
QSIC
2010
PROCEEDINGS OF THE TENTH
INTERNATIONAL CONFERENCE ON
QUALITY SOFTWARE
ZHANGJIAJIE, CHINA, JULY 14-15, 2010
EDITED BY JI WANG, W. K. CHAN, AND FEI-CHING KUO
TECHNICAL SPONSOR:
• IEEE RELIABILITY SOCIETY
CO-SPONSORS:
• NATIONAL LABORATORY FOR PARALLEL AND DISTRIBUTED PROCESSING, CHINA
• THE UNIVERSITY OF HONG KONG, HONG KONG

2010 10th International Conference on Quality Software

# A Methodology for Continuos Quality Assessment of Software Artefacts

Moisés Rodríguez
Alarcos Quality Center S.L. (AQC)
Ciudad Real, Spain
moises.rodriguez@alarcosqualitycenter.com

Marcela Genero, Damiano Torre, Belen Blasco and Mario Piattini
ALARCOS Research Group, University of Castilla-La Mancha
Ciudad Real, Spain
Belen.Blasco@alu.uclm.es (Marcela.Genero, Mario.Piattini, DamianoCosimo.Torre)@uclm.es

***Abstract*— Although some methodologies for evaluating the quality of software artifacts exist, all of these are isolated proposals, which focus on specific artifacts and apply specific evaluation techniques. There is no generic and flexible methodology that allows quality evaluation of any kind of software artifact, regardless of type, much less a tool that supports this. To tackle that problem in this paper, we propose the CQA Environment, consisting of a methodology (CQA-Meth) and a tool that implements it (CQA-Tool). We began applying this environment in the evaluation of the quality of UML models (use cases, class and statechart diagrams). To do so, we have connected CQA-Tool to the different tools needed to assess the quality of models, which we also built ourselves. CQA-Tool, apart from implementing the methodology, provides the capacity for building a catalogue of evaluation techniques that integrates the evaluation techniques (e.g. metrics, checklists, modeling conventions, guidelines, etc.) which are available for each software artifact. CQA Environment is suitable for use by companies that offer software quality evaluation services, especially for clients who are software development organizations and who are outsourcing software construction. They will obtain an independent quality evaluation of the software products they acquire. Software development organizations that perform their own evaluation will be able to use it as well.**

***Keywords*— software quality; quality evaluation; software models; UML models; methodology, CQA Environment, CQA-Meth, CQA-Tool.**

## I. INTRODUCTION

Software firms are paying more and more attention to the quality of their software products, focusing not only on the quality of the final product but rather considering quality to be an integral part from the beginning of the software development life cycle. This importance is seen in the boom in certifications based on models such as CMMI (Capability Maturity Model Integration) [1], ISO 15504 [2], ISO 90003 [3], etc. These highlight the activities for evaluating the quality of software artifacts within key areas in the maturity of an organization that is developing or maintaining software.

Even though some methodologies and standards which deal with software quality do exist, they are isolated proposals which focus on specific software artifacts and apply specific evaluation techniques. There is no generic and flexible methodology that allows the continuous quality assessment of any kind of software artifacts, nor are there any tools which have that as their goal.

We have analyzed different proposals: EMISQ [7, 8], IEEE 1012-1998. Standard for Software Verification and Validation [4], IEEE 1028-2008. Standard for Software Reviews [5], IEEE Std 1061-1998. Standard for a Software Quality Metrics Methodology [6], CMMI (Capability Maturity Model Integration) [7], ISO/IEC 9126. Software engineering-Product quality [8], ISO/IEC 12207. Systems and software engineering-Software life cycle processes [9], ISO/IEC 14598. Software product evaluation [10], ISO/IEC 15504. Software Engineering-Process assessment [11], ISO/IEC 25000. Software engineering-Software product Quality Requirements and Evaluation (SQuaRE) [12],.

In the quest to make up for the lack we have just outlined, we propose a comprehensive solution, called CQA Environment. This considers some of the advantages of the proposals shown in Table 1, and consists of:

1) A methodology (CQA-Meth) for the continuous quality assessment of software artifacts. The main objective of CQA-Meth is to set out a working framework which would allow us to establish the processes (as defined in section 4 of this paper) needed to carry out the evaluation of any kind of software artifacts, as well as to make communication possible between the client (the one sponsoring the evaluation) and the evaluation team.

1550-6002/10 $26.00 © 2010 IEEE
DOI 10.1109/QSIC.2010.24

2) A set of tools that supports such methodology, which is composed of a vertical tool (CQA-Tool) that supports the methodology, as well as specific horizontal tools for each particular software artifact. Until now, we have had at our disposal three tools for evaluating UML model quality (use cases, clsss and statechart diagrams). Moreover, the CQA-Tool provides the facility for building a catalogue of evaluation techniques that integrates those techniques available for each software artifact (for example metrics, checklists, modelling conventions, guidelines, etc.). In the near future we will modify CQA-Tool to allow existing evaluation tools to be plugged into this generic one.

CQA Environment will open up various possibilities, such as its use by companies which are dedicated to providing quality evaluation services for software factories. It could also be employed by clients who have outsourced the construction of their software systems, thus obtaining an independent quality evaluation of the software products they develop. Software development organizations that carry out their own evaluation will be able to use it as well.

The structure of the paper is as follows. An overview of CQA-Meth is presented in Section 2. Section 3 details the processes of CQA-Meth. An application example of the CQA-Meth and the CQA-Tool is presented in Section 4. Finally, section 5 draws conclusions and outlines future work.

## II. AN OVERVIEW OF CQA-METH

The main characteristics of CQA-Meth can be summarized in the following basic principles:

- It is made up of a structured set of processes
- It is directed at the relationship with the client and at the outsourcing of quality evaluation.
- It is intended to be an easily-adaptable methodology.
- It is supported by a set of techniques and tools.

The CQA-Meth provides a work framework which clearly identifies what?, when? and who? in each of the phases and activities in the processes, as well as the sequence of steps to be followed when the evaluation is being carried out.

Apart from all the above, the methodology is oriented towards the relationship with the client, so that at different points in the evaluation process, the client finds himself /herself involved in the decision-making. In addition, outsourcing is taken on as the modus operandi of this methodology, which means that the evaluation does not have to be carried out within the client's premises. The planning, design and carrying out of the evaluation can be done outside; the client can be contacted whenever it is deemed necessary.

Similarly, the methodology is designed to be adapted to the different needs of the client. To that end, there are catalogues of evaluation techniques which establish the level to which and with what depth we wish to perform the evaluation, as well as setting out the particular quality measurements to be obtained and what specific evaluation tools will be used

CQA-Meth was also implemented with SPEM (Software & System Process Engineering Meta-model) (OMG, 2007) and EPFC (Eclipse Process Framework Composer) (Eclipse, 2004).

Lastly, CQA-Meth is supported by a set of evaluation techniques and support tools, which makes for a swifter and more agile development of the phases of the methodology and allows the evaluation to be carried out semi-automatically.

### A. *Methodology roles*

In the following lines, the roles and groupings of these identified throughout the evaluation process, are set out in detail, along with the main tasks and responsibilities of each of them.

In the figure below, *"Fig. 1"*, we may observe the roles involved in the evaluation process as established in CQA-Meth, together with the relationship that exists between these roles. As can be seen in the figure, the communication between the firms (client and evaluator) takes place through the sponsor and the chief evaluator. These individuals will in turn be those who feed back to the teams within their own respective firms.

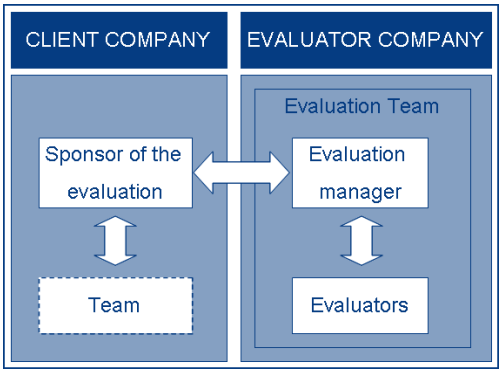


Figure 1. Methodology roles

- Client Company. This is the organization which has expressed its intention and/or need to contract the services of evaluation of the quality of its UML models.
- Evaluation sponsor. This is the representative of the client company who contacts the evaluating firm to express the need for quality evaluation of its UML models (this is normally the project head or the person in charge of the quality department).
- Team. This acts as support to the sponsor of the evaluation. Its main responsibility lies in analyzing the results reported by the evaluation team, aiming to detect defects or inconsistencies. This team is optional and its existence depends on the relative complexity of the project under evaluation (*"Fig. 1"*, broken-lined rectangle). Evaluation company. This is the organization which has been contracted to carry out the evaluation process. The evaluating team will need to have at its disposal a chief evaluator and one or more evaluators, who will make up the work team to perform the evaluation.
- Evaluation manager. This is the representative of the evaluating firm and head of the evaluating team, whose job it is to ensure the proper functioning of the evaluation process.
- Evaluator. He/she is in charge of performing the activities and tasks which are an integral part of the evaluation process. This role should be taken on by a person who has knowledge of UML modeling and of the evaluation methodology.
- Evaluation team. This is the team from the evaluator firm, made up of the evaluation manager and a set of one or various quality evaluators (as the project requirements may demand). Its main goal is to fulfil the evaluation requirements agreed on with the sponsor, within the time limits set for that.

### *B. Catalogue of elements*

All the elements (documents, reports, catalogues, artifacts, etc.) which are identified in the evaluation activities are listed below. These elements may be input or output for the evaluation process activities and some of them may also be end-deliverables for the activities of the evaluation process.

- Contact information. Document where the evaluating firm gathers the contact details of the client firm.
- Evaluation contract. This document is the first deliverable (output document delivered to the client firm) of the evaluation project and it will be generated as a result of phase 1 (planning). It gathers the conditions under which the evaluation project will be carried out.
- Generic quality model. This is a document which belongs to the evaluating firm. It contains the set of characteristics, sub-characteristics and attributes of quality that can be evaluated in each software artefact.
- Specific quality model. This is a document which contains only the set of characteristics, sub-characteristics and attributes of quality that the client firm has selected for the evaluation project.
- Catalogue of evaluation tools. This is a document which belongs to the evaluating firm, allowing the identification of the tools needed, as well as the way to use them, according to the evaluation techniques which are going to be used to evaluate the UML models.
- Evaluation plan. This document is the second deliverable and is generated as a result of phase 1 (planning) of the evaluation process. It is, nonetheless, subject to change during the evaluation process and any delays and re-planning should be registered in this document.
- Set of artifacts to be evaluated. We understand by "artifacts" the UML models which the client firm wishes to validate, as well as the documentation needed for the nature and content of these models to be understood correctly.
- Specification of the evaluation. This element is the third deliverable from the evaluation project and it will be generated as a result of phase 2 (specification) of the evaluation process. It registers the specific quality model which is going to be analyzed, along with what techniques will be used, what measurements are expected to be obtained, what tools are going to be used, etc.
- Evaluation report. This artifact is the fourth deliverable and it will be generated as a result of phase 4 (conclusion). It is made up of a document which sets out: all the results obtained through evaluation techniques, the conclusions obtained from the analysis of these results and a set of recommendations and future actions for improvement of the UML models.
- Request for modification. This artifact is the fifth deliverable from the evaluation process and will be able to be generated as a result of phase 4 (conclusion). It is the document whereby the client firm can set forth its opinions and possible changes with respect to the results presented in the evaluation report.

- Internal evaluation report. Apart from the deliverables themselves and the results of the evaluation, a set of documents related to the process itself is generated. This information consists mainly of reports on the process of management and monitoring, reports on measurements of the evaluation itself (progress, delays, cost/benefits, etc.), reports on changes and adjustments carried out during the evaluation, reports on defects and adaptations undertaken in the evaluation infrastructure, etc.

### C. *Implementation with SPEM and EPFC*

SPEM 2.0 (Software & System Process Engineering Meta-model) (OMG, 2007) is a meta-model of OMG, seen as a *de facto* standard in the industry for the representation of process models of software engineering and system engineering. At the same time, within the open framework of Eclipse, an SPEM 2 editor called EPFC has been developed. It allows us to define, manage and re-use a repository of fragments of method and processes. So, with EPFC, implementations can be created in the SPEM 2 format of any method, process or methodology of software engineering.

## III. PROCESS OF CQA-METH

CQA-Meth is made up of three main processes *"Fig. 2"*.

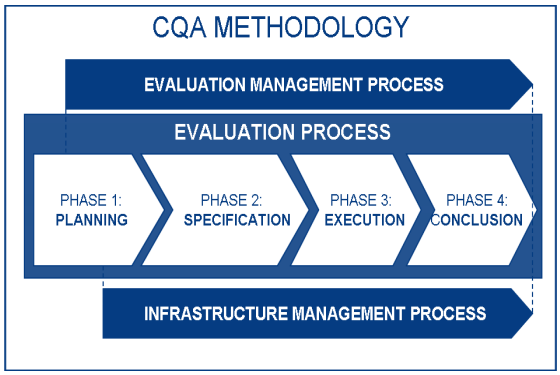


Figure 2. Methodology process and phases

### A. *The evaluation process*

This process is the core structure of the evaluation methodology, since it contains the stages of planning, specification, execution and conclusion of the evaluation.

The assessment process can be broken down into four phases.

**Phase 1.** Planning: The aim of this phase is to produce a contract and to obtain a plan for evaluating the UML models. This phase is made up of four activities: Contracting, Starting-up of project, Detailed planning, Consolidation of the plan.

**Phase 2.** Specification: The goal of this phase is to define the scope of the evaluation, setting out the set of characteristics which are going to be evaluated for each of the UML models (artifacts), the techniques and tools which are going to be used to carry out the evaluation, the levels of evaluation (low, medium, or high) which will be applied to each artifact, along with the list of indicators and metrics that are expected to be obtained. This phase is made up of four activities: Obtaining and analyzing the artifacts to be evaluated, Selecting the quality model and techniques, Internal planning of the evaluation and Verifying the specification.

**Phase 3.** Execution: The objective in this execution phase is to apply the evaluation techniques, as well as to set in motion the tools (using as starting point the artifacts which are to be assessed), thereby obtaining the initial results (lowest level measurement) about the quality of the UML models. Once the initial results have been obtained, their precision needs to be ensured- (there may be defects in these results, such as the so-called "false positives"). A consistent and well-coordinated storage of these must also be carried out, to allow them to be accessed and used on subsequent occasions. This phase consists of three activities: Application of evaluation techniques, Analysis of the execution, Unification of the results.

**Phase 4.** Conclusion: The objective of the conclusion phase is to draw up the evaluation report, together with the presentation of the results, for the sponsor as well as for the rest of those people involved in the client firm. The phase is composed of three activities: Production of the evaluation report, Presentation of the results, Correction of the report and completion of the evaluation.

### B. *The evaluation management process*

This is the process which supports the evaluation process and which lets us control, evaluate and improve the evaluation process itself. This process is made up of just one single phase, which has the same name as the process itself. This phase, in turn, is

divided into three well-defined activities: Monitoring, Documenting, Adjusting and Developing.

### C. *The infrastructure management process*

This is the support process which allows us to manage all that has to do with the infrastructure needed for carrying out the evaluation process (techniques and evaluation tools). This process is made up of one single phase, which is given the same name as the process itself. This phase, in turn, consists of three well-defined activities: Specification of the infrastructure, Maintenance of the infrastructure, Adaptation and transference of the infrastructure.

## IV. AN EXAMPLE OF APPLICATION

We have previously remarked on the fact that we began using CQA Environment to assess the quality of UML models, given the importance that these models have in the development of software. Space restrictions hinder us from describing an example of the application of the whole methodology. To illustrate the use of CQA-Meth by means of the CQA-Tool, we will set out the end result, which is the publication of a final report which we call "Evaluation Report". This contains the following sections, in line with the template given by the methodology itself: Identification data, Evaluation requirements, Evaluation specifications, Evaluation methods and Evaluation results.

We will evaluate the quality of the class diagram shown in *"Fig. 3"*, using the CQA-Tool, which implements the methodology and we will also use the CD-Tool [13], which is the specific tool for class diagrams. The CD-Tool provides the CQA-Tool report with the "Evaluation specifications" and the "Evaluation results" by means of a set of 45 checklists and 70 metrics for class diagrams, which evaluate the syntactic, pragmatic and semantic quality. These three types of quality for class diagrams are taken into account, also bearing in mind the recommendations provided in [14]. In Table 2, part of the section "Evaluation specification" contained within the "Evaluation Report" generated by the CQA-Tool, is displayed.

In Table 2, the first column represents each item number of each checklist. The second column describes the defect that the checklist item detects. The third column indicates the bibliographic reference where the item of the checklist was proposed, as well as the references that show empirical evidence of its usefulness.

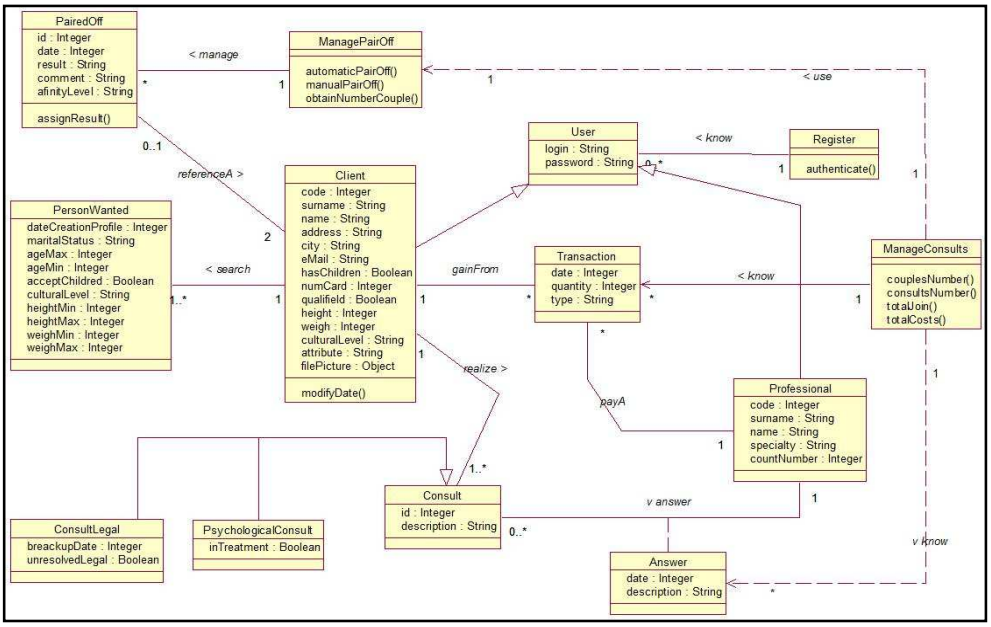


Figure 3. Example of class diagram

TABLE I. EXAMPLE OF SYNTACTIC, SEMANTIC AND PRAGMATIC CHECKLISTS

| Code | Description | A/M | Reference | Result |
|---|---|---|---|---|
| *Syntactic Quality* | | | | |
| I.1 | The classes should have names unique to them. | A | [15] | YES |
| I.2 | Class abstract names should be in italics | M | [16] | NOT APPLICABLE<br>There are no abstract classes in the class diagram |
| I.3 | If more than one word is needed to name a class, these words should be so combined as to make them readable. | M | [16] | YES |
| I.4 | The associations should be specified by means of a straight line. | M | [16] | NO |
| I.5 | Multiplicity should be shown only for associations and aggregations. | M | [16] | NO |
| I.6 | All the names of the classes should begin with capital letters. | A | [16] | YES |
| *Semantic Quality* | | | | |
| I.31 | The class name should use common terminology. | M | [17] | YES |
| I.32 | Class meaning should represent only one logical concept | M | [16] | YES |
| I.33 | The attributes must be nouns based on the domain. | M | [17] | YES |
| I.34 | The meaning of the operation should be reflected in its name and format | M | [16] | YES |
| *Pragmatic Quality* | | | | |
| I.35 | Diagrams should not contain crossed lines. | M | [15] | NO |
| I.36 | The class should not have circular references. | M | [18] | YES |
| I.37 | The package should not have circular dependencies to other packages. | M | [19] | NO APPLICABLE<br>There aren´t packages in these class diagram |
| I.38 | The class should not have more than 60 attributes and operations. | A | [16] | YES |
| I.39 | The number of methods may vary between 10 to 21. | A | [16] | YES |
| I.40 | The depth of inheritance should be not more than three. | M | [16] | YES |

From the complete catalogue of evaluation techniques incorporated within the CD-Tool and from the diagram of *"Fig. 3"*, an evaluation has been conducted, bearing in mind all the automatic checks, as well as the manual ones. The results of these checks are stored in an XML file, subsequently used as an input to the CQA-Tool, which takes on the task of unifying the results with those of other tools, in order to generate the Evaluation Report.

In the case of our example, the results of the evaluation of the class diagram are included in the Evaluation Report, under the section "Evaluation Results", obtaining the information which is set out in *"Fig. 4"*. As may be observed, in the first place a table is presented where the results are summarized, expressed in percentages, for each one of the types of quality (syntactic, semantic and pragmatic) evaluated.

We have added the items of the checklists seen as "NOT APPLICABLE" to those of "YES", because we believe that they are not really defects in the class diagram.

Then graphs are given, the first one being a bar graph where the difference between the number of items on each checklist, evaluated positively and negatively, is shown. The second one is a Kiviat diagram, where the total score of each type of quality is shown for the class diagram evaluated.

Finally, in the report, a summary of the total percentages is set out. In the case of the example,

these results are obtained from the sum of the results of each one of the quality types evaluated. It is also possible to weigh up the partial results according to the client's particular interests or to contrast th em with the results obtained from other metrics or too ls.

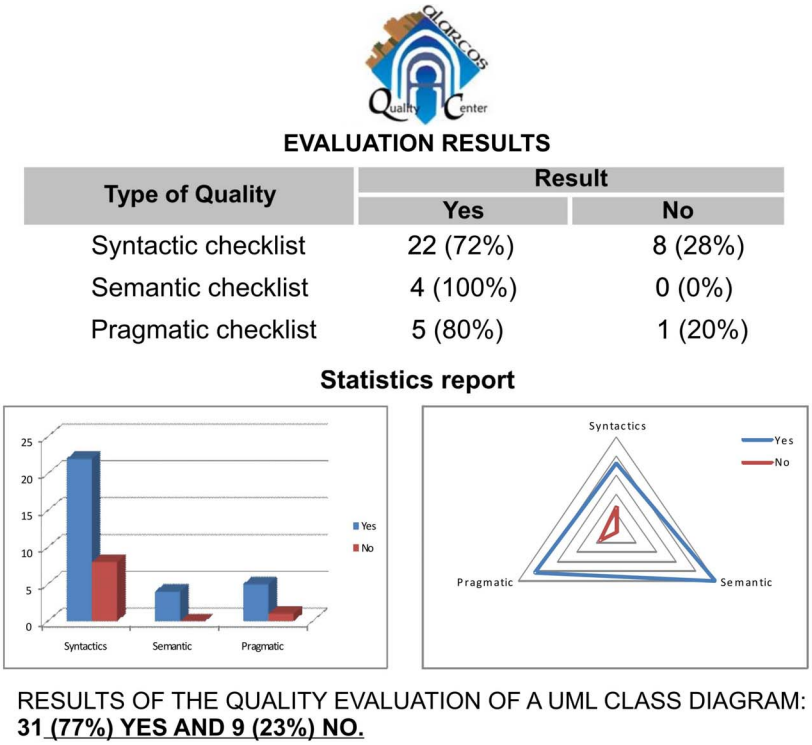


Figure 4. Example of the "Evaluation Results" section contained within the "Evaluation Report"

## V. CONCLUSIONS AND FUTURE WORK

The main contribution of this article is CQA Environment, which consists of a methodological component, CQA-Meth, and a technological component, the CQA-Tool.

The main advantages of CQA Environment are:

- It is not limited to quality evaluation, but also supports the configuration management process and some aspects of process planning. while supporting the infrastructure management too.
- Flexibility in carrying out the evaluations, allowing the users to choose between different quality models, or giving them the possibility of defining their own models. In that way it is possible to obtain evaluation results according to a specific project which focuses on particular quality characteristics.
- It is designed for the evaluation of any type of software artifacts. Until now, we have applied it in the quality assessment of UML models, as these are the first software artifacts to be generated in the development life-cycle. We have also built a catalogue of evaluation techniques and the vertical tools that calculate them. However, it could easily be made applicable to any software artifact, adding new vertical tools for the different types of artifact which we want to assess (source code, test cases, etc.).

In future studies it is hoped to broaden the area of quality evaluation reached with CQA-Meth, to allow us to evaluate the software artifacts generated from the first phases of the life cycle until the delivery and installation of the product (analysis, design, codification and testing) The new environment will thus permit a continuous evaluation of the software product, carrying out evaluation from the stages of analysis and design right up to the code, test cases and final support. It thereby allows us to perceive any abnormality in

the course of the development process, without having to wait for the final tests.

Once we have used the methodology in various different real projects, we will gather feedback on the applicability of the methodology in practice and, if necessary, we will make modifications to fit it to specific requirements Any changes which we may make in the methodology will be supported by changes in the CQA-Tool. In addition, an important improvement will be the adaptation of the CQA-Tool so that it can be connected to existing tools for evaluating the quality of any software artifact.

In the immediate future, we shall be applying the CQA Environment in real projects in software factories installed in the region of Castilla-La Mancha, Spain.

## ACKNOWLEDGEMENTS

This research is part of the following projects financed by the "Ministerio de Ciencia e Innovación": EECCOO (TRA2009_0074), PEGASO (TIN2009-13718-C02-01) and EVVE (PTQ-08-03-07931 co-financed by Fondo Social Europeo), and the following projects financed by "Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha": IDONEO (PAC08-0160-6141) and MECCA (PII2I09-0075-8394).

## REFERENCES

[1] SEI, CMMI for Development Version 1.2, Technical Report, 2006.
[2] ISO/IEC 15504-2: 2003/Cor.1:2004(E). Information technology - Process assessment - Part 2: Performing an assessment in International Organization for Standardization, 2004.
[3] ISO/IEC 90003, Software and Systems Engineering - Guidelines for the Application of ISO/IEC 9001:2000 to Computer Software in International Standards Organization, 2004.
[4] IEEE Std. 1012-1998: Standard for Software Verification and Validation, 1998.
[5] IEEE Std 1028-2008 Standard for Software Reviews and Audits, 2008.
[6] IEEE Std 1061-1998. IEEE Standard for a Software Quality Metrics Methodoloy, 1998.
[7] M.B. Chrissis, M. Konrad, and S. Shrum, "CMMI: Guidelines for Process Integration and Product Improvement", 2nd ed. Addison-Wesley Professional, 2006.
[8] ISO/IEC DTR 9126 Software Product Evaluation - Quality Characteristics and Guidelines for their Use, 2001.
[9] ISO/IEC 12207. International Standard. Information technology - Software life-cycle processes. Amendment 1, 2002.
[10] ISO/IEC 14598: 1999-2001. Information Technology - Software Product Evaluation - Parts 1-6, 1999.
[11] ISO/IEC 15504-5:2003, Information technology - Process assessment - Part 5: An exemplar Process Assessment Model, 2006.
[12] ISO, Software product Quality Requirements and Evaluation (SQuaRE) - Evaluation planning and management in ISO/IEC FDIS 25001 Software and system engineering, vol.2005, 2006.
[13] D. Torre, B. Blasco, M. Genero, and M. Piattini, "CQA-ENV: An Integrated Environment for the Continuous Quality Assessment of Software Artifacts" in SoMeT 2009, vol.199, 2009, pp.148-164.
[14] O. Lindland, G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modelling" in IEEE Software, vol.11, 1994, pp.42-49.
[15] C.F. Lange, Assessing and Improving the Quality of Modeling, Technische, Universiteit Eindhoven, 2007.
[16] B. Unhelkar, "Verification and Validation for Quality of UML 2.0 Models", I. John Wiley & Sons, 2005.
[17] S.W. Ambler, "The Elements of UML™ 2.0 Style, Agile Modeling", Cambridge University Press, 2005, pp. 200.
[18] J. Wüst, "The Software Design Metrics tool for the UML SDMetrics", 2005.
[19] B. Berenbach, "The evaluation of large, complex UML analysis and design models" in ICSE '04: Proceedings of the 26th International Conference on Software Engineering, 2004, pp.232-241.