Qualitative Analysis for Maintenance Process Assessment

Lionel Briand
CRIM
Montréal, Québec, Canada

Yong-Mi Kim
Walcélio Melo
Carolyn Seaman
Victor Basili
Institute for Advanced Computer Studies
University of Maryland
College Park, MD, USA

Abstract

In order to improve software maintenance processes, we first need to be able to characterize and assess them. These tasks must be performed in depth and with objectivity since the problems are complex. One approach is to set up a measurement-based software process improvement program specifically aimed at maintenance. However, establishing a measurement program requires that one understands the problems to be addressed by the measurement program and is able to characterize the maintenance environment and processes in order to collect suitable and cost-effective data. Also, enacting such a program and getting usable data sets takes time. A short term substitute is therefore needed.

We propose in this paper a characterization process aimed specifically at maintenance and based on a general qualitative analysis methodology. This process is rigorously defined in order to be repeatable and usable by people who are not acquainted with such analysis procedures. A basic feature of our approach is that actual implemented software changes are analyzed in order to understand the flaws in the maintenance process. Guidelines are provided and a case study is shown that demonstrates the usefulness of the approach.

This work was supported by NASA grant NSG-5123, NSF grant 01-5-24845, and by NSERC, Canada.

E-mails: {basili | kimy | melo | cseaman }@cs.umd.edu and lbriand@crim.ca

1. Introduction

During the past few years the definition and improvement of software processes has been playing an increasingly prominent part in software development and maintenance. The improvement of software maintenance processes is of particular interest because of the length of time spent in maintenance during the software life cycle, and the ensuing lifetime costs, as well as the large number of legacy systems still being maintained. Improvement requires building an understanding of what is actually happening in a project (the term "project" here refers to the continuous maintenance of a given system), in conjunction with building a measurement program.

Establishing a measurement program integrated into the maintenance process is likely to help any organization achieve an in-depth understanding of its specific maintenance issues and thereby lay a solid foundation for maintenance process improvement [RUV92]. However, defining and enacting a measurement program takes time. A short term, quickly operational substitute is needed in order to obtain a first quick insight, at low cost, into the issues to be addressed. Furthermore, defining efficient and useful measurement procedures first requires a characterization of the maintenance environment in which measurement takes place, such as organization structures, processes, issues, and risks [BR88].

Part of this characterization is the identification and assessment of issues that must be addressed in order to improve the quality and productivity of maintenance projects. Because of the complexity of the phenomena studied, this is a difficult task for the maintenance organization. Each project may encounter specific difficulties and situations that are not necessarily alike across all the organization's maintenance projects. This may be due in part to variations in application domain, size, change frequency, and/or schedule and budget constraints. As a consequence, each project must first be analyzed as a separate entity even if, later on, commonalities across projects may require similar solutions for improvement. Informally interviewing the people involved in the maintenance process would be unlikely to accurately determine the real issues. Maintainers, users and owners would likely each give very different, and often contradictory, insights on the issues due to their biased or incomplete perspectives.

This paper presents a qualitative and inductive analysis methodology for performing objective characterizations and assessments that addresses both the understanding and measurement aspects of improvement It encompasses a set of procedures which aids the determination of causal links between maintenance problems and flaws in the maintenance organization and process. Thus, a set of concrete steps for maintenance quality and productivity improvement can be taken based on a tangible understanding of the relevant maintenance issues. Moreover, this understanding provides a solid basis on which to define relevant software maintenance models and metrics.

Section 2 gives an overview of the basic steps of the proposed assessment methodology, and a discussion of the supporting technologies and capabilities it requires. Section 3 presents the supporting technologies we actually used to execute the assessment method. Section 4 presents each step of the assessment process in detail by going through a case study. The experience we gained conducting this case study is the source for much of the guidance we offer in this paper, as well as the basis for the fine-tuning of the assessment methodology itself. Section 5 describes the next logical step after a qualitative assessment, the design of a measurement program aimed at quantitatively monitoring and improving software maintenance. Section 6 outlines the main conclusions of this experience and the future research directions.

2. Overview of the Maintenance Assessment Method

We present below a general description of the maintenance assessment method and the capabilities it requires. Maintenance is defined here as any kind of enhancement, adaptation or correction performed on an operational software system. At the highest level of abstraction, parts of the assessment process are not specific to maintenance and could be used for development. However, the taxonomies and guidelines developed to support this process and presented in Section 3 are specifically aimed at maintenance.

2.1 The steps of the method

We propose a qualitative and inductive methodology in order to characterize and assess software maintenance processes and organizations and thereby identify their specific problems and needs. This methodology encompasses a set of procedures which attempt 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 are 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, working groups, and their roles in the change process are identified. Information flows between actors are also determined.

Step 2: Identify the phases involved in the creation of a new system release. As opposed to the notion of activity, defined below, phases produce one or several intermediate or final release products which are reviewed according to quality assurance procedures, when they exist, and are officially approved. In addition, the phases of a release are ordered in time, although they may be somewhat overlapping, and are clearly separated by milestones. 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 representative past releases for analysis in order to better understand process and organization flaws.

Step 5: Based in part on release documents and error report forms, analyze the problems that occurred while performing the software changes in the selected releases in order to produce 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.

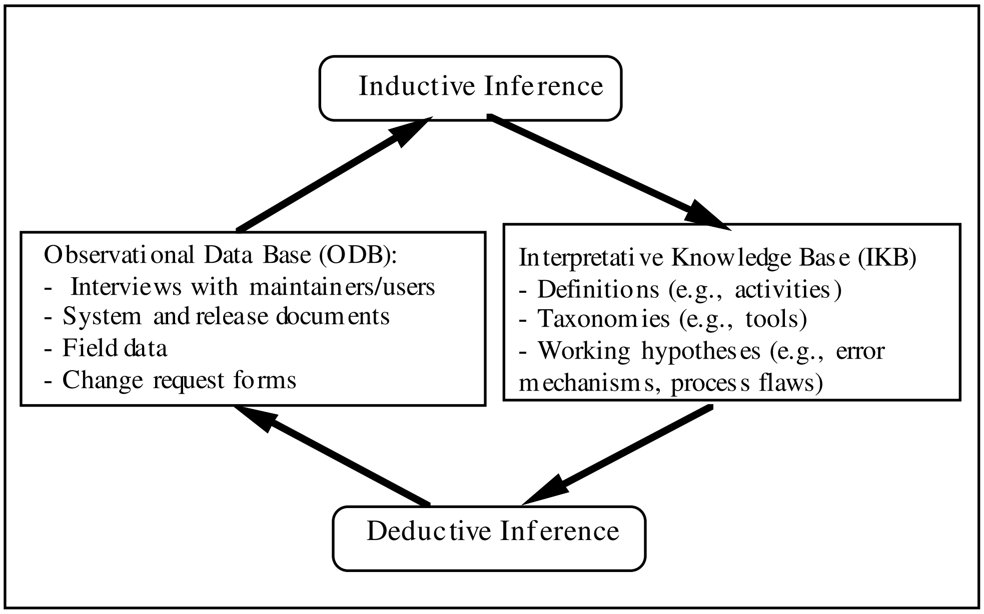

Figure 1: Qualitative Analysis Process for Software Maintenance

This process is essentially an instantiation of the generic qualitative analysis process defined in [SS92]. Figure 1 illustrates at a high level our maintenance-specific qualitative analysis process. It is a combination of both inductive and deductive inferences. The collected information, such as field notes or interviews, comprise the *Observational Database* (ODB), while the models built using the information from the ODB goes into the *Interpretative Knowledge Base* (IKB). Inductive inferences are made from the collected information. Deductive inferences occur when, based on our IKB, we derive expectations about the real world. An example of such an expectation might be that all errors can be exhaustively classified according to defined taxonomies. When comparing these expectations with new field information feeding the ODB, we can experimentally validate and refine our taxonomies, process models, organizational models and working hypotheses (all in the IKB). Then the data collection process is refined in order to solve ambiguities and answer new questions, which leads to refined and revised inductive inferences. The process continues in an iterative fashion. This iterative pattern not only applies to the overall assessment process, but also to several of the individual steps. For example, in our case study, performing Step 1 revealed additional issues to be addressed in building an organizational model, and led to the selection and use of a more sophisticated modeling approach. The iterative nature of these steps is described in more detail in Section 4.

2.2 Capabilities required

During the case study, we were faced with several tasks which required supporting pieces of technology. In this section, we describe the requirements of these technologies, which could be satisfied in a number of different ways. In section 3, we describe in detail the representations and taxonomies we chose to satisfy these requirements during our case study.

2.2.1 Step 1: Organizational Modeling

The first technology we needed was a representation in which to build the organization model (the first step of the assessment method). We identified the following requirements for an optimal organizational modeling approach:

Req1: 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.

Req2: 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 and we believe it is likely to be relevant in other environments as well.

Req3: 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. It 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.

Req4: Links between the organization and the maintenance process model had to be represented explicitly.

Req5: The notation had to aid in communication through intuitive concepts and graphical representation.

Req6: We had to be able to flexibly capture information about the maintenance working environment, e.g., available tools and methods.

The process modeling literature provides many examples of techniques for representing various aspects of process. Process modeling is performed for a number of different purposes, including process analysis and process improvement. The "process" under study here includes all the activities involved in the development (and sometimes maintenance) of software. The representations we considered, and the one we finally chose, are described in sections 3.1 and 3.2.

2.2.2 Steps 2 and 3: Process Modeling

Another technology required is a way to model relevant aspects of the maintenance process. However, we found that once the above requirements have been satisfied by the organizational model, the process "model" does not need to be sophisticated. All that is required is a breakdown of the process into its constituent parts, and identification of the inputs and outputs (artifacts) of each part. The classification schemes we used are described in section 3.3.

2.2.3 Step 5: Causal Analysis

The other technologies required to implement the assessment method take the form of descriptions of different entities that are specific to the environment being studied. For example, the causal analysis step requires a list, or taxonomy, of types of maintenance flaws that take place in the environment. Also required by the causal analysis step is a data collection guide which describes, in terms tailored to the studied environment, the information that needs to be collected in this step. These taxonomies and guides are described in section 3.4.

3. Technologies Used

This section describes the technologies used by the authors to satisfy the requirements listed in the last section, to conduct the case study described in section 4. The pieces of technology described here are not the only possible choices, but they represented the best options for our circumstances and environment.

3.1 An organizational modeling representation

During the case study, it was clear that the organizational model built in step 1 would be central to the assessment. Thus, we were faced with the crucial task of finding an appropriate organizational modeling approach. We first looked in the process literature for such a technology. Representation of organizational structure in most process work is limited to the representation of roles [BK94, LR93], with a few notable exceptions. One is the approach presented in [K91], which uses Statemate. Statemate models include three perspectives, one of which is the organizational perspective, which identifies the people who participate in the process and the information channels between them. Role Interaction Networks (RINs) [R92] (implemented in the modeling tool Deva [MCC92]) also describe process participants and the information that passes between them. The use of CRC cards in the Pasteur tool [CC93] completely describes a process in terms of the process participants and their work dependencies. None of these approaches, however, provides the ability to represent the richness of types of persons, groups, relationships, and structures that we require, e.g., conflicting objectives, synergy between objectives, risk management mechanisms. Furthermore, these approaches do not provide straightforward links between quantitative data and organizational models. This issue is important, as we shall see in Section 5, because the organizational model will be an important tool in the eventual quantitative analysis of the maintenance process.

Yu's Actor-Dependency (A-D) model [YM94] is another modeling notation that shares some of the characteristics of the three described above. In particular, A-D models are based on process participants and the relationships between them, including information flow relationships. These relationships are not limited, however, to information flows. In addition, A-D models provide a variety of ways to represent members of the organization that we believe to be based on a clear and convenient paradigm (see section 3.1.2). Like the above approaches, A-D models were not designed originally to facilitate quantitative data collection and analysis. However, in practice, we observed that the A-D model was easily modified for this purpose. In Section 5, we describe some of these modifications and in [B+95] we provided a more detailed list of enhancements we proposed to the A-D model. Because of the richness of its underlying paradigm, this modeling approach has been chosen as the representation for our organization model. A-D models are described in more detail in the sections below.

This modeling 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 [YM93].

3.1.1. 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, documents, 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 2 shows a simple example of an AD model. A Manager oversees a Tester and a Developer. The Manager depends on the Tester to efficiently and effectively test the product. 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.

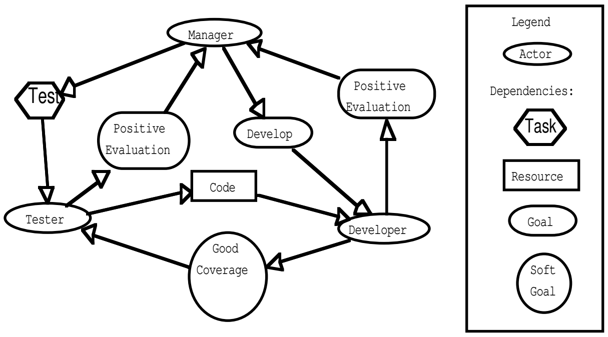

Figure 2: A simple example of an AD model

3.1.2. 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 3 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. Positions provide a high-level view of the organization whereas roles provide mores details. The use of agents allows the modeler to go even further and specify specific individuals. In addition, the ARP decomposition could be especially useful when extending the use of AD models to quantitative analysis, as explained in Section 5.

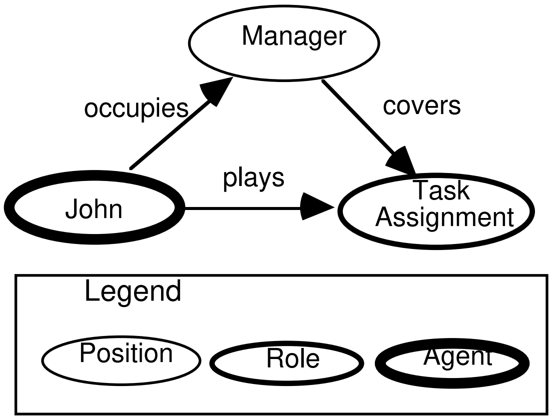

Figure 3. Associated Agent, Position, and Role

3.1.3. Limitations of the organizational model

The AD modeling method satisfied, at least partially, all of the modeling requirements presented in section 2.2, except requirement Req6. However, there were some difficulties. Fulfillment of Req1 and Req2 was impeded by the difficulty of distinguishing between task, goal, resource, and soft-goal dependencies, and between critical, committed, and open dependencies. These categories did not always adequately describe the dependencies that arose in our model. In addition, although the notions of enforcement, assurance and insurance helped to satisfy requirement Req5, they are difficult to represent explicitly in the AD model representation. For more details on that subject, see [B+95].

3.1.4. Value of the organizational model

Modeling the organizational context of the maintenance process was a very important step in the maintenance 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 roles in the process model were assigned to actors in the organization.

In addition, we found that several actor decomposition patterns that were of particular interest in identifying potential organizational problems:

- unassigned roles: nobody has official responsibility for a given role and its associated activities.
- numerous roles associated with a position: the position may be overloaded and/or incompatible roles may be played by one position.
- shared roles across positions: can the responsibility be shared or is this an opportunity for confusion?
- variations of role-position associations across maintenance projects: is this variation due to a lack of definition or to necessary adjustments from project to project?

3.2 Taxonomy of maintenance methods and tools

It was not possible to capture attributes of the maintenance working environment (i.e. tools and methods) within an AD model (requirement Req6). For our case study, we collected this information during step 1 but kept it separate from the organizational model. To organize this information, we found it useful to develop a taxonomy of tools and methods which are relevant in the modeled environment. Figure 4 shows a taxonomy that we think is a good characterization of the information to be gathered about the maintenance environment under study. The taxonomy shows only the first level of abstraction, so that it can be specialized for a particular maintenance environment.

Maintenance tools:
• impact analysis and planning tool • tools for automated extraction and representation of control and data flows • debugger • generator of cross-references • regression testing environment (data generation, execution, and analysis of results) • information system linking documentation and code.
Maintenance methods:
• rigorous impact analysis, planning, and scheduling procedures • systematic and disciplined update procedures for user and system documentation • user communication channels and procedures

Figure 4. The first level of abstraction of taxonomies of relevant maintenance methods and tools (see [BC91])

3.3 Process taxonomies

Our experience has shown that most of the information needed to carry out our assessment process is contained in the organizational model (built in step 1). The process model built in steps 2 and 3 is also necessary, but it can be fairly simple and straightforward. The process model we built for our case study (described in section 4.2) is simply a breakdown of the maintenance process into phases, with activities and relevant documents identified for each phase. Thus, the supporting technologies needed for the process modeling step are simply a taxonomy of maintenance documents and a taxonomy of generic activities. These taxonomies are shown in Figures 5 and 6, respectively.

Product-related:

- software requirements specifications
- software design specifications
- software product specifications

Process-related:

- test plans
- configuration management plan
- quality assurance plan
- software development plan

Support-related:

- software user's manual
- computer systems operator's manual
- software maintenance manual
- firmware support manual

Figure 5. A generic taxonomy of maintenance documentation (see [BC91]).

The taxonomy of generic maintenance activities is shown in Figure 6. All these activities usually contain an overhead of communication (meeting and release document writing) with owners, users, management hierarchy and other maintainers, which should be estimated. This is possible through data collection or by interviewing maintainers.

Acronym	Activity
DET	Determination of the need for a change
SUB	Submission of change request
UND	Understanding requirements of changes: localization, change design prototype
IA	Impact analysis
CBA	Cost/benefit analysis
AR	Approval/rejection/priority, assignment of change request
SC	Scheduling/planning of task
CD	Change design
CC	Code changes
UT	Unit testing of modified parts, i.e., has the change been implemented?
IC	Unit Inspection, Certification, i.e., has the change been implemented properly and according to standards?
IT	Integration testing, i.e., does the changed part interface correctly with the reconfigured system?
RT	Regression testing, i.e., does the change have any unwanted side effects?
AT	Acceptance testing, i.e., does the new release fulfill the system requirements?
USD	Update system and user documentation
SA	Checking conformance to standards; quality assurance procedures
IS	Installation
PIR	Post-installation review of changes
EDU	Education/training regarding the application domain/system

Figure 6. Taxonomy of generic maintenance activities (see [BC91])

Error origin: when did the misunderstanding occur?

- Change requirements analysis
- Change localization analysis
- Change design analysis
- Coding

Error domain: what caused it?

- Lack of application domain knowledge: *operational constraints (user interface, performance), mathematical model*
- Lack of system design or implementation knowledge: *data structure or process dependencies, performance or memory constraints, module interface inconsistency*
- Ambiguous or incomplete requirements
- Language misunderstanding <semantic, syntax>
- Schedule pressure
- Existing uncovered fault
- Oversight.

Figure 7. Taxonomy of human errors.

3.4 Causal analysis technologies

The causal analysis part of the assessment method requires several taxonomies which are used to categorize the problems found. The first taxonomy required is one of human errors which lead to maintenance problems. This taxonomy is shown in Figure 7. Another substep in causal analysis is to categorize the findings according to a taxonomy of common maintenance process and organization flaws. The taxonomy we used, another piece of supporting technology, is shown in Figure 8. As a guide for conducting interviews and studying release documents, an outline of the information that should be collected for each software change is provided in Figure 9.

Organizational flaws:

- communication: interface problems, information flow "bottlenecks" in the communication between the maintainers and the
    - users
    - management hierarchy
    - quality assurance (QA) team
    - configuration management team
- roles:
    - prerogatives and responsibilities are not fully defined or explicit
    - incompatible responsibilities, e.g., development and QA
- process conformance: no effective structure for enforcing standards and processes

Maintenance methodological flaws

- Inadequate change selection and priority assignment process
- Inaccurate methodology for planning of effort, schedule, personnel
- Inaccurate methodology for impact analysis
- Incomplete, ambiguous protocols for transfer, preservation and maintenance of system knowledge
- Incomplete, ambiguous definitions of change requirements
- Lack of rigor in configuration (versions, variations) management and control
- Undefined / unclear regression testing success criteria.

Resource shortages

- Lack of financial resources allocated, e.g., necessary for preventive maintenance, unexpected problems unforeseen during impact analysis.
- Lack of tools providing technical support (see previous tool taxonomy)
- Lack of tools providing management support (i.e., impact analysis, planning)

Low quality product(s)

- Loosely defined system requirements
- Poor quality design, code of maintained system
- Poor quality system documentation
- Poor quality user documentation

Personnel-related issues

- Lack of experience and/or training with respect to the application domain
- Lack of experience and/or training with respect to the system requirements (hardware, performance) and design
- Lack of experience and/or training with respect to the users' operational needs and constraints

Figure 8. Taxonomy of maintenance process flaws

1 Description of the change

1.1 Localization

- subsystem(s) affected
- module(s) affected
- inputs/outputs affected

1.2 Size

- LOCs deleted, changed, added
- Modules examined, deleted, changed, added

1.3 Type of change

- Preventive changes: improvement of clarity, maintainability or documentation.
- Enhancement changes: add new functions, optimization of space/time/accuracy
- Adaptive changes: adapt system to change of hardware and/or platform
- Corrective changes: corrections of development errors.

2 Description of the change process

2.1 effort, elapsed time

2.2 maintainer's expertise and experience

- How long has the person been working on the system?
- How long has the person been working in this application domain?

2.3 Did the change generate a change in any document? Which document(s)?

3 Description of the problem

3.1 Were some errors committed?

- Description of the errors (see taxonomies in Figure 7)
- Perceived cause of the errors: maintenance process flaw(s) (see Figure 8)

3.2 Difficulty

- What made the change difficult?
- What was the most difficult activity associated with the change?

3.3 How much effort was wasted (if any) as a result of maintenance process flaws?

3.4 What could have been done to avoid some of the difficulty or errors (if any)?

Figure 9 Guide to data collection in Step 5.

4. Case Study

In the subsections below, the case study we conducted is described. The individual steps of the maintenance process assessment method, as they were implemented in the case study, are described in detail. For each step, a set of substeps and/or guidelines is presented which facilitates the understanding and implementation of the step. In addition, those steps that are iterative in nature include an explanation of how they fit into the general qualitative analysis process shown in Figure 1.

This case study was performed with the team maintaining GTDS (Goddard Trajectory Determination System), a 26 year old, 250 KLOC, FORTRAN orbit determination system. It is public domain software and, as a consequence, has a very large group of users all over the world. Usually, 1 or 2 releases are produced every year in addition to mission specific versions that do not go into configuration management right away (but are integrated later into a new version by going through the standard release process). Like most maintained software systems, very few of the original developers are still present in the organization and turnover still remains a crucial issue in this environment.

GTDS has been maintained by the Flight Dynamics Division (FDD) of the NASA Goddard Space Flight Center for the last 26 years and is still used daily for most operating satellites. Our case study takes place in the framework of the NASA Software Engineering Laboratory (NASA-SEL), an organization aimed at improving FDD software development processes based on measurement and empirical analysis. Recently, responding to the growing cost of software maintenance, the NASA-SEL has initiated a program aimed at characterizing, evaluating and improving its maintenance processes. The maintenance process assessment methodology presented in this paper was created as part of that effort.

4.1 Modeling the Organization

Step 1 Identify the organizational entities with which the maintenance team interacts and the organizational structure in which maintainers operate.

The output of this step is a model which represents the organizational context of the maintenance process. Building this model is a very important step in the analysis process. Gathering organizational information and constructing the model is critical to understanding the work environment. This understanding makes it possible to accurately analyze flaws in the environment later in the analysis process. This model is also useful in checking the consistency and completeness of the maintenance process model constructed in Steps 2 and 3. For example, the organizational model allows us to determine whether or not all roles implied by the process (based on its constituent activities) are officially assigned to organizational actors.

Like many steps in the assessment process, this first step is iterative. In order to illustrate this, we map this step back into the qualitative analysis process shown in Figure 1. Executing this step usually corresponds to a set of iterations of the qualitative analysis process. The input into the process consists of (structured) interviews, organization charts, maintenance standards definition documents, and samples of release documents. These elements comprise the *Observational Database* (ODB). The organization model, which includes roles, agents, teams, information flow, etc., is the resulting characterization model that goes into the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify the correctness of the organization model. Questions asked during validation include the following:

- Are all the standard documents and artifacts included in the modeled information flow?
- Do we know who produces, validates, and certifies the standard documents and artifacts?
- Are all the people referenced in the release documents a part of the organization model?

The answers to these questions motivate the collection of more material for the ODB. The process iterates with updates and modifications to the organization model (IKB).

We have defined three major subtasks which comprise this first step. The focus of each step is a particular type of information that should be included in the resulting organization model.

1.1 Identify distinct organizational entities, i.e., what are the distinct roles, teams, and working groups involved in the maintenance project?

1.2 Characterize various types of dependencies between entities, e.g., information flows: the types and amounts of information, particularly documents, flowing between organizational entities.

1.3 Characterize the working environment of each entity. In particular, knowledge of the tools and methods (or lack thereof) available to maintainers is useful in identifying and understanding potential sources of problems.

For the case study, we built a model of the entire NASA-FDD maintenance organization. A simplified version of this model is shown as an A-D model (see section 3.1) in Figure 10. In the sections below, we present the experience gained building and using this model. First, we present the procedures we used for gathering the information we needed to begin building the model. Then we present the details of the model itself.

4.1.1. 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 many 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:

A1: First, we determine 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.

A2: 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.

A3: 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.

A4: Next 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.

A5: 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, given that they belong to different teams.

4.1.2. The Organization Model

The organizational model in Figure 10 is very complex despite important simplifications (e.g., agents and roles are not included). This shows how intricate the network of dependencies in a large software maintenance organization can be.

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 A-D 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. Below are listed the positions shown in the figure, and a short explanation of their specific roles:

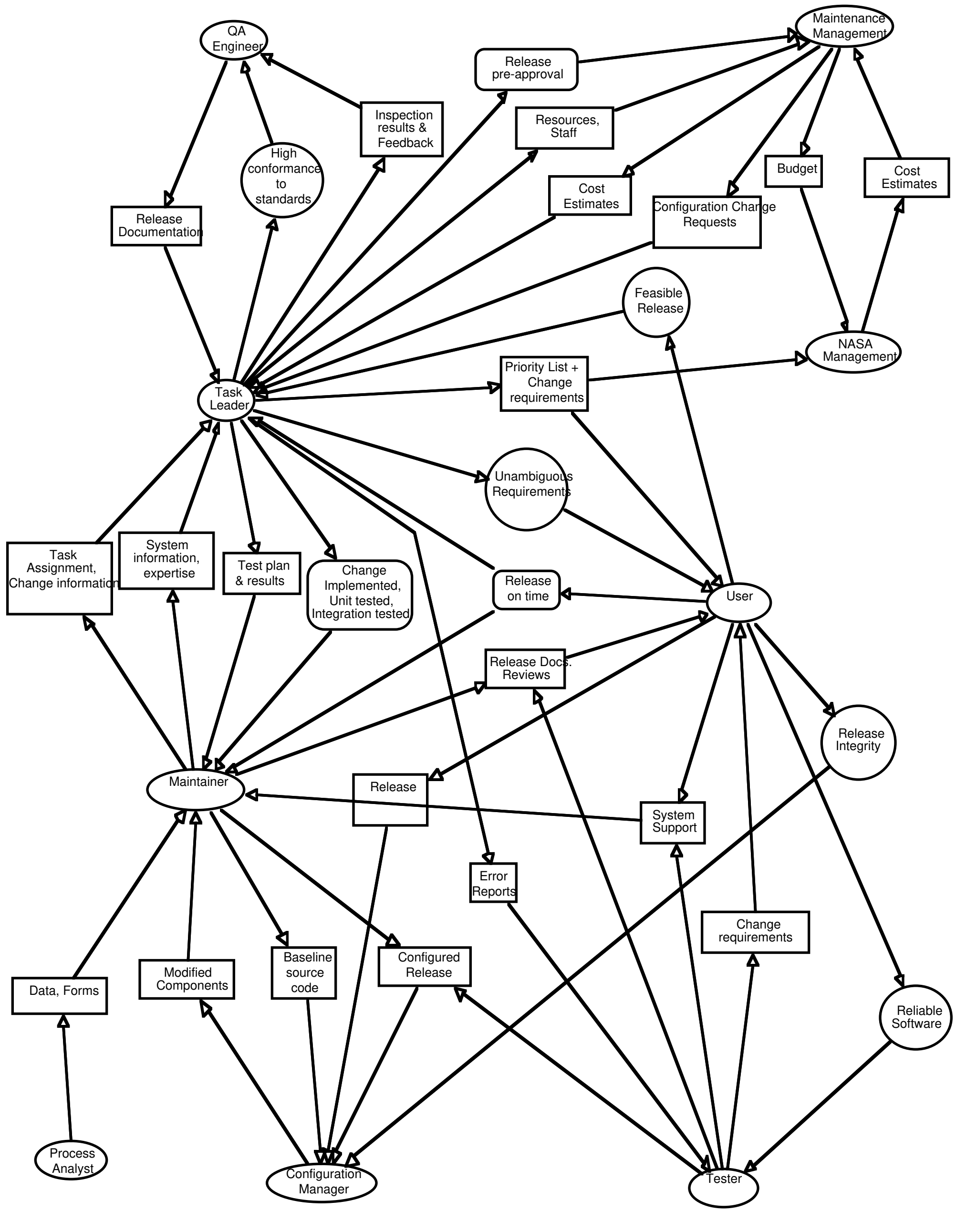

Figure 10 AD Model of a Maintenance Organization.

- *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 maintenance assessment methodology. 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.

We also collected data on the maintenance tools and methods which were available and in use. This data collection effort, along with input from the literature, contributed to the creation of the taxonomy of tools and methods in Figure 4. This information turned out not to be relevant in the causal analysis step in this study, so we will not take the space to present it here. However, we believe that information about tools and methods is very important to collect and understand in order to consider all possible sources of maintenance problems.

4.2 Modeling the Process

Step 2 Identify the phases involved in the creation of a new system release.
Step 3 Identify the generic activities involved in each phase.

Phases and activities are defined and differentiated in the following way:

- Phases are ordered tasks with clearly defined milestones and deliverables going through a review and formal approval process.
- Activities are tasks which cannot be a priori ordered within a phase and do not produce deliverables going through a formal approval process although they can be reviewed (e.g., peer reviews).

- Phases contain activities but activities may belong to several phases, e.g., coding may take place during requirement analysis (e.g., prototyping) and, of course, during implementation.

Steps 2 and 3, together, result in the construction of a process model for a maintenance environment. Both project phases and activities may be defined at several levels of decomposition depending on their complexity. It is important to note that the goal here is to better understand the particular release process of the studied environment and not to enact and/or support such a process. We have separated Step 2 from Step 3 because we have found it useful in practice, based on the differences presented above, to separate the characterization of the process into two levels of abstraction. For example, looking at the distribution of activities across phases is often enlightening. The appropriate granularity of a process model is still, from a general perspective, an open issue but can usually be addressed in practice.

Like Step 1, these two steps together are iterative, and thus we can map them back into the qualitative analysis process shown in Figure 1. The material in the *Observational Database* (ODB), with which we begin the process, consists of (in decreasing order of importance) maintenance standards definition documents, interviews, release documents, and the organization model from Step 1. The resulting output is the process model which becomes part of the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify the correctness of the process model. Validation questions include:

- Are all the people in the process model a part of the organization model?
- Do the documents and artifacts included in the process model match those of the information flow of the organization model?
- Is the mapping between activities and phases complete, i.e., an exhaustive set of activities, a complete mapping?
- Are a priori relevant types of activities (e.g., defined in Figure 6) missing from the process model?

As before, these questions motivate the collection of more data to be collected in the ODB, which in turn modifies the process model and possibly the organization model (IKB), thus continuing the iterative qualitative analysis process. It is also important to continuously verify that the taxonomies of maintenance tools, methods, and activities are adequate, i.e., that the classes are unambiguous, disjoint and exhaustive **.**

We have identified the following subtasks for Step 2 (identifying phases):

2.1 Identify the phases as defined in the environment studied. At this stage, it is important to perform a bottom-up analysis and avoid mapping (conciously or not) an *a priori* external/generic maintenance process model and terminology.

2.2 Each artifact (e.g., document, source code) which is input or output of each phase has to be identified. The taxonomy in Figure 5 is used for this purpose.

2.3 The personnel in charge of producing and validating the output artifacts of each phase must be identified and located in the organization model defined in Step 1.

Phase 1. Change analysis	
Input:	change requests from software owner and priority list
Output:	Release Content Review (RCR) document which contains change design analysis, prototyping, and cost/benefit analysis that may result in changes in the priority list that will be discussed with the software owner/user.
Activities:	UND, IA, CBA, CD, some CC, UT and IT for prototyping

Phase 2. RCR meeting	
Input:	draft of Release Content Review document proposed by maintainers is discussed, i.e., change priority, content of release.
Output:	Updated Release Content Review document
Activities:	AR, SA (QA engineers are reviewing the release documents and attending the meeting)

Phase 3. Solution analysis	
Input:	updated Release Content Review document
Output:	devise technical solutions based on prototyping analysis they performed in Step 1, Release Design Review (RDR) document.
Activities:	SC, CD, CC, UT (prototyping), (preparation of test strategy for) IT (based mainly on functional testing: equivalence partitioning)

Phase 4. RDR meeting	
Input:	RDR documentation
Output:	approved (and possibly modified) RDR documentation
Activities:	review and discuss CC, UT, (plan for) IT, SA

Phase 5. Change implementation and test	
Input:	RDR and prototype solutions (phases 1, 3)
Output:	changes are completed and unit tested; change validation test is performed with new reconfigured system (integration test); formal inspections are performed (when quality of code and design allows it) on new or extensively modified components; some (usually superficial) regression testing is performed on the new system to insure minimal quality before AT; a report with the purpose of demonstrating that the system is ready for AT is produced: Acceptance Test Readiness Review document (ATRR)
Activities:	CC, UT, IC, IT, RT, USD, SA

Phase 6. ATRR meeting	
Input:	Acceptance Test Readiness Review document
Output:	The changes are discussed and validated and the testing strategy used is discussed. The acceptance test team presents its acceptance testing plan.
Activities:	review the current output of IT, SA

Phase 7. Acceptance test	
Input:	the new GTDS release and all release documentation
Outputs:	A list of Software Change Requests (SCRs) is provided to the maintainers. These changes correspond to inconsistencies between the new reconfigured release and the general system requirements.
Activities:	RT, AT

Figure 11. **Overview of the Process Model**

The process shown in Figure 11 represents our partial understanding of the working process for a release of GTDS and the mapping into standard generic activities (using the taxonomy in Figure 6). This combines the information gained from Steps 2 and 3 of the assessment process. Activity acronyms are used as defined in Figure 6. In this case, each phase milestone in a release is represented by the discussion, approval and distribution of a specific release document (which are defined in the taxonomy shown in Figure 5).

4.3 Selecting Releases for Analysis

Step 4 Select one or several past releases for analysis.

We need to select releases on which we can analyze problems as they are occurring and thereby better understand process and organization flaws. However, because of time constraints, it is sometimes more practical to work on past releases. We present below a set of guidelines for selecting them:

- Recent releases are preferable since maintenance processes and organizational structure might have changed and this would make analyses based on old releases somewhat irrelevant.
- Some releases may contain more complete documentation than others. Documentation has a very important role in detecting problems and cross-checking the information provided by the maintainers.
- The technical leader(s) of a release may have left the company whereas another release's technical leader may still be contacted. This is a crucial element since, as we will see, the causal analysis process will involve project technical leader(s) and, depending on his/her/their level of control and knowledge, possibly the maintainers themselves.

The release selected for analysis in the case study was quite recent, most of the documentation identified in Step 2 was available, and most importantly, the technical leader of the release was available for additional insights and information.

4.4 Causal Analysis

Step 5 Analysis of the problems that occurred while implementing the software changes in the selected releases.

For each software change (i.e., error correction, enhancement, adaptation) in the selected release(s), information should be gathered about the difficulty of the change and any problems that arose during the implementation of the change. This information can be acquired by interviewing the maintainers and/or technical leaders and by reading the related documentation (e.g., release intermediate and final delivrables, error report forms from system and acceptance test)

This step, like several of the previous steps, is iterative, and can thus be mapped into the qualitative analysis process shown in Figure 1. This step usually corresponds to a set of iterations of the qualitative analysis process. The *input* to causal analysis (the contents of the *Observational Database* (ODB)) consists of the results of interviews, change request forms, release delivrables, the organization model (from Step 1), the process model (from Step 2 and 3), and maintenance standards definition documents. The *output* of each iteration is the actual results of the causal analysis, described in the next section. These results constitute the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify that the taxonomies of errors and maintenance process flaws are adequate, i.e., unambiguous, disjoint and exhaustive classes. This is checked against actual change data and validated during interviews with maintainers.

The following subtasks of Step 5 define the types of information that should, to the extent possible, be gathered and synthesized during causal analysis. For each software change implemented:

5.1. Determine the difficulty or error-proneness of the change.
5.2. Determine whether and how the change difficulty could have been alleviated or the error(s) resulting from the change avoided.
5.3. Evaluate the size of the change (e.g., # components, LOCs changed, added, removed).
5.4. Assess discrepancies between initial and intermediate planning and actual effort / time.
5.5. Determine the human flaw(s) (if any) that originated the error(s) or increased the difficulty related to the change (using the taxonomy shown in Figure 7).
5.6. Determine the maintenance process flaws that led to the identified human errors (if any), using the taxonomy of maintenance process flaws proposed in Figure 8.
5.7. Try to quantify the wasted effort and/or delay generated by the maintenance process flaws (if any).

The knowledge and understanding acquired through steps 1-3 of the assessment process are necessary in order to understand, interpret and formalize the information in substeps 5.2, 5.5 and 5.6. The guide in Figure 9 facilitates subtasks 5.1-5.4.

Step 5 involved a causal analysis of the problems observed during maintenance and acceptance test of the release studied. These problems were linked back to a precise set of issues belonging to taxonomies presented in Figures 7 and 8. Figure 12 summarizes Step 5 as instantiated for this case study. This step required extensive collaboration from the GTDS maintenance task leader, as well as examination of the documents generated during the release process. A questionnaire was also used to gather additional information regarding changes that were part of the release studied. The questionnaire used the taxonomies presented for Step 5. Changes that generated error correction requests from the acceptance testing team were analyzed in particular detail.

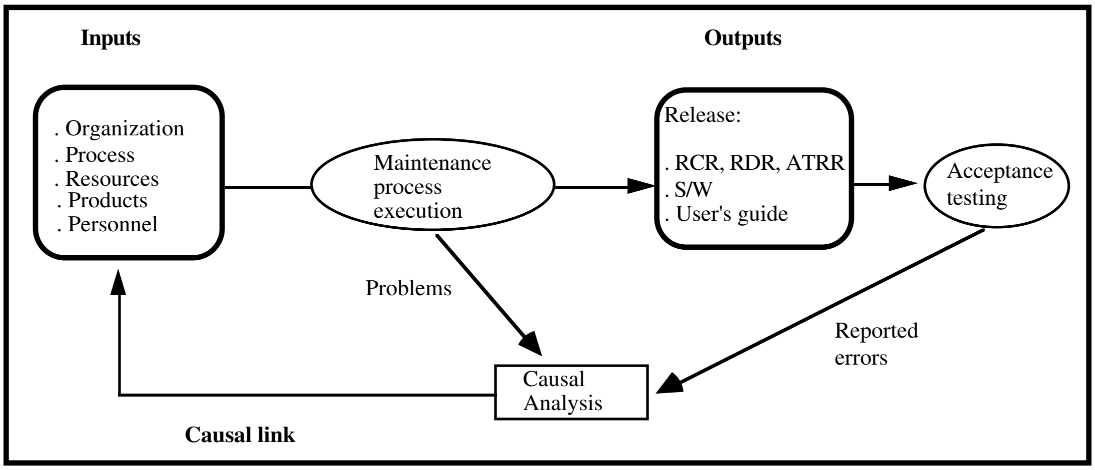

Figure 12: Causal Analysis Process

In order to illustrate Step 5, we provide below an example of causal analysis for *one* of the changes in the selected release (change 642). Implementation of this change resulted in 11 errors that were found by the acceptance test team, 8 of which had to be corrected before final delivery could be made. In addition, a substantial amount of rework was necessary. Typically, changes do not generate so many subsequent errors, but the flaws that were present in this change are representative of maintenance problems in GTDS. In the following paragraphs, we discuss only *two* of the errors generated by the change studied (errors A1044 and A1062).

Change 642 Description: Initially, users requested an enhancement to existing GTDS capabilities. The enhancement involved vector computations performed over a given time span. This enhancement was considered quite significant by the maintainers, but users failed to supply adequate requirements and did not attend the RCR meeting. Users did not report their dissatisfaction with the design until ATRR meeting time, at which time requirements were rewritten and maintainers had to perform rework on their implementation. This change took a total of 3 months to implement, of which at least 1 month was attributed to rework.

Maintenance process flaw(s):

Organizational: a lack of clear definitions of the prerogatives/duties of users with respect to release document reviews and meetings (roles), and a lack of enforcement of the release procedure (process conformance); methodological: incomplete, ambiguous definitions of change requirements.

Errors caused by change 642

The implementation of the change itself resulted in an error (A1044) found at the acceptance test phase. When the correction to A1044 was tested, an error (A1062) was found that could be traced back to both 642 and A1044.

A1044

Description: Vector computations at the endpoints of the time span were not handled correctly. But in the requirements it was not clear whether the endpoints should be considered when implementing the solution.

Error origin: change requirement analysis

Error domain: ambiguous and incomplete requirements

Maintenance process flaw(s):

Organizational: communication between users and maintainers, due in part to a lack of defined standards for writing change requirements; methodological: incomplete, ambiguous definitions of change requirements.

A1062

Description: One of the system modules in which the enhancement change was implemented has two processing modes for data. These two modes are listed in the user manual. When run in one of the two possible processing modes, the enhancement generated a set of errors, which were put under the heading A1062. At the phase these errors were found, the enhancement had already successfully passed the tests for the other processing mode. The maintainer should have designed a solution to handle both modes correctly.

Error origin: change design analysis.

Error domain: lack of application domain knowledge.

Maintenance process flaw(s):

Personnel-related: lack of experience and/or training with respect to the application domain.

4.5 Synthesis

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.

Based on the results from Step 5, further complementary investigations (e.g., measurement-based), related to specific issues that have not been fully resolved by the qualitative analysis process, should be identified. Moreover, a first set of suggestions for maintenance process improvement should be devised.

The lessons learned are classified according to the taxonomy of maintenance flaws defined in Figure 8. By performing an overall analysis of the change causal analysis results (Step 6), we abstracted a set of issues detailed in the following sections.

4.5.1. Organization

- There is a large communication cost overhead between maintainers and users, e.g., release standard documentation, meetings, and management forms. In an effort to improve the communication between all the participants of the maintenance process, non-technical, communication-oriented activities have been emphasized. At first glance, this seems to represent about 40% (rough approximation) of the maintenance effort. This figure seems excessive, especially when considering the apparent communication problems (next paragraph).
- Despite the number of release meetings and documents, disagreements and misunderstandings seem to disturb the maintenance process until late in the release cycle. For example, design issues that should be settled at the end of the RDR meeting keep emerging until acceptance testing is completed.

As a result, it seems that the administrative process and organization scheme should be investigated in order to optimize communication and sign-off procedures, especially between users and maintainers.

4.5.2. Process

- The tools and methodologies used have been developed by maintainers themselves and do not belong to a standard package provided by the organization. Some ad hoc technology transfer seems to take place in order to compensate for the lack of a global, commonly agreed upon strategy.
- The task leader has been involved in the maintenance of GTDS for a number of years. His expertise seems to compensate for the lack of system documentation. He is also in charge of the training of new personnel (some of the easy changes are used as an opportunity for training). Thus, the process relies heavily on the expertise of one or two persons.
- The fact that no historical database of changes exists makes some changes very difficult. Maintainers very often do not understand the semantics of a piece of code added in a previous correction. This seems to be partly due to emergency patching for a mission which was not controlled and cleaned up afterwards (this has recently been addressed), a high turnover of personnel, and a lack of written requirements with respect to performance, precision and platform configuration constraints.
- For many of the complex changes, requirements are often ambiguous and incomplete, from a maintainer's perspective. As a consequence, requirements are often unstable until very late in the release process. While prototyping might be necessary for some of them, it is not recognized as such by the users and maintainers. Moreover, there is no well defined standard for expressing change requirements in a style suitable for both maintainers and users.

4.5.3. Products

- System documentation other than the user's guide is not fully maintained and not trusted by maintainers. Source code is currently the only reliable source of information used by maintainers.
- GTDS has a large number of users. As a consequence, the requirements of this system are varied with respect to the hardware configurations on which the system must be able to run, the performance and precision needs, etc. However, no requirement analysis document is available and maintained in order to help the maintainers devise optimal change solutions.
- Because of budget constraints, there is no document reliably defining the hardware and precision requirements of the system. Considering the large number of users and

platforms on which the system runs, and the rapid evolution of users' needs, this would appear necessary in order to avoid confusion while implementing changes.

4.5.4. People

- There is a lack of understanding of operational needs and constraints by maintainers. Release meetings were supposed to address such issues but they seem to be inadequate in their current form.
- Users are mainly driven by short term objectives which are aimed at satisfying particular mission requirements. As a consequence, there is a very limited long term strategy and budget for preventive maintenance. Moreover, the long term evolution of the system is not driven by a well defined strategy and maintenance priorities are not clearly identified.

4.5.5. General Recommendations

As a general set of recommendations and based on the analysis presented in this paper, we suggested the following set of actions to the GTDS maintenance project:

- A standard (that may simply contain guidelines and checklists) should be set up for defining and documenting change requirements. Both users and maintainers should give their input with respect to the content of this standard since it is intended to help them communicate with each other.
- The conformance to the defined release process should be improved, e.g., through team building and training. In other words, the release documents and meetings should more effectively play their specified role in the process, e.g., the RDR meeting should settle all design disagreements and inconsistencies.
- Those parts of the system that are highly convoluted as a result of numerous modifications should be redesigned and documented for more productive and reliable maintenance. Technical task leaders should be able to point out the sensitive system units.

5. Quantitative Analysis

The use of quantitative data is critical to the useful analysis of development processes and organizations. Quantitative information is needed to effectively compare alternatives and to make decisions. However, as mentioned earlier, quantitative endeavors can be expensive and take time to initiate. For this reason, qualitative approaches like the one presented in this paper are necessary to obtain meaningful insights in a reasonable period of time. But qualitative analysis must be taken further to provide a basis for action. And qualitative approaches are best when they are designed to incrementally incorporate quantitative results as they become available.

There is a need to clearly define the quantitative information that needs to be collected and its relationship to organization and process models. This careful definition of data entities must take place when a quantitative measurement program is being planned and designed. The data entities themselves must be identified, along with their relevant attributes, and the relationships between entities must be defined. Entity-Relationship-Attribute (ERA) models are often used for this purpose. Such a model helps clarify data collection and analysis issues, as well as to define how the data will be stored. The partial E-R model shown in Figure 13 (we've omitted the attributes for this discussion) is a generic template that describes how quantitative information about process and organization could be stored together. This E-R model does not intend to be complete but can be refined to fit the needs of the measurement program being designed. For example, phases could be decomposed through a reflexive "Is part of" relationship between Process Phases entities. The attributes that will characterize the entities will depend on the goals of the data collection, the resources available, and specifics of the studied environment and process. Visualization, enactment, and analysis tools can be built upon such a database and provide a consistent process-centered environment for improvement.

AD models are 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.

In building the E-R model in Figure 13, we began with the entities already present in A-D models, then added others we felt were relevant for the quantitative analysis of maintenance processes and organizations. One entity that we have added in Figure 13 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.

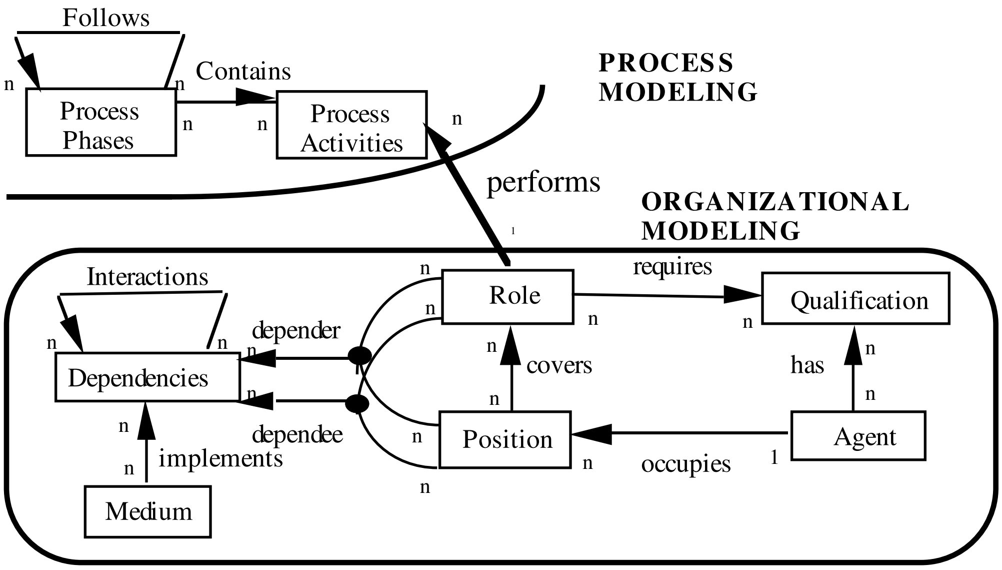

Figure 13: ER model for quantitative analysis using AD graphs

We have retained the agent/role/position decomposition of an actor defined by the A-D modeling formalism, which we found very useful. The E-R 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. Interdependent positions are usually so 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 may be used in some types of quantitative analysis. Also, dependencies are related to each via the interaction relationship, which describes the risk management mechanisms (enforcement, assurance, insurance) that are implemented between dependencies.

The E-R 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). However, 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 **Req4**). 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 that are not specified in Figure 13, e.g., process artifacts.

One type of quantitative analysis 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 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 can 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. This is just one example of how A-D models can be used along with measurement to provide quantitative results for the purposes of decision making. 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 [S94].

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.

Another example of the many possibilities for analysis of the role/position/agent structure of actors is qualification analysis where required and actual qualifications are compared for roles and positions. 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.

6. Conclusion

Characterizing and understanding software maintenance processes and organizations are necessary, if effective management decisions are to be made and if adequate resource allocation is to be provided. Also, in order to plan and efficiently organize a measurement program—a necessary step towards process improvement [BR88]—, we need to better characterize the maintenance environment and its specific problems. The difficulty of performing such a characterization stems from the fact that the people involved in the maintenance process, who have the necessary information and knowledge, cannot perform it because of their inherently partial perspective on the issues and the tight time constraints of their projects. Therefore, a well defined characterization and assessment process, which is cost-effective, objective, and applicable by outsiders, needs to be devised.

In this paper, we have presented such an empirically refined process which has allowed us to gain an in-depth understanding of the maintenance issues involved in a particular project, the GTDS project. We have been able to gather objective information on which we can base management and technical decisions about the maintenance process and organization. Moreover, this process is general enough to be followed in most maintenance organizations.

However, such a qualitative analysis is a priori limited since it does not allow us to quantify precisely the impact of various organizational, technical, and process related factors on maintenance cost and quality. Thus, the planning of the release is sometimes arbitrary, monitoring its progress is extremely difficult, and its evaluation remains subjective.

Hence, there is a need for a data collection program for GTDS and across all the maintenance projects of the organization. In order to reach such an objective, we have to base the design of such a measurement program on the results provided by this study. In addition, we need to model more rigorously the maintenance organization and processes so that precise evaluation criteria can be defined [SB94]. Preliminary results from the current maintenance measurement program can be found in [B+96].

This approach is being used to analyze several other maintenance projects in the NASA-SEL in order to better understand project similarities and differences in this environment. Thus, we will be able to build better models of the various classes of maintenance projects.

7. References

[B+95] L. Briand, W. L. Melo, C. Seaman, V. Basili. "Characterizing and Assessing a Large-Scale Software Maintenance Organization". ICSE'95, Seattle, WA, 1995.

[B+96] V. Basili, L. Briand, S. Condon, W. L. Melo, C. Seaman, J. Valett. "Understanding and Predicting the Process of Software Maintenance Releases". ICSE'96, Berlin, Germany, 1996.

[BC91] K. Bennett, B. Cornelius, M. Munro, D. Robson, "Software Maintenance", *Software Engineering Reference Book*, Chapter 20, Butterworth-Heinemann Ltd, 1991

[BK94] I.Z. Ben-Shaul and G. Kaiser, "A paradigm for decentralized process modeling and its realization in the OZ environment", ICSE 16, May 1994, Sorrento, Italy.

[BR88] V. Basili and H. Rombach,"The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Trans. Software Eng.*, 14 (6), June, 1988.

[C88] N. Chapin, " The Software Maintenance Life-Cycle", CSM'88, Phoenix, Arizona, 1988.

[CC93] B.G. Cain and J.O. Coplien, "A Role-Based Empirical Process Modeling Environment", ICSP 2, Berlin, Germany, February 1993.

[HV92] M. Hariza, J.F. Voidrot, E. Minor, L. Pofelski, and S. Blazy, "Software Maintenance: An analysis of Industrial Needs and Constraints", CSM'92, Orlando, Florida.

[K91] M.I. Kellner, "Software Process Modeling Support for Management Planning and Control", ICSP 1, Redondo Beach, CA, October 1991.

[LR93] C.M. Lott and H.D. Rombach, "Measurement-based guidance of software projects using explicit project plans", *Information and Software Technology*, 35:6/7, June, 1993, pp. 407-419.

[MCC92] "Deva, A Process Modeling Tool", MCC Technical Report, June 1992.

[R92] G.L. Rein, "Organization Design Viewed as a Group Process Using Coordination Technology", MCC Technical Report CT-039-92, February 1992.

[RUV92] D. Rombach, B. Ulery and J. Valett, "Toward Full Cycle Control: Adding Maintenance Measurement to the SEL", Journal of systems and software, May 1992.

[S94] C.B. Seaman, "Using the OPT improvement approach in the SQL/DS development environment", in *Proceedings of CASCON '94*, IBM Canada Ltd. Laboratory Centre for Advanced Studies and National Research Council of Canada, Toronto, Canada, October 1994.

[SB94] C. Seaman and V. Basili, "OPT: An Approach to Organizational and Process Improvement", AAAI 1994 Spring Symposium Series, Stanford University, March 1994.

[SS92] A. Shelly and E. Sibert, "Qualitative Analysis: A Cyclical Process Assisted by Computer", *Qualitative Analysis*, pp 71-114, Oldenbourg Verlag, Munich, Vienna, 1992

[YM93] E. Yu and J. Mylopoulos, "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 1993.

[YM94] E. Yu and J. Mylopoulos, "Understanding 'why' in software process modeling, analysis, and design", ICSE 16, Sorrento, Italy, May 1994.