30

A Unifying Model for Software Quality Engineering

Ilkka Tervonen

Department of Information Processing Science, University of Oulu,
FIN-90570, Oulu, Finland
e-mail: tervo@rieska.oulu.fi

Abstract

The demand for quality in software production raises new challenges for software engineering process. The solutions proposed for the quality problem are based on standards which provide an abstract frame and drive further tailoring and installation of quality assurance methods and quality control principles. This installing in an enterprise requires a lot of time and effort. The methodologists and persons who are responsible for the choice of the method demand a framework, which helps them to assess the cover and appropriateness of alternative methods from quality engineering viewpoint in particular. The present paper introduces a unifying model which can be used for this assessing of the necessary characteristics of a method. In addition, we briefly illustrate how to apply this model to the definition of a specific review method.

Keyword Codes: D.2.0; K.6.3; K6.5
Keywords: Software Engineering, General; Software Management; Security and Protection

1. MOTIVATION

Following Yourdon [1], "software quality is like loyalty, bravery and thrift: everyone is in favor of it, but it's not always clear that enyone practices it". Although standards (e.g. ISO 9000-3, [2]) form a good starting point for installing of quality assurance in an enterprise and slogans such as "Quality is job number one" motivate emphasis on quality, its installing in an enterprise requires a lot of time and effort. We need also a comprehensive conceptual model (a Unifying Model, [3]) of software quality engineering which defines the quality assessment and quality assurance throughout the software development. This model would not necessarily be of immediate use to a system builder, but it would be a tool for academic analysis that could in turn yield structures and tools useful to a practitioner [3]. It can also be used as a framework which helps a software engineer or a methodologist to assess the necessary characteristics of a method from a quality engineering viewpoint. The Unifying Model can also be applied to the definition of a specific review method.

2. THE UNIFYING MODEL

The Unifying Model is a conceptual model and is defined in a space of three dimensions, levels of abstraction (LoA), software configuration management (SCM) and quality (cf. Figure 1). The dimensions are justified by their necessity in quality engineering with respect to a software object or software product, and there are two-way relationships in each dimension

M. Lee et al. (eds.), *Software Quality and Productivity*

which explain the change from generality/wholeness to specifity/parts and reverse.

The LoA dimension represents a scale of definition specificity and implementation dependence, i.e. the levels of description (different viewpoints), refinements inside a level and mapping between levels. The SCM dimension adds the aspect of a software product intended to focus on software configuration complexity, i.e. software objects (a software description or a software module), their versions or revisions, and their configuration into a software product and further into a software product family.

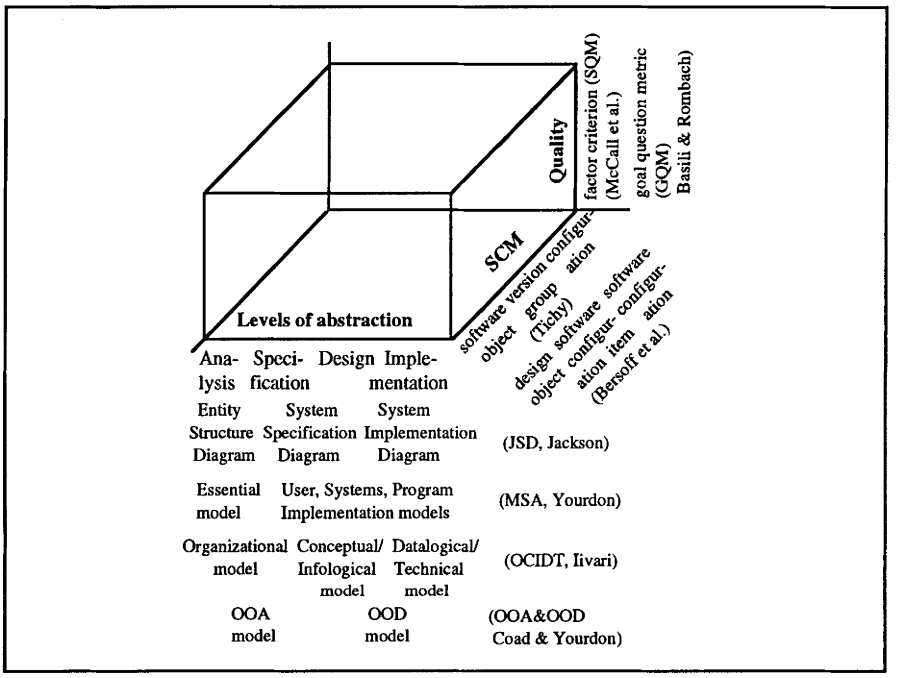

Figure 1. The space of three dimensions

The quality dimension explains the quality of a software object or a software product at a specific level of abstraction and defines quality specificity in terms of a quality factor and criterion, for example.

3. ALTERNATIVE MODELS IN THE UNIFYING MODEL

In the previous section we presented general definitions for the dimensions. We now instantiate and refine these definitions, choosing a specific model for each dimension.

3.1. Levels of abstraction

The levels of abstraction (LoA) dimension encompasses different aspects on the progress of

software development such as levels of description (a specific viewpoint at each level), refinements inside a level and mappings between levels. Typically, a specification level depicts a logical viewpoint (implementation independent) and a design level a more implementation dependent viewpoint. The mappings between levels ensure that the characteristics specified at upper levels remain and will be implemented at lower levels. In object-oriented approach the characteristics such as seamless transition cause a blend of refinement and mapping (i.e. an incremental development of a class hierarchy is equal to refinement although the characteristics added are from levels of abstraction).

Among the alternatives (i.e. JSD [4], MSA (Modern Structured Analysis [5], OCIDT [6], [7] and OOA/OOD [8], [9]) we have chosen the OCIDT model, and the OCT (Organizational=O, Conceptual=C, Technical=T) model as its successor. According to Iivari [6] the three levels of abstraction (or modelling) are (1) the organizational level, which defines the organizational role and context of the IS/SW (information system/software) product, (2) the conceptual/infological level, which defines an "implementation independent" specification for the IS/SW product, and (3) the datalogical/technical level, which defines the technical implementation of the IS/SW product. Due to our focus on software engineering, in which the terms infological and datalogical are not very familiar, we use the term OCT model. Although the choice of the OCT model can be seen partly as "taken for granted", we also can justify the choice by its theoretical soundness, i.e. the OCT model is defined by three models, a software model, a process model and a quality model. The software model ties it to information systems research tradition, and the process model defines a hierarchical spiral model which has a number of similarities with the Boehm's spiral model [10]. Both models are tailored to object-oriented approach in our application (cf. [11]). The quality model presents a framework for choices between alternatives and supports the quality dimension, too. The most serious disadvantage of the OCT model is caused by its "unknown" status in the field of software engineering.

The relationships on this dimension are specialization/generalization and decomposition/ aggregation. A specialization describes a refinement using transition inside a level and from a higher level of abstraction to a lower one, whereas generalization is the reverse. Decomposition is largely similar to specialization, and describes how a software object is decomposed into more precisely specified software objects, whereas aggregation forms a composite software object from constituent software objects.

3.2. Software configuration management

Software configuration management is an umbrella activity that is applied throughout the software engineering process. It identifies, controls, audits and reports modifications that invariably occur while software is being developed and after it has been released to the customer. A software object [12] or a design object [13] is a core unit in our SCM concept. We borrow the definition of Tichy [12], who defines it as any kind of identifiable, machine-readable document generated during the course of a project (e.g. requirements documents, design documents, specifications, interface descriptions, program code, test programs, test data, binary code, user manuals, etc.). Tichy also places emphasis on two "orthogonal refinements" of software objects, one according to how they were created, the other according to the structure of their body. Creation of a software object depends on its type, since a source object requires human action but a derived object is generated automatically by a program, usually from other software objects. Corrective, adaptive, perfective and preventative maintenance activities produce a steady stream of updates. Since most changes are incremental, they are best viewed as producing related

versions of objects rather than separate, unrelated objects. According to Tichy [12], source objects are connected via the relations revision-of and variant-of. The subtypes of a revision-of relation are derived from maintenance activities, being correction-of, adaptation-of, enhancement-of and preventative-of. The revision-of relation forms a directed, acyclic graph reflecting the development history. Each revision may be composed of different variants and thus a variant-of relation identifies the starting points of parallel lines of development.

Although we identify these revisions and versions even in small-scale software development, they will be more inportant in the case of a software product or software product family. The relationships on the SCM dimension also follow this classification. The relationships used here are versioning/configuring, in which versioning describes the change history of the software object or software configuration using an evolution graph, for example, and configuring works in the reverse order, compiling a software composition, a software product and finally a software product family. The major idea is that this grouping allows more flexibility in the evolution of software products and makes the management of software evolution easier.

3.3. Quality

The software engineering area offers some alternative quality "approaches", such as the Software Quality Metrics (SQM) (cf. [14], [15]) and Goal Question Metric (GQM) (cf. [16]) models. Quality is managed in the SQM model by means of quality factors, a set of criteria and metrics for estimating the value of these criteria and of the factors. In the GQM model the terms are goals, a set of questions and metrics for measuring achievement of the goals. Although the terms are slightly different, the models are rather similar because the quality factors selected can be understood as goals, the quality criteria are explained by means of checklists corresponding to the questions, and the metrics are similar in both approaches. In view of its more comprehensive taxonomy of quality factors and criteria, we have chosen the SQM model.

Our synthesis of the SQM model is based mainly on the reports of McCall et al. [14] and Boehm et al. [15], and on the IEEE standard [17]. The factors in the SQM model depict more user-perceived characteristics than do the criteria, which characterize lower-level software-oriented aspects. The IEEE standard [17] defines the factors as management-oriented views of quality. Associated with each factor is a direct metric which serves as a quantitative representation of the quality factor. The second level in the hierarchy consists of quality criteria (sub-factors), which represent technically-oriented concepts. The criteria may correspond to more than one factor, and they are concrete attributes that are more meaningful than factors to the technical staff, i.e. analysts, designers, programmers, testers and maintainers. At the third level in the hierarchy, the criteria are decomposed into metrics used to measure system products and processes during the development life-cycle. The existence of an interrelationship between the factors denotes that they may have cooperative or conflicting relationships, i.e. emphasizing expandability can cause conflicts with efficiency, for example.

The relationships on the quality dimension follow this classification. When the first relationship in quality dimension is a quality break-down, the other one is a quality accumulation. It is based on the fact that direct metric values (factor values) are typically unavailable or expensive to collect early in the software life cycle. For this reason, metrics on the criteria level are used, either collectively or independently, to estimate factor values. There are also some problems, in that the metrics are programming language-dependent (e.g Fortran), their use requires a substantial amount of subjective judgement, and some metrics do not require specific checking if good compilers are used [15].

4. APPLYING THE UNIFYING MODEL TO THE DEFINITION OF A REVIEW METHOD

The characteristics of a Unifying Model are now applied to the definition of a pre-review method - a quality-driven assessment method. The definition of the method is based on object-oriented tailoring of the model. The structure in LoA and SCM dimensions is presented in the form of a class hierarchy. In the LoA dimension it represents a development history and in SCM dimension a evolution history. When defining the quality-driven assessment activities, we focus on a general pre-review aspect and on the specific quality assessment activities defined as relationships earlier in the Unifying Model. From the viewpoint of a pre-review method, quality-driven assessment serves the goal of a review, i.e. it removes defects in software definitions and code. The prefix "pre" implies that quality-driven assessment also supports the essential review process by preparing material which helps reviewing. This material encompasses design rationales which justify trade-off situations in terms of quality issues.

The specific quality assessment activities are derived from relationships in the Unifying Model. Quality break-down activity implements the break-down of a quality factor to a set of quality criteria, whereas in the quality accumulation the value of a quality factor is estimated from a set of criteria. The quality-driven specialization describes the transition of quality requirements from a superclass to subclasses. Quality-driven decomposition is largely similar to quality-driven specialization. The major difference is that decomposition always produces a set of subclasses. Quality-driven association is a constructing activity and works together with quality-driven specialization and quality-driven decomposition. It means link to library components or their further applications, i.e. how the quality of a class can be composed from quality estimates of "used" classes. Quality-driven aggregation is a strong form of association and means constructing inside an application class hierarchy, i.e. how the quality of a superclass can be composed from quality estimates of subclasses. Quality-driven generalization leads to the production of a new and more reusable superclass. Quality-driven versioning describes the change history of a software object, a software configuration, a software product or a product family. The choices are based on a quality estimate of an alternative version or revision and the history can be illustrated by means of an evolution graph, for example. Quality-driven configuring is a reverse activity combining a software composition, a software product and finally a software product family. Collection is also here based on quality estimates and allows different versions of software products which vary in relation to specific quality characteristics, for example.

5. CONCLUSIONS

The present paper formulates a comprehensive conceptual model of software quality engineering which defines quality assessment and quality assurance throughout software development. We define a Unifying Model in a space of three dimensions, levels of abstraction (LoA), software configuration management (SCM) and quality, and choose a specific model for each dimension. The models chosen are the OCT (O=Organizational, C=Conceptual, T=Technical) model on the LoA dimension, a software configuration model of Tichy on the SCM dimension and a SQM (software quality metrics) model on the quality dimension.

When applying the Unifying Model to the definition of the quality-driven assessment, we

focus on a general pre-review aspect and on the specific quality assessment activities defined as relationships earlier in the Unifying Model. The specific quality assessment activities are derived from relationships in the Unifying Model, being quality-driven specialization, quality-driven decomposition, quality-driven association, quality-driven aggregation and quality-driven generalization on the LoA dimension, quality-driven versioning and quality-driven configuring on the SCM dimension, and quality break-down and quality accumulation on the quality dimension.

REFERENCES

1. Yourdon E.: 'Editor's notes', *American Programmer*, 1993, vol 6, (6), pp. 1-2
2. ISO: Quality management and quality assurance standards, Part 3: guidelines for the application of ISO 9001 to the development, supply and maintenance of software, ISO 9000-3, International Organization for Standardization, Geneve, 1991
3. CSTB: 'Scaling up: a research agenda for software engineering', *Communications of the ACM*, 1990, vol 33, (3), pp. 281-293
4. Jackson M.: 'System development', Prentice Hall, Englewood Cliffs, NJ, 1983
5. Yourdon E.: 'Modern structured analysis', Prentice Hall, Englewood Cliffs, NJ, 1989
6. Iivari J.: 'Hierarchical spiral model for information system and software development. Part 1: theoretical background', *Information and Software Technology*, 1990, vol 32, (6), pp. 386-399
7. Iivari J.: 'Hierarchical spiral model for information system and software development. Part 2: design process', *Information and Software Technology*, 1990, vol 32, (7), pp. 450-458
8. Coad P. and Yourdon E.: 'Object-oriented analysis', Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1991
9. Coad P. & Yourdon E.: 'Object-oriented design', Prentice Hall, Englewood Cliffs, NJ, 1991
10. Boehm B.W.: 'A spiral model of software development and enhancement', *ACM SIGSOFT Software Engineering Notes*, 1986, vol 11, (4), pp. 14-24
11. Tervonen I.: 'Quality-driven assessment: a pre-review method for object-oriented software development', Dissertation thesis, University of Oulu, Department of Information Processing Science, Reserch papers, Series A19, 1994
12. Tichy W.: 'Tools for software configuration management', In: Winkler J.F.H. (ed.), *Proceedings of the International Workshop on Software Version and Configuration Control*, Stuttgart, 1988, pp. 1-32
13. Bersoff E.H., Henderson V.D. and Siegel S.G: 'Software configuration management: an investment in product integrity', Prentice Hall, Englewood Cliffs, NJ, 1980
14. McCall J.A, Richards P.K and Walters G.F.: 'Factors in software quality', Volumes I, II, and III, RADC reports, 1977
15. Boehm B.W., Brown T.R., Kasper H., Lipow M., Macleod G.J. and Merritt M.J: 'Characteristics of software quality', North-Holland, Amsterdam, 1978
16. Basili V.R. and Rombach H.D.: 'The TAME project: towards improvement-oriented software environments', *IEEE Transactions on Software Engineering*, 1988, vol 14, (6), pp. 758-773
17. IEEE: 'IEEE Standard for a software quality metrics methodology (Draft)', P-1061/D20, IEEE Computer Society Press, New York, NY, 1989