New Trends in Software Methodologies, Tools and Techniques

Proceedings of the Eighth SoMeT_09

Software is an essential enabler for science and the new economy, but software often falls short of our expectations, remaining expensive and not yet sufficiently reliable for a constantly changing and evolving market.

This publication, which forms part of the SoMeT series, consists of 41 papers, carefully reviewed and revised on the basis of technical soundness, relevance, originality, significance, and clarity. These explore new trends and theories which illuminate the direction of developments which may lead to a transformation of the role of software in tomorrow's global information society. The book offers an opportunity for the software science community to think about where they are today and where they are going.

The emphasis has been placed on human-centric software methodologies, end-user development techniques, and emotional reasoning, for an optimally harmonised performance between the design tool and the user. The handling of cognitive issues in software development and the tools and techniques related to this form part of the contribution to this book. Other comparable theories and practices in software science, including emerging technologies essential for a comprehensive overview of information systems and research projects, are also addressed.

This work represents another milestone in mastering the new challenges of software and its promising technology, and provides the reader with new insights, inspiration and concrete material to further the study of this new technology.

ISBN 978-1-60750-049-0
ISSN 0922-6389

199

NEW TRENDS IN SOFTWARE METHODOLOGIES, TOOLS AND TECHNIQUES

H. Fujita
V. Maríik
(Eds.)

IOS Press

Frontiers in Artificial Intelligence and Applications

NEW TRENDS IN SOFTWARE METHODOLOGIES, TOOLS AND TECHNIQUES

Proceedings of the Eighth SoMeT_09

Edited by
Hamido Fujita
Vladimír Maříik

IOS Press

Frontiers in Artificial Intelligence and Applications

FAIA covers all aspects of theoretical and applied artificial intelligence research in the form of monographs, doctoral dissertations, textbooks, handbooks and proceedings volumes. The FAIA series contains several sub-series, including "Information Modelling and Knowledge Bases" and "Knowledge-Based Intelligent Engineering Systems". It also includes the biennial ECAI, the European Conference on Artificial Intelligence, proceedings volumes, and other ECCAI – the European Coordinating Committee on Artificial Intelligence – sponsored publications. An editorial panel of internationally well-known scholars is appointed to provide a high quality selection.

Volume 199

Recently published in this series

ISSN 0922-6389

New Trends in Software Methodologies, Tools and Techniques

Proceedings of the Eighth SoMeT_09

Edited by

Hamido Fujita

Director of Intelligent Software System Laboratory
Iwate Prefectural University, Iwate, Japan

and

Vladimír Mařík

Czech Technical University, Prague, Czech Republic

IOS Press

Amsterdam • Berlin • Tokyo • Washington, DC

ISBN 978-1-60750-049-0
Library of Congress Control Number: 2009933845

Publisher
IOS Press BV
Nieuwe Hemweg 6B
1013 BG Amsterdam
Netherlands
fax: +31 20 687 0019
e-mail: order@iospress.nl

Distributor in the UK and Ireland
Gazelle Books Services Ltd.
White Cross Mills
Hightown
Lancaster LA1 4XS
United Kingdom
fax: +44 1524 63232
e-mail: sales@gazellebooks.co.uk

Distributor in the USA and Canada
IOS Press, Inc.
4502 Rachael Manor Drive
Fairfax, VA 22032
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

New Trends in Software Methodologies, Tools and Techniques
H. Fujita and V. Mařík (Eds.)
IOS Press, 2009

Preface

Software is the essential enabler for science and the new economy. It creates new markets and new directions for a more reliable, flexible and robust society. It empowers the exploration of our world in ever more depth. However, software often falls short of our expectations. Current software methodologies, tools, and techniques remain expensive and are not yet sufficiently reliable for a constantly changing and evolving market, and many promising approaches have proved to be no more than case-by-case oriented methods.

This book explores new trends and theories which illuminate the direction of developments in this field, developments which we believe will lead to a transformation of the role of software and science integration in tomorrow's global information society. By discussing issues ranging from research practices and techniques and methodologies, to proposing and reporting solutions needed for global world business, it offers an opportunity for the software science community to think about where we are today and where we are going.

The book aims to capture the essence of a new state of the art in software science and its supporting technology, and to identify the challenges that such a technology will have to master. It contains extensively reviewed papers presented at the eighth International Conference on New Trends in software Methodology Tools, and Techniques, (SoMeT_09) held in Prague, Czech Republic, with the collaboration of Czech Technical University, from September 23rd to 25th 2009. (http://www.action-m.com/somet_2009/).

This conference brought together researchers and practitioners to share their original research results and practical development experience in software science and related new technologies.

This volume participates in the conference and the SoMeT series[1], of which it forms a part, by providing an opportunity for exchanging ideas and experiences in the field of software technology; opening up new avenues for software development,

[1] Previous related events that contributed to this publication are: **SoMeT_02** (the Sorbonne, Paris, October 3rd to 5th 2002); **SoMeT_03** (Stockholm, Sweden); **SoMeT_04** (Leipzig, Germany); **SoMeT_05** (Tokyo, Japan); **SoMeT_06** (Quebec, Canada); **SoMeT_07** (Rome, Italy); **SoMeT_08** (Sharjah, UAE) and **SoMeT_09** (Prague, Czech Republic). This series of conferences will be continued with **SoMeT 10** in Japan from September 29th to October 1st 2010.

methodologies, tools, and techniques, especially with regard to software security and programme coding diagnosis and aspects of related software maintenance techniques. The emphasis has been placed on human-centric software methodologies, end-user development techniques, and emotional reasoning, for an optimally harmonised performance between the design tool and the user.

This book, and the series it forms part of, will continue to contribute to and elaborate on new trends and related academic research studies and developments in SoMeT_2010 in Japan.

A major goal of this work was to assemble the work of scholars from the international research community to discuss and share research experiences of new software methodologies and techniques. One of the important issues addressed is the handling of cognitive issues in software development to adapt it to the user's mental state. Tools and techniques related to this aspect form part of the contribution to this book. Another subject raised at the conference was intelligent software design in software security and programme conversions. The book also investigates other comparable theories and practices in software science, including emerging technologies, from their computational foundations in terms of models, methodologies, and tools. This is essential for a comprehensive overview of information systems and research projects, and to assess their practical impact on real-world software problems. This represents another milestone in mastering the new challenges of software and its promising technology, addressed by the SoMeT conferences, and provides the reader with new insights, inspiration and concrete material to further the study of this new technology.

The book is a collection of 41 carefully refereed papers selected by the reviewing committee and covering:

- Software engineering aspects of software security programmes, diagnosis and maintenance
- Static and dynamic analysis of software performance models
- Software security aspects and networking
- Practical artefacts of software security, software validation and diagnosis
- Software optimization and formal methods
- Requirement engineering and requirement elicitation
- Software methodologies and related techniques
- Automatic software generation, re-coding and legacy systems
- Software quality and process assessment
- Intelligent software systems and evolution
- End-user requirement engineering, programming environment for Web applications
- Ontology and philosophical aspects on software engineering
- Cognitive Software and human behavioural analysis in software design.

All papers published in this book have been carefully reviewed, on the basis of technical soundness, relevance, originality, significance, and clarity, by up to four reviewers. They were then revised on the basis of the review reports before being selected by the SoMeT international reviewing committee.

This book is the result of a collective effort from many industrial partners and colleagues throughout the world. We would like to thank Iwate Prefectural University, in particular the President, Prof. Yoshihisa Nakamura; SANGIKYO Co., especially the president Mr. M. Sengoku; the Czech Technical University of Prague (Czech Republic); ARISES of Iwate Prefectural University and all the others who have contributed their invaluable support to this work. Most especially, we thank the reviewing committee and all those who participated in the rigorous reviewing process and the lively discussion and evaluation meetings which led to the selected papers which appear in this book. Last and not least, we would also like to thank the Microsoft Conference Management Tool team for their expert guidance on the use of the Microsoft CMT System as a conference-support tool during all the phases of SoMeT.

The Editors

SoMeT 2009 Organization

Program Chairs

Hamido Fujita, *Iwate Prefectural University, Iwate Japan*
e-mail: issam@soft.iwate-pu.ac.jp

Vladimír Mařík, *Czech Technical University, Prague, Czech Republic*
e-mail: marik@labe.felk.cvut.cz

Program committee of SoMeT_09

http://www.action-m.com/somet2009/

Anna-Maria Di Sciullo	University de Quebec de Montreal, Canada
Kamel Adi	University of Quebec at Outaouais, Quebec, Canada
Luigi Logrippo	University of Quebec at Outaouais, Quebec, Canada
Mourad Debbabi	Concordia University, Montreal, Canada
Bechir Ktari	University of Laval, Quebec, Canada
Mohamed Mejri	Laval University, Quebec, Canada
Bipin Indurkhya	IIIT, Hyderabad, India
Armin Eberlein	American University of Sharjah, UAE
Imran A. Zualkernan	American University of Sharjah, UAE
Marite Kirikova	Riga Technical University, Latvia
Rimantas Butleris	Kaunas University of Technology, Lithuania
Remigijus Gustas	Karlstad University, Sweden
Paul Johannesson	Royal Institute of Technology, Sweden
Love Ekenberg	Stockholm University, Sweden
Rudolf Keller	PMOD, Zurich, Switzerland
Gregers Koch	University of Copenhagen, Denmark
Soundar Kumara	The Pennsylvania State University, USA
Margeret M Burnett	Oregon State University, Corvallis, USA
Gregg Rothermel	University of Nebraska, Lincoln, USA
Kenneth Baclawski	Northeastern University, Boston, USA
Michael Oudshoorn	Montana State University, Bozeman, USA
Saeed K. Rahimi	University of St. Thomas, Minnesota, USA
Ernest Edmonds	University of Technology, Sydney, Australia
Colette Rolland	University de Paris_1-Pantheon Sorbonne, Paris, France
Pierre-Jean Charrel	University of Touluse_2, Toulouse, France
Victor Malyshkin	Russian Academy of Sciences, Russia
Yury Zagorulko	A.P. Ershov Institute of Informatics System, Russia
Khaled Ghedira	Institute Superior of Informatique, Tunisia

Rafa Al-Qutaish	Al-Zaytoonah University of Jordan, Jordan
Moacyr Martucci Jr.	University of Sao Paulo, Sao Paulo, Brazil
Sergei Gorlatch	University of Muenster, Germany
Volker Gruhn	Leipzig University, Germany
Clemens Schaefer	Leipzig University, Germany
Heinrich Herre	Leipzig University, Germany
Elke Pulvermueller	University of Osnabrueck, Germany
Domenico Pisanelli	ISTC-CNR, Rome, Italy
Claudio De Lazzari	IFC-CNR, Rome, Italy
Zdeněk Kouba	Czech Technical University, Prague, Czech Republic
Roman Bartak	Charles University, Prague, Czech Republic
Milan Vlach	Charles University, Prague, Czech Republic
Zuzana Sochova	CertiCon, Prague, Czech Republic
Michal Pěchouček	Czech Technical University, Prague, Czech Republic
A min Tjoa	Vienna University of Technology, Vienna, Austria
Luca Dan Serbanati	Politehnica University, Bucharest, Romania
Tu Bao Ho	JAIST, Japan
Nikolay Mirenkov	Aizu University, Fukushima, Japan
Yasuaki Nishitani	Iwate University, Japan
Jun Sasaki	Iwate Prefectural University, Japan
Masaki Kurematu	Iwate Prefectural University, Japan
Jun Hakura	Iwate Prefectural University, Japan

Organizing Chairs

Barbora Jeník	Czech Technical University, Prague, Czech Republic
Pavla Kozáková	Action M Agency, Prague, Czech Republic
Milena Zeithamlová	Action M Agency, Prague, Czech Republic

Contents

Chapter 4. Software Securities and Program Correctness

Chapter 5. Software Development and Related Verification

Chapter 6. Software Methodologies Related Techniques

Chapter 7. Ontological Views on Software Developments

Chapter 8. Software Visualization Related Developments

Chapter 9. Intelligent User Interaction and Software

Chapter 10. Natural Language Related Software Development

Chapter 11. Service Development Systems Applications

Chapter 1

Requirement Engineering and Methods Engineering

New Trends in Software Methodologies, Tools and Techniques
H. Fujita and V. Mařík (Eds.)
IOS Press, 2009

doi:10.3233/978-1-60750-049-0-148

CQA-ENV: An Integrated Environment for the Continuous Quality Assessment of Software Artifacts

Damiano Torre[a], Belen Blasco[b], Marcela Genero[b] and Mario Piattini[b]

[a]*Department of Informatics*
University of Bari, Bari, Italy
DamianoTorre@gmail.com

[b]*ALARCOS Research Group*
Department of Technologies and Information Systems
University of Castilla-La Mancha, Ciudad Real, Spain
Belen.Blasco@alu.uclm.es
{Marcela.Genero, Mario.Piattini}@uclm.es

Abstract. At present, the quality of software artefacts is an increasing concern for software development organizations. It is widely acknowledged that the quality of the software product that is finally implemented is influenced to an enormous extent by the quality of software artefacts (commonly models) that are produced throughout the software development process. Quality assessment and assurance techniques must therefore be applied from the early development stages onwards. The quality of the models is gaining even more relevance with the appearance of the Model Driven Development Model paradigm, which consists in the production of software as successive transformations of models. Although some methodologies for evaluating the quality of software artefacts do exist, all of them are isolated proposals, which focus on specific artefacts and apply specific assessment techniques. There is no generic and flexible methodology that allows the quality assessment of any kind of software artefact, regardless of type, much less a tool that supports it. When tackling this problem in this paper, we propose an integrated environment called "CQA-ENV", consisting of a) Methodology for the continuous quality assessment of software artefacts, based on the ISO 14598 standard and other relevant proposals, 2) A set of tools that supports such methodology, which is composed of a vertical tool (CQA-Tool) that supports the methodology, along with several specific tools for the assessment of the different software artefacts.. Current evaluation tools will also be able to be plugged into this generic tool. Moreover, the CQA-Tool provides a capacity for building a catalogue of assessment techniques that integrates available assessment techniques (e.g. metrics, checklists, modelling conventions, guidelines, etc.) for each software artefact. CQA-ENV can also be used by companies that offer software quality assessment services, especially for clients who are software development organisations, outsourcing software construction, thus obtaining an independent quality evaluation of the software products they acquire. Software development organisations that perform their own evaluation will be able to use it as well.

Keywords. Quality, Assessment, Methodology, Software Artifacts, UML models ISO/IEC 14598, Tool

1. Introduction

The continuous assessment of software artefact quality from the initial phases of the lifecycle is taking on an ever-increasing amount of importance owing to:

1) The growth that is coming about due to outsourcing of software, a market whose turnover is expected to reach some 1.3 billion dollars in 2009 [1]. This means that, on the one hand, the organizations which work in "factory mode" must invest resources to "assure" the quality of the software they manufacture, while on the other hand, the clients have to "control" the quality of software produced by the factories.
2) The increasing importance of the quality of computer systems, which is something that society is demanding with greater and greater emphasis, a fact that is explained by the way companies are ever-increasingly dependent on information systems for their survival. This importance is seen in the boom in certifications based on models such as CMMI (Capability Maturity Model Integration) [2], ISO 15504 [3], ISO 90003 [4], 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.
3) The appearance of model driven engineering (MDE), also known as Model-Driven Development (MDD),which has contributed considerably to shifting the focus of the quality assessment activities from the code towards the specifications of the software systems, more specifically to the software models. In particular, the Model-Driven Architecture proposal (MDA) of the OMG encourages the use of models throughout the whole development process and allows these to be transformed until the code of the final software product has been obtained. That being so, in this approach the models are no longer mere ways to describe software, but are rather the cornerstone of its development. The quality of the models is thus of great importance, since it will determine the quality of the software products that are implemented in the end. In this approach, one of the most important languages is the Unified Modeling Language (UML), which is rapidly gaining broad acceptance in the industry as "THE" standard modeling language for expressing software models.

Even though there exist some methodologies for evaluating software artefact quality, all of these are isolated proposals which focus on specific artefacts and apply specific assessment techniques. There is no generic and flexible methodology that allows the quality assessment of any kind of software artefact, regardless of type, never mind a tool that supports it.

To address this problem, the main goal of this paper is to propose an integrated environment called "CQA-ENV", consisting of:

1) A methodology (CQA-Meth) for the continuous quality assessment of software artefacts, based on the ISO 14598 standard [5], and the new series of standards ISO 25000 [6] (especially ISO 25040), and with ideas from other relevant proposals [7], [8], [9] and [10].
2) A set of tools that supports such methodology, which is composed of a vertical tool (CQA-Tool) that supports the methodology, together with several specific tools for the assessment of the different software artefacts. Existing evaluation tools will also be able to be plugged into this generic tool. Moreover, the CQA-Tool provides the facility of building a catalogue of assessment techniques that integrates

available assessment techniques for each software artefact (for example metrics, checklists, modelling conventions, guidelines, etc.).

CQA-ENV will open up the possibility of its being used by companies which are dedicated to providing quality assessment services for software factories, as well as 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 organisations that carry out their own evaluation will be able to use it as well.

The structure of the paper is as follows. In section 2 we explore related work. A brief description of CQA-Meth can be found in section 3. An overview of CQA-ENV is presented in section 4. In addition, section 4 describes the CQA-Tool (the tool which implements the methodology) and the set of UML-Tools the tools that evaluates the quality of UML models). Section 5 presents the CQA-Catalogue, which stores techniques for assessing the quality of any kind of software artifacts, regardless of type, and provides an example of the content of catalogue for UML class diagrams. Finally, section 6 draws the conclusions and outlines the future work.

2. Related work

We will consider related work, both in terms of already-existing methodologies for the evaluation of the quality of software artifacts and also in terms of the tools that automate the assessment of the quality of software artifacts.

2.1. Methodologies for quality assessment of software products

Some approaches for supporting the evaluation of software products do already exist. One of the most important is the ISO/IEC 14598 standard. It provides a general process for the evaluation of the quality of software products. But this standard is very general and does not provide enough details for it to be readily applied [1],[11] think that the standard does not support goal formulation, nor does it indicate how to adapt a quality model. They propose their own method, called EMISQ (Evaluation Method for Internal Software Quality). They have also developed a tool that supports the method, SPQR. But the method and the tool focus on code evaluation.

The Franhoufer IESE has made several proposals about quality assurance strategies. First they defined different factors to take into account to define a quality assurance strategy [7]. Later, they proposed a conceptual process with the basic steps to define a quality assurance strategy [8]. This strategy can be applied to any software artifact. The problem is that it can only be used in software development companies, not in companies which evaluate software quality as part of an outsourcing service. Moreover, such proposals have not been implemented in any tool yet.

PSM [12] is a process for designing and implementing project-based software measurement programs. It provides essential information about planning, resources assignment, and technological performance. PSM includes an activity which is to evaluate the measurement process itself. This idea may turn out to be of great interest, since it would allow us to detect problems in the evaluation process itself and thus permit us to carry out the most fitting improvements to correct it.

2.2. Tools for the quality assessment of software artefacts

There are a large number of tools which permit the assessment of software artefact quality, but each of them focuses on specific artefacts. There exists, for instance, a great variety of tools for working out metrics for the code on OO languages, amongst which we could mention CKJM, PMD/CPD, Flawfinder, CheckStyle FindBugs, JDepend, Splint, McCabe Iq, JavaNCSS, Logiscpe, Fortify SCA, Together, Eclipse, etc. A more thorough and in-depth listing of these tools can be found in [13].

As regards the tools for the assessment of models (which calculate metrics or which check control lists or style), these tend to be either autonomous tools, or modeling tool plug-ins. The assessment tools read a representation of the model and give a report with the results of the evaluation. Some examples of these assessment tools are DesignAdvisor [14], SQUAM [15], SDMetrics [16], Metricview [17], UMP [18], Fast&Furious [19], UMM [20], MUM [21] and GenMETRIC [22], amongst others.

Though it is indeed the case that a great number of tools already exist, from our point of view many of these tools are restricted to specific domains, measurement standards or evaluation process or quality models, which may reduce their generality and scope. Each tool works according to its own philosophy, collects its own data and calculates specific measures. Currently, companies have to measure highly heterogeneous software artefacts whose results have to be stored and processed from a common repository to facilitate the making of decisions, as they cannot depend only on specific software artefacts. None of the tools implements a methodology that allows a complete quality assessment of any software project.

2.3. Conclusions of the related work

We can conclude that there is no methodology that defines a process for evaluating the quality of any kind of software artefacts, regardless of type, with sufficient precision and completeness and which integrates different techniques for evaluating software artefact quality (e.g. metrics, checklists, modelling conventions, etc. There is, hence, no tool that supports the ongoing quality assessment of software products automatically, from the initial phases of the software development life-cycle onwards.

To provide a generic support for software measurement that is not restricted to only one kind of software artefact to be measured, or to any single quality or evaluation model, offering the basis for a common quality assessment process in companies, we have developed a generic approach which is described in the following sections.

To deal with all we have just outlined and considering ideas from the different proposals presented above, we will propose:

1) A methodology with the following characteristics:
   - It pprovides a thoroughly detailed generic process, describing the activities, tasks, deliverables and roles that will be able to be used for the quality assessment of any kind of software artifacts.
   - It integrates different quality models and techniques for the ongoing quality assessment of any kind of software artifacts.
   - As each client may have different requirements and priorities with respect to the quality attributes to be evaluated, the proposed setting should allow us to

carry out quality assessment of software artefacts at various levels, for example: low, medium, high.

- It contains an activity for the assessment of the evaluation process itself, one which will allow there to be an ongoing improvement in this, thus obtaining feedback from the people who use it.

2) An integrated environment "CQA-ENV", which consists of a set of tools that supports the methodology, which is composed of a vertical tool (CQA-Tool) that supports the methodology, together with several specific tools for the assessment of the different software artefacts. Existing evaluation tools will also be able to be plugged into this generic tool. Moreover, the CQA-Tool provides the facility of building a catalogue of assessment techniques that integrates available assessment techniques (e.g. metrics, checklists, modelling conventions, guidelines, etc.) for each software artefact .

3. Description of CQA-Meth

In this section we briefly introduce the main activities of the proposed methodology, which was built taking into account ideas from, mainly, [5], [12], [16] and [7]. CQA-Meth consists of six activities (see Figure 1). The first five activities are based on [5], and we have added a new activity that allows the evaluation of the proper evaluation process.

Next we describe the five activities of CQA-Meth:

- **Establish the evaluation requirements**: the goal is to reach an agreement through a contract for satisfying the client's needs related to evaluation requirements. The outcome of this activity is the contract signed by both parties, who are the client and those responsible for the evaluation.
- **Specify the evaluation:** once the contract is signed, the client project begins. The purpose of this activity is to define the artifacts to be evaluated in a concrete way artifacts and to specify which quality characteristics the client requires to be evaluated. In order to tailor the methodology to different clients´ needs, the methodology allows the definition of different levels of evaluation, according the degree of the checks to be done (low, high, medium). To this end, we provide the client with a catalogue that describes, for each type of artifact, the quality characteristics that we provide support for in evaluation, along with the corresponding techniques proposed for such evaluation. The result of this activity is a document with the specification of the evaluation.
- **Design the evaluation:** produces an evaluation plan based on the specification of the evaluation. Here it is necessary to consider the different artifacts to evaluate, the evaluation techniques to apply, the level of evaluation, etc. In this activity we can consider the influence factors and the variation factors proposed by Denger and Elberzhager [7]. In addition, an estimate of the resources needed to carry out the evaluation will be provided. Estimations will be based on the type of artifact, its size, complexity, level of evaluation, percentage of automatic evaluation, etc.
- **Execute the evaluation:** the goal is to obtain the results derived from the application of the evaluation techniques to the different artifacts specified in the evaluation plan, applying the appropriate evaluation tools. These results are

reported in a document we called "draft of the evaluation report". In addition, data about the execution of the evaluation such as, time, effort, etc.. must be recorded.

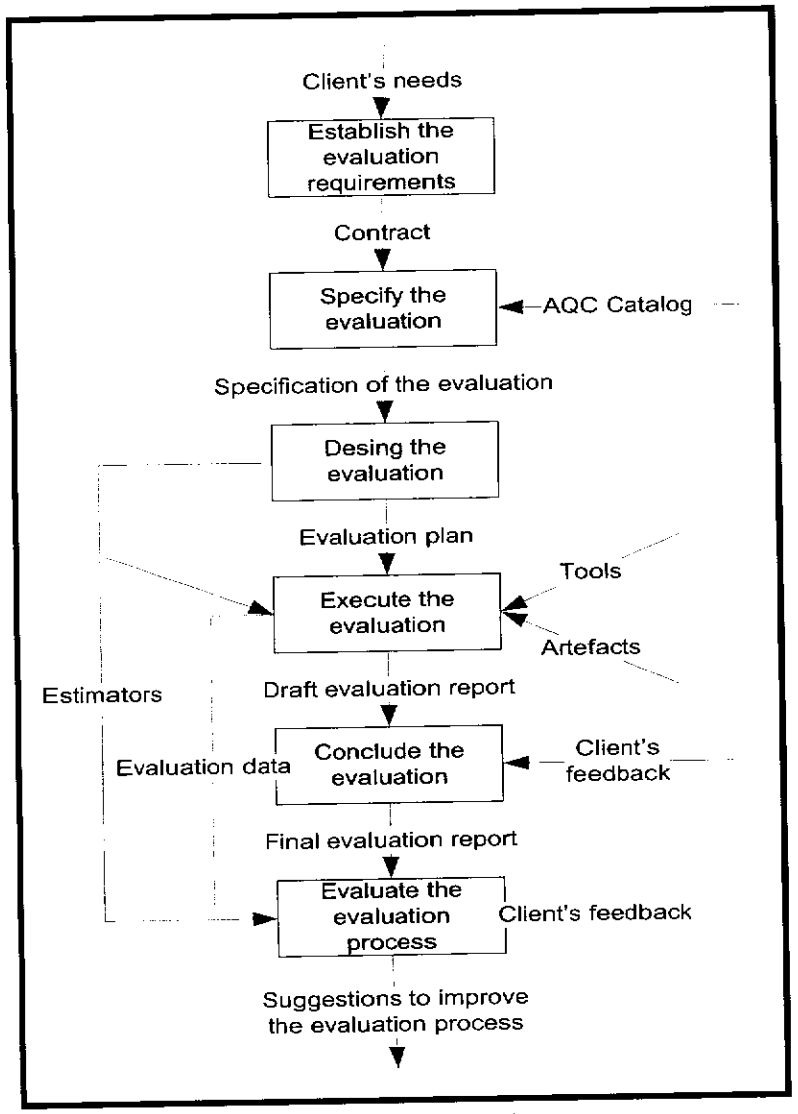

Figure 1. CQA-Meth overview

- **Conclude the evaluation:** the draft evaluation report is analyzed by the person responsible for the evaluation project and the client. Based on the feedback provided by both parts, the draft is updated and the final evaluation report is obtained.
- **Evaluate the evaluation process:** evaluation data (time, effort, etc.) obtained during the execution of the evaluation are analyzed and compared with estimates and client's feedback, thus obtaining the document with suggestions for improving the evaluation process and the estimation techniques.

4. CQA-ENV description

CQA-ENV (see Figure 2) consists of two main kinds of tools:

1) A horizontal tool such as CQA-Tool, that supports the whole process proposed within the methodology CQA-Meth. Moreover, the CQA-Tool provides the facility of building a catalogue of assessment techniques that integrate available assessment techniques for each software artifact (for example metrics, checklists, modelling conventions, guidelines, etc.).
2) A set of several vertical specific tools for the assessment of the different software artifacts. In Figure 2 three tools we used for building UML models are represented, which are CD-Tool, UCD-Tool, STD-Tool for class, use cases and statechart diagrams, respectively.

The set of tools of CQA-ENV are implemented in JAVA to be run under WINDOWS XP or advanced versions.

CQA-Tool will be plugged in to the each of the vertical tools for assessing different kind of software artifacts. Existing evaluation tools will also be able to be plugged into this generic tool by means of a specific wrapper (represented as other tools in Figure 2).

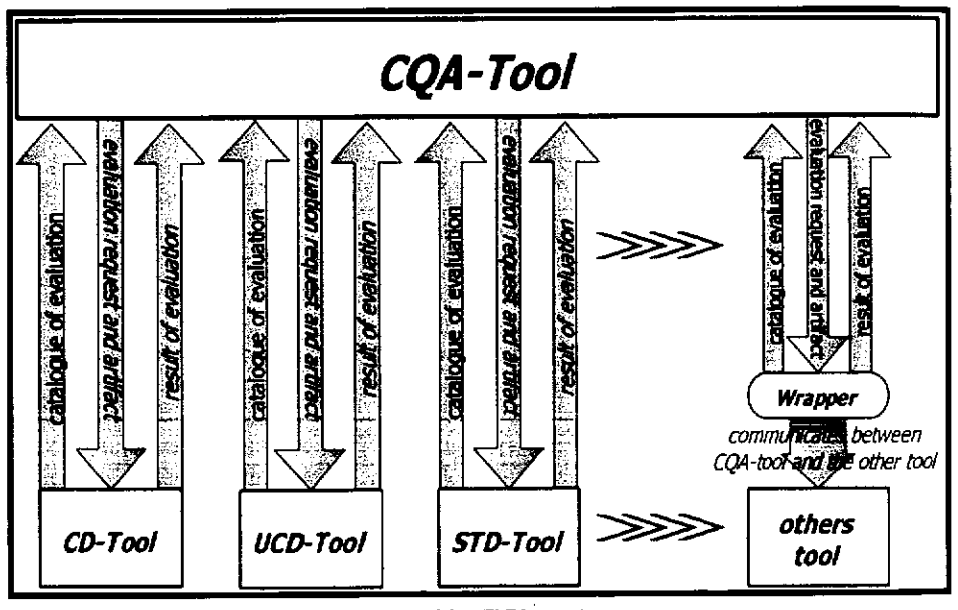

Figure 2. CQA-ENV overview

4.1. Description of CQA-Tool

As we commented before, the main goal of CQA-Tool is to support all the activities proposed in the process proposed in the methodology CQA-Meth. The database conceptual model used by CQA-Tool is illustrated in Figure 3.

Hereafter we will summarise the main business rules supported by the model shown in Figure 3:

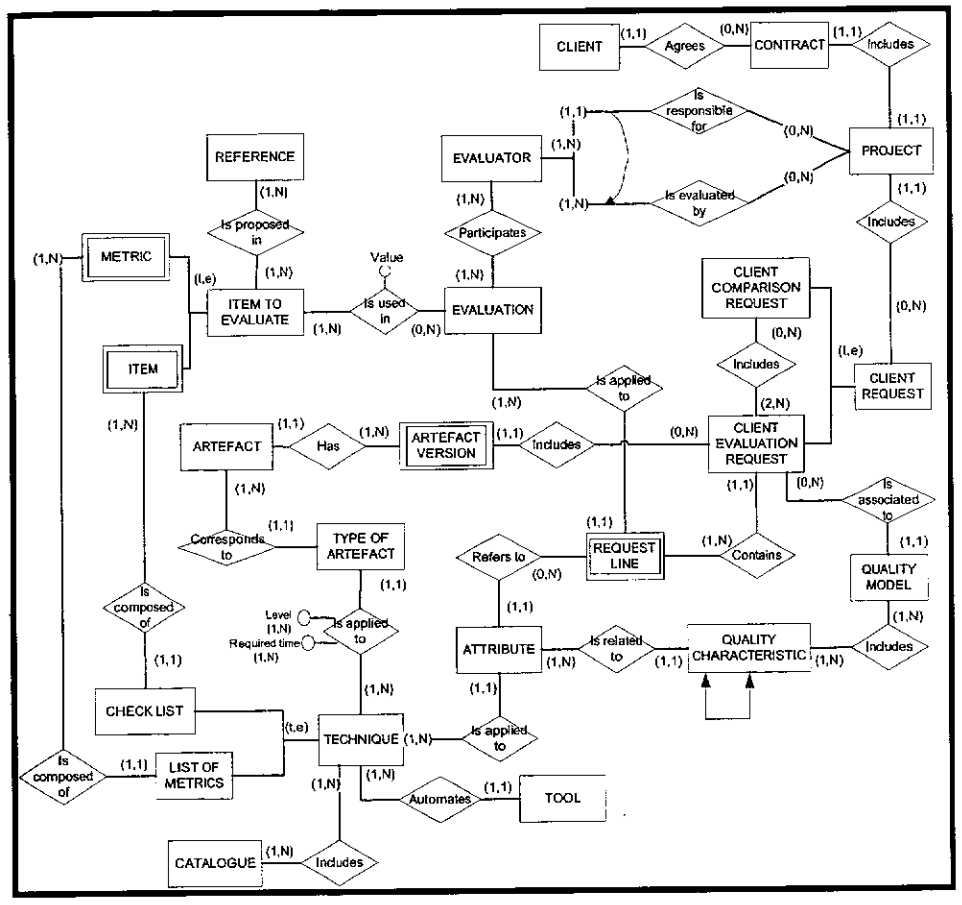

Figure 3. Database Conceptual model of the CQA-Tool

- It should be possible to register the clients with whom a contract will be signed for the carrying out of a quality evaluation service on their projects.
- Each project will include various requests for evaluation and/or comparison. Each request will have a version of a software artifact associated with it (UML class diagrams, Use case diagrams, code, etc.) and it will have a quality model associated with it, which will consist of a series of quality characteristics and sub-characteristics, to which a series of quality attributes will correspond.
- For a single software artifact in a client's request, different quality attributes will be assessed.
- It is necessary to store the information about those clients who have agreed on a contract with the evaluator company. A project is defined for each contract, and each project has an evaluation team. One of the evaluators is the person responsible for the team. For each project several client requests are received,

which can be evaluation or comparison requests. A comparison request corresponds to at least two evaluation requests.

- In each request the level of evaluation asked for by the client will be included (low, medium, high).
- The evaluation techniques should also be registered. Each technique corresponds to a type of software artifact, for a specific quality attribute.
- The techniques may be either a list of metrics, or checklists consisting of a series of verification items. Included in the checklists are guidelines, heuristics, modeling conventions, etc. Information is stored about the references in the literature from which each one of the metrics or items corresponding to each evaluation technique has been taken. References which offer empirical evidence of the use of such techniques are similarly saved.
- The set of assessment techniques available for the different software artifacts which permit the assessment of the CQA-ENV environment will make up what we have called the Evaluation Catalogue.
- Each request will have a person/persons responsible for it, as well as an assessment team.
- The CQA-Tool should also manage the assessment tools, that is, for each evaluation technique the tools available for its evaluation should be registered.

We will now list the exchange of information between the CQA-tool and the tools for UML diagrams which, to simplify matters, have been given the generic term "UML-tools" , (in Figure 2: CD-Tool, UCD-Tool and STD-Tool) and "Other tools" , which could be added with a wrapper in the future. We will highlight two types of communication: 1) that between CQA-Tool and UML-Tools and 2) the one between CQA-Tool and Other tools. We will call "standard-communication" the communication between the CQA-Tool and the UML-tool and we will give the name "other-communication" to that between CQA-Tool and new tools called "Other-tools". We make special mention of this type of communication, because the UML-Tool is built by us within the CQA-ENV, and the Other tools are existing tools like measurement tools.

The "standard-communication" consists of:

- The UML-Tool sends the CQA-Tool the catalogue called "XML-Catalogue"
- The CQA-Tool receives the "XML-Catalogue" from the UML-Tool.
- The CQA-Tool makes a request for evaluation to the UML-Tool, sending it a file called "XML-Request" and the software artifact to be evaluated.
- The UML-Tool receives the request and the software artifact from CQA-Tool and carries out the evaluation.
- After finishing the requested evaluation, the UML-Tool sends a file called "XML-Result" to CQA-Tool.
- The CQA-Tool receives the "XML-Result" from UML-Tool and provides facilities for visualizing and reporting the results.

In order to make an "other-communication" between the CQA-tool and Other-tools that "*don't speak the same language*", a wrapper will manage the communication between both tools (Figure 2).

4.2. Description of the UML-Tools

We tried to build each tool with the highest level of generality possible to make its portability easier, allowing us to manage nearly all types of standard files of various UML editors.

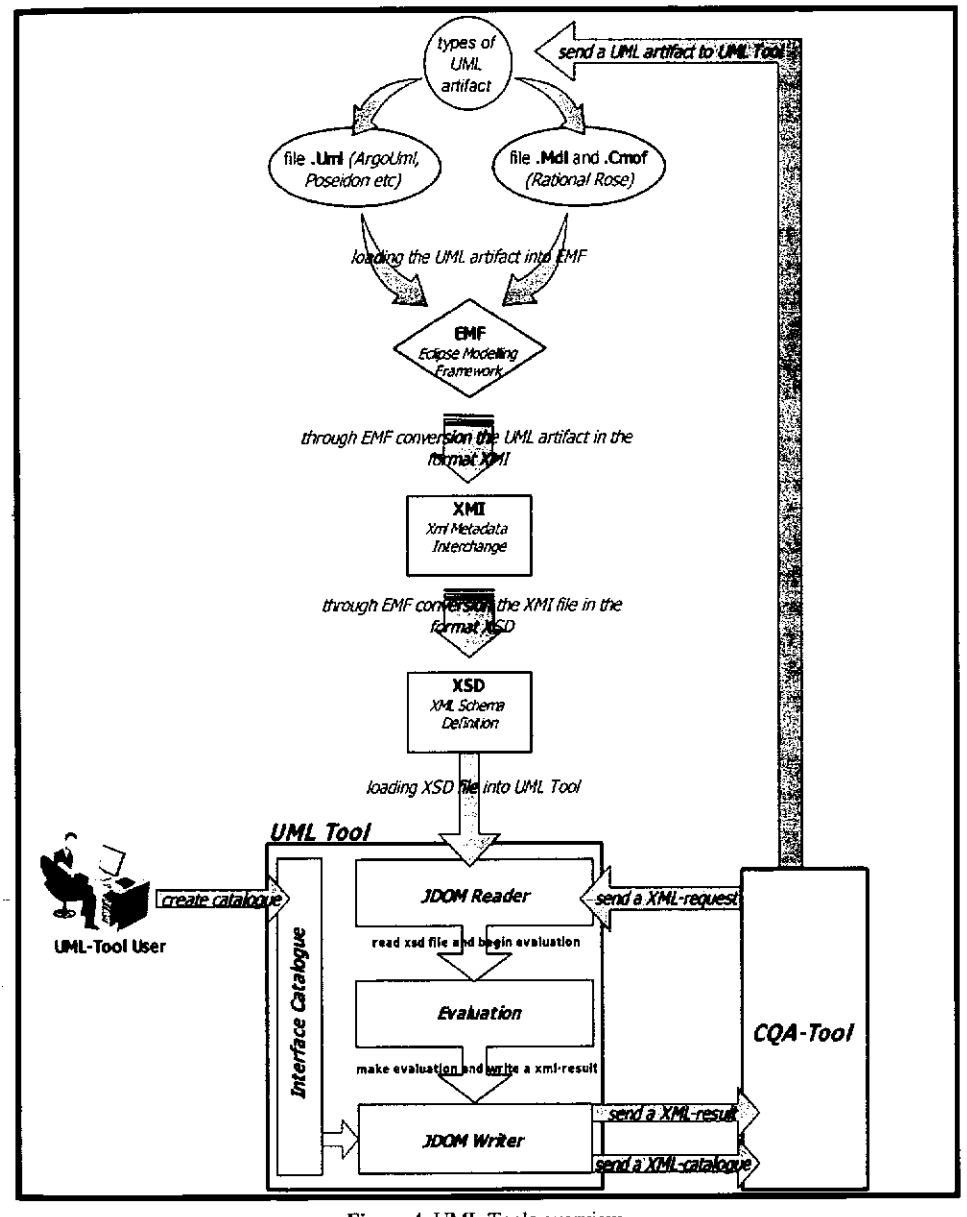

Figure 4. UML-Tools overview

In the lines below, we will illustrate in detail the operation of a generic UML-Tool for UML diagrams that communicates with the CQA-tool (Figure 4).

The UML-Tool user creates a catalogue, inserting the relative data to the catalogue through an appropriate graphical interface. The system produces the previously inserted data and through JDOM libraries creates an XML file called "XML-catalogue" that is then sent to CQA-Tool.

The CQA-tool reads the catalogue and chooses the type of evaluation that it wants. Having chosen the evaluation, the CQA-tool sends the UML-tool a XML request file called "Xml-petition" (where it illustrates in detail it what wants to estimate, based on the previous catalogue) and the UML artefact to be evaluated.

For the reading of the UML artefact from the UML-Tool we analyzed various tools on the market, trying to select between proprietary software and non-proprietary software.

As editor UML the following proprietary ones were considered: Rational Rose[23] and Poseidon[24] ,while as non-proprietary ArgoUML[25] and Acceleo[26] were considered. All these editors support the export of the file to standard format XMI (XML Metadata Interchange), but our tool is also structured to process files in MDL and CMOF (Rational Rose) and UML (Poseidon, ArgoUML), convertible later into XMI format through the EMF [27]. The files produced with the Acceleo editor are convertible into XMI with EMF, because Acceleo is natively integrated with Eclipse and EMF.

The XMI file obtained performs the second conversion into XSD (XML Schema Definition) format, which is more congenial to management of UML diagrams [28]. In this way, UML-tool has a XML-Request ready and an XSD file to be managed.

The UML-tool, through JDOM libraries, reads a XSD file and catches only relevant data, indicated in the XML-Request.

At the same time as reading the XSD artefact and XML-Request files, the system completes the requested evaluation, and the data collected from the evaluation is stored, through JDOM libraries (writer), into an XML result file called "XML-result" with the results of the evaluation.

This tool provides the evaluation of the quality attributes for UML diagrams, through metrics and checklists. Both kind of techniques are further divided into degrees of depth, depending on the level of the evaluation requested, as specified in the XML-Request.

Finally, the UML-Tool, carries out the evaluation and sends an XML-Result file to the CQA-Tool, to communicate the results obtained from the evaluation requested through the XML-Request.

5. CQA-Catalogue

We propose the building of a Catalogue, which acts as a repository to integrate existing evaluation techniques for different software artefacts. We show part of the Catalogue for evaluating the quality of UML class diagrams below . To build this catalogue for each software artefact, the already-existing techniques need to be gathered together, by means of a thorough review of the relevant literature. Similarly, the papers that show empirical evidence of the usefulness in practice of the techniques stored in the catalogue ought to be taken well into account.

5.1. Example of a Catalogue for UML Class diagrams

We will consider two types of techniques: metrics and checklists (including checklists, modelling conventions, heuristics, guidelines, etc.). Tables 1 and 2 show examples of, respectively, metrics and a checklist, for assessing UML class diagrams. The metrics and the items of the checklists were obtained by integrating different existing proposals.

The first column of the Tables is either the name of a metric or a number of the item of the checklist. The second column describes either what each metric calculates, or the defect that the checklist item detects. The third column expresses the level of the evaluation, and the fourth column if such a metric or item can be automated or not. Finally, the last column indicates the bibliographic reference where the metric or the item was proposed, and also the references that show empirical evidence of its usefulness.

Table 1. Example of items in a Checklist for UML class diagrams

Item number	Description	Quality Characteristic	Degree of depth	A/M	References
IT 1	Class names must be unique to them.	Syntactic Correctness	Low	A	[29]
IT 2	If more than one word is needed for class names, the words should be combined into compact, readable entities.	Syntactic Correctness	High	A	[20]
IT 3	Class names start with a capital letter.	Syntactic Correctness	High	A	[20]
IT 4	Classes should not have two or more properties with identical names.	Syntactic Correctness	Medium	A	[18]

Table 2. Example of metrics for UML class diagrams

Name	Description	Category	degree of depth	A/M	References
DIT	In cases involving multiple inheritance, the DIT (depth of inheritance) will be the maximum length from the node to the root of the tree.	Length	High	A	[29] - [20] [18] - [19]
NAGM	Number of the aggregations in a model	Size	Low	A	[18]
NPriA	Number of the Private Attributes for Class	Size	Medium	A	[21]

5.1.1. An example of the assessment of the syntactic correctness of a UML class diagram

In this section we will illustrate the application of the checklist stored in the CQA-Catalogue corresponding to the assessment of the syntactic correctness of a UML class (Table 3). The checklist will be applied to the class diagram of Figure 5. The last column of Table 3 shows the result of the assessment, i.e., if the class diagram fulfils each item of the checklist or not.

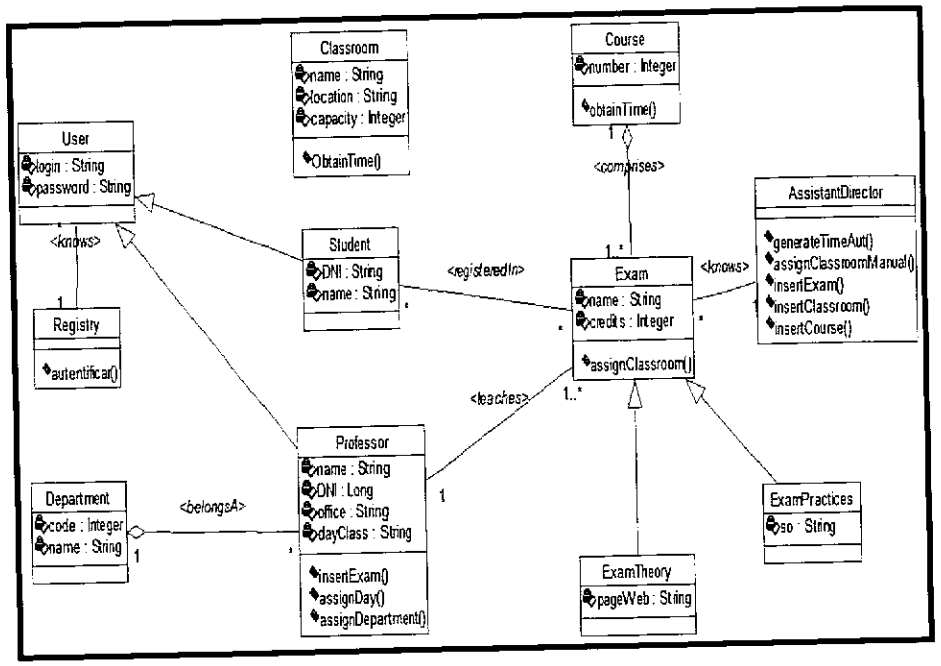

Figure 5. Example of Class Diagram

Figure 6 shows part of the .XSD file corresponding to the class diagram of Figure 5. As we commented before, within the UML-Tool, the class diagram built with Rational Rose (a .mdl file) is converted through EMF into a standard .XSD file. So, within the UML-Tool, the items of a checklist are applied to this .XSD file.

Once the evaluation of the syntactic correction of the class diagram of Figure 5 has been carried out, we could say that of the 21 items on the list, it fulfils only 16, that is to say it is 76% syntactically correct.

This information, which corresponds to the evaluation, will be stored in the XML-Result file within the UML-Tool, and later sent to the CQA-Tool, which finally shows the results to the clients.

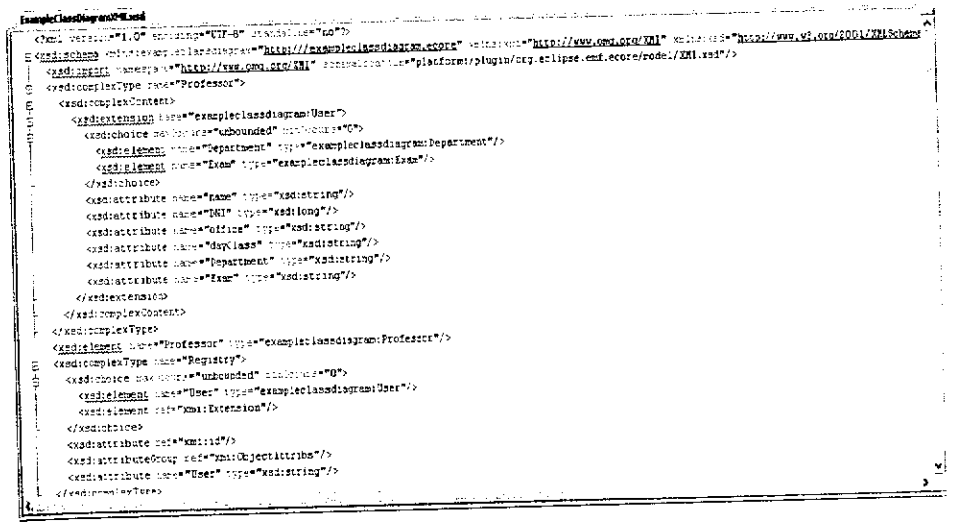

Figure 6. Screenshot of a part of the code of .XSD file

Table 3. Syntactic correctness checklist for class diagram

Item number	Description	Reference	Result of the evaluation
IT1	The classes should have names unique to them.	[30]	YES – Because all the classes have different names.
IT2	If more than one word is needed to name a class, these words should be so combined as to make them readable.	[31]	NO – Because there are classes which have names containing more than one word.
IT3	All the names of the classes should begin with capital letters.	[31]	YES - All the names of the classes begin with capital letters.
IT4	The name of each class should be in the singular.	[31]	YES -All the names of the classes should be in the singular.
IT5	All the attributes should specify their type.	[31]	YES - All the attributes specify their type.
IT6	All the attributes should specify their visibility.	[31]	YES - All the attributes have their visibility specified.
IT7	A class should not define an attribute with the same name as an inherited attribute.	[16]	YES – All the attributes have different names from the inherited attributes.
IT8	A class should not define a method with the same name as an inherited method.	[16]	YES – All the methods have different names from the inherited methods.

Item number	Description	Reference	Result of the evaluation
IT9	The classes should not have duplicate methods.	[16]	YES – There are no classes with duplicate methods.
IT10	The names of the methods should begin with small letters.	[16]	NO – Some methods do not begin with small letters.
IT11	Multiplicity is represented by the number or the rank "N" or "*".	[31]	YES – Multiplicity is recognized by"*".
IT12	Multiplicity should be shown only for associations and aggregations.	[31]	YES – The remaining relationships do not as a rule have multiplicity.
IT13	All the relationships and aggregations should have their multiplicity specified.	[14]	YES – All the relationships and aggregations have their multiplicity specified.
IT14	The associations should be specified by means of a straight line.	[31]	YES – All the associations are represented by means of a straight line.
IT15	The lines which represent associations should have an arrow.	[31]	NO – There are lines of association which do not have an arrow.
IT16	Each class should be associated with at least one class.	[14]	NO – There are classes not associated to any other classes
IT17	Circular associations between two or more classes should not exist.	[14]	YES – There are no circular associations.
IT18	The lines which represent aggregations should have a diamond beside the class "all"	[31]	YES – All the aggregations have a diamond beside the class "all".
IT19	The aggregations should have open arrows.	[31]	NO – Not all the aggregations have open arrows.
IT20	The names of the methods should be verbs.	[32]	NO – There are some methods whose names are not verbs.
IT21	The attributes must be nouns based on the domain.	[32]	YES – The names of the attributes are nouns representing the modeled domain.

Table 3. Syntactic correctness checklist for class diagram (continue)

6. Conclusions and future work

In this paper we have proposed CQA-ENV, an integrated environment that may be used for the ongoing assessment of the quality of any type of software artifact. The main objective is for this evaluation to be done in a way that is automatic and methodical. To this end we have set out the CQA-Meth methodology, accompanied by the set of tools that make up the CQA-ENV environment. These are the ones which allow the evaluation process to be automated.

The main advantages of the CQA-ENV environment are:

- It is generic, which means that it can be applied to any software artifact produced throughout the development lifestyle.
- It is flexible, since it gives us the chance to add, in a very simple way, new tools for evaluating different software artifacts .
- It does not hold fast to any one particular assessment technique. By means of a catalogue it rather allows us to integrate the available techniques for evaluating different software artifacts, allowing us to define for each one of them the appropriate quality models.
- It is standard compliant, due to the fact that it is aligned with the ISO 14598 [5] and the new series of ISO 25000 standards [6].

In this way we cover a need that exists in the software industry, a gap that has come about due to the fact that, although isolated tools do exist for specific software artifacts, none of them covers methodological aspects, nor does any integrate various evaluation techniques at the same time. We might add that none of them can be generally applicable to any software artifact, regardless of the specific type of the artifact.

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

Acknowledgements

This research is part of the MEDUSAS project (IDI-20090557) financed by the "Centro para el Desarrollo Tecnológico Industrial. Ministerio de Ciencia e Innovación"(CDTI), the following projects financed by the "Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha": IDONEO (PAC08-0160-6141), EVVE (HITO-2008-49) and MECCA (PII2I09-0075-8394) and the ISECOMO project (TC20091092) financed by the University of Castilla-La Mancha.

References

[1] Gartner, Analysis of Spain as an Offshore Services Location. ID Number: G00152674, (2007).
[2] SEI, CMMI for Development, Version 1.2. Software Engineering Institute, (2006).
[3] ISO, ISO/IEC 15504-2: 2003/Cor.1:2004(E). Information technology -Process assessment - Part 2: Performing an assessment, *International Organization for Standardization*, Geneva, Switzerland, 2004a.
[4] ISO, ISO/IEC 90003, Software and Systems Engineering - Guidelines for the Application of ISO/IEC 9001:2000 to Computer Software. International Standards Organization, Geneva, Switzerland, 2004b.
[5] ISO, ISO/IEC 14598, Information Technology -Software Product Evaluation, 1998.
[6] ISO, ISO/IEC 25000, Software engineering - Software Product Quality Requirements and Evaluation (SQuaRE) -Guide to SQuaRE, 2005.

[7] C. Denger, and F. Elberzhager, *Basic Concepts to Define a Customized Quality Assurance Strategy IESE- Report Nº 013.07/E version 1.0.*, Fraunhofer, 2007.

[8] C. Denger, F. Elberzhager, and T. Schulz, *A light-weight approach to capture relevant quality characteristics to focus quality assurance. IESE-Report Nº 090.06/E*, 2006.

[9] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, *Practical Software Measurement. Objective Information for Decision Makers,* Addison-Wesley, New York, 2002.

[10] R. Plösch, H. Gruber, A. Hentschel, C. Körner, G. Pomberger, S. Schiffer, and S. Storck, The EMISQ method and It's Tol Support-Expert Based Evaluation on Intrenal Software Quality, *Journal of Innovations in Systems and Software Engineering, Springer London* **4**(1), (2008), p. 3-15.

[11] H. Gruber, C. Körner, R. Plösch, and S. Schiffer, Tool Support for ISO 14598 based code quality assessments, *Sixth International Conference on the Quality of Information and Communication Technology*, 2007, p. 21-29.

[12] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, *Practical Software Measurement, Objective Information for Decision Makers,* A. Wesley, 2002.

[13] M. Piattini, F. García, J. Garzás, and M. Genero, *Medición y estimación del software: Técnicas y métodos para mejorar la calidad y la productividad,* RA-MA, 2008.

[14] B. Berenbach, The evaluation of large, complex uml analysis and design models, *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, I.C. Society, Washington, DC, USA, 2004, p. 232-241.

[15] J. Chimiak-Opoka, G. Giesinger, F. Innerhofer-Oberperfler, and B. Tilg, Tool-Supported Systematic Model Assessment. Modellierung, 2006, p. 183-192.

[16] J. Wüst, *The Software Design Metrics tool for the UML SDMetrics*, 2005, <www.sdmetrics.com/index.html>.

[17] A. Martijn, and M. Wijns, *MetricView Evolution*, 2006, <www.win.tue.nl/empanada/metricview/>.

[18] H. Kim, and C. Boldyreff, Developing Software Metrics Applicable to UML Models, *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)*, 2002.

[19] G. Santucci, and M. Carbone, Fast&&Serious: a UML based metric for effort estimation, *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02)*, Spain, 2002, p. 12.

[20] L. Lavazza, and A. Agostini, Automated Measurement of UML Models: an open toolset approach, *Jurnal of Object Technology* **4**(4), (2005), p. 117-123.

[21] D. Zage, T. Chaffins, and W. Zage, *Metrics Directed Verification of UML Designs*, in *SERC Technical Report*, PHD Thesis, Ball State University, June 2006.

[22] F. García, M. Serrano, J. Cruz-Lemus, F. Ruiz, and M. Piattini, Managing software process measurement: A metamodel-based approach, *Information Sciences* **177** (2007), p. 2570-2586.

[23] IBM, *Rational Rose*, <www-01.ibm.com/software/rational>.

[24] Gentleware, *Poseidon for UML*, <www.gentleware.com/>.

[25] Tigris, *Argouml*, <www.argouml.tigris.org/>.

[26] Acceleo, *Acceleo MDA Generator*, <www.acceleo.org/>.

[27] Eclipse, *Eclipse Modeling Framework Project (EMF)*, <www.eclipse.org/modeling/emf/>.

[28] B. Bordbar, and A. Staikopoulos, Automated Generation of Metamodels for Web service Languages, *Second European Workshop on Model Driven Architecture (MDA)*, University of Kent, 2004.

[29] M. Genero, M. Piattini, and C. Calero, A Survey of Metrics for UML Class Diagrams, *Jurnal of Object Technology* **4**(9), (2005), p. 60-92.

[30] C. F. J. Lange, *Assessing and Improving the Quality of Modeling*, PHD Thesis, Technische Universiteit Eindhoven, 2007.

[31] B. Unhelkar, *Verification and Validation for Quality of UML 2.0 Models,* I. John Wiley & Sons, August 2005.

[32] S. W. Ambler, *The Elements of UML™ 2.0 Style,* A. Modeling, May 2005.

New Trends in Software Methodologies, Tools and Techniques
H. Fujita and V. Mařík (Eds.)
IOS Press, 2009

doi:10.3233/978-1-60750-049-0-165

Dynamic AspectC++: Generic Advice at Any Time[1]

Reinhard TARTLER,[a,2], Daniel LOHMANN[a],
Wolfgang SCHRÖDER-PREIKSCHAT[a], Olaf SPINCZYK[b],
[a] *Friedrich-Alexander University Erlangen-Nuremberg*
[b] *Technical University Dortmund*

Abstract. In theory, the expressive power of an aspect language should be independent of the aspect deployment approach, whether it is static or dynamic weaving. However, in the area of strictly statically typed and compiled languages, such as C or C++, there seems to be a feedback from the weaver implementation to the language level: dynamic aspect languages offer noticeable fewer features than their static counterparts. Espccially means for generic aspect implementations are missing, as they are very difficult to implement in dynamic weavers. This hinders reusability of aspects and the application of AOP to scenarios where both, runtime and compile-time adaptation is required. Our solution to overcome these limitations is based on a novel combination of static and dynamic weaving techniques, which facilitates the support of typical static language features, such as generic advice, in dynamic weavers for compiled languages. In our implementation, the same AspectC++ aspect code can now be woven statically or dynamically into the Squid web proxy, providing flexibility and best of bread for many AOP-based adaptation scenarios.

Keywords. AOP, C++, AspectC++, Programming Languages

1. Introduction

To address the problem of *crosscutting concerns*, a multitude of languages for aspect-oriented programming (AOP) have been proposed over the last decade, with AspectJ being the most prominent example [12]. These languages provide mechanisms to support what is (arguably) considered as the fundamental principles of AOP: *obliviousness* and *quantification* [8]. Obliviousness means that the application of aspects should be completely oblivious to the component code, in the sense that neither components nor their developers have to be aware of the aspects. Quantification stands for the property that the same advice code can easily affect (large) sets of join points.

Quantification is mostly perceived as an issue of the pointcut language, which has to provide means for *join-point set specification*. However, in all nontrivial cases, quantification requires also that the *aspect implementation* is generic in the sense that the aspect behavior adapts automatically to each of the actually affected join points. For instance,

[1]This work was partly supported by the German Research Council (DFG) under grant no. SCHR 603/4, 603/7-1, and SP 968/2-1.
[2]Corresponding Author: Reinhard Tartler, Martensstr. 1, D-91058 Erlangen; E-mail: tartler@cs.fau.de.