be used for users to establish their own system prototypes. If the user can create his or her own simple applications or prototypes which are some small piece of larger more complex systems, the programmers could be freer to devote their time to the development of those more complex systems, and much of the current system backlog could be eliminated.

Some expert systems are designed in such a way that the user can directly input his or her knowledge on a subject and have the computer generate the resulting application. The realm of artificial intelligence applications often addresses direct user input and system development.

Useful Interfaces

The way the user interfaces with the system is crucial. The interface is the user's view of the system. Since most users are right brain and nonverbal, that view is extremely important to system success.

At one conference, the speaker told of a system developed to
automate the arrival of goods at the loading dock of one midwestern company. The environment was studied, meetings were held, specifications were drawn up, then the programmers developed handheld computers on which the receivers could log the day's deliveries as they arrived at the dock, rather than using the shipping forms provided by the deliverer. After the system was implemented, it was discovered that, after a couple months of use, it was abandoned by the receivers. A visit to the receiving dock pointed out the problem: during the winter months, the receivers wore heavy gloves and could no longer push the keys. The user interface was useless.

The programmer must design an interface that is also compatible with the user's concept of reality as the system itself: holistic and nonverbal. As Arthur (1983) points out, "we perform wonderfully, creating the black box, but we fail to engineer the human-machine interface, which forces employees to perform in an unnatural way, reducing their productivity" (p. 91).

Close the Gap in the Test Phase

A system is successful if it meets the user's business need, if it solves a business problem, if it makes the user more productive at his work (Brackett, 1987; McConnell, 1987; Schneiderman, 1987). Users must be able to get productive work done within an hour when using a new system or they won't persevere at it. They must be able to do repetitive operations conveniently and consistently, and be able to customize the system to their way of working.

Schneiderman (1987) suggests some measurable human factors that
the programmer might address. The programmer could determine how long it takes for the users to learn how to use the commands, how long it takes to carry out a benchmark set of tasks, and how many and what kind of errors are made in carrying out the benchmark set of tasks. And, of course, the programmer should inquire as to how pleased the user is with the system. As McConnell (1987) says:

It is not the technical system which ultimately determines MIS success. Success lies within another system—the human system....Success occurs when the user is able to convert the available data from the technical system into useful information which supports and aids the human system. If the technical systems are to survive, they must be directed toward serving the human system....The human system can survive without the technical system but the reverse is not true. In other words, the user can work and function without a computer, but the computer cannot work and function without the user. (p. 5)

Therefore, the true test of a system is whether it is useful to the users. One way to insure that is to have the testing done by the actual users, and not just the users that helped in the development process. There is a good reason for this. System developers and the user representatives who work closely with them lose objectivity and cannot see the system as an ordinary user would.

Prizig (1974) in his book, Zen and the Art of Motorcycle Maintenance sums it up well: "The material objects...can't be right or wrong. They don't have any ethical codes to follow except those people give them. The test of the machine is the satisfaction it gives you. There isn't any other test. If the machine produces tranquility, it's right. If it disturbs you, it's wrong until either the machine or your mind change (p. 40)."
Making the Right and Left Brains Fit

This paper has attempted to suggest ways in which we can bridge the communication gap between programmers and nonprogrammers. Sometimes, though, the differences in thinking patterns and motivational factors are too different to provide a good blend. There are two methods that could be employed to narrow the gap.

First, match jobs with individuals. Programmers, like all people, have a blend of right brain/left brain tendencies. Those programmer that show the greatest tendency towards right brain thinking style should be used as analysts and those with a strong left brain orientation, as developers. As Cougar and Zawicki (1980) maintain, "Behavioral science researchers have proved in other industries that productivity can be increased by a better matching of jobs with individuals" (p. 31).

Donaldson (1978) even goes so far as to suggest that a single person does not see the project from beginning to end. He asserts that the best business analyst should be in charge at the start of the system development cycle and the best technical manager at the end. Other authors suggest that we also change programmer career paths. Cougar and Zawicki (1980) were not surprised that programmers showed a low social need in the Hackman/Oldman studies but they were surprised that analysts had an equally low SNS score. They see the problem as being the career paths in most departments that move programmers into analysis and often into management. A low social need causes little problem at the programmer level but may be a
primary reason for the difficulty in maintaining satisfactory relationships between the programmer and the end user.

The Index Group believes that not all system development is the same (Crescenzi & Gugliotti, 1987). They point out four profiles from that with low business change and low technological challenge to those with high business change and high technological challenge. Staffing for the project should be chosen to best fit the profile in consideration. Low business change and technological challenge projects should be staffed with the technical type or left brain oriented individuals. On the other hand, high business change and technological challenge requires computer personnel who must be able to see the whole picture in their mind and translate that vision into reality. In other words, those analysts that work on the high impact applications must be able to tap the resources of their right brain, be able to see holistically.

A second method to bridge the gap is to train programmers in communication techniques. A glaring problem is the fact that, although most computer professionals claim that good communication is critical to the success of a system, very few colleges include a class in communications as a required course in their computer science curriculum and very few industries provide that type of training for their current employees. Several authors (Arthur, 1983; Blank, 1988; Cougar & Zawacki, 1980; Levanthal, 1987) agree that to lessen the effects of the low social need of computer professionals, formal training should be offered in behavioral concepts and communication techniques. As Blank (1988) states:
Most computer science departments do a fine job of producing people well prepared for careers in front of the terminal. There is little or no emphasis on interpersonal skills....These graduates have been trained to perform as islands. It has been estimated that approximately 15% of the population have good interpersonal skills without any formal training. They have a knack for getting along with people, and possess the needed communication skills to work well with others. At the other extreme are the approximately 10% who don't develop these skills regardless of the training level. These people would do well to choose a career accordingly. The remaining 75% could develop necessary skill levels with proper training. (p. 10)

What should be included in such a course? The training should emphasize both verbal and non-verbal communication, including the skills of listening, speaking, language context, analysis and body language recognition. The students should be taught how to analyze the brain dominance and thinking styles of themselves and any individuals with which they must communicate.
CHAPTER V

CONCLUSION

Good communication between programmers and users in system development is essential to the success of a computer project. Lack of it can result in well coded systems that are never successfully implemented.

This paper established, by citing various authors, that a serious communication problem exists in most system development. It then investigated various theories which possibly contribute to those communication problems. In order to explore those theories, an exploratory survey was conducted on a small group of programmers and system users in a midwestern corporation, assessing their perception of the communication process in the system development cycle. Based on available literature, the survey results and the author's experience in the field of computer science, several suggestions were made as to how the communication gap between programmers and end users might be narrowed through all phases of system development.

The theories of hemispheric preference of the brain and thinking styles suggest that most programmers are different in the way they solve problems, think and communicate from a large portion of the user community for whom they design systems. Those differences lead to the problem of programmers and users talking to each other but not
understanding each other. Motivational factors only serve to complicate matters further. Due to the low social need of most programmers, there is little impetus to communicate. The programmer's high growth need often leads to over-running the user in computer jargon and or in system features which the user doesn't need or finds confusing.

What results is two well-meaning groups of people with different views of what is needed and wanted. Because it is difficult for either group to convey its view to the other, they end up with a system that is useless or needs revision to be of value.

The literature suggests many ways in which the communication gap between programmers and users can be narrowed. The author perceives the ideal situation would be one where, a programmer that showed right brain preference or the ability to switch easily between hemispheres would be assigned to a operational business unit of an organization. That "end user liaison" would become thoroughly familiar with the group's business and physically reside with the staff. He/she would be responsible for helping the group identify its system needs, based on business needs, integrate that into the overall information system plan, and coordinate any necessary system development, insuring that communication was maintained between users and programmers. He/she would determine whether the proposed systems met business needs and was within the limitations of current computer capabilities.

Before development of a system is started, the programmer or analyst, assigned to the task of developing the system, should spend
a week or more side by side with a user, experiencing his/her job. A formal method of establishing specifications for the system would be used, preferably a computer program that could glean information from the right brain of the user and from the left brain of the programmer and then generate working code from the information gathered. Each system must evolve gradually, starting with a small core module and building from there. Constant communication and feedback should be maintained with the users throughout the entire development and they should participate as much as possible in all phases, from planning through testing.

A programmer should be taught the principles of good communication and how to identify and deal with different thinking styles. He/she need to know his/her own hemispheric brain preference and how that influences the way he/she deals with others and the world.

This thesis provides possible reasons for the communication problems that exist between programmers and end users in system development and offers suggestions as to how those communication problems might be addressed. As this problem has not been adequately addressed by computer professionals, further work needs to be done. Formal methods of system development need to be established which dictate when communication should take place. A computer system needs to be developed that allows users and programmers to jointly input their views of the finished system and then blends the two views into one. Work must continue on good prototyping tools that allow the user and programmer to sit side by side and develop systems.
truly useful to the user. A programmer communication class curriculum must be developed in universities.

More formal testing and studies of computer professionals need to be done on large populations to better determine the impact of hemispheric preference, thinking styles, and motivational factors on system success. The survey, addressed in this thesis, attempted that on a small scale. The survey instrument used needs to be improved, using more formal testing instruments to measure hemispheric preference and thinking styles. The information gathered from such a survey could be used in future system development to place programmers in the best position within the system development cycle to insure optimal communication and ultimate system success.
Appendix A

Survey Questionnaire
Evaluation of Programmer and End User Communication
in System Development

System Name: ____________________________________________

Evaluator's Name: _______________________________________

On a scale of 1 to 10 (1 = very poor; 10 = excellent), how would your rate the usefulness of the system when it was first developed? ________

On a scale of 1 to 10 (1 = none; 10 = a lot), how would you rate the amount of communication between you and the programmers when the system was being developed? ________

How could the communication process between programmers and end users have been improved in the development of this system?

There are many ways of solving problems. Most people use a combination of all the factors listed below, but for this question, please pick one of the following two groups which best describes your way of thinking or solving problems: __________

I. Logical (drawing conclusions based on logic), linear (thinking in terms of linked ideas), digital (using numbers), rational (drawing conclusions based on reason or fact), abstract (taking a small bit of information to represent the whole thing), analytic (figuring things out step by step), & symbolic (using a symbol to stand for something).

II. Holistic (perceiving overall patterns and structures), intuitive (making leaps of insight, often based on incomplete patterns, hunches or feelings), spatial (seeing where things are in relation to others), nonrational (not requiring a basis of reason or facts), analogic (seeing likeness between things), concrete (relating to things as they are at the present moment), synthetic (putting things together to form wholes).
There are also many different ways of approaching a problem. In your estimation, which of the following best describes your style of thinking or way of approaching a problem: __________

I. Reflective: integrates experience with self, seeks meaning and clarity, needs to be personally involved, learns by listening and sharing ideas, favorite question is "why?".

II. Conceptual: forms theories and concepts, seeks facts and continuity, needs to know what the experts think, values sequential nature of thinking and detail, favorite question is "what?".

III. Practical: practices and personalizes, seeks useable information that has utility, needs to know how things work, edits reality, favorite question is "How does this work?".

IV. Creative: seeks hidden possibilities, responds to new and different, is adaptable to change, learns by trial and error, favorite question is "If?".

If you had any communication with a programmer during system development, please answer the following questions:

On a scale of 1 to 10 (1 = very poor; 10 = excellent), how would you rate the quality of communication between you and the programmers when the system was being developed? ________

Name the one programmer with which you had the most contact. _______________
(If you had contact with more than one, just pick one.)

In your estimation, which of the following best describes that person's way of thinking or solving problems: __________

III. Logical (drawing conclusions based on logic), linear (thinking in terms of linked ideas), digital (using numbers), rational (drawing conclusions based on reason or fact), abstract (taking a small bit of information to represent the whole thing), analytic (figuring things out step by step), & symbolic (using a symbol to stand for something).

IV. Holistic (perceiving overall patterns and structures), intuitive (making leaps of insight, often based on incomplete patterns, hunches or feelings), spatial (seeing where things are in relation to others), nonrational (not requiring a basis of reason or facts), analogic (seeing likeness between things), concrete (relating to things as they are at the present moment), synthetic (putting things together to form wholes).
In your estimation, which of the following best describes that person's style of thinking or approach to solving problems: __________

I. Reflective: integrates experience with self, seeks meaning and clarity, needs to be personally involved, learns by listening and sharing ideas, favorite question is "why?".

II. Conceptual: forms theories and concepts, seeks facts and continuity, needs to know what the experts think, values sequential nature of thinking and detail, favorite question is "what?".

III. Practical: practices and personalizes, seeks usable information that has utility, needs to know how things work, edits reality, favorite question is "How does this work?".

IV. Creative: seeks hidden possibilities, responds to new and different, is adaptable to change, learns by trial and error, favorite question is "If?".
Evaluation of Programmer and End User Communication in System Development

System Name: ______________________________________________________

Evaluator's Name: _________________________________________________

On a scale of 1 to 10 (1 = very poor; 10 = excellent), how would you rate the usefulness of the system when it was first developed? ________

On a scale of 1 to 10 (1 = none; 10 = a lot), how would you rate the amount of communication between you and the end users when the system was being developed? ________

How could the communication process between programmers and end users have been improved in the development of this system?

There are many ways of solving problems. Most people use a combination of all the factors listed below, but for this question, please pick one of the following two groups which best describes your way of thinking or solving problems: __________

I. Logical (drawing conclusions based on logic), linear (thinking in terms of linked ideas), digital (using numbers), rational (drawing conclusions based on reason or fact), abstract (taking a small bit of information to represent the whole thing), analytic (figuring things out step by step), & symbolic (using a symbol to stand for something).

II. Holistic (perceiving overall patterns and structures), intuitive (making leaps of insight, often based on incomplete patterns, hunches or feelings), spatial (seeing where things are in relation to others), nonrational (not requiring a basis of reason or facts), analogic (seeing likeness between things), concrete (relating to things as they are at the present moment), synthetic (putting things together to form wholes).
There are also many different ways of approaching a problem. In your estimation, which of the following best describes your style of thinking or way of approaching a problem: 

I. Reflective: integrates experience with self, seeks meaning and clarity, needs to be personally involved, learns by listening and sharing ideas, favorite question is "why?".

II. Conceptual: forms theories and concepts, seeks facts and continuity, needs to know what the experts think, values sequential nature of thinking and detail, favorite question is "what?".

III. Practical: practices and personalizes, seeks useable information that has utility, needs to know how things work, edits reality, favorite question is "How does this work?".

IV. Creative: seeks hidden possibilities, responds to new and different, is adaptable to change, learns by trial and error, favorite question is "If?".

If you had any communication with an end user during system development, please answer the following questions:

On a scale of 1 to 10 (1 = very poor; 10 = excellent), how would you rate the quality of communication between you and the end users when the system was being developed? 

Name the one end user with which you had the most contact. (If you had contact with more than one, just pick one.)

In your estimation, which of the following best describes that person's way of thinking or solving problems: 

III. Logical (drawing conclusions based on logic), linear (thinking in terms of linked ideas), digital (using numbers), rational (drawing conclusions based on reason or fact), abstract (taking a small bit of information to represent the whole thing), analytic (figuring things out step by step), & symbolic (using a symbol to stand for something).

IV. Holistic (perceiving overall patterns and structures), intuitive (making leaps of insight, often based on incomplete patterns, hunches or feelings), spatial (seeing where things are in relation to others), nonrational (not requiring a basis of reason or facts), analogic (seeing likeness between things), concrete (relating to things as they are at the present moment), synthetic (putting things together to form wholes).
In your estimation, which of the following best describes that person's style of thinking or approach to solving problems: 

I. Reflective: integrates experience with self, seeks meaning and clarity, needs to be personally involved, learns by listening and sharing ideas, favorite question is "why?".

II. Conceptual: forms theories and concepts, seeks facts and continuity, needs to know what the experts think, values sequential nature of thinking and detail, favorite question is "what?".

III. Practical: practices and personalizes, seeks usable information that has utility, needs to know how things work, edits reality, favorite question is "How does this work?".

IV. Creative: seeks hidden possibilities, responds to new and different, is adaptable to change, learns by trial and error, favorite question is "If?".
Appendix B

Survey Cover Letter
Dear:

I am currently working on a thesis for a Masters Degree in Computer Science at Western Michigan University. The topic is End User/Programmer Communication in System Development. As part of the research, I am interviewing programmers who have recently developed systems for DMA and the users of those systems to explore how the communication process was perceived during the development process.

Would you please help me out by taking a couple minutes to fill in the attached survey. Answer all questions with the answer that best fits. (Affected DMA supervisors have approved of this survey.)

In order to match people to certain types of communication patterns, I had to ask for names. Those will be kept strictly confidential. Only I will see these surveys. When I enter the results on the computer, I will use numbers assigned to a name, rather than the name, and then destroy the paper survey forms. Names of the systems, users and programmers of the systems, and the company will not be used in the thesis.

I need this form returned to me no later than (date). Please answer all the questions and send this back to me in the enclosed envelope. Thank you very much for your assistance on this project.

Sincerely,

Terry Hart
Appendix C

Survey Database Listing
Appendix D

Human Subjects Institutional Review Board Approval Memo
TO: Theresa Jan Hart
FROM: Ellen Page-Robin, Chair
RE: Research Protocol
DATE: December 8, 1988

This letter will serve as confirmation that your research protocol, "Programmer and End User Communication in Computer System Development" has been approved as exempt by the HSIRB.

If you have any further questions, please contact me at 387-2647.
BIBLIOGRAPHY


Brooks, P. (1986). People are our most important product. In N. E. Gibbs, & R. E Fairly (Eds.), *Software engineering education, the educational needs of the software community* (pp. 40-49). New York: Springer-Verlag.


Data Training, 6, 18-20.

Data Management, 24(6), 25-29.

Cliffs, NJ: Prentice Hall.

Sagamore Institute.

interaction requires teamwork, trust. Data Management, 24(7), 23- 
28.


4(3), 4-6.

J. Hartmanis (Eds.), Lecture notes in computer science (pp. 82- 

93-100.


Olson, M., & Schott, F. (1988). Designing usability in systems: 
Driving for normalcy. Datamation, 34(10), 68-76.

New York: Morrow Quill.

Richardson, W. (1986). Software Engineering Education, the 
Educational Needs of the Software Community. New York: Springer- 
Verlag, New York.

Computerworld, 19(40), 11-16.

New York: Addison-Wesley.

systems. Santa Monica, CA: Rand Corporation.


