APPROVAL SHEET

To the Graduate Council:

I am submitting herewith a thesis written by Laura L. Bresko entitled "Technical Communication and the Software Development Effort." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Arts with a major in English.

Donald Samson

Donald Samson, Major Professor

Norman Sanders

Lori Hall Burghardt

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Laura L. Breske

Date May 9th, 1989

TECHNICAL COMMUNICATION
AND THE
SOFTWARE DEVELOPMENT EFFORT

A Thesis

Presented for the

Master of Arts

Degree

The University of Tennessee, Knoxville

Laura L. Bresko

August 1989

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In 1988, the Society for Technical Communication (STC), the largest organization of technical communication professionals in the United States, conducted a survey to establish the current composition of the technical communication industry. The STC reported many findings on salary, education level, and employment, all of which indicate that the field is stable and growing. The computer software industry remains the largest employer of STC members, totalling 28%, with research and development employing 21%. This figure does not deviate from the findings of the STC's 1985 survey.[1]

The computer software industry is highly diversified; it combines the best (and often the worst) practices from the business world with innovations and information from the research and development world. Technical communicators in this industry usually see only two parts. Most of the work they perform is linked to either software development or product maintenance. Software development refers to those tasks that turn an idea or need for computer automation into an operational piece of software. Product maintenance refers to the activities that occur after an operational system has been released, such as fixing problems in the way the software operates. It also includes all those activities that provide support for the people who will use the final operational system (the users). These people are collectively referred to as "users." A great deal of literature has been generated by technical communication professionals on the writing of specific product maintenance documents known as user manuals. Very little technical writing literature addresses the activities of the technical communicator in the software development process, the very activity that creates the need for user manuals. The lack of publications on software development-related issues by technical communicators suggests that the technical

[1] Kenneth Cook Jr. and William Stolgitis, "Profile '88 - Survey of STC Membership," *Technical Communication* 36, no. 1 (1989): 39 - 42.

communication profession is ignoring any potential opportunities for growth within the computer software industry and compounding the communication problems that already exist within the software development effort.

The lack of research and involvement in software development by technical communication professionals has many critical ramifications for both the computer software industry and the technical communication profession. For the computer software industry, little guidance is available on improving communication with the end user during the development effort. Traditional methods for communication tasks, such as those used in the assessment of user needs (discussed in Chapter III), cause many product maintenance problems including cost overruns, design flaws, and end user dissatisfaction. Improvements on traditional methods are not being developed by technical communication professionals. Rather, systems analysts and software designers with little technical communication training develop and implement the "improvements." And as the writing of documents is currently the major method for communicating software development information, the lack of support by trained technical writers for software development documentation creates many problems. For the technical communication profession, the lack of research and associated literature limits the focus of the profession within the computer software industry, thereby limiting the scope and impact of its practitioners.

This thesis suggests that documentation of the software development effort is a critical process that is often mismanaged by software technicians and neglected by technical communication professionals in the computer software industry at large. It examines several causes of poor documentation and the problems that have occurred as the results. Finally, it offers a solution to these problems, and examines the solution's impact on a software development effort. Chapter II gives a brief history of technical writing and computer software development, presenting some of the historical perspectives that have led to the current state of communication in the software development effort. A brief survey of

technical writing literature that does address the software development effort is also presented here.

Chapter III examines the current state of the software development process and the documentation involved in that process. Software development methodologies are examined to establish the causes of poor documentation, and the effect of poor communication on the development process. Finally, a proposal is made to include a technical writer at the beginning of the software development life cycle as an optimal solution to software development and documentation problems. Support for this discussion is provided through the literature, an industry mail-response survey designed to assess the industry's potential acceptance of the technical communicator solution, and personal experience as a technical publications manager on various software development projects at Oak Ridge National Laboratory, a national research and development facility located in Oak Ridge, Tennessee. The findings of the mail-response survey are also examined in detail in order to support assertions of the importance of documentation to the software development process, the feasibility of the technical communicator solution, and the need for better training of the technical communication professional.

Chapter IV explores the proposal to include a technical writer at the beginning of the software development life cycle by examining the proposal within the framework of a generic software development effort. Both the benefits and drawbacks of the technical communicator solution are explored. Impacts of state-of-the-art technologies on the technical communicator solution will also be briefly explored.

Chapter V concludes by summarizing the new role of technical communication professionals in the software development effort, as presented in Chapters III and IV, in light of its overall contribution to the current and future computer software industries.

CHAPTER II

A BRIEF HISTORY OF TECHNICAL WRITING AND THE COMPUTER SOFTWARE INDUSTRY

"Technical communication" is a broad term used to describe the process whereby information (data) is transmitted accurately from one party to another. Transmission of data occurs in many ways, including verbal, physical, and written forms. The concern here is mainly with written transmission: the act of technical writing is the accurate recording of data (information) for the purpose of disseminating or transmitting it. The formal discipline of technical writing embodies the act or practice of technical writing, along with effective methods for disseminating recorded information.

HISTORY OF TECHNICAL WRITING

Recording and disseminating information is hardly a new concept; technical writing is an ancient skill that can even be traced back to prehistoric times when a cave man first decided to illustrate his hunting techniques on the walls of a French cavern. The act of attempting to communicate information by writing it down in words or illustrations qualifies the humble cave drawings as technical writing.

As societies continued to evolve from that of the caveman, so too did technical writing. The modern world is in debt to the ancient Babylonians and Egyptians for developments in art, agriculture, architecture, astronomy, medicine, and mathematics. And technical writing as well, for along with developments in these fields came the recording of new ideas for the purposes of sharing and preserving information. Though still rudimentary, the basic structure of technical writing was developed at this time. In fact, it is mainly through technical writing samples that have survived the ages that we are able to attribute so many intellectual advances to these early societies.

The ancient Greeks improved upon Babylonian and Egyptian advances. Philosophy, biology, psychology, and physics entered the realm of human thought through the Greeks. Their influence is present today in areas such as mathematics and medicine, and much of their technical writing, such as the Hippocratic Rules and Oath, is still used throughout the world.

The first known technical writer in the English language was Geoffrey Chaucer. He won this distinction with his instruction manual entitled "A Treatise on the Astrolabe." Issues still of great concern to modern technical writers are skillfully dealt with by Chaucer in this piece. According to Carol Lipson, "Five hundred and ninety years ago, Chaucer recognized the need for specifying at the outset of his work the purpose of the writing, the intended audience, the significance of the piece of writing to the reader and its place in the existing literature, the scope of the piece and its plan of development, and the existence of sources."[2] With Chaucer's treatise, technical writing emerged as a field of intellectual endeavor.

Many technical writers followed Chaucer's lead and the field grew. The most common genre for technical writers then, as it is now, was the instruction manual. Herman Weisman states that "modern instruction books originated as far back as the sixteenth century when the first manual on military weapons was written. Early instruction books covered the complete range of weapons for a branch of service."[3] Modern technical writing grew out of the writing of instruction manuals for all kinds of equipment, machinery, and other devices. The complexity of the manuals increased proportionately as the technology did, especially during World War I, and by the outbreak of World War II, a formal profession known as technical writing emerged.

---

[2] Carol S. Lipson, "Descriptions and Instructions in Medieval Times: Lessons to be Learnt From Geoffrey Chaucer's Scientific Instruction Manual," *Journal of Technical Writing and Communication* 12, no. 3 (1982): 248.

[3] Herman M. Weisman, *Basic Technical Writing* (Columbus: Charles E. Merrill: 1980): 4.

> "World War II brought a tremendous speed-up in research and technology. The country needed a quick and efficient way to explain new scientific devices and weapons to the nonscientists and soldiers who were going to use them. How could these recruits be taught to quickly operate and maintain their equipment? The answer was instructional manuals. Every piece of equipment needed an instruction manual, and the field of technical writing grew up almost overnight."[4]

Impact of the Computer Age on Technical Writing

The next notable impact on the field was brought about by the Computer Age, which began shortly after World War II. Before the war, the Germans made the "Enigma," a machine used for encoding military secrets and transmitting them to other installations. During the war, the Germans greatly improved upon the Enigma's original design. Allied Intelligence Forces spent most of the war trying to crack the improved Enigma codes.

> "It is known that the British rapidly constructed several new versions of the machine. . . . Changes in the German military Enigma, such as increasing the number of code wheels in their Navy version, soon led to the British machines becoming too small and slow to be of any use in deciphering German secret messages and new methods had to be found."[5]

Consequently, a great deal of research involving computers had brought great advances to computing technology. When the war was over, the focus of the computer research switched from time-sensitive military applications to improvements on computing technology as a whole. It is here that technical writing and computer development histories become intertwined and, consequently, interdependent.

The First Generation of computers was born in the early 1950s. It consisted of vacuum tube computers that were programmed in languages developed specifically to operate on only those machines. These languages were known as "machine languages" and had no real meaning to nontechnical humans. This first generation of vacuum tube, machine language computers had little effect on the field of technical writing. The users of these systems were

---

[4] Weisman, 4.

[5] Michael R. Williams, *A History of Computing Technology* (Englewood Cliffs: Prentice-Hall: 1985): 289.

mostly government facilities that had sponsored the work because it was far too costly and experimental for private industry to finance. Any necessary documentation was usually accomplished by printouts of the programming code or hardware operations guides. Both kinds of documentation were intended only for an extremely technical audience that had an understanding of machine languages.

The Second Generation of computers substituted transistors for vacuum tubes, increasing computing speed and cost efficiency and enabling the design of systems for business applications. Once again, however, all aspects of the system could only be handled by technicians. According to Shirk, "documentation was virtually nonexistent. It was not until the Third Generation of microminiaturized integrated circuit computers in the late 1960s that the use of problem- and procedure-oriented languages made any sort of documentation necessary."[6]

Documentation for the Third Generation was far from useful. Though the improved computing capabilities and performance costs made the systems affordable for many applications, the computer industry made scant use of technical writers. The 1970 STC membership profile survey showed only 11% of the members working in the computer industry.[7] Therefore, the authors of any documents produced during this period were most likely technicians, systems analysts, and/or programmers who assumed that their audience was made up of technicians just like themselves. Hence, the new-found computing power was, in many cases, unleashable.

---

[6] Henrietta Nickels Shirk, "Technical Writing's Roots in Computer Science," *Journal of Technical Writing and Communication* 18, no. 4 (1988): 308.

[7] Austin C. Farrell, "A Membership Profile of the Society for Technical Communication," *Journal of Technical Writing and Communication* 18, no. 4 (1971): 4-8.

The Fourth Generation of computers, emerging in the middle 1970s, had an impact on all facets of life. These minicomputers and microcomputers were intended, for the most part, to assist in the automation of daily tasks. Fourth Generation computers:

> "perform the basic arithmetic and logic functions and also support some of the programming languages (such as machine language) used with larger computers but are physically smaller, less expensive, and are generally limited in storage capability. These smaller systems are ideally suited for processing tasks not requiring access to large volumes of stored data. As a result of their low cost, ease of operation by non-computing personnel, and versatility, they have gained rapid acceptance. . . .[8]

Computing technology was finally affordable for the masses and analysts tried to consider this new variable in their designs. They attempted to make the Fourth Generation "user-friendly," a term given to computer systems that enable nontechnical individuals to use the computer by making the computer's capabilities transparent; machine languages are not user-friendly. But as the need for user-friendliness in computer systems increased, a serious communications problem occurred. Non-technical users were the main recipients of hardware, software, and documentation. Weiss describes users as:

> "people who must be *satisfied*. Organizations or individuals buy and develop computer technology with some goals, in pursuit of particular advantages. Generally, the users are the ones who must be convinced that the goals have been met and the advantages realized."[9]

Though the technicians, analysts, engineers, and programmers understood that the people buying computer systems (the users) had changed, they did not have the necessary communication skills, design techniques, or desire to satisfy the new breed of nontechnical users. Although Fourth Generation design and development methodologies incorporated new techniques aimed at increasing user-friendliness in the final product, results had not improved significantly from those obtained during the previous Generations. The theories and methods for developing software (known as "software development methodologies") were historically not concerned with incorporating user needs into the software design, and many

---

[8] William M. Fuori, *Introduction to the Computer: The Tool of Business* (Englewood Cliffs: Prentice-Hall: 1981): 57-61.

[9] Edmond H. Weiss, *How To Write A Usable User Manual* (Philadelphia: ISI Press: 1985): 4.

of the technical biases that the methodologies embraced are still present today in the Fourth Generation.

HISTORY OF COMPUTER SOFTWARE DEVELOPMENT

During the 1950s when the First Generation of computers was born, software development was approached in terms of the hardware it operated on. "Even advanced software 'systems' tended to be limited in scope and interacted little, if at all, with users and other systems."[10] Formal software development techniques had not yet been established and most work proceeded on a trial and error basis.

Traditional Software Development

It was not until the Second Generation, after the successes of the Fleet Ballistic Missile (FBM) and Polaris Projects, (both projects sponsored and run by the U.S. Navy Special Projects Office), that formal development and management techniques emerged.

> "The Special Projects Office has gained an international reputation for the innovativeness and effectiveness of the management control system it has employed in the development of the FBM weapon system. PERT, a computerized research and development planning, scheduling, and control technique developed initially in the FBM program, has been extensively used in numerous private and public organizations. Its adopters frequently note that PERT originated in the successful development of the Polaris. Similarly, management concepts and techniques such as project management, program budget, management control centers, and charting- all highly regarded by management analysts- are often acknowledged to have been developed or perfected by the Special Projects Office."[11]

The FBM and Polaris projects began in 1956, and their contribution to the world of research and development is still noticeable today. Although they were not specifically computer software development efforts, the usefulness of the new techniques had been demonstrated to be applicable to any research and development effort. Therefore, the same management and development techniques from these projects were used on many subsequent hardware and

---

[10] Robert L. Baber, *Software Reflected* (Amsterdam: North-Holland: 1982): 25.

[11] Harvey M. Sapolsky, *The Polaris System Development* (Cambridge: Harvard UP: 1972): 94.

software development efforts (collectively referred to as "systems development efforts") throughout the 1960s, 1970s, and 1980s. These techniques, which came to be known as "traditional," will be discussed in detail in Chapter III. Traditional techniques were designed to manipulate highly technical environments, and therefore certain biases were inherent, such as the lack of concern for the non-technical user. Such biases in systems development efforts caused many problems with a significant number of the final operational systems.

No significant improvements to the Polaris techniques were introduced until late in the 1970s with the Fourth Generation of computer technology. By this time the computer industry realized that there were very serious problems with the way they did business, as indicated by lack of user acceptance of Third and early Fourth Generation systems. Traditional development methods and design techniques were not working, as evidenced by soaring maintenance requirements and extremely low end user satisfaction. Much research was conducted to establish the reasons for so many difficulties. According to Harvey Sapolsky, "There is no one answer, but the following four factors certainly have an effect: premature installation of systems, lack of user involvement in system development, inconsistent design practices, and absence of documentation."[12] In response to these problems, "structured techniques" were developed. Structured techniques consist of methods for analyzing and documenting a development effort using diagrams and other graphic representations of information in order to represent the information sequentially and show interrelationships. The parent of structured techniques was the Structured Revolution.

The Structured Revolution

The Structured Revolution had its roots in the concepts that Alan Turing, a British mathematician, had researched during World War II. "One of Turing's more important concepts was that a limited set of only two or three basic constructs was required to build a

---

[12] David King, *Current Practices in Software Development* (New York: Yourdan Press: 1984): 4-5.

logical computing machine. The same constructs later became well known as the basic building blocks of structured programming."[13] Structured programming is the modular development of software based on the sequence of the process that the software is automating, the possible procedural selections within the sequence, and the iterative nature of the calculations involved in the automation of these sequences and selections. It took quite some time for structured programming to gain validity, as no successful system had yet been built using the technique. Finally, in 1972 a major breakthrough in systems development--using structured programming techniques--was achieved by Terry Baker. The new methodology swept through the industry, but by 1974 "it was painfully apparent bad systems were still being written and were absorbing multitudes of programmers into the maintenance black hole. . . . Clearly there was a large gap in the spectrum of structured methodologies: there was no effective method for *designing* programs and systems."[14] In response to this realization, Wayne Stevens, Glen Myers, and Larry Constantine introduced structured design in their 1974 article of the same name.

Stevens, Myers, and Constantine's methodology was by no means complete. Many organizations furthered their work, the most widely known being Yourdan, Inc. Ed Yourdan combined Terry Baker's systems work with Stevens, Myers, and Constantine's methodology for design, added some of his own input, and came up with the activities that the computer software industry refers to today as "structured techniques."

Structured techniques consist of methods for conducting and documenting software needs analyses and software design by exposing the interrelationships of information within a system and then using that information to program the software based upon those interrelationships. Martin and McClure, experts in the field of structured techniques, define

---

[13] King, 10.

[14] King, 11.

structured techniques in terms of objectives. "The primary objectives of structured techniques are as follows:

- Achieve high-quality programs of predictable behavior
- Achieve programs that are easily maintainable
- Simplify programs and the program development process
- Achieve more predictability and control in the development process
- Speed up system development
- Lower the cost of system development"[15]

The secondary objectives of structured techniques are:

- "Decompose complex problems and constructs into successively similar ones.
- Achieve simplicity of design.
- Control complexity.
- Achieve clear thinking about systems and programs.
- Use diagramming techniques that are as clear as possible.
- Improve the readability of diagrams and code."[16]

Structured analysis and design are, then, the attempt at a methodology that can solve the problems of lack of documentation, user involvement, and inconsistent design. Various diagramming techniques are employed to capture the design and user requirements in a "non-technical" way. These graphics, combined with text, are then supposedly schemas, or models, upon which the system can be built or implemented.

> "Given an appropriate diagramming technique, it is much easier to describe complex activities and procedures in diagrams than in text. A picture can be much better than a thousand words because it is concise, precise, and clear. It does not allow the sloppiness and woolly thinking that are common in text specifications."[17]

It would be truly revolutionary if a simple set of diagramming tools could clear up all the discrepancies involved in establishing a design. Unfortunately, structured diagramming techniques are tools for communication, as is the English language, and are

---

[15] James Martin and Carma McClure, *Structured Techniques and the Basis for CASE* (Englewood Cliffs: Prentice-Hall: 1988): 4.

[16] Martin and McClure, 5.

[17] James Martin and Carma McClure, *Diagramming Techniques for Analysts and Programmers* (Englewood Cliffs: Prentice-Hall: 1985): 2.

therefore equally subject to interpretation and manipulation. This makes the technique just as "woolly" as any other communications medium because it, too, relies on the communication skills of the person using it. As structured development teams currently make no use of trained technical communication specialists, the communicability of the techniques is highly questionable.

Although structured analysis is an attempt at user-friendliness and a significant contribution to the development process, it promises more than it can truly deliver, as evidenced by implemented systems with the same maintenance problems as traditional systems even though structured techniques were employed. Consequently, traditional and structured development techniques exist side by side in the development world, as neither is consistently successful at enabling the development of low maintenance, user-friendly computer systems that satisfy the user's needs. The current trend is to combine proven techniques from both methodologies. This, of course, has not solved the maintenance dilemma either; on-going maintenance difficulties point to a problem with user needs assessment and the requirements definition phase of the development process (refer to Chapter III). What both software development techniques have been successful at is creating a great deal of product maintenance work for technical communication specialists. Consequently, the current role for technical communicators in the computer software industry entails the explanation of systems, *including* operating problems, to end users.

THE TECHNICAL COMMUNICATOR AND THE COMPUTER SOFTWARE INDUSTRY

The maintenance problems experienced by end users during the Third and early Fourth Generations caused great consternation for the technicians of those systems. Consequently, professional communicators were brought into the computer industry to work with and satisfy the end users' needs. Shirk summarizes the difficulties in her article entitled "Technical Writing's Roots in Computer Science." She states that:

> "As the computer industry developed from the early 1970s into its present Fourth Generation of widespread mini- and microcomputers, rapid processing

> techniques, and multiplicity of software applications, a communication crisis occurred. The documentation techniques employed by the technicians of the past became increasingly ineffective as vehicles for communication to a newly emerging audience."[18]

Several factors contributed greatly to the communication problem that the computer software industry was experiencing.

> "First, computer technology became more accessible to 'end users' (people with little technical background), who began to demand professional documentation to tell them how to use the software products that promised to help them employ the computer as a problem-solving tool. Second, computer technicians began to develop improved, 'structured' techniques for designing software. As they became more involved in the documentation of their design processes, they became even less interested and less qualified to write end user documentation."[19]

Partially in reaction to the above situations, technical writing specialists (usually with humanities rather than technical backgrounds) were hired into the ranks of the computer industry for the purpose of documenting (translating) software systems for use by end users. As stated earlier, only 11% of the STC's members worked in the computer industry in 1970. By 1988, that number had risen to 44%.[20]

Prior to the computer industry's need for the technical writer's skill, most technical writers "served as middlemen--interpreters or chroniclers of scientific or technological matters researched, developed, or created by another person," usually a physical or life scientist or engineer.[21] The Third and Fourth Generations of computer technology changed the spirit of that definition. Technical writers were still interpreters, but their contributions were desperately needed by the computer industry in order to ensure its continued widespread usefulness and success. Without "usable" user manuals, maintenance problems would multiply, thereby increasing costs, ruining credibility, and causing the loss of

---

[18] Shirk, 309.

[19] Shirk, 309.

[20] Cook, 42.

[21] Herman M. Weisman, *Basic Technical Writing* (Columbus: Bell & Howell: 1980): 5.

customers (the end users).

The technical writer's initial introduction into the computer industry was based on the need for clear, user-friendly user manuals describing the operations of a software package or a piece of hardware. Technical writers are still writing user manuals or participating in product maintenance. So new is the computer industry to the technical writing professional that the extent of her contribution has not evolved into participating in the software development effort, or so pertinent research and literature would indicate.

SOFTWARE DEVELOPMENT PUBLICATIONS BY TECHNICAL COMMUNICATION PROFESSIONALS

Prior to the personal computing revolution, professional and scholarly technical writing publications focused on two areas: applied theory and theory. How-to-write manuals for business, engineering, and science comprised the main body of applied theory literature. The topics discussed most frequently included audience definition, style, expository techniques, and report writing. The second area that occupied most scholars' time was giving legitimacy to the profession by improved research and theory. The field of technical communication was experiencing rapid growth and many scholars were concerned that current practices were inadequate.[22] In the midst of this unresolved debate, the Personal Computing (PC) Revolution began in 1981, making Fourth Generation computing capabilities affordable for even modest budgets. The PC Revolution also brought with it a new focus for professional technical communication literature.

During the PC Revolution, the scope and direction of professional literature changed to include computer-oriented material. While debates over legitimacy continued, establishing a paradigm for the technical communicator's new role, as liaison between the user and

[22] See *Journal of Technical Writing and Communication* 10, no. 4, (1980) for a complete discussion of the debates and issues.

computer technology, inadvertently settled most of those issues. Much of the new literature and research focused on solving problems encountered by the technical communicator within the framework of his new role. Methodologies for writing user manuals, using desktop publishing effectively, and generating graphics became the new thrust. And as the technical communication professional's need for support increased, so did the frequency of computer-oriented publications. In 1981, only three computer-oriented articles were published in the *Journal of Technical Writing and Communication.* In the years following, the numbers increased annually. In addition, *IEEE's Transactions on Professional Communication* and *Technical Communication, Journal of the Society for Technical Communication* were devoting whole issues to the subject. In 1988, over 40 computer-oriented articles were published by the three professional journals, and at least 15 computer-oriented books written by technical communication professionals were in print during that year.

Of the many articles and books published over the past eight years, only *four* publications address the technical communicator in the software development effort. Of these, the article most germane to the present discussion is Bernice E. Casey's "The Impact of the Technical Communicator on Software Requirements."[23] In this article, Ms. Casey describes an area of software development--requirements analysis--that is especially suited for technical communicators. She proposes that technical communicators should make their expertise available to software engineers and help out during requirements analysis. The current state of software development (as discussed in Chapter III) proves that very few have taken Ms. Casey's suggestion seriously. However, technical communicators should not merely offer their expertise; they should perform accurate user requirements analyses, not systems analysts.

The current situation in the computer software industry is unfortunate; maintenance

---

[23] Bernice E. Casey, "The Impact of the Technical Communicator on Software Requirements," *Journal of Technical Writing and Communication* 11, no. 4 (1981): 361-372.

problems are worse than ever, and end user satisfaction has not improved. According to David King, "organizations are increasingly devoting most of their DP (data processing) resources to maintaining existing systems. . . . We see that many large DP organizations are expending more than seventy-five percent of their people resources on this (maintenance) type of work."[24] Even with technical communication professionals working to perfect applied theories in the computer industry, as evidenced by the growing number of articles being published annually, maintenance problems are multiplying. Efforts to improve software development techniques have failed to solve the targeted problem-- inadequate user requirements assessments. Though we are currently in the Fourth Generation, the Fifth Generation, characterized by smart computer systems or artificial intelligence, is beginning to emerge, carrying with it all of the problems of the current day. There is potentially a greater role for technical communicators to play in the software development process-- the role of the user expert. This role is designed to solve maintenance problems before they occur by conducting accurate user analyses. It is also to enable the incorporation of communication and documentation expertise into all software development work in order to improve the quality and utility of the final product. The role is critical if computers and software are ever to serve the nontechnical user fully. But as the next chapter demonstrates, the opportunity is currently passing the technical communicator by.

---

[24] King, 3.

CHAPTER III

THE CURRENT STATE OF SOFTWARE DEVELOPMENT AND OPPORTUNITIES FOR THE TECHNICAL COMMUNICATOR

The brief histories and problems presented in the previous chapter help define the current state of development in the computer software industry. The historical problems of maintenance still exist and have multiplied. Therefore, an examination of the two most widely used software methodologies is necessary.

TRADITIONAL SOFTWARE DEVELOPMENT LIFE CYCLE

Traditional software development employs the "life cycle" techniques generated by the Polaris project. The life cycle of a project can be defined as the common way project members conduct business and the phases of the business being conducted.[25] There are usually five phases to a software development project. They are as follows:

- Analysis
- Concept Development
- Requirements Definition and Design
- Implementation
- Installation

Many different names are applied to the various phases, depending on the organization. What remains consistent are the activities that take place during each phase. Though these activities may proceed in different fashions, or at different times, or even in slightly different sequences, they are all conducted to discover the same kinds of information.[26]

In addition to similar information needs, most organizations also use the same kinds of development personnel. Usually a team consisting of systems analysts, designers,

---

[25] Edward Yourdan, *Managing the System Life Cycle* (Englewood Cliffs: Yourdan Press: 1988): 43.

[26] The discussion of the traditional software development life cycle is based upon information contained in ORNL/DSRD-13, Mil Std 1521.B, DoD 7935.1, and DoD 2167A.

programmers, and managers is assigned to a specific project. Systems analysts examine both existing and proposed systems in order to establish deficiencies, alternatives, user needs, and impact. Designers work closely with systems analysts and are frequently the same person. Designers establish the model for the proposed system, incorporating all the findings of the analyst's analysis and including technical attributes of the new system, such as operating capabilities. Programmers develop the instructions for a computer to follow in order to enable the new system to operate according to the designer's model. Managers handle political and administrative functions both within their own organization and the sponsor's (user's) organization. The development team is responsible for the "deliverables" associated with the development effort. Deliverables are completed activities and documents that satisfy the information requirements of a given phase of development.

Phase 1 - Analysis

The objective in the analysis phase is to identify and validate the project sponsor's need for an automated system to replace an inadequate system. Occasionally, computer automation of a process may not be the best solution. Once the decision to use an automated system is made, possible systems solutions to the sponsor's problem are analyzed along with significant difficulties, assumptions, or constraints on the solutions. The development team then makes a preliminary recommendation of an alternative system to replace the current way of doing business. At this point, the sponsor decides whether or not to proceed with the exploration of the problem and its possible solutions. These activities occur in the next phase of the project.

During the analysis phase, the major deliverable is a document entitled *Mission Element Need Statement*. A Mission Element Need Statement is usually written when a sponsor identifies a need for an automated system and requests the problem to be solved. The purpose of the Mission Element Need Statement is to describe the deficiencies of the current system, to justify the exploration of alternative solutions, and to identify estimated

costs for developing those solutions. The author of the Mission Element Need Statement is usually a systems analyst. The document is usually prepared on government contracts.

Phase 2 - Concept Development

The Concept Development phase proceeds after the Mission Element Need Statement has been approved by the project sponsor. During this phase of development, an evaluation of the possible alternative solutions to the problems presented in the Mission Element Need Statement is conducted. An initial economic analysis of the various solutions is performed. Finally, a recommendation of the most feasible solution(s) for the sponsor's problem is made, along with a plan for the way the project shall proceed. The sponsor then decides which solution to employ to solve the problem presented in the Mission Element Need Statement. The sponsor's decision is based on the feasibility study conducted on all alternative solutions and the methods for developing the system. If the decision is favorable, money is committed and the project proceeds to the next phase of development.

Two major deliverables for the Concept Development phase of a project are the *Project Management Plan* and the *Systems Decision Paper*. The Project Management Plan presents the project history and controls continuity of operations throughout the life of the project. The Project Management Plan implements the project manager's management strategy, outlines a course of action and method of execution for system development, and serves as the foundation for the development of the Systems Decision Paper. The Systems Decision Paper summarizes the alternatives considered in the Mission Element Need Statement and any progress made on identification of optimal solutions to the sponsor's problem. It also addresses key issues in the Project Management Plan and presents a recommendation(s) for the most feasible of all alternative solutions. Systems analysts and project leaders are usually responsible for the Systems Decision Paper and the Project Management Plan. (These documents are usually written for government contracts.)

Traditional software development has a very regulated slant to its phases. This is because the methodology grew out of government-sponsored projects. The methodology is still most frequently employed on government-sponsored projects. Phase 1 and 2 are more involved in administration than actual development. In a generic form of traditional software development, Phases 1 and 2 would be incorporated into the activities outlined in Phase three.

Phase 3 - Requirements Definition and Design

In the Requirements Definition and Design phase, the user's needs are assessed in order to define and validate detailed functional requirements for the system's operational performance. After the user's needs have been assessed, various design alternatives that conform to the recommendation(s) made in the Systems Decision Paper are evaluated in order to determine the best design for the system. When a decision has been made, the economic analysis performed previously in Phase 2 is refined and a recommendation is made to the sponsor on the best design for full-scale development. If the sponsor accepts the recommendation, the software design is finalized, and the project proceeds to the next phase of development.

Many deliverables are associated with the Requirements Definition and Design phase of a project. Since the objective is to assess user needs and devise an optimal solution, all of the system's attributes are analyzed. Documents are then written that catalogue the results of these analyses. These documents usually include the *Functional Description*, *Data Requirements Document*, *System Specifications*, *Program Specifications*, and *Database Specifications*. Frequently another Systems Decision Paper is also written.

The Functional Description defines system and user requirements and provides users with a clear statement of the operational capability to be developed for them. The major accomplishment of the Functional Description should be development personnel's description of the proposed system in non-technical terms so that the user is able to understand it.

Systems analysts and software designers are usually responsible for analyzing user requirements and writing the Functional Description.

The Data Requirements Document is an extremely technical document. Its purpose is to list and define data elements that the system must handle. (Data elements may be defined as all the pieces of information that the system processes.) The Data Requirements Document also describes the types of information required by the developers in order to establish values for the system's data elements so that the system can perform its computations accurately. In addition to values, business rules (operational constraints) for each data element are also defined. Data elements also help to define the structure of the system's database, if it has one. A database is defined as an electronic storage device that allows for the storage and retrieval of a system's information, or data. Because of the relationship, the database administrator, who designs the structure of the database and keeps the information contained in it current, is usually responsible for accurately identifying all the data elements in a system and cataloging them in the Data Requirements Document.

The System Specifications is written to provide detailed definitions of the all the functions the proposed system must perform. It is also the means for communicating the details of the on-going analysis of user and system requirements to the sponsor. The environment the final operational system must function in is detailed here as well as the means for communicating with any other systems that current system under development may share information with. This is known as "establishing system interfaces." Many additional design elements are also included in the System Specifications. The document is usually written by many members of the development team, including (but not limited to) the systems analyst, software designer, database administrator, and systems administrator. The systems administrator is responsible for the operation of the hardware that the software under development must run on. This includes facilitating interfaces to other systems.

The purpose of the Program Specifications is to describe the program design in sufficient detail to enable the generation, or writing, of code by a programmer. The purpose of the Database Specifications is similar to that of the Program Specifications. It describes storage needs, database organization, and other basic design data necessary for the construction of system files, directories, tables, and dictionaries. The database administrator uses this information to construct the database. Both the Program Specifications and the Database Specifications are usually written by the same people who write the System Specifications.

Phase 4 - Implementation

The Implementation phase is conducted in order to develop, test, and evaluate an operable system that satisfies the system specifications as presented in the System Specifications and the Functional Description. The economic analysis is also updated at this time. Programmers are the main contributors in this phase. When they have completed coding of the system, tests of the system are performed to establish its strengths and deficiencies. Any deficiencies are corrected, and the sponsor then decides whether to implement the system.

The major documentation deliverables for this phase are the *User manual*, *Computer Operation Manual*, *Program Maintenance Manual*, *Test Plan*, and *Test Analysis Report*. The User manual is written to provide the user group with the instructions and references necessary for using the system effectively. Occasionally the programmers who coded the software write the User manual, but more frequently technical writers are used. The purpose of the Computer Operation Manual is to provide interested users with a detailed operational description of the system and its associated environment. The systems administrator is responsible for this document. The Program Maintenance Manual provides maintenance programmers with the information necessary to maintain the system effectively. Development programmers are responsible for the Program Maintenance Manual. The Test

Plan provides guidance for system testing and communicates to users the nature and extent of the tests necessary for evaluating the system. The Test Analysis Report documents the results of any testing, including the system implementation test. It also assigns responsibility for the correction of any deficiencies in the system and attempts to establish user confidence in the operation of the system. The task of testing, when actually included in a development effort, is usually subcontracted to an independent agency so as to ensure the integrity of the test results. These subcontractors usually prepare both the Test Plan and the Test Analysis Report, enlisting development personnel help as needed.

Phase 5 - Installation

Installation proceeds only when the sponsor approves the system and authorizes its deployment. During Installation, the system is set up for operation at its location. It is then operated and maintained throughout its lifetime according to appropriate guidelines. Once the system is installed, the development team's work is done, but the technical writer's and maintenance programmer's work is just beginning. Depending on the operating quality of the system and the Computer Operation Manual, maintenance programmers either have an easy job or (usually) an extremely difficult one. Maintenance programmers must smooth out any operating difficulties in the system by re-programming the software. They are then responsible for recording the software changes (updates) in the Computer Operation Manual. During Installation, the User manual is also frequently rewritten to reflect the actual, or real, operating capabilities of the system. User feedback on the software usually guides the rewriting effort, in addition to frequent updates to the system. These tasks continue until the software is either perfect, revised, or completely replaced.

STRUCTURED DEVELOPMENT LIFE CYCLE

As stated in Chapter II, the structured techniques were developed in the late 1970s by Yourdan, Stevens, Myers, and Constantine. The discussion presented here is based on Ed Yourdan's structured development methodologies as outlined in his book *Managing the*

System Life Cycle.[27] Yourdan's work is highly recommended for a detailed discussion on the use of structured techniques and diagrams.

There are nine phases or "activities" to a structured development life cycle. Though different names are used, depending on the organization, the activities are as follows:

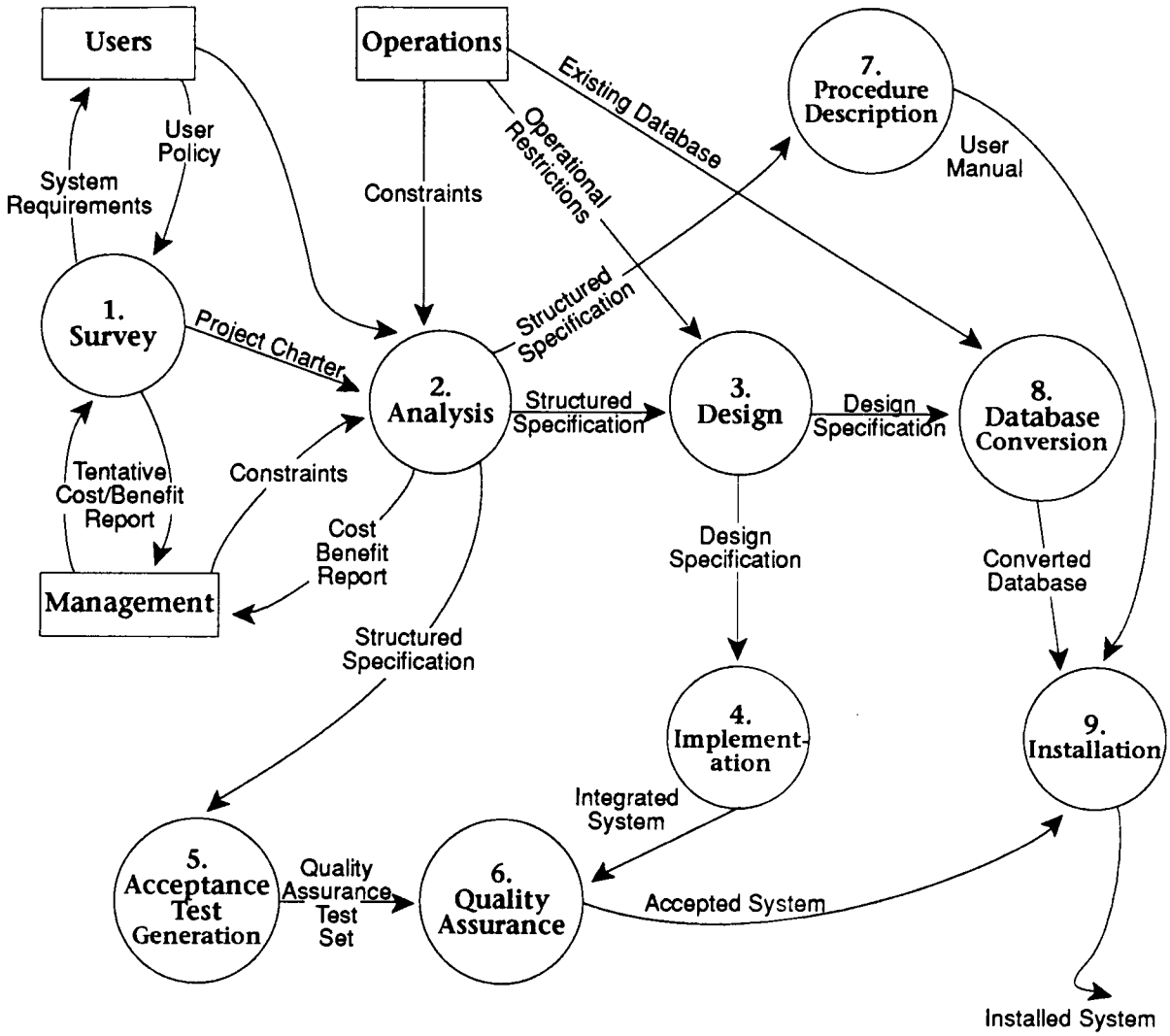
- Survey
- Analysis
- Design
- Implementation
- Acceptance Test Generation
- Quality Assurance
- Procedure Description
- Database Conversion
- Installation

These activities are non-sequential. Several may be going on simultaneously. Figure 1 shows the relationship of the activities using a structured diagramming technique known as the data flow diagram. It also exemplifies, in its complexity, the communication difficulties inherent in structured diagramming techniques, as discussed in Chapter II.

Activity 1 - Survey

Also known as the feasibility study, the Survey activity begins when a user requests that a portion of his business be automated. The Survey activity identifies current problems and deficiencies within the user's system environment and establishes new goals for an information system. It is also to determine whether it is feasible to solve the user's problem via automation and, if it is feasible, to suggest some acceptable ways of doing so. The major deliverable in this activity is a *Project Charter* that will be used as a contract to guide all activities involved in the remainder of the project. A systems analyst usually writes the Project Charter with the help of the manager/project leader and the sponsor.

[27] The discussion of structured development techniques is based upon Edward Yourdan's *Managing the System Life Cycle* (Englewood Cliffs: Prentice-Hall: 1988): 12-64.



Data Flow Diagram of the Structured Development Life Cycle

Figure 1.[28]

---

[28] This figure is adapted from Ed Yourdan, *Managing the System Life Cycle* (Englewood Cliffs: Prentice-Hall: 1988): 52.

Activity 2 - Analysis

The primary purpose of the Analysis activity is to transform user requirements and the Project Charter into a structured specification consisting of data flow diagrams and user business policies, which are the equivalent of "user requirements" in the traditional methodology presented earlier. The major deliverables in this activity are the *Structured Specification*, *entity-relationship diagrams*, *state transition diagrams*, and *process specifications*.

The Structured Specification consists of *data flow diagrams* and the *data dictionary*. The data flow diagram is a graphic means of modeling the flow of data through a system. Most systems require several levels of data flow diagrams. To create a level, a bubble (process) on the diagram is graphically "exploded," or focused on in greater detail, in order to examine closely the activities taking place in it. The data dictionary, which is usually a part of data flow diagrams, is simply an organized collection of definitions of all the pieces of information manipulated by the system and contained in the model. These pieces of information are referred to as "data elements." Systems analysts/designers are usually responsible for data flow diagrams and the attendant data dictionaries.

The entity-relationship diagram highlights the major organizations, systems, departments, or "entities," with which the system must interact. Relationships between the entities are also represented. Entity-relationship diagrams, then, show the relationships between information. They are different from data flow diagrams in that data flow diagrams only show *flows* of information. Systems analysts/designers are usually responsible for entity-relationship diagrams.

State transition diagrams show the time-dependent behavior of a system. In these diagrams, the various states that a system can be in are modeled, along with the relationships between these states and the activities that cause these states to change. These diagrams are

important when developing systems that handle rapidly changing pieces of information. Systems analysts/designers are usually responsible for state transition diagrams.

Process specifications allow the systems analyst to describe accurately the business policy underlying the lowest level of the exploded processes on a data flow diagram. Process specifications can be written in a number of ways, the most popular being structured English. Structured English combines programming language with the English language in order to describe an operation or a process. An example of structured English is as follows:

1. IF graduate student and writing thesis THEN
    a. choose topic
    b. read all available literature on topic
    c. write long research paper on topic
    d. take 1 oral exam

2. OTHERWISE IF graduate student and not writing thesis THEN
    a. take 1 written exam
    b. take 1 oral exam

Activity 3 - Design

The activity of Design is concerned with allocating the various parts (modules) of the structured specification to the appropriate development personnel. The appropriate development personnel then develop modules and design a structure for linking the modules of the system together. This design must reflect the structured specification and enable implementation. The major deliverable for this activity is usually a *structure chart*. Systems analysts/designers are usually responsible for structure charts.

Activity 4 - Implementation

Implementation includes both the programming of the modules and the integration of these modules into a progressively more complete system. Structured programming techniques are used in this activity. The major deliverables for this activity are coded, *integrated modules of the system* that are ready to be implemented. Programmers usually handle the deliverables for this activity under the direction of the systems analyst/designers.

Activity 5 - Acceptance Test Generation

Once the Structured Specification has been produced, work may begin on generating acceptance tests. Acceptance tests are devised that will test the system's capabilities and serve to assure the user that the system they ordered is the one they are going to receive. The major deliverable for this activity is a set of *acceptance test cases* that will be used during quality assurance testing. Systems analysts/designers usually generate acceptance tests, although independent sub-contractors are frequently employed to ensure the validity of the test results.

Activity 6 - Quality Assurance

Quality Assurance is also known as final testing or acceptance testing. This activity requires for input the acceptance tests developed in activity 5 and an integrated, operational system produced by activity 4 in order to test the design and capabilities of that system. The results are then compared with the user's needs as stated in the Structured Specification developed in activity 1. Sub-contractors may be asked to perform this activity to ensure testing validity. If not, systems analysts/designers and the end users work together on this activity.

Activity 7 - Procedure Description

The Procedure Description activity generates a formal description of the portions of the new system that will be automated and those that will be manual. It also provides a description of how the users will actually interact with the automated portion of the system. The major deliverable for activity 7 is a *User Manual.* Systems analysts/designers are usually responsible for the User Manual.

Activity 8 - Database Conversion

Database Conversion requires the user's current data, as found in the current database, as input into the new database. The current database is converted to accommodate

the requirements of the new system. The major deliverable for this activity is a *functional database*. If the design team has among its ranks a database specialist, then this person is usually responsible for delivering a functional database. If not, the systems analysts/designers handle the task.

Activity 9 - Installation

Installation is the final activity. In order to install the new system at the user's chosen site, all the major deliverables and assurances from all other activities must be completed and in final form. When possible, portions of the system are already operational at the user's site, thereby lessening the impact of installation on the users. The whole design team, consisting of the project leader, systems analysts/designers, and programmers, is responsible for an operational system.

PROBLEMS WITH CURRENT METHODOLOGIES

The two development methodologies just presented differ from one another mainly in how the development efforts proceed and what types of deliverables are produced. But they are theoretical depictions only. In the practice of software development, most efforts combine certain phases and/or activities. Combining methodologies is done for several reasons. Neither methodology is completely successful at producing an operational system. In fact, the low success rate for traditional techniques was one of the major motivators in creating structured development techniques. Had structured techniques become the cure-all, traditional methods would have been rendered obsolete, which they haven't. Because of this, traditional and structured techniques, and the associated deliverables, are frequently combined in order to employ what is perceived as the best attributes of both methodologies, with the hope of creating a newer, improved methodology that can be used to build an operational system.

Another reason for combining the two methodologies is to accommodate the talents of the project's development personnel. Software development personnel may be facile in only one methodology or the other. Frequently, senior personnel have strong backgrounds and years of experience with traditional techniques while younger personnel may have been taught structured techniques in college.

What becomes apparent is that regardless of which methodology is employed, none are able to solve the problems that currently exist in software development, and ultimately with the final system (if one emerges). Most software development efforts continue to have an extremely low success rate. In a report to Congress in 1979 by the Comptroller General, General Accounting Office (GAO), the findings on nine major computer software development contracts were as follows:

1. Dollar overruns are fairly common in more than 50% of cases.
2. Calendar overruns occur in more than 60% of cases.
3. Of the nine contracts examined (eight of which were admittedly in trouble), of $6.8 million expended the results were:
   a. Software delivered, but never used: $3.2 million.
   b. Software paid for, but never delivered: $1.95 million.
   c. Software extensively reworked before use: $1.3 million.
   d. Software used after changes: $198,000.
   e. Software used as delivered: $119,000.[29]

One of the major causes of these problems, as cited in the report, was the premature rush to develop systems before an adequate requirements analysis was completed. In 1988, a study was conducted by Kalle Lyytinen to determine the failure rate for information systems. According to his findings, 20% - 50% of information systems fail. "Despite the lower figures, these failure rates should still be a cause for concern. About 70% of all systems analysts find that from every fifth to every second information system fails in one way or another."[30] One of the major causes cited was inadequate communications about user needs

---

[29] These statistics were reported in Werner L. Frank, *Critical Issues In Software* (New York: John Wiley & Sons: 1983): 77-78.

[30] Kalle Lyytinen, "Expectation Failure Concept and Systems' Analysts' View of Information System Failures: Results of an Exploratory Study" *Information & Management* 14, no. 1 (1988): 49.

and problems. Ben and Marthalee Barton attribute systems failures to poor information technology models that do not focus on communications as a critical factor. In their article entitled "Communication Models for Computer-Mediated Information Systems," they state, with supporting research, that the "major reason most information systems have failed is that we have ignored organizational behavior problems in design and operation."[31]

A requirements analysis is the study of the user's organizational needs, behaviors, concerns, and current problems. A document cataloging requirements analysis findings is usually produced during or after the analysis has been completed. An adequate requirements analysis is critical to the success of a software development effort. According to Charles Sides, "It should be the first step in the design phase for a new product or for a product which is to be updated or changed."[32] An inadequate requirements analysis points to problems with the documentation process and/or the personnel in charge of conducting and/or managing the requirements analysis.

Inadequate documentation of a software development project, especially during the analysis phase, can destroy that project. According to W. Dwain Simpson, "the three most important components to a successful software project are 'Documentation,' 'Documentation,' and 'Documentation.'"[33] The reasons for such emphasis on documentation are good ones. The whole process of developing software depends mainly on the quality and accuracy of documentation. Analysts must accurately capture user and system requirements in a document (or diagram). Designers must then use the analysts' documentation in order to extract the needed information for producing a design. Programmers use the designer's

---

[31] Ben F. and Marthalee S. Barton, "Communication Models for Computer-Mediated Information Systems" *Journal of Technical Writing and Communication* 14, no. 4 (1984): 292.

[32] Charles H. Sides, *How to Write Papers and Reports About Computer Technology* (Philadelphia: ISI Press: 1984): 60.

[33] W. Dwain Simpson, *New Techniques in Software Project Management* (New York: John Wiley & Sons: 1987): 34.

documentation to code the system. Technical writers use design documents to begin user manuals. Funding, control and monitoring of a project rely heavily on the same documentation. In addition, "it is obvious that proper maintenance could never be performed if the documentation does not reflect the true state of the finished product."[34] Due to the dependency of development tasks on documentation in general, and more specifically to requirements specification documents, such as the traditional Functional Description or the structured specification, the lack of accurate documentation makes success an impossibility.

The use of personnel with inappropriate backgrounds for requirements analysis aggravates software development problems. Traditionally, systems analysts handle requirements specifications and the documentation of those requirements. They are usually responsible for writing user requirements documents. The act of specifying requirements is a complex communications task that involves interviewing the users, becoming completely familiar with the business needs of the user's organization, and becoming sensitive to the politics of that organization. Systems analysts are computer technology specialists who usually have a background in programming and/or engineering. This is not to assert that engineers and programmers cannot be good communicators. But there are specialists within the computer software industry who are trained in communicating--technical communicators and writers--who might be better suited for the job. It is no wonder that, according to the GAO, systems analysts want to ignore requirements specifications and move on to the building of the software; this is what they are trained for.

SOLUTION TO DEVELOPMENT METHODOLOGY PROBLEMS

Two of the main problems, then, with the real world of software development are inappropriate documentation techniques and inappropriate tasking of development personnel. The obvious solution to both problems is allowing the technical communicator to perform the

[34] Simpson, 67.

user requirements analysis and then including the technical communicator in all subsequent phases of development. This solution allows the appropriate personnel--technical communicators--to conduct user requirements analyses. By tasking the technical communicator with the first and most user-intensive phase in the development effort, the technical communicator becomes the user expert. The expertise in user needs then qualifies the technical communicator for participation in all phases of software development, improving overall project documentation, and ensuring user satisfaction. By improving communications with the user, and employing a communications expert to capture user needs, an accurate and complete requirements analysis is more likely. Consequently, all software development phases and attendant documentation will improve, including the final system and the user manual, simply because the crucial development/documentation task upon which all other development/documentation tasks rely--the user requirements analysis--has been performed by qualified professionals and therefore has a greater probability of success.

The technical communicator solution to software development problems is a massive expansion of the technical communicator's current role in the computer software industry. It is not likely to be well received by the industry at large. If the activities of technical writers in San Diego, one of California's silicon valleys where 90% of all technical writers are employed by the computer software industry, can be used as a representative sample for technical writers around the world, then technical writers within the computer industry are mostly preparing and/or writing operation manuals. In a survey, conducted by Little and McLaren, of technical writers in the San Diego area, 74% of all respondents reported that preparing operation manuals was their main task.[35] In the highly technical computer software industry, technical writers are mainly considered qualified for writing and editing operation manuals. Yet it is this very activity, coupled with the training provided by most academic programs, that qualifies the technical communicator for *any user-related tasks*,

[35] S.B. Little and M.C. McLaren, "Profile of Technical Writers in San Diego County: A Pilot Study," *Journal of Technical Writing and Communication* 17, no. 1 (1987): 9-23.

especially user requirements analyses.[36] But looking at the scope of the current tasks assigned to technical communicators, industry acceptance of having a technical communicator in the requirements or any other "technical" phase of a software development effort would not appear to be forthcoming. Within my own organization at Oak Ridge National Laboratory, project management has been told of the critical need for technical communicators throughout the software development effort. Only recently has management support been offered for the idea. Even now, the management technocracy sees the main benefit as improved documentation support. This is evidenced by the continued practice of incorporating communicators only when the writing process begins and not during the analysis activities.

Industry perception of the technical communicator is a major factor in the technical communicator's current role. The computer software industry is run by Third and early Fourth Generation technocrats who are not yet willing to relax their technological grip and allow for enhanced communications attributes in all aspects of a computer system. To find out the attitudes of *software development* project managers, systems analysts, and other technocrats in the computer software industry at large about the technical communicator's role on a software development project, and to establish whether or not an expanded software development role for the technical communicator would be accepted by them, I prepared and distributed a survey.

Survey of the Computer Software Industry

A survey was sent to 240 businesses nationwide that are involved with developing computer software (see Appendix A for complete survey and results). The surveys were addressed to owners, managers, and systems analysts within the individual companies as

[36] See *Academic Programs In Technical Communication* (Washington, D.C.: Society for Technical Communication: 1985) for a comprehensive discussion of technical communication programs in colleges across the nation.

opposed to technical communicators. 65% of the respondents handle government contracts, 20% work on private contracts exclusively, and 15% handle both government and private contracts (Question 14). The average number of years of professional experience for the respondents was 11. 107 responses were completed and returned by the cut-off date, giving the survey a 45% response rate. The survey questions and results are shown in subsequent figures.

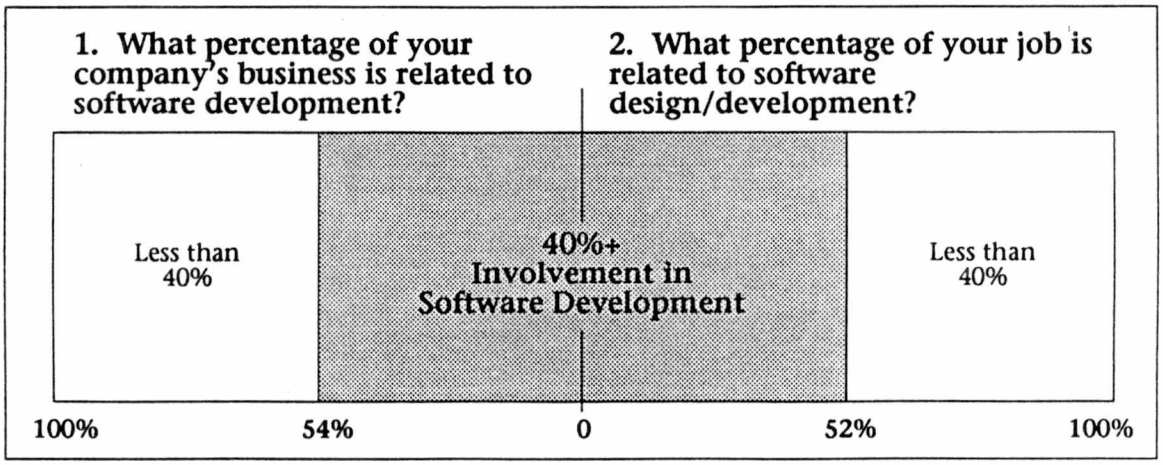
Figure 2 displays the results for questions 1 and 2. These questions were asked to establish the respondent's level of involvement in software development activities. Software development accounted for at least 40% of business in 54% of the respondents' companies. And for 52% of the respondents, software development accounted for at least 40% of their job activities. These percentages represent a moderate to high level of respondent involvement in software development activities.

The results of question 3 are contained in Figure 3. The question asked whether technical writers were employed by the respondent's organization. 67% of those responding to question 3 have technical writers on staff while only 33% do not. This percentage is relatively large and means that every two out of three businesses have technical writers on staff. However, the percentage coincides with the STC survey findings as reported in the February 1989 issue of *Technical Communication*.[37]

Question 4 was asked to determine the major activities for technical writers at the respondents' companies. The results are shown in Figure 4. Editing other people's documents is the primary task (71%) for technical writers in the computer software industry. Writing documents (i.e. user manuals) is the secondary task (52%). Development of graphics,

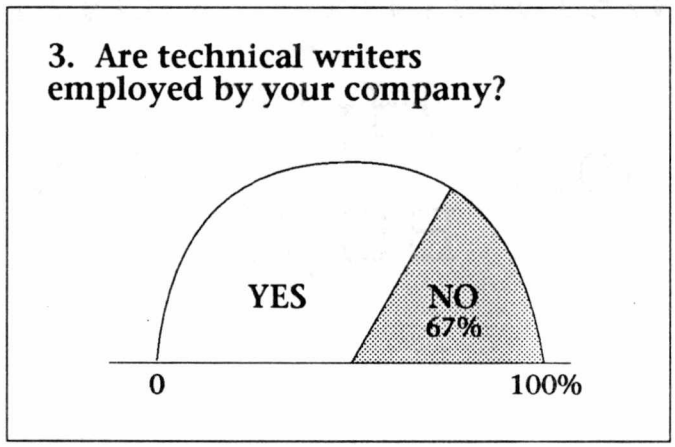
---

[37] Cook, 39 - 42.



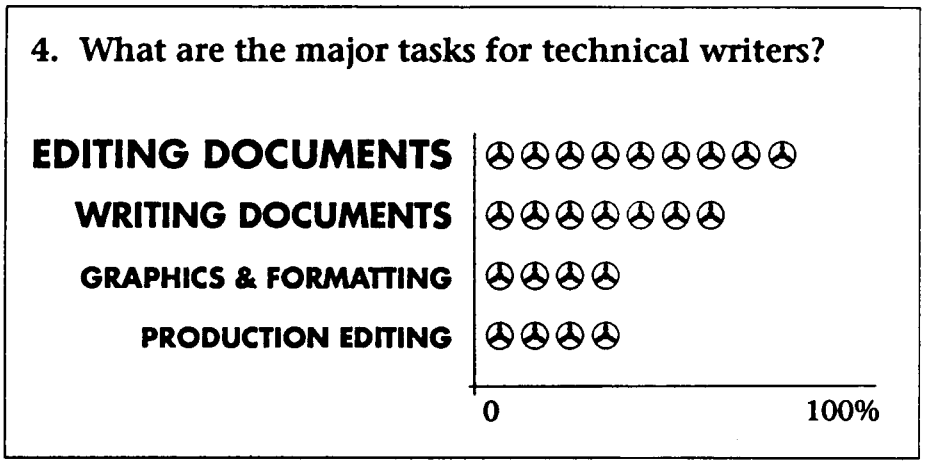
Responses to Question 1 and 2

Figure 2.



Responses to Question 3

Figure 3.



Responses to Question 4

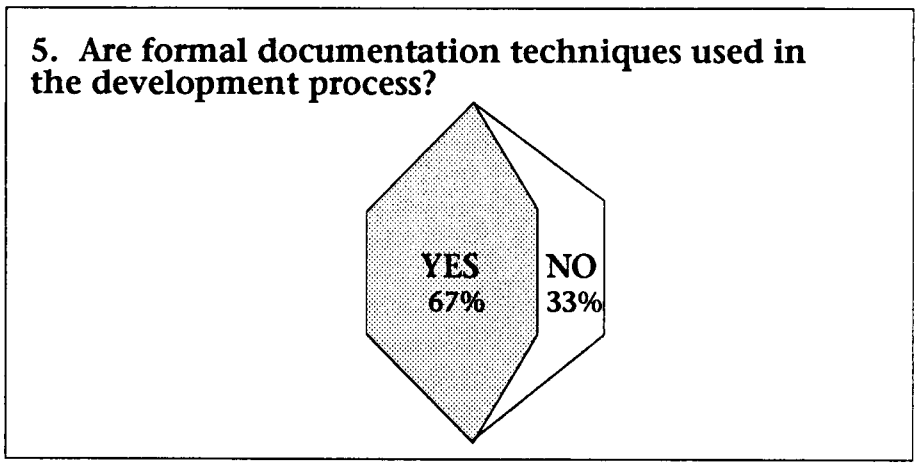
Figure 4.

formatting of documents (38%), and production editing (37%) are tertiary activities by only a small margin.

Question 5 asked whether formal documentation techniques are used during the software development effort. The results are shown in Figure 5. 67% of the respondents to this question use formal documentation techniques; 33% do not. If a negative response was given to this question, respondents proceeded to question 9. If a positive response was given, respondents then also answered questions 6, 7, and 8.

Most respondents who answered question 6, which asked about the effectiveness of currently employed documentation techniques, found their documentation techniques to be effective (58%). 32% responded that their techniques were very effective. 10% found their techniques to be *not* very effective. Question 7 inquired as to whether or not a technical writer was involved in the preparation these documents. 63% said that technical writers were involved. 37% did not use technical writers. Of those respondents who reported the use of technical writers during their software development effort, 47% of those technical writers did not begin involvement until the end of the writing process. 22% started work on the project when the writing process began. Only 31% began involvement at the beginning of the project (i.e. analysis phase). The responses to questions 6, 7, and 8 are shown in Figure 6.

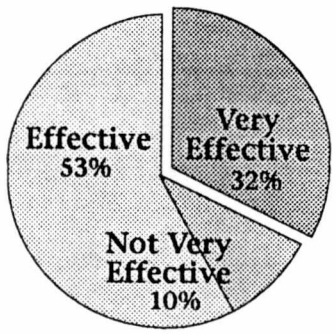
Question 9 was to determine whether a technical writer's contributions to a project would be perceived of any value. The results are displayed in Figure 7. 40% felt that the inclusion of a technical writer would definitely improve the documentation of the system. 34% felt that the inclusion of technical writer would help somewhat. 19% were only willing to commit to a maybe and 8% felt sure that a technical writer would not improve documentation. No respondent felt that a technical writer might make documentation worse.



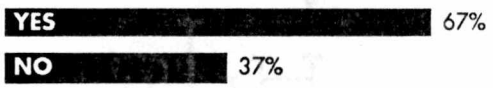
Responses to Question 5

Figure 5.

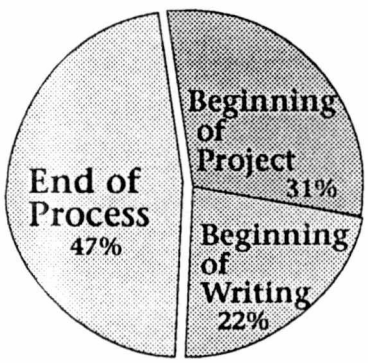
6. If formal documentation techniques are used, do you feel them to be effective?



7. Is a technical writer employed in the writing/ preparation of these documents?

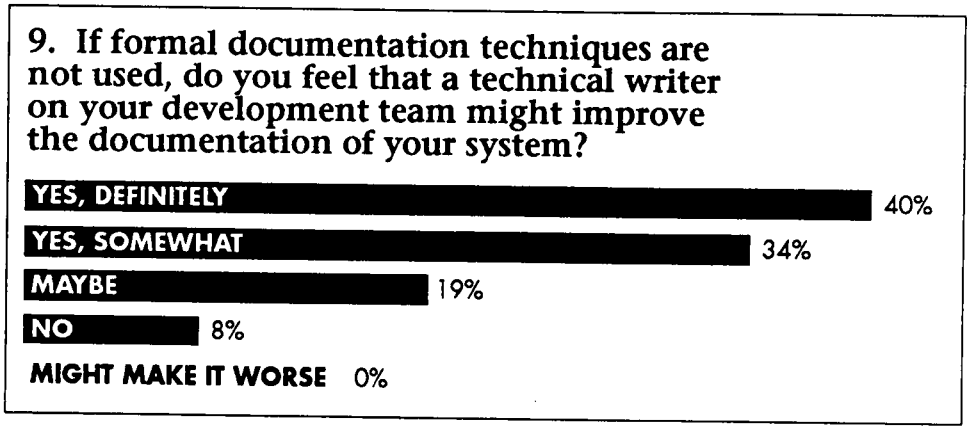


8. At what point does his/her involvement begin?



Responses to Questions 6, 7, and 8

Figure 6.



Responses to Question 9

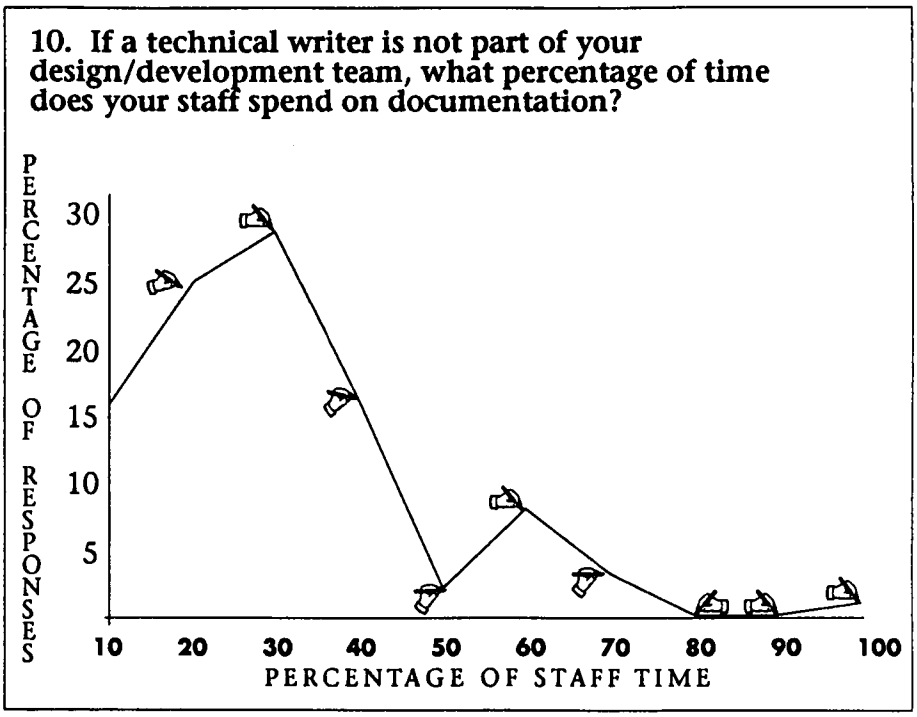
Figure 7.

Question 10 asked the respondents to determine how much time development staff spent on documentation when a technical writer was not used. The results are displayed in Figure 8. 29% of the respondents reported that their development staff spends 20 - 30 % of their time on documentation. 16% reported 30 - 40% of development staff time spent on documentation and 25% reported in the 10 - 20% range. The average for time spent on documentation for all respondents' software development staffs is 29%.

Figure 9 displays the responses to question 11, which asked about the importance of documentation. Respondents overwhelmingly considered documentation to be important (92%). Only 1% of the respondents felt that documentation was not important. 7% of the respondents had no opinion.

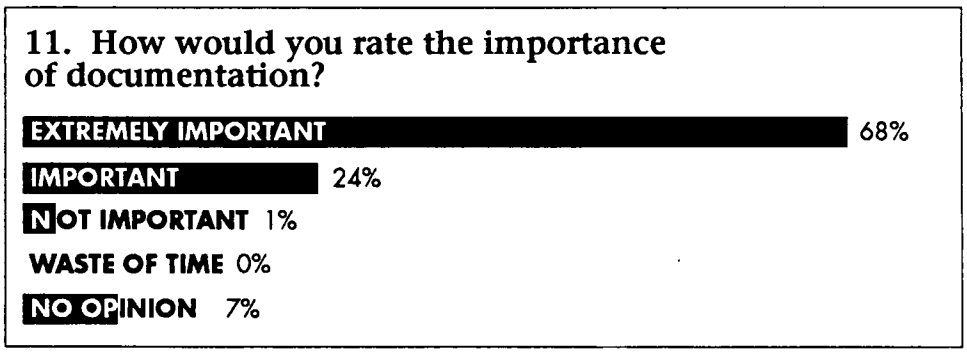
Question 12 was aimed at determining what backgrounds the respondents felt were critical for technical writers. Figure 10 shows that most respondents felt a multidisciplinary background was essential. Consequently, multiple fields were selected. Almost all respondents selected English/technical writing (92%). Most chose at least one other discipline. Computer science was considered the next most important field (45%). Communications closely followed with 43%. After these top three choices, no other consensus emerged.

In question 13, respondents were asked if they would hire a technical writer onto their software development team if she/he had the appropriate background. 83% of the respondents would hire a qualified technical writer. Only 17% reported that they would not, and in the majority of these responses, a note accompanied the answer explaining that they could not afford *such a luxury*. (Figure 11.)



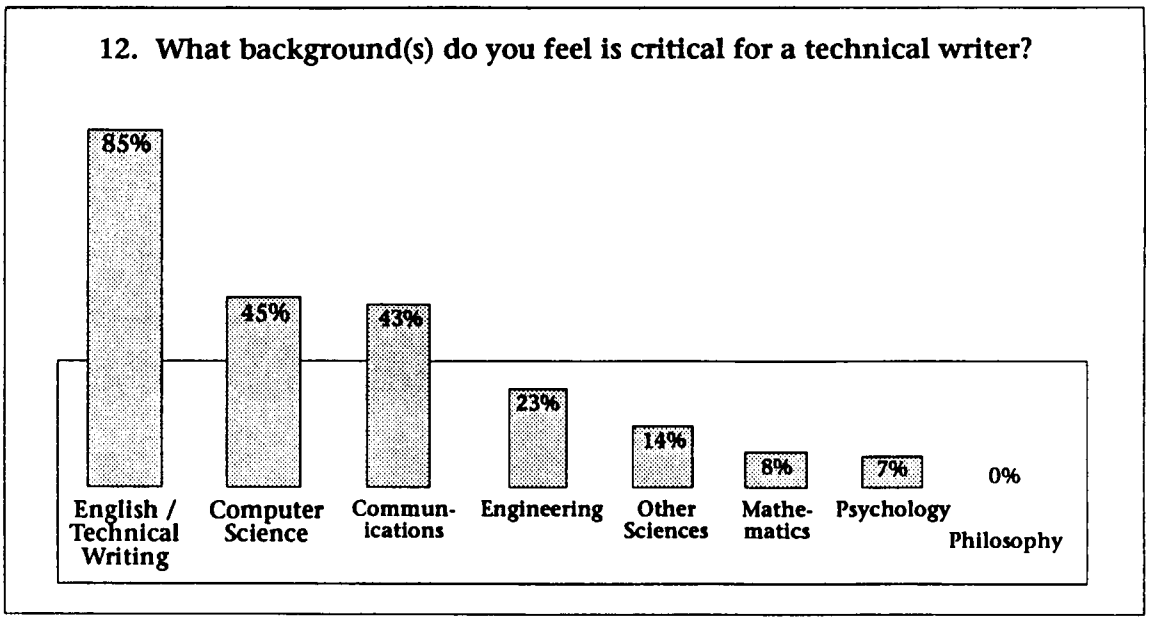
Responses to Question 10

Figure 8.



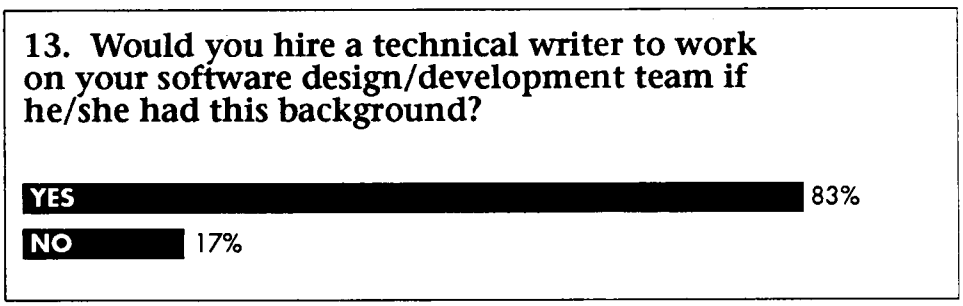
Responses to Question 11

Figure 9.



Responses to Question 12

Figure 10.



Responses to Question 13

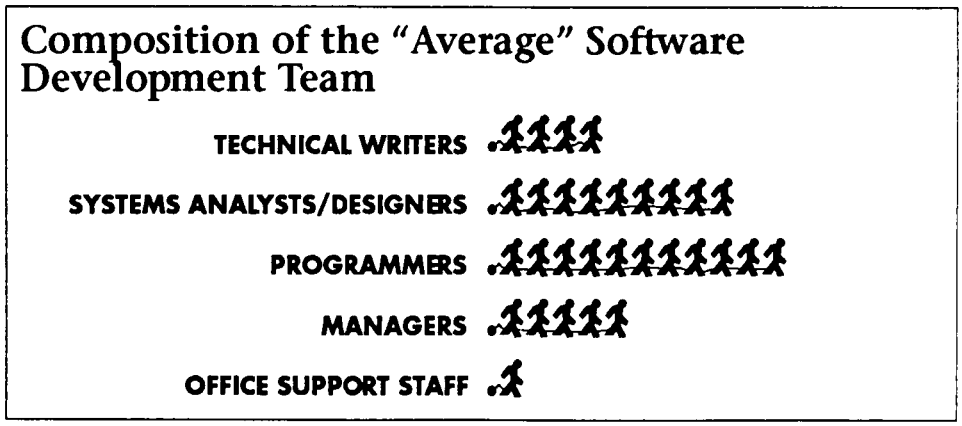
Figure 11.

Question 14 was asked to establish the kinds of software development contracts the respondent's companies worked on. As stated earlier, traditional techniques are usually employed on government projects. 65% of the companies represented in the survey responses handled mostly government accounts. 20% handled mostly private accounts and 15% handled both government and private accounts. (The ratio of government to private contracts was not stated.)

Questions 15 and 16 were designed to establish the composition of software development teams. According to the responses, the average software development team has 30 members. There are nine systems analysts/designers (these two are combined as most respondents felt they were the same), five managers, 11 programmers, one support person (i.e. secretary), and four technical writers. The results are shown in Figure 12.

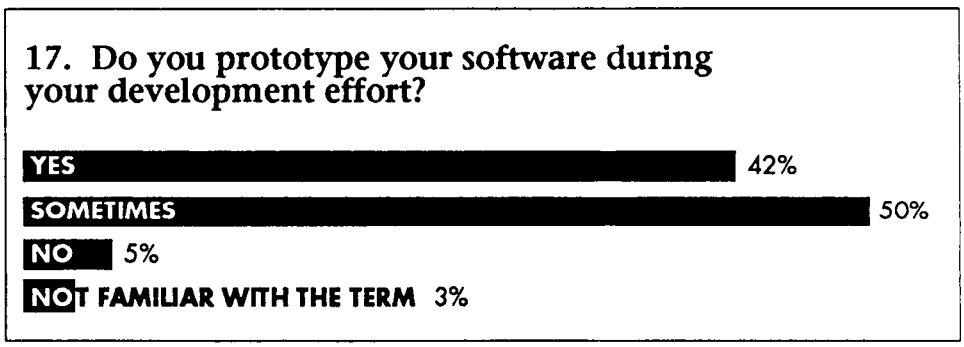
Question 17 asked about the use of prototyping. Prototyping is a technique where trial software is developed interactively with the user in order to better understand and capture the user's system requirements. It is a relatively new technique that has gained acceptance in the computer software industry. 42% of the respondents always prototype their software. 50% use prototyping techniques some of the time. 5% never prototype their software, and 3% are not familiar with the term (Figure 13). (Although question 17 seems to be outside of this survey's scope, it is actually a good indicator for the level of concern over capturing user requirements.)

Question 18 was the final question in the survey (Figure 14). It asked about the backgrounds of people in the respondent's organization that perform technical writing duties. 33% of the respondents' technical writers have formal training in technical writing. 30% have some formal training in the field. 36% have no formal training in technical writing, or on-the-job training only.



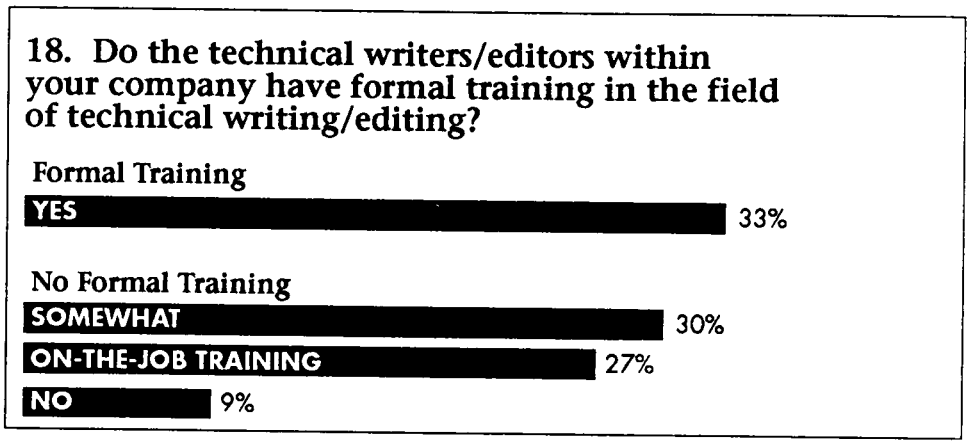
Responses to Questions 15 and 16

Figure 12.



Response to Question 17

Figure 13.



Response to Question 18

Figure 14.

Analysis of Survey Results

The results of the survey can be interpreted in many ways. Some questions were very general in order to allow the respondent's perceptions and biases to surface. The disparities in the responses are revealing. Most respondents feel that documentation is very important and that their documentation techniques are effective. The percentages correlate with the number of respondents who reported the use of technical writers in the writing/preparation of the documents. But the respondents' feelings clash significantly with the results of the GAO, scholarly, and private studies mentioned earlier. Two-thirds of the software development companies use formal documentation techniques when developing software. But only a little over one-third of the companies include technical writers at the beginning of the project, which is the most critical period. These technical writers are usually not trained in the field of technical communication. In addition, the role these technical writers play is mostly limited to documentation preparation and their main task is editing. Therefore, the four technical writers who are employed on the average software development team usually are not trained in technical communication and contribute mainly to the documentation editing and preparation process, not the software development work. If respondents were to hire trained technical writers to actually be a part of their software development teams, these writers would need an English/technical writing/computer science background (or other sciences). Most respondents would be willing to hire trained technical writers with this background. But according to the STC's 1988 survey, the respondents would have an extremely difficult time locating technical writers with such qualifications, as only two percent of the current STC membership have any computer science background.[38]

Impact of Survey Results on the Technical Communicator

What ramifications do the findings of the survey have on the technical communicator solution to software development problems? The current role for the technical communicator

---

[38] Cook, 40.

employed by software development companies is documentation support and preparation. Yet the survey findings indicate that the industry would be very receptive to the addition of trained technical communicators with computer science backgrounds to their software development teams. Therefore, the technical communicator solution might be received favorably by the computer software industry if the technical communication professional is able to respond to the industry's requirement of a computer science background. In order for the technical writing professional to help with the software problems plaguing the industry, it is logical that this professional be required to first have a demonstrable understanding of the theory and process that drives software development.

The need for a computer science background in order to be accepted as a colleague in the software development industry should not intimidate the technical communication professional. The need for the increased role of the technical communicator is too great. And there already exist many similarities between computer science and technical writing, such as the process by which both programs and documents are composed. Programmers gather all available, pertinent materials and write, in a logical order, according to the rules of the language they are using, programming code for the computer to interpret and execute. Technical writers also gather all available, pertinent material and write, in a logical order, according to the language they are using, words and sentences for their audience to interpret and frequently execute.

Several scholars have noted other similarities between computer science and technical communications. They have also observed the computer software industry's need for the expertise a technical communication specialist can offer. Bernice E. Casey cites process similarities. She finds the phases of development (both the software engineer's and the technical writer's) to be essentially the same. Both professionals have requirements, design, and implementation phases where the same analytical activities take place, only the products are different. She therefore concludes that:

> "the parallels between technical writing and software engineering are indeed striking. And yet there is no real cause for surprise, for similar phases of development are common to every well organized human endeavor. The discipline of software engineering has its roots in a scientific and literary tradition that is centuries old. Writers in particular have internalized this tradition, and follow it from day to day, almost unconsciously. And so it makes sense that we, as technical writers and editors, offer our communication skills to software engineers. We can do this most effectively by stepping out of our cubicles and offices in order to become active members of software development teams."[39]

Shirk notes historical similarities between computer science and technical writing. In her article entitled "Technical Writing's Roots in Computer Science," she explains the symbiotic relationship between the two disciplines and declares that

> "it is now appropriate for technical writers to join force with their colleagues in Computer Science on software development teams and in academia for the purpose of creating new visual and conceptual metaphors for communicating effectively. Technical writers will then become creators of the future rather than reinterpreters of the past."[40]

Ben and Marthalee Barton note the interdependencies and similarities between applied computer science and technical communication but state that computer science, in its applied role in society, is primarily a communications function. Computer scientists are continuing to ignore this key attribute of a computer system, and these scholars feel that:

> "we need systems based on models derived from sounder communication theory, for we need to close the gap between the rapidly growing perception of the centrality of communication and the technocentric focus of current system implementations. The gap is enormous, and will not be bridged by those who equate systems with word-processing units. Nor will it be bridged by those who settle for the modest role of handmaiden to technologists. It will be bridged only when modern communication theory is reduced to practice, and this calls for the central involvement of technical-communication specialists in system research and design."[41]

Taking the Bartons' vision even further is William L. Benzon. In his article entitled "The Computer and Technical Communication," he states that

---

[39] Casey, 363.

[40] Shirk, 322.

[41] Barton and Barton, 303.

> "the technical communicator has an essential role to play in ameliorating the present software crisis. At the very most, the communicator of today is the applications programmer of tomorrow."[42]

What is common amongst all five scholars is their belief that computing is a communications function and therefore communications experts should be involved in the process. Since the process revolves around computers, technical communicators must have a background in computers and/or computer science in order to have a complete understanding of the process. This becomes a prerequisite for the job. Mechanics must know about cars, doctors must know about their patients, and technical communicators who work in the computer software industry, must know about computers and software. When this becomes a standard for the profession, the technical communicator's impact on the software development effort will be pervasive.

---

[42] William L. Benzon, "The Computer and Technical Communication" in *Journal of Technical Writing and Communication* 11, no. 2 (1981): 110-111.

CHAPTER IV

THE TECHNICAL COMMUNICATOR'S NEW ROLE IN THE SOFTWARE DEVELOPMENT EFFORT

How would the software development effort proceed with the technical communicator involved in user requirements analyses? What impact would the technical communicator solution have on the current ways of conducting a software development project? In order to answer these questions, current software development methodologies must be reworked to include the technical communicator. Instead of concentrating on traditional or structured development methodologies, which have certain ideologies or slants imbedded in them, the following discussion will use generic phases of software development. As stated in Chapter III, the activities that occur during the development effort are the same; it is their sequence, names, or execution that may be different. Therefore, the discussion will concentrate on the activities of the technical communicator in the generic process of building a software system. Since we are only concerned with the technical communicator, the reader should assume that all other software development personnel's roles have remained the same except where explicitly stated.

THE TECHNICAL COMMUNICATOR IN THE SOFTWARE DEVELOPMENT EFFORT

As discussed in Chapters II and III, there are several areas within a software development effort that need communication expertise. By placing the technical communicator in a hypothetical, generic development effort, it is possible to explore those areas and assess what the new role for technical communicators actually entails. The phases in the generic software development methodology that follows are:

1. Requirements Analysis
2. Design
3. Implementation
4. Installation

These phases are the lowest common denominators between traditional and structured

techniques. In order to further mediate between the two methodologies, in the hypothetical case study that follows, deliverables have also been mixed.

Phase 1 - Analysis

In order to conduct a useful analysis of any system, the organization in which the system operates must first be fully understood. This level of understanding is too critical to the design effort to rely on third party information. Presently, the systems analyst learns from the manager of an organization how the individual jobs within the organization are performed. This is not optimal because a non-communications professional is getting information from only a secondary source.

In the technical communicator solution to software development problems, the technical communicator must go to the organization and become a part of it for a period of time, observing the organization in action. The amount of time spent there would depend on the size and complexity of both the organization and the proposed task. The technical communicator would conduct interviews with employees of all departments within the organization, observe all procedures for processing information, and become familiar with the behavior of the organization.

The deliverable for organization observation would be a business model. The business model graphically identifies the organization, its activities, and the interactions between activities. The business model would consist of organization charts by job functions, descriptions, and data domains. In addition to the organization charts, the business model would include data flow and entity-relationship diagrams for current processes, external events that effect the organization, and current logical and physical data structures.

An organization chart by job function would provide the project leader, systems analysts, and software designers with information about the hierarchical composition of the

organization. This chart would be not be very detailed but merely provide the skeleton on which the subsequent charts would be based. The diagramming technique used for this would be a structure chart.

An organization chart by job description would provide project leaders, systems analysts, and software designers with the political composition of the organization. It would provide information as to the nature of the positions in the organization such as:

- job functions
- necessary education
- required years of experience
- required skills
- special skills
- room for promotion
- average amount of time before promotion
- critical job functions
- organization interrelationships

These pieces of information would allow the political (and human) nature of the organization to be modelled and therefore considered in the ultimate design. The lack of such considerations puts the design team at a disadvantage when considering solutions to an organizational problem because they are not aware of all the factors influencing the situation. The diagramming technique best suited for this activity is a detailed structure chart.

The next organizational chart would detail the current information inputs and outputs of all departments. The information contained on this chart would include:

- department name
- department functions
- types of hardware used
- types of software used
- data input needs
- data origination
- data formats
- update sources
- data dependencies
- data type
- data description
- existing internal systems
- updates to external systems
- existing communications links
- frequency of use
- data outputs

- means of display
- user recipients
- security
- processes
- data ownership
- business rules

This information is necessary in order to establish the usefulness of the organization's current data and systems. All too frequently, software developers rely on bad data or system outputs that existed prior to their involvement with the organization. Not realizing the inefficiencies in the information architecture already in place, they develop an additional unstable information structure that is strained from the start because of its foundation. Without knowledge of the organization's whole information structure, it is impossible for designers to be sure that the user's stated problem is indeed the real problem. In addition to supporting the analysis of the organization's information architecture, information on data also provides the basis for the new system's data dictionary. The diagramming technique used for this chart would be the structure chart. (Showing flows of data is not necessary at this level.) In addition to the structure chart, the technical communicator should also store data domain information in a database.

The next part of the business model would be the data flows in the organization, including every department. The logical model would show current data business policies. The physical model would show the current implementation of those business policies as manifested in the set-up of current systems. And the event model would show all internal and external influences on the organization. Data flow diagrams would be the most effective diagramming tool to use. The actual activities that a business performs, known as processes, and the data relationships between these processes must also be represented. The entity-relationship diagram is the best tool for representing the process model.

After the technical communicator has finished the business model, work begins immediately on cataloging user requirements for the proposed system. Since users are not

necessarily computer-literate, extracting clear requirements from them is sometimes difficult. King states that "a claim often made by otherwise intelligent DP experts is that the users don't know what they want. Few statements can be guaranteed to raise quite so much anger. As the end-users are usually paying for the system in terms of development, operational, and maintenance costs, it is ludicrous for the DP staff to maintain that these same users are not aware of their own needs."[43] User requirements for the new system are usually based on user experiences with the old system. Therefore, instead of assuming that users are naive when they deliver design "wish lists" to the builders of a new system, the impetus behind the wishes, or the negative capability, should be uncovered and documented. Therein lies valuable insight for the whole development team as to the user's problem.

A requirements list should be drawn up based on the findings from the complete exploration of the user wants and needs. It should be stored in a database. The type of requirement should be listed. Categories should be devised by the technical communicator and the user that arrange requirements by type or function. The requirement for the new system should then be stated. The current system correlative should be listed next. The correlative allows for further identification of the nature of the requirement by plainly stating why and where the old system is deficient. The correlative is established through the negative capability in user experiences with their old system. The user's objective in stating a specific requirement should also be noted. Although it may seem that the objective is included in the statement of the requirement, analysts' and designers' perceptions frequently pervert the objective or intent of the user. Therefore stating it clearly removes any debate. Finally, the requesting user group within the organization should be noted, including the organization as a whole if the capability is needed by all users. Knowing the requesting user group allows for cross referencing with the information contained in the organization charts. By cross referencing user groups with the information contained in the organization charts,

[43] King, 5.

the nuances of that user group can be incorporated into the design of their requirement. When the requirements list is completed, it should also be cross-referenced with an generic attributes list for any computer system to ensure that all attributes have been defined or else purposefully omitted. If the requirements list is stored in a database, this should not be difficult. In addition to cross-referencing, the requirements list should also be hierarchically ordered within each category in order to provide another completeness check.

Preceding the requirements list should be a narrative where the proposed system is described. Included in this narrative should be:

- a description of the process that is being automated or updated
- the data handled in the process
- the physical environment that the proposed system will be installed in/on (photographs are appropriate when applicable)
- the affected departments within the organization
- the goals of the organization in automating the process
- user interfaces
- the length of time that the system must remain useful and operational
- any legal constraints

The more technical and detailed aspects of the system such as the operating system that the new system must work under, or the user's choice of a programming language (if a preference exists), are contained in the requirements list. The narrative provides only a high-level understanding of the process and proposed system.

When the requirements list is completed, the organizational and data flow charts that comprise the business model should be attached to the back of the list. The resulting document then transforms into a *Requirements Document*. A Requirements Document:

> "defines the problems that a system is to solve or the objectives it is to achieve. A requirements document describes the needs that the system is to satisfy for the expected user(s) and the constraints imposed by its intended environment. A requirements document should contain all the relevant information that a prospective user could possibly supply to those who will

develop the system. In effect, a requirements document is a description of the world into which the intended system must fit and the goals that it is to meet."[44]

After completing the Requirements Document, the systems analyst and software designer have all the pertinent information to make an informed recommendation for a solution to the user's problem. With the technical communicator's help, their recommendation should be documented and presented to the user, according to whatever methodology is governing the project.

Phase 2 - Design

During the design phase, the technical communicator's role as the user expert should continue. While analysts and designers are developing the design and database structure, the technical communicator should be involved in validating the design. "Although design issues are often technically oriented, the ability to relate the design to the user's operation is extremely important."[45] Due to the technical communicator's complete knowledge of the user's organization obtained during phase 1, he/she has this ability. It is critical to the overall success of the project to ensure that the users are getting the system they requested while the design is being formulated. Trying to match the system to the user's view after the design has been completed is far too costly. The technical communicator should provide the liaison between designers and users.

The technical communicator also has a specific design task to complete during this phase. Since the technical communicator has already specified user interface requirements in the Requirements Document, it is only logical that an area of the system design that

---

[44] Russell J. Abbott, *An Integrated Approach to Software Development* (New York: John Wiley & Sons: 1986): 9.

[45] Robert O. Peterson, *Managing the Systems Development Function* (New York: Van Nostrand Reinhold: 1987): 60-61.

depends so much on being able to see through the eyes of the user be completed by the user expert-- the technical communicator. The user interface is the view through which the user will look at the final system. (The user interface in this thesis is the printed page arranged according to the guidelines established by the University of Tennessee Graduate School.) A user can request a system to look a number of ways, based on needs. If simplicity and readability of system output are requirements, then information that is displayed on the computer screen must be arranged accordingly. The technical communicator would work with designers and users to complete this task. The deliverable would be a user interface that the users found acceptable.

The technical communicator's other task and deliverable during the design phase should be the creation of the proposed system's data dictionary. Although the data dictionary is usually associated with the database, "a data dictionary system is viewed simply as a tool for managing corporate data resources."[46] Since it is the technical communicator's job to ensure that the user's corporate data resources are being effectively managed by the new system, the technical communicator should (logically) develop the data dictionary. Having examined the user's data in the analysis phase, the technical communicator is extremely familiar with the nature of the user's data. The technical communicator can perform the data dictionary task with the greatest ease and accuracy. The technical communicator will also then be able to ensure the accuracy of information contained in other development documents, such as the System Specifications and the Program Specifications, because the technical communicator was instrumental in generating the information that the analysts, designers, and programmers used in writing them. In addition, the data dictionary task dovetails with the technical communicator's role as user liaison.

---

[46] Ronald G. Ross, *Data Dictionaries and Data Administration* (New York: AMACON: 1981): 15.

Throughout the design phase, as well as any other phase, the technical communicator should provide documentation support for all project deliverables. By conducting the requirements analysis and creating the data dictionary, the technical communicator is able to enhance the quality, authenticity, integration, consolidation, and availability of documentation. In doing so, valuable information can be recorded and disseminated to the users, current project members, future project members, management, current and future software development efforts, and the professional community at large, facilitating the success of the current project and also adding to the body of professional software development knowledge.

Phase 3 - Implementation

Implementation is usually the most lengthy stage of development in current development life cycles. It is during this phase that most problems with the system's design and the user's level of satisfaction surface. If the technical communicator has done an adequate job at defining the user's problem in the requirements phase, defending the user's position in the design phase, documenting the user's data in a data dictionary, and providing content validity checks and editorial support on all development documentation, then Implementation should become substantially simpler.

The technical writer must continue his vigil as user watchdog throughout Implementation. However, his major task and deliverable is the user manual. Since he must thoroughly familiarize himself with the new system in order to write the user manual, the task facilitates his watchdog role. It also puts the technical writer in the critical position of testing the usability of the system and identifying any "bugs," which are problems within a system that cause that system to malfunction, that may exist in the software. This allows for early detection, thereby allowing corrections to be incorporated into the design and program.

It also reduces the problems that could potentially occur during testing, helping to keep costs down. In addition to these tasks, the technical writer must provide documentation support for all other deliverables during this phase.

Phase 4 - Installation

The technical writer's role during Installation is to validate and update the information contained in the user manual. If updating or changes are required, the technical writer must also field user questions about the system's operation until a correct user manual is available. The technical writer should also provide documentation support to all other deliverables and any eventual updates.

ANALYSIS OF THE TECHNICAL COMMUNICATOR IN THE SOFTWARE DEVELOPMENT EFFORT

The technical communicator solution has many benefits for the software development effort. To answer the earlier question of the effect of the technical communicator on the software development effort, it appears that the technical communicator can make the effort easier and more efficient. In addition the technical communicator can improve documentation for the entire project. By having the technical communicator perform the requirements analysis, the probability of a complete analysis is greater. By performing the analysis according to the method described, the technical communicator should be able to document all the information about a system that the design team could ever require. In addition to facilitating a good design and developing quality documentation, the technical communicator also develops a relationship with the user based on mutual respect and trust. That relationship benefits the design team in many ways, including acceptable user interfaces and end-user satisfaction. The user organization benefits by having a champion on the design team that will protect their interests in the technical environment of a software development effort. In addition, they receive a business model that they can use on future system endeavors.

In allowing the technical communicator to complete the data dictionary, the project reaps additional benefits. It is safeguarded from the inappropriate manipulation of data. Frequently, data needs are made to conform to design needs, making the data less accessible for the user. And quite often, data needed by the user is simply omitted from the system due to the design team's lack of awareness. Since the technical communicator is the user's watchdog on the development effort, these difficulties are less likely to occur if the technical communicator develops the data dictionary, ultimately saving both time and money.

The impact of the technical communicator on current software development methodologies is minimal. The expanded role for the technical communicator should not disturb any organization's particular development methodology. By employing a technical communicator to perform requirements analyses, the systems analyst is allowed to concentrate on the task he or she was trained for--analyzing and designing systems. And by allowing a development staff member to become a user expert, both the user and the rest of the development staff are assured that their questions and concerns are being addressed. The technical communicator solution, therefore, has in it all the elements necessary to solve the two major problems in software development today.

There is one significant drawback to the technical communicator solution. It relies heavily on a technical background. This is not to assert that every technical communicator in the computer software industry needs to take another Bachelor's degree in computer science. But it does assume that a technical communicator has a thorough understanding of software development techniques and has mastered the tools used in those techniques. It is essential to the technical communicator solution that development team members be able to assume that the technical communicator is able to use development tools and also be able to rely on the quality of the results. A degree is not required for this, but a high level of commitment and a great deal of background work is. As soon as the technical communicator can convince the computer software industry that the techniques have been mastered, then

the technical communicator will be able to give highly valuable assistance to the software development effort.

THE TECHNICAL COMMUNICATOR IN THE TECHNOLOGICAL FUTURE

The last several chapters have demonstrated the need for enhanced technical communication in the computer software industry in order to resolve many of the difficulties now facing the industry. In order for the resolution to occur, the technical communicator must become a part of the software development effort, bringing to it invaluable communication expertise. But in order for the technical communicator to be effective and accepted in the role of communication expert, the technical communicator must have a background in computer science.

A background in computer science will not only allow the technical communicator to be more effective within the current computer software industry, it will also provide the skills necessary for future participation in a technological industry that will increasingly rely upon communication expertise.

In his article entitled "The Future of 'Writing' for the Computer Industry," Haselkorn cites the following specific changes to the role of the technical writer in the computer industry:

> "(1) 'Writers' will be involved earlier and earlier in the process. . . .
>
> (2) 'Writers' will test for usability, not only to adjust documentation, but to feed results back into design and development. . . .
>
> (3) 'Writers' will become more technically sophisticated (without sacrificing their special knowledge of user perspectives and communication skills). . . .
>
> (4) 'Writers' will play a major role in the design and development of user interfaces. . . .
>
> (5) 'Writers' will help develop formal methods for linking program functionality with interface and documentation. . . .

(6) 'Writers' will apply work in natural language processing and knowledge engineering to help develop smart, online documentation systems. . . ."[47]

Haselkorn shares the vision of a future role for technical communicators that will call for a greater degree of technical sophistication. Many of his changes have already been called for in this paper as necessary for the current world. Regardless of the time frame, technological developments will necessitate change. A technical communicator with a computer background will be ready to respond to these changes.

There are already many technological advances under development today that will change the current composition of the computer software industry, and the role of the technical communicator within the industry. Online help facilities and online documentation, coupled with hypertext capabilities, are quickly becoming standards for computer software packages and will eventually all but replace traditional product maintenance documentation. Hypertext can be defined as the three-dimensional linking of information in a database in order to facilitate access and retrievability of that information in whatever sequence the user deems necessary. Hypertext, along with online documentation and help facilities, are electronic in format, and therefore the developer of these capabilities must be able to write programs and design databases. They are also tools that facilitate communication of technical information, therefore the developer must be a technical communicator.

In addition to product maintenance, advances in the specific area of software development are forthcoming. Applications generators and application code generators linked to CASE tools are currently available. Applications generators enable a customized piece of software to be built within a matter of minutes. Applications code generators produce executable lines of programming code based on the requirements of the application generator or the design details contained within a CASE (computer-aided software engineering) tool.

---

[47] Mark P. Haselkorn, "The Future of 'Writing' for the Computer Industry" in *Text, ConText, and HyperText* (Cambridge: M.I.T. Press: 1988): 8-11.

Design details are entered into a CASE tool which currently performs consistency checks on the details. In the future, CASE tools may be able to build whole systems, via applications code generators, based on design details that a designer, or quite possibly a technical communicator, enters into the tool.

Although these technologies have not yet been perfected, they will affect the way that systems are built. Currently, prototyping is the major way to capture user requirements and design details, as evidenced by the widespread use of this technique among the respondents to the survey presented in Chapter III. Because code must be written in the prototyping environment, programmers or systems analysts currently handle the task. With the advent of applications generators, technical communicators may be handling that activity as well as populating CASE tools. If the technical communicator is willing to obtain a computer background, his/her role as detailed in Chapter IV may become a reality. If the technical communicator, with a new level of technological sophistication, is conducting user analyses, the logical extension of this activity is developing design details, hence the use of CASE. As stated earlier, Benzon feels that the technical communicator of today is the applications programmer of tomorrow. It is mainly due to the advent of applications generators and CASE tools that he is able to feel this way. But it is only through the enhanced technological sophistication of technical communicators that Benzon's vision for the future will be realized.

CHAPTER V

CONCLUSION

The rapid pace of technology has permeated every facet of modern life. Whenever we do business with a bank, pay our taxes, use a credit card, or place a phone call, a computer system is involved in the transaction. In addition to these every-day activities, computers have helped to lessen the severity of the life-threatening situations that we occasionally face, such as surgery or an accident. Though most individuals are aware of the benefits, when computers were still a promise for the masses, back in the late 1960s, technology was not always held in a positive light. Brian Murphy summarized the attitudes of the day in the following passage:

> "We are vaguely aware that computers are powerful extensions of man's intellect but they are inexplicable to most of us and consequently we stand in awe of them much as primitive man stood in awe of powerful demonstrations of natural phenomena such as thunder and lightning, regarding them as work of gods. Like thunder and lightning, computers are the product of explicable factors; with understanding of these factors, awe fades though wonder may remain."[48]

Unfortunately, the awe has not faded for many people due to a number of reasons. As this thesis has demonstrated, the lack of effective communication is at the base of the problem.

Covvey and McAlister summarize the reason for the communication problem that currently exists in the computer industry:

> "Just as you the user don't totally understand the computer, computer specialists cannot be expected to understand your business, institution, or personal interests as well as you do. Therefore the first step in coming to grips with automation is communication: communication between user and implementor."[49]

Given the current state of affairs in the software development industry, it is apparent that the first step has not yet been taken successfully. Technical communicators can resolve

---

[48] Brian Murphy, *The Computer in Society* (London: Anthony Blond Ltd.: 1966): 4.

[49] H. Dominic Covvey and Neil Harding McAlister, *Computer Consciousness: Surviving the Automated 80s* (Reading: Addison-Wesley: 1980): 5.

the communication dilemma that is diminishing the benefits of technology. But first they must be willing to achieve technological sophistication. When that occurs, users won't have to come to grips with automation; automation will come to grips with them.

In achieving technological sophistication, technical communicators can make a significant contribution to improving the quality of life in the Computer Age. And they are able to do so without losing their special skills for transforming the complex into the simple, for according to William Benzon, "the technical communicator's deepest contribution stems from the fact that he or she is deeply a humanist. No matter what changes the computer brings to our world or to our profession, we will be doing then as we do now, bridging the gap between man and machine."[50]

---

[50] William L. Benzon, "The Computer and Technical Communication," *Journal of Technical Writing and Communication* 11, no. 2 (1981): 112.

BIBLIOGRAPHY

BIBLIOGRAPHY

Russell J. Abbott, *An Integrated Approach to Software Development*, John Wiley & Sons, New York, 1986.

Academic Programs In Technical Communication, Society for Technical Communication, Washington, D.C, 1985.

Paul V. Anderson, "The Need for Better Research in Technical Communication," *Journal of Technical Writing and Communication*, 10:4, 1980.

Robert L. Baber, *Software Reflected*, North-Holland, Amsterdam, 1982.

Edward Barrett, editor, *Text, ConText, and HyperText*, MIT Press, Cambridge, 1988.

Ben F. and Marthalee S. Barton, "Communication Models for Computer-Mediated Information Systems" *Journal of Technical Writing and Communication*, 14, no. 4, 1984.

William L. Benzon, "The Computer and Technical Communication," *Journal of Technical Writing and Communication*, 11, no. 2, 1981.

R. John Brockmann, *Writing Better Computer User Documentation*, John Wiley & Sons, New York, 1986.

Christine Browning, *Guide to Effective Software Technical Writing*, Prentice-Hall, Englewood Cliffs, 1984.

Charles T. Brusaw, "Dismantling the Tower of Babel in Computer Documentation," *Journal of Technical Writing and Communication*, 10:2, 1980.

Albert F. Case, *Information Systems Development*, Prentice-Hall, Englewood Cliffs, 1986.

Bernice E. Casey, "The Impact of the Technical Communicator on Software Requirements," *Journal of Technical Writing and Communication*, 11, no. 4 1981.

Robert N. Charette, *Software Engineering Environments*, Intertext Publications, New York, 1986.

Robert Chartrand, editor, *Computers in the Service of Society*, Pergamon Press, New York, 1972.

Kenneth Cook Jr. and William Stolgitis, "Profile '88 - Survey of STC Membership," *Technical Communication*, 36:1, 1989.

H. Dominic Covvey and Neil Harding McAlister, *Computer Consciousness: Surviving the Automated 80s*, Addison-Wesley, Reading, 1980.

Tom DeMarco, *Structured Analysis and System Specification*, Yourdan, New York, 1979.

J. Van Duyn, *The DP Professional's Guide to Writing Effective Technical Communications*, John Wiley & Sons, New York, 1982.

Werner L. Frank, *Critical Issues in Software*, John Wiley & Sons, New York, 1983.

William M. Fuori, *Introduction to the Computer: The Tool of Business*, Prentice-Hall, Englewood Cliffs, 1981.

Tom Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Wokingham, England, 1988.

Susan J. Grimm, *How to Write Computer Documentation for Users*, Van Nostran Reinhold, New York, 1987.

Mark P. Haselkorn, "The Future of 'Writing' for the Computer Industry," *Text, ConText, and HyperText*, M.I.T. Press, Cambridge, 1988.

Journal of Technical Writing and Communication, 10, no. 4, 1980.

Emanuel Katzin, *How to Write a Really Good User's Manual*, Van Nostran Reinhold, New York, 1985.

David King, *Current Practices in Software Development*, Yourdan Press, New York, 1984.

Henry Ledgard, *Software Engineering Concepts*, Addison-Wesley, Reading, Mass., 1987.

Carol S. Lipson, "Descriptions and Instructions in Medieval Times: Lessons to be Learnt From Geoffrey Chaucer's Scientific Instruction Manual," *Journal of Technical Writing and Communication*, 12:3, 1982.

S.B. Little and M.C. McLaren, "Profile of Technical Writers in San Diego County: A Pilot Study," *Journal of Technical Writing and Communication*, 17, no. 1, 1987.

Kalle Lyytinen, "Expectation Failure Concept and Systems' Analysts' View of Information System Failures: Results of an Exploratory Study," *Information & Management*, 14, no. 1, 1988.

James Martin and Carma McClure, *Diagramming Techniques for Analysts and Programmers* Prentice-Hall, Englewood Cliffs, 1985.

----------, *Structured Techniques: The Basis for CASE*, Prentice-Hall, Englewood Cliffs, 1988.

Carma McClure, *Managing Software Development and Maintenance*, Van Nostrand Reinhold, New York, 1981.

Brian Murphy, *The Computer in Society*, Anthony Blond Ltd., London, 1966.

Leslie A. Olsen, *Principles of Communication for Science and Technology*, McGraw-Hill, New York, 1983.

Robert O. Peterson, *Managing the Systems Development Function*, Van Nostrand Reinhold, New York, 1987.

Ronald G. Ross, *Data Dictionaries and Data Administration*, AMACON, New York, 1981.

Harvey M. Sapolsky, *The Polaris System Development*, Harvard UP, Cambridge, 1972.

Kenneth D. Shere, *Software Engineering and Management*, Prentice-Hall, Englewood Cliffs, 1988.

Henrietta Nickels Shirk, "Technical Writing's Roots in Computer Science," *Journal of Technical Writing and Communication*, 18:4, 1988.

Charles H. Sides, *How to Write Papers and Reports About Computer Technology*, ISI Press, Philadelphia, 1984.

W. Dwain Simpson, *New Techniques in Software Project Management*, John Wiley & Sons, New York, 1987.

Joseph N. Ulman and Jay R. Gould, *Technical Reporting*, Holt, Rinehart, and Winston, New York, 1972.

Herman M. Weisman, *Basic Technical Writing*, Charles E. Merrill, Columbus, 1980.

Edmond H. Weiss, *How To Write A Usable User Manual*, ISI Press, Philadelphia, 1985.

Charles J. Wertz, *The Data Dictionary*, North-Holland, Wellesley, Mass., 1986.

Michael R. Williams, *A History of Computing Technology*, Prentice-Hall, Englewood Cliffs, 1985.

Edward Yourdan, *Managing the System Life Cycle*, Prentice-Hall, Englewood Cliffs, 1988.

APPENDIX

SAMPLE QUESTIONNAIRE AND SURVEY RESULTS

Survey

Your Company or Firm Name_____

Your Name (optional)_____

Your Job Title_____

# of Years in Computer Industry_____

Please circle the appropriate answer(s) to the question, or use the blank line to answer the question.

Questions

1. What percentage of your company's business is related to software design/development?

   A. 0 - 20% (28%)
   B. 20 - 40% (18%)
   C. 40 - 60% (19%)
   D. 60 - 80% (12%)
   E. 80 - 100% (23%)

2. What percentage of your job is related to software design/development?

   A. 0 - 20% (39%)
   B. 20 - 40% (9%)
   C. 40 - 60% (11%)
   D. 60 - 80% (10%)
   E. 80 - 100% (31%)

3. Are technical writers/editors employed by your company?

   A. Yes. - 67%
   B. No. - 33%

4. What are the major tasks for a technical writer/editor at your company?

   A. Writing documents - 52%
   B. Helping others write - 20%
   C. Editing of documents - 71%
   D. Production editing - 37%
   E. Graphics and formatting - 38%
   F. Pseudo-secretarial

5. Are formal documentation techniques used in the development process?

A. Yes. - 67%

B. No. (Proceed to #9) - 33%

6. If formal documentation techniques are used, do you feel them to be effective?

A. Very effective - 32%

B. Effective - 58%

C. Not very effective - 10%

D. Not at all effective - 0%

7. If formal documentation techniques are used, is a technical writer employed in the writing/preparation of these documents?

A. Yes. - 63%

B. No. - 37%

8. If a technical writer is used, at what point does his/her involvement begin?

A. Beginning of project - 40%

B. Start of writing process - 28%

C. End of writing process - 60%

D. Other_____

_____

9. If formal documentation techniques are not used, do you feel that a technical writer on your development team might improve the documentation of your system?

A. Yes, definitely. - 40%

B. Yes, somewhat. - 34%

C. Maybe. - 19%

D. No. - 8%

E. Might make it worse.

10. If a technical writer is not part of your design/development team, what percentage of time does your staff spend on documentation?

A. 0 - 10% (16%)
B. 10 - 20% (25%)
C. 20 - 30% (29%)
D. 30 - 40% (16%)
E. 40 - 50% (2%)
F. 50 - 60% (8%)
G. 60 - 70% (3%)
H. 70 - 80%
I. 80 - 90%
J. 90 - 100% (1%)

11. How would you rate the importance of documentation?

A. Extremely important - 68%
B. Important - 24%
C. Not important - 1%
D. Waste of time - 7%

12. What background do you feel is critical for a technical writer?

A. English/Tech Writing - 85%
B. Communications - 43%
C. Psychology - 7%
D. Philosophy
E. Engineering - 23%
F. Computer Science - 45%
G. Mathematics - 8%
H. Other Sciences - 14%

13. Would you hire a technical writer to work on your software design/development team if he/she had this background?

A. Yes. - 83%
B. No. - 17%

14. Does your company handle mostly government or private accounts? (Please circle the appropriate answer.)

Government - 65%
Private - 20%
Mixed - 15%

15. How large is your software development staff?

A. 1 - 10 - 40%

B. 11 - 20 - 12%

C. 21 - 30 - 11%

D. 31 - 40 - 4%

E. 41 - 50 - 2%

F. 51 or more - 28%

16. How is your staff composed?

# of Tech Writers	15%	# of Programmers	38%
# of Analysts	28%	# of Designers	28%
# of Support Staff	3%	# of Managers	16%

17. Do you prototype your software during your development effort?

A. Yes. - 42%

B. Sometimes. - 50%

C. No. - 5%

D. Not familiar with the term. - 3%

18. Do the technical writers/editors within your company have formal training in the field of technical writing/editing?

A. Yes. - 33%

B. Somewhat. - 30%

C. No. - 9%

D. On-the-job training. - 27%

If you have any comments that you would like to express, please use the remainder of this page.

VITA

Laura Leigh Bresko was born in Brooklyn, New York on July 23, 1961. She attended elementary schools in that city and was graduated from John Dewey High School in June of 1978. In 1981, she entered the University of Tennessee at Knoxville. She majored in English and received her Bachelor of Arts degree in June of 1985. She began study towards a Master's degree shortly thereafter. The Master of Arts degree was awarded to Ms. Bresko in August 1989.

In 1986, she accepted a position with the Energy, Environment, and Resources Center, a research arm of the University of Tennessee. She was assigned to computer software development projects at Oak Ridge National Laboratory and has been working there ever since. She was promoted from technical editor to technical publications manager and is currently working on the Airlift Deployment Analysis System project for the Military Airlift Command, United States Air Force. Ms. Bresko will continue in her current employment.

The author is a member of the Modern Language Association, Society for Technical Communications, and the Association for Computing Machinery. In addition to her professional actvities, she is also active in many community, civic, and artistic organizations.