# Modeling and Improving an Industrial Software Process

Sergio Bandinelli, Alfonso Fuggetta, *Member, IEEE*,
Luigi Lavazza, Maurizio Loi, and Gian Pietro Picco

*Abstract*— **This paper discusses the problems that a software development organization must address in order to assess and improve its software processes. In particular, the authors are involved in a project aiming at assessing and improving the current practice and the quality manual of the "Business Unit Telecommunications for Defence (BUTD)" of a large telecommunications company. The paper reports on the usage of formal process modeling languages to detect inconsistencies, ambiguities, incompleteness, and opportunities for improvement of both the software process and its documentation.** [0]

*Keywords*— **Software process, process-centered software engineering environments (PSEEs), process modeling, process modeling languages, process assessment, process improvement.**

## I. Introduction

Software processes are complex entities whose performance in terms of costs, expected quality of delivered products, and development time are more and more critical. Even if the technology supporting software production is undergoing an impressive evolution, the productivity of software development processes is still difficult to estimate and improve. In many cases, management methods and technologies have not been able to substantially improve the way software development projects are tracked, controlled, and improved. At the same time, investments in software are continuously increasing worldwide.

Software process improvement, therefore, has become one of the most important targets for many industrial and research initiatives [23]. From a methodological point of view, there has been an increasing interest in the development of frameworks and guidelines to support the evaluation of software process maturity and to identify strategies and key areas for improvement.

- The SEI Capability Maturity Model (CMM) is a framework to assess process maturity ([27] and [28]). It is based on the assumption that there are five possible levels of maturity for a software process. At the first level, software production activities are carried out in a chaotic and uncontrolled way. By progressively introducing techniques and methods such as configuration management, project management, explicit process definition, quality control, and quantitative process and product evaluation, it is possible to increase process maturity. At level 5, a software process is continuously improved by carefully assessing quantitative data that guide the introduction of new technology and the enhancement of the process organization and structure. The SEI CMM is one of the models currently being used to assess and select contractors [30].
- Bootstrap [6] is an ESPRIT project funded by the European Union that aims at improving and extending the SEI approach by merging it with the ISO 9000 and ESA (European Space Agency) standards. The final goal is to produce an assessment methodology to guide and support European software companies in improving their maturity and capability.
- Another important contribution to process improvement is the Japanese strategy for quality enhancement, called Kaizen [16]. Kaizen means *continuous improvement*. It is the basis of the Japanese approach to industrial production and is applied in a large variety of development activities. Kaizen is certainly an important factor also in Japanese software processes [9]. The Kaizen approach is based on few, very simple principles. First, human resources are the most important company asset. Second, processes must evolve by gradual improvements rather than by radical changes. Third, improvement must be based on statistical/quantitative evaluation of process performance.
- A similar approach is advocated by Basili's Quality Improvement Paradigm (QIP) [5]. QIP is a framework to guide and support process improvement. It is based on four steps: characterization of the process maturity; definition of quantifiable goals and of a corresponding improvement strategy; evaluation of the achievements based on measures of process performance; packaging of the experience, dissemination, and reuse within the software development organization.

As for technology, in recent years there has been an increasing interest in the development of a new generation of software engineering environments based on the explicit definition of the software process ([8], [13]). This explicit definition (called *process model*) can be used by the environment to guide software developers, to automate activities where no human intervention is required, to provide information and data on the performance and state of the process. For this reason, these environments are often called Process-centered Software Engineering Environments

This work has been partially funded by the Italian National Research Council (CNR - Progetto Finalizzato Informatica).

[0] Sergio Bandinelli and Luigi Lavazza are with CEFRIEL, Via Emanueli 15, 20126 Milano, Italy.
Alfonso Fuggetta is with Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.za Leonardo da Vinci 32, 20133 Milano (Italy), and with CEFRIEL.
Maurizio Loi is with Italtel, BUTD-Q-SW, P.za Zavattari 12, 20149 Milano, Italy.
Gian Pietro Picco is with Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy.

(PSEEs). The languages used to express process models are often called *Process Modeling Languages* (PMLs).

Although based upon different motivations and approaches, all of the above methodologies and technologies are centered around the same important concept. *Software processes are complex entities that need to be described and assessed in a precise and unambiguous way* [26]. This is necessary to support process comprehension and communication, and to evaluate if and how improvements can be achieved. Indeed, this is the rationale for the requirement that both the SEI CMM and the ISO 9000 standard introduce as one of the most important factors to support improvement, i.e., explicit process description.

Urged by the problems discussed so far, software development organizations are trying to apply the above methodologies and technologies within their software processes. They are facing several difficulties ranging from budget constraints to lack of know-how and available experience in the usage of these methodologies and technologies (as we will see in Section II.E). It is more and more necessary, therefore, to "package" existing experiences and disseminate them to a wider audience.

This paper presents the results of a process improvement experience jointly carried out by CEFRIEL, a university-industry consortium for applied research, and by Italtel, the largest Italian company operating in the telecommunications market. The goal of this activity is to improve the software process used in one of the Italtel departments, the Business Unit Telecommunications for Defense (BUTD). The experiment has been focused on the improvement of the *Quality Manual* that describes the BUTD software process [17]. Improvement is expected as the result of modeling critical parts of the Quality Manual using formal process languages. In particular, finite state machines (FSM) and the SLANG PML [2] have been used. The expected result was a reduction of the inconsistencies and ambiguities in the Quality Manual and a more consistent and systematic application of the document's rules and procedures within the software process. In turn, this was expected to increase the reliability, visibility, and manageability of the process.

The paper is structured as follows. Section II surveys related experiences. Section III discusses in more detail the experiment motivations. It also provides a framework for concepts and issues, which are then exploited in the presentation of the experiment results. Section IV introduces the part of the BUTD software process that has been studied and modeled, along with a short introduction to the formalisms used for modeling it. Section V briefly presents the history of the experiment and the methodology that has been followed. Section VI discusses the experiment results and some of the benefits for the development organization, derived from the formalization activity. Finally, Section VII summarizes the main contributions of the paper and illustrates future activities.

## II. Related Work

There are several ongoing efforts and publications discussing guidelines and approaches to support software process improvement. There are very few published contributions, however, that describe real experiences and empirical studies [33]. We survey here some significant examples.

### A. The work by Kellner and Hansen

A first interesting experience is reported by Kellner and Hansen in [21]. This case study analyses the "process used by the Air Force to modify Technical Orders (TOs), i.e., user documentation, to correspond to software changes to a weapon system. [...] In particular, it was decided that the effort would focus on the Operational Flight Program (OFP) for the F16 A/B aircraft". The authors summarize the major objectives of this case study as follows:

1. Enable effective communications regarding the process.
2. Facilitate process reuse.
3. Support process evolution.
4. Facilitate process management.

The case study was centered on the formalization of the target process using Statemate, an analysis and design workbench based on an extended finite state machine formalism called Statecharts [15]. This workbench is typically used to specify reactive systems with real-time constraints. Thus, the modeling activity was carried out using a "traditional" specification language that was not specifically designed to model software processes.

This formalization activity has made it possible to achieve several interesting results:

- There has been a "notable increase in understanding of the process by those involved in executing and managing it."
- The authors have identified a set of proposals for process improvement derived from the analysis of the process model.
- The authors were able to identify requirements for the supporting technology to be used in the process, and also better ways to use the already available development tools and environments.

One of the "lessons learned", was that collecting information on the software process is, in turn, an iterative "process" requiring several rounds of interviews.

### B. The SPMS project

This experience [22] was conducted within the Software Process Modeling System project (SPMS), funded by the DARPA/STARS program, whose primary objective is "to build, demonstrate, and empirically validate an SPMS that includes capabilities for process model definition, validation, automated generation of project-specific plans with tailorable execution constraints, continuous process evaluation and model-driven enaction in the STARS Software Engineering Environment (SEE)."

Process models are represented in SPMS through special networks of process components (essentially, subnetworks which can be hierarchically organized) and process primitives (tasks, milestones, products, AND/OR operators, temporal constraints links, and phase summary links).

The basic assumption behind SPMS is that requirements, technology, system structure, and expertise evolve throughout the early life of the system. This raised the need for the exploration of the application domain, i.e., process modeling, in order to define the desired features and needs to be included in the system.

In this perspective, the very first software process modeled using SPMS was the design process of the SPMS project itself. In addition, more complex processes have been selected to test the process modeling environment. The Software First Life Cycle (SFLC), developed by IBM, was the first full life cycle model defined using SPMS. The SPMS group also described the IEEE Working Group Draft Standard P-1074.

The SPMS project team identified some basic important issues in defining a process model. First, graphical tools are needed to enter, browse, and modify process models. This makes the process modeling activity less error-prone, by giving the modeler a more intuitive understanding of the model. A second, important requirement is the provision of highly integrated tools (both project management and project development tools). A third relevant issue is concerned with process description incompleteness. The developers found that few high-level components described in SFLC and P-1074 models were refined to a satisfactory level of description. In particular, the documentation describing these life cycles were lacking information about the roles that the various process actors may play while enacting the process. The developers believed that the availability of a "view" of the process in which the different roles are clearly identified could be very helpful in obtaining and validating process models.

### C. Experiences at Raytheon's Equipment Division

Raytheon's Equipment Division started in 1988 an improvement initiative to increase the maturity of the software process used in its Software System Laboratory (SSL) [11]. This initiative was steered by a benchmarking of the software process maturity that was accomplished in 1988 using the SEI capability assessment questionnaire. In five years, the SSL has improved from level 1 to level 3 (in the SEI CMM). Current initiatives aim at approaching level 4. The approach followed by SSL has been based on a process-improvement paradigm which applies the principles of Deming and Juran ([10] and [20]).

The lessons learned and comments that can be derived from the Raytheon experience can be summarized as follows:

1. In order to justify the high investments in process improvement, it is necessary to provide a quantitative evaluation of the benefits deriving from the improvement effort.
2. The increase in process maturity makes it possible to reduce rework. In the Raytheon case, they have experienced a reduction of rework from 41% to 11% of total project costs.
3. There has been a culture shift in the organization as far as training policies are concerned. A comprehensive, fully funded training program has been put in place. This is a major change from the initial situation, where practitioners were expected to learn by autonomous initiatives or by absorbing information and notions from colleagues.

The most interesting contribution of this experience concerns the effort SSL put in the evaluation of savings deriving from achieving a higher maturity level. SSL was able to demonstrate to top-level management that improvements have yielded a $7.7 return on every dollar invested.

### D. Experiences at Corning Incorporated

Corning Incorporated has launched an improvement initiative whose goal is to increase the quality of all its departments [18]. In particular, the Information Services Division started an assessment of its software process using the SEI CMM. Based on the result of this assessment, Corning established a Process Improvement Team to develop and plan specific improvement activities. The initial results of this initiative can be summarized by the following general observations:

1. To justify funding for improvement initiatives, managers need solid economical justification for the required investment.
2. Managers' commitment is a key success factor.
3. The process improvement initiative was carried out by people who explicitly indicated interest in the effort.
4. The SEI CMM proved to be useful and reliable in planning and scheduling the improvement effort.
5. Competition is a key driving factor to stimulate improvement.
6. The decision process followed to select technologies and to guide the improvement effort is a critical factor.
7. Internal diffusion of results and achievements is needed. This must be done frequently (more than once per month). Moreover, it is necessary to explain the rationale and motivations of the different choices made during the improvement effort.
8. The improvement effort is successful because of the availability of a set of metrics that have been collected over the years at Corning. These metrics provided the grounds to quantify achievements and to justify investments in the improvement initiative. This probably means that it is not possible to wait until level 4 and 5 of the SEI CMM are reached to start collecting process measures. Quantitative evaluation of process performance should steer the improvement initiative from the very beginning (as suggested by Basili's QIP).

### E. The research by Johnson and Brodman

Johnson and Brodman have recently conducted an interesting research [19]. They have contacted a large number of small-to-medium software development organizations[1] in

[1] By "small-to-medium organizations", the authors mean either in-

order to determine the applicability of the SEI CMM to small businesses. The results illustrate the difficulties and problems that these organizations have to face while carrying out an improvement effort.

1. "There are many CMM practices that physically cannot be accomplished by a small organization [...]: Separate independent groups, e.g., SEPG (Software Engineering Process Group), CM (Configuration Management) [...]."
2. In-house training program.
3. Hierarchical management structure.

An important issue is, therefore, the staff required to support process assessment and improvement. For example, the SEI CMM suggests 2% of software professionals for the SEPG staff. In small organization this might mean a small fraction of a person.

In conclusion, the research by Johnson and Brodman has indicated that small-to-medium organizations have several problems in supporting a CMM-based improvement initiative, especially as far as the availability and cost of human resources are concerned.

### *F. The Experience Factory at Software Engineering Laboratory*

The Software Engineering Laboratory (SEL) [4] was established in 1976 as a cooperative effort among the University of Maryland, the National Aeronautics and Space Administration (NASA), and the Computer Science Corporation (CSC). Its goal is to improve the software process used within the Goddard Space Flight Center's Flight Dynamics Division. SEL is based on the Experience Factory, i.e., a framework to consistently apply a set of techniques and methods for process improvement. The main concepts supporting the Experience Factory are the Quality Improvement Paradigm (see Section I) and the Goal/Question/Metric approach for the definition/selection of appropriate software metrics [5].

During 15 years, SEL has collected data from over 100 projects. This information has been used to build predictive models and provide rationale to existing process practices. It has also been used to identify opportunities for improvement and to evaluate the impact of new technologies, methodologies, or process strategies on process performance. Two notable examples are the assessment of the Cleanroom Approach, and of Ada as a replacement of FORTRAN.

### *G. The importance of organizational issues in process improvement*

Another experience in software process assessment is reported in [32]. The assessment has been carried out in a mid-size Italian company[2] that provides services relying on internally developed software. The assessment has been based on the CMM and has also taken into account the indications offered by QIP.

The key lesson learned is that while it was reasonably easy to identify the technical areas where improvement is needed (e.g., configuration management and requirements engineering), it is much more difficult to understand how to conduct the analysis of a company in order to identify its organizational and strategical deficiencies, and how to identify and propose reasonable changes and improvement actions accordingly. In the case reported, for instance, the main problem was in the company organization, and the first step for process improvement was the reorganization of the company by services, through the merging of the software development department and the software operation department. Other organizational measures adopted concerned personnel education and requalification programs, and human resource hiring and redistribution among existing departments/areas.

In conclusion, this experience indicated that CMM and QIP did not help in the critical part of the assessment, i.e., in problems elicitation (analysis of the organization). This is one of the most complex parts of the job, that needs to be supported by suitable methods and techniques. Therefore the paper argues that it is necessary to broaden the scope of assessment/improvement methods, and to evaluate the applicability to software processes of the experiences, methods, and techniques developed in other business domains.

### *H. Some general observations*

The experiment conducted at Italtel shares many of the motivations, rationale, problems, and approaches described for previous experiments.

1. Even if Italtel is a large company (about 15000 employees), BUTD is a small organization, which accounts for about 400 people. Thus, it falls into the small-to-medium organization category discussed in the Johnson and Brodman's report. In particular, during the experiment, we had to keep the usage of BUTD internal resources as low as possible.
2. Italtel management asks for some justification of the investment in improvement initiatives (as in the Raytheon and Corning cases). Thus, it was crucial for our project to prove that we were able to enhance the quality of the BUTD documentation describing the software development process and to facilitate the definition and collection of suitable process metrics.
3. We have exploited the usage of formal PMLs (as in the Kellner and Hansen's experiment, and in the SPMS project) to reduce inconsistencies and ambiguities in the process documentation.
4. We followed an incremental approach (as suggested by the QIP) in defining goals and the related improvement strategy.

The next section will focus on the specific problems that we have found in the BUTD software process, and on the contributions of our experiment.

dependent companies or software development organizations operating within a parent company. The maximum number of employees in the selected organizations was 500, while in most cases the number of employees was less than 40.

[2] The identity of this company is confidential.

## III. The BUTD software process

### A. State of practice

The problems and open issues in the BUTD software process can be characterized as follows:

- Software development projects carried out by BUTD must comply with severe security and quality control procedures. In particular, BUTD is periodically assessed for compliance with international standards, such as NATO AQAP-13, MIL-STD 2167A, and ISO 9001-3.
- Even if all of the above standards require a written and detailed documentation of the software process (i.e., the *Quality Manual*), process agents (i.e., managers, programmers, software engineers, etc.) experience discrepancies between the contents of the Quality Manual and the actual process followed in daily operations. This is due to the informality of the Quality Manual (it is written in natural language prose). Few parts of the software process are described using formal notations (e.g., the states of several important configuration items are described using finite state machines[3]). The almost generalized usage of an informal notation generates inconsistencies and ambiguities in the Quality Manual that make it difficult to effectively use it.
- It is not easy to identify, package, and reuse experiences. Innovations in the process are difficult to describe, and the corresponding modifications of the Quality Manual are difficult to accomplish.

There are several, more specific phenomena that we have observed in the BUTD process. The Quality Manual contains several deficiencies that can be coarsely classified as follows:

1. *Inconsistencies.* The same concept is described by conflicting statements in different parts of the Quality Manual. For example, the diagrams describing flows of information and their textual descriptions are not always consistent.
2. *Underspecified parts.* We may roughly distinguish them in incompleteness and ambiguities. Process documentation is said to be incomplete if entire parts of process descriptions are missing. For example, the Quality Manual provides no indication about the different steps in the evolution of a modification request. An ambiguous piece of process documentation, leaves the choice among different alternatives open, without making clear how to select among them. For example, the Quality Manual does not specify explicitly when the requalification of an updated document has to be accomplished.
3. *Poor organization.* The process documentation lacks adequate organization and needs restructuring to improve its readability and clarity (which does not imply changes in the process semantics). For example, the documentation of many human resource management aspects, such as criteria for task assignment, is separated and scattered in the Quality Manual. This information may be restructured by organizing it in an unique centralized description.

These problems have been spotted in many organizations and have already been discussed in several workshops and conferences (see for example the $8^{th}$ edition of the International Software Process Workshop [31], or the International Workshop on Software Process Evolution [25]). They are mainly caused by the complexity and size of processes, the very high costs to maintain quality manuals (in terms of quality and quantity of the resources needed), and the lack of an effective support to guide, speed-up, and, whenever possible, automate the process itself.

The first effect of this situation (see Figure 1), confirmed by the BUTD experience, is that process agents very often have a perception of the Quality Manual (i.e., the *Official Process*) that does not correspond to what this document is supposed to convey. Each process agent's view of the Official Process can be called his/her *Perceived Process.* The Official Process, in turn, is the formalization of the Process Owner's *Desired Process.* The Process Owner is responsible for the software process under consideration. He/she determines the policies, rules, tools, and any other choice pertaining to the software process. The Official Process might differ from the Desired Process, since, in general, it is difficult to correctly capture the Process Owner's requirements and choices.

In many cases, it is also possible to spot another interesting phenomenon. Each process agent has in general a poor visibility of the *Actual Process*, i.e., the actual evolution of software production activities ("process performances", according to Dowson and Fernström's terminology [12]). What the agent sees (the *Observed Process*) is something that is in general different from both the Perceived Process, the Actual Process, and the Desired Process. In this situation, the Actual Process (i.e., what really happens) tends to progressively diverge from the Desired Process (i.e., what the process owner would expect to happen). In fact, agents' actual behavior is determined by a combination of the Observed Process (i.e., what they see) and the Perceived Process (i.e., what they understand it ought to be done), along with agents' stimuli, mistakes, and personal preferences. This behavior, in turn, influences the Actual Process. Eventually, we have an unstable feedback system in which stability can only be enforced by reducing the sources of divergencies among the Desired, Official, Observed, Perceived, and Actual processes.

### B. Rationale and goals of the experiment

In order to improve the maturity of its software process and eliminate or reduce the problems discussed in the previous section, the BUTD management decided to launch an improvement initiative with four specific objectives:

1. Increase the conformity of the Official Process to the Desired Process.
2. Reduce the inconsistencies, incompleteness, and ambiguities in the Quality Manual.

[3]Typical configuration items are specification documents, anomaly reports, change proposals, source modules.

3. Reduce the discrepancies between the Quality Manual and the Actual Process followed in daily activities.
4. Establish effective means and procedures to continuously update and improve the Quality Manual according to the best practices and innovations experienced in the process.

*The basic assumption that has generated and driven the work discussed in this paper is that the adoption of formal process modeling languages can help reduce the divergence among the Desired, Official, Perceived, Observed, and Actual processes.* To prove this hypothesis, CEFRIEL and BUTD decided to use process formalisms to model a significant part of the BUTD software process, namely, the management of anomaly reports. In particular, the process modeling language SLANG [2] and finite state machines (FSMs) have been used as formal notations to model the process. The expected benefits from the introduction of formal process languages to describe the Quality Manual can be detailed as follows:

- Inconsistencies and ambiguities in the process documentation can be removed or reduced (e.g., as a direct consequence of the formalization process).
- Formal process modeling languages can help improve the organization and structure of process documentation.
- It is easier to assess the weaknesses of the process and to identify opportunities for improvement.
- Discrepancies between the Desired Process, the Official Process, and the Perceived Process can be reduced or, at least, more clearly emphasized.
- Each agent's Observed Process can be more easily checked against the formal definition, thus facilitating the identification of deviations and misunderstandings.
- By introducing environments based on the formal definition of the Official Process ([8], [13]), it is possible to reduce and better control the discrepancies between the Official Process and the Actual Process.

In conclusion, the work discussed in this paper shares several objectives of the Kellner and Hansen's case study. Its main goal, however, is *to enhance the level of process comprehension, by integrating and reconciling process agents' views and the Official Process definition.*

## IV. Baseline of the experiment

BUTD develops complex telecommunication systems. Since its products are mostly embedded systems, both hardware and software development issues must be addressed, taking into account specific development processes and integration aspects. For these reasons, BUTD is organized in two departments — BUTD Software and Hardware Departments — with different Official Processes.

Our activity has taken into account only the software process used in the BUTD Software Department. Nevertheless, the Quality Manual used in this division was still too large for our purpose. Thus, we decided to limit the scope of the work to one of the most critical part of the BUTD software process, namely, anomaly management.

This section briefly describes the activities involved in BUTD anomaly management together with the basic concepts of the formalisms used to describe them.

### A. The BUTD anomaly management

BUTD Software Quality Manual defines an anomaly as a "*software product irregularity with respect to the expected behavior*". Generally, anomalies are discovered during verification & validation activities. Anomaly management consists of the set of procedures and actions to be performed in order to communicate anomalies and their related information, evaluate them, and eventually remove them from the software product. This involves the modification and testing of different software product components. Anomaly management includes also configuration management, since multiple versions of the same software components have to be managed consistently. Summing up, BUTD anomaly management defines a typical *change management* process, whose critical impact on software process quality has been widely pointed out in the literature [24].

The anomaly management policies are partly described by means of finite state machines. This is to improve the document readability by reducing and, whenever possible, eliminating ambiguities. The documents used in the anomaly management process are *Anomaly Reports* (ARs), whose states are described by the FSM of Figure 2. An AR is a form filled by the process agent who detected the anomaly. It illustrates anomaly characteristics, along with other relevant information such as the phase in the life cycle when the anomaly was originated, the expected severity, and the expected effort to solve it. This form is sent to the configuration management group (SGMR[4] in BUTD jargon), which is in charge of handling it. The action of sending the AR form is the formal event that actually starts anomaly management.

According to the Quality Manual, anomaly management consists of the following activities:

1. SGMR checks the formal correctness of the AR. For instance, it controls that all the relevant fields are filled. This operation is performed whenever a configuration item is sent to SGMR. Upon acceptance by the SGMR, the state of the AR is set to *Originated*, the AR is stored under configuration control, and notifications to involved agents (e.g., members of the quality control board and implementors) are issued. Finally, the anomaly state of the AR is set to *Notified.*
2. The quality control board (SVVP[5] in BUTD terminology) summons the Change Control Board (CCB). This is a board of experts who have both the competence and the authority to determine the actions to be performed in order to evaluate and eventually solve the anomaly.

[4] SGMR is the Italian acronym for *Sottogruppo Gestione Modifiche e Rilasci.*

[5] SVVP is the Italian acronym for *Sottogruppo Verifica e Validazione di Progetto.*

3. The CCB examines the anomaly. The following cases may occur:
   - The anomaly is found not to be a real anomaly. Namely, there has been an erroneous evaluation of the originator who has misinterpreted some information or result. The anomaly becomes *Rejected* and the anomaly management process ends.
   - The anomaly is recognized as a real one, but its solution is delayed until a future CCB meeting decides to start the anomaly solution. The AR state becomes *Suspended*.
   - The anomaly is recognized as a real one and a set of update requests are issued, described in *Modification Requests* (RIMO[6] in the BUTD jargon) forms. The AR state is changed to *Pending*.
4. Development and Qualification groups implement the modifications stated in RIMO forms, performing the tests indicated by the CCB. Then the updated software configuration items along with the test results (*Test Launch Reports* — TLRs in BUTD jargon) are sent to SGMR, which puts them under configuration control.
5. A review team is summoned, to perform verification & validation[7] activities on both the updated items and the TLR, taking into account RIMO forms. The review ends by issuing a document, the *Verification & Validation Report* (VVR), which specifies if all the requested modifications have been performed completely and correctly.
6. The SGMR receives the VVR, evaluates it and determines the appropriate state transitions for RIMOs, ARs, and any other related items.

Anomaly management, as described in BUTD Quality Manual, is a complex process involving many activities. To have a complete and coherent view of anomaly management, we have also modeled the process of allocating human resources to maintenance and requalification activities. No information about these aspects of the process was provided in the Quality Manual, despite their obvious importance as far as process management and control is concerned. Consequently we had to model them "from scratch", based on the process owner's knowledge.

### B. The SLANG process modeling language

SPADE (Software Process Analysis, Design and Enactment) is a Process-centered Software Engineering Environment, currently being developed at CEFRIEL and Politecnico di Milano [1], [2], [3]. SPADE provides a domain-specific language for modeling software processes called SLANG (Spade LANGuage). This section provides only a brief overview of the language. The complete language definition is available in [2]. SLANG is based on high-level Petri nets and is given formal semantics in terms of a translation scheme from SLANG objects into ER nets[14], a mathematically defined class of high-level Petri nets that provide the designer with powerful means to describe concurrent and real-time systems.

A SLANG process model consists of a set of type and activity definitions. They are used to describe process data and process activities respectively.

Type definitions are organized in a generalization hierarchy following an object-oriented style. Attributes and methods can be defined for each type. Figure 3 presents a user-defined SLANG type hierarchy, including some of the types from the BUTD model. As an example, consider the definition of type *ItemObjType*:

```
class ItemObjType inherit ModelType
    type tuple (
        ID:           integer;
        Kind:         string;
        Status:       StatusType;
        ToBeTested:   Boolean;
        Distrib:      list(Person))
end;
```

*ItemObjType* represents a generic object under version and configuration control. This type has several subtypes which define specific configuration items (e.g., source code files and documents). Its attributes have the following meaning:

- *ID* is an item identifier.
- *Kind* indicates the type of object (RIMO, TLR, etc.).
- *State* indicates the item's state with respect to the corresponding FSM.
- *ToBeTested* is TRUE if tests may be applied to the item (e.g., source code), FALSE otherwise (e.g., a requirement document).
- *Distrib* is the list of people that have to be notified of item's changes.

Activity definitions are structured in a hierarchical manner as well, with one root activity at the top of the hierarchy. Each activity encapsulates a set of logically related process steps and may include invocations of other (simpler) activities. Activities at the leaves of the hierarchy are defined only in terms of high-level Petri nets and do not include invocations to other activities.

An activity state is represented by a Petri net marking, i.e., an association of tokens with places. Tokens are typed objects and may model documents, tools, resources, test data, programs, etc. Places are distributed persistent object repositories; they are typed and may only contain tokens of the declared type. Transitions represent events that may, or may not, occur in a given state. A transition firing represents the occurrence of an event, taking a negligible amount of time. Arcs connect transitions to places and places to transitions. The arcs' weights indicate the number of tokens that may flow through them at each transition firing. Weights can be statically defined (the default value is 1), or dynamically computed. In the latter case, the arc weight is indicated by a "*", and it models events consuming a token set, whose cardinality is not statically known. In addition to "normal" arcs, SLANG provides two special kinds of arcs: *read-only* and *overwrite* arcs. A read-only

[6] RIMO stands for *RIchiesta di MOdifica*.

[7] The terms review and verification & validation are interchangeable within the BUTD terminology.

arc, represented by a dashed arrow, may connect a place to a transition. The transition can read and use token values from the input place in order to evaluate the guard and the action, but tokens are not removed. An overwrite arc, represented by a double arrow, may connect a transition to a place. When the transition fires the output place is emptied of all its tokens. Then, the token(s) produced by the firing are inserted in the output place. The overall effect is that the produced tokens *overwrite* the previous contents of the output place.

The net topology describes precedence relations among events; it also describes parallelisms and conflict situations. Each transition is associated with a guard and an action. The transition's guard is a predicate on input tokens and is used to decide if an input token tuple enables the transition. The dynamic behavior of a transition is described by the *firing rule*. The firing rule states that when a transition fires, tokens satisfying the guard are removed from input places and the transition's action is executed. As a result of executing the action, a token tuple is inserted in the output places of the fired transition.

Following the principle of information hiding, each activity definition has an interface and an implementation part. The *activity interface* includes a set of interface transitions, called *starting events* or *ending events*; and a set of interface places, classified in *input places* (input to starting events), *output places* (output from ending events), and *shared places*, that are shared by different activities and play the role of communication and synchronization variables during activity execution. Figure 4 provides an example for the interface of activity *ImplementRIMO*. This activity interface has two starting events (*Start Update* and *Update Again*), two ending events (*Transfer Item&TLR* and *Transfer Item*), four input places, two output places, and three shared places.

When an activity starting event is fired, an activity instance (*active copy*) is generated and its execution is started. The execution ends when one of the ending events is fired. In particular, it is possible to instantiate the same activity several times, generating different execution threads for the activity. Each active copy enactment proceeds concurrently with the enactment of other active copies.

Software development is a multi-person process that involves the use of a large variety of software tools. A SLANG process model may also describe the interaction with tools and users. This is done with a special kind of transitions, called black transitions, and a special kind of places, called user places. A black transition represents the invocation of an external (non-SLANG) executable routine (e.g., a Unix executable file). The external routine is executed *asynchronously*, that is, once the external tool has been started, the activity execution is resumed. User places are used to capture events that occur in the external environment and are relevant to process enaction. These events can be generated by users through tools provided by the environment.

Figure 5 shows the complete SLANG definition of activity *ImplementRIMO*. The activity interface is outside the dashed box and its implementation is contained inside the box. *ImplementRIMO* models how RIMOs (i.e., modification requests) are executed, according to the guidelines described in Section IV.A. *ImplementRIMO* can be started when there is a new RIMO to consider or when there is an already processed RIMO that has to be re-considered because of some problem. In the first case, *StartUpdate* fires by consuming the new RIMO (from place *RIMOs*) and the corresponding item to be modified (from place *Items*). In the second case, *UpdateAgain* fires and consumes the RIMO that has to be reconsidered (from place *RIMONotOK*) and the related item (from place *UpdatedItem*). The activity can also be terminated in two ways. If the modified item is subject to testing (which is indicated by the boolean attribute *ToBeTested*), *ImplementRIMO* ends by firing *TransferItem&TLR* and produces the updated item together with a test launch report (*TLR*). Otherwise, if testing does not apply to the item, *TransferItem* fires and only the updated item is produced.

Activity *ImplementRIMO* invokes two (sub)activities, called *UpdateRIMOItem*, and *RequalificationOfUpdates*. *UpdateRIMOItem* produces an updated item and the RIMO with the state changed accordingly. The ending event *TransferItem* fires when no testing is required. Otherwise, *StartRequalification* fires, starting the activity *RequalificationOfUpdates*, that tests the updated item and produces a set of results in place *TLROutput*. The availability of these results allows the activity to terminate by firing the ending event *TransferItem&TLR*.

Figure 6 provides the definition of activity *RequalificationOfUpdates*. This activity receives as input a terminated RIMO and the corresponding updated item. This item can either be a generic configuration item (e.g., a source file) or a test item (e.g., test data, procedures, stubs, drivers, etc.). In the former case the tests selected by the CCB are used. Otherwise, the items that are associated with the modified test are retrieved. When the activity has collected all the needed information, black transition *RunTest* fires and calls the external tool that actually executes tests.

As a last example, we present the guard and action associated with black transition *RunTest*. Assume that place *Tests* has type *TestType* and place *Items* has type *ItemObjType*. Both types define the attribute *name*. Assume also that place *TLRs* has a type *TLRType* defining method *initializeFrom*, which initializes the object with the values read from a file whose name is given as a parameter. We can now specify the guard and action associated with transition *RunTest*:

```
TRANSITION {
  name  RunTest
  guard true
  local
    string testname;
  action
    testname = Tests->name+","+Items->Object->name;
    extAction = "TestRunTool " + testname +
                " -o" + testname + ".out";
    TLRs->initializeFrom(testname+".out"); }
```

## V. History of the experiment

The experiment was started in October 1992, as a joint effort between CEFRIEL and BUTD. The first part of the experiment was terminated in June 1993. The experiment was primarily carried out by two experts from CEFRIEL and Italtel [29]. The CEFRIEL person (the *process engineer*) is an expert in process modeling technology and methodologies, and in the process modeling formalisms used in the experiment. The Italtel person (the *process owner*) is member of BUTD management staff and has a deep knowledge of the BUTD software process.

The first phase of the experiment was centered around a preliminary study of the BUTD Quality Manual. This study originated an initial set of questions about the BUTD process, that were submitted to the process owner and other process agents. In general, to understand the characteristics, weaknesses, and inadequacies of the BUTD process, the process engineer collected, compared, merged, and organized different kinds of information. After this initial knowledge acquisition phase, the process engineer started the actual formalization effort. In particular, he used SLANG to structure the process knowledge collected during the previous phase. The output of this phase, concluded in mid November, was a first draft of the BUTD model. This first model was jointly reviewed by the process engineer and the process owner to evaluate how accurately the process was described. This initial revision pointed out that more iterations of the above "process" were necessary to refine, enrich, and complete the existing process model.

In conclusion, the modeling activity is composed of several iterations. In this iterative *meta-process* (i.e., a process whose target is another process [7]), process knowledge has to be "elicited" from a large amount of information sources through several iterations. The meta-process followed in the experiment has been therefore structured in the following phases:

1. **Knowledge elicitation**. It consists of document reading sessions and interviews with process agents, in order to extract relevant data on the BUTD software process.
2. **Formalization**. It is centered on the use of process modeling formalisms to create and refine the formal model of the BUTD process.
3. **Model review and assessment**. It consists of the revision of the model to correct flaws in the formalization and to assess the accuracy of the model with respect to the Actual Process.

The transitions among the three activities in the meta-process are motivated as follows:

- Cycling back from formalization to knowledge elicitation is needed whenever additional information is required in order to clarify or complete the description of specific process aspects.
- Cycling back from model review and assessment to formalization is needed when the review task highlights errors in the formal process model, or when additional modeling effort is required.
- Cycling back from model review and assessment to knowledge elicitation is needed when the review task shows that some part of the process has not been correctly understood, and, consequently, the formal model along with the informal knowledge has to be refined and enriched.

It is worthwhile describing how the interaction between the process engineer and the process owner and other process agents was structured. The process owner and process agents were not available full time (a very common situation!), since they were involved in several ongoing projects. In order to reduce the meeting time and to cooperate as asynchronously as possible, the interaction was managed by exploiting as much as possible the use of question lists. In many cases, questions were given as sets of SLANG fragments, representing alternative formalizations of specific BUTD process details. The process owner and other process agents had to select the correct one or to derive it by comparing the available alternatives. In general, the use of SLANG made it possible to clearly formulate hypotheses and questions, and therefore it was possible to reduce the number of face-to-face discussions. Clearly, this was feasible mainly during the model review and assessment activity. Meetings were unavoidable during the first iterations of the knowledge elicitation activity (when modeling from scratch). In this case, face-to-face meetings are more appropriate since they allow a quicker comparison and merging of (sometimes very different) viewpoints.

A satisfactory preliminary version of the model (in terms of correctness and consistency) was made available in January 1993. This version has been heavily modified in April to better cope with more advanced process issues. The final version was released at the end of June 1993 [29]. The size of the experiment can be summarized by the following figures:

- Number of pages of the BUTD Quality Manual examined: nearly 200.
- Number of question lists sent to the process owner: 6.
- Average number of questions per list: 10.
- Number of "face-to-face" meetings: 5
- Average duration of "face-to-face" meetings: 2 hours.
- Number of SLANG activities in the model: 24.
- Total number of places: 210 (21 of them are user places).
- Total number of transitions: 180 (30 of them are black transitions).

## VI. Experiment results

This section describes some results of the formalization effort which have produced remarkable improvements of the Quality Manual. Results are presented in terms of the feedback loop model of Figure 1, indicating, where relevant, the original Official Process, its differences from the Actual Process and from the Desired Process, the possible Perceived Processes and the final formal reformulation of the Official Process. The Actual Process has been derived by comparing, merging, and refining the observed processes of the process owner and of other process agents.

### A. Requalification of updated items

**Official Process**. According to the Quality Manual summarized in Section IV.A, the Qualification Subgroup starts performing the requalification of the configuration items related to the identified anomaly, as soon as they have been updated by implementors. Quoting from the Quality Manual,

> "The Qualification Subgroup has to verify the effectiveness of the modifications accomplished on the software. In order to do this, the Subgroup executes the tests defined within CCB, on the software already managed by SGMR."

**Perceived Process**. The previous statement contains some ambiguities and different interpretations may be given to it. Some obscure points are:

*a.* What does "verify the effectiveness of the modifications" mean? Are error metrics in use? Are dedicated test tools used? Is any standard procedure defined to execute tests? None of these questions are answered by the Quality Manual.

*b.* What does it mean that tests are "defined" by CCB? The process engineer proposed two interpretations:
- During CCB meetings, the whole set of test cases are designed.
- During CCB meetings, only test data are defined, leaving the definition of the needed test code (e.g., stubs and drivers) and of the way tests are to be carried out to the process agent who will actually run the tests.

*c.* Are tests always performed? The process engineer argued about a design anomaly detected at the beginning of the development phase, when no source code has been produced yet. In this case, no meaningful test can be done, even though the Quality Manual does not distinguish between this case and the detection of anomalies in the code.

**Actual Process**. The Actual Process can be described by the following scenario:

1. Standard procedures for test execution and analysis were already defined and in use. They are based on CASE tools collecting measures and recording errors. It was acknowledged, however, that no reference to this procedure was included in the Quality Manual.
2. The CCB does not define any test case. Actually, references to the test cases that have to be performed are already attached to every configuration item when it is created. As a consequence, the CCB only selects, among the test cases already defined, the ones that have to be repeated in order to guarantee that the modification is acceptable. The process of assigning test cases to a configuration item is external to anomaly management, and was considered to be not relevant.
3. Tests are not always performed, as argued by the process engineer. In some situations (e.g., when no code is involved, as mentioned above), the implementation of modification requests ends by returning only the modified configuration item, which is submitted for review. On the contrary, the general case described in Section IV.A prescribes that the implementation of RIMOs produces both an updated item and the corresponding Test Launch Reports, and the review is performed by taking into account both reports.

**New Official Process**. The elicitation process showed that there where many discrepancies between the Quality Manual description and current practice. To amend the Quality Manual, the process engineer derived a formal model in SLANG, which is shown in Figures 5 and 6 (the SLANG process model also contains the textual description of transitions, that has been omitted here for space reasons). In this model, the ambiguity described in c) has been removed, since any token stored in place *Item* always carries the boolean attribute *ToBeTested*, that indicates if the item has to be tested. If such an attribute is true, *StartRequalification* is fired, thus starting activity *RequalificationOfUpdates*. The termination of this activity causes the whole activity *ImplementRIMO* to end, producing tokens in both output places *TLR* and *UpdatedItem*. Otherwise, *ImplementRIMO* terminates by firing transition *TransferItem*, that produces just the updated item.

### B. Finite state machines for describing anomaly management

**Official Process**. A preliminary assessment of the FSM that governs the BUTD anomaly management (Figure 2) indicated that the FSM already provided a reasonable description of the Desired Process. However, before entering the state *Originated*, an AR has to be accepted by the configuration management group (SGMR). The Quality Manual literally states that:

> "When receiving the element [...] SGMR either accepts or refuses it [...]."

**Perceived Process**. Although the FSM clearly identifies the AR possible states and transitions, the process owner and the process engineer found out several underspecified issues, not taken into account by the FSM, that led to different process perceptions:

1. The meaning of "refuses" in the Quality Manual documentation is not clear. What happens when SGMR finds something incorrect in an AR form? There are different possible interpretations. Is the incorrect form stored under configuration management anyway? What is its state? Is the originator notified of the refusal? How? Sometimes, the precise identity of the originator may not be obvious. For example, if the CCB generates incorrect RIMO forms, who is to be notified? The RIMO's author or all people in the meeting?
2. According to the FSM, once an anomaly enters the *Pending* state (i.e., it is being processed) it is no longer possible to cancel it. What happens if during the solving process it is discovered that the anomaly should be either rejected or suspended? and what procedures have to be followed in this case?
3. How does an anomaly exit from the *Suspended* state? In the original FSM no transition originates from this

state, and no information was provided in the Quality Manual about this issue.

**Actual Process**. Interaction with the process owner and process agents unveiled the following situations:

*a.* An incorrect form is kept under configuration control until the new (hopefully correct) version is received. A copy of the form is re-sent to the originator. The originator is always assumed to be the drafter of the form. When the form is created during a meeting (e.g., RIMO forms in a CCB meeting), the originator is the CCB chair, who is always a SVVP member, and consequently has both the authority and competence to re-edit the form and re-send it to SGMR.

*b.* Situations where anomalies are canceled or suspended during their solution are very common, due to many factors such as changes in the project goals, unexpected system hardware/software limitations, changed time schedules. Very often, people working on the anomaly solution communicate to project managers the problems that makes it difficult or even impossible to effectively pursue the given goal. The evaluation of such complaints is made during CCB meetings.

*c.* A suspended anomaly is frozen until the CCB decides that the conditions that determined the suspension have to be reconsidered. The anomaly solution can be either resumed or canceled.

**New Official Process**. Based on this additional knowledge on the process, the process engineer modified the existing FSM for the Anomaly Report (Figure 2), producing the FSM shown in Figure 7.

The complexity of the FSM is clearly increased. The original FSM had 6 states and 5 transitions, while the modified FSM consists of 10 states and 25 transitions. This is probably unavoidable: the process is intrinsically complex and the more carefully we describe it, the more complex are the models that represent it.

### *C. Does BUTD really need SGMR?*

**Official Process**. A significant part of the documentation corresponding to configuration management activities in the Quality Manual are based on unclear definitions, written in natural language.

**Actual Process**. The process owner noticed that SGMR takes a key part, not only in anomaly management but also within the whole BUTD software process. Henceforth, BUTD is interested in keeping SGMR as efficient as possible. Most SGMR tasks are routine jobs, such as checking formal correctness of a form, or determining the status which has to be assigned to an updated configuration item. These repetitive tasks were enacted by humans, while the usage of tools was limited to handling software components in the file system.

**New Official Process**. Starting from the situation described in the Quality Manual, the process engineer produced a first version of a formal model for configuration management activities. However, as the work proceeded, the improvement of the FSM definitions and the description of a type hierarchy — as shown in Section IV.B — unveiled the possibility of further improving the process model. Consequently, the process engineer decided to redefine SGMR tasks in the SLANG model. In doing this, the contribution of human beings to the enactment of configuration management activities was more and more reduced. The formalization of the process, along with the detailed description of those aspects that were left unspecified in the Quality Manual, made it possible *to automate a part of the process by deriving a formal and enactable process model that does not require any human intervention.* As an example, Figure 8 shows the *SendItemOp* activity which accepts and manages configuration items.

### *D. Human resource allocation*

**Official Process**. The BUTD Quality Manual indicates the working units assigned to a task. These assignments, however, very often specify just a department or a group. They do not select the person who is responsible for the task. For example, the Quality Manual states:

> "If the anomaly has been accepted, and RIMOs have been issued, the Development Subgroup and the Qualification Subgroup, upon receiving RIMO notifications, have to [...]"

The Quality Manual contains no indication about how a person is assigned to a task, nor it is clear how, for example, the notification of the presence of RIMO is sent to the above groups.

**Desired Process**. This incompleteness in the process documentation leaves the door open for various different interpretations. For example, is the notification broadcast to all the people in the subgroup or just to the subgroup leader? Moreover, how are RIMO deadlines defined? how is the new task made compatible with the other tasks already assigned to the same person? and how are task conflicts managed? Despite their importance, no answer can be found in the Quality Manual for the above questions. Based on this fact, it was decided to model this activity from scratch, as far as the part directly involved in anomaly management was concerned. The anomaly management process, in fact, imposes specific constraints on the allocation procedure. For example, the CCB has both the authority and the competence to determine who has to implement a RIMO, and consequently can make the assignment.

**New Official Process**. The modeling activity was carried out jointly by the process engineer and the process owner by directly using SLANG, during a later phase of the experiment, when the process owner had gained enough knowledge about the language to easily understand and handle SLANG specifications. A high-level view of the human resource allocation activity is presented in Figure 9. The final result is a formal, consistent, unambiguous, and extensible definition of the human resource allocation process.

## VII. Conclusions and future work

Process improvement is a critical issue for any software development organization. Improvement must be based on a solid methodological and organizational background to guide and direct assessment and decision-making processes effectively. Improvement can involve different parts of the process: technology, methodology, organization. In this paper, we started from some published experiences in process improvement, ranging from technology enhancement to organizational and strategic innovation. Our experiment inherits several motivations from the surveyed experiences. It focuses, however, on a specific issue, namely, how to reduce the divergency among the different process views to increase process comprehension. We believe that this is a critical factor to make it possible to manage, control, and improve the process.

In conclusion, the main contributions of the paper can be summarized as follows:

1. Section III.A discussed the problems and difficulties that Italtel BUTD is facing in following the software process defined in its Quality Manual. To present these problems, we introduced a model to describe the different views of the software process that coexist in BUTD. This model defines the Desired, Official, Perceived, Observed, and Actual Processes. We believe that this model is significant not only for BUTD but also for other software development organizations.
2. In Section V, we have shown how the experiment has been conducted and how we have overcome the problem of interacting with people who cannot be distracted from their work.
3. In Section VI, we have shown how the adoption of formal PMLs can help reducing errors and deficiencies in the documentation (Quality Manual) of the software process. This, in turn, allows process agents to better apply process rules to actual software development activities, and also facilitates process improvement initiatives.

As an important by-product of the experiment, a detailed and careful assessment of the weaknesses and strengths of SLANG has been accomplished. SLANG is largely experimental in its present form. The experiment proved the adequateness of the basic assumptions that have driven SLANG design and provided valuable suggestions for language improvements, e.g., in terms of specific process constructs.

BUTD was satisfied with the qualitative assessment and improvement of its process. The modeling activity at BUTD is still ongoing, in order to study a larger part of the software development process. Moreover, we plan to use the SPADE process-centered software engineering environment, in which SLANG process models can be enacted, to guide and support those parts of the process that have been already modeled. In particular, one of the key issues that we are addressing in this second phase of the experiment is the assessment of SPADE as a suitable means to collect process and product-related metrics. This is essential to provide the rationale and quantitative foundations for future process improvement initiatives.

## Acknowledgment

Authors wish to thank Carlo Ghezzi for his support, and for many helpful comments and suggestions.

## References

[1] S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza, "The Architecture of the SPADE-1 Process-Centered SEE," *Proc. 3rd European Workshop on Software Process Technology*, Grenoble, France, Feb. 1994.

[2] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza, "The SLANG 1.0 Process Modeling Language Reference Manual", Technical Report RT93032, CEFRIEL, Sept. 1993.

[3] S. Bandinelli, A. Fuggetta, and C. Ghezzi, "Software Process Model Evolution in the SPADE Environment," *IEEE Trans. Software Eng. Special Issue on Process Evolution*, vol. 19, pp. 1128-1144, Dec. 1993.

[4] V.R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page, and S. Waligora, "The Software Engineering Laboratory - an operational software experience factory," *Proc. 14th Int. Conf. Software Engineering (ICSE 14)*, ACM Press, May 1992.

[5] V.R. Basili and H.D. Rombach, "The TAME Approach: Towards Improvement-Oriented Software Environments", *IEEE Trans. Software Eng.*, vol. 14, pp. 758-773, June 1988.

[6] Bootstrap Project Team, "Bootstrap: Europe's Assessment Method," *IEEE Software*, vol. 10, n. 3, pp. 93-95, May 1993.

[7] R. Conradi, C. Fernström, and A. Fuggetta, "Concepts for Evolving Software Processes," in *Software Process Modeling and Technology*, A. Finkelstein, J. Kramer, and B.A. Nuseibeh, Eds. Research Studies Press Limited, 1994, pp.9-32.

[8] B. Curtis, M.I. Kellner, and J. Over, "Process modeling," *Communications of ACM*, vol. 35, n.9, pp. 75-90, Sept. 1992.

[9] M.A. Cusumano, *Japan's Software Factories,* Oxford University Press, 1991.

[10] W.E. Deming, *Out of the Crisis,* MIT Centre for Advanced Engineering Studies, Cambridge, MA, 1982.

[11] R. Dion, "Process improvement and the corporate balance sheet," *IEEE Software*, vol. 10, n. 4, pp. 28-35, July 1993.

[12] M. Dowson and C. Fernström, "Mechanisms for Process Change," *Proc. 3rd European Workshop on Software Process Technology*, Grenoble, France, 1994.

[13] A. Finkelstein, J. Kramer, and B.A. Nuseibeh, Eds, *Software Process Modeling and Technology,* Research Studies Press Limited, 1994.

[14] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè, "A unified high-level Petri net formalism for time-critical systems," *IEEE Trans. Software Eng.*, vol. 17, pp. 160-172, Feb. 1991.

[15] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtul-Trauring, "Statemate: A working environment for the development of complex reactive systems," *Proc. 10th Int. Conf. Software Eng. (ICSE 10)*, IEEE Computer Society, 1988, pp. 396-406.

[16] M. Imai, *Kaizen - The Key to Japan's Competitive Success,* McGraw-Hill, 1986.

[17] Italtel BUTD, *Manuale del Sistema di Qualità Software*, Italtel, 1988.

[18] A. Johnson, "Software Process Improvement Experience in the DP/MIS Function," *Proc. 16th Int. Conf. Software Eng. (ICSE 16)*, IEEE Computer Society, 1994.

[19] D.L. Johnson, J.G. Brodman, "What Small Organizations Say About the CMM," *Proc. 16th Int. Conf. Software Eng. (ICSE 16)*, IEEE Computer Society, 1994.

[20] J.M. Juran, *Juran on planning for quality,* The Free Press, New York, 1988.

[21] M.I. Kellner and G.A. Hansen, "Software Process modeling: A Case Study," *Proc. 22nd Annual Hawaii Int. Conf. System Sciences*, Jan. 1989.

[22] H. Krasner, J. Tirrel, A. Linehan, P. Arnold, and W.H. Ett, "Lessons learned from a software process modeling system," *Communications of ACM*, vol. 35, n.9, pp. 91-100, Sept. 1992.

[23] R. Lai, "The move to mature processes," *IEEE Software*, vol. 10, n. 4, pp. 14-17, July 1993.

[24] N. Madhavji, "The PRISM Model of Change," *Proc. 13th Int. Conf. Software Eng. (ICSE 13)*, IEEE Computer Society Press, 1991.

[25] N. Madhavji, Ed, International Workshop on Software Process Evolution, Unpublished documentation, Mc Gill University, Montreal, Canada, Jan. 1993.

[26] L. Osterweil, "Software processes are software too," *Proc. 9th Int. Conf. on Software Eng. (ICSE 9)*, IEEE Computer Society, 1987.

[27] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "Capability Maturity Model for Software, Version 1.1," Technical Report CMU/SEI-93-TR-24, Feb. 1993.

[28] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, n. 4, pp. 18-27, July 1993.

[29] G.P. Picco, Modeling a real software process with SLANG, Master Thesis, CEFRIEL Report RI 93059, June 1993.

[30] D. Rugg, "Using a capability evaluation to select a contractor," *IEEE Software*, vol. 10, n. 4, pp. 36-45, July 1993.

[31] W. Schäfer, Ed., *Proc. 8th Int. Software Process Workshop (ISPW 8)*, IEEE Computer Society, 1993.

[32] F. Cattaneo, A. Fuggetta, and L. Lavazza, "An Experience in Process Assessment," *Proc. 17th Int. Conf. on Software Eng. (ICSE 17)*, IEEE Computer Society, 1995.

[33] A. Fuggetta, and G.P. Picco, "An annotated bibliography on software process improvement," *ACM SIGSOFT Software Engineering Notes*, July 1994.

**Sergio C. Bandinelli** received his Dr. degree in Computer Science at ESLAI, University of Lujan, Argentina in 1989. He has been instructor at the University of Buenos Aires (Argentina) and at the Politecnico di Milano (Italy). He is currently completing his PhD thesis on process-centered software engineering environments at Politecnico di Milano. He is also research assistant at CEFRIEL, a research and educational institute established in Milano by universities and industries, operating in the information technology market, and local public administrations. He is co-author of over 20 scientific papers. His research interests are in the field of software engineering, specifically in process modeling, process improvement, software engineering environments, and software architectures.

**Alfonso Fuggetta** received his Dr.Eng. degree in Electrical Engineering from Politecnico di Milano, where he holds the position of Associate Professor of Software Engineering. He is also Senior Researcher and member of the operating board of CEFRIEL. His research interests are in software process modeling, executable specifications, and architectures of advanced software engineering environments. Alfonso Fuggetta has been member of the programme committee for international conferences and workshops and is member of the Steering Committees of the European Software Engineering Conference (ESEC) and of the European Workshop on Software Process Technology (EWSPT). He is co-author of over 40 scientific papers. He is member of the editorial board of the journal "Software Process: improvement and practice", published by J. Wiley. He is member of ACM and IEEE.

**Luigi Lavazza** received his Dr.Eng. degree in Electrical Engineering from Politecnico di Milano in 1984. From 1984 to 1988 he was with TXT (Milan) where he was involved both in industrial software development and research projects. In 1989 - 90 he was with AITEC (Milan) where he participated in the definition of the company's software development standards. In 1990 he joined CEFRIEL where he is now in charge of the Software Engineering research group. He is co-author of over 30 scientific papers. His research interests include advanced software engineering environments, software process modeling and improvement, databases for software engineering and object-oriented technology.

**Maurizio Loi** received his Dr.Eng. degree in Electrical Engineering from Politecnico di Milano in 1973. He worked in the development of large real-time software systems for major Italian communications companies, in cooperation with U.S. and European associates. In 1985 he participated in an ESPRIT project for the design of a software engineering support environment (SPMMS). Recently he joined Italtel SIT, as responsible of software quality for the Business Unit Telecommunication for Defence. He is responsible for the implementation and maintenance of the software quality manual, in compliance with the international military standards, the systematic application of product and process metrics, and the activity of quality control of the current projects.

**Gian Pietro Picco** received the Dr.Eng. degree in electrical engineering from Politecnico di Milano, in 1993. He was with CEFRIEL, formerly as a master student and then as a Junior Researcher. He is now a Ph.D. student at Politecnico di Torino. His interests include software process modeling, software process improvement, and object-oriented databases. He is member of AICA.

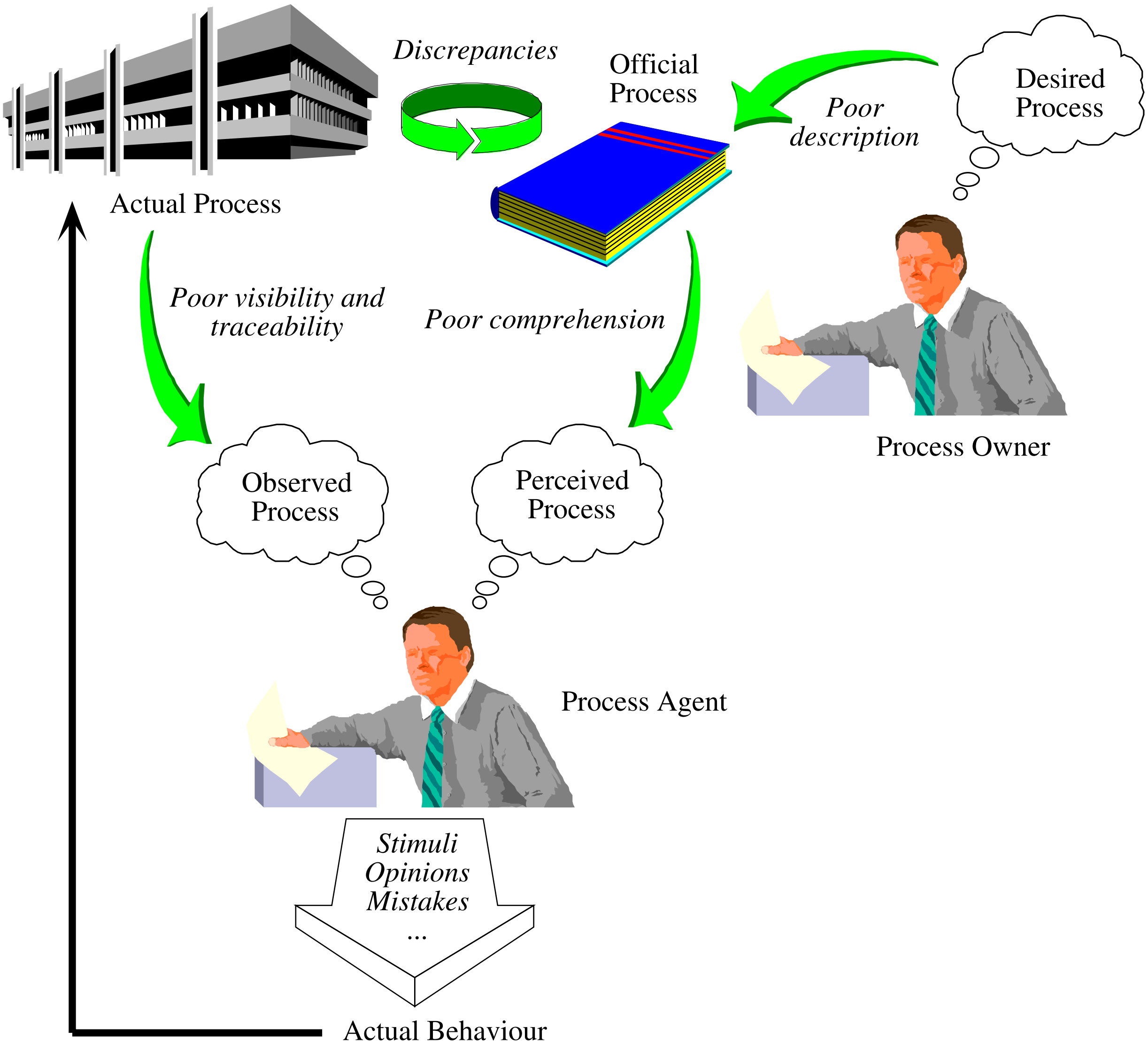


Fig. 1. The feedback loop in software developmnet organizations.

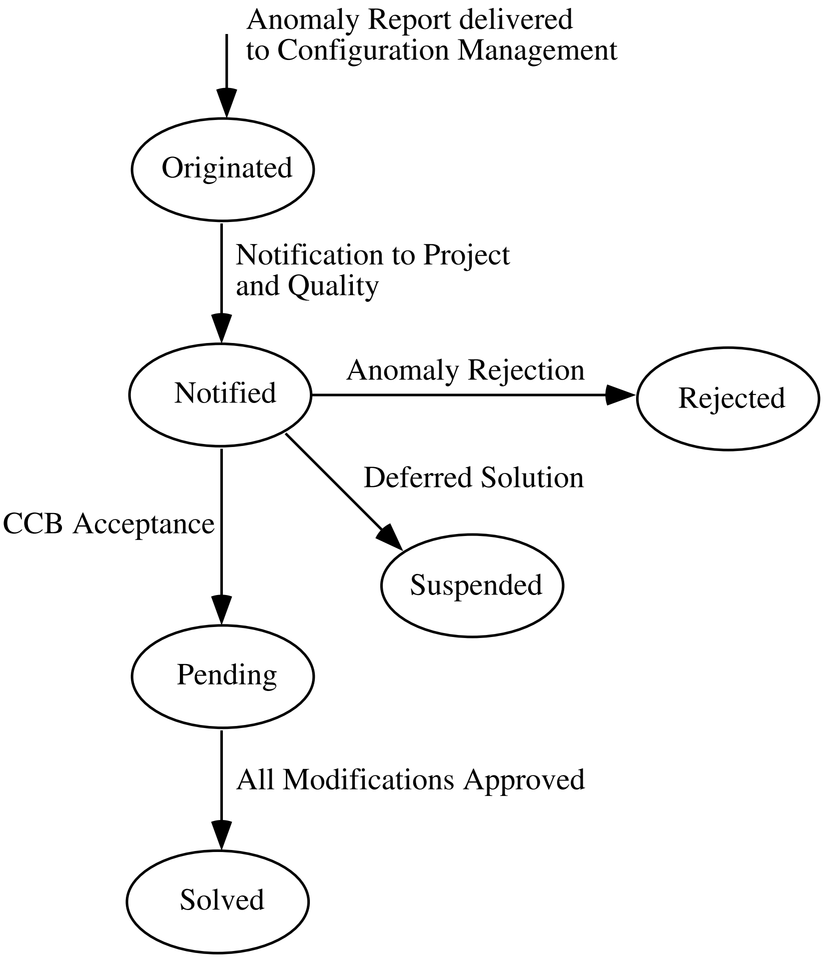


Fig. 2. FSM for Anomaly Reports (ARs).

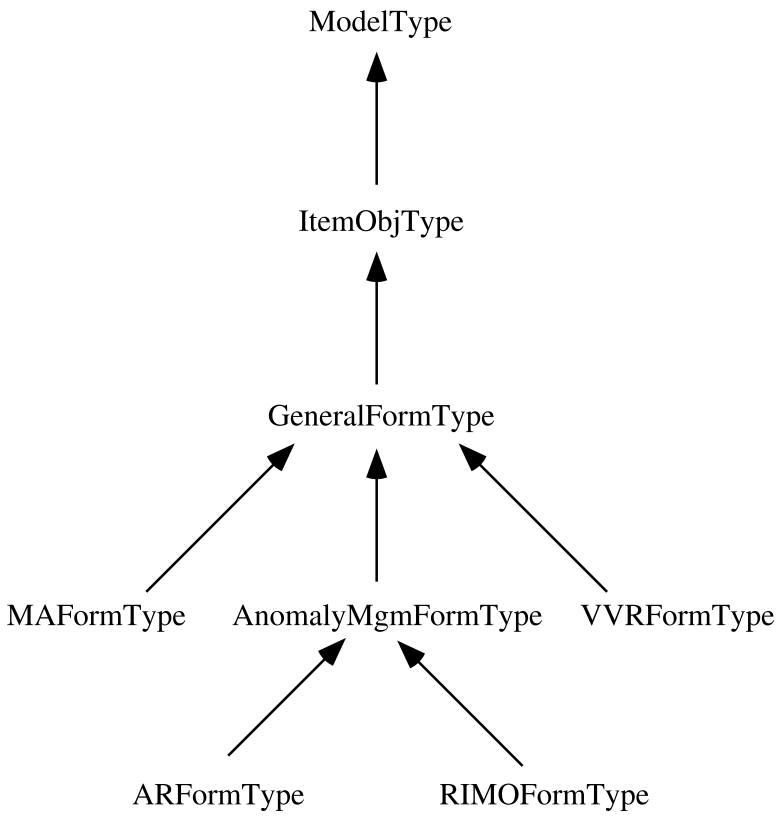


Fig. 3. The type hierarchy describing configuration items in the BUTD model.

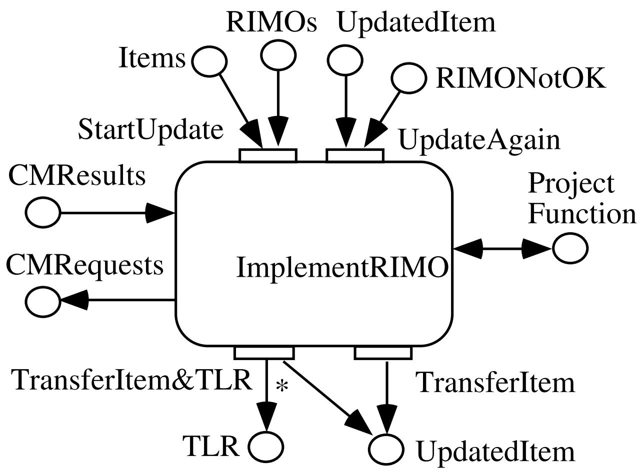


Fig. 4. The interface of activity *ImplementRIMO*.

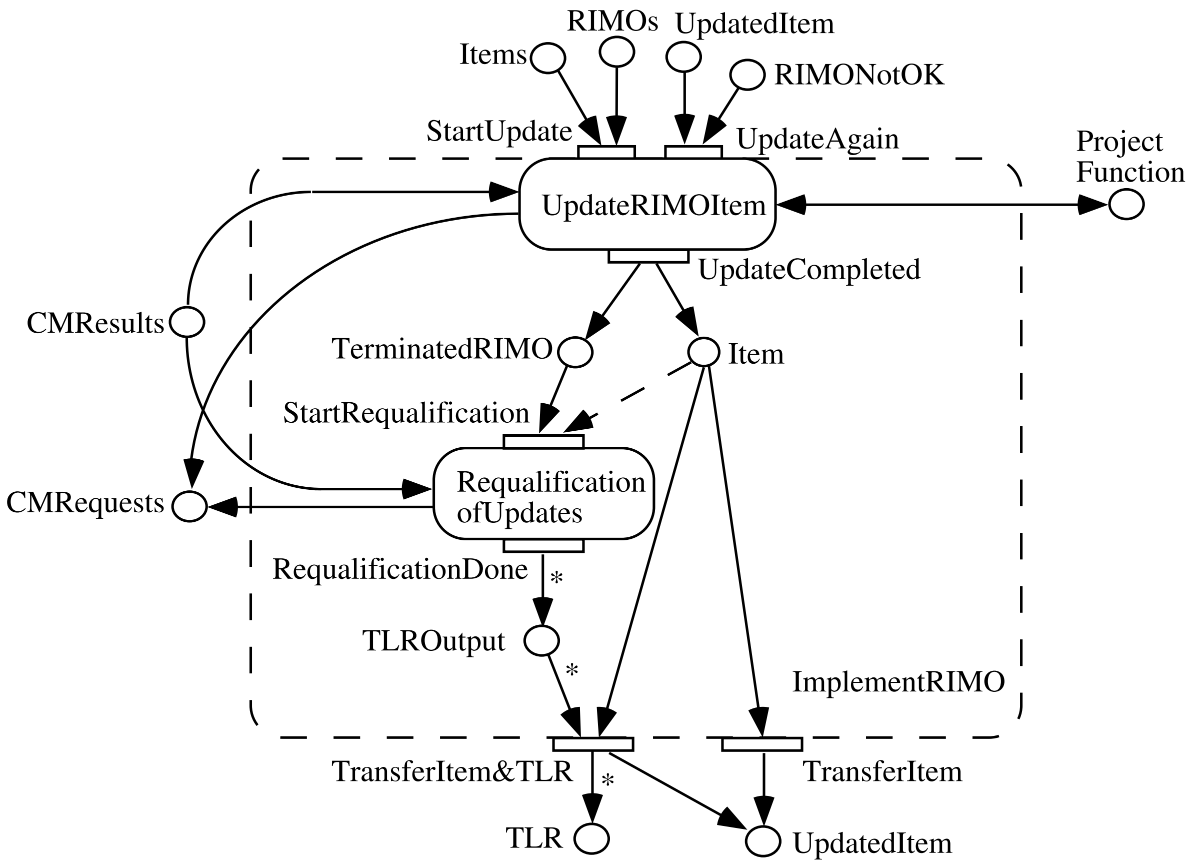


Fig. 5. The definition of activity *ImplementRIMO*.

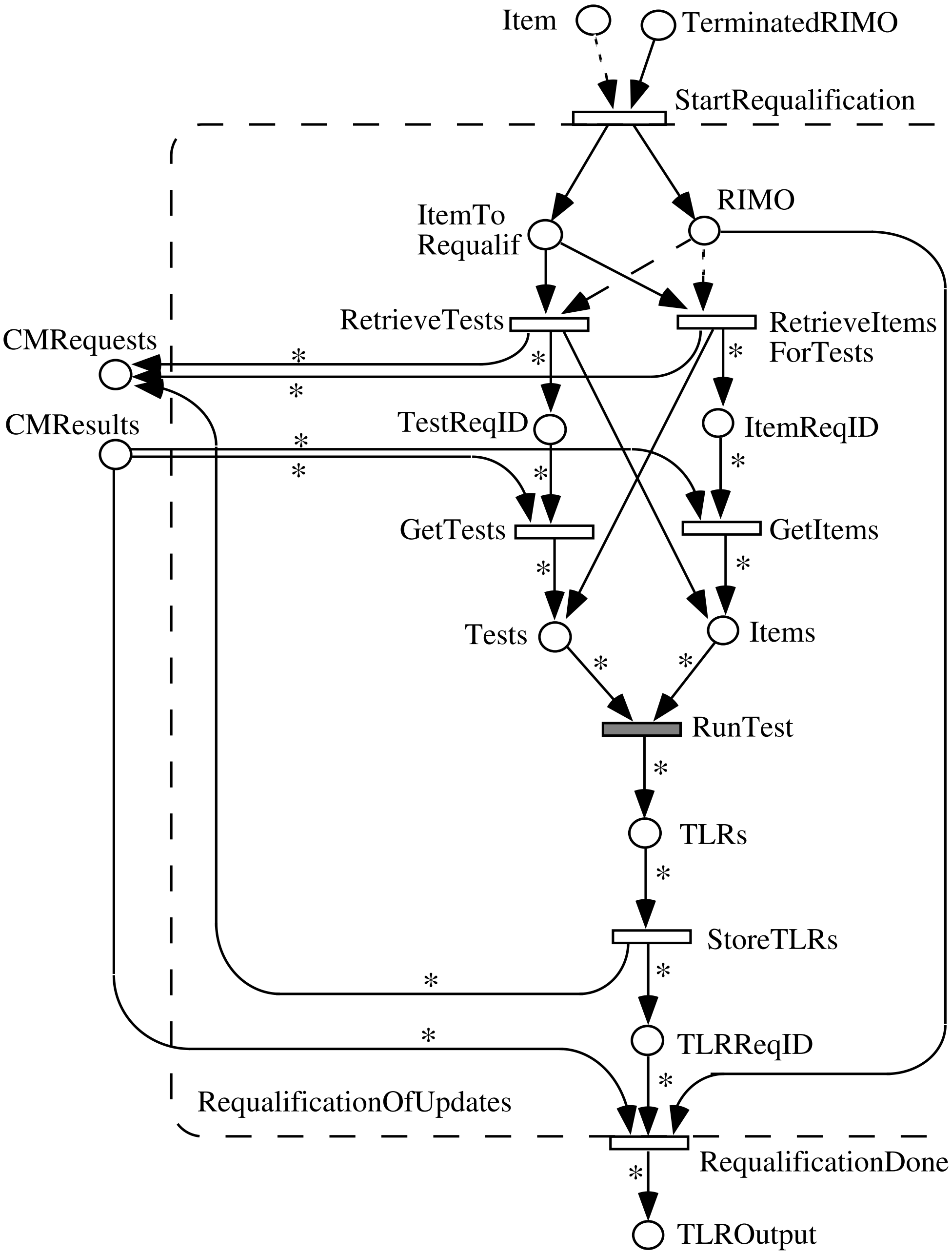


Fig. 6. The definition of activity *RequalificationOfUpdates*.

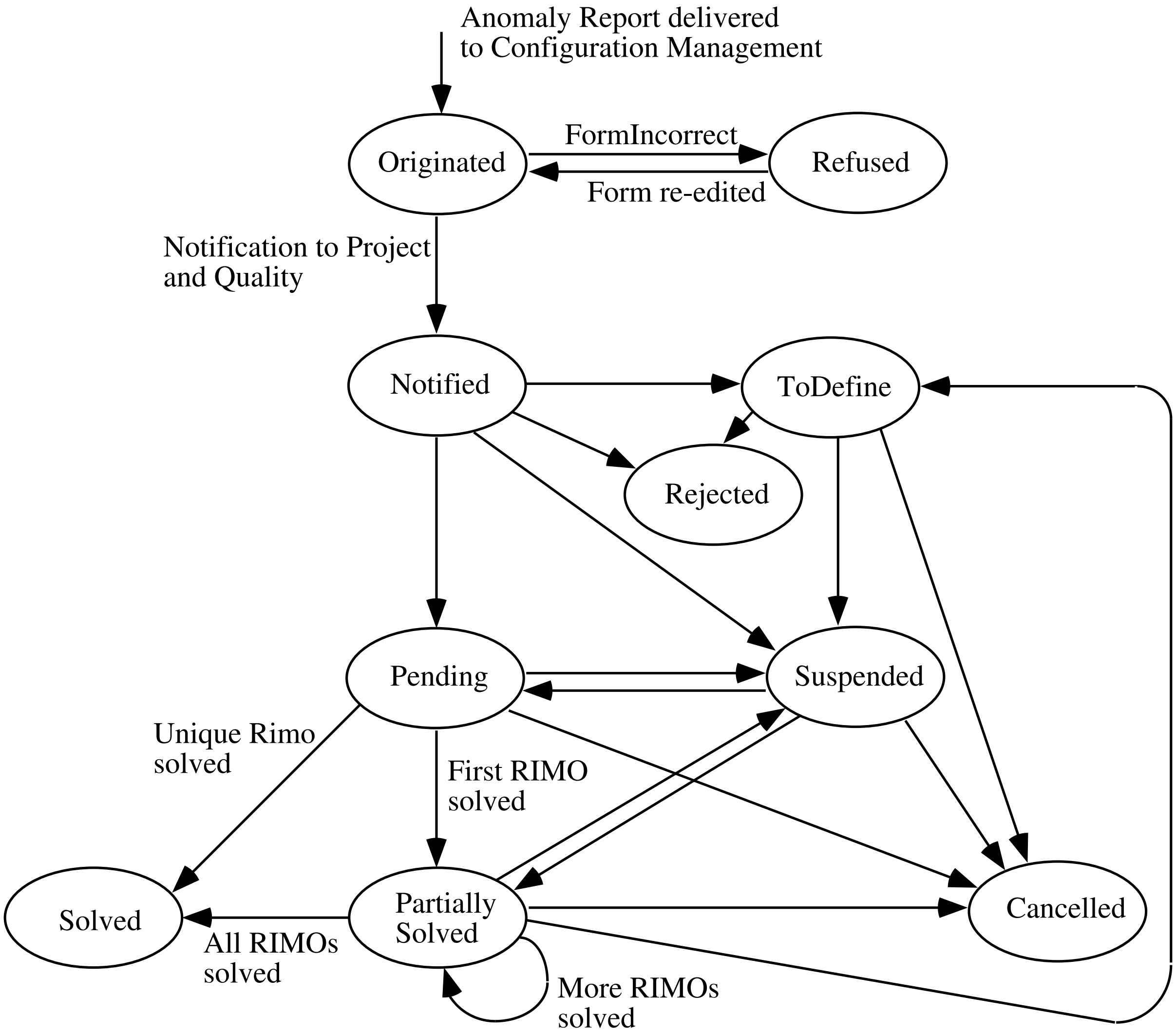


Fig. 7. The new version of the FSM for Anomaly Reports.

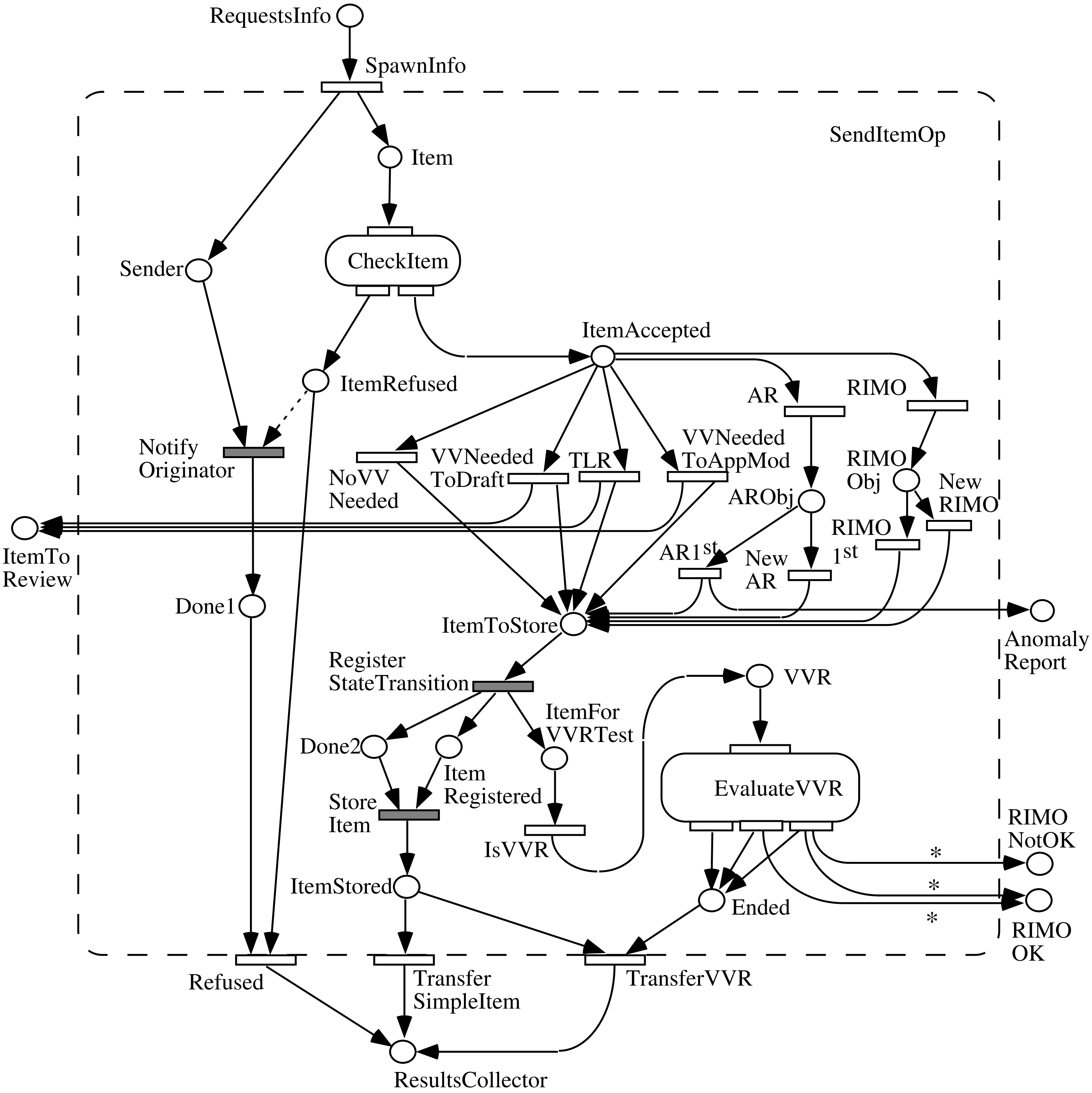


Fig. 8. The implementation of activity *SendItemOp*.

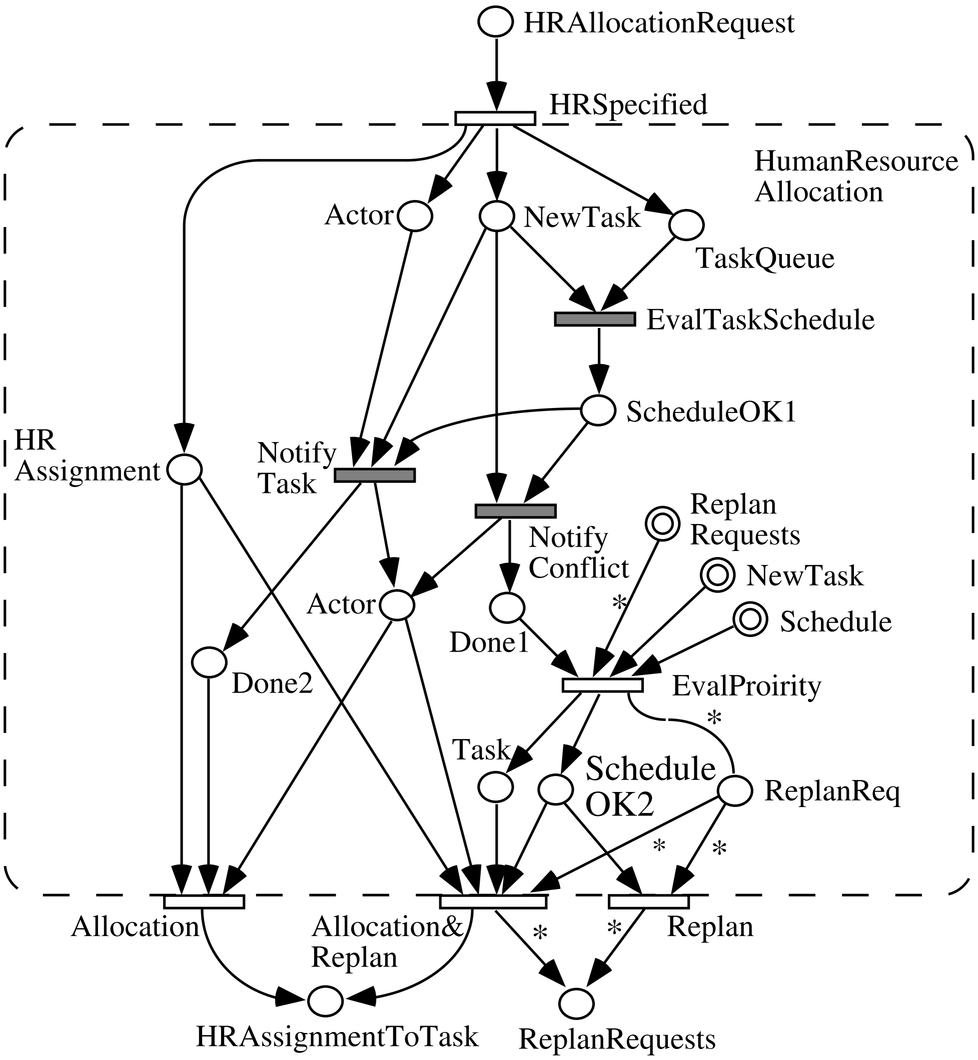


Fig. 9. The implementation of the activity *HumanResourceAllocation*.