Software Engineering:
Practice and Experience

Proceedings of the Second Software
Engineering Conference

4–6 June 1984
Nice, France

Optimizing the utilization of human resources: a framework for research

A. BAILEY, V. BASILI and F. YOUSSEFI
Department of Computer Science, University of Maryland, College Park, MA 20742, U.S.A.

ABSTRACT

With the continuous increase in the size of software development teams, optimizing the utilization of human resources has become a growing concern in the software community. This paper proposes a model for assessing and improving human resource utilization in software development organizations. The process of setting up these organizations is described as an iterative process consisting of three steps: developing the structure, establishing the communications flow and determining the availability of human resources. With appropriate metrics introduced for each of these steps, the model serves as a framework for recording and periodic assessment of organizational changes throughout the software development life cycle.

Keywords: software management, software management metrics, human factors.

1. Introduction

Since people develop software, any problem relating to human resources would have a direct impact on the cost of software development. In recent years, we have been witnessing a continuous increase in the size of software development teams, sometimes in excess of a few hundred people. The software community has become more and more aware of the importance of optimizing utilization of human resources.

For many years, problems of manpower management have received much attention both in the academic and managerial literature [Piskor 76]. Management research studies have indicated that size, technology and environment have an impact on the organizational structure and vice versa [Aldrich 72; Child 72; Laurence 76]. These studies have also shown that each of these factors affects the organizational efficiency and thus should be given consideration. Few of these management techniques have been suggested to improve planning, organizing, staffing, directing and controlling of human resources within a software organization.

The chief programmer team proposed by [Mills 71] and implemented by [Baker 72], and the egoless programming team proposed by [Weinberg 71] have been widely used for organizing teams. Shneiderman discussed the advantages and disadvantages of these teams [Shneiderman 80]. Basili and Reiter found relationships between the size of a programming group and several software metrics [Basili 79]. Rogers suggested substituting network analysis field work to understand the effects of group structures [Rogers 76]. Scott and Simmons concluded in their experiments [Scott 75] that productivity is affected by the organizational structure and the communications requirements. The Thayer and Lehman survey [Thayer 79] demonstrated the necessity for the improvement of planning and communications, delegation of authority, assignment of responsibility and documentation of decisions. Furthermore, Mantei investigated the effects of various management structures and communication channels on performing programming tasks [Mantei 81].

A major source of insight when analysing the utilization of human resources within a software development

project is a record of organizational changes throughout the software development life-cycle. The objective of this paper is to describe a model which can serve as a framework for periodic assessment of human resource utilization. By collecting organizational data at each phase of the software development life-cycle, the model can show where changes were made, what kinds of changes were made and the effects of these changes on the overall utilization of human resources.

The model provides metrics for recording the history of organizational changes in terms of goals, structures, communications flow, resource allocation, etc. to promote visibility. By monitoring these changes, managers can determine whether their original organizational plan was perceived correctly and if not, they can learn where problem areas and strengths lie, why these conditions exist and what can be done to improve conditions where necessary. An entire software organization, or any part thereof, can be analysed on the basis of the presence or absence of conditions which contribute to its effectiveness.

We define the process of setting up a software development organization and identify the steps entailed in the process. These steps include: developing the structure, establishing the communications flow and determining the availability of human resources. A set of factors effecting each step and their interrelationships are defined and built into this model. The model exhibits certain behaviour which are discussed and an emphasis is made on its iterative nature. An example is given to illustrate the model.

2. The process of building a software development organization

Traditionally, a software organizational manager is given the responsibility to produce a software product based upon a set of requirements. He classifies these requirements into a set of tasks. Each task is then assigned to a team of individuals who are capable of performing on that task. For example, the analysis and design is assigned to software analysts, the coding to programmers, the quality control to quality assurance experts, etc. At this point, the manager feels he has done what he can and hopes that tasks are completed as scheduled.

The traditional ways of developing software organizations are in most cases *ad hoc*. There is often a lack of structured techniques to design and implement these organizations. However, upon closer examination, we believe the process follows certain necessary steps: tasks are identified, communication flows are established and availability of human resources is assessed. It is along these lines that we have developed the effectiveness model to be used in assessing human resource utilization.

3. The effectiveness model

The effectiveness model consists of three components, each receiving input from the previous component and providing results to the next. The model itself takes software goals as inputs and generates a set of effectiveness indicators as outputs. These indicators can further be analysed to assess the degree of optimization of human resource utilization.

To begin with, a set of requirements are derived from user's goals and expectations. These goals may include satisfying certain boundaries on cost and/or schedule, attaining certain performance criteria such as high reliability, fast response time or portability, or performing configuration management of the software.

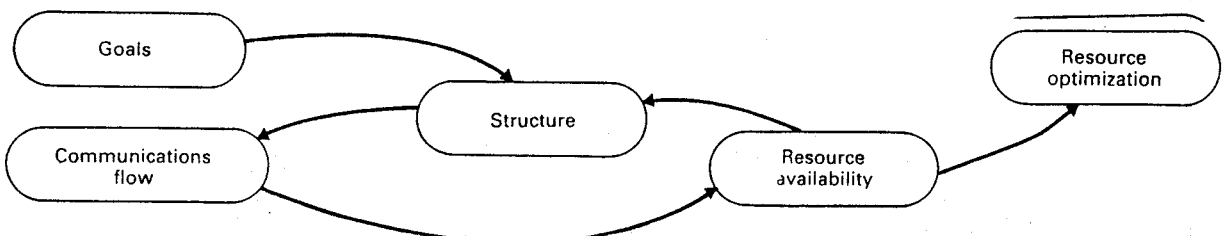
These requirements are then classified into a set of tasks which are built into a structure. A possible structure may consist of tasks such as building the data base, setting up the communications network and controlling quality assurance and configuration management.

Each of these tasks will be designated as a node. The location of these nodes within the structure bears a relationship to the success of the project. The nodes are a means of delineating tasks or functions. There often exists an interdependency among tasks. Sometimes, a task may need information from another task for completion. These functional dependencies cause certain tasks to overlap each other. Naturally, since people have to work together as a team to get these tasks done, it is imperative that ease of communication flow be established within the structure.

It is only after the structure and communication flow are investigated that the manager should begin to concern himself with gathering human resources needed to complete the tasks.

Figure 1 shows the process. Note that the process is iterative, i.e. a change in goals would result in a change in the structure and in turn in a change in the communications flow, all of which would have a bearing on the availability of human resources. Furthermore, any change in either component would affect another component so that if availability of human resources changes and new people are hired into the organization, then the structure as well as the communications flow has to be re-analysed and changed accordingly. If the goals of the software project originally envisioned are revised due to a large turnover, the circle will be reinitialized with new goals. The proposed model, being iterative, makes provision for such changes.

The following example is used throughout the paper to illustrate the workings of the effectiveness model. An organization wants to build a network of computer facilities to serve users at its dozen or so departments in processing on-line transactions. Each department has its own computer facility, geographically distant from the



others. In most cases, transactions processed by each department are unique to that department and must be dealt with in a separate analysis and implementation process. A data base is distributed among the sites of the network and will be fully replicated, meaning that a complete copy exists at each site of the network. Since users make occasional updates to remote copies of the data base, there are strict requirements for maintaining consistent copies of the data bases. It is also important to guard against site failures and build mechanisms for prompt recovery. In addition, there is a need for quality control, general organizational accounting and configuration management.

These are the goals of the example software organization. These goals, as they are manifested in the requirements document, will be translated to a structure and then carried through each step of the process in examples given to illustrate the effectiveness model.

3.1. Structure

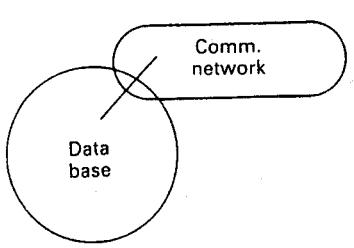
Structure relates to the method by which a software organization organizes, distributes and manages the relevant tasks in the production of software. These tasks are derived from the requirements document and first grouped together as primary functions. At this time, each function or task is represented as a circle network; creating and maintaining the distributed data base, etc. are examples of primary functions in the illustrative organization. These functions then need to be broken down into lower-level functions, i.e. an expansion of major nodes.

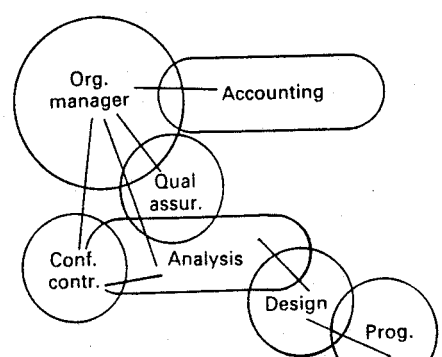
The structure of a software development organization primarily depends upon the type of project involved. When the project is a very high-priority, schedule-driven task, an open loop structure develops where the success of the project depends solely on the manager's experience and the availability of critical resources. Smaller organizations that are newly formed usually possess an open loop structure. As the organization grows to assume larger responsibilities, it faces more complex and product-oriented communication problems. Most communication in large organizations is of closed loop form, one that relies on the team approach where the balance between cost, schedule and performance requires the presence of a well-defined organizational structure. The methodology proposed in this paper models only closed loop management.

3.1.1. *Factors affecting structure*

Factors affecting structure are the size and grouping within the project, as well as the flexibility of work conditions and recording history of changes. All examples given in the following sections stem from the goals of our representative organization. These factors are discussed below.

- Size of the structure. Requirements determine the complexity of tasks to be accomplished. As tasks increase in complexity so does the size of the organization, resulting in a proportional increase in the size of the structure. The complexity of a task is measured in terms of level of effort. As an example, suppose a requirement for building our communications network originally calls for transmission of data using one type of protocol and is later expanded to many protocols. The node associated with this task must expand to accommodate the increase in task complexity which would impact the size of the overall structure.
- Grouping of the structure. A typical software organization is composed of tasks that are highly interdependent. These interdependencies have to be accounted for in the structure. Tasks that are interdependent should be located closer together, i.e. as the dependencies of tasks increase, their distances decrease. For example, the tasks of building the communications network and the creation of the distributed data bases are interdependent, meaning the decisions made regarding network topology will directly have an impact on decisions made about the data base. The nodes associated with these tasks are positioned close together. Similarly, quality control and accounting tasks are positioned near the top level to best support the manager. On the other hand, configuration management task requires structural interfaces on a par with lower levels. The location of each task as a node within the structure also depends on the amount of decision-making involved in that task. The more critical the decision-making, the higher the level of the node associated with that task. Figure 2 demonstrates one particular grouping for our example organization. Note that many more interdependencies exist, but what is shown serves as simple grouping to illustrate its rationale in terms of the degree of communication and decision-making





responsibilities distributed in the organization. High-level decision-making tasks are here denoted by circles. When they overlap, it indicates their interdependencies. Note that the quality assurance task is positioned close to the organizational management task; if not, communication channels would be long and unnecessarily complicated, which could result in decision-making delays in critical periods.

- Flexibility of the structure. Thus far, the main issue of concern has been to build the skeleton of the organization, its structure. Since a project, as it proceeds, may go through changes, either due to a modification of goals or a need for additional human resources, the structure should allow flexibility for shifting human resources as required.

  If the underlying environment of the structure is standardized by the parent organization, the structure itself can be modified with greater ease and flexibility. A standardized environment refers to minimizing differences across software teams. Some factors affecting the flexibility of a structure may be: availability of tools where some teams have access to more tools than others, e.g. computing capabilities, secretarial services, etc.; compensation where only members of certain teams are given flexitime, overtime, etc.

  In our example organization, suppose the programming task is decomposed into teams, each responsible for one department. Later, a few departments are merged and personnel have to be transferred. If the operating environments of these departments are not standardized, then programmers may be reluctant to transfer. With some planning and expenditures, the effect of these factors can be minimized resulting in long-term gains for the software organization.

- Tools for recording and measuring structural changes. The utility of documenting changes in any system is a well-known premise. The effectiveness model also takes advantage of documenting or recording the history of the structural changes of the software organization. This can be valuable, both from an internal perspective when managers involved in the organization evaluate its evolving structure, and from an external one when evaluation is made by auditors or administrators. And as far as evaluating the people involved in the organization is concerned, such reporting can possibly produce answers to many intangible questions that are frequently asked: What happens when experience is lost when a person is moved from a project? What happens when a person is moved to a project where he is not an expert? Who do we reward for a job well done? If the organizational structure abruptly changes, how are the people with the best track record found and called to the rescue?

  A recorded history of the structure can provide a manager with valuable information about how requirements were met and people's productivity, as well as bottlenecks and problem areas. There have been numerous cases where a task was accomplished exceptionally well by an individual or a team, but due to organizational changes at a later stage, this individual or team was placed at a level where potentials for performance were under-utilized. With recorded history of structural changes, it is possible to optimize human resource utilization.

3.1.2. *Metrics for structure*

Going back to our example organization, suppose the tasks to be accomplished $t1$, $t2$, ... $t8$ are initially identified as establishing the communication network, building the distributed data base, etc. as shown in Fig. 2. All tasks are considered to be of equal complexity, $W1$. $WT1$ is the workload of the organization. The interdependent tasks are grouped into classes $C1$, $C2$ and $C3$.

$$WT1 = W1 * (t1 + t2 + t3 + \ldots + t8)$$
$$C1 = (t1, t2)$$
$$C2 = ((t3, t4), (t4, t5), (t5, t6), (t6, t7))$$
$$C3 = (t5, t8)$$

The levels in which the nodes are positioned are determined by the amount of decision-making affecting the organization. The structure derived is $S1$ (see Fig. 3a). Later, a task $t9$ is added, complex enough to warrant a change in the structure ($t9$ has complexity $W2$). A new class $C4$ is formed to indicate that $t3$ and $t9$ are interdependent.

$$WT2 = W1 * (t1 + t2 + \ldots + t8) + W2 * t9$$
$$C4 = (t3, t8)$$

The new structure $S2$ is shown in Fig. 3b.

3.2. COMMUNICATIONS FLOW (Fig. 4)

Once the structure of a software organization is established, the channels of communication must be investigated, since the effects of poor communications flow can be devastating to the organization. Communications flow can be defined as the transformation of a request to a response:

$$Response_i = T(Request_j)$$

The request originates at node j and the associated response is generated by node i. The relationship between the requesting node j and the responding node i is characterized by the following coupling:

- vertical: communication involves two nodes from different levels, i.e. $Lj > Li$ or $Lj < Li$ where L is the level of nodes i and j.
- lateral: communication involves two nodes at the same level, i.e. $Li = Lj$.
- feedback: communication takes place within the same node, i.e. $i = j$.

The information communicated in a software organization is classified as: task assignments, problem areas, possible solutions and task completions. These classes of communication normally display inherent properties of coupling. Communicating task assignments is usually assumed to be vertical top-down, e.g. the higher-level node of design communicates specifications to the subordinating node of programming. Communicating problem areas usually displays a vertical bottom-up coupling. It is important that problems are communicated to higher-level nodes, since the higher a level, the more visible becomes the impact of a problem to the organization. Communicating solutions has an inverse

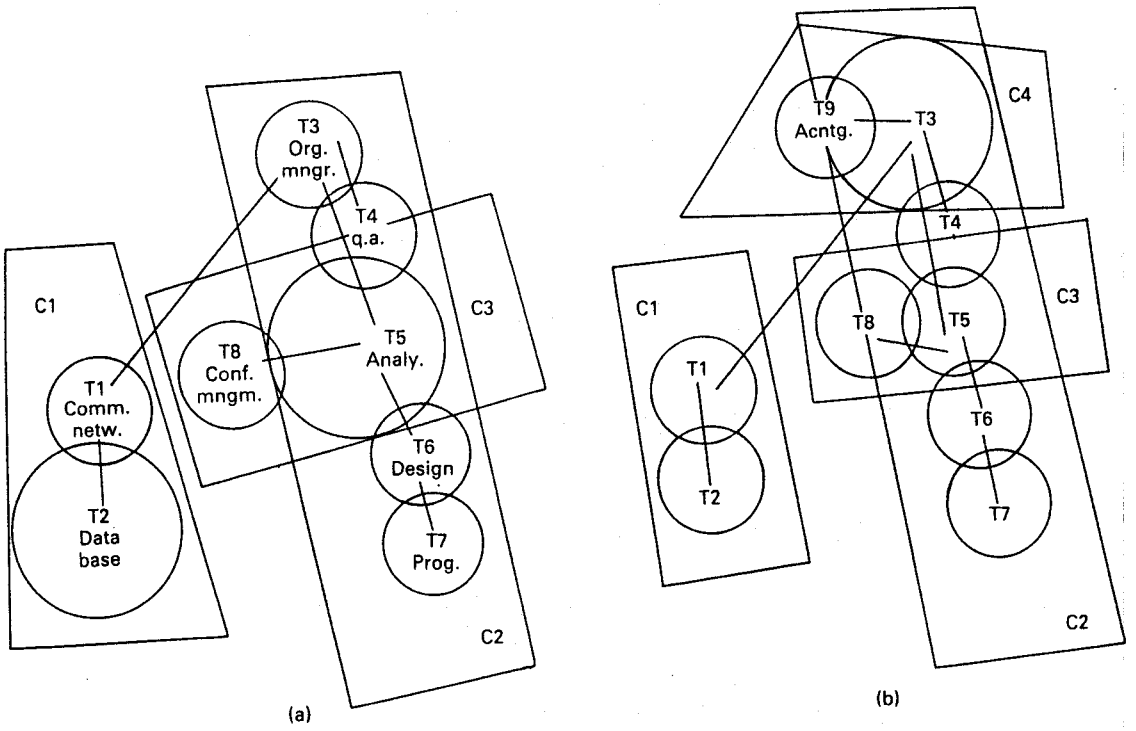


Fig. 3.—Two examples of structure.

coupling to that of problems, namely vertical top-down. Communicating task completions is vertical bottom-up as well as lateral, e.g. configuration manager reports a successful review up to the organizational manager and to the analysis team directly involved in the review, see Fig. 2.

There are three types of communication nodes: input node, output node and intermediate node. An input node receives a request, an output node transmits a response whether or not a request is received. An intermediate node transmits a response to a request it has just received. An organizational structure is characterized by a graph, where incoming and outgoing edges denote communications flow, see Fig. 2. Of course, in reality, there are many more edges than depicted but the simple example shows how all nodes are input and output nodes, and that some nodes, such as the data base node and the programming nodes, cannot be intermediate nodes. Intermediate nodes, by nature, are transmitters of information. Inevitably, information may get more or less detailed as it is transmitted, but a change in the content of information is considered to be a transmission malfunctioning. For example, an organizational manager may assign a task to the network team, who as an intermediate node has to pass it down to the data base team, but somehow the person in charge forgets or misinterprets the command. So, it is important that these nodes be good translators.

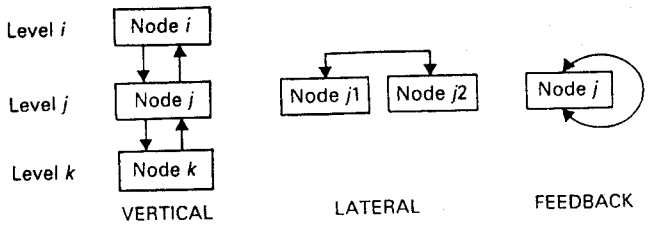


Fig. 4.—Types of communications flow.

3.2.1. *Factors affecting communications flow*

Many complex organizations subsist under loss and inefficient operations because of poorly designed or ignored communications flow. Once communications flow of an organization is defined, understanding factors that affect the flow can be instrumental in evaluating inefficiencies of the organizations and setting up measures for improvement.

The factors affecting communications flow are defined as:

- Authority: who is authorized to assign a task?
- Problem confinement: which problems fall under jurisdiction of a task at a certain level?
- Problem refinement: when a problem occurs what is its final impact to the organization?

In a software organization, there must exist a document that defines the lines of authority, so that each person knows exactly what his assigned tasks are and who will be notified of task completion. An unnecessary amount of confusion arises when the lines of authority are not well defined. What happens as far as communications flow is concerned is that communication will not adhere to its predefined patterns of coupling. For example, task assignment may be lateral rather than vertical top-down when a member of one programming team tells a member of another what to do. One task may be assigned vertically top-down by two output nodes to the same input node.

Problem confinement refers to the identification of the task that has jurisdiction over a particular problem and its solution. In an organization, we are often witness of such inefficient communications flow as a person assigned to a programming task who is put under pressure to solve a problem dealing with bad design. These mishaps can be avoided if those who must become involved in solving a particular problem are identified, as well as those who must not be burdened with this added responsibility.

An organization becomes aware of the impact of a problem through problem refinement. A problem is refined when it is translated up through the level of an organization; at each level its solution and impact become more visible. Many organizations suffer because problems are not reported up, either due to fear of reprisal or underestimation of the problem's impact by a lower-level team that is inadequately equipped to make such a judgement. It is important to establish communications flow to force all problems to be reported up by enforcing correct patterns of coupling for communicating problems.

As communication is passed down to lower-level nodes, it gets more detailed. As it is passed back up again, it gets less detailed. The reason for this is that higher-level nodes deal with cost and schedule problems, and lower-level nodes deal with performance problems. For managers, technical issues get translated into issues pertaining to resources; what is needed to get the task done? For lower-level nodes, requirements are eventually detailed into how to do them.

To ensure that the least amount of information is lost due to translation, the following measures can be adopted. First, parallel communication lines are established to reduce noise in translation. In the example organization of Fig. 2, configuration and quality control managers are two separate paths with alternative feedbacks to the top. Second, a two-level review process is established where problems are reported up two levels for solution approval. This allows auditing of a solution given by a team that is either distorting the truth or is zealous about a solution that is truly inadequate. For example, if problems occur in the design task, the analysis and quality control experts are notified (see Fig. 2). Those problems without a satisfactory solution will proceed up in levels until solved plus one level. The propagation of solutions to the extra level prevents an inadequate solution, given say by quality assurance, being accepted as a final one. The two-level review process minimizes the chance of optimistic middle managers overlooking potential problems.

When structure changes, communications flow can be affected in two ways: as nodes are expanded or compressed, the nature of communications flow remains the same but the quantity changes, as new nodes are added or deleted, there is a change in both the nature and quantity of communications flow.

3.2.2. *Metrics for communications flow*

Communications flow can be evaluated by recording instances of communication, *CFi*. Given a structure *Si* with *T*1, ... *Tn*, an instance of a communication *CFi* can be defined as:

$$Response_i = T(Request_j)$$

where i and j are two different nodes. The transformation can be said to be the requirements imposed on a request. The communication flow *CF* is vertical if $Li \neq Lj$, it is lateral if $Li = Lj$, and it is feedback if $i = j$, where *Li* and *Lj* are levels of nodes i and j.

The overall communications flow then would be:

$$CF = CF1, CF2, \ldots$$

3.3. Availability of human resources

Often, a software organization having received a large contract begins a hiring binge. At this time, it appears that securing people to realize the objectives set forth in the contract is the best way to proceed. However, if the structure and communications flow are already designed, as presented here in the effectiveness model, then optimum human resource utilization can be realized by employing people most competent to do the required tasks.

Each person has unique talents. There must exist a match between these talents and the skills required to perform the assigned tasks. Deciding whether or not match exists becomes much easier if tasks are clearly identified, and requirements for decision-making and communication skills are known for each task.

The effectiveness model proposes a mechanism to evaluate each available person's skills based upon the requirements of the software project. Skills can be assessed throughout common methods such as interviews and questionnaires. But, since the structure of the organization is now well defined, the questions asked are much more goal-oriented, enabling the organization to assign people to tasks where they will be utilized optimally.

3.3.1. *Metrics for resource availability*

For tasks $T = T1, T2, \ldots Tn$, we have available a set of people P and for each person we define

$$Pi — (\%a1 * T1) + \ldots + (\%an * Tn)$$

where *Pi* is the sum of percentages of tasks each person is skilled and willing to do. Availability of resources is:

$$A = p1 + p2 + \ldots + pm$$

In the example organization, suppose:

$$p1 = (\%20 * t1) + (\%80 * t2)$$
$$p2 = (\%40 * t2) + (\%60 * t7)$$

where *T*1 is establishing communications network, *T*2 is creating the data base and *T*7 is programming (see Fig. 3a). Notice that %120 of *T*2 must be regarded as %100 of *T*2 with an extra %20 of effort to be utilized elsewhere. If $A < WT$, then there is a lack of manpower to do the task.

4. The nature of the process

Through many years of experience and observation, the authors conclude that the process of developing a software organization follows certain principles. The results of the effectiveness model will be evaluated based upon these principles. The model proposed here makes the following assumptions:

- The process is iterative. The process is iterative for two reasons: there are always changes in the organization either relating to goals, structures, communication flows or availability of resources; and since software development has a cycle, new tasks have to be accomplished resulting in expansions of nodes within the structure, the communications flow has to be re-established incorporating these new tasks and, finally, the availability of people with new skills to complete these tasks has to

be assessed. In both cases, the change of one component would affect another.

- The process is sequential. There is always a sequence to managing for effectiveness, i.e. there is an order in which the effectiveness of various steps must be attended, beginning with the identification of goals and proceeding with each step of the process in order. Effectiveness analysis will provide an examination of an organization to determine whether or not it is being managed in a proper sequence. If not, it will indicate areas in which effectiveness is being 'short-circuited' and will make recommendations for consideration.
- The process is dynamically balanced. Effectiveness organizations are characterized by a balance between the degree of well-defined structure, communications flow and availability of resources. Furthermore, the balance must be dynamic as opposed to static, i.e. the managers must work continuously to create conditions for productive movement.

5. The results of the analysis

The results of this model can be formulated as:

$$R1 = (WT(t1), CF(t1), A(t1))$$
$$R2 = (WT(t2), CF(t2), A(t2))$$
$$R3 = (WT(t3), CF(t3), A(t3))$$

where $R1$, $R2$ and $R3$ are results taken by measuring the components WT, CF and A at various points in time $t1$, $t2$, $t3$, ... tn. Time may be chosen by: phases of a software project beginning with the requirements phase and ending with maintenance; any change in the three phases of the process; and any predefined time period such as quarterly or semiannually. The results obtained at various times can then be compared in order to assess overall effectiveness.

6. Conclusion

This paper summarized a model for developing and evaluating a software organization's effectiveness in terms of its human resources. It promotes recording of the history of the organization as it evolves, indicating possible problem areas and strengths. For the first time, managers can use this recorded evidence to compare how software business was first conducted when the organization was created with how it does business now, and they can learn from any mistakes made in the past.

This model is based upon many years of observations and represents only one viewpoint. There may be many equally valid viewpoints. This representation is specifically designed to support objective metrication and as such could be used to measure the total organizational structure, communications flow and resource availability in comparative analyses. It is a baseline for further detailed examination of the various aspects of a software organization with respect to its effectiveness.

REFERENCES

[Aldrich 72] H. B. ALDRICH: *Technology and organizational structure: A reexamination of the findings of the Aston group*; Administrative Science Quarterly, **17**, 26–41, 1972.

[Baker 72] F. T. BAKER: *Chief programmer team management of production programming*; IBM Systems Journal, **1**, 57–73, 1972.

[Basili 79] V. R. BASILI and R. W. REITER, JR: *The investigation of human factors in software development*; Computer, **12**, 21–38, Dec. 1979.

[Child 72] J. CHILD and R. MANSFIELD: *Technology, size and organizational structure*; Sociology, **6**, 369–393, 1972.

[Laurence 76] P. R. LAURENCE and J. W. LORSCH: Organization and Environment; 1976, Harvard University Press.

[Mantei 81] M. MANTEI: *The effect of programming team structures on programming tasks*; Communications of the ACM, 106–113, Mar. 1981.

[Mills 71] H. D. MILLS: *Chief programmer teams: principles and procedures*; IBM Rep. FSC, 71-5108, IBM FSD, Gaithersburg, MD, U.S.A., 1971.

[Piskor 76] G. PISKOR: *Bibliographic survey of quantitative approaches to manpower planning*; Working Paper 833-76, Sloan School of Management, NIT, 1976.

[Rogers 76] E. M. ROGERS and R. AGRAWALA-ROGERS: Communication in Organizations; 1976, Free Press, New York.

[Scott 75] R. F. SCOTT and D. B. SIMMONS: *Predicting programming group productivity – a communications model*; IEEE Transactions on Software Engineering, **1** (4), 411–414, Dec. 1975.

[Shneiderman 80] B. SHNEIDERMAN: Software Psychology; 1980, Winthrop, Cambridge, Mass.

[Thayer 79] R. H. THAYER and J. H. LEHMAN: *Software engineering project management: a survey concerning U.S. aerospace industry management of software development projects*; IEEE Tutorial on Software Management, 340–354, 1979.

[Weinberg 71] G. WEINBERG: The Psychology of Computer Programming; 1971, Van Nostrand Reinhold, New York.