Characterizing and Assessing a Large-Scale Software Maintenance Organization*

Lionel Briand
CRIM
1801 McGill College Av.
Montréal (Quebec)
H3A 2N4, Canada
Lionel.Briand@crim.ca

Walcélio Melo, Carolyn Seaman and Victor Basili
Computer Science Department
Institute for Advanced Computer Studies
University of Maryland
College Park, MD, 20742
{melo | carolyns | basili}@cs.umd.edu

Abstract

One important component of a software process is the organizational context in which the process is enacted. This component is often missing or incomplete in current process modeling approaches. One technique for modeling this perspective is the Actor-Dependency (AD) Model. This paper reports on a case study which used this approach to analyze and assess a large software maintenance organization. Our goal was to identify the approach's strengths and weaknesses while providing practical recommendations for improvement and research directions. The AD model was found to be very useful in capturing the important properties of the organizational context of the maintenance process, and aided in the understanding of the flaws found in this process. However, a number of opportunities for extending and improving the AD model were identified. Among others, there is a need to incorporate quantitative information to complement the qualitative model.

1. Introduction

It has now been recognized that, in order to improve the quality of software products, it is necessary to enhance the quality of the software processes used to develop and maintain them. This requires the understanding and modeling of these processes in order to be able to analyze and assess them. Following the example of other engineering disciplines, where empirical approaches to management have been successfully applied, several methodologies have been defined for allowing the characterization and incremental improvement of software processes [Basili, 1989; Lanphar, 1990]. The modeling of software development and maintenance processes is a necessary component of these approaches. A number of modeling techniques have been developed, e.g., [Lott and Rombach, 1993; Finkelstein et al, 1994; Melo and Belkhatir, 1994].

What has received less attention in the literature is the need to model the organizational context in which a development process executes. It is not possible to fully understand and analyze such process issues as information flow, division of work, and coordination without including the organizational context in the analysis. Organizational context refers to characteristics of relationships between process participants. Such relationships include, among others, the management hierarchy, the structure of ad hoc working groups, and seating arrangements. Some process modeling approaches attempt to include mechanisms in their formalisms to deal with organizational structure [Curtis et. al., 1992], but not to any great level of detail. Some formalisms have specifically focused on organizational modeling [Rein, 1992; Benus, 1994], but these lack the mechanisms and flexibility necessary for quantitative analysis (discussed later).

One approach to organization and process modeling appears particularly promising [Yu and Myopoulos, 1994]. This approach is very new (it was presented at last year's ICSE) and thus lacks significant validation through use. One goal of this paper is to report an early experience with this promising new approach.

Consistent with the philosophy presented above, [Briand et. al., 1994] have developed an auditing process specifically aimed at software maintenance processes and organizations.

*During this work the authors were, in part, supported by the following grants: Briand by NSF grant 01-5-24845 and CRIM, Melo by UMIACS, Seaman by IBM-Toronto grant 91-1-20011, and Basili by NASA grant NSG-5123.

Paper published in the Proc. of the 17th International Conference on Software Engineering, Seattle, WA, April 1995.

Such an approach requires, to a certain level of detail, the modeling of processes and organizations. In this context, the Actor-Dependency modeling technique [Yu and Mylopoulos, 1994] mentioned above appeared to be suitable because of its capability to capture numerous kinds of constraints and dependencies frequently encountered in complex organizations. In addition, this technique proved in practice to be intuitive and to facilitate communication with the maintenance staff of the studied organization. This paper reports one experience of using the Actor-Dependency technique to help analyze a large software maintenance organization. We evaluate the approach's strengths and weaknesses while providing practical recommendations for improvement. In Section 2, we briefly describe the Actor-Dependency modeling approach and one of its extensions that we have used in our study. Section 3 presents the case study we conducted. In Section 4 we evaluate the advantages and weaknesses of the AD approach. Finally, in Section 5, we present a number of suggestions for future work, both with the AD model and software organization modeling in general.

2. The Actor-Dependency Modeling Approach

The most important characteristic of the modeling approach presented by Yu et al, for our purposes, is its capability to fully represent the organizational context in which a development process is performed. This language provides a basic organizational model with several enhancements, only one of which we will describe here. The basic Actor-Dependency model represents an organizational structure as a network of dependencies among organizational entities, or actors. The enhancement which we have used, called the Agent-Role-Position (ARP) model, provides a useful decomposition of the actors themselves. These two representations are described briefly in the following sections. For a more detailed description, see [Yu and Mylopoulos, 1993].

2.1. The basic Actor-Dependency (AD) model

In this model, an organization is described as a network of interdependencies among active organizational entities, i.e., actors. A node in such a network represents an organizational actor, and a link indicates a dependency between two actors. Examples of actors are: someone who inspects units, a project manager, or the person who gives authorization for final shipment. Documents to be produced, goals to be achieved, and tasks to be performed are examples of dependencies between actors. When an actor, A1, depends on A2, through a dependency D1, it means that A1 cannot achieve, or cannot efficiently achieve, its goals if A2 is not able or willing to fulfill its commitment to D1. The AD model provides four types of dependencies between actors:

- In a *goal dependency*, an actor (the depender) depends on another actor (the dependee) to achieve a certain goal or state, or fulfill a certain condition (the dependum). The depender does not specify how the dependee should do this. A fully built configuration, a completed quality assessment, or 90% test coverage of a software component might be examples of goal dependencies if no specific procedures are provided to the dependee(s).
- In a *task dependency*, the depender relies on the dependee to perform some task. This is very similar to a goal dependency, except that the depender specifies how the task is to be performed by the dependee, without making the goal to be achieved by the task explicit. Unit inspections are examples of task dependencies if specific standard procedures are to be followed.
- In a *resource dependency*, the depender relies on the dependee for the availability of an entity (physical or informational). Software artifacts (e.g. designs, source code, binary code), software tools, and any kind of computational resources are examples of resource dependencies.
- A *soft-goal dependency* is similar to a goal dependency, except that the goal to be achieved is not sharply defined, but requires clarification between depender and dependee. The criteria used to judge whether or not the goal has been achieved is uncertain. Soft-goals are used to capture informal concepts which cannot be expressed as precisely defined conditions, as are goal dependencies. High product quality, user-friendliness, and user satisfaction are common examples of soft-goals because in most environments, they are not precisely defined.

Three different categories of dependencies can be established based on degree of criticality:

- *Open dependency*: the depender's goals should not be significantly affected if the dependee does not fulfill his or her commitment.
- *Committed dependency*: some planned course of action, related to some goal(s) of the depender, will fail if the dependee fails to provide what he or she has committed to.
- *Critical dependency*: failure of the dependee to fulfill his or her commitment would result in the failure of all known courses of action towards the achievement of some goal(s) of the depender.

The concepts of open, committed, and critical dependencies can be used to help understand actors' vulnerabilities and associated risks. In addition, we can identify ways in which actors alleviate this risk. A commitment is said to be:

- *enforceable* if the depender can cause some goal of the dependee to fail.
- *assured* if there is evidence that the dependee has an interest in delivering the dependum.
- *insured* if the depender can find alternative ways to have his or her dependum delivered.

In summary, a dependency is characterized by three attributes: type, level of criticality, and its associated risk-management mechanisms. The type (resource, soft-goal, goal, and task) represents the issue captured by the dependency, while the level of criticality indicates how important the dependency is to the depender. Risk-management mechanisms allow the depender to reduce the vulnerability associated with a dependency.

Figure 1 shows a simple example of an AD model. A Manager oversees a Tester and a Developer. The Manager depends on the Tester to test. This is a task dependency because there is a defined set of procedures that the Tester must follow. In contrast, the Manager also depends on the Developer to develop, but the Developer has complete freedom to follow whatever process he or she wishes, so this is expressed as a goal dependency. Both the Tester and the Developer depend on the Manager for positive evaluations, where there are specific criteria to define "positive", thus these are goal dependencies. The Tester depends on the Developer to provide the code to be tested (a resource), while the Developer depends on the Tester to test the code well (good coverage). Assuming that there are no defined criteria for "good" coverage, this is a soft-goal dependency.

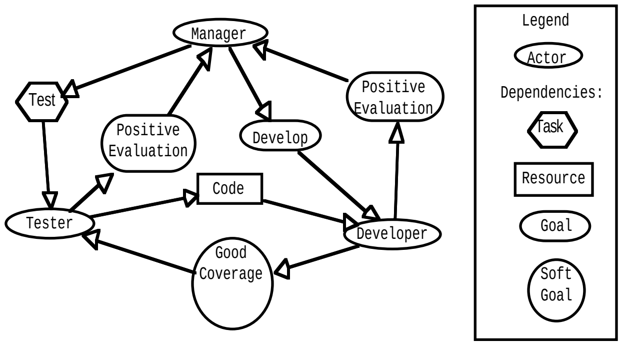

Figure 1: A simple example of an AD model

2.2. The Agent-Role-Position (ARP) decomposition

In the previous section, what we referred to as an actor is in fact a composite notion that can be refined in several ways to provide different views of the organization. *Agents*, *roles*, and *positions* are three possible specializations of the notion of actor which are related as follows:

- an agent occupies one or more positions
- an agent plays one or more roles.
- a position can cover different roles in different contexts

Figure 2 shows an example of an actor decomposition. These three types of specialization are useful in several ways. They can be used to represent the organization at different levels of detail. At a very high level, one might use only unspecialized actors. Positions provide more detail, but still provide a high-level view. Roles provide yet more detail, and the use of agents allows the modeler to specify even specific individuals. The ARP decomposition could be especially useful when extending the use of AD models to quantitative analysis. This is described in more detail in Section 5.3.3. A Case Study using the AD Model

3.1. Background

Before describing our results, some background is necessary in order for the reader to understand the analysis which follows. We will first describe the development environment which serves as our context. Second, the auditing process used in the case study will be described.

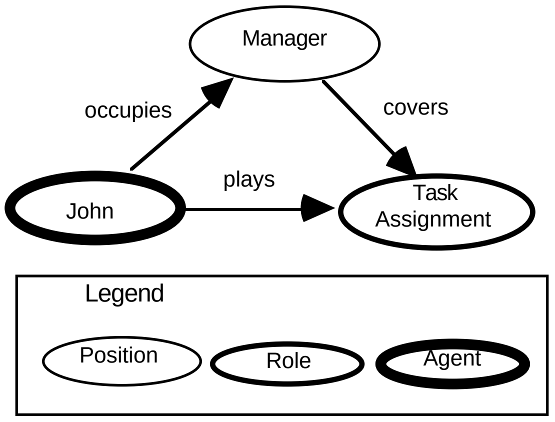

Figure 2. Associated Agent, Position, and Role

3.1.1. The Studied Maintenance Organization

This study took place in the Flight Dynamics Division (FDD) of the NASA Goddard Space Flight Center (GSFC). Over one hundred software systems for the control and prediction of satellite orbits, trajectories and attitude, totalling about 3.5 million lines of code, are maintained. Many of these systems are maintained over a very long period of time, and regularly produce new releases. About 80 people are involved in the maintenance of these systems. This study focused on three systems in particular, which ranged from 156 to 260 thousand lines of FORTRAN code, and from 7 to 26 years of age.

Numerous communication, schedule, budget and technical problems arise with each release. This results in somewhat unstable change requirements all along the release process, a high turnover in some projects and difficulties in meeting deadlines. There was a need to study these phenomena.

More precisely, our framework for this study is the Software Engineering Laboratory (SEL). The SEL is a joint venture between the University of Maryland, CSC and NASA. The SEL is an organization aimed at improving NASA-FDD software development processes based on measurement and empirical analysis**.** Recently, responding to the growing cost of software maintenance at NASA-FDD, the SEL has initiated a program aimed at characterizing, evaluating and improving its maintenance processes. The first step in this direction was a set of studies conducted using the auditing technique described below.

3.1.2. The Maintenance Process Auditing Methodology

In [Briand et. al., 1994], a qualitative and inductive methodology has been proposed in order to characterize and audit software maintenance processes and organizations and thereby identify their specific problems and needs. This methodology encompasses a set of procedures which attempts to determine causal links between maintenance problems and flaws in the maintenance organization and process. This allows for a set of concrete steps to be taken for maintenance quality and productivity improvement, based on a tangible understanding of the relevant maintenance issues in a particular maintenance environment. The steps of this methodology can be summarized as follows:

Step 1: Identify the organizational entities with which the maintenance team interacts and the organizational structure in which maintainers operate. In this step the distinct teams and their roles in a change process are identified. Information flows between actors should also be determined.

Step 2: Identify the phases involved in the creation of a new system release. Software artifacts produced and consumed by each phase must be identified. Actors responsible for producing and validating the output artifacts of each phase have to be identified and located in the organizational structure defined in Step 1.

Step 3: Identify the generic activities involved in each phase, i.e. decompose life-cycle phases to a lower level of granularity. Identify, for each low-level activity, its inputs and outputs and the actors responsible for them.

Step 4: Select one or several past releases for analysis in order to better understand process and organization flaws.

Step 5: Analyze the problems that occurred while performing the software changes in the selected releases in order to produced a causal analysis document. The knowledge and understanding acquired through steps 1-3 are necessary in order to understand, interpret and formalize the information described in the causal analysis document.

Step 6: Establish the frequency and consequences of problems due to flaws in the organizational structure and the maintenance process by analyzing the information gathered in Step 5.

Modeling the organizational context of the maintenance process was a very important step in the above analysis process. A model of the organization was necessary for communication with maintenance process participants. Gathering organizational information and building the model was critical to our understanding of the work environment and differences across projects. The model was also useful in checking the consistency and completeness of the maintenance process model. For example, the organizational model allowed us to determine whether or not all organizational actors had defined roles in the process model. During this preliminary study, the following requirements were identified for an optimal organizational modeling technique:

R1: The modeling methodology had to facilitate the detection of conflicts between organizational structures and goals. For example, inconsistencies between the expectations and intentions of interfacing actors seemed to be a promising area of investigation.

R2: We needed to capture many different types of relationships between actors. These included relationships that contributed to information flow, work flow, and fulfillment of goals. The explicit and comprehensive modeling of all types of relationships was necessary in this context.

R3: Different types of organizational entities had to be captured: individuals, their official position in the organizational structure, and their roles and activities in the maintenance process. This was important not only to be able to model at different levels of detail, but also to provide different views of the organization, each relaying different information.

R4: Links between the organization and the maintenance process model had to be represented explicitly.

R5: The notation had to aid in communication through intuitive concepts and graphical representation.

As a starting point, we decided to use the Actor-Dependency model introduced by Yu et al in order to reach these objectives. The AD model, as we shall see, meets many of our requirements.

In the next section, we provide the extended AD model of our maintenance organization where, for the sake of simplification, we use only positions (one possible specialization of actors) as vertices of the graph.

3.2. AD Organizational Model

The organizational model in Figure 3 is very complex despite important simplifications (e.g., agents and roles are not represented). This shows how intricate the network of dependencies in a large software maintenance organization can be. The lessons learned with respect to the maintenance organization are presented below and the approach's advantages and drawbacks are the focus of the next section.

The organizational model presented in Figure 3 was built using information from a variety of sources: we read maintenance standards release documents, interviewed people involved in the change and configuration management process, analyzed release management reports, and studied the official organization charts.

The model is by necessity incomplete. We have focused on those positions and activities which contribute to the maintenance process only. So there are many other actors in the NASA-FDD organization which do not appear in the AD graph. As well, we have aggregated some of the positions where appropriate. For example, Maintenance Management includes a large number of separate actors, but for the purposes of our analysis, they can be treated as an aggregate. Because only the primary dependencies are shown at this level of detail, nearly all of them are shown as critical. This issue will be discussed in more detail later. Below are listed the positions shown in the figure, and a short explanation of their specific roles:

Testers present acceptance test plans, perform acceptance test and provide change requests to the maintainers when necessary.
Users suggest, control and approve performed changes.
QA Engineer controls maintainers' work (e.g., conformance to standards), attends release meetings, and audits delivery packages.
Configuration Manager integrates updates into the system, coordinates the production and release of versions of the system, and provides tracking of change requests.
Maintenance management grants preliminary approvals of maintenance change requests and release definitions.
Maintainers: analyze changes, make recommendations, perform changes, perform unit and change validation testing after linking the modified units to the existing system, perform validation and regression testing after the system is recompiled by the *Configuration Manager.*
Process Analyst collects and analyzes data from all projects and packages data to be reused.
NASA Management is officially responsible for selecting software changes, gives official authorizations, and provides the budget.

The resulting organizational model was validated through use, within the context of the auditing methodology presented above. The modeling of the maintenance process, the release documents, and the causal analysis of maintenance problems allowed us to check the model for consistency and completeness.

3.3 Lessons Learned

Below are the main flaws that were found in the maintenance process and which we reported to the maintenance organization. In all cases, the flaws were uncovered, or at least better understood, by studying the AD model.

Task Leader

From our analysis, it appears that the Task Leader is a very central position. This is clearly illustrated in Figure 3. The centrality of the Task Leader gives rise to two possible problems: overloading of the person filling this position, and over-dependence of the project on this one position. Analysis of the Task Leader's role decomposition, especially in conjunction with quantitative analysis, would be helpful in determining the extent of these problems, and possible solutions.

Quality Assurance

Standards conformance and quality inspections were not perceived by the task leaders and maintainers as critical. They considered these processes mainly bureaucratic. This is reflected in the (non-)criticality symbols on the corresponding dependencies in Figure 3. This pointed out a weakness in the process and organization that could be remedied through more suitable inspection procedures and better definition and communication of quality needs.

Requirements

In Figure 3, Unambiguous requirements (a dependency between the Task Leader and the User) is not an enforceable soft-goal dependency since the users and maintainers (including the Task Leader) belong to two different management hierarchies. In other words, the Task Leader and User are so far removed from each other in the network of management dependencies that the Task Leader has no practical recourse for ensuring that the User provides unambiguous requirements. Note that the management dependencies are included in the AD model, but have been omitted from Figure 3 to simplify the diagram. Moreover, the fact that this dependency is a soft-goal and not a goal raises another issue: standards for defining unambiguous requirements should be defined and applied. The lack of such standards indicates that the organization is still immature in this area.

Data Collection

Process analysts attempt to collect data in order to evaluate and better predict the maintenance process. However, such a procedure is inherently difficult to enforce when maintainers do not clearly understand the benefits of such data collection, in terms of useful feedback. In terms of the AD model, the Process Analyst's dependency on the Maintainer is a vulnerability, with no reciprocal dependency to serve as a risk management mechanism available to reduce that vulnerability.

4. Evaluation of the AD Model

4.1. Advantages

The notions of enforcement and assurance, as well as the modeling of goal and soft-goal dependencies, helped us to detect potential problem areas, such as critical dependencies that are not enforceable and for which there were no clear assurances of commitment. The Task Leader's need for unambiguous requirements is an example of such an inconsistency. This seemed to fulfill, at least partially, requirement R1.

The AD model captured all the information, work, and resource flows through resource and task dependencies. This allowed us to identify inconsistencies between what some agents needed and the support that they were actually getting. The problem of the Process Analyst's need for development data from maintainers is an example of this. We also found that the soft-goal dependency in particular was useful in highlighting areas in which the environment was immature. The unambiguous requirements dependency exemplifies this situation. The various types of dependencies in the AD model therefore fulfilled requirement R2.

The actor decomposition extension to the AD model makes a clear distinction between various organizational entities by defining and differentiating roles, positions and agents as different specializations of actors (requirement R3). This allowed us to extract different information from different versions of the AD model, using different specializations of the actors. For example, we found that the model remained fairly stable from project to project when nodes represented positions (as in Figure 3). However, when we used the role specialization, significant differences appeared between projects. For example, roles of managers often varied significantly, depending on their technical background. This served to show that process participants found the freedom to tailor their work to the situation, while the official organizational structure could remain stable. Roles also provide a way to create explicit links between the organizational model and any process model composed of consistently defined activities (requirement R4).

Many interactions with various members of the maintenance organization were necessary in order to clarify inconsistencies and insure completeness. The AD model played an important role in this communication, because it facilitated the exchange and comparison of perceptions about the organizational structure. It served as a good communication tool (requirement R5).

4.2. Issues

Despite the numerous advantages of the AD model mentioned in the previous section, some problems have been identified and should be the subject of further research.

Classification of dependencies

Once a dependency has been identified, it is not always straightforward to classify it according to the defined taxonomy (requirement R2). One example is the difficulty in distinguishing between a task dependency and a goal dependency. A task may be partially defined (e.g., through standards) but some significant degree of freedom can exist for the dependee whose understanding of the task objectives may or may not be complete. It is for this reason that we have included no task dependencies in our AD model (see Figure 3). Also, the borderline between soft-goals and (hard-)goals is not always clear. When is a goal sufficiently defined to be classified as a (hard-)goal? More precise guidelines are needed in order to classify dependencies in an appropriate fashion.

Another inadequacy of the classification scheme is in the case of information dependencies. As defined, information dependencies are one type of resource dependency. However, a need for information is different in nature from a need for time, money, or personnel resources. From a data analysis point of view, information dependencies are described by different attributes than those that would be used to describe other resource dependencies. For this reason, any kind of information flow analysis necessitates the treatment of information as a separate type of dependency.

Criticality of dependencies

No precise and unambiguous definition exists to classify a dependency as critical, committed or open, which impedes fulfillment of requirement R1. Because of this, most of the dependencies in our context appeared critical since they were certainly important from the dependee's perspective. It was, from a practical perspective, difficult to determine if they were really indispensable.

Another difficulty with identifying committed and open dependencies is that practitioners often do not mention them in interviews and they are usually not included explicitly in process documents. We have found that direct observation is the only effective way to capture such secondary dependencies. This is time- and effort-consuming. Furthermore, when modeling at the level of detail of our model, it is sufficient to include only the primary dependencies, which are usually critical .

Interactions between dependencies

The notions of enforcement, assurance and insurance are extremely useful but they are difficult to represent explicitly in the AD model representation (requirement R5). These notions need to be captured explicitly by the organizational model. In the next section, we suggest a way to do this by treating these three mechanisms as interactions between dependencies.

5. Suggestions

Based on this case study and our evaluation of the AD model, we provide some suggestions which may be useful to those wishing to extend the AD concepts, and to those who are engaged in organization modeling.

5.1. An Entity-Relationship Model

We believe two of the most important problems that arose in our work with Actor-Dependency models have a common solution. The first issue is the need to clearly define the information that needs to be collected, particularly in a quantitative analysis effort. The second issue is that of separating organizational from process concerns since they require different types of analyses and solutions. The Entity-Relationship Model, shown in Figure 4 and discussed in the next two sections, addresses both of these issues.

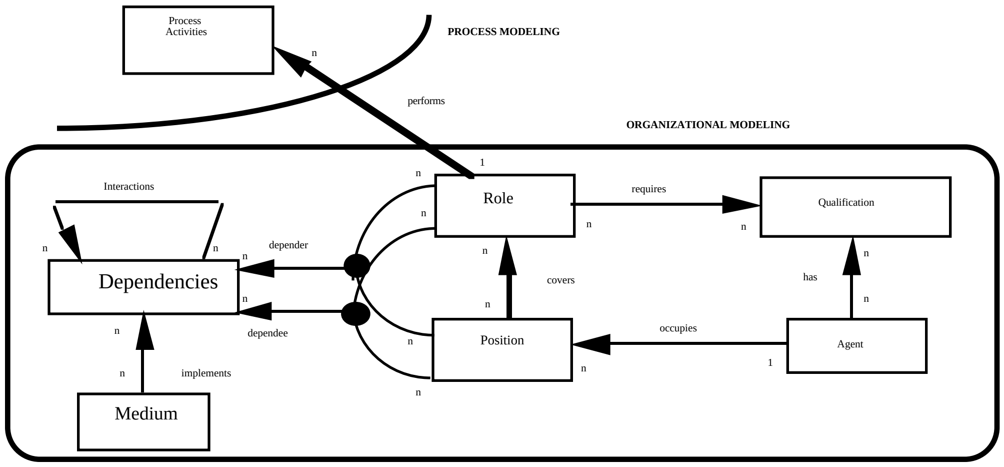

Figure 4: Modified ER model for AD graphs

5.1.1. ER Model

Defining precisely the entities and attributes of interest is not only necessary for data analysis, but also helps clarify the modeling approach itself. One entity that we have added in Figure 4 is the Qualification entity. An agent "has" one or more qualifications, e.g., maintaining ground satellite software systems. Moreover, based on experience, it may be determined that some role "requires" specific qualifications, e.g., experience with Ada. Comparison of the required qualifications and the actual organizational set-up appears useful for identifying high-risk organizational patterns.

We have retained the agent/role/position decomposition of an actor defined by Yu et al, which we found very useful. The ER model also shows "depender" and "dependee" as ternary relationships. This reflects the fact that a depender or dependee of a dependency can be either a role or a position. A role may be functionally dependent on another role in order to perform a given process activity. Positions are usually interdependent because of the need for authorization or authority. However, we believe that dependencies are not inherent to agents themselves, at least not in our context.

We have also added a new entity, Medium, which is the communication medium used to implement a particular dependency (especially information dependencies). This entity is used in some types of quantitative analysis, which is described in a later section. Finally, dependencies are related to each other and this is captured through the interaction relationship, also described in a later section. However, this ER model requires further definition (e.g., attributes should be specified), validation, and refinement.

5.1.2. Linking an organization model with a process model

The ER model also makes explicit the relationship, and the separation, between process and organization. Analysis of an organization is aided by the isolation of organizational issues (e.g., information flow, distribution of work) from purely process concerns (e.g., task scheduling, concurrency). This is especially true when dealing with quantitative data analysis. Process entities and organization entities are described by different quantitative attributes. Separation of these attributes clarifies the analysis. Although organization and process raise separate issues, their effects are related. Understanding the relationship between organization and process is crucial to making improvements to either aspect of the environment (requirement R4). For example, the "performs" relationship can link a role to a set of activities, which may be seen as lower-level roles. The entity Process Activity is itself related to other entities in the process model not specified here.

5.2. Dependency Interactions

Interactions between dependencies need to be modeled. There are several different types of these interactions which may be seen as relationships from a source to a target dependence:

1) Being committed to the source dependency makes the commitment to the target dependency more difficult. This represents a *negative assurance*.
2) The source dependency is an additional motivation to the target dependency. This represents a *positive assurance.*

3) The source dependency's failure can provoke the failure of the target dependency. One dependency's depender is the other dependency's dependee and vice-versa. This represents a dependency *enforcement*.
4) Failure of one dependency is mitigated by the other dependency. Both dependencies have the same depender but different dependees. This is a dependency *insurance*.

If a depender can count on many dependees to deliver a dependum, we can say that the dependency is *insured*. In this case, different dependees can be committed to the same dependum. This can be graphically represented in a AD graph in a fashion similar to OR branches in AND-OR trees. For example, Figure 5 shows a case where a Task Leader (depender) can count on a Maintainer or a Tester (two different dependees) for delivering the "Test Plan & Results" (a particular dependum).

We can also provide a representation for expressing assurance interactions between dependencies, shown in Figure 6. All nodes in the diagram are dependencies, and the arrows between them represent either negative or positive assurances.[1] In Figure 6, all the soft-goals contribute positively (are positive assurances) to the goal "High-quality release", but all but one contribute negatively to "Release on time". All of these dependencies have to be previously defined in the AD model.

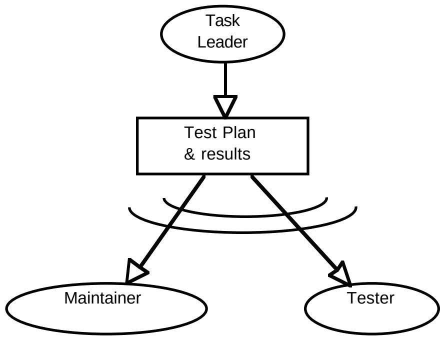

Figure 5: Insurance Representation.

Our suggestion for representing dependency enforcements is a variation of the above. A dependency which enforces another dependency can be seen as one which completely assures it. So our representation uses the same arrows between dependencies shown in Figure 6, with the infinity symbol ("∞") in place of the plus ("+") or minus ("-").

[1] Readers familiar with the work of Yu et al will find this notation similar to their Issue Argumentation model, which we did not make use of in our work. However, our notation which we present here has different semantics than the IA model, and the two should not be confused.

5.3. Use of quantitative data

The use of quantitative data is critical to the useful analysis of development processes and organizations. Without quantitative information, the analysis results are not sufficient to effectively compare alternatives and to make decisions. Qualitative analysis, while important for intuitive understanding and insight, must be taken further to provide a basis for action. For example, [Perry et. al., 1994] have recently attempted to characterize and quantify the workload of software developers across software development process activities.

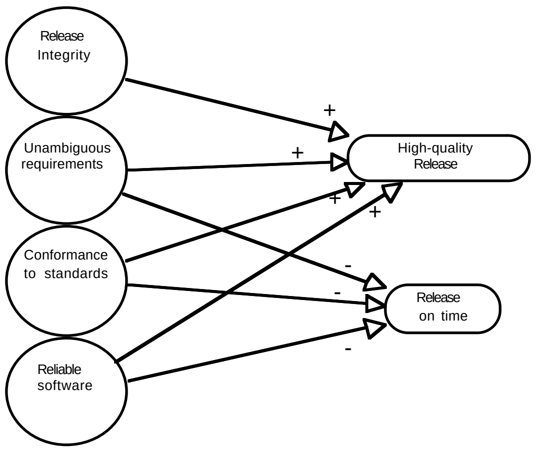

Figure 6: Representing Assurances

In fact, the AD model is particularly well suited to incorporating data, although there is not an explicit facility for this in the modeling methodology. One way to perform such analysis is to associate attributes with the various AD entities (positions, roles, dependencies, etc.). The attributes could be used to hold the quantitative information. Then analysis tools can be used to analyze the AD graph, by making calculations, based on the data, according to the structure represented in the graph.

One type of quantitative analysis, which has already been alluded to, is information flow analysis. Information dependencies (one type of resource dependency) can have attached to them attributes such as frequency and amount of information. Each information dependency is also related to the different communication media (the entity Medium in Figure 4) that it uses to pass information, e.g. phone, email, formal and informal documents, formal and informal meetings. The many-to-many relationships between dependencies and their media also have attributes (e.g., effort). Such attributes are captured by defining metrics and collecting the appropriate data. An example of such an attribute is the computation, for each information dependency, of the product of the dependency frequency, the amount of information, and the effort associated with the medium related to the dependency. This product gives a quantitative assessment of the effort expended to satisfy the information dependency. Summing these values for each pair of actors in the AD graph shows how much effort the pair expends in passing information to each other. This information can be used to support such management decisions as how to fill different positions, how to locate these people, and what communication media to make available. Without quantitative analysis, these decisions are subject to guesswork, trial and error, and the personal expertise of the manager. For more on metrics for organizational information flow, see [Seaman, 1994].

There are several possible applications of quantitative analysis in relation to the actor/position/role decomposition. For example, during the course of our study, we noticed that many differences between projects were reflected in variations in the breakdown of positions into roles. In other words, the people filling the same positions in different projects divided their effort differently among their various roles. These variations were usually symptomatic of differences in management strategy and leadership style. Data needs to be collected to capture the important variations in effort breakdown across organizations and projects. This data must then be attached to entities in the AD model so that it can be used to analyze variations in job structure. For example, suppose that we wanted to find out which projects require a manager with technical expertise. If we have quantitative data available on the effort breakdown of the different managers, then we can easily see which managers spend a high proportion of their time on technical activities. This information can be used in choosing people to fill different management positions. Variations in effort breakdown can also be represented in an AD graph by varying the thickness of the lines which join a position with its various roles, as shown in Figure 7.

Effort breakdown is only one example of the many possibilities for analysis of the role/position/agent structure of actors. Qualification analysis, which would involve the Qualifications entity in Figure 4, is another example. Understanding the sharing of tasks and responsibilities is another area in which quantitative analysis could be useful. All of these involve the evaluation of quantitative attributes attached to roles, positions, agents, and the links (occupies, contains, performs) between them.

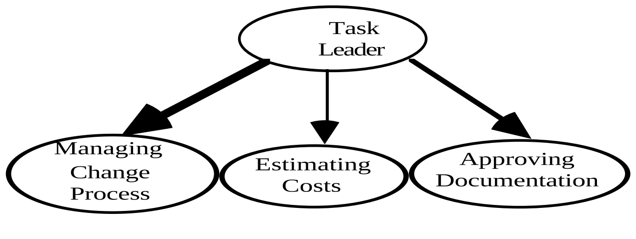

Figure 7. Representing effort breakdown per role

5.4. Acquisition process

Any modeling effort requires that a great deal of information be collected from the environment being modeled. Building an AD model requires collecting information about all the people in the environment, the details of their jobs and assignments, whom they depend on to complete their tasks and reach their goals, etc. Our experience has shown that it is useful to follow a defined process for gathering this information, which we will call an *acquisition process*. The acquisition process which we followed, with modifications motivated by our experience, is briefly presented in this section. The steps are as follows:

Step 1: First, we determined the official, (usually) hierarchical structure of the organization. Normally this information can be found in official organization charts. This gives us the set of positions and the basic reporting hierarchy.

Step 2: We determine the roles covered by the positions by interviewing the people in each position, and then, to check for consistency, their supervisors and subordinates. Process descriptions, if available, often contain some of this information. However, when using process descriptions, the modeler must check carefully for process conformance.

Step 3: In this step, we focus on the goal, resource, and task dependencies that exist along the vertical links in the reporting hierarchy. To do this, we interview members of different departments or teams, as well as the supervisors of those teams. Also, direct observation of supervisors, called "shadowing", can be useful in determining exactly what is requested of, and provided by supervisors for their subordinates.

Step 4: Here we focus on resource (usually informational) and goal dependencies between members of the same team. Direct observation (through shadowing or observation of meetings) is also useful here. Interviews and process documents can also be used to identify dependencies.

Step 5: Finally, we determine the informational and goal dependencies between different teams. These are often harder to identify, as they are not always explicit. Direct observation is especially important here, as often actors do not recognize their own subtle dependencies on other teams. It is also very important in this step to carefully check for enforcement, assurance, and insurance mechanisms, since dependers and dependees work in different parts of the management hierarchy.

6. Conclusions

This paper presents the experience of using the Actor-Dependency modeling approach to model and analyze a large scale maintenance organization. The AD model was found to be very useful at capturing the important properties of the organizational context of the maintenance process, and aided in the understanding of the flaws found in this process. There were, however, some inadequacies of the approach, which we have addressed through a set of proposed suggestions. However, those must be seen as research directions and need to be further investigated.

One major potential extension of the approach is to use quantitative data and analysis methods, within the framework of an AD model. Qualitative methods are not sufficient to differentiate organizations and especially variations across projects. Measurement is therefore necessary for studying organizations.

The AD model also needs automated support for real-scale organizations. This is required to allow the user to analyze a real-time organization and define complementary views of the studied organizations, at different levels of refinement, at different levels of completeness. Automated support is especially crucial for the use of quantitative analysis.We need also to better define the relationship between the organization and the development process. Separating organizational concerns from process concerns, but considering them in conjunction with each other, is a crucial element in the comprehensive study of development environments (see [Seaman, 1994]). Finally, collecting information about an organization for building an accurate AD model is a complex task. Therefore, based on experience, we need to define an optimal data acquisition process that can be tailored to various maintenance environments.

Acknowledgements

We want to thank the referees and Khaled El Eman for their suggestions that helped improve both the content and the form of this paper. We would also like to thank the people at NASA and CSC who participated in this work.

References

Basili, V.R. (1989). "Software Development: A Paradigm for the Future". In *Proceedings, COMPSAC '89*, Orlando, FL, September.

Benus, B. (1994). "Detailed Design Document; Organisation Model Tool". Technical Report KADS-II/M6/UvA/057/1.0, University of Amsterdam, May.

Briand, L. C.; Basili, V. R.; Kim, Y.M.; Squier, D. R. (1994). "A Change Analysis Process to Characterize Software Maintenance Projects". In *Proc. of the IEEE Int'l Conf. on Software Maintenance*. Victoria, Canada, September.

Curtis, B.; Kellner, M.I.; Over, J. (1992). "Process Modeling". *Communications of the ACM*, 35(9):75-90, September.

Finkelstein, A.; Kramer, J.; Nuseibeh, B., eds.(1994). *Software Process Modeling and Technology*. Research Studies Press (distributed by Wiley & Sons).

Lanphar, R. (1990). "Quantitative Process Management in Software Engineering, a Reconciliation Between Process and Product Views". *Journal of Systems and Software*, 12:243-248.

Lott, C.M.; Rombach, H.D. (1993). "Measurement-based Guidance of Software Projects Using Explicit Project Plans". *Information and Software Technology*, 35(6/7):407-19, June/July.

Melo, W. and Belkhatir, N. (1994). "Collaborating Software Engineering Processes in Tempo". *Journal of the Brazilian Computer Society*, 1(1):24-35.

Perry, D.; Staudenmayer, N.; Votta, L. (1994), "People, Organizations, and Process Improvement". *IEEE Software*, 11(4):36-45.

Rein, G.L. (1992), "Organization Design Viewed as a Group Process Using Coordination Technology". MCC Technical Report No. CT-039-92, February.

Seaman, C.B. (1994), "Using the OPT Improvement Approach in the SQL/DS Development Environment". In *Proceedings of CASCON'94*, (CD-ROM version), Toronto, Canada, October.

Yu, E.; Mylopoulos, J. (1993). "An Actor Dependency Model of Organizational Work - with Application to Business Process Reengineering". In *Proc. Conference on Organizational Computing Systems (COOCS 93)*, Milpitas, CA, November.

Yu, E. S.; Mylopoulos, J. (1994). "Understanding 'Why' in Software Process Modeling, Analysis, and Design". In *Proc. of the 16th IEEE Int'l Conf. on Software Engineering*. Sorrento, Italy. pp. 159-168.

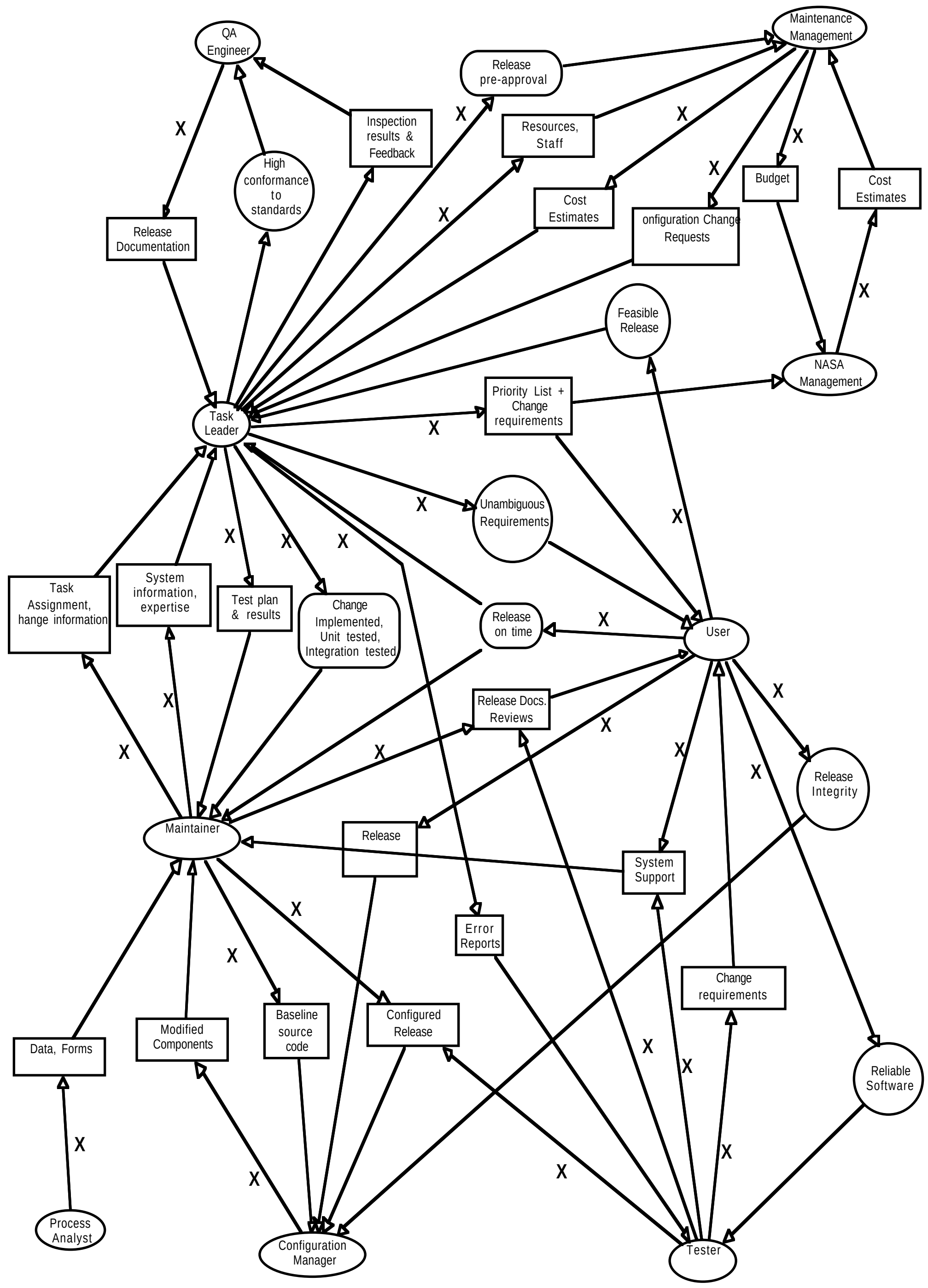

Figure 3: AD Model of a Maintenance Organization.