1985

Reframing the Role of Computers in Organizations The Transaction Costs Approach

Claudio U. Ciborra
Politecnico de Milano, Universita' di Trento

Reframing the Role of Computers in Organizations The Transaction Costs Approach

Claudio U. Ciborra
Politecnico de Milano
Piazza L. Da Vinci, 32
Milano—Italia

Universita' di Trento
Via Verdi, 7
Trento—Italia

ABSTRACT

The traditional role of computer-based information systems is to provide support for individual decision making. According to this model, information is to be seen as a valuable resource for the decision maker faced with a complex task. Such a view of information systems in organizations does however fail to include such phenomena as the daily use of information for misrepresentation purposes. The conventional systems analysis methods; whether they be data- or decision-oriented, do not help in understanding the nature of organizations and their ways of processing information. This paper proposes what appears to be a more realistic approach to the analysis and design of information systems. Organizations are seen as networks of contracts which govern exchange transactions between members having only partially overlapping goals. Conflict of interests is explicitly admitted to be a factor affecting information and exchange costs. Information technology is seen as a means to streamline exchange transactions, thus enabling economic organizations to operate more efficiently. Examples are given of MIS, data base and office automation systems, where both the organization and its information system were jointly designed. These examples illustrate the power of the approach, which is based on recent research in the new institutional economics.

Introduction

Despite the confidence of many in the relentless advancement of computer-based information systems, as witnessed by such innovations as Decision Support Systems, Distributed Data Bases, Expert Systems, Office Automation annd suchlike, critics object that "Management Information Systems above the level of simple counting and comparing fail because theory is missing to make them work" (Wildavsky, 1983).

This is a recent example of skeptical comments on how computers are misapplied in organizations; the earliest criticisms stem from almost twenty years ago with scholars such as Ackoff (1967), who stated: "I believe that these near-and far misses in MIS implementation could have been avoided if certain false (and usually implicit) assumptions on which many such systems have been erected had not been made".

Even among the users, i.e. those who actually operate in organizations and deal daily with the small and large scale problems arising from the introduction of computers (conflicts, hidden resistance, lack of integration, skyrocketing development cost, education problems, to name a few), there is a growing awareness that theory and practice are still at the mercy of events.

It would however be unfair to suggest that scholars and practitioners just muddle through when analysing and designing systems. On the contrary, the current state of the art is dominated by a conventional wisdom, which is composed of a comparatively longstanding set of assumptions and frames which seem to guide the practical theories and actions of designers. It is this conventional wisdom which must be explored: its concepts, views and stereotypes must be critically examined and reframed, in order to improve our understanding of such basic issues as:

- why and how is information processed and communicated within and between organizations?
- what impact does information technology have on organizational processes and structures?

- what organizational models can guarantee that systems analysis and design are sound and effective?

Present day designers turn to two theories when addressing the above issues: they either tend to a "data view" of organizations, or, in the case of those most influenced by business needs, to a decision-making view. These two ways of looking at the problems of computerization are so widely accepted and have been so much taken for granted that they can be said to form the conventional wisdom of today. The origins of the former can be traced directly back to the EDP field, while the latter stem from the influential work of Herbert A. Simon (1976a).

It is somewhat surprising that although information technology has gone through an almost revolutionary process of miniaturization, sophistication and diffusion, the design models and criteria concerning its application in organizations are still based on the concepts of the early sixties. This appears still more puzzling when we examine the fields of sociology, political science, organization theory, economics of information and organizations, which have also undergone a sharp innovative process. But none of the new developments in these disciplines seems to have filtered through to the field of MIS, apart from such aspects, as the political view of system development (Keen, 1981; Kling, 1980; Markus, 1983).

The aim of this paper is to open the MIS disciplines to recent developments in social sciences and economics. The ultimate goal is to define both a new framework and a new language, so that the role of information technology in organizations can be better understood. To anticipate it is argued that a new *organizational* understanding of information processing must go beyond the individual decision-making paradigm, which at present lies at the core of the conventional wisdom. The concepts of "exchange" (transaction)[1] and "contract" between at least *two* individuals or organizational units must become the new center of attention. This alternative tact enables us to use the results of a new paradigm emerging both in institutional economics and the sociology of organizations. The paradigm, known as the transaction costs approach (Williamson, 1975, 1981), links the notions of information, uncertainty and organization in an original way. Phenomena such as resistance to change, retention of information are not seen anymore as irrational, unexpected flaws in a structured system design, but as factors and behaviors which can be rationally understood and carefully anticipated; and issues such as centralization versus decentralization can be viewed in a different light. The presentation of the argument starts in Section 2 with a critique of the received tradition and its implicit, but widespread assumptions: it is shown that the data- and decision-making views are inadequate and irrealistic, because they are based on a view of organizations as perfectly cooperative systems. The need for an alternative framework based on the new institutional economics is addressed in Section 3: it is shown that by considering organizations as networks of exchanges and contracts between members, both cooperation and conflict can be taken into account together with the various usages of information that individuals employ when cooperating and conflicting. Also, the specific role of information technology is illustrated as a means to lower transaction costs. In Section 4 the new design principles are discussed using examples drawn from the fields of data bases, office automation and MIS. The concluding remarks concern the further research paths opened up by the new framework.

A Critique of the Conventional Wisdom

Two Current Views

In order to reframe our understanding of computer-based information systems in organizations, an essential, preliminary step is to discuss two approaches which are at present in good currency: the *data approach* and the *decision approach*. According to the data approach, in applying a computer to an organization it is only necessary to consider (i.e. analyze and design) the data flows and files in that organization. The analyst ascertains management information requirements by examining all reports, files and other information sources currently used by managers. The set of data thus obtained is considered to be the information which management needs to computerize (Davis, Munro, 1977). The data approach ignores the economic and social nature or organizations and is exposed to the hazards of those economic and social processes which characterize the daily life of organizations and which we, as members of organizations, all know (see below).

The second tradition is more sophisticated from an organizational point of view. It can be traced back to Simon (1977) and was further developed by scholars such as Galbraith (1977), Keen and Scott Morton (1978) etc. According to this approach information technology is support to *decision making*. Managers facing complex tasks and environments use information in order to reduce the uncertainty associated with decision making: "the greater the task uncertainty, the greater the amount of information that must be processed among decision makers during task execution in order to achieve a given level of performance," states Galbraith (1977). Simon writes in a similar vein about programmable and unprogrammable decisions (Simon, 1977) (see also for applications Ackoff (1967); Keen, Scott Morton (1978); Sprague (1980); Pava (1982); Huber (1984)).

It could be argued that the diffusion of communications and data processing technology poses some limits to the scope of the decision-making view, which emphasizes control and feedback rather than communication processes. But, of greater interest here are some puzzling organizational phenomena which challenge that view and invite the suspicion that it is incomplete. Consider the following evidence by scholars in the field of organizations:

- information is gathered and taken into account only after the decision has been already made, that is to say, as an *a posteriori* rationalization (many computer print-outs are used as high-tech cosmetics to already made resolutions),
- much of the information gathered in response to requests is not considered in the making of those decisions for which the information is requested (Feldman, March, 1981),
- most of the information generated and processed in organizations is open to misrepresentation, since it is gathered and communicated in a context where the various interests conflict,
- when, on the other hand, organizations are informationally transparent, as many DP specialists wish, it has been shown that the decision makers in two different departments, say Production and Sales, could be playing never-ending information games which lead to overall suboptimality (Ackoff, 1967),
- information is not only used as an input for the individual decision maker, but is also used to persuade and induce the receiver to action. It could indeed be argued that this use of communication is the essence of authority and management (Flores, Ludlow, 1981).

Thus information is not simply interpreted data, rather it is an argument to convince other decision makers to be effective it must have attributes other than exactness, clarity, etc.: rather than being purely objective, it must be convincing and adequate to the situation at hand.

Flaws in the Decision Making View

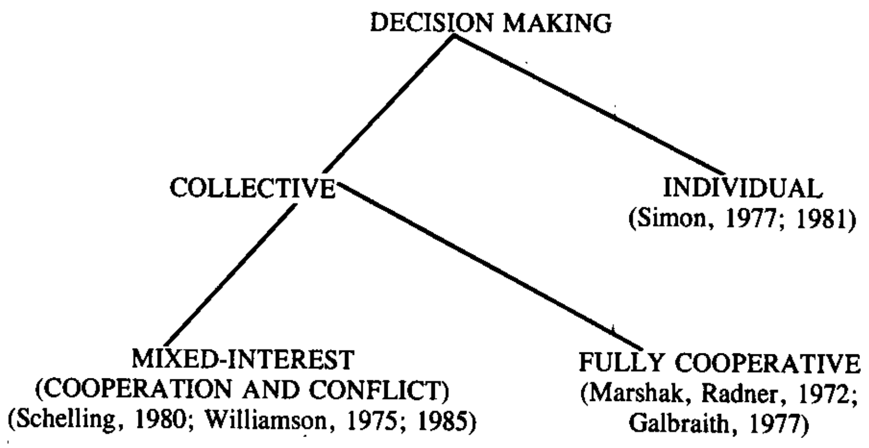
We now turn to an analysis of the reasons why the conventional decision-making view cannot explain phenomena such as those just described:

- Firstly, the decision-making approach tends to be *individualistic*. Decision-oriented design strategies focus on the information needs and cognitive styles of the individual decision maker facing a complex and uncertain task. Take, for example, Rockart's design method based on the analysis of the Critical Success Factors, which stresses 'the investigation of current information needs of *individual* managers'' (Rockart, 1979). While it is worth investigating the role that computers play in individual problem solving, a manager in a particular organization cannot be seen as a solo chess player whose only opponents are the ''technology,'' a ''random environment'' or ''nature''. In organizations the key issue if *collective, coordinative* problem solving (Schelling, 1980) (Turoff, Hiltz, 1982). Though this obvious consideration is beginning to make its way in the recent DDS literature (Huber, 1981), few practical suggestions are provided regarding its implications in systems analysis and design (Sprague, 1980; De Sanctis, Gallupe, 1985).
- Secondly, the decision-making control model ignores the fact that organizations are mixtures of cooperation and conflict between participants; its implicit reference is in fact to man-machine systems (Simon, 1977). When dealing with collective problem solving, the model assumes that all the participants share common goals (i.e., a team, Marschak, Radner 1972): information problems related to task execution and coordination are once again considered to be caused by environmental or technological uncertainty only. It is, however, more realistic to say that all coordinative problem solving and the relevant information processing take place in a *mixed-interest* context (Figure 1). A minimal respect for the well-known conflictual processes existing in organizations would indicate that there are other incentives to gather and use information, apart from task uncertainty: information can be misrepresented; promises and committments can be false; data incomplete; tracks covered etc., all in order to induce others to make decisions most benefiting us in the first place. Or, another possibility is that information can be selectively disclosed to persuade and bias; what this in fact means is that it can be used as an instrument of power to win or gain a better position in the daily organizational games.

The upshot is that in collective coordination and action there is a distinct form of uncertainty besides that characterizing the task, the technology or the environment: it is an uncertainty of behavioral, strategic nature, which has its origins in the conflict of interests between organiza-

Figure 1

Types of decision-making models



tion members. The information which the decision maker receives or gathers both within and outside the organization, may well be "unreliable" with the result that he has to perform a surplus of information processing in order to evaluate its reliability. The fact that it is obtained from human sources means that it cannot be trusted *a priori*. It can therefore be stated that in an organization at least half of the on-going information processing is dedicated to the solution of tasks and problems by cooperative means, while the other half is concerned with solving problems of cooperation among members who behave opportunistically.

To analyze information requirements and design a system without considering the inevitable opportunistic information processing which takes place in organizations appears to be risky. System implementation can lead to conflict, resistance, and other negative attitudes which, far from being irrational, represent the members' response to the attempt of changing the way of producing and using information in a mixed-interest organizational setting (Markus, 1983).

- Thirdly, the conventional wisdom is *one-dimensional*: it takes hierarchical organizations for granted, thus ignoring many important facets of the economics of organizing. For example, it must be remembered that the boundary and structure of an organization are not indefinitely fixed: they change every time a manager implements a make-or-buy decision, or he/she decides to integrate or disintegrate a stage of the production process, an office or a department. Moreover, it is insufficient to consider large pyramidal corporations only, since regional networks of small firms, which are even more diffuse, operate in a manner more like a peer group, or family, than a formal bureaucracy (Piore, Sabel, 1984). And even within large corporations changes all take place at shop floor level, where work groups are being introduced at the expense of formal hierarchies. All these developments, which stem from the effort of organizations to respond to the turbulence of the environment, challenge the approach which identifies management and information systems with hierarchies (Simon, 1981; Arrow, 1974). It is in fact time to acknowledge that many systems, including airline reservation, EFT, remote office work, etc. have little to do with the workings of organizations conceived as pyramids of strategic, managerial and operational control systems. They must rather be seen as exchange or market support systems, in that they support market transactions and not procedures of a hierarchy.

- Finally, even recent amendments to the conventional wisdom leave contradictions unresolved. Consider the introduction of computers in organizations. At present this process tends to be regarded as a bargaining process between conflicting parties, the decision-making taking place during system implementation is looked at from a political perspective (Keen, 1981; Markus, 1983). However, even these very authors, when considering a specific managerial decision for automation, (for example a DSS for budgeting) switch the analysis framework back to the conventional wisdom: the decision maker is seen as a component of a control system, where the system is uncertain and complex, and factual information is needed to keep it under control (Keen, Scott Morton, 1978). How can one agree with such a contradictory treatment of two organizational processes, the implementation of a system and the use of information for managerial decision making? If the former is a bundle of political decisions, why should the latter represent a neutral, purely algorithmic exception? The political view of system implementation has had the merit of breaking the ice and showing that, in certain areas at least, organizations cannot be analyzed and changed by using frameworks exclusively derived from systems theory and computer science, but what they in fact require are investigation and design methods which consider political, economic and sociological phenomena. This particular point of view has not however succeded in providing a complete and coherent reframing of the entire field of MIS.

A Transactional View of Organizations and Their Information Systems

Economic Organizations: Markets, Hierarchies and Groups

It is a tenet of this paper that the processes involved in socio-economic organizations cannot be analyzed correctly unless formal systems analysis methodologies, such as HIPO, BSP, SADT, or other structured analysis techniques (see Couger *et al*, 1982), are grounded on an understanding of the nature of organizations and of the way computers can be fitted to support their effectiveness. If we take the field of economic organizations, which is to say firms operating on a market, I argue that the classic answer provided by Coase (1937) to the fundamental question "Why are there firms and markets?" is of the greatest interest to the whole field of MIS.

A *market* is an assemblage of persons desirous of exchanging property, with prices serving both as incentives and coordinating guides to producers in so far as they affect what and how much is produced and demanded. At an equilibrium free-market price the amount produced equals the amount demanded—*with no necessity for a central all-knowing authority*. Individual self interest, an incentive to obtain greater gains together with lower costs, is what permits resources to be efficiently allocated. Note that the market system requires very *little knowledge* of the participants, i.e. their own needs and the prices (Alchian, Allen 1977). The same problems of economic organization, i.e. the control and coordination of diverse, specialized activities, is solved differently in a *hierarchy* or the firm. In a firm, market transactions are eliminated and in their place we find an entrepreneur-coordinator who is *the authority* who directs production (Coase, 1937). Markets and firms are thus substitutes and the replacement of one by the other is a common event. Think again of any make-or-buy decision. A market contract displaces a bureaucratic contract when a travel agency replaces its ticket delivery person with a messenger service. A hierarchy supplants a market when a firm begins photocopying its own circulars rather than paying for the services of a printer (Hess, 1983).

Given the case with which an economic system, with its essential functions of coordination and control, can flow from market to hierarchical organization and back, it should be clear that there is a need for a framework for defining the special role of computer-based information systems in such a diverse organizational context. If systems do in fact support organizational control and coordination mechanisms, what mechanism should they specifically support, the price or the authority relation? In what circumstances should they switch from one to the other, and what criteria are there to tell whether systems are supporting the "right," i.e. more efficient mechanism? A temptative answer to these questions is the following (Williamson, 1975):

- when transactions are fairly well patterned, the services or products to be exchanged are fairly standardized and all participants possess the relevant information, i.e. the price, then the perfect *market* is the most efficient resource-saving way of organizing the division of labor with each person producing a service or product and selling it on a market, where he/she can also buy the necessary inputs: the "invisible hand" (Smith, 1976) coordinates the individual decisions of producing, buying and selling among a large number of independent agents.

- in some contingencies, however, the use of the price mechanism involves costs, prices must be discussed, transactions encounter difficulties due to the complex search for partners; the contract model specifying the terms of exchange is difficult to develop and it is costly to control *ex post* the execution of the contract. In these contingencies the product/service exchanged is complex and the transaction uncertain due to a conflict of inter-completion of the transaction. Thus it can be better, or rather more efficient to avail of organizing agents within the *firm* to mediate economic transactions, rather then to trust entirely to the market mechanism. In this case the "invisible hand" of the market is replaced by the "visible hand" of management (Chandler, 1977)[2].

- Finally, there are situations where coordination can neither take place through a market nor through a hierarchically organized firm: products and services are so complex, transactions so ambiguous that the parties involved in the exchanges have to trust each other and give up any attempt at a short-sighted calculation of the reciprocal costs and benefits accruing from the exchange. The "invisible" and "visible" hands are replaced by the "invisible handshaking" (Okun, 1981). The organizational arrangement whereby networks of exchanges are governed in a stable manner by informal relationships of trust, has been called a *group* or *clan* (Ouchim 1980).

Remember that, in general, the obstacles to transacting, justifying the use of the three alternative arrangements, stem from two distinct sources: one is *natural uncertainty* (the product/service is complex and unique, difficult to evaluate and price; there are barriers to communication during the exchange, etc.); the other is *behavioral* or *strategic uncertainty*, which originates out of the joint effect of informational asymmetries and lack of trust between the parties. To sum up, if the world was certain to evolve according to one pattern only, the coordination of activities could easily be streamlined. If people could fully agree, cooperation would be smoothly achieved even in an uncertain and complex world. But when uncertainty, complexity, information asymmetries and lack of trust cannot be ruled out *a priori*, then the multitude of contingencies which affect work in organizations may require the negotiation of complicated contractual plans to arrange cooperation. Depending upon the degrees of ambiguity in the service or product object of exchange and the goal congruence among the parties, the three arrangements: the market, the hierarchical firm and the clan or group, are the most efficient organizational mechanisms for solving the fundamental problems of organizing.

Information Systems

Galbraith's (1977) hypothesis can be now enlarged, if it is to totally comprehend what goes on in organizations: The more complex cooperation and bargaining are, not only because of the uncertainty of the product/service to be produced and exchanged, but also because of the hazards of opportunism, the more difficult it is to achieve a contract to regulate cooperation and exchange, and the more information has to be processed in order to set up and maintain the organizational relationships between contracting members.

Having thus linked the notion of information within organizations to those of uncertainty and opportunism (lack of trust among cooperators), we are now able to reframe the concept of "information system". If we look at organizations as networks of exchanges, information systems, whether they be computer-based or not, are made up of the networks of information flows and files needed to create, set up, control and maintain the organization's network of exchanges and relevant contracts. Obviously, an information system will prove contingent upon the nature of the organization to which it belongs. In a *perfect market* where coordination and control are achieved through the price mechanism and spot contracting, the information system is highly standardized, formalized, a-procedural, responsive and extremely simple: the *price* is the only input needed to support members of the market make basic decisions, such as buying or selling.

In the *hierchical firm*, or *bureaucracy*, where open, longer term contracts regulate the exchange of products and services through the employment relation and the authority relationship (Simon, 1957), the information system is represented by the *rules*, norms and plans which convey, mostly in a procedural fashion (Simon, 1976b), the information concerning what should be done under what circumstances, and how it should be controlled. Finally business, in a *clan*, is carried out by parole contract, and partners bind themselves by word or handshake to a complex web of mutual, stable and long term obligations. Its information system consists of the *rituals*, stories and ceremonies which convey the values and beliefs of the organization. It is highly informal and idiosyncratic: an outsider cannot gain quick access to the decision rules of a clan; on the other hand its information system, which is anything but transparent, has no need for an army of accountants, computer experts and managers: it is just there as a by-product of well-knit social relations (Ouchi, 1979; Wilkins, Ouchi 1983; Schein 1984).

It goes without saying that real organizations include a mixture of the three coordination and control mechanisms outlined thus far, and consequently they avail of a variety of information systems. It is however possible to distinguish the prevailing one locally. For example, in a multi-divisional company, one can identify an overall bureaucratic, hierarchical structure which links through authority relations the various divisions with the central office, and a corresponding information system, say for budgeting, planning and control. Internal markets regulate the exchange of products and services between the divisions, the relative computer-based information system is a data base containing all the transfer prices. Finally, both within the division departments and in the central office, clans exist among managers, among workers in production work groups, among the employees of an office, with each subculture having its own peculiar jargon, set of symbols, rituals, etc. Present MIS theory has focused on bureaucratic organization to the exclusion of all else. By considering the plurality of organizations and information systems we should however realize that there are *multiple strategies for computerization*, all of which are contingent upon the nature of information processing taking place in a specific organization or part of it. Our framework, then, enables us to overcome the problem of one-dimensionality discussed above.

The Role of Information Technology

If organizations are seen as networks of exchanges,then the organizational use of information technology concerns not only "data" or "individual decision making" but also *interdependent* decision making and communication related to exchanging. Information technology belongs to those technologies, like the telephone and money itself, which reduce the cost of organizing by making exchanges more efficient: it is thus a *mediating technology*, i.e. a technology which links several individuals through the standardization and extension of the linkages (Thompson, 1967). The costs of organizing, i.e. costs of coordination and control, are decreased by information technology which can streamline all or part of the information processing required in carrying out an exchange: information to search for partners, to develop a contract, to control the behavior of the parties during contract execution and so on. The functions of a computer-based information system can thus be reframed as an "exchange support system". And in analogy to Simon's typology of decision making (Simon 1977), a classification of exchanges and the contracts regulating them can be developed.

- *Structured contracts*, i.e. spot contracts which govern transactions such as those occuring in an ideal market.
- *Semistructured contracts*, i.e. longer term, open contracts, such as the employment relation, where adaptation, sequential modifications at low renegotiation costs are permissible.
- *Unstructured contracts*, related to those exchanges which cannot be modelled or "written down" in an explicit contract form, either because communication between the parties is difficult or because they cannot be satisfactorily spelled out and formalized.

Data processing can support all these types of exchanges and related contracts. Consider first the *structured* contracts. Many of the structured market exchanges have already been automated, from airline reservation systems to EFT in banking, to data banks selling pieces of information. Note that the recasting of data processing as a mediating technology indicates that information technology is a means for creating/expanding markets, by lowering search, contracting and control costs. It would be interesting to carry out a census of the running DP applications in the commercial sector today: it is the author's conviction that market transactions rather than bureaucratic firms are at present the main field of application of DP technology, since the structured and standardized nature of those transactions make them more suitable to automation[3]. In what way then can computers support *semistructured* and *unstructured* exchanges? Systems can be dispatchers of heuristicism committments and promises which streamline the negotiation process embedded in any exchange. And this can take place not only on markets. The organization of work, in an office for example, can be seen in terms of coordinative problem solving which is achieved by the exchange, storage, control and retrieval of committments between the various employees working in that office. In particular, the computer system together with the local network, could enable the parties required for the execution of a given job to be identified, their mutual interests communicated, their previous/pending committments recorded and their discretion noted. In this way a personal and collective agenda is built up (Flores and Ludlow, 1981; Fikes and Henderson, 1980), which could support office work conceived as a complex group problem solving (Suchman, Wynn, 1984). Operating systems, such as UNIX, through commands such as "make," and other OA facilities act as a mediating technology which supports software development performed by various programmers linked to the system (Ritchie, Thompson, 1974). For other applications see Lee (1980); Turoff, Hiltz (1982); Jarke, Jelassi and Shakun (1985).

Strategies for the Joint Design of the Organization and its Information Systems

In general, design is concerned with adapting a system to its surrounding environment (Simon, 1981): it is a question of "fit" between the two (Alexander, 1967). In our case, it deals with adapting a computer-based information system to the existing organization with the latter constituting the environment of the former. But this is true only as a first approximation, when the organization can be held constant and the system varied. According to the framework proposed here, systems can also streamline exchanges by altering the contractual arrangements which build up the organization. Suppose, for example, that a hierarchy, based on the authority relationship is necessary to overcome the information barriers hindering the negotiation of the employment relation through the market. A new information processing utility which eliminates those barriers would make the hierarchy itself both inefficient and superflows.

There are then many possible interactions between information technology and organization, and the transaction costs framework indicates them in a clearer way then ever before: it also shows specific strategies for the *joint design* of the organization *and* its information systems, i.e. for the best match between alternative computer systems and organizational forms (Ciborra, 1981). As a first operationalization of such strategicism consider the following applications.[4] To begin with, consider the information systems models which have been put forward in connection with data bases (Emery, 1969; Chen, 1977). A very attractive alternative offered by these systems is that the hierarchical coordination channels of the firm are superimposed by an information system linking each task to a common information pool. This "common data base" offers economy in information channels, closer coupling between activities, tighter coordination between decision makers and a common view of the enterprise.

In real applications, however, this technical approach does not appear to work. Empirical surveys have shown that in organizations using data bases, data is far from being shared in a common scheme. "Political" problems impeding the centralized standardization and storage of data are reported by the majority of data administrators interviewed (Davenport, 1979). "Resistance" to standards, data "ownershipness" and other such phenomena are usually associated with the psychology of the recalcitrant user and indicate that the implementation of the common data base is at variance with hierarchical organization (Sibley, 1977). Other signals seem to confirm this conclusion. Consider first some technical evidence. The growing interest in distributed data bases having a more complex architecture than the centralized version, can be interpreted as a failure of the latter and a need to accomodate data base management systems to the idiosyncrasies of the extant, departmentalized organization[5]. Concepts such as "site autonomy" of a local data base in respect to the global one, or as heterogeneous, multi-data bases, where different data models coexist illustrate further the point (Ceri, Pelagatti 1984).

That such idiosyncrasies play an important role in determining the system architecture is supported by the experience of leading companies who have pushed data base integration quite far: "Although a homogeneous architecture is attractive at first, writes Beeby (1983), previously at Boeing Commercial Airplane Co.'s Engineering Division, it is less attractive over the long run. Factors that argue for a heterogeneous implementation are: diverse applications that impose diverse requirements on data management[6]; need to exchange product data with industry partners and subcontractors who employ different hardware. In the *next* generation system at Boeing, a homogeneous solution will be pursued whenever practical, but the advantages—and frequently the necessity—of heterogenous implementation will not be ignored." And beyond the supposedly technical reasons, experience suggests that important organizational issues are involved. By using the transaction costs framework, flaws in data base implementation can be understood, and even anticipated. Although, the original idea of scholars like Emery, Chen and others, is technically sound, it contains a hidden organizational dilemma. Let a common data base be available to manage the data of a whole enterprise efficiently. Each departmental manager could, in principle, access the overall system schema to retrieve the relevant data for his/her decision making. Moreover, the output of each decision taken in any organizational unit would be likewise made generally available through the database. Now, if this were all possible, the enterprise would not have any reason to exist according to the transaction costs view: its dissolution would be warranted on efficiency grounds (reduction of overhead costs). The single units or individuals would transact by exchanging services and intermediate products with each other through market relationships with the information provided by the common data base becoming the main coordination mechanism.

To conclude, the hierarchy exists because uncertainty and opportunism make market exchanges too costly to negotiate, execute and monitor. It could happen that the common data base is able to standardize the specialized pockets of knowledge scattered throughout the hierarchic organization, thus eliminating both existing information barriers and departmental idiosyncrasies. In this way uncertainty and opportunism play no role, but neither is there a need to use a hierarchy instead of a market as a more efficient control and coordination mechanism. But there also exists a possibility that the whole idea of a common database is doomed to failure because it clashes with the nature of managing hierarchical, departmentalized organizations. Experience seems to show that the latter conclusion is nearer to the truth and the reasons are explained by the transaction costs framework. As a second example, consider the award-winning computer-based information system employed by Benetton, the leading Italian company in fashion knitwear.

At a first glance, the system architecture seems to violate the iron law of the layered, pyramidal MIS concept. The network of production plants, design bureaus, subcontractors, warehouses, points of sale (many hundreds scattered all over the world), which build up Benetton's loosely-coupled organization (Weick, 1976), is held together by a DP network, whose aim to decrease transaction costs between the various units, and between them and the market. Data links have been established between the central office in Treviso, Italy and cash registers in the shops: these links enable production and reorder plans to swiftly adapt to market vagaries, by shortening the time lag between customers' needs, as expressed in purchase transactions, and the company's adaptive response. CAD systems decrease the time lag between the design of models and their production. Data bases support the quasi-market relationships between the company and its subcontractors, and so on. Thus, the system seems to fit the nature of Benetton's organization, because it maintains and strengthens its flexibility. It streamlines crucial transaction costs instead of superimposing a rigid, pyramidal system configuration. In this case too, the role of the computer-based information system in supporting Benetton's organizational effectiveness cannot be explained in terms of the misleadingly narrow ideas of the conventional wisdom.

A final example comes from public administration. Consider the organizational rearrangement, suggested by Strassman (1980), of a large public bureaucracy oriented to providing complex services to customers. Figure 2 shows the functional administrative units (1,2,3. . .) organized on a specialized basis and integrated by coordinating units (I1,I2, . . .) to deal with customers (A,B,C . . .). The cost of organizing the whole administrative structure depends on the size and complexity of the coordinating and controlling mechanism. The cost of producing a single service is obviously correlated to these factors, because of the services requested by units. Given the size of the whole administration, the central office usually finds it difficult to understand delays or mistakes in the service delivered: costs due to control loss effect both the efficiency of the internal organization and the provision of a service to the client. Again an application of the transaction costs framework enables us to identify an architecture whereby the handling of information is linked to a rearrangement of the organization. Finally, in order to achieve greater organizational effectiveness and higher efficiency, markets can be introduced into the structure to simplify transactions.

Strassman suggests creating *information middlemen* between the customer and the bureaucracy, who can package the information products which they buy from the administration and sell in response to customers' needs (Figure 3). This arrangement decreases the information load necessary to centrally coordinate the internal administrative workload. It is the middlemen who selectively access the administrative functions on the basis of customers' requests. Each administrative unit provides the middlemen with a discrete and standardized product, so that it is easier for the central office to monitor the function's performance. Information handling is reduced for the customer too: he/she now faces the single middleman and not the complicated and geographically dispersed bureaucracy. Secondly, data processing applied as a mediating technology can further decrease transactions costs. By using personal workstations, the middlemen can access the specialized functions of the bureaucracy via a communication network. The network becomes the means for the customer, via the middleman, to aggregate and coordinate the various tasks required in generating the complex service he/she needs.

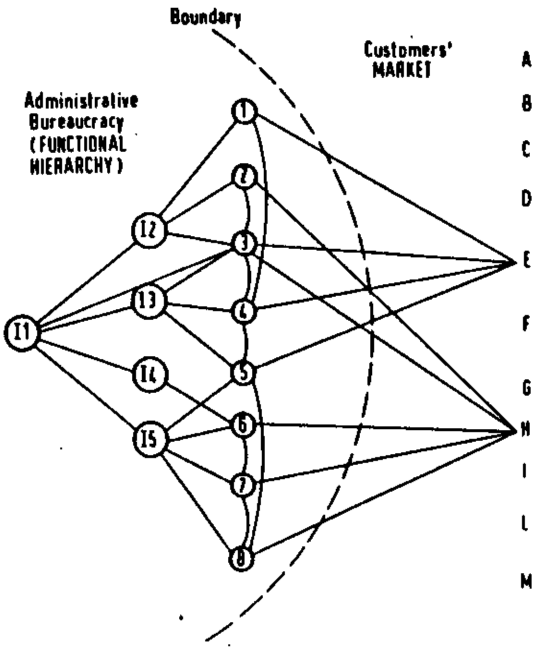


Figure 2

Administrative and market transactions between bureaucracy and customers

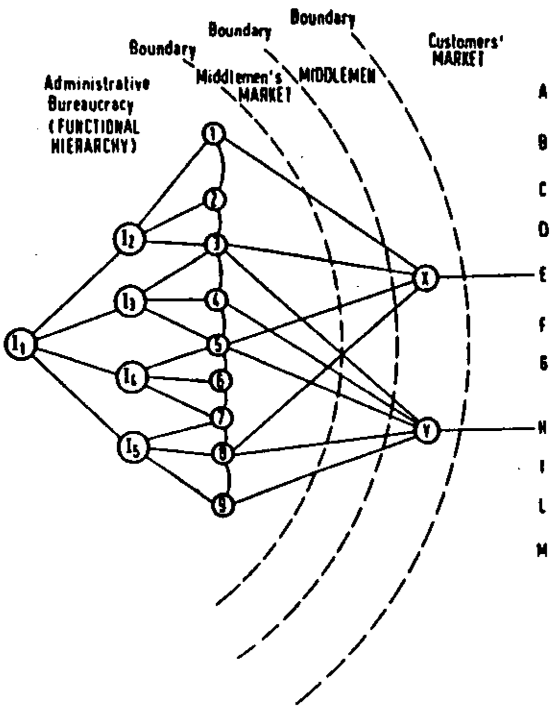


Figure 3

Modified pattern of transactions and boundaries among bureaucracy, middlemen, and customers

By redefining the boundaries between the bureaucracy, the middlemen's markets and the customers, data base technology can also immediately find a wider domain of application than in the previous hierarchic-functional arrangement where interfunctional barriers are difficult to overcome (see above). It could in fact quite easily support market transactions between the bureaucracy and the middlemen. Not only organizational imagination is needed in identifying such changes, as Strassman indicates, but also new intelligence and a new language, so that change is justified according to efficiency criteria and organizational dilemmas are avoided.

Concluding Remarks

An understanding of the nature of economic organizations is an essential prerequisite not only in governing the development of computer-based information systems, but also in analyzing and designing them in an effective way. The transaction costs perspective can help design information systems appropriate to the functioning of institutions such as markets, bureaucracies and groups. The foregoing analysis can be summarized as follows:

- *Exchange transactions* represent the fundamental organizational relationships between human agents;
- The organization of exchange transactions depends upon contingencies which are both *environmental* (uncertainty and complexity) and *behavioral* (bounded rationality and opportunism);
- Organizations can be regarded as *stable networks of contracts* which govern transactions enabling coordination and control;
- Transacting requires *information processing* to identify partners, define a contract, control its enforcement, etc;
- Information technology acting as a *mediating technology* can, by lowering transaction costs, improve information handling needed in transacting;
- The application of information technology should not contradict *the nature of the organizational transactions* supported;
- Information technology can, in the interests of efficiency, influence the shift from one organizational form to another. The possibility of lowering transaction costs should be considered in any attempt at *joint design*.

Note that this new framework does not render obsolete the standard systems analysis methods, be they data- or decision-making oriented. On the contrary, it augments them with a new organizational and economic background, so that when an analyst goes into an organization with his/her toolbox, he/she has a theory with which to select the relevant organizational phenomena, identify the information requirements and make a forecast of the organizational implications of any redesign put forward.

Obviously, the hypotheses and principles outlined invite further reflection and research *per se*. It might be a good idea to direct *requirements and systems analysis methods* to the structured investigation of key organizational exchanges, the contracts dedicated to their governance, the information processing resources deployed to create, control, change and maintain contracts. Software might be designed to explicitly decrease costs of transacting, i.e. to provide search, contracting and control routines. In formulating an *MIS strategic plan*, the market-hierarchies paradigm, (Williamson, 1975, 1985) could provide new insight regarding the dynamic links between business policy, organization structures and information systems.

Finally, empirical research on *the impact of computers in organizations* could be used to test the validity of the framework and the hypotheses generated by its application (Ciborra, 1983). A research project of this kind could also help define more precisely the ways whereby a fit could be obtained between organizational structures and processes, information systems and transaction requirements.

NOTES

[1]The terms transaction and transaction costs are used here in an economic and organizational sense. Traditionally, for the dp profession, transactions refer to the computer operations triggered by a user message and satisfied by the corresponding computer response, i.e. an *exchange of data* between parts of the machine, and between the machine and the user (Bucci, Streeter, 1979). Economic transactions refer instead to the transfer of a good or service between individuals, departments of organizations (Williamson, 1981). It is a *social relationship* which results where "parties in the course of their interactions systematically try to assume that the value gained for them is greater or equal to the value lost" (Barth, 1981). However, it should not be excluded that the two concepts might be linked in the case of computer-mediated economic transactions (Ciborra, 1891).

[2]More precisely, the firm supersedes the market when the service "labor" is the object of exchange: spot contracts regarding the use of labor are in fact exposed to various hazards, since it is difficult to fully specify in advance the

precise, services required during the execution of a complex/uncertain task; and to control the real effort provided by the worker, especially in the case of teamwork (Alchian, Demsetz, 1972). Under the contingencies determined by environmental and human factors (uncertainty and opportunism, respectively), market contracts are replaced, for the sake of efficiency, by a longer term, open contract, *the employment relation*, whereby the worker accepts, by a longer term, open contract, *the employment relation* whereby the worker accepts, within certain limits (the indifference zone (Barnard, 1938)), that someone, the authority specifies in a procedural way what should be done during the unfolding of events.

[3]This conviction is easily justified: in most advanced countries, banks are the main users of data processing. But it is not so much their internal bureaucracy that is automated, it is rather their function as intermediary agents on market transactions (Switzerland has the highest percentage of computers per inhabitant . . .). Consequently, applications in the credit sector should be looked on as market transactions support systems.

[4]Only market and bureaucracy support systems will be considered in the following.

[5]A distributed database is a collection of data distributed over different computers in a network. Each node has autonomous processing capability and can perform local applications, besides global ones. "The organizational and economic motivations are probably the most important reason for developing distributed databases" (Ceri, Pelagatti, 1984). The transaction costs framework justifies this appreciation.

[6]For example, in some cases a relational data management may be required, i.e. in an engineering data base, where queries are unpredictable. A hierarchical system is more efficient when the nature of the queries is known in advance, as in a data base for cost accounting (Beeby, 1983).

ACKNOWLEDGMENT

Many thanks to P. Boegh Andersen, S. Ceri, M. Colombetti, G.B. Davis, A. Merten, B. Swanson for their useful suggestions.

BIBLIOGRAPHY

Ackoff, R.L., Management Misinformation Systems, *Management Science*, 14, 4, December, 1967, pp. 147-156.

Alchian, A.A., Allen, W.R., *Exchange and Production: Competition, Coordination and Control* (2nd ed.), Blaton, Wadsworth Pub. Co., 1977.

Alchian, A.A. and Demsetz, H. Production, Information Costs and Organization, *American Economic Review*, 62, 5, December, 1972, pp. 777-795.

Alexander, C. *Notes on the Synthesis of Form*, Cambridge, Mass.: Harvard University Press, 1967.

Arrow, K.L. *The Limits of Organization*, New York: W.W. Norton & Company, 1974.

Barnard, C.I., *The Functions of the Executive*, Cambridge, Mass.: Harvard University Press, 1938.

Barth, F. *Process and Form in Social Life*, London: Routledge & Kegan Paul, 1981.

Beeby, W.D. The Heart of Integration: a Sound Data Base, *IEEE Spectrum*, 20, 5, May, 1983, pp. 44-48.

Bucci, G, Streeter, D.N., A Methodology for the Design of Distributed Information Systems, *Communications of the ACM*, 22, 1979: 233-245.

Ceri, S., Pelagatti, G., *Distributed Databases-Principles & Systems*, New York: McGraw-Hill Book Co., 1984.

Chandler, D.A. Jr. *The Visible Hand-The Managerial Revolution in American Business*, Cambridge Mass.: Harvard University Press, 1977.

Chen, P.P. The Entity Relationship Model—A Basis for the Enterprise View of Data, *AFIPS Conf. Proc.*, 46, 1977 pp. 77-84.

Ciborra, C.U., Information Systems and Transactions Architecture, *Inter. Journal of Policy Analysis and Information Systems*, 5, 4, December, 1981, pp. 305-324.

Ciborra, C.U., Markets Bureaucracies and Groups in the Information Society, *Journal of Information Economics and Policy*, 21, December, 1983, pp. 145-160.

Coase, R., The Nature of the Firm, *Economica*, November, 1937, pp. 387-405.

Couger, J.D., Colter, M.A., Knapp, R.W., *Advanced System Development/Feasibility Techniques*, New York: J. Wiley & Sons, 1982.

Davenport, R.A., Data Analysis-Experience with a Formal Methodology, Samlet, P.A. (ed), *EUROIFIP*, 1979, Amsterdam: North-Holland Pub. Co., 1979, pp. 53-64.

Davis, G.B., Munro, M.C., Determining Management Information Needs: A Comparison of Methods, *MIS Quarterly*, June, 1977, pp. 55-66.

De Sanctis, G., Gallupe, B., Group Decision Support Systems: a New Frontier, *Data Base*, winter 1985, pp. 3-10.

Emery, J.C. *Organizational Planning and Control Systems-Theory and Technology*, New York: The Macmillan Co., 1969.

Feldman, M.S., March, J.G. Information in Organizations as Signal and Symbol, *Administrative Science Quarterly*, 26, june, 1981, pp. 171-186.

Fikes, R.E., Austin Henderson, D. Jr., On Supporting the Use of Procedures in Office Work, mimeo, Palo Alto: Xerox PARC, December, 1980.

Flores, F., Ludlow, J.J. Doing and Speaking in the

Office, Fisk, G., Sprague, R. (eds), *DSS: Issues and Challenges*, London, Pergamon Press, 1981.

Galbraith, J.R. *Organization Design*, Reading Mass., Addison Wesley Pub. Co., 1977.

Grandori, A., A Prescriptive Contingency View of Organizational Decision Making, *Administrative Science Quarterly*, 29, June, 1984, pp. 192-209.

Hess, J.D. *The Economics of Organization*, Amsterdam, North-Holland Pub. Co., 1983.

Huber, G.P., The Nature and Design of Post-Indutrial Organizations, *Management Science*, 30, 8, August, 1984. 928-951.

Huber, G. P. Issues in the Design of Group Decision Support Systems, *MIS Quarterly*, 1985.

Jarke, M., Jelassi, M.T., Shakun, M.F., Mediator: Towards a Negotiation Support System, mimeo, Indiana University, June 1985.

Keen, P.G.W. Information Systems and Organizational Change, *Communications of the ACM*, 24, 1, January, 1981, pp. 24-33.

Keen, P.G.W., Scott Morton, M.S., *Decision Support Systems: An Organizational Perspective*, Reading Mass., Addison Wesley Pub. Co., 1978.

Kling, R., Social Analyses of Computing: Theoretical Perspectives in Recent Empirical Research, *Computing Surveys*, 12, 1 (1980), pp. 61-110.

Lee, R.M. CANDID: A Logical Calculus for Describing Financial Contracts, PHD Dissertation, Dep. of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1980.

Markus, M.L. Power, Politics and MIS Implementation, *Communications of the ACM*, 26, 6 June 1983, pp. 430-444.

Marschak, J., Radner, R., *Economic Theory of Teams*, New Haven: Yale University Press, 1972.

Okun, A., *Prices and Quantities*, Oxford, Basil Blackwell, 1981.

Olson, M.H., Remote Office Work: Changing Work Patterns in Space and Time, *Communications of the ACM*, 26, 3, March 1983, pp. 182-187.

Ouchi, W.G., A Conceptual Framework for the Design of Organizational Control Mechanisms, *Management Science*, 259, September, 1979, pp. 838-848.

Ouchi, W.G. Markets, Bureaucracies and Clans, *Administrative Science Quarterly*, 25, March 1980, pp. 129-141.

Pava, C. Microelectronics and the Design of Organization, working paper, Division of Research, Graduate School of Business Aministration, Harvard University, April 1982.

Piore, M.J., Sabel, C.F., *The Second Industrial Divide*, New York, Basic Books, 1984.

Rockart, J.F. Chief Executives Define Their Own Data Needs, *Harvard Business Review*, 57, 2, March-April 1979, pp. 81-93.

Schein, E.H. Coming to a New Awarness of Organizational Culture, *Sloan Management Review*, 26, 5, Winter 1984, pp. 3-16.

Schelling, T.C., *The Strategy of Conflict* (2nd Edition), Cambridge Mass., Harvard University Press, 1980.

Sibley, E.H., The Impact of Database Technology on Business Systems, Gilchrist, B. (ed.), *Information Processing 77*, Amsterdam, North Holland Pub. Co., 1977.

Simon, H.A., A Formal Theory of the Empolyment Relation, Simon, H.A. *Models of Man*, New York, Wiley, 1957, pp. 183-195.

Simon, H.A., Applying Information Technology to Organization Design, *Public Administration Review*, May-June, 1973, pp. 268-278.

Simon, H.A., *Administrative Behavior*, New York, The Free Press, 1976a.

Simon, H.A., From Substantive to Procedural Rationality, Letsis, S.J. (ed.), *Method and Appraisal in Economics*, Cambridge, Cambridge University Press, 1976b.

Simon, H.A., *The New Science of Management Decision*, Englewood Cliffs N.J., Prentice Hall, 1977.

Simon, H.A., *The Sciences of the Artificial* (2nd edition), Cambridge, Mass., The MIT Press, 1981.

Smith, A., *The Wealth of Nations*, 1776.

Sprague, E.H. A Framework for the Development of Decision Support Systems, *MIS Quarterly*, 4, 4, December, 1980, pp. 1-24.

Stone, P.J. Social Evolution and a Computer Science Challenge, *Scientia*, 25, 1980, pp. 125-146.

Strassman, P.A., The Office of the Future, *Technology Review*, January, 1980, pp. 56-64.

Strauss, A. *Negotiations-Varieties, Contexts, Processes and Social Order*, San Francisco, Jossey-Bass Pub., 1978.

Suchmann, L.A., Wynn, E. Procedures and Problems in the Office, *Office Technology and People*, 2, 2, January, 1984, pp. 133-154.

Thmpson, J.D. *Organizations in Action*, New York, Russel Sage, 1967.

Toffler, A. *The Third Wave*, New York, Morrow, 1980.

Turoff, M., Hiltz, S.R. Computer Support for Group Versus Individual Decisions, *IEEE Trans. on Communications*, 30, 2, January, 1982, pp. 82-91.

Weick, K.E., Educational Organizations as Loosely Coupled Systems, *Administrative Science Quarterly*, 21, March 1976, pp. 1-19.

Wildavsky, A. Information as an Organizational Problem, *Journal of Management Studies*, 20, 1, January, 1983, pp. 29-40.

Wilkins, A.L., Ouchi, W.G., Efficient Cultures: Exploring the Relationship between Culture and Organizational Performance, *Administrative Science Quarterly*, 28, September, 1983, 468-481.

Williamson, O.E. *Markets and Hierarchies: Analysis and Antitrust Implications*, New York, The Free Press, 1975.

Williamson, O.E. The Economics of Organization: The Transaction Costs Approach, *American Journal of Sociology*, 87, 3, November, 1981, pp. 548–577.

Williamson, O.E. *The Economic Institutions of Capitalism*, New York, McMillan, 1985.

Techniques for Understanding Unstructured Code

Mel A. Colter
Associate Professor of Management Science and Information Systems
College of Business Administration
University of Colorado at Colorado Springs
P.O. Box 7150
Colorado Springs, Colorado 80933-7150

ABSTRACT

Within the maintenance activity, a great deal of time is spent in the process of understanding unstructured code prior to changing or fixing the program. This involves the comprehension of complex control structures. While automated processes are available to structure entire programs, there is a need for less formal structuring processes to be used by practicing professionals on small programs or local sections of code. This paper presents methods for restructuring complex sequence, selection, and iteration structures into structured logic. The procedures are easily taught and they result in solutions of reduced complexity as compared to the original code. Whether the maintenance programmer uses these procedures simply for understanding, or for actually re-writing the program, they will simplify efforts on unstructured code.

Introduction

The maintenance of existing software comprises a major portion of the productive effort of the software industry. Though estimates vary and questions concerning the exact demarcation between development and maintenance persist, most authorities agree that 40-75% of all DP budgets are expended on maintenance (Boehm, 1981; Couger and Coulter, 1985; Elshoff, 1976; Lientz and Swanson, 1978). From another perspective, Boehm (1981) has surveyed estimates which indicate that up to 75-80% of all life cycle costs are expended on maintenance. The increasing average life of software (Boehm, 1981), along with the growing amount of software entering the maintenance process, indicate that maintenance costs will continue to rise, both in terms of absolute budgets and in terms of total life cycle costs.

A growing body of literature reflects this importance by treating the maintenance effort from multiple perspectives. Some authors have contributed papers which aid understanding of the maintenance process itself (Colter and Couger, 1984; Harrison, et. al., 1982; Vessy and Weber, 1983). Others have concentrated on factors affecting maintenance loads (Berns, 1984; Colter and Couger, 1984; Elshoff, 1976; Gremillion, 1984). Another subset of the literature discusses the management and productivity issues related to the maintenance task (Colter and Couger, 1984; Couger and Colter, 1985; Guimaraes, 1983; Lientz and Swanson, 1981).

Despite the growing literature on maintenance, however, very little published support for practical tools and techniques for performing maintenance exists. While we have greatly improved our understanding of the maintenance process, we have done little to aid the maintenance programmer directly. Early work on translating programs with GOTO's into programs with DOWHILES was published in 1971 (Ashcroft and Manna, 1971). Except for a few other translation and style articles, little else of direct applicability to maintenance programmers appeared until 1982. Then, Elshoff and Marcotty suggested a method for improving the readability of existing code through a series of transformations of the code (Elshoff and Marcotty, 1982).

It is the thesis of this paper that tools, techniques, and methodologies are badly needed to aid the maintenance programmer. In the maintenance mode, most of the target code is of substantially lower quality than we would like. At the same time, a growing percentage of our new employees are coming into maintenance with a good background in structured programming but absolutely no preparation for understanding, modifying, and retesting unstructured programs. This paper extends the work of Elshoff and Marcotty by providing some simple techniques to aid maintenance programmers in the understanding of poorly structured code.

The Unstructured Code Problem

THE QUALITY OF MAINTAINED CODE

A large percentage of the code in maintenance fails to meet today's generally accepted program quality standards. The reasons for this are many, but the following points are most explanatory. First, much of the industry's existing code is old, having been written prior to conversion to the structured techniques. Second, many organizations have yet to implement improved program design and construction standards. Finally, for those shops where clean code is delivered into maintenance, that clean code often degenerates rapidly due to unconstrained maintenance efforts.

The sad truth is that much of the code in maintenance today is of poor design and construction. This problem was noted in a survey of programmers by Lientz and Swanson (1980) and in another survey of programmers and managers by Couger and Colter (1984, 1985). In those studies, programmers reported that poor program design and poor program code accounted for the majority of their problems in the maintenance environment.

The concept of code quality may be discussed at a number of different levels. For the purposes of this paper, a well-structured program is considered to be one which is comprised of a set of hierarchically related modules where the individual modules are of low complexity and easy to understand. In addition, at the code level, the control structures are expected to be predictable and recognizable, reflecting the practices of structured logic.

Unfortunately, much of the code in maintenance consists of large programs (hundreds and even thousands of lines of code per module), which reflect anything but structured logic. This code exhibits complex control structures which must be understood before any maintenance efforts can be successful.

CONTROL FLOW COMPLEXITY

A great deal of discussion on complexity as it relates to maintenance has appeared in the literature. Harrison, et. al. (1982) suggest that control flow metrics do a good job of differentiating between two programs which are otherwise similar. In addition, it appears that the complexity of the control flow of a program is directly related to the amount of time spent in understanding existing code prior to making a change. Therefore, this paper concentrates on a set of techniques for understanding the control flow of unstructured code.

There are two types of control structures contained within any program. They are:

1) Problem-related control structures

2) Implementation-related control structures.

Problem-related control structures are those necessary to solve the program problem effectively. Implementation-related control structures, on the other hand, exist in code only because of the nature of the program solution chosen by the programmer or maintainer and these structures may have little or nothing to do with the original program problem. The issue here is that the maintenance programmer, upon examining the program, has no simple way to determine which control structures are integral to the functionality of the module and which are there simply as a result of poor design or coding practices.

The study and understanding of these combined sets of control structures comprise a significant portion of the amount of time necessary to perform a specific maintenance task. In an informal study of over 200 maintenance programmers undertaken by this author, respondents reported that over 50% of their time in a maintenance effort is taken up by the efforts necessary to understand code prior to making a change. When questioned about this understanding effort, the vast majority of respondents indicated that the clarification of control structures accounted for a large portion of the understanding effort.

Because of the importance of the control structure to maintenance efforts, a simple control related complexity measure will be used throughout the remaineder of this paper to provide comparisons between similar pieces of code. That measure will be the number of branching statements plus 1, which is an approximation to McCabe's cyclomatic complexity number and the lower bound on the complexity calculated by Myers (See Harrison, et. al. (1982)). While a number of other code measures and metrics exist, this simple metric is useful for the comparison of alternative solutions.

THE NEED FOR TOOLS

As indicated above, a great deal of software maintenance is performed on large, complex programs which exhibit unpredictable control flows which require up to 50% of the maintenance effort to understand. Worse yet, much of the effort spent on understanding the existing code is of only short term value since maintenance programmers' notes and other on-the-spot documentation are usually thrown away after the change is successfully made. As a result, the same understanding effort may occur on the same piece of code multiple times over the life of the program. This is an unnecessarily redundant expenditure of scarce resources in the maintenance environment. If the understanding component can be reduced, and if the results of that understanding can be saved effectively, then it should be possible to dramatically reduce the cost of maintaining many systems.

Clearly, the maintenance programmer needs tools to aid in the understanding of existing code. In this paper, a set of procedures are offered to meet this need. As noted before, these procedures are extensions of the technique offered by Elshoff and Marcotty (1982). While their approach results in the restructuring of code, it suffers from three major weaknesses. First, it is highly formal and implies that the programmer will actually rewrite the code as a part of the restructuring process. Unfortunately, the rewriting of code is often frowned upon in shops which subscribe to the old adage, "If it ain't broke, don't fix it!" As a result, maintenance programmers who could otherwise benefit from the Elshoff and Marcotty approach fail to reap those benefits because of their perception that they must use all of the procedure and not just part of it. Second, the procedures described by Elshoff and Marcotty require more detailed instructions to be useable to most practicing professionals. Finally, the procedure appears highly formalized. As a tool, it is therefore hard to expect maintenance programmers to use it frequently. Weiser (1982) comments that programmers approach complex programs by using tools in a hierarchical manner. That is, they first attempt to use simple techniques to solve their problems, then move to more complex approaches only when necessary. They continue to apply stronger tools in a stepwise fashion until they succeed in using a tool strong enough to meet the complexity of the problem. The techniques presented in this paper may be used in a highly informal manner, yet they are sufficiently robust to aid in the understanding of extremely complex code.

The goal of the paper is therefore to describe code analysis and understanding tools which:

1) are easy to use

2) significantly decrease the understanding component of a maintenance effort

3) support documentation to aid future maintenance efforts

4) support actual code rewriting when desired.

The processes discussed here use pseudocode as an alternative representation of logic. If unstructured logic can be represented in pseudocode with only sequence, selection, and iteration, then complexity is reduced and understandability is increased.

AUTOMATED VERSUS NON-AUTOMATED RESTRUCTURING

In the past several years, a growing number of automated code restructuring systems have become available. One can now submit COBOL programs to one of several companies and receive a restructured version which meets the rules of structured programming. While some programs are candidates for this type of automated restructuring, the process is not without problems. First, the number of source lines usually increases significantly as a result of the process. Second, the size of the load module increases, as does the average run time for most such programs. Third, the control structures inserted by the automatic restructuring routines seldom have anything to do with the original problem, resulting in a preponderance of implementation related control which obscures the problem related control. Therefore, the understandability of the resulting code remains lower than one would like. Finally, it is often helpful if small programs or program segments can be restructured for understanding purposes without submitting a large program or system for automatic restructuring. It is clear that a large amount of code in existing production libraries will remain in its present state for some time and that human intelligence will be the vehicle for understanding of code prior to maintenance. As Elshoff and Marcotty said in 1982,

> "The understanding developed by the programmer is generally well beyond the capability of artificial intelligence, and the undesirable side-effects often introduced by automatic restructuring techniques can be avoided."

The following section describes tools and techniques which utilize the knowledge of the programmer to achieve a true understanding of code.

Restructuring Techniques

THE RESTRUCTURING PROCESS

When working with poorly structured, complex code, it is generally impossible to attach all of the weakness of the program simultaneously. As a result, programmers usually seek to identify subsets of the program which support meaningful efforts. Weiser (1982) refers to these program subsets as "sliced" which represent relevant portions of a program for the purposes of specific analysis.

The techniques presented here explicitly assume the use of slices to segment code into understandable and modifiable segments. As Weiser points out, there are many different types of slices, and more than one will be used here. However, the most common slice will be the *code block*. A code block is defined as a set of contiguous statements which have a single entry and a single exit. The code block may be a few lines of code, or it may be an entire program. The importance of the code block in the analysis of a program for understandability is twofold. First, statements in code blocks may be clumped together to simplify the program portion in which the block resides. Second, in order to reorder or otherwise modify the logic in a section of code, that section of code must represent a code block. That is, the single entry, single exit criterion is critical to the re-representation of logic.

In this paper, it is suggested that the restructuring and understanding process begin with the most straightforward targets of opportunity and progress towards the more difficult portions of code. In general, the easiest way to simplify a program is to deal with code blocks which are simply out of place. When code blocks are moved to their appropriate location in the program, control structures are reduced and the sequential nature of the logic is clarified.

After the sequential nature of the logic is cleaned up, then the selection constructs are usually the next easiest portions of the code to understand. In languages which do not support the IF-ELSE-ENDIF structure, the selection construct accounts for a great deal of implementation related control. As a result, the re-representation of unstructured selection contructs greatly simplifies the logic of the program. Finally, after the sequence and selection contructs are understood, the maintenance programmer can concentrate on the iteration constructs. Unstructured loops are among the most difficult to understand and it is best to simplify the program to the greatest extent possible before tackling them. The following discussions present detailed examples of the understanding and re-representation of sequence, selection, and iteration.

CODE BLOCKS—THE SEQUENCE PROBLEM

The simplest code block to recognize and deal with results when a block of code simply resides in one portion of the program while its execution belongs in another location. For example, Figure 1 shows a situation in which code block C has been added to the end of a subroutine rather than inserted into the logic where it belongs. This type of situation may reflect a last minute addition during design, or it may be the result of an addition of code during maintenance. In any case, it decreases the readability of the program and increases the complexity through the addition of two unnecessary control statements. These control statements are classic examples of implementation related control. They have absolutely nothing to do with the original problem and greatly decrease understandability.

In this situation, code block C cannot be reached through sequential execution and it is clear that it can simply be moved to the appropriate location in the program. This is illustrated in Figure 1, resulting in a reduction in the control flow complexity of two. Notice that, assuming no other reference to Label-1 and Label-2, they may be removed, further simplifying the program. Furthermore, note that the new configuration of code blocks A, C, and B may support their merging into a single conceptual block, since no control structures exist to separate them.

A more common type of sequential code block problem is illustrated in Figure 2. There, the code block is reused rather than duplicated in the code. For the purposes of understanding the section of code in which this structure resides, it is worth copying the code block to achieve a reduction in the control complexity of the code of interest. As shown in Figure 2, the copying of the code block C allows us to remove two inplementation related control structures, delete the use of the control variable, FLAG, and remove references to Label-1 and Label-2.

When trying to simplify code to aid understanding, this treatment of code blocks is the best place to begin restructuring the code. First, the structures are relatively easy to identify in the code. Second, each time a code block is moved or copied to its proper sequential location, the control complexity is reduced by two and the understandability is greatly increased. Even though this process may result in an increase in the actual amount of code in the program, that increase is easily offset by the positive results of the process. Once all of the opportunities for the clarification of the sequential structure of the program have been exhausted, then the more complex structures may be examined.

THE SELECTION STRUCTURE

Of the three major logical structures (sequence, selection, and iteration), the selection construct becomes the most awkward when it is not implemented cleanly. When the IF-ELSE-ENDIF structure is not available in a lan-

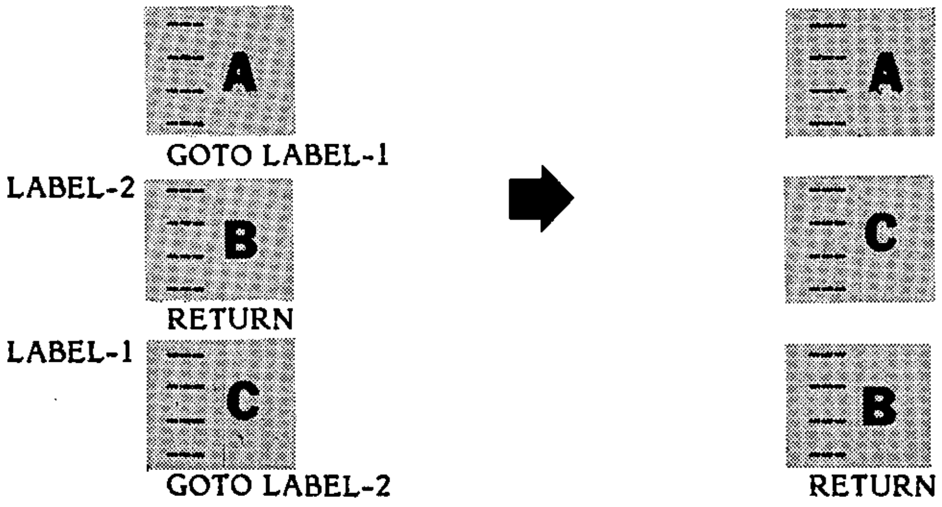


Figure 1

Code Block Out Of Place

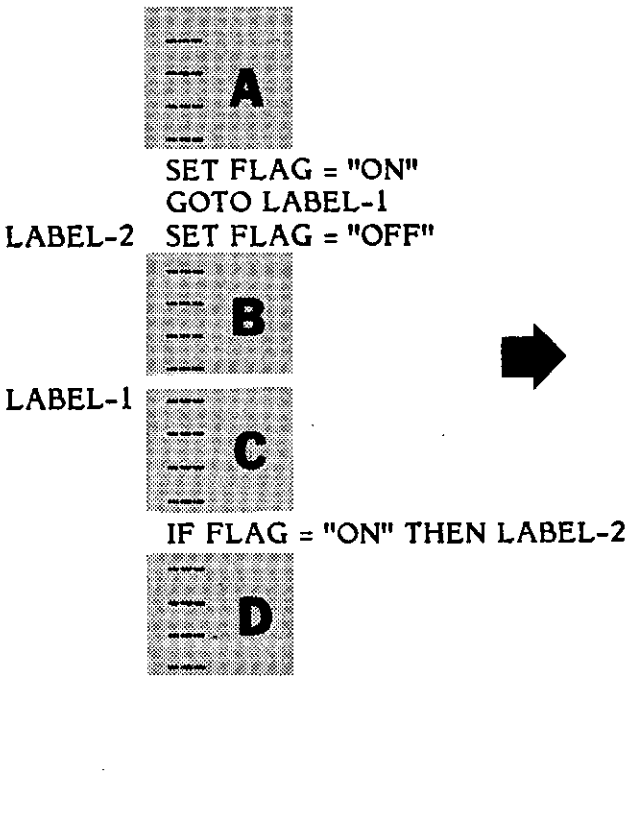


Figure 2

Code Block Duplicated

guage, or when it is available but not used, the program will exhibit complex combinations of conditional and unconditional branches. As a result, the exact nature of the original problem becomes obscure and maintenance efforts are extremely difficult.

The restructuring of complex selection contructs requires a careful set of steps as indicated below.

1) Isolate the selection structure as a code block with single entry and exit.

2) Expand the structure by formalizing the IF-ELSE-ENDIF structures.

3) Collapse the structure into itself by moving internal code blocks.

4) Remove redundant control statements.

This process is illustrated in Figure 3, and the following discussions clarify the series of transformations suggested for the example. This code is a simplification of code from an actual program, and it is common within the programs of many shops. First, note that the structures of interest for this example are simplified by summarizing all but the important control structures. For example, the line • PROCESS-A • represents an internal code block withing the structure. That code block may be a single line of code or it may contain significant complexity of its own. Second, the structure must be recognized as a selection structure and isolated as a code block with single entry and exit.

Considering the problem of recognizing this type of structure, note that the initial version of the code in Figure 3 represents the original unstructured code. In that version, note that the control flows are all downward and intersecting with a subset of the control branches terminating at a common exit. This set of characteristics is commonly seen in structured implementation of the selection construct.

The second version of the program in Figure 3 results from the expansion of the structure through the formalization of the IF-ELSE-ENDIF structures. In this process, a simple translation into formalized pseudocode has occurred. The IF-ELSE-ENDIF structures are clarified through some expansion of the original code. In addition, note that an implied GOTO has been added at the end of the code at LABEL-5. In the original code, there is a sequential execution of LABEL-4 after LABEL-5. However, the restructured version will probably result in the movement of LABEL-5 as a single entry, single exit code block and the transfer code of control to LABEL-4 must be maintained. As a result, the implied (GOTO LABEL-4) at the end of the structure is always added to the code at this point.

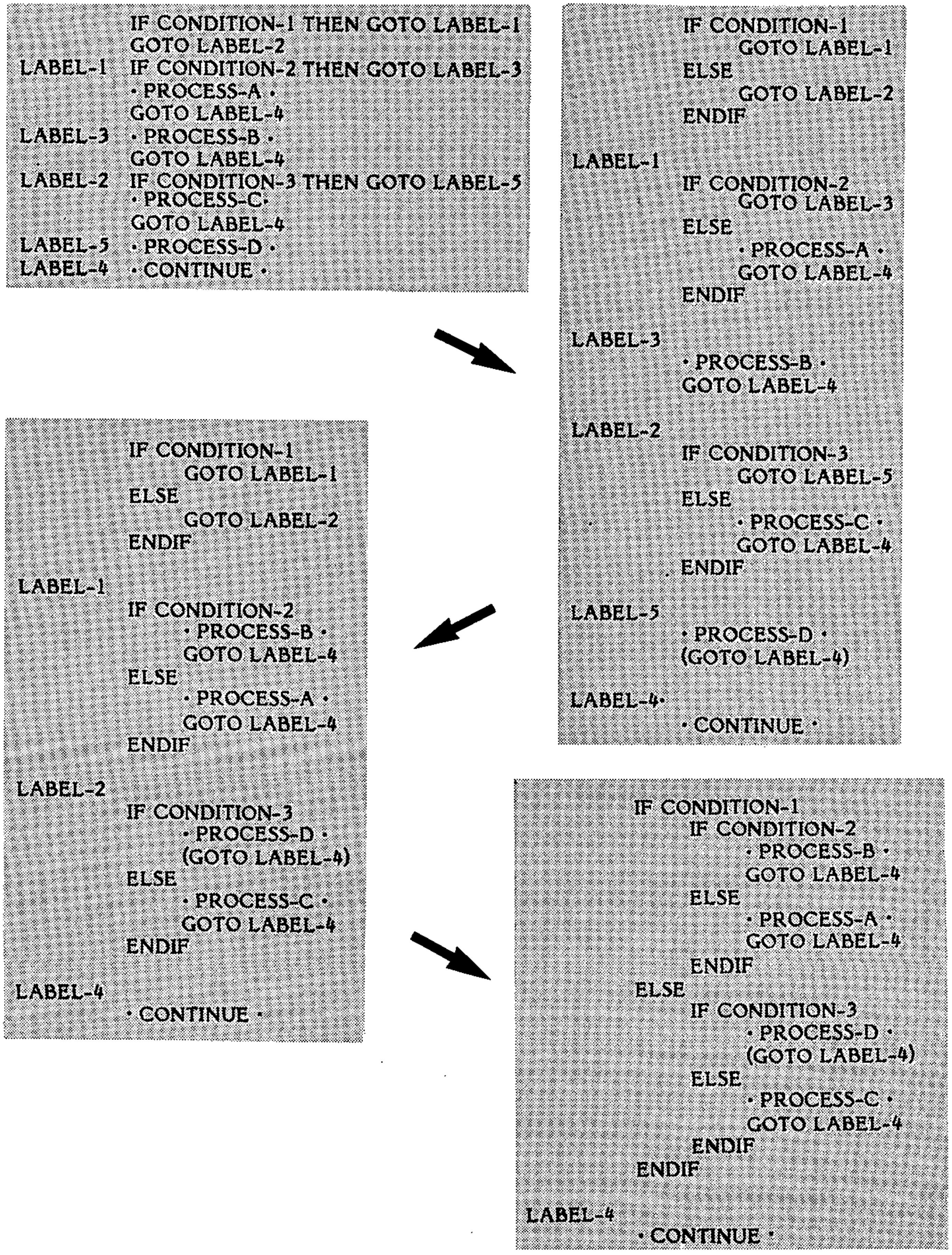
The next step in the restructuring of the code involves the collapsing of the structure into the selection constructs. While this process may be performed quickly by a professional maintenance programmer who is familiar with the process, it is broken into two steps here for the purposes of illustration. The key to the collapsing process is to realize that the second version of the code contains two code blocks which present opportunities for relocation. First, note that the code at LABEL-3 and LABEL-5 ends with control transfers to the end of the code segment. Second, note that both of these code blocks are single entry, single exit, and that they are accessed only through the execution of additional GOTO's in LABEL-1 and LABEL-2. As a result, the code block at LABEL-3 can replace the GOTO LABEL-3 within LABEL-1 and the code block at LABEL-5 can replace the GOTO LABEL-5 within LABEL-2. The third version of the program segment reflects this set of code block movements.

Now, recognize that all of the code under LABEL-1 represents a code block with a single entry and exits to a common location. Furthermore, this block is directly accessed through the execution of the GOTO LABEL-1 at the top of the code. As a result, all of the code under LABEL-1 can be moved to replace the GOTO LABEL-1 statement. The same argument allows us to move all of the code under LABEL-2 to replace the GOTO LABEL-2 statement. Note here that the explicit declaration of the implied (GOTO LABEL-4) which was added early in the process is now critical. Without that implied GOTO, the code block movement would be highly constrained.

The fourth version of the program segment illustrates the complete collapsing of the structure into the set of IF-ELSE-ENDIF structures. Nore that there are four occurrences of the statement, GOTO LABEL-4, in this version. However, the natural operation of the selection construct makes these statements totally redundant. Whenever the execution reaches one of these statements, the operation of the selection constructs would result in a clean jump to the end of the construct anyway. The removal of these unnecessary control statements is illustrated in Figure 4, along with the original code for comparison. It is clear that the restructured logic is much easier to read and that the programmer who understands the restructured version will be able to work with the original version if necessary. Note also that the complexity of this code has been reduced to a value of 4 from an original value of 8.

UNSTRUCTURED LOOPS

The last major structure causing problems in unstructured code is the iteration structure. Here, because of the weaknesses of specific languages or due to improper use of stronger languages, programmers create multiple exit loops and intersecting loops which make maintenance



```
           IF CONDITION-1 THEN GOTO LABEL-1
           GOTO LABEL-2
LABEL-1    IF CONDITION-2 THEN GOTO LABEL-3
           · PROCESS-A ·
           GOTO LABEL-4
LABEL-3    · PROCESS-B ·
           GOTO LABEL-4
LABEL-2    IF CONDITION-3 THEN GOTO LABEL-5
           · PROCESS-C·
           GOTO LABEL-4
LABEL-5    · PROCESS-D ·
LABEL-4    · CONTINUE ·
```

```
           IF CONDITION-1
               GOTO LABEL-1
           ELSE
               GOTO LABEL-2
           ENDIF

LABEL-1
           IF CONDITION-2
               GOTO LABEL-3
           ELSE
               · PROCESS-A ·
               GOTO LABEL-4
           ENDIF

LABEL-3
           · PROCESS-B ·
           GOTO LABEL-4

LABEL-2
           IF CONDITION-3
               GOTO LABEL-5
           ELSE
               · PROCESS-C ·
               GOTO LABEL-4
           ENDIF

LABEL-5
           · PROCESS-D ·
           (GOTO LABEL-4)

LABEL-4·
           · CONTINUE ·
```

```
           IF CONDITION-1
               GOTO LABEL-1
           ELSE
               GOTO LABEL-2
           ENDIF

LABEL-1
           IF CONDITION-2
               · PROCESS-B ·
               GOTO LABEL-4
           ELSE
               · PROCESS-A ·
               GOTO LABEL-4
           ENDIF

LABEL-2
           IF CONDITION-3
               · PROCESS-D ·
               (GOTO LABEL-4)
           ELSE
               · PROCESS-C ·
               GOTO LABEL-4
           ENDIF

LABEL-4
           · CONTINUE ·
```

```
           IF CONDITION-1
               IF CONDITION-2
                   · PROCESS-B ·
                   GOTO LABEL-4
               ELSE
                   · PROCESS-A ·
                   GOTO LABEL-4
               ENDIF
           ELSE
               IF CONDITION-3
                   · PROCESS-D ·
                   (GOTO LABEL-4)
               ELSE
                   · PROCESS-C ·
                   GOTO LABEL-4
               ENDIF
           ENDIF

LABEL-4
           · CONTINUE ·
```

Figure 3

Understanding Unstructured Selection Constructs

```
         IF CONDITION-1 THEN GOTO LABEL-1
         GOTO LABEL-2
LABEL-1  IF CONDITION-2 THEN GOTO LABEL-3
         • PROCESS-A •
         GOTO LABEL-4
LABEL-3  • PROCESS-B •
         GOTO LABEL-4
LABEL-2  IF CONDITION-3 THEN GOTO LABEL-5
         • PROCESS-C•
         GOTO LABEL-4
LABEL-5  • PROCESS-D •
LABEL-4  • CONTINUE •
```

➡

```
         IF CONDITION-1
             IF CONDITION-2
                     • PROCESS-B •
             ELSE
                     • PROCESS-A •
             ENDIF
         ELSE
             IF CONDITION-3
                     • PROCESS-D •
             ELSE
                     • PROCESS-C •
             ENDIF
         ENDIF

LABEL-4
         • CONTINUE •
```

Figure 4

The Restructured Selection Construct

efforts extremely difficult. This section first treats the multiple exit loop problem. Then, the intersecting loop problem is discussed at length.

Multiple Exit Loops

Loops with multiple exits are quite common in older, unstructured code. Additionally, newer code often exhibits this characteristic due to the need for multiple paths out of iterative structures. For example, in on-line systems, loops may terminate normally, because of a bad data value, or because of the use of an interrupt key by the operator. While these problems may be handled with purely structured logic, the resulting solutions often contain multiple levels of nested IF structures and programmers commonly refuse to implement such structures.

Figure 5 shows a multiple exit loop and an alternative solution to the code segment. First, notice that the original code has two branches to labels which are external to this code segment. These branches violate the single entry, single exit criterion. Worse, they may transfer control to portions of the program which are far away from the segment of interest. The maintenance programmer who must trace an error through this loop will have to locate the external labels. In some cases, it may be difficult or impossible to determine exactly which exit from the loop was accomplished for a given situation.

In the alternative solution, the total amount of code has been increased in order to clarify the loop structure. It is assumed in this case that the variable, I, was originally used simply to create a looping structure which would be exited through one of the internal exit structures. However, that variable has been included in the alternative solution to provide an error procedure in case the loop is not exited in a normal fashion. Otherwise, the loop will be terminated when the variable, EXIT-CONDITION, is set to anything other than "NULL". The form of this solution requires that GOTO statements be embedded in the code, but they branch downward and only to the end of the logical structure. This use of GOTO statements, while not approved by purists, is still an improvement over the original code.

The real strength of the new solution is that it explicitly indicates the methods by which the loop exit can be accomplished at the end of the loop structure. An examination of the current value of EXIT-CONDITION will clearly indicate the nature of the last loop exit. Furthermore, the structure easily accommodates the later insertion of additional exit criteria during maintenance of the program.

The Intersecting Loop Problem

Of all unstructured program problems, the intersecting loop situation is among the most difficult to understand, debug, or modify. This section presents a stepwise transformation process which converts intersecting loops into a set of logical structures using only DOWHILE and DOUNTIL structures. The discussion uses the example

```
    DO 100  I = 1 TO 9999
          • PROCESS-A •
          IF (CONDITION-1) THEN GOTO LABEL-1
          • PROCESS-B •
          IF (CONDITION-2) THEN GOTO LABEL-2
          • PROCESS-C •
100 CONTINUE
```

```
    SET EXIT-CONDITION = "NULL"
    SET COUNT = 0
    DOWHILE EXIT-CONDITION = "NULL"
       INCREMENT COUNT
       • PROCESS-A •
       IF (CONDITION-1)
          SET EXIT-CONDITION = "BAD DATA"
          GOTO 100
       ENDIF
       • PROCESS-B •
       IF (CONDITION-2)
          SET EXIT-CONDITION = "EDIT ERROR"
          GOTO 100
       ENDIF
       • PROCESS-C •
       IF (COUNT.GE.9999)
          SET EXIT-CONDITION = "ERROR"
       ENDIF
100 ENDDO
    IF (EXIT-CONDITION.EQ."BAD DATA")
       GOTO LABEL-1
    ELSEIF (EXIT-CONDITION.EQ."EDIT ERROR")
       GOTO LABEL-2
    ELSEIF (EXIT-CONDITION.EQ."ERROR")
       • HANDLE ERROR •
    ENDIF
```

Figure 5

Multiple Loop Exits

in Figure 6 and consists of the following steps.

1) Isolate the looping structure as a code block with single entry and exit.

2) Simplify the structure by identifying internal code blocks.

3) Represent the simplified structure as a flowchart.

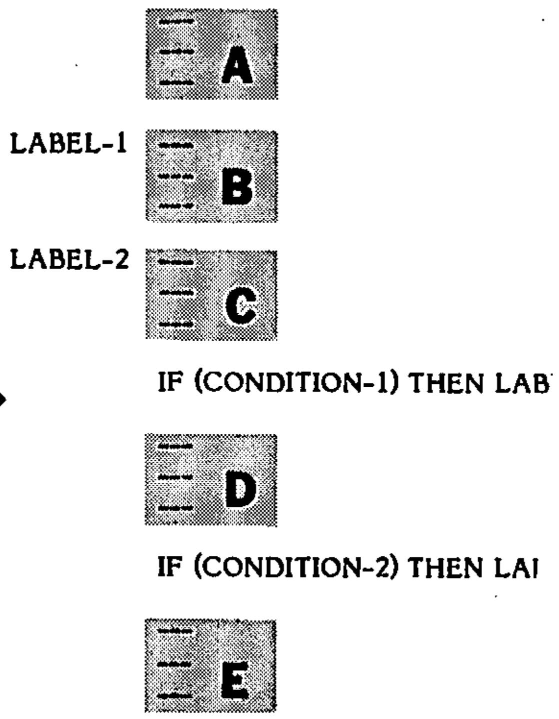
4) Convert the flowchart to pseudocode using only structured logic.

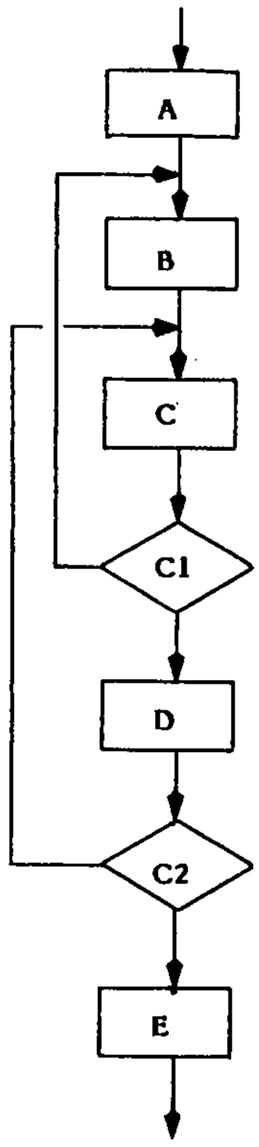
5) Simplify the pseudocode.

In Figure 6, it is assumed that the code represented in the example is a single entry and exit code block and that no other references to the statement labels 100 and 200 exist. Furthermore, assume that the lines of code between the labels and the control statements are irrelevant to this analysis. That is, those lines of code are either pure sequence, or they contain control structures of their own, but they may be represented as code blocks for the purposes of understanding the looping constructs. It is critical to this analysis that only the looping structures remain in the target code. This is why the simplification of the sequence and selection structures is performed first. If all other opportunities for simplification have been taken, then only looping structures remain for consideration. The second portion of Figure 6 shows the introduction of code blocks A through E to achieve the simplification necessary to consider the loops.

Once the code is simplified and the loops are clearly identified, a simplified flowchart of the program may be drawn. This step is important in the transformation of the intersecting loops into structured logic. Remember that intersecting loops are not possible when only sequence, selection, and iteration are used. Therefore, the original program cannot be converted directly into structured pseudocode. In this case, the more general logical representation available through flowcharts must be used as an

```
          ---
          ---
          ---
LABEL-1   ---
          ---
          ---
          ---
LABEL-2   ---
          ---
          ---
          IF (CONDITION-1) THEN LABEL-1
          ---
          ---
          ---
          IF (CONDITION-2) THEN LABEL-2
          ---
          ---
          ---
```





```
DO A
DO B
DO C
DOWHILE C1
    DO B
    DO C
ENDDO
DO D
DOWHILE C2
    DO C
    DOWHILE C1
        DO B
        DO C
    ENDDO
    DO D
ENDDO
DO E
```

```
DO A
DO B
DOUNTIL NOT C2
    DO C
    DOWHILE C1
        DO B
        DO C
    ENDDO
    DO D
ENDDO
DO E
```

Figure 6

Intersecting Loops

intermediate transformation. Only then can the structured logical constructs be identified.

The fourth section of Figure 6 shows the pseudocode equivalent of the flowchart. The process by which this pseudocode is obtained is simplified if a few straightforward rules are followed. First, simply represent a single flowchart symbol, one at a time, resulting in a single code block operation for each line of pseudocode. Second, never anticipate loops. Always wait to implement a loop until the condition branch symbol is encountered. In this case, the first conditional branch checks the value of Condition-1. At this point, since the program checks the condition prior to the execution of the loop, the pseudocode representation requires a DOWHILE structure. This should always be the case when translating from flowcharts to pseudocode. Always wait until the conditional control transfer is encountered and then implement a DOWHILE. This is not to say that no DOUNTILs will be encountered in this process. They will be discovered in later steps as the pseudocode is simplified.

After the pseudocode representation is obtained, examine the structure for opportunities for simplification. In general, this simplification will occur when common code blocks are identified and recombined. In this case, the two shaded portions of the pseudocode solution are duplicate blocks. In fact, the common block is performed once and then performed again in a DOWHILE structure. This can be re-represented as a DOUNTIL. The final portion of Figure 6 shows that simplification. At this point, there is no further obvious simplification possible of the logic.

Note that the original problem in this case contained two loops which intersected. However, the structured solution contains two nested loops. Though no formal proof is known to this author, it has been my experience and the experience of others using this simplification process that a set of n intersecting loops always converts into a set n nested loops in the structured logic.

A GENERAL EXAMPLE

The preceding sections have discussed a set of procedures by which problems of sequence, selection, and iteration can be restructured through re-representations in structured logic. This section provides a detailed discussion of a code section in which a number of such problems exist. Within the code section presented in Figure 7, there is a code block out of place due to the re-use of code within the program. Additionally, there is a selection construct which has been implemented with conditional and unconditional branching statements. Finally, when all of the other issues are clarified, a set of intersecting loops are found to exist. The following paragraphs detail the use of the rules discussed in the earlier sections to obtain an alternative representation of this logic. The goal of the new representation is to obtain a version of the logic which can be understood by the maintenance programmer.

First, in Figure 7, assume that the relevant code has been examined and that there are no external references to any of the statement labels indicated in the code. Furthermore, assume that the segment is single entry and single exit. Also, in Figure 7, note that the program segment has been broken into a set of code blocks to simplify discussion of the problem.

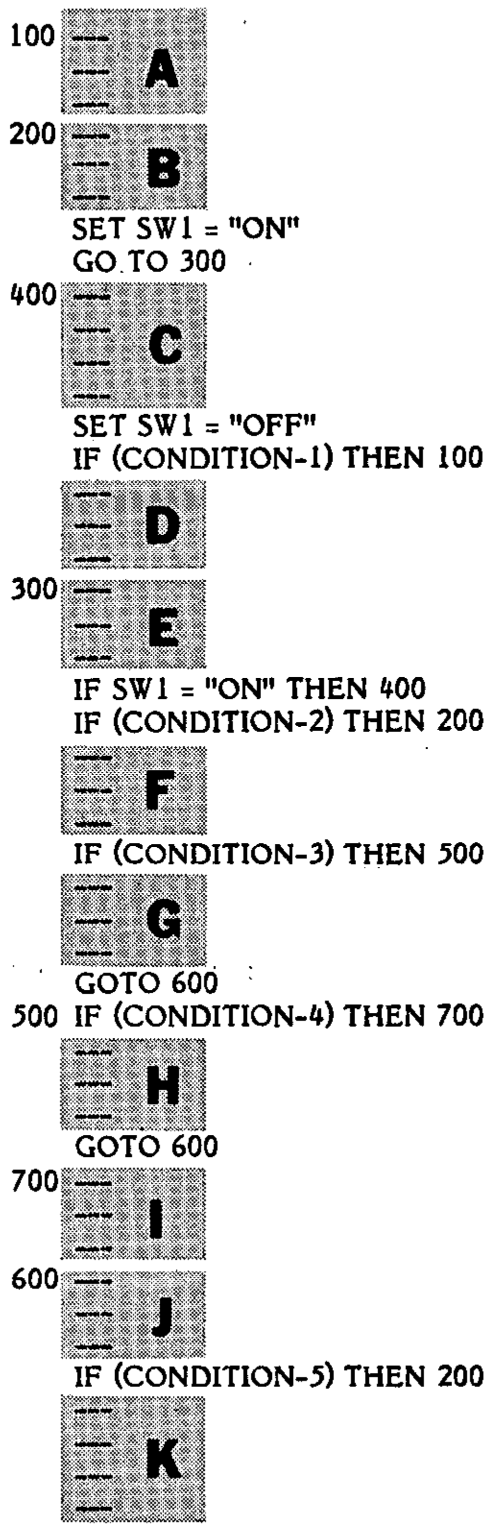
The next step involves the identification and handling of any code blocks which are out of sequence. Examination of the code reveals that code block E is used in two ways. First, it is executed sequentially immediately after block D. However, it is also executed through the use of a switch and some control code after the processing of block B. Here, the copying of code block E between blocks B and C allows us to delete both references to SW1, and remove the two control statements, GOTO 300 and IF SW1 = "ON" THEN 400. Finally, statement labels 300 and 400 are no longer needed.

Figure 8 shows the results of moving the code block and the removal of the implementation related control. The control related complexity of the original program was 10. Now, that complexity has been reduced to 8.

Figure 8 also shows that, without the control statements and unnecessary labels, the blocks within the program may be re-identified. Blocks B, E, and C may now be combined for the remainder of the analysis, resulting in the new block designations shown in the figure.

Since there are no more opportunities to move code blocks to simplify the sequence structure of the program, we now seek to identify selection constructs within the code. It is clear that the code in blocks D, E, F, G, and H is related through a set of control structures which are downward branching and intersecting. Furthermore, there is a common exit indicated at line 600. As noted earlier, this is an example of the kind of solutions which result when selection constructs are built with conditional and unconditional branches.

Figure 9 shows the program segment after the selection structure has been restructured. As a result of this restructuring, the complexity has dropped to 6. Therefore, the understanding process has begun with a program with a complexity of 10 and reduced it to a complexity of 6 in only two steps. The restructuring process used here was exactly the same as that discussed in the section on selection construct. In the first version in Figure 9, the code blocks are labeled as they were in Figure 8. However, the entire section from block D through block H may now be treated as a single code block because there are no further opportunities to simplify any code within that section.



```
100 A
200 B
    SET SW1 = "ON"
    GO TO 300
400 C
    SET SW1 = "OFF"
    IF (CONDITION-1) THEN 100
    D
300 E
    IF SW1 = "ON" THEN 400
    IF (CONDITION-2) THEN 200
    F
    IF (CONDITION-3) THEN 500
    G
    GOTO 600
500 IF (CONDITION-4) THEN 700
    H
    GOTO 600
700 I
600 J
    IF (CONDITION-5) THEN 200
    K
```

Figure 7

Spaghetti Code Program With Code Blocks Identified

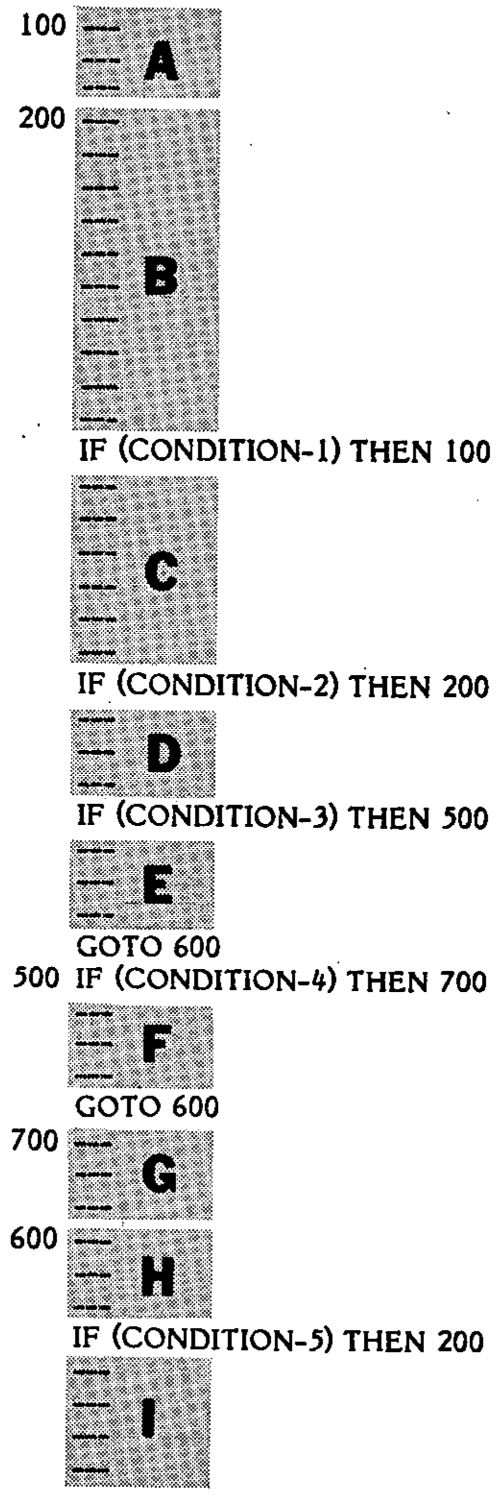

100
A
200
B
IF (CONDITION-1) THEN 100
C
IF (CONDITION-2) THEN 200
D
IF (CONDITION-3) THEN 500
E
GOTO 600
500 IF (CONDITION-4) THEN 700
F
GOTO 600
700
G
600
H
IF (CONDITION-5) THEN 200
I

Figure 8

Code Block E Duplicated in Proper Sequential Position

Therefore, the second version of the problem in Figure 9 has compressed all of that code into a single code block and relabled the blocks. This leaves the problem ready for treatment of the looping constructs. Clearly, there are three loops in this code, all of them intersecting.

Since the first two steps of the loop restructuring process have already taken place (isolating the code and labeling the code blocks), we are now ready to represent the program as a simplified flowchart which shows only the looping structures. That flowchart is illustrated in Figure 10. Also in Figure 10, the parallel representation of the problem in pseudocode is shown. Obviously, the pseudocode solution is much longer and appears to be more complex than the flowchart. This is because of the limited representational ability of pseudocode. Because only sequence, selection, and iteration may be used, the complexity of the flowchart solution must be handled through an expanded use of a limited set of structures. However, opportunities for simplification exist in this pseudocode solution.

In Figure 10, two large code blocks exist. These blocks are exact duplicates of each other. Furthermore, the first block is performed and then the second block is immediately performed inside of a DOWHILE loop. The conversion of this structure into a DOUNTIL reduces the amount of pseudocode by approximately one-half. This reduction is shown in Figure 11. There, another set of common code blocks exist. Again, the blocks are duplicates with one performed just prior to the performance of the other inside of a DOWHILE. The second portion of Figure 11 shows the further reduction of the code which is possible as a result of this second set of duplicate blocks.

This completes the simplification of the pseudocode. Furthermore, no further restructuring of the program is necessary. The original code has been simplified, restructured, and clarified through the processes discussed in earlier sections. To clearly show the differences in the two versions, Figure 12 contains the original program, along with the final version. Note that the complexity of the original version is 10, while the complexity of the simplified version is reduced to 6.

USING THE RESTRUCTURED SOLUTION

The previous sections have detailed methods for understanding complex, unstructured code by restructuring it into structured pseudocode. The primary direction of the presentation has been to provide an aid to understanding code in the maintenance environment.

Once the code is re-represented, one must decide what to do with the simplified solution. In the most informal case, the programmer simply uses these techniques with a primary goal of understanding the code to support maintenance changes. In the most formal case, the maintenance programmer actually uses the simplified solution to rewrite the section of code of interest. Between these two extremes, there are other alternatives. First, the simplified representation may be added to the program documentation package to support future maintenance efforts. Second, the pseudocode may be added in comments just prior to the affected code segment. This saves the results of the understanding effort in the most usable location and makes them easily available to future maintenance personnel.

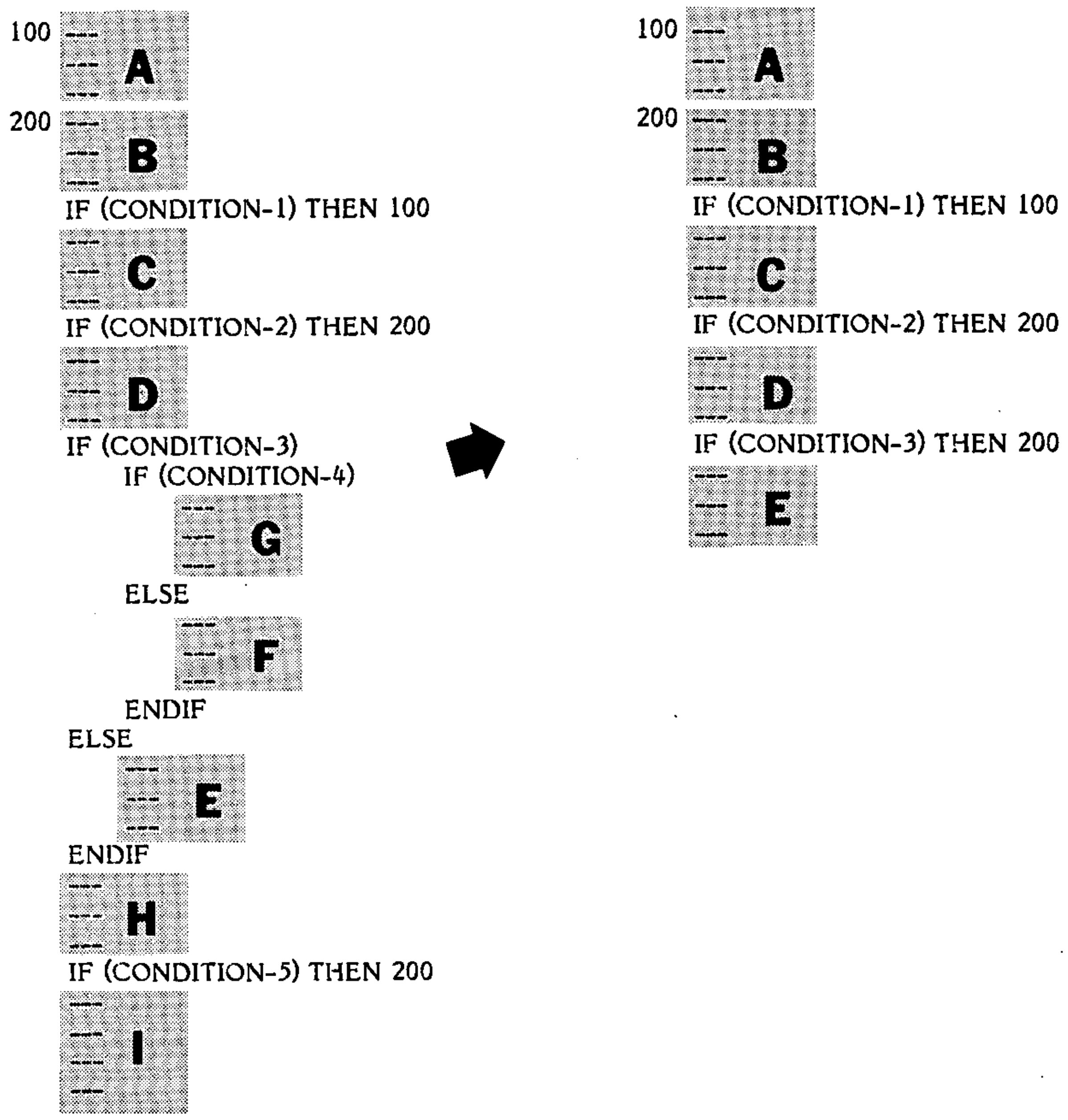
The maintenance programmer who uses these techniques and then saves the results, either by rewriting code or by formalizing the simplified solution into the documentation, benefits in two ways. First, the understanding of the existing program will take significantly less time with these methods. Second, if the results are saved, the understanding component will be reduced for all future maintenance efforts on that code section.

Summary

There is no argument as to the scope and importance of maintenance expenditures. Also, there is little doubt that much of the maintenance effort is spent on the understanding of code prior to debugging and modifying programs. Clearly, the understanding component of maintenance is a major target of opportunity for those seeking to reduce or control maintenance expenditures.

This paper strongly suggests that the understanding effort can be significantly reduced through the formalization of techniques which may be used in that effort. The approaches discussed here have been used successfully by a number of organizations in the public and private sectors with great success. The key to this approach lies in its ability to reduce the complexity of a code section through the creation of predictable code structures. Furthermore, the method can be applied to localized code sections when automated restructuring is unavailable or not desired.

In summary, complex code may be understood best by concentrating first on the sequential aspects of the program. Next, the selection constructs may be examined, particularly if the selection structures are implemented with conditional and unconditional branches instead of the IF-ELSE-ENDIF structures. Finally, the looping constructs may be simplified. With this set of procedures, each step yields reductions in control flow complexity and makes it possible for the next set of logical structures to be isolated and simplified.

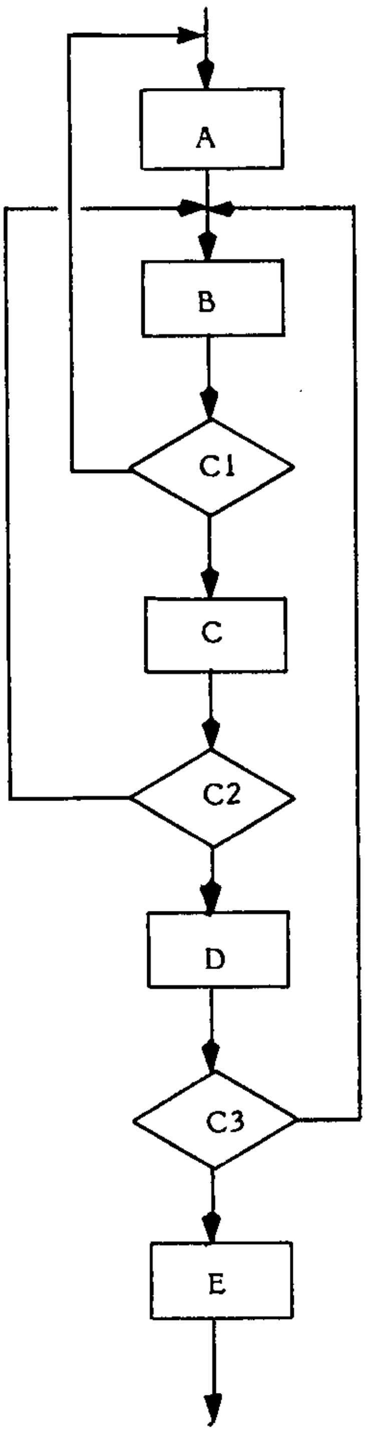


```
100 A
200 B
    IF (CONDITION-1) THEN 100
    C
    IF (CONDITION-2) THEN 200
    D
    IF (CONDITION-3)
        IF (CONDITION-4)
            G
        ELSE
            F
        ENDIF
    ELSE
        E
    ENDIF
    H
    IF (CONDITION-5) THEN 200
    I
```

```
100 A
200 B
    IF (CONDITION-1) THEN 100
    C
    IF (CONDITION-2) THEN 200
    D
    IF (CONDITION-3) THEN 200
    E
```

Figure 9

Selection Construct Structured and Blocked



```
DO A
DO B
DOWHILE C1
    DO A
    DO B
ENDDO
DO C
DOWHILE C2
    DO B
    DOWHILE C1
        DO A
        DO B
    ENDDO
    DO C
ENDDO
DO D
DOWHILE C5
    DO B
    DOWHILE C1
        DO A
        DO B
    ENDDO
    DO C
    DOWHILE C2
        DO B
        DOWHILE C1
            DO A
            DO B
        ENDDO
        DO C
    ENDDO
    DO D
ENDDO
DO E
```

Figure 10

Flowchart Representation and Pseudocode Translation of Loops

```
DO A
DOUNTIL NOT C5
     DO B
     DOWHILE C1
          DO A
          DO B
     ENDDO
     DO C
     DOWHILE C2
          DO B
          DOWHILE C1
               DO A
               DO B
          ENDDO
          DO C
     ENDDO
     DO D
ENDDO
DO E
```

```
DO A
DOUNTIL NOT C5
     DOUNTIL NOT C2
          DO B
          DOWHILE C1
               DO A
               DO B
          ENDDO
          DO C
     ENDDO
     DO D
ENDDO
DO E
```

Figure 11

Final Contraction of the Structured Logic From the Spaghetti Code

```
100 ---
    ---
    ---
200 ---
    ---
    ---
    SET SW1 = "ON"
    GO TO 300
400 ---
    ---
    ---
    ---
    SET SW1 = "OFF"
    IF (CONDITION-1) THEN 100
    ---
    ---
    ---
300 ---
    ---
    ---
    IF SW1 = "ON" THEN 400
    IF (CONDITION-2) THEN 200
    ---
    ---
    ---
    IF (CONDITION-3) THEN 500
    ---
    ---
    ---
    GOTO 600
500 IF (CONDITION-4) THEN 700
    ---
    ---
    ---
    GOTO 600
700 ---
    ---
    ---
600 ---
    ---
    ---
    IF (CONDITION-5) THEN 200
    ---
    ---
    ---
    ---
```

```
---
---
---
DOUNTIL NOT CONDITION-5
    DOUNTIL NOT CONDITION-2
        ---
        ---
        ---
        DOWHILE CONDITION-1
            ---
            ---
            ---
        ENDDO
        ---
        ---
        ---
    ENDDO
    ---
    ---
    ---
    IF CONDITION-3
        IF CONDITION-4
            ---
            ---
            ---
        ELSE
            ---
            ---
            ---
        ENDIF
    ELSE
        ---
        ---
        ---
    ENDIF
    ---
    ---
    ---
ENDDO
---
---
---
```

Figure 12

Original Spaghetti Code versus Restructured Version

REFERENCES

Ashcroft, E., and Manna, Z., "The Translation of 'GOTO' Programs to 'WHILE' Programs", *Proceedings of the 1971 IFIP Congress*, Ljubljana, Yugoslavia, August 1971, pp 250-255.

Bartol, K.M., "Turnover among DP personnel: a causal analysis", *Communications of the ACM*, Volume 26, Number 10 (October 1983), pp 807-811.

Berns, G.M., "Assessing Software Maintainability", *Communications of the ACM*, Volume 27, Number 1 (January 1984), pp 14-23.

Boehm, B.W. *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

Colter, M.A., and Couger, J.D., "Management and Employee Perceptions of the Maintenance Activity", *Proceedings of the Software Maintenance Workshop*, IEEE Computer Society Press, Silver Springs, Maryland, 1984, p 86.

Couger, J.D., and Colter, M.A., "The Effects of Maintenance Assignments on Goal Congruence for Programmers and Analysts", *Proceedings of the Fifth International Conference on Information Systems*, Tucson, Arizona, (November 1984), pp 83-100.

Couger, J.D., and Colter, M.A., *Maintenance Programming: Improved Productivity Through Motivation*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

Curtis, B., Sheppard, S.B., Milliman, P., Borst, M.A., and Love, T., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", *IEEE Transactions on Software Engineering*, SE-5,2 (March 1979), pp 96-104.

Elshoff, J.L., "An Analysis of Some Commercial PL/I Programs", *IEEE Transactions on Software Engineering*, SE-2, 2 (June 1976), pp 113-120.

Elshoff, J.L., "The Influence of Structured Programming on PL/I Program Profiles", *IEEE Transactions on Software Engineering*, SE-3, 5 (September 1977), pp 364-368.

Elshoff, J.L., and Marcotty, M., "On the Use of the Cyclomatic Number to Measure Program Complexity", *SIGPLAN Notices*, Volume 13, Number 12 (December 1978), pp 29-40.

Elshoff, J.L., and Marcotty, M., "Improving Computer Program Readability to Aid Modification", *Communications of the ACM*, Volume 25, Number 8 (August 1982), pp 512-521.

Gremillion, L.L., "Determinants of Program Repair Maintenance Requirements", *Communications of the ACM*, Volume 27, Number 8 (August 1984), pp 826-832.

Guimaraes, T., "Managing Application Program Maintenance Expenditures", *Communications of the ACM*, Volume 26, Number 10 (October 1983), pp 739-746.

Harrison, W., Magel, K., Kluczny, R., and DeKock, A., "Applying Software Complexity Metrics to Program Maintenance", *IEEE Computer*, Volume 15, Number 9 (September 1982), pp 65-79.

Lientz, B.P., Swanson, E.B., and Tompkins, G.E., "Characteristics of Application Software Maintenance", *Communications of the ACM*, Volume 21, Number 6 (July 1978) pp 466-471.

Lientz, B.P., and Swanson, E.B., *Software Maintenance Management*, Addison-Wesley, Reading, Mass., 1980.

Lientz, B.P., and Swanson, E.B., "Problems in Application Software Maintenance", *Communications of the ACM*, Volume 24, Number 11 (November 1981), pp 763-769.

Parikh, G., and Zvegintzov, N., *Tutorial on Software Maintenance*, IEEE Computer Society Press, Silver Spring, Maryland, 1983.

Vessy, I. and Weber, R., "Some Factors Affecting Program Repair Maintenance: An Empirical Study", *Communications of the ACM*, Volume 26, Number 2 (February 1983), pp 128-134.

Weiser, M., "Programmers Use Slices When Debugging", *Communications of the ACM*, Volume 25, Number 7 (July 1982), pp 446-452.