QualiCES: a method for verifying the consistency among documents of the engineering phase

Conference Paper · October 2012

DOI: 10.1145/2379057.2379077

3 authors:

Luã Marcelo Muriana
State University of Campinas (UNICAMP)
14 PUBLICATIONS **69** CITATIONS
SEE PROFILE

Cristiano Maciel
Federal University of Mato Grosso
339 PUBLICATIONS **1,577** CITATIONS
SEE PROFILE

Fabiana Freitas Mendes
Aalto University
46 PUBLICATIONS **207** CITATIONS
SEE PROFILE

QualiCES, A Method for Verifying the Consistency Among Documents of the Requirement Engineering Phase

Luã Marcelo Muriana
Federal University of Mato Grosso
UFMT
Cuiabá, Mato Grosso, Brazil
luamarcelo17@gmail.com

Cristiano Maciel
Federal University of Mato Grosso
UFMT
Cuiabá, Mato Grosso, Brazil
cmaciel@ufmt.br

Fabiana Freitas Mendes
University of Brasilia
UnB
Brasilia, Brazil
fabiana.mendes@unb.br

ABSTRACT
During the initial software specification phase, requirement document, use cases description and interface prototypes can be generated as a way to aid in the construction of system data. The consistency among these documents is a quality attribute which must be emphasized at this phase of the software development process. The QualiCES method is presented herein; it allows assessing the consistency among these software documents, and is supported by a checklist and by a consistency metrics developed to this end. As benefits, there is defect detection and a software quality warranty from the beginning of software development. The method was executed in a case study. Based on the results, the viability for applying the method can be verified, as well as the proposal innovation degree.

Categories and Subject Descriptors
D.2.4 [**Software/Program Verification**]: *Model checking, validation*; D.2.8 [**Metrics**]: *Performance measures;* D.2.9 [**Management**]: *Software quality assurance (SQA)*

Keywords
Software Quality; Software Inspection; Requirement Engineering; Consistency.

1. INTRODUCTION
Software development is a process that has gained increasing importance in the pre-sent society. As software development has grown, so has the complexity of clients' specifications. Thus, development costs, production time and maintenance difficulties increase, and the finished products are also subjected to different kinds of problems.

So that these problems do not last, there is a series of software validation and verification activities, such as the software inspection techniques that help to discover problems and, as a consequence, cooperate towards attaining software quality. Software quality is understood as the degree at which a system, component or process complies with the specified requirement, and/or with the end user's expectations [7].

SIGDOC'12, October 3–5, 2012, Seattle, Washington, USA.

Studies such as [15] and [11] show that conducting this validation and verification process from the very beginning of the software lifecycle reduces the costs and efforts spent to repair errors that may occur, besides ensuring that software quality is present since the beginning of a software lifecycle process. For this, it is important for quality assurance techniques to be used. In this study, this is obtained by means of a validation and verification method proposed for the requirement engineering phase, which is supported on the application of a checklist and on a consistency analysis metrics among documents generated in the initial phase of the software development process; the method was named QualiCES (Quality via Consistency in Software Specifications). Therefore, our aim is to define, to develop and to analyze a method that helps with the perception of consistency among software documents.

Faced with the existence of other works that verify the quality of a particular artifact in software design, this research focuses on the analysis of the consistency among three types of documents still at the requirement engineering phase: requirement document, use case description, and interface prototypes, viewing problem detection. Added to the motivation of this work is the fact that works with this research focus have not been found.

After the development of the QualiCES method, it was tested in the "Social eGov" design [9] allowing the verification of its documentation quality by means of the consistency among artifacts. .This research can be seen completely in [17].

The following Sections will detail this method. Initially, issues related to requirement engineering and to software quality are approached in Section 2. In Section 3, studies related to this research are presented. In turn, Section 4 presents the QualiCES method developed in this study. Section 5 reports the case study conducted as a way of verifying the viability of the method proposed. Lastly, Section 6 provides the final considerations.

2. THEORETICAL REFERENCES
According to [14], software development is composed of four basic and fundamental stages: requirement engineering, development, software validation and evolution.

During the requirement engineering phase, the needs and functionalities to be met by the software are identified and formalized in different documents. Despite being such an initial phase, quality must be taken into account, since requirements are the software foundations.

When the phase is conducted having quality in mind, it results in complete, clear, non-ambiguous requirements that follow the

industry standards. Yet, for this to be attained, a lot of effort is necessary [8; 2].

During the requirement engineering phase, different software documents are generated, among which the following may be detached: requirement document, description of use cases, and interface prototypes.

The requirement document serves as a reference for the next phases in the software development process. If an error is not detected at the very beginning of this process, it may propagate and thus affect the other artifacts to be developed. Hence, finding and correcting errors in the requirement artifacts early in the software development process prevents future rework.

The use case description may derive from a requirement document, or may be developed so as to represent the system requirements. However, when building a use case model from the requirements, [3] says that the developer must be very aware as there will be defects deriving from previous phases, even if they have been inspected.

Another way to obtain requirements and/or verifying them is by means of prototyping the desired software. [14] says that a prototype may be used in a software process in several ways, and within requirement engineering, a prototype may aid the discovery and verification of the system requirements.

Dealing with requirements also involves their management, that is, once defined, they have to be made stable enough to ensure an adequate execution in later phases. If changes are necessary, they have to be adequately controlled. A tool used to this end is traceability, which usually denotes the degree at which a relation can be established among different development artifacts [11], i.e., traceability defines the relationship among the different parts of a system. [11] also says that the information obtained from traceability may help to ensure quality.

The QualiCES method analyzes the consistency among requirement documents and, as a consequence, assumes they are related. Hence, in case there is any inconsistency among them, they must be corrected. Therefore, requirement management and especially traceability are not directly related to this study; however, the method contributes to a correct traceability among the requirement documents.

Another stage of software development is validation, also known as Verification and Validation (V & V). However, "the term 'validation' is used inconsistently, both in the literature and in practice" [11]. Therefore, the definition by [16] is adopted herein, for both validation and verification. [16] says Validation means to analyze whether or not the right system is being built. On the other hand, [16] says Verification analyses whether or not the system is being built correctly. Therefore, this study is concerned with verifying if there is consistency among the documents of the requirement engineering phase so as to build the right system with quality.

[11] believes that software validation at initial phases of the software development process reduces the costs and the effort for correcting errors in phases following the development process. Detecting defects in the requirement document is seen as one of the most efficient and effective techniques of quality assurance in software engineering [12].

Within the V & V process, there are several approaches such as software inspection, software test, walkthrough and peer review [13]. We conducted a more in-depth study when approaching software inspection, seeing that this is currently one of the most common ways of software quality assurance [4]. Thus, this work is restricted to the software specification and software validation phases, considering that the latter should occur all along the development process.

Software inspection is defined as a formal assessment technique in which requirements and software design and even the program code are thoroughly examined by a person or a by a group of people aiming to detect defects, development standard violations and other problems [1]. [10] generalizes saying that software inspection is a specific type of revision which may be applied to all software artifacts and has a strict and well defined defect detection process, as can be seen in Figure 1 as follows.

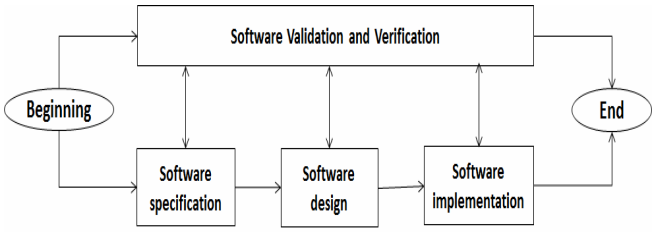

Figure 1. The software development process adopted for this work.

So as to improve understanding and the search for defects, there are different revision techniques in the software inspection process, among which the Checklist-Based Technique (CBT). This technique creates a guide (the checklist) which reminds the tester of the topics to be analyzed in the artifact and hence also helps to search for defects or deficiencies. "Individual items in a checklist may enumerate, prioritize or propose questions to help the reviewer to discover defects" [10].

Checklist use is considered a standard software inspection technique in many companies, as stated by [15]. This may be justified, as CBT may be applied as a complement in any other validation and verification technique [11]. For this reason, CBT was chosen as the base for this study.

Therefore, assuring software quality is ensuring that there is coherence in the ideas and information presented by the requirement documents, use cases and interface prototypes by the detection of problems, thus assuring that documents are consistent.

3. RELATED WORKS

In this work context, several studies were conducted in order to obtain software quality from the very beginning of the software development process.

Table 1. A summary of the focus of the studies.

Approach	Focus	Author
Requirement	Requirement quality.	Jani (2010)
	Validation and verification of requirements by means of a more interactive process.	Gusev (2010)
Use case	Construction of use cases for detecting defects in the requirement document.	Belgamo et al. (2005)
	Defects detection by the application of a checklist.	Anda, Sjøberg, (2002)
	Quality of the description of use cases.	Aurum *et al*. (2004)
	Quality of the description of use cases, and Quality of the use cases representation.	Gregolin (2007)
Software inspection	Comparison between software inspection techniques.	Thelin et al. (2003)

[8] conducted a study which analyzed the software specification requirement quality to assure that the quality is acceptable. A software quality assurance technique, a checklist was applied to the study to determine whether the requirement standards and procedures were or not being followed along the requirement specification phase. For this, the author identified quality attributes expected from a requirement specification document: complete, consistent, correct, modifiable, relevance rating, traceable, non-ambiguous, understandable, testable, verifiable, validable.

[6] goes further and proposes a more iterative verification and validation way. The author states that a way of obtaining a better quality not only in requirement specification, but all along the software development process, is to verify and validate all the artifacts generated all over the software process. The author says that, when the option is to work with an activity flow in which an activity will only be started after the previous activity is fully completed and approved, some defects will not be detected once they will only appear when tested in other phases of the software development process. Hence, the author proposes that other artifacts are generated for requirement verification and validation, such as prototypes, and that verification is not limited to the checklist alone, for example. To conclude, the author stresses that this activity flow does not assure that requirements are error-free, but assures that potentially critical defects affecting the implementation are identified.

In [15] study, two software inspection techniques are compared: the use-based one, and the checklist-based one. The authors say that the use-based technique is better than the checklist-based one in terms of effectiveness and efficiency, as it is capable of finding the defects that most affect users, besides its application being faster.

Both [6] and [15] criticize the checklist-based software inspection technique in their studies. However, the comparison proposed by the authors analyzes the detection of the most serious defects in the user's perspective, and the method proposed here analyzes more comprehensive problems such as the detection of lack of requirements, and inconsistency among software documents and artifacts.

[3] compare the checklist efficiency and effectiveness with the tool developed in their study: TUCCA (Technique for Use Case Model Construction and Construction-based Requirements Document Analysis). TUCCA is composed of two inspection techniques: AGRT (Actor Goal Reading Technique) which views determining the system actors and their goals, and UCRT (Use Case Reading Technique) which aims to determine the use case model. Jointly, these techniques collaborate with the construction of the use case besides revising the requirement specification document. With the study and the experiment conducted, the authors showed that TUCCA is effective in building the use case and in defects detection, once the technique developed obtained results as good as the checklist-based technique. The authors consider that there are always defects in an artifact developed and that the requirement engineering phase is essential for the success of the software development next phases. Nevertheless, the method proposed by the authors is driven towards defect detection during the use cases building process; in turn, the method proposed herein is guided towards the identification of incoherence problems among use cases, requirement document and prototypes.

Once the use case is built, [1] say that its quality in terms of correctedness, completion, consistency and good understanding of the functional requirement are important to the final quality of the software product. Thus, [1],[2] and [5] developed, in each of their studies, a checklist that sought the identification of defects in the use cases. Some of the items analyzed by their respective checklists were considered in the elaboration of the QualiCES method checklist, as a dependence among use cases, actors and the lack of functionalities, for example. Yet, the method here proposed adapts these items so that a more interactive analysis can be performed, as proposed by [6] and thus not being limited to a single software document or artifact at a time.

[6] showed that when the validation and verification process occurs since the initial phase of the software development, more defects are likely to be detected. Hence, the author proposes a more interactive inspection process. The method developed herein considers as documents to be analyzed: requirement specification document, description of use cases and interface prototypes. That is, a method that analyzes the consistency between documents and artifacts generated during the requirement engineering process is proposed, not being limited to the quest for use case quality or requirement quality alone.

Therefore, as in the studies aforementioned, the QualiCES (Quality de Consistency in Software Specification) method proposed here is supported on the use of the checklist technique and, at the end of its application, quality attributes are expected to be attained as proposed by [5] and [8]. A summary of the focus of the studies aforementioned is shown in Table 1.

4. QualiCES METHOD

QualiCES (Quality via Consistency in Software Specifications) is a method that aims to assess the consistency among software documents, especially among requirement specification, use cases and interface prototypes. For this reason, it is a quality assurance method that can be used at the very initial phases of software development, contributing to reducing design costs.

QualiCES is a method that does not aim to analize the quality or content of documents; instead, it deals with the consistency among documents. It is assumed that the documents have been developed in a high-quality manner, and that possible problems have already been checked.

For the QualiCES method to be applied, the documents to be verified must be *frozen*, i.e., the documents assessed cannot undergo any kind of alteration during the application of the method proposed.

The method is composed of five stages, described as follows:

1. **Identification of quality needs:** this stage consists of identifying the needs of the software development team to assure software quality, as for example, efficiency in the delivery of services or products, a fact that represents quality improvement; or dissemination of good practices related to improvement in the organization as a whole and not in specific projects. Having in mind the importance of software quality from the very beginning of the software development process, detecting problems still at the requirement engineering phase cooperates towards the final efficiency of products delivered, for example. Thus, for applying the QualiCES method, requirement documents, description of use cases and interface prototypes have to be available;

2. **Analysis of the necessary documents**: this stage accounts for gathering the necessary documents for analyzing and for verifying whether:

- The requirements of the requirement document are correctly identified; and
- Each use case has an interface prototype related to it or the other way round.

Use cases that do not have a corresponding interface prototype, or vice versa, cannot be analyzed. The method proposed requires that all the use cases and prototypes necessary are developed so as to have an analysis and a greater quality assurance. This stage is composed of the following activities:

a) Obtaining the necessary documents for the method application;
b) Analyzing the documents, verifying whether there is a use case and a prototype or a set of corresponding prototypes, as well as verifying whether the requirement document is complete.

3. **Checklist application:** The checklist proposed by the method has to be filled separately for each use case having its respective prototype, so as to detect defects among the documents and artifacts of the requirement engineering process. Section 4.1 details the checklist of the method created. The following activities compose this stage:

a) Choosing a use case and its corresponding prototype and filling in the checklist heading, so as to identify the application made;

b) For each of the checklist questions, analyzing the rule and providing the final analysis report; and

c) If some defect is detected, filling the fields directed to identifying defects.

4. **Consistency metrics application:** With the checklist application, defects are detected, and from them, it is possible to apply the consistency metrics for analyzing the quality of the artifacts and documents inspected. The metrics description can be found in Section 4.2. For that, conducting the following activities is made necessary:

a) After answering the whole questionnaire, collecting data for applying the metrics, i.e., adding the points of each group of defects, per items, and obtaining the total valid questions; and

b) Applying the consistency metrics according to the formulas defined in Section 4.2.

5. **Analysis of results:** the results obtained from the consistency metrics application should be analyzed according to the parameters defined in Section 4.3. The following activities compose this stage:

a) Analyzing the result obtained and recording it;

b) Going back to stage 3 and conducting the other subsequent activities and stages until all the use cases and their prototypes are inspected; and

c) Reporting to those responsible for the software specification so that the due corrections can be made, when necessary.

The QualiCES method execution flowchart can be seen in Figure 2. The numbering of the activities refers to the numbers of each stage of the method, as well as the activity to be conducted at each stage (Table 2).

The following sections will describe the three last stages of the method. Stages 1 and 2, respectively, are assumed not to require specific procedures for being carried out. Stage 1 requires the application of the method. In Stage 2, in turn, the existence of all the necessary documents for applying the method is verified, as previously described in item 2 of this section.

4.1 Stage 3 – Checklist

A checklist was developed to help to perceive inconsistencies among software documents, guiding the search for defects among documents.

In the checklist construction, questions were generated to analyze the documents, as well as the consistency among them. The questions were separated into five items to be considered during the inspection process. Each checklist question is composed of an assessment rule, which is found with the question, and the purpose of which is to help to fill in the checklist. As an alternative to answering the questions, four op tions are proposed. Whenever a problem is detected, the checklist provides the analysis of the problem identified in 3 different perspectives which have to be compulsorily filled in. The checklist proposed is composed of four parts, as depicted in Figure 3, which are: i)

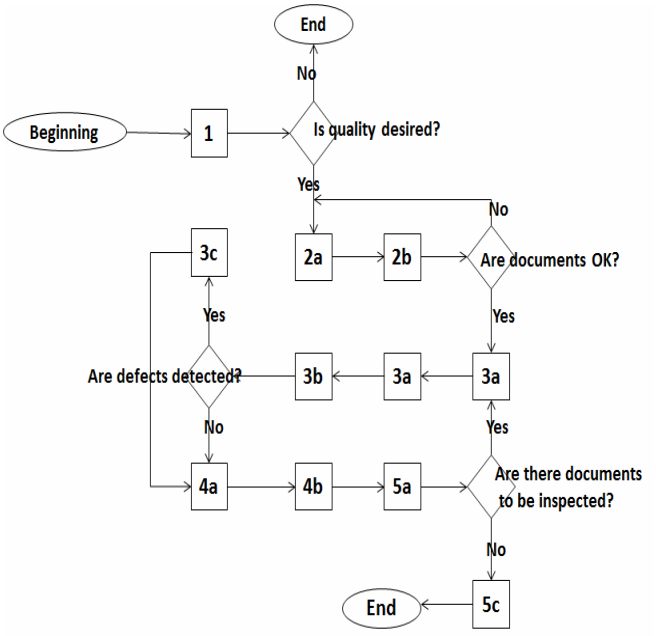

Figure 2. QualiCES method flowchart.

Table 2. Flowchart activities

Activity ID	Activity Description
1	**Identification of quality needs**
2	**Analysis of the necessary documents**
2a	Obtaining the necessary documents for the method application
2b	Analyzing the documents.
3	**Checklist application**
3a	Choosing a use case and its corresponding prototype and filling in the checklist heading.
3b	Analyzing the rule and providing the final analysis report
3c	Filling the fields directed to identifying defects.
4	**Consistency metrics application**
4a	collecting data for applying the metrics.
4b	Applying the consistency metrics
5	**Analysis of results**
5a	Analyzing the result obtained and recording it;
5b	Going back to stage 3
5c	Reporting to those responsible for the software specification.

items analyzed/questions/rules; ii) conclusion, that is, a response given to the question; iii) classification of the problem found; and iv) justification and/or observations concerning the item analyzed.

4.1.1 Items analyzed.

The checklist contains 28 questions, divided into five items to be simultaneously analyzed among the requirement document, use case description document and interface prototype. They are:

- **Content:** includes four questions analyzing the use case general description, verifying the reach of the objective proposed and of the functionality provided in the requirement document, both by means of the use case, and by means of the prototype. Furthermore, it groups items analyzing the data dictionary, as well as the terms used to identify the functionality objective;
  Functionality: counts on seven questions analyzing if both the use case and the prototype consider all the functional requirements related to the functionality described in the use case. Moreover, they aim to detect the omission of a requirement and/or of a requirements that does not apply to the use case or to the prototype inspected;
- **Functionality execution:** so that a certain system functionality is attained, some steps have to be followed, some restrictions and/or alternatives also have to be considered. Therefore, this section counts on questions analyzing whether the major and alternative flows, as well as the restrictions of a use case, can be followed in the interface prototype. Again, the existence of missing or spare steps in both artifacts is verified. There is also an analysis of whether the pre and post conditions of the use case are perceived in the prototype. This Section contains nine questions;
- **Dependence between functionalities:** some use cases need another one to complete a certain functionality in a system. Hence, this item accounts for analyzing whether the use case description and the prototype consider an include and exclude relationship among the use cases. This item is composed of four questions;
- **Users:** any system developed may have several types of users classified according to their function within a system. This item thus analyzes whether all the users/actors have been modeled both in the use case and in the prototype, besides analyzing whether roles and responsibilities were respected within the functionality proposed. This item is composed of four questions.

1	2 Conclusion				3			4
Items analyzed/Questions/Assessment rules	**yes**	**no**	**partly**	**not applicable**	**Hinders the objective**	**Correction Urgency**	**Functionality importance**	**Justification**
Item analyzed 1								
Question 1 *Rule refereted to question 1*								
Question 2 *Rule refereted to question 2*								
Item analyzed 2								
Question 3 *Rule refereted to question 3*								

Figure 3. Structure of the checklist proposed.

4.1.1.1 Questions assessments rules

So that each of the questions related to each item analyzed could be correctly interpreted, validation rules were defined, as in the examples as follows:

- **Rule 2.1** – The use case should describe all the functionalities

desired for it and for the related functional requirements. Related question: 5;
- **Rule 3.2** – The use case description should specify the main and alternative flow as completely as possible, allowing each of the flow steps to be executed by means of the prototype. Related questions: 13 and 16.

The purpose of the validation rules is to aid the one applying the method during the inspection process. The rules defined for each question show what and how it can be analyzed so that a problem is/is not identified.

4.1.2 Possible conclusions for the questions

By means of the validation rules of the questionnaire, it is possible to have four response options:

- **Yes/No:** for some questions, *yes* refers to situations in which the rule defined for a certain question analyzed is fully complied with, whereas *no* refers to situations in which the rule defined is not complied with. Yet, depending on the question, the opposite will also be possible for both possibilities.
- **Partly:** when the rule defined is partly complied with, i.e., when it is complied with, but some information lacks to complete it;
- **Not applicable:** when it is not possible to respond to a question or when it is impossible to analyze the request, or because some other issue has not been met and, therefore, the related questions cannot be answered. Attention must be given to this report, as it will be important for the consistency metrics application.

4.1.3 Classification of the problems detected

The aim of the QualiCES method is to identify problems by filling in the checklist developed herein. For this, the validation rules defined for each of the 28 questions composing the checklist must be taken as a base for interpretation. Thus, whenever a rule fails to be fully or partly complied with, a problem is detected, and therefore, the problem has to be analyzed in three different ways:

1. **Hinders the objective:** a defect may or not affect the implementation of a functionality. When a problem is perceived, the reviewer may therefore say whether it affects (*yes* or *no*) the use case objective and its functionality.

- **Yes:** When the objective affects the user, preventing him/her from using the functionality analyzed. Examples: a functional requirement is missing, some characteristics of the role of a system user are not considered;
- **No:** When the defect does not affect the reach of the functionality objective. Example: difference between the attributes of a data dictionary and the attributes identified in the prototype.

2. **Correction urgency:** a defect differs from another in several items, and one of them is related to correction urgency. Thus, when a problem is detected, the correction urgency should be analyzed. The urgency can be classified into three levels, as follows:

- **Short term:** When the defect detected has to be corrected before starting the software design phase. Example: missing requirements;
- **Medium term:** When the software design process can be followed, but the defect has to be corrected before the implementation. Example: adding a missing key in the prototype because the use case failed to foresee this action.
- **Long term:** When the defect can be corrected during or after the software implementation. Example: alteration in the name of the fields.

3. **Functionality importance:** Based on the user's perspective, the defect detected should be classified considering the importance of the functionality to the user, which may be:

- **Very important:** if the defect found prevents the user from executing a functionality. Example: the lack of a requirement;
- **Important:** if the defect found partly affects the user's interaction with the system. Example: the lack of a "cancel" key;
- **Not very important:** if the defect found does not at all affect the user's interaction with the system. Example: name divergence between the data dictionary of the use case description, and the name used in the prototype.

Hence, after filling in the checklist, and with the defects detected, the QualisCES method recommends the generation of indicators by means of the consistency metrics. The metrics details are presented in the next section.

4.2 Stage 4 – Consistency metrics

The consistency metrics was developed to support the QualiCES method so as to cooperate with a faster visualization of the number of problems detected once the metrics proposed specifically points to where the greatest problems lie. Therefore, the consistency metrics is a simple way of verifying the presence of quality as early as in the requirement engineering phase.

The consistency metrics is composed of three stages: data collection, data analysis and result presentation. The subsections as follows detail each of these stages.

4.2.1 Data collection and data tabulation .

At this stage, the data necessary to generate each of the indicators proposed by QualisCES are presented and described. This stage is composed of the following steps:

a) For each question answered in the checklist in which a problem is found, the *poX* scoring obtained is added in relation to the attributes "hinders the objective", "correction urgency", and "functionality importance", in which *X* is the number of the question being considered. This scoring is given considering Table 3.

Table 3. Definition of values for the possible answers.

Hinders the objective	Correction urgency	Functionality importance
Yes = 1	Short term = 3	Very important = 3
No = 0	Medium term = 2	Important = 2
	Long term = 1	Not very important = 1

b) For each item (content, functionality, functionality execution, dependence between functionalities, and users), the X scoring (*poX)* of the questions composing it is added, thus obtaining

the scoring of the questions related to the content item (poC), the scoring of the questions related to the functionality item (poF), the scoring of the questions related to the functionality execution item (poEF), the scoring of the questions related to the dependence between functionalities item (poDF) and the scoring of the questions related to the user item (poU). The scorings are recorded in a checklist as that illustrated in Figure 3.

c) Each item scoring previously obtained is multiplied by its corresponding weight; for this, consider Table 4.

Table 4. Weights concerning each item analyzed.

Item	Weight (P)
Content (C)	PC = 0.1
Functionality (F)	PF = 0.3
Functionality execution (EF)	PEF = 0.4
Dependence between functionalities (DF)	PDF = 0.1
Users (U)	PU = 0.1

The weights defined in the previous table were obtained by means of the number of questions existing in each item analyzed. For this, the total questions in each item analyzed was divided by the total questions in the checklist. The results obtained were approximated and thus the values in Table 3 were obtained.

PC = 4 questions / 28 questions = 0.14

PF = 7 questions / 28 questions = 0.25

PEF = 9 questions / 28 questions = 0.32

PDF = 4 questions / 28 questions = 0.14

PU = 4 questions / 28 questions = 0.14

It is worth stressing that, in case there are modifications in the checklist so that it can be adapted to other applications, such as the number of questions to be considered; for example, it will be necessary to make alterations in the values in Table 3.

d) The scoring obtained between the items and their respective weights are added, and the result obtained is divided by the total valid questions (TPV), thus obtaining the consistency quality coefficient (QC), as can be seen at number 1, in which TPV is the total number of valid questions (number 2), TP is the total number of questions in the checklist, and TPNA is the total non-applicable questions.

$$QC = (poC*PC + poF*PF + poFE*PFE + poDF*PDF + poU*pU)/TPV \quad (1)$$

$$TPV = TP - TPNA \quad (2)$$

4.3 Stage 5 – Analysis of results

After the application of the consistency metrics for each checklist filled in, the method proposed expects the results to be analyzed. Hence, this study considers that the consistency among the documents analyzed during the requirement engineering phase is given by using the following values:

- **Consistent:** when the result of the metrics application (QC) yields results smaller or equal to 0.199;
- **Not very consistent**: when the result of the metrics application yields results greater than 0.,3 and smaller than 0.699;
- **Inconsistent:** when the result of the metrics application yields results greater than 0.7.

The values defined for rating the results obtained were chosen based on the average of the results obtained with the application of QualiCES in the case study described in Section 5. Hence, more tests with these values are necessary.

4.3.1 Presentation format

For presenting the data collected, tabulated and analyzed, the tabular format was chosen. The table should have the following columns: use case, items, scoring of each item, total non-applicable questions (TPNA), consistency quality (QC), and final rating. Every time the checklist is filled in, an input is generated in the Table with sublines containing the name of the use case or of the prototype and each of the items analyzed. For each item, the scoring obtained is placed beside it, along with the TPNA, QC and the rating obtained. Observe Table 5.

5. CASE STUDY

The software design chosen for applying the method is described in this Section, as well as the results attained with the verification.

The case study was carried out with the verification of the documentation in a software development design being implemented by the "Social eGov" research project, which is currently composed by 14 researchers, comprising undergraduate students, master's candidates and professors of the Computer Science course. So far, the project has produced the following documents: requirement specification, use cases and interface; and artifacts such as modeling entity-relationship. The purpose of the "Social eGov" design is to cooperate and to expand the citizens' participation in the Government-related issues. For this, the group has developed a component framework, which has allowed the customization of resources during the e-participative systems development [9].

As determined by stage 2 of the QualiCES method, it is necessary to analyze the documents to be inspected. In this project, the following were documented:

- In the system requirement document, 23 functional requirements;
- In the requirement document, as well as in the use cases description document, four classifications of users for the system;
- 42 use cases;
- 17 interface prototypes.

Table 5. Model for tabulating the result obtained from the QualiCES method.

Identification	Item	Scoring	TPNA	QC	Rating
<name of use case or of prototype>	Content	<poC>	TPNA	QC	<Consistent or not very consistent or Inconsistent>
	Functionality	<poF>			
	Functionality execution:	<poEF>			
	Dependence between functionalities:	<poDF>			
	Users	<poU>			

After the documents analysis, it was possible to apply the checklist (stage 3), thus obtaining the necessary data for the application of the consistency metrics provided in stage 4 of the method proposed.

After the documents analysis, it was possible to apply the checklist (stage 3), thus obtaining the necessary data for the application of the consistency metrics provided in stage 4 of the method proposed.

As a result of the application of the checklist and of the metrics, it was possible to detect inconsistencies in all of the documents analyzed. The items that most presented defects were "functionalities" and "functionality execution".

According to the consistency metrics proposed herein, the checklist application showed that the documents and artifacts analyzed are: consistent, not very consistent or inconsistent. As from the values obtained with the QualiCES method application, it was possible to elaborate Table 6, which contains the rating of the results analyzed, as well as the number of defects found overall at every consistency level possible.

Table 6. Number of defects found for each final rating of the metrics.

Consistency level	Final classification	Number of problems detected	Percentage
Consistent	6	12	32.73%
Not very consistent	7	54	43.63%
Inconsistent	4	44	23.64%
TOTAL	**17**	**110**	**100%**

6. QualiCES METHOD ANALYZIS

The method QualiCES' aplication in the project "Social e-Gov" allowed to realize the lack of consistency among documents analyzed. As can be seen in Table 6, the results show that eleven documents were either not very consistent or inconsistent, totaling 98 defects detected. These problems could represent a loss to the quality of the system longer. Therefore, detection of them at this point of software development as stated [11] will reduce the cost and effort need for future corrections.

Based on the results obtained with this research, the QualiCES method showed to be viable for applications. The method viability is due to the advantages it presents:

- During the method application, three software documents are concurrently and interactively analyzed, allowing problems to be detected in any of them;
- The method proposed presents a metrics that analyzes the consistency among these documents according to the amount of problems detected;
- The way the results obtained are presented with the application of the checklist (see Table 4), facilitates the perception of where the greatest problems lie.

Therefore, as can be analyzed, the QualiCES method goes beyond the studies mentioned in Section 3. A large share of the studies conducted is supported on a single document for analysis so as to detect problems, such as [1], [2] and [5]. This study, in turn, as suggests [6], uses several documents simultaneously.

The V & V process is a way of assuring software quality. In this sense, with the QualiCES method application, some quality attributes could be analyzed, identified and verified, such as: completeness, internal consistency, reachability, traceability, correctedness and accuracy. While [5] adapts these attributes to the use cases quality, [8] adapts them to requirement quality. Here, quality attributes are adapted so as to verify the consistency among the requirements and use cases and prototypes. Therefore, in the requirement engineering phase, the QualiCES method allows for:

- **Completeness:** all the functionalities described by the use cases and represented in the prototypes are specified in the requirement document;
- **Internal Consistency:** functional requirements, use cases and prototypes are not conflicting among themselves;
- **Reachability:** the prototype related to the use case represents the description and the functional requirement specification, also meeting possible non-functional usability requirements;
- **Traceability:** being each requirement and use case duly identified, it is possible to trace the consistency among them;
- **Correctedness:** being the functionality correctly described by the use case and represented by the prototype, their definition in the requirement document describes their objective accurately;
- **Accuracy:** once accurately described, functional requirements and use cases assure the prototype reachability.

7. FUTURE WORKS AND CONCLUSIONS

The QualiCES method (Quality via Consistency in Software Specifications) proposes a sequence of steps aligned with the software development phases and may be applied in the requirement engineering phase, by analyzing the consistency among software documents by means of problem detection.

For this, the method proposes a checklist, which supports the analysis of content, functionalities, functionality execution, the dependence between functionalities and the role of users by verifying the requirement document, description of use cases and prototypes. The problems detected by the checklist application are classified, and are then analyzed by means of a consistency metrics which allows classifying the documents analyzed into consistent, not very consistent and inconsistent. The method also proposes a way of syntactically presenting the inspection results.

Besides problem detection, the method proposed showed to support the quality assurance process. Thus, by means of the checklist developed, several software quality attributes are contemplated: completeness, internal consistency, reachability, traceability, correctedness and accuracy. For these attributes to be attained, the work proposed presented a set of assessment rules for the checklist questions, which serve as design guidelines in the software engineering area, independently of the checklist. The rules defined in the QualiCES method provide important recommendations concerning content, functionality, functionality execution, dependence between functionalities and the role of system users.

However, the focus of the method proposed lies on the analysis of the functional requirements; hence, non-functional requirements are not considered during the checklist application. Yet, as future works, we intend to use usability and accessibility factors.

Also, the method is limited for requiring that only use cases that have their respective prototypes, or vice-versa, may be inspected. Furthermore, only inspecting use cases that have their respective prototype constrains the method, once prototypes are not always produced as a detailing of use cases. For this, as a future work, the suggestion is that the method be able to adapt to situations in which not all the requirement documents are available.

For being a use case model, the method proposed was based on the case model by the "Social eGov" research team, which was used as a case study for this work. Thus, as future work, our intention is to make the method adequate to internationally acknowledged use case templates.

Moreover, this study needs further tests so that both the checklist and the consistency metrics may be validated and proved, thus improving them. The other studies may allow calibrating the consistency metrics QC developed herein, since all the values suggested are considered suggestions that may be adapted at each design, and are referred to in Tables 1 and 3.

Again, the method proposed was applied by a single tester and it is thus necessary for other testers to use the method as a way of verifying its behavior in other designs, along with analyzing testers' understanding in the application of the checklist and of the metrics developed.

The method is still limited to requiring the tester that applies the method to know software engineering; he/she is also recommended to know the design to which the method is applied. Thus, as future works, refining and optimizing the questions of the checklist proposed are also suggested.

Besides the suggestions considered, the following works may evolve from the present study: experiments and comparatives of software inspection of the method proposed with the existing ones; comparison and comparatives of applying the method proposed with methods that separately consider the requirement document and the description of use cases; and, along the software development process, assessing the decrease in implementation errors due to the application of the method.

8. REFERENCES

[1] B. Anda, D. I. K. Sjøberg.: Towards in the Inspection Technique for Use Case Models. In SEKE'02 -14th IEEE Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 15-19, 2002, pp. 127-134.

[2] Aurum, K. Cox, R. Jeffery.: An Experiment in Inspection the Quality of Use Case Descriptions. In Journal or Research and Practice in Information Technology, 2004, Vol. 36, No. 4.

[3] Belgamo, S. Fabbri, J. C. Maldonado.: TUCCA: Improving the Effectiveness of Use Construction and Requirement Analysis. In Empirical Software Engineering, 2005 International Symposium, 2005, vol., no., pp. 10 pp., 17-18.

[4] F. Elberzhager, J. Münch, B. Freimut.: Optimizing Cost and Quality by Integrating Inspection and Test Processes. Waikiki, Honolulu, HI, USA, 2011.

[5] R. Gregolin.: "Uma proposta de inspeção de em modelos de caso de uso." São Paulo, 2007. Dissertação de mestrado. Instituto de Pesquisas Tecnológicas do Estado de São Paulo.

[6] G. Gusev.: Practical Review of Software Requirements. In Software Engineering Conference (CEE-SECR), 2010 6th Central and Eastern European, 2010, pp.185-188, 13-15.

[7] IEEE, IEEE Std 610.12-1990.: IEEE Standard Glossary of Software Engineering Terminology. Corrected Edition, in IEEE Software Engineering Standards Collection, The Institute of Electrical and Electronics Engineers, New York, 1991.

[8] JANI, H. M.: Applying Case-Based Reasoning to Software Requirements Specifications Quality Analysis System. In Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on, 2010, vol., no., pp.140-144, 23-25.

[9] Maciel, C. SLaviero, P. C. Souza, M. A. D. CAMPOS, E. C. SANTANA.: Platform design details to support eparticipation environments deployment. In: Third IFIP WG 8.5 International Conference on eParticipation (ePart 2011), Delf. Electronic Government and Electronic Participation. Osterreich : Trauner Verlag, 2011. v. 37. p. 382-391.

[10] S. M. Melo.: "Inspeção de Software". University of São Paulo: São Carlos, SP, 2009. Available at: <http://moodle.stoa.usp.br/file.php/559/InspecaoArtigo.pdf?forcedownload=1> [Accessed 15 September 2011]

[11] K. Pohl.: Requirement Engineering. Springer – Verlag Berlin Heidelberg, Springer, 2010.

[12] A.A. Porter, L. G. Votta Jr., V. R. Brasili.: Comparing Detection Methods for Software Requirements Inspections: A replicates Experiments. 1995 Available at < http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=391380> [Accessed 09 octover 2011]

[13] SOFTEX, "MPS. BR – Guia de Melhoria de Processo do Software Brasileiro", 2011. Available at: <http://softex.br/mpsbr/_guias/guias/MPS.BR_Guia_de_Impl

ementacao_Parte_4_2011.pdf>, [Accessed 10 November 2011]

[14] Sommerville.: “Engenharia de Software São Paulo”. Pearson Adisson-Wesley. Brazil, 7th edition, 2007.

[15] T. Thelin, P. Runeson, C. Wholin.: An Experimental Comparison of Usage-Based and Checklist Reading. In IEEE Transictions on Sofware Engineering, 2003, vol.29, no 8.

[16] Boehm, B. W.: Verifying and Validating Software Requirements and Design Specifications. Reprinted in Boehm, B. W. (ed.), Software Risk Management, pp. 205 - 218, IEEE Computer Society Press, 1989.

[17] Muriana, L. M. : Um Metódo para Garantia da Qualidade de Software na Fase de Engenharia de Requisitos. Cuiabá, Mato Grosso, 2011. Trabalho de Conclusão de Curso apresentado ao Instituto de Computação da UFMT.