Abstract - Requirements documents tend to be inundated by requirements that are no longer meaningful, descriptions that are unnecessarily long and convoluted, duplication of information, among other shortcomings. These syntactical problems hinder the overall understandability of requirements documents throughout the whole development process. Quality evaluation strategies are not very helpful if they only include lists of expected product qualities. They should also incorporate guidelines to help practitioners to effectively measure and improve the qualities of requirements documents. AIRDoc1 is a step forward to fill this gap proposing a process that supports the evaluation and improvement of requirements documents specified based on use cases. This process is founded on the elaboration of goals and definition of questions that will be answered by hypotheses and metrics. The quality of requirements documents is improved through refactorings and patterns. A case study demonstrates how the process of improvement has been successfully applied in an industrial requirements document.

Keywords - use case model, refactoring, GQM.

I. INTRODUCTION

The definition of requirements is critical for the success of software system development [1]. However, the current practices, fail to satisfy this, leading to inaccurate and faulty descriptions. Requirements documents may come in different styles, for example, some advocate the use of goal oriented modeling languages such as i* [2] and KAOS [3]. Those developing critical and safe applications may prefer a more formal approach. Nevertheless, the great majority of requirements documents are written either in Natural Language or in some semi-structured notation. According to [14] 79 % of requirements documents are written in plain natural language, 16 % in structured natural language and only 5 % in a more formalized language.

This work focuses on requirements documents specified with use case models. Besides being part of an international standard [15], it is also very popular in our subject domain, i.e., the Brazilian software industry. A use case should, at least, provide the main and secondary flow of actions, post conditions and business rules [4]. Additionally, we suggest the inclusion of its source of information, which may be useful if further clarification is required. The level of detail of a use case description is defined by the stakeholder. Of course, it will influence the measurement of its size.

Some attention has been dedicated to discover typical syntactical problems that compromise the quality of a requirements document, such as requirements that have been abandoned and that are no longer meaningful, descriptions that are too long and difficult to read, and information that is duplicated [6]. These shortcomings hinder the overall understandability and reusability of requirements documents throughout the development process [7]. These problems can be minimized by the early identification of symptoms and the removal of their causes. Hence, this will contribute to keep the software development within budget and time.

In this paper we claim that some of the potential problems with the requirements documents could be solved using the suitable requirements refactoring [8] or patterns [9], [10] at the appropriate time and place.

We propose AIRDoc, an approach to improve the quality of requirements documents. It relies on the well-known Goal Question Metrics (GQM) technique [11] to help the evaluation of the requirements documents. Thus, our objective is to:

1. Evaluate requirements documents by elaborating goals, defining questions and hypotheses that will be addressed by metrics. Hence, we elaborate hypothesis which are then associated with possible answers to some questions;
2. Improve the quality of the requirements documents by using requirements refactorings or patterns. AIRDoc relies
on a catalog of well-known requirements problems and their possible solutions.

The remaining of this paper is structured as follows. Section 2 presents the AIRDoc process, its activities and tasks. Section 3 describes the application of our process to improve an industrial requirements document conducted by SERPRO, a Brazilian Government software company (More information at http://www.serpro.gov.br/). This section also presents some considerations about this case study. Section 4 discusses some related work. Finally, Section 5 draws some conclusions and points directions for future work.

II. THE AIRDOC APPROACH

The AIRDoc process is based on GQM [11] and in experimentation techniques [12], and complies with the IEEE Standard for a Software Quality Methodology [13]. It is divided into two stages: (1) Evaluation that consists of four Activities: (E.1) Elaboration of Evaluation Plan; (E.2) Definition of GQM Activities, (E.3) Collection of the Metrics Values; (E.4) Interpretation of GQM Activities; and (2) Improvement that consists of two Activities: (I.1) Elaboration of Improvement Plan; and (I.2) Application of Improvement. Fig. 1 illustrates the process, showing the activities modeled with BPMN (Business Process Modeling notation) [16]. Each activity encloses a sub-process that helps the quality team to conduct the process.

The quality assurance team decides where the process will start. Hence, the first stage, the “evaluation phase”, is optional. This is the case, for example, when a potential requirements problem has already been identified in the document. However, if the existence of problems is known but their localization can not be determined, it is advisable to run the evaluation phase (which requires more careful planning and appropriate budget and time). In doing so, other problems that could have gone undetected may be identified. Fig. 1 (Evaluation Lane) shows the four activities of the Evaluation Stage together with their inputs and outputs (note that they are numbered). Due to lack of space, we only present an overview of the evaluation activities. More details can be found at the AIRDoc site [17].

(1) Evaluation - Activity E.1. Elaboration of Evaluation Plan: This activity complies with the IEEE Standard for a Software Quality Methodology [13] which recommends starting with the definition of an evaluation strategy. The inputs of this activity are: (1) the use case model to be evaluated, and (2) the quality model (we can use ISO 9126 [18], or [19], [20], among others). The outputs of this activity are: (3) schedule of tasks; (4) decisions about training; (5) selected quality team members; (6) tools and/or resources; (7) selected scope (this output indicates which part of the use case model will be under evaluation); (8) quality requirements (this defines, in agreement with the quality model selected, which quality attribute will be measured); and (9) the evaluation plan.

(1) Evaluation - Activity E.2. Definition of GQM Activities: The main objective of this activity is a rigorous definition of measurement. It should include the description of the evaluation goals, questions and hypotheses, as well as
the metrics. It should also define how a measurement will be performed and the analysis will be executed. The inputs of this activity are: (1) the use case model; (2) the quality model; (7) the selected scope; (8) the quality requirements; (9) the evaluation plan; and (10) the AIRDoc template hypotheses [17] (this template helps the quality team to elaborate the answers for the questions). The outputs are defined in agreement with the evaluation strategy and the quality model: (11) goals; (12) questions; (13) hypotheses, which were generated using the template; and (14) the set of metrics.

(1) Evaluation - Activity E.3. Collection of the Metrics

Values: When all the definition activities are completed, the actual measurement can start. The success of every project evaluation depends on accurate measures. Sometimes the measurements can be obtained without human intervention. But in the case of process and resource measurements that is usually not possible [21]. All the results of the data collection are filled in forms stored in a measurement database. The inputs of this activity are: (1) the use case model; (9) the evaluation plan; and (14) the set of metrics selected in the past activity. The outputs are (15) the metrics values collected.

(1) Evaluation - Activity E.4. Interpretation of GQM

Activities: Interpretation is an essential phase of the AIRDoc process. During this phase, the metrics values collected are used to interpret the hypotheses and answer the stated questions with the purpose of identifying whether the evaluation goals have been achieved as well as to indicate potential problems in the use case model. The inputs of this activity are: (1) a use case model, (9) an evaluation plan, (11) evaluation goals, (12) questions, (13) hypotheses, and (15) metrics values collected. The outputs are: (16) the meeting report, that contains all questions answered by the hypotheses and the evaluation goals achieved, and (17) the indication of potential problems with their localization in the use case model. This last output is optional, if potential problems have not been detected, or if some fault occurred in the evaluation process.

The second stage of our approach, “Improvement Lane” in Fig. 1, is composed of two activities and is related to solving the potential problem(s) previously detected, either by the evaluation stage described above or by the judgment of the requirements engineer. Each of these activities is described in more detail in Fig. 2 and Fig. 4, respectively.

(2) Improvement - Activity I.1. Elaboration of Improvement Plan: All problems, or symptoms, detected are discussed and plans for improvement are proposed. As mentioned before, if appropriate we can skip the evaluation stage and start right from this activity (see Fig. 2).

(2) Improvement - Activity I.1 - Task I.1.1: Problem Analysis: To be able to select the appropriate improvement, it is necessary to analyze the problem(s) detected. The inputs of this task are: (1) the use case model; (17) the indications of potential problems, this document can be generated as output of the evaluation stage or generated in an ad-hoc fashion, as previously mentioned; and (19) the catalog of problems and possible solutions – this catalog contains a brief description of the problems and their possible solutions using refactorings or patterns. For example, Table I shows a description of the Large Use Case problem and possible solutions. The current catalog of problems and solutions includes the following entries: Duplicated Steps, Complex Conditional Structures, Lazy Use Case, Use Case Naming Problem, Tangled Requirements and Scattered Requirements. More details in [17]. The output of this activity is (19.1) the “conclusions about the problems”, with its characterization and localization in the use case model.

<table>
<thead>
<tr>
<th>Problem Name: Large Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Use Case occurs when (i) a use case is trying to handle several different requirements at the same time or (ii) there are many alternative flows and steps [22], [23]. This problem is particularly significant when the maximum size of each use case has already been set by the organization’s Software Quality Assurance Team.</td>
</tr>
</tbody>
</table>

### Possible Solutions

Use the Extract Use Case refactoring to extract information related to a given requirement and insert it into a new use case. This operation could be repeated for each major requirement addressed by this large use case. If the use case steps could be moved to another use case, they could be used the Move Use Case Steps refactoring [24].

After extracting or relocating use cases, it may be necessary to rename them to better express the intention of the newly created one or of the one that was modified. In this case, the Rename Use Case refactoring [24] could be used to provide more appropriate names. Another possible solution is to use the Extract Early Use Case Aspect refactoring [25]. This solution employs the aspect-oriented requirements engineering and may be a good way if the requirements engineer desires to work with Aspect-Oriented Requirements Engineering [26].
(2) Improvement - Activity I.1 - Task I.1.2: Definition of the Solution: The outputs of this task are: (20) the improvement plan, this document contains the definition of the best solution to be used and the plan of how it should be applied; and (21) the refactoring or pattern selected. To generate its output, their inputs are: (19.1) the Conclusions about the Problems; and (18) the catalog of refactorings and patterns. To achieve the output, the quality assurance team needs to consider each refactoring that solves the problem detected. For example, the Large Requirement problem (Table I) may be solved in three ways (Extract Use Case, Move Use Case Steps or Extract Early Use Case Aspect).


Each refactoring contains the name, the context that suggests the use of the refactoring, the type of solution provided, the reason for the transformations, its mechanics (i.e., a set of well defined activities) and an example of the refactored description. Following is presented the Extract Use Case refactoring.

**Name:** Extract Use Case Refactoring

**Context:** A set of inter-related information is used in several places or could be better modularized in a separate use case. Moreover, a use case may be too large or contain information related to a feature that is scattered across several use cases or is tangled with other concerns.

**Solution:** Extract the information (a requirement structured in use case steps.) to a new use case and name it according to the context.

**Motivation:** This refactoring should be applied when there are large use cases that can be split into two or more new requirements. These large use cases include a great deal of information that is difficult to understand. Furthermore, it may be difficult to quickly locate the needed information [23], [28].

**Mechanisms:** Fig. 3 illustrates the mechanisms to extract information from a use case to a new use case. These mechanisms are detailed in algorithm format, as follows.

1. Create a new use case and name it
   //Verify if exists a use case that
   //encapsulates the same segment of information
   //that will be encapsulated in the new u.c.

   IF exist = TRUE THEN
   //Verify if exists the possibility for
   //the existing use case to encapsulate the
   //new information:
   IF possibility = TRUE THEN
   The new use case is not created;
   END_IF;
   ELSE
   The new use case is created;
   END_IF;
   END_IF;

2. Select the information you want to extract.
   //Verify in each step IF exists the
   //information you want to extract
   IF information = TRUE THEN  //TRUE means that
   //the information exists
   // verify if the selected information is
   // tangled with another information.
   IF tangled = TRUE THEN
   Re-write the step splitting the
   information.
   END_IF
   Select the segments that correspond to the
   desired information;
   Go to the next step;
   ELSE
   Go to the next step;
   END_IF
   END_IF.

3. Add the selected information to the new use case.
   //In the original use case verifies if exists
   // a priority order:
   IF priority = TRUE THEN
   Order the selected information;
   //Verify if the selected information already
   //exists in the new use case
   IF exist = TRUE THEN
   Go to the next selected information;
   ELSE
   Add (copy) the selected information to
   the new use case;
   Go to the next selected information;
   END_IF;
   ELSE
   //Verify if the selected information already
   //exists in the new //use case
   IF exist = TRUE THEN
   Go to the next selected information;
   ELSE
   Add (copy) the selected information to
   the new use case;
   Go to the next selected information;
   END_IF;
   END_IF;

4. Remove the information from the original requirement.
   WHILE not at the end of the original use case
   steps DO
   //Verify in each step IF exists the
   //information you want to extract
   IF information = TRUE THEN  //TRUE means that
   //the information exists
   // verify if the selected information is
   // tangled with another information.
   IF tangled = TRUE THEN
   Re-write the step splitting the
   information.
   END_IF
   Select the segments that correspond to the
   desired information;
   Go to the next step;
   ELSE
   Go to the next step;
   END_IF
   END_WHILE.

5. Make sure the original use case is acceptable without the
   removed information.

Figure 3. Illustration of the steps to extract use case refactoring.
6. Define the best relationship between the original and the new use case that preserves the semantic functionality

/ * Verify the type of information that was removed and choose the more convenient relationship.* /

7. Update the references in dependent use cases.

Having concluded the first activity of the improvement phase we can proceed to the last one.

(2) Improvement - Activity I.2. Application of Improvement: After the selection of appropriate means to solve the problems, all the refactorings and/or patterns selected in the previous phase are applied in the requirements document. Fig. 4 shows how the improvements are applied, defining their outputs and inputs.

(2) Improvement - Activity I.2 - Task I.2.1: Application of the Selected Solutions. This task is performed (manually) by the software quality assurance team. Its inputs are: (1) the use case model; (20) the improvement plan; and (21) the refactoring or pattern selected. Following the improvement plan with the solution selected, the quality team generates (22) the use case model improved.

(2) Improvement - Activity I.2 - Task I.2.2: Evaluation of the New Document. The output of this task is: (22.1) the analysis about improvement values. This output contains a comparison of the original use case model with the new one generated by the AIRDoc process. The inputs are: (14) the metrics, that need be collected from the new use case model; (15) the metrics values obtained before the improvement stage; and (22) the use case model improved. The output needs to be stored to create a base line of the bad and good solutions to some context. This task is optional, because the process may have been started on the Improvement stage and the metrics from the evaluation stage may not have been collected.

III. APPL YING AIRDOC IN A INDUSTRIAL AND LARG E REQUIREMENTS DOCUMENT

The Brazilian Federal Revenue Service (FRS) (more information at http://www.receita.fazenda.gov.br) is a department of the Ministry of Finance (More information at http://www.fazenda.gov.br/) and is responsible for collecting, administrating and auditing a plethora of federal taxes. To process the huge amount of data that originates from its auditing activities (billions of registers of all sorts), the FRS holds a partnership with SERPRO (More information at http://www.serpro.gov.br), a Ministry of Finance subsidiary software development company, for the development of automated solutions to support tax analysis. SERPRO is a large company with development units broadly spread throughout 10 capital Brazilian cities. The company employs more then 2500 software engineers and has a history of successful and awarded solutions built over 40 years as a partner of the Brazilian RF. These solutions offer fully-automated support for a multitude of aspects comprising auditing actions at individual business segments of the Federal Revenue Service. The growing numbers of data and surpassing complexity of Brazil’s tax system, however, have fostered the need for an integrated vision of fiscal actions at the highest administration levels of the Brazilian FRS.

Our case study is based on large real requirements document provided by SERPRO. In this paper, the chosen artifact is the “Adjustment Tax” requirements document. Basically, it describes the correction of the amount of taxes paid by citizens. The requirements document includes a use case diagram, shown partially in Fig. 5, contains 50 use cases and 1553 steps counting all use cases descriptions. Due to confidential restrictions, the descriptions of the use cases are not shown. The requirements document provided by SERPRO contained problems detected by their own requirements engineers. According to them, there are two large use cases that are too complex, thereby compromising their understandability and maintenance. The use cases ("Display spreadsheet control" and "Display screen of user analysis") highlighted in Fig. 5 deal with the Display requirement that contains 798 steps, more than 51 % of the overall steps. These use cases specify functionality related to the display of information on screens. This information may be the result of a search or a simple menu where the user can access another system’s option.

The paper’s authors and the manager of the Recife2 Unit of SERPRO were the participants of case study. Our intent in this case study is to solve the Large Use Case problem. Therefore, we will improve the understandability and consequently the maintainability [18] of the use cases model, with particular focus on the “Display Requirement”. Since the requirements team was already aware of this problem, we can skip the first stage of AIRDoc (related to the detection of potential problems) and go straight to the second one which deals with the improvements. Subsequently, we provide a

2 “Information that appears on the screen of a computer Terminal”. Definition found at http://www.library.yale.edu/~license/definiti.shtml

3 Recife is the capital of the state of Pernambuco in the North East of Brazil
short description of how our approach was used to ameliorate the Adjustment Tax requirements document.

The time spent to find the large use cases problem could not possible be measured, because it be a problem detected by the own SERPRO requirements engineer. However, all time spend to perform the evaluation was 11 hours and 50 minutes. Each activity below shows the time spent for its realization. This time includes the time spent by all participants to perform the activity.

A. Improvement - Activity I.1 - Elaboration of Improvement Plan

i) Task I.1.1: Problem Analysis

Usually, when a potential problem is identified, the quality team should consult the catalog of problems to characterize it. The problem detected by the SERPRO requirements engineers was related to the “large number of alternative flows and steps”. Thereby, we identified the problem as being of type Large Use Case (see Table I). Time spend to perform this activity: **1 h**.

ii) Task I.1.2: Definition of the Solution

There are 3 possible refactorings to solve the Large Use Case problem, namely Extract Early Use Case Aspect, Move Use Case Steps or Extract Use Case (shown in Section II). Now, we need to choose the most appropriate solution for the given context. As the AORE (Aspect Oriented Requirement Engineering) [26] paradigm is not currently adopted at SERPRO, the Extract Early Use Case Aspect refactoring option is not recommended, since this is an aspect-oriented solution [25]. Similarly, the Move Use Case Steps refactoring does not help much as it will just create another large use case.

We (manually) analyzed the description of both use cases and noticed that the use case steps were grouped by sub-requirements. Thus, we decided to apply the Extract Use Case refactoring in each sub-requirement creating a new use cases for each one. Despite leading to an increase in the overall number of use cases, this option can solve the Large Use Case problem as well as enhance the sub-requirements modularity. Consequently, we could infer that this solution would improve the understandability and maintainability of Display Requirement.

Table II shows all the identified sub-requirements in both use cases. It lists the original use case considered, the short description of sub-requirements found in both use cases, and the type of relationship that may be used by the Extract Use Case refactoring. This is part of improvement plan that is the output from the first improvement activity proposed by AIRDoc. Time spend to perform this activity: **6:30 h**.

<table>
<thead>
<tr>
<th>Original Use Case</th>
<th>Short description of Sub-requirements</th>
<th>Type of Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display spreadsheet control</td>
<td>Screen with the result of search about analyzed compensation document without verification period</td>
<td>«extend» with original use case</td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about analyzed deleted compensation document</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Menu/Screen to delete compensation document</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about analyzed historical of compensation document</td>
<td></td>
</tr>
<tr>
<td>Display screen of user analysis</td>
<td>Menu/Screen to Finish document</td>
<td>«extend» with original use case</td>
</tr>
<tr>
<td></td>
<td>Menu/Screen to Analyze share</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about detailed share “Payment”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about detailed share “payment out of the country”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about detailed share “estimate shared”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen with the result of search about detailed share “payment in PNF”</td>
<td>«extend» with sub requirement</td>
</tr>
</tbody>
</table>
B. Improvement - Activity I.2. Application of improvement

Figure 6 shows, the new use case diagram (only the part that was improved), after the use of the Extract Use Case refactoring in each one sub-requirement. The Display requirements were divided into other use cases, and <<extend>> links were used. According to Jacobson [29] these <<extend>> links cause less coupling than <<include>> links. The new use cases used to describe the display requirements are now smaller and more homogeneous. Time spend to perform this activity: 4:20 h.

C. Considerations about the Case Study

After the improvement we presented the new use cases diagram to the SERPRO requirements engineers and discussed the proposed solution. Our joint conclusions are the following:

- The use case diagram shown in Fig. 6 and its descriptions are more understandable than the original use case diagram shown in Fig. 5, because it is much simpler due to reduction of the size the use cases descriptions.
- The Display Requirement has, due to their sources, many sub-requirements and is “unworkable” to describe it in few use cases.
- The problem of the Large Use Case was raised in the planning of requirements document. Because there are much iteration in the software development process adopted by SERPRO, it became obvious that it would tend to aggravate. In that case we were able detect early the problem, i.e. during the requirements development process.
- The improvement stage of the AIRDoc process was easy to use, due to the available catalogs [17] that helped to characterize the problems and the algorithmic description of the refactorings. However, it was noted that the quality team already had the advanced skills in the structuring of the use cases templates for modelers to add information to their use cases.
- The improvement proposed in this Case Study was analyzed only by the unit of Recife. For the AIRDoc to be part of SERPRO process the following step is to pass through the various bureaucratic phases, until AIRDoc is approved by other units that are part of the development of the “Adjustment Tax”.

IV. Related Work

Some strategies have been proposed to improve the overall quality of use cases. Lilly [6] and Overgaard and Palmkvist [9] produced lists of use cases mistakes, many of which can be detected by systematic inspection of the use case diagrams. Guidelines for authoring use cases descriptions were presented by Cockburn in [4].

El-Attar and Miller [10] proposed anti-patterns to detect potential defective areas in use cases models. Anti-patterns may include some suggestions for improvement of running examples. Berenbach [28] presents a tool that provides a template for modelers to add information to their use cases. The tool scans these use cases descriptions and identifies risky situations such as incomplete or weak phrases. Current evaluation strategies often list desired product qualities but fail to provide guidelines to assist the practitioners to effectively measure, detect and improve those qualities. The AIRDoc fills this gap proposing a process that supports the evaluation (by support of metrics and a catalog of problems description) and improvement of requirements documents specified via use cases.

Yu et al. [8] explain how refactoring can be applied to improve the organization of use case models. They focus on the decomposition of a use case and the reorganization of relationships between use cases. They also describe ten refactorings that could be used to improve the overall organization of use case models, such as inclusion or extension introduction mechanisms, use case deletion or refactorings that manipulate the inheritance tree. While Yu et
al. focus on refactoring the use case models, we focus on refactoring requirements descriptions. Thus, our refactorings are finer grained than theirs. We also discuss in detail the mechanics of each refactoring and possible refactoring opportunities in the context of use case descriptions. Moreover, the strategy of Yu et al does not provide mechanisms to locate where the refactoring should be applied.

The AIRDoc helps the requirements engineer to locate the right point for its application. Moreover, their refactorings are solutions to specific problems, while ours may be used in different types of problems. For example, the Extract Use Case refactoring may be used to solve Large Use Case and Tangled Requirements [17] problems, among others.

In [30] a preliminary version of the AIRDoc was presented. Table III shows the 3 main differences between the old version of the AIRDoc and the version presented in this paper. Other improvements come from the fact that the AIRDoc approach is now defined as a process with well defined activities and tasks.

<table>
<thead>
<tr>
<th>TABLE III. SUB-REQUIREMENTS IDENTIFIED IN THE DISPLAY REQUIREMENTS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIRDoc old version [30]</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>The guidelines are general, not specific to be applied in use case diagrams.</td>
</tr>
<tr>
<td>It has no defined process with the inputs and outputs.</td>
</tr>
<tr>
<td>Refactorings are described as a sequence of steps, without details.</td>
</tr>
</tbody>
</table>

V. CONCLUSIONS AND FUTURE WORK

The low quality of requirements documents cause defects that may be propagated to later phases, where the cost of fixing such defects escalates significantly. In this paper, we presented the AIRDoc process to evaluate and improve requirements documents specified by use cases. We have outlined how to use the improvement stage in a large and real requirements document. The case study based on a real and complex requirements model provides some indications that AIRDoc may be applied to an industrial scale requirements documents. Of course, more empirical evaluation is required to validate the process, and the next step is to do systematic qualitative evaluation of the AIRDoc process. Several on-going case studies are also under way.

The AIRDoc is not appropriate to find semantics problems, that happens due the evaluation phase is based on use of metrics support by GQM [11]. However, metrics provide a more reliable support for improvement decisions [11], [21]. Some current limitations should be addressed for possible improvements in future work. Firstly, the evaluation stage may take a considerable amount of time of the quality team, which may compromise the adoption of AIRDoc. The new version of the AIRDoc process is modeled with BPMN, and the refactorings are described using the algorithm format. This helps to define it more precisely, which may enable the development of some tool support, which in turn can help accelerate this stage of the process.

ACKNOWLEDGMENT

The authors thank Helena C. Bastos (Manager of the Recife Unit of SERPRO) for her support. This work was funded by several research grants (CNPq Proc. 141987/2008-1, CAPES Proc. BEX 0894/08-7, CAPES/CGCI/MECD Proc. 167/08).

REFERENCES


