AIRDoc – An Approach to Improve Requirements Documents

Ricardo Ramos[1], Jaelson Castro[1], João Araújo[2], Ana Moreira[2], Fernanda Alencar[1], Emanuel Santos[1], and Rosangela Penteado[3]

[1] Centro de Informática - Universidade Federal de Pernambuco (UFPE)
Recife, Pernambuco, Brasil

[2] Departamento de Informática, FCT, Universidade Nova de Lisboa (UNL)
Lisboa, Portugal

[3] Departamento de Computação - Universidade Federal de São Carlos (UFSCar)
São Carlos, São Paulo, Brasil

{rar2, jbc, ebs}@cin.ufpe.br, {ja, amm}@di.fct.unl.pt,
fmra@ufpe.br, rosangel@dc.ufscar.br

Abstract. *Requirements models and specifications tend to be plagued by many problems such as requirements that have been abandoned and that are no longer meaningful, descriptions that are unnecessarily long and convoluted, and information that is duplicated. These problems hinder the overall understandability and reusability of software models throughout the whole development process. In this paper we propose an approach called AIRDoc that supports the identification of potential problems that may be present in requirements models. AIRDoc is based on the elaboration of goals and the definition of questions and hypotheses that will be addressed by requirements metrics. In order to improve the quality of requirements models we advocate the use of refactorings and requirements patterns. A case study demonstrates how the AIRDoc has been successfully applied in large requirements model conducted by SERPRO, a Brazilian Government software company.*

1. Introduction

Recently, some attention has been dedicated to discover typical problems that seem to plague requirements models and specifications, such as requirements that have been abandoned and that are no longer meaningful, descriptions that are unnecessarily long and convoluted, and information that is duplicated [1, 2]. These shortcomings hinder the overall understandability and reusability of requirements models throughout the development process [3]. However, these problems can be minimized by the identification of symptoms and the removal of their causes. Moreover, if this diagnosis is performed in the early stages of the software development process some reduction in the costs associated with software evolution may be obtained [4].

Unfortunately, the early identification of the symptoms during the initial development stages is unusual. Some catalogs of problems that occur in requirements are described in [2] and works about inspection and reading techniques in requirements documents are proposed in [5] and [6]. Regrettably, these approaches do not provide well defined guidelines on how to identify the potential problems in requirements documents and models. We claim that metrics could be used to help identify some of the

problems that occur in these artifacts. But their adoption is a difficult endeavor. Collecting, interpreting and analyzing metrics has proved to be a major challenge [3]. Moreover there is a high cost of their adoption.

In this paper we claim that some of the potential problems with the requirements models could be removed using appropriate requirements refactoring [7, 10] or requirements patterns. In general, refactorings and patterns can be classified according to the quality attributes they affect [11]. Note that for some requirements models, such as use cases diagrams [9], it is already possible to specify their external quality attributes (such as correctness, reusability, efficiency, ease of use, among others). Hence, we advocate that the requirements engineer can improve the quality of requirements models by applying the relevant requirements refactorings or patterns at the right places.

We adopt the Goal Question Metrics (GQM) [12] approach to evaluate requirements models. Thus, we provide an approach called AIRDoc (Approach to Improve Requirements Documents) which provides improved support to the software quality assurance team to elaborate goals and define questions and hypotheses that will be measured by metrics. Our approach contributes to:

- The diagnoses of requirements problems (a catalog of well known troubles is provided);
- Provide solutions to requirements problems (by means of requirements refactorings and patterns);
- The adoption of requirements metrics;

This paper is structured as follows. Section 2 presents our approach and its steps. Section 3 describes the application of our approach to the improvement of a large requirements model conducted by SERPRO, a Brazilian Government software company; Section 4 discusses related works. Finally, Section 5 draws some conclusions and points out directions for future work.

2. The AIRDoc Approach

AIRDoc is based on GQM [12] and in experimentation techniques [13] and complies with the IEEE Standard for a Software Quality Methodology [14]. It consists of six steps: (1) Plan elaboration, (2) GQM definition, (3) GQM interpretation, (4) data collection, (5) plan of requirements model improvement, and (6) requirements model improvement. Figure 1 illustrates these steps and their relationships.

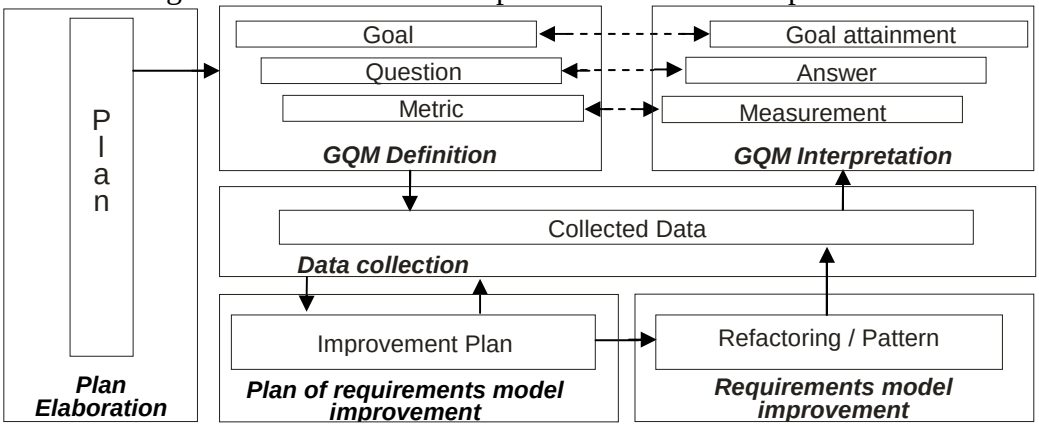

Figure 1. AIRDoc's main steps

AIRDoc's main objective is to detect potential problems in requirements models and specifications as well as to solve those using refactorings or requirements patterns. Figure 2 depicts an activity diagram with AIRDoc's main steps.

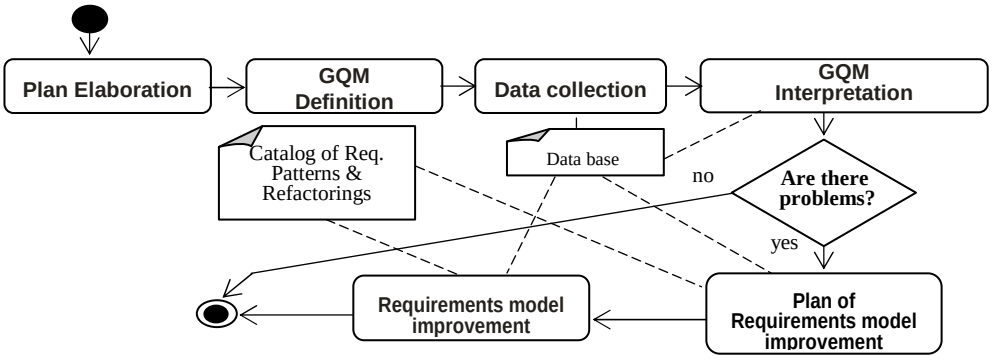

Figure 2. Activities diagram of the AIRDoc

Step - 1. Plan Elaboration

This step is performed to fulfill all the necessary requirements to make the evaluation a success, including training of the quality team, management involvement and project planning. This step can be divided into four sub-steps:

1.1. Establish Software Quality Assurance Team. The quality team should be knowledgeable on the subject, objective and since they are in charge of planning the whole project (including definitions), they also take care of data collection and prepare everything that is necessary for data interpretation. Also, the team should consider factors, such as time and costs. In this sub-step, each quality team member is assigned at least a role. Bellow, we show a summary of the actions required for this step. Table 1 shows a template that can be used in this step for documentation purposes.

Actions required to establish a software quality assurance team

1. Define the project team roles.
2. Define the responsibilities of each role.
3. Assign roles to members of team.

Table 1. Template to define the quality team

Project Team Role	Quality Control and Quality Assurance Responsibilities	Assigned Resource / Members
<Project team role>	*<Specific project quality-related responsibilities>*	*<Resource fulfilling the role>*

1.2. Select Tools or Other Resources. The quality team member responsible for reviewing and selecting tools should indicate which tools can be used in the quality evaluation. The team may require some training to get acquainted with the chosen tools. As in the first step, the choice of the tool is based on the time available to learn to work with the tool as well as with estimated costs of added to overall quality evaluation process. Table 2 shows a template that could be used in this step for documentation purposes.

Actions required to select tools and other resources

1. Select the tools and others resources.
2. Justify the tool and resources (purpose).
3. Detail their costs, including the time needed to learn it.

Table 2. Template for tools and/or others resources

Tool / Resource Name	Tool/Resource Purpose/Use	Costs Details (time and/or monetary)
<Tools/ resources needed>	*<Purpose or use of each tool/resource>*	*< Cost details of the of each tool/resource>*

1.3. Establish Software Quality Requirements. This sub-step selects the scope of the requirements model, the quality attributes and the requirement that will be evaluated and improved. These selections are performed with respect to business goals, such as cost, time, risk and quality. After selecting a suitable area, the team should consider all the details, such problems that might occur, all external influences, stakeholders, processes and products involved, and the previous knowledge on measurement of the persons who are going to participate in this project.

Actions required to establish the software quality requirements

1. Define the quality evaluation scope. The quality team selects the requirements model, or part of it, to be evaluated. In some cases, the requirements model may be obsolete in relation to the implemented system; in these situations it may be necessary first to update the model (requirements baseline).
2. Select one or more goals of a given list (eg., Table 3), or use specific goals established by the quality team.
3. Describe each goal (eg., fill Table 4). This information will be used in the project plan phase.

Table 3. Goals suggested by the AIRDoc (partially)

Goal	Motivation for Choosing the Goal
Reusability	Reduces time spent on rework. Increases the productivity of the software development.
Maintainability	Reduces time spent to correct a specific problem or to add a new requirement.
Understandability	Requirements Model with low understandability is error-prone and hard to maintain. The easier a requirements model is to understand, the simpler it is to learn it.

Table 4. Template to quality requirements

Goal	Requirements Model	Requirement in focus	Quality attribute
<Goal>	*<Requirements Model name>*	*<Requirement that will be evaluated>*	*<Quality attribute that will be evaluated >*

1.4. Project Plan - The project plan is the output of the first step of the AIRDoc. It is created by the software quality assurance team based on inputs from the project team, and links all information collected in the three previous sub-steps. A project plan should include the following items:

- Measurement program, in short, condensed version (Tables 1 and 4).
- Schedule (with complete description of tasks that should be performed), list of resources to be used with a schedule, list of results that should be obtained and, finally, expected costs and benefits.
- Management process, which contains priorities, descriptions of reporting procedures and risk control activities. Create a list of risks and add details on how to solve them? Relate the costs with each risk involved.
- Training and promotion. Use a table (such as Table 2) to show the training needed.

After the elaboration of the plan of measurement, it is necessary to define formally the goal, question and metrics, and to instantiate the metrics. These tasks are performed in the next step.

Step - 2. GQM Definition

The main task of this step is a rigorous definition of measurement, including describing questions and hypotheses, reviewing, checking and producing measurement and analysis plans. The definition step can be divided into 4 sub-steps:

2.1. Measurement Goals Definition. During this sub-step, on the basis of improvement goals, measurement goals should be formally defined as well as be properly structured. This sub-step consists in detailing Table 4.

Actions required for measurement goals definition.

1. Describe the selected part or the requirements model scope that will be evaluated.
2. Describe the requirements that will be evaluated.
3. Describe the quality attribute focused by the measurement.
4. Create a document with the information collected in this sub-step.

2.2. Questions Elaboration. The aim of this sub-step is to obtain operational definitions, i.e. a question is a goal refined to operational level. It must be emphasized that this level should not be too detailed or too abstract, but intermediate to provide an optimal interpretation. An interesting question is "what problems might occur as a result of mistake during this sub-step?". First, it is important to define questions precisely, because otherwise they will not represent measurement goals [12].

AIRDoc contains a quality model [15] (Figure 3) to assist in the preparation of the questions; it is based on quality attributes that will be evaluated by the software quality assurance team. The quality attributes of interest (Reusability and Maintainability in this example) are decomposed into quality factors (such as Understandability and Flexibility) and these in internal attributes of the requirements model (eg., Separation of Requirements, Size and Coupling). The status of internal attributes (their value) is obtained by the use of a set of metrics. It is important to note that the quality model (Figure 3) is not a hard rule to be followed by all evaluations. It serves as a model and should be adapted for each performed evaluation. Fore more details about this quality model see [15].

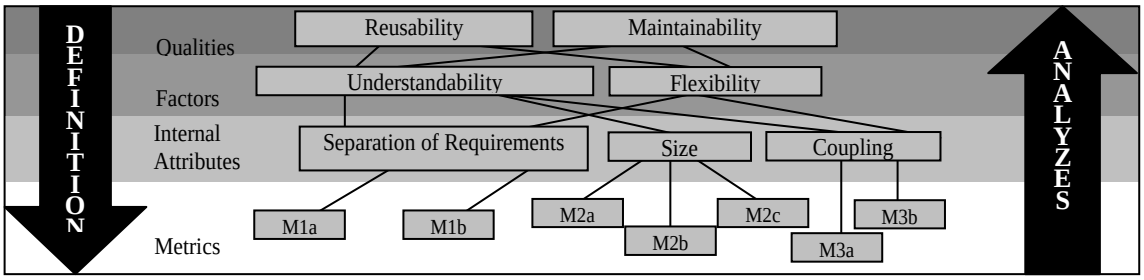

Figure 3. Reusability and Maintainability quality model [15]

Goal – Assess in a <*Requirements Model*> the <*requirement_in_focus*> to predict **Maintainability** quality.
1. How easy is it to understand the <*requirement_in_focus* >? **1.1.** How is the model structured? **1.1.1.** How many modules specify <*requirement_in_focus* >? **1.1.2.** How many elements (or parts) specify the modules of one <*requirement_in_focus* >? **…**

Figure 4. Goal and (partial) questions related to maintainability of models [15]

The questions provide insights for the Goal to be achieved. Figure 4 presents an abstract goal that can be instantiated for evaluating a requirements model and the possible questions that may be useful to evaluate the Reusability of the model (Figure 3). The instantiation of the Goal and questions are made by replacing the text that is between '<' and '>', with the definitions already decided and documented in Table 4. It is important to note that the quality model and the questions must be adapted to the domain of the evaluation that is being made. The quality team might decide to include new questions, delete irrelevant questions or to refine a question in two or more parts.

Actions required to define a set of questions

1. Define a set of questions based on the quality model and the questions provides by AIRDoc (Figure 4). Alternatively, you may define a set of questions which, when answered, provide the insights necessary to evaluate the goals.
2. Generate the Goal/Questions relationship document. This output might be in tabular format.

2.3. Metrics Definition – After defining the goal and a set of questions, we can describe the metrics to be used. The metrics aim at answering the questions that have been established. AIRDoc proposes a metrics template that can be adapted for some types of requirements models. But the metrics should be carefully chosen and should ensure that their results are consistent.

In Table 5 shows two types of metrics that the can be used to answer some questions. However, the software quality team has the final word, that it they must decide which quality metrics should be added to the set of evaluation metrics. More details about the template of metrics in [16].

Table 5. Template (partial) of metrics [16]

M1	**Separation of Requirements Metrics**		
Concern Diffusion over Components (CDC) is a requirement metric that counts the number of primary components (main decomposition structure – Artifact); the main purpose is to contribute to the specification of a specific *requirement in focus*.			
How many	**<Artifacts>**	are related to (contribute) to the specification of the	***<requirement in focus>***
M2	**Size Metrics**		
Vocabulary Size (VS). VS measure the requirements vocabulary size. This metric counts the number of main decomposition structure (artifact) that exist in the requirements model.			
Total number	of **<Artifacts>**	in the **[selected part, full requirements model**]	

Actions required for determine a set of metrics

1. Define a set of metrics, based on the template in Table 5. Alternatively, define a set of metrics which can be collected and analyzed to help to answer each question (based on internal attributes).
2. Generate a document with the relationship between Metrics/Data required and Questions (internal attributes)/Metrics. This output might be in tabular format.

2.4. Hypothesis Elaboration – the hypothesis are the expected answers, and they are going to be examined during the interpretation step. To make the hypotheses, the software quality assurance team needs to propose some values that indicate how good or how bad the value of measured value is metrics. The hypotheses can also be based on results of previous applications of the metrics.

Moreover, since we are interpreting results in terms of hypotheses, many misinterpretations are likely to occur. Hence, questions and hypotheses should be continuously reviewed and, if necessary, reformulated.

AIRDoc has a template with 3 hypotheses one for each set of metrics. Due to the restriction of space only the hypotheses for separation of requirements are showed. Table 6 shows the argument, functions, the analysis of the value obtained by the functions, as well as the possible hypotheses.

Table 6. Hypotheses for separation of requirements

H1	Hypotheses for Separation of Requirements
Argument	The more split a requirement is, the less understandable and flexible it becomes.
Function H1	This function is obtained by metrics CDC / VS. CDC is the number of modules in which the requirement in focus is separated and VS is the total number of the requirements model modules.
Analysis	The result of this function is between 1 and 0. The nearer to 1, the less understandable and flexible the requirements model is. The nearer to 0 the better understandable and flexible the requirements model is.
Hypothesis H1a	The <requirement_in_focus> is *easy* to understand and to extend, because the function H1 has its value *lower* than <base number defined by the quality team>
Hypothesis H1b	The <requirement_in_focus> is *hard* to understand and to extend, because the function H1 has its value *higher* than <base number defined by the software quality assurance team>
Note	The quality team need to calibrate the acceptable values and generate a base number to be use in hypothesis H1a and H1b.

The number of hypotheses can be raise since this number depends on some factors like: i) the metrics developed in the previous sub-step, ii) the quality attribute selected, among others.

Actions required to elaborate a set of hypotheses.

Use the hypotheses templates available in the AIRDoc to help to instantiate the hypotheses.

1. Create an argument, based on an agreement among software quality assurance team that indicates when good or bad result for a value is obtained by a metric or a function.
2. Elaborate function(s) to be used in the hypotheses. This action may be optional, because the software quality assurance team may analyze the metrics values directly, i.e. without the assistance of a function.
3. Elaborate at least one hypothesis for each question. The hypotheses are the expected answers to the questions.

Step - 3. Data Collection

When all the definition activities are completed the actual measurement can start. The success of every project 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 [12, 17].

All the results of a data collection phase are filled in forms stored in a measurement database. The whole procedure of data collection can be divided in two sub-steps:

3.1. Hold Trial Period - In order to avoid mistakes and to test and validate the data collection procedures, tools and forms, a trial measurement period should be held before the actual data collection period. See Table 14 (generated as output of sub-step 3 step 2 in the case study) needs to be understood/validated in this sub-step.

Actions required for hold trial period.

1. The software quality assurance team member responsible for applying the metrics needs to test/validate each metric described in a table (see Table 14 - metrics / data required).

2. The software quality assurance team member responsible for applying the metrics needs to check if all values obtained are consistent with the value established by the quality team.
3. Update Table (metrics / data required), if necessary. In order to better understand the metrics, a new column with notes on lessons learned in this sub-step might be added to the table (see Table 14 - metrics / data required).

3.2. Metrics Base - Data collection forms should be filled in and checked for correctness. If any mistakes occur they should be immediately corrected. A metrics baseline is the first part of Measurement Support System (MSS) which plays an important role in a measurement program.

Actions required to maintain a measurement support system.

1. Collect and store the measurement data.
2. Create a metrics baseline with the data collected.
3. Process measurement data. This action is responsible for combining, sorting and dividing data to provide the required metrics, if necessary.
4. Create a suitable form for data presentation, like tables and charts.
5. Create a baseline to be used in future evaluation.

Step - 4. GQM Interpretation

Interpretation is the essential phase of the AIRDoc approach. During this phase, the collected data are used for answering the stated questions with the purpose of identifying whether the goals are being achieved as well as to anticipate *problems* in the models. In other words, results of the measurements are discussed and conclusions are made in terms of measurement results. The following sub-steps should be performed during this step:

4.1. Feedback Session – The software quality assurance team members should prepare feedback material, such as: analysis sheets, presentation slides, handouts and, if necessary, some additional material. Feedback material should be very useful to the project team members during feedback sessions. During these sessions, project team members should analyze and interpret collected data, drawing the necessary conclusions.

Actions required to create a feedback session.

1. Prepare feedback material, such as: analysis sheets, presentation slides, handouts and, if necessary, some additional material.
2. Analyze and interpret the collected data based on proposed hypotheses.
3. All analysis performed should be stored in the baseline.

4.2. Measurement Results - After a feedback session, the quality team writes a meeting report containing all relevant observations, conclusions and action points that were raised during the session. If the analyzed data shows some possible *problems in the requirements models*, the meeting report should pinpoint (localize) these *symptoms*.

A table containing the time and the cost spent on the implementation of activities might be generated at the end of this step. This information will be useful to evaluate the overall AIRDoc approach.

Actions required to store the measurement results.

1. Write a meeting report containing all relevant observations, conclusions and action points that were formulated during the feedback session.

2. If the analyzed results were negative data, create a table to point where the worst results were found and what types of problem were identified.
3. If the analyzed results were positive, the evaluation is finished.

Step - 5. Plan of Requirements Model Improvement

In this step all problems/symptoms detected are discussed and some plans for improvement are proposed. The following sub-step should be performed during this step:

5.1. Patterns and Refactorings Analysis – In order to be able to select the appropriate requirements Refactorings or Patterns it is necessary to analyze the problems identified and discover what type of problems exists as well as to chose the patterns or refactorings that might be used. The catalog of refactorings and patterns, showed partially in Table 7, has a column that establishes the relationship between the problems and the quality attribute that was addressed. Each problem listed in this catalog has a description that helps to typify it. In Table 8, the *Large Requirements problem* is described.

Table 7. (Partial) Catalog of problems and possible solutions

Problems/Symptoms	Refactoring
Large Requirements	Extract Requirement
	Move Requirement
	Extract Early Requirement

Table 8. Description of the large requirement problem [20]

Problem Name: **Large Requirements**
Large requirements occur when (i) a requirement is trying to handle several concerns at the same time or (ii) there are many alternative flows and steps [19, 23].
Solutions Possible: **with Refactoring**
Use the Extract Requirement refactoring [20] to extract information related to a given concern and insert it into a new requirement. This operation could be repeated for each major concern addressed by this large requirement. If the flows or other components of a requirement could be moved to another requirement, it could be used the Move Activity refactoring [20]. After extracting or relocating requirements, we sometimes need 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 Requirement refactoring [20] could be used to provide more appropriated names. This refactoring opportunity is particularly important when there is a limit for the size of each requirement, set by the organization's Software Quality Assurance Team.

Table 9. (Partial) Extract Requirement Refactoring [20]

Name	Extract Requirement
Context	A set of inter-related information is used in several places or could be better modularized in a separate requirement. Alternatively a requirement is too large or contains information related to a feature that is scattered across several requirements or is tangled with other concerns.
Solution	Extract the information to a new requirement and name it according to the context.
Motivation	This refactoring should be applied when there are large requirements that can be split into two or more new requirements. These large requirements include a great deal of information that is difficult to understand. Furthermore it is not easy to locate the needed information quickly [23, 24].
Mechanics	The following activities should be performed: **1**. Create a new requirement and name it. **2**. Select the information you want to extract. **3**. Add the selected information to the new requirement. **4**. Remove the information from the original requirement. **5**. Make sure the original requirement is acceptable without the removed information. **6**. Update the references in dependent requirements.

The currently catalog has the following refactorings: *extract requirement*, *rename requirement*, *move activity*, *inline requirement*, and *extract alternative flows* [20], *Extract Early Aspects* [21]. Each refactoring contains the context that suggests the application of the refactoring, the type of solution provided, a motivation for the transformations, its mechanics (i.e., a set of well defined activities) and an example of with the refactored description. For example, the Table 9 shows the *Extract Refactoring*. Due to restriction of space the example of this refactoring is not showed.

The Patterns currently described in the Catalog are: *Untangled Requirement*, *Gathered Requirement*, *Unique Requirement* [22], *Uncoupled Requirement*, *Small Requirement* [10], *Simple Requirement* and *Useful Requirement*. The patterns follow the style of [25], like the patterns described in design level [26] with the inclusion of many sections: i) pattern name, ii) problem, iii) context, iv) forces, v) solution, vi) example, vii) resulting context, viii) rationale, ix) related patterns and x) know uses.

Step - 6. Requirements Model Improvement

After all cost analysis and 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 model. The following steps should be performed during this phase:

6.1. Apply the solutions selected - this task is (manually) performed by the software quality assurance team quality. The team should be careful when applying the solution, to obtain the benefits of the approaches. For example, following the steps all the mechanisms described in a Refactoring, analyze if the functionally remains the same and update the references.

6.2. Store the results - all results of the solutions applied need be stored in the baseline; this action will help the improvement of future evaluation with the AIRDoc. Solutions that needed to be modified, by some specific characteristic, could be reused in later projects.

6.3. Compare the models (before and the after the use the AIRDoc) - after making the improvements it is important to collect the metrics again and to compare the level of improvement gained. This data needs to be stored to create a base line of the bad and good solutions to some context.

3. Case Study: Applying the AIRDoc in a Real and large Requirements Document

The Brazilian Federal Revenue Service (Receita Federal - RF) is subordinated to the Finance Ministry and responsible for the collection, administration and auditing of a plethora of federal taxes. To process the huge amount of data (billions of registers of all sorts) that originates from its fiscal activities, the RF holds a partnership with SERPRO, a Finance Ministry subsidiary software company, for the development of automated solutions in support of tax analysis.

SERPRO is a large company with development units broadly spread throughout 10 capital cities of Brazil. The company employs more then 2.500 software engineers and has a history of successful and awarded solutions [27] built along 40 years as a partner of the Brazilian RF. These solutions offer full-automated support for a multitude of aspects comprising fiscal 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 in the highest administration levels of the Brazilian RF.

To validate our approach we asked SERPRO to provide a large real requirements document. For this case study, the chosen artifact is the "adjustment tax" requirements model. Basically it describes the correction of the amount of taxes paid by citizens. The requirement document consists of a use case model, shown partially in the Figure 5. Due to space limitations, only some descriptions of use cases are shown in this paper. Following are showed how the AIRDoc's approach could be used to improve the quality of the "Adjustment tax" requirements model.

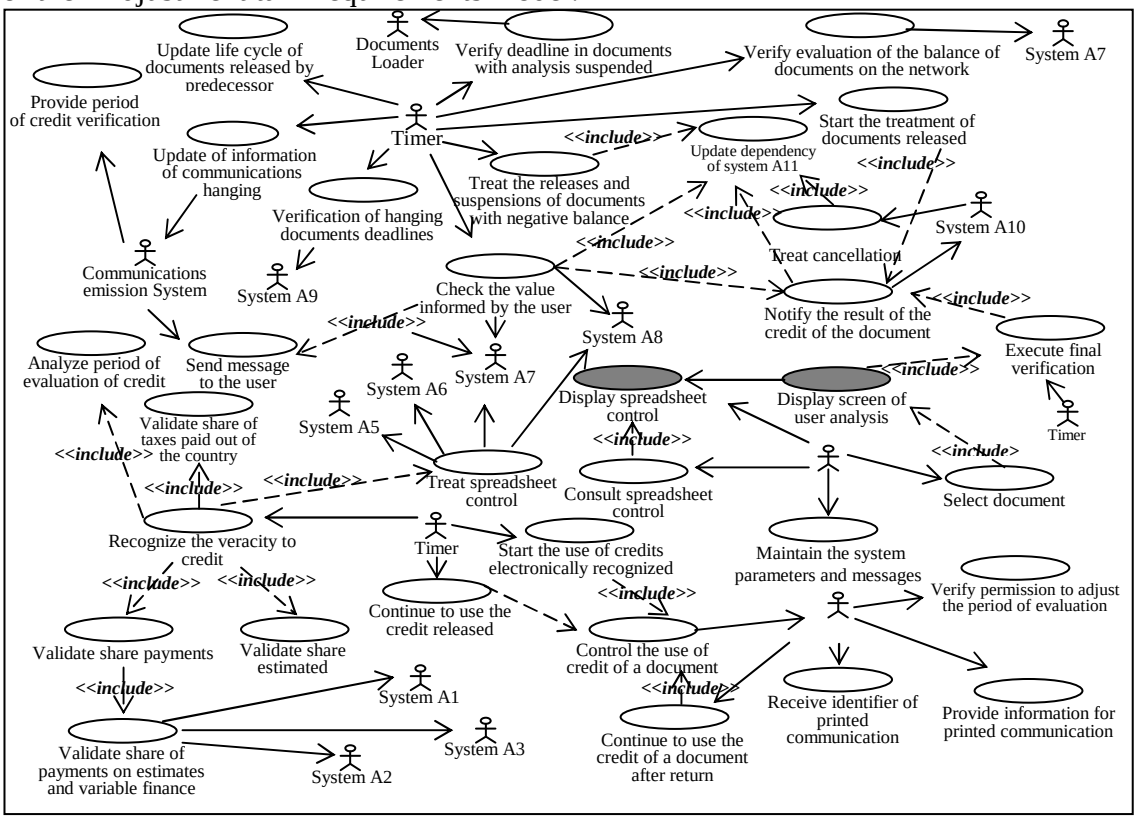

Figure 5. Partial use cases model of the Adjustment Taxes system

Step - 1. Plan Elaboration

The tables 10, 11 and 12 show, (partially and in condensed version), the artifacts of the first AIRDoc step.

Table 10. The quality team (partially)

Project Team Role	Quality Control and Quality Assurance Responsibilities	Assigned Resource / Members
Requirement Engineer	Development and management of the requirements document that are being evaluated	Helena Cristina Bastos
Reviser/Reader	Revise the metric applications and others data generated	Castro / Alencar / Araújo / Moreira / Penteado

Table 11. Tools and/or others resources (partially)

Tool / Resource Name	Tool/Resource Purpose/Use	Costs Details (time and/or monetary)
Text editor	Generate all the documentation of the evaluation.	A role needs to be allocated to this resource.
Spreadsheets editor	Generate graphics and reports about the metrics applications	A role needs to be allocated to this resource.

Table 12. Template to quality requirements

Goal	Requirements Model	Requirement in focus	Quality attribute
Improve the Maintainability	Adjustment Taxes	Display - use cases: i)Display spreadsheet control, and ii)Display screen of user analysis	Maintainability – we want to know how easy it is to maintain or/and to add new features to the "Displays requirements".

Step - 2. GQM Definition

Description of requirement model functionality

The requirements model called "Adjustment Taxes" makes corrections to the amount of taxes paid by citizens. At a given date, citizens send to RF, by Internet, a document (the so called "Declaração de Ajuste Anual de IRPF") where among other information it is declared the amount of taxes collected. The system checks this information and sends out messages notifying the citizen if values informed are correct.

Description of the requirements that will be evaluated

The system was developed in modules by several developers distributed in different parts of country. We are concerned in evaluate and improve the modules that deal with concern of display, which is described by the use cases "Display spreadsheet control" and "Display screen of user analysis", Figure 5.

The requirement of display is related with to show the screens with some information. This information may be a result of a source or a simple menu where the user could be to access another options of the system.

Description of the quality attributes and justify

In order to decide which quality attributes could be evaluated and improved, we considered the goals of the company as well as some key concerns mentioned by the requirements engineer in charge of the requirements models. We also discussed the nature of the system being developed, which demanded the definition of modules to be developed by several teams which were geographically distributed in Brazil. Given these constraints, it became clear that it was important to have high standards of understandability and maintainability of requirements models by all teams involved.

In this paper we focus on maintainability since this is an important concern related to the time spent to correct or modify a given requirement. Another characteristic that was considered is the fact of the company is certified at CMMI level 2 and desires in the near future to achieve CMMI level 3, thus requiring a good discipline of the requirements management.

We chose to use the quality model described in [15] and shown in Figure 3. This quality model captures the relationships between the maintainability attribute together with its factors, internal attributes and metrics.

Definition of the goal of the evaluation

"Assess in the Adjustment Taxes requirements model the **Display Requirement** with a view to predict its **maintainability**".

Questions Elaboration

Table 13 shows the set of question related to the given quality model [15].

Table 13. (Partial) question from the case study

Q1			How easy is it to understand the display requirement? **(Understandability)**
	Q1.1		How is the document composed? **(size)**
		Q1.1.1	How many steps are required to specify the display requirement?
		Q1.1.2	How many steps are there in the overall requirements document*?*
		Q1.1.3	How many use cases are required (contribute) to specify the display requirement?

Metrics Definition

For each question related to some internal attribute a set of metrics is defined. Thus questions: *Q1.1.1, Q1.1.2* and Q1.1.3 are directly related (answered) by metrics M6, M7 and M8 respectively (see Table 14).

Table 14. Relationship of metrics and data required

Metrics		Data required
Q1.1.1	M6 - How many steps are required to specify the display requirement?	Count the total numbers of steps that describe the display requirement.
Q1.1.2	M7 - How many steps are there in the overall requirements document*?*	Count the total number of steps that exist in overall requirements document.
Q1.1.3	M8 - How many use cases are required to specify the display requirement?	Count the number of use cases where there is, at least, one step that contributes to the specification of display requirements

Hypothesis Elaboration

Table 15 shows the hypothesis that answering the question Q1 using a function that make a relationship between the metrics shown in the Table 14.

Table 15. Function and hypotheses for size for our case study

Q1	How easy is it to understand the display requirement? **(Understandability)**
H1a	The display requirement is *easy* to understand, because the value of the H1 function is *lower* than 0,04
H1b	The display requirement is *hard* to understand and / or to extend, because the value of the H1 function is *greater* than 0,04
Function H1	(M6/M8)/M7 → this function shows the relationship between the average size of use cases steps used to describe the display requirement and size of all use cases of the requirements model. Thus we expected examine how homogeneous is the size of the use cases.
Note	If the value of function H1 is between 0,04 and 0,01 then the size of the use cases used to describe the display requirement is acceptable.

Step - 3. Data Collection

Table 16. M6, M7 and M8 metrics values

Metrics	Value
M6 - How many steps are required to specify the display requirement?	798
M7 - How many steps are there in the overall requirements document*?*	1533
M8 - How many use cases are required to specify the display requirement?	2

Step - 4. GQM Interpretation

Table 17. Analysis of the hypothesis H1a and H1b from our case study

Function H1	(M6/M8)/M7 → (798/2)/1533 = **0,26**
Conclusion	The hypothesis H1a was refuted, because the value of the function H1 is **0,26 (>0,04)**. Hence hypothesis **H1b** was supported.
Answering the question: "**Q1.1.** How easy is it to understand the requirement of display? *Display requirements* are hard to understand (Hypothesis **H1b**)	

We have analyzed the entire set of hypothesis for our case study. The results for hypothesis H1a, which is based on function H1, are described in Table 17. After

analyzing all the hypotheses of the case study, it was inferred that the *Display requirements* is **not ease to understand**. Moreover, the analysis of the 2 use cases that describe the *Display requirements* indicates symptoms of **large requirement** (Table 8).

Step - 5. Plan of Requirements Model Improvement

We choose the Extract requirement [20] solution (see Table 9) to address the **large requirement** problem. Then, we followed the proposed solution. The result is a revised requirements model depicted in Figure 6.

Step - 6. Requirements Model Improvement

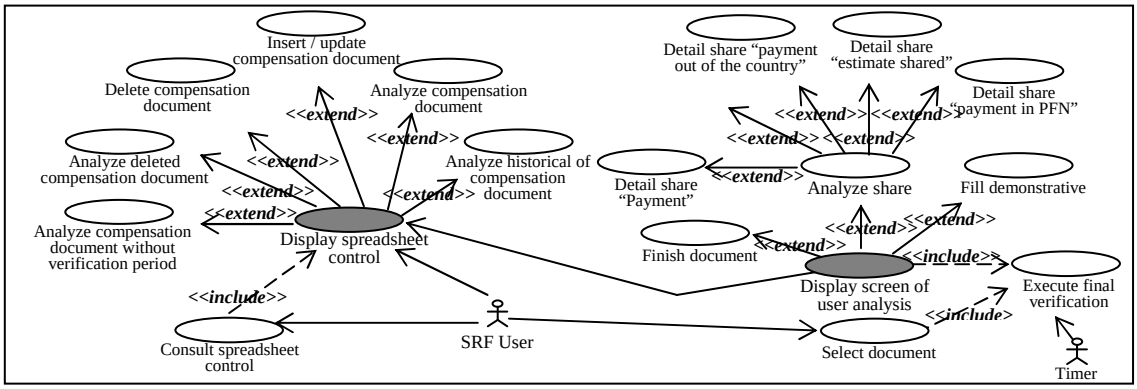

Figure 6. Partial use case model of the Adjustment Taxes system (after the suggested improvement)

Figure 6 shows the results of your case study, after the use of the *Extract Requirement* refactoring [20]. The **Display requirements** were divided in others module, and <<extend>> links were used. According to Jacobson [28, 29] these "<<extend>>" links cause less coupling than <<include>> links. In this case study we came to the conclusion that new model, now with the *Display requirements* divided in others use case modules, is more understandable than the original requirements model showed in Figure 5. The same metrics were used and the data collected indicated that the new model is more understandable (M6=767, M7=1532 and M8=15), although it has increased the amount of use cases. The new value of function H1 ((M6/M8)/M7) is 0,03 between the acceptable range shown in note of Table 15. The improvement in the understandability derives from the fact that the new use cases used to describe the display requirement are now smaller and more homogeneous.

4. Related Work

Inspections, such as reading techniques can be used to identify defects in software artifacts. In this way, inspection methods help to improve software quality, especially when used early in software development [30]. The idea of using metrics in requirements models as proposed by AIRDoc is to have a tool support that can be implemented to store, apply and help to summarize the data results. Software metrics provide a way to automate the extraction of reusable software components from existing systems, reducing the amount of information that experts must analyze [31]. In this way we expect to have a truthful picture of the requirements model. Using metrics together with the QGM approach we can interpret the metrics results to propose possible solutions.

Design patterns [26] and refactorings [15] are reusable solutions applicable to software artifacts. Patterns are well structured solutions that where used in another artifacts and by others software engineer, in the context of the solution of a pattern exists a little description of "how" and "when" to use the pattern. Refactorings provide solutions to specific problems without to change of the behavior. Both techniques are usually applicable to design and code level. In our work we make a novel contribution as we use these techniques at the problem level. Hence, AIRDoc proposes a set of patterns and refactorings to apply at the requirement level. In doing so it also suggests some direct mappings between the problems found with the solutions proposed.

5. Conclusions

The goal of the AIRDoc is to evaluate and improve requirements models. Our current focus is related to reusability and maintainability. But the AIRDoc might be used to evaluate other quality attributes. Table 18 summarizes the context where the AIRDoc is applicable and what could be possible results of its use.

Table 18 – When to use the AIRDoc

When to use?	Negative Forces	Positive Forces
a. During the **initial phases** of the software development: If the requirements models are reasonably complete and the requirements engineer (or the software quality assurance team) decides to evaluate and improve the quality of the requirements models.	The requirements models describe some kind of a contract (between the client and the software developers), so the proposed techniques must preserve the semantics of these models.	Techniques such as requirements refactoring and patterns could be used. Moreover, if adequately applied they can preserve the requirements model semantics.
b. During **corrective evolution**: The requirements models are released, together with other software artifacts and the requirements engineer (or the software quality assurance team) identifies **problems** or **errors** in the requirements models. **c.** During the process of **perfective evolution**: Requirements models are released together with other software artifacts, and the requirements engineer (or the software quality assurance team) decides to **improve** the requirements model.	The cost of the structural changes in the requirements phase needs be calculated. This is necessary because structural changes may impact the next artifacts in the software development process.	The improvement in the requirements models also impacts and contributes positively to the quality of others artifacts (those that depend on and are generated from the requirements models).

The case study based on a real and complex requirements model provides some indication that the AIRDoc may be applied to an industrial scale requirement models. Moreover, based to the value of metrics collected during the exercise we might infer if there was any quantitative improvement with respect to the quality attribute at study (i.e. namely **Understandability**). Of course more empirical evaluation is required to validate our approach, and the next step is to do some qualitative evaluation of the AIRDoc approach. Several on-going case studies are also under way. Moreover, tool support is under development.

Acknowledgements

The authors thank Helena C. Bastos (Manager of SERPRO – Recife) on your fundamental contribution in this work. This work was supported by several research grants: CAPES/GRICES Proc. 129/05, CNPq Procs. 308587/2007-3, 478132/2007-7.

References

1. Wiegers, K. E. (2003) Software Requirements. Microsoft Press, Second Edition.
2. Firesmith, D. (2007)Common Requirements Problems, Their Negative Consequences, and Industry Best Practices to Help Solve Them, in Journal of Object Technology, vol. 6, no. 1, January-February 2007, pp. 17-33.
3. Boehm, B.W., Sullivan, K.J. (2000) Software economics: a roadmap. In: ICSE – Future of SE Track. 319–343.
4. Pressman, R. (2005) Software Engineering: A Practitioner's Approach. McGraw-Hill.
5. Schneider, G., Martin, J., Tsai, W. (1992) An experimental study of fault detection in user requirements documents, ACM Transactions on Software Engineering and Methodology (TOSEM), v.1 n.2, p.188-204.
6. Travassos, G.H., Shull, F., Carver, J. and Basili, V. (1999) "Reading Techniques for OO Design Inspections," Proceedings of NASA/GSFC, Greenbelt, MD, December.
7. Elssamadisy, A., Schalliol, G. (2002) Recognizing and responding to bad smells in extreme programming. In: Proceedings of the 24th International conference on Software Engineering.
8. Xu, J., Yu, W., Rui, K., Butler, G. (2004) Use case refactoring: a tool and a case study. In: Software Engineering Conference, 2004. 11th Asia-Pacific. 484–491.
9. Meyer, B. (1997)Object-oriented software construction (2nd ed.), Prentice-Hall, Inc., Upper Saddle River, NJ.
10. Overgaard, G., Palmkvist, K. (2004) Use Cases Patterns and Blueprints. Addison Wesley Professional.
11. Mens, T. and Tourwe, T. (2004) A Survey of Software Refactoring. IEEE Transactions on Software Engineering, Vol. 30, nº 2, February.
12. Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The goal question metric approach. Encyclopedia of Software Engineering, pages 528–532.
13. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2000) Experimentation in Software Engineering: An Introduction Series: International Series in Software Engineering , Vol. 6, 228 p.
14. IEEE Standard for a Software Quality Metrics Methodology, IEEE Std. (1992) 1061-1992.
15. Ramos, R. A., Araújo, J., Castro, J. F. B., Moreira, A., Alencar, F., Silva, C. (2006) *"A Quality Model to Evaluate Requirement Documents"(in portuguese*) In: XV Jornadas de Ingeniería del Software y Bases de Datos. Sitges - Barcelona.
16. Ramos, R.A., Araújo, J., Castro, J., Moreira, A., Alencar, F., Silva, C. (2006): "Uma abordagem de instanciação de métricas para medir documentos de requisitos orientados a aspectos", in: 3º Brazilian Workshop on Aspect Oriented Software Development - WASP2006. Florianopolis, Brazil.
17. Fenton, N.E., Pfleeger, S.L. (1997) Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company.
18. Jacobson, I., Griss, M., Jonsson, P. (1997) "Software Reuse: Architecture, Process, and Organization for Business Success", in: Addison Wesley.
19. Firesmith, D. (2007) "Common Requirements Problems, Their Negative Consequences, and Industry Best Practices to Help Solve Them", in Journal of Object Technology, vol. 6, no. 1, January-February 2007, pp. 17.
20. Ramos, R., Piveta, E., Castro, J., Araújo, J., Moreira, A., Guerreiro, P., Pimenta, M., and Tom Price, R.(2007). Improving the Quality of Requirements with Refactoring. In: VI Simpósio Brasileiro de Qualidade de Software – SBQS2007, Porto de Galinhas, Recife, Pernambuco, Brasil, Junho 27 – 30.
21. Ramos, R. A.; Araújo, J. ; Moreira, A. ; Castro, J. ; Alencar, F. and Penteado, R. (2008) Early Aspects Requirements. In: XI Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS 08), Recife - Pe. Proceedings of XI Iberoamericano de Ingeniería de Requisitos y Ambientes de Software, feb.
22. Ramos, R. A.; Araújo, J.; Moreira, A.; Castro, J.; Alencar, F. and Penteado, R. (2007) "A Pattern to Duplicated Requirements", (*in portuguese*) in: 6th Latin American Conference on Pattern Languages of Programming (SugarLoafPlop'2007), Porto de Galinhas, Recife, Pernambuco , Brazil.
23. Alexander, I.F., Stevens, R. (2002) "Writing Better Requirements", Pearson Education Limited..
24. Sommerville, I. (2004) Software Engineering, 7th edition. Pearson Education.
25. Alexander, C., et. al. (1977) A Pattern Language, Oxford University Press, New York.
26. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995) Design Patterns - Elements of Reusable Object-Oriented Software. Reading-MA, Addison-Wesley.
27. Brazilian Federal Revenue Service Technologic Awards. (2008): see http://www.serpro.gov.br/
28. Jacobson, I. (2003) Use cases and aspects - Working seamlessly together. Journal of Object Technology 2(4).
29. Jacobson, I., Ng, P.W. (2005) Aspect-Oriented Software Development with Use Cases. Addison-Wesley.
30. Travassos, G. H., Shull, F., Fredericks, M. and Basili. V. (1999) Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality. Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Denver, Colorado.
31. Caldiera, G. and Basili, V. (1991)"Identifying and Qualifying Reusable Software Components," IEEE Computer, vol. 24(2): 61-70, February.