Modelling Method Heuristics for Better Quality Products

Naveen Prakash and Ritu Sibal

Division of Computer Engineering
Netaji Subhas Institute of Technology (Formerly DIT)
Kashmere gate, Delhi –110003, India
email : [ritusib, praknav]@hotmail.com

Abstract. Development methods contain heuristics and constraints that help in producing good quality products. Whereas CASE tools enforce method constraints, they rarely support heuristic checking. This paper develops a generic quality model, capable of handling both method constraints and heuristics, which forms the basis of a uniform mechanism for building quality products. The model is metric based, hierarchical in nature, and links metrics to the developmental decisions that are available in a method. The use of this model and the associated quality assessment process is demonstrated through an example of the Yourdon method.

1 Introduction

Broadly speaking, methods provide three features for quality (a) guidelines, (b) method constraints and (c) method heuristics. In supporting the use of methods, CASE tools have particularly looked after method constraint satisfaction. However, it is more difficult to find CASE tools that support heuristic checking. Thus, application engineers are expected to examine products manually and determine whether method heuristics are satisfied or not. The purpose of this paper is to remove this deficiency of CASE tools.

Our approach is based on two assumptions. The first is that a generic solution to the problem of assuring product quality should be found. This genericity is in the sense that both, constraint enforcement as well as heuristic satisfaction can be handled in a uniform manner. The second assumption is that product quality should be metric-based.

A number of metric based quality models [1, 2, 3, 4] have been developed in the area of Software Engineering.These models deal with a given, fixed set of quality factors whereas others [5, 6] cater to only one quality factor. All these models decompose quality factors into quality criteria with which quality metrics are directly associated. In the area of Information Systems also, attempts have been made to develop quality models [7,8]. The former also relies on the decomposition approach. The latter proposes a framework within which quality issues affecting information systems can be formulated and discussed. However, this work is not directly concerned with quality assessment of information system products.

It can be seen that aside from dealing with a model-defined set of quality factors these models do not relate quality factors/criteria to the development process. Thus, it is not known what development decisions affect which factors/criteria in what way.

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 429-433, 1999.

As a result, these quality models are stand-alone and divorced from the development activity. Further, none of these models deal with issues of method constraint enforcement, heuristic satisfaction etc. which are interesting in methods.

Thus, the generic quality model developed here must (a) handle different types of quality factors and decompositions and (b) relate development decisions with quality factors/criteria. In [9,10] a method has been viewed as a set of decisions and dependencies between them. The set of decisions has been partitioned into product manipulation and fixed structure enforcement decisions. Product manipulation decisions affect quality factors/criteria whereas fixed structure enforcement decisions determine metric values. Thus, as the product is manipulated, its quality changes and the new product quality can be determined by using fixed structure enforcement decisions. We propose to use this view of decisions in our quality model.

In the next section, we consider heuristics in detail and identify the manner in which we shall represent them. Thereafter, in section III, we present our generic quality model. In section IV the manner in which quality is assured using heuristics is shown through a Data Flow Diagram (DFD) design example.

2 Method Heuristics

When application engineers use heuristics then they may decide that heuristic satisfaction in the product is not mandatory. Thus for example, the number of levels in the specialisation hierarchy in an OMT product may be 4 whereas the corresponding OMT heuristic on specialisation hierarchy suggests that this should be less than 3. Even though the product violates the OMT heuristic, an application engineer may choose to accept this product. This situation is quite different from that in method constraint enforcement where the satisfaction of method constraints in the product is mandatory.

Now, the form of a method heuristic is textual. This is a descriptive statement and therefore, neither amenable to metric based quality calculations nor adequate to form a basis of computer based guidance. In accordance with our interest in metric based quality, we recast a method heuristic as a heuristic function .To do so ,we start by express a method heuristic as

(method concept, metric, operation, value)

where method concept refers to the concept on which the heuristic is defined, metric is the metric used to assess heuristic satisfaction, operation is relational operator and value is the bound within which the metric value should lie. For instance, a design heuristic in the Yourdon method[11] states :

„Avoid Processes that have inputs but no outputs“. This means that a process should produce at least one output. This is expressed as

(Process, outpt _cnt, >, 0)

This form is then converted to a heuristic function, F(H). F(H) has two parts to it, a header and a function body. Inside the function body, metric is related to the value through the operation and a Boolean value is returned. This value is checked to ascertain for the satisfaction of the method heuristic.

3 The Generic Quality Model

The central notion (see Fig. 1) of our model is that of a quality requirement. We associate an attribute, *satisfaction*, with the quality requirement which takes the value from the domain {mandatory,optional}. For example, if the quality requirement is a method constraint then its satisfaction attribute has the value mandatory whereas for a heuristic this value is optional.

A quality requirement can be of two kinds, simple or complex. A simple quality requirement cannot be decomposed into simpler requirements whereas a complex quality requirement can be decomposed into simpler ones. Quality requirements are related to one another. One requirement may support another or may be in opposition to it. Thus, when computing the extent to which a requirement has been satisfied, it is necessary to add/subtract the effect of requirements which support/oppose it respectively. This computation is based on the notion of quality metrics.

There are two kinds of metrics, simple and complex. A simple metric is one which can be directly measured in the product whereas a complex metric can only be computed from simpler ones. Clearly, simple metrics supply the 'base' values from which the value of any complex metric is calculated. A metric is associated with every quality requirement.

The value of the metric can be changed by product manipulation decisions. Further, the value of a metric can be checked by a quality enforcement decision which, as defined in [10] returns a value. Based on this value, the application engineer decides which product manipulation decision is to be executed next.

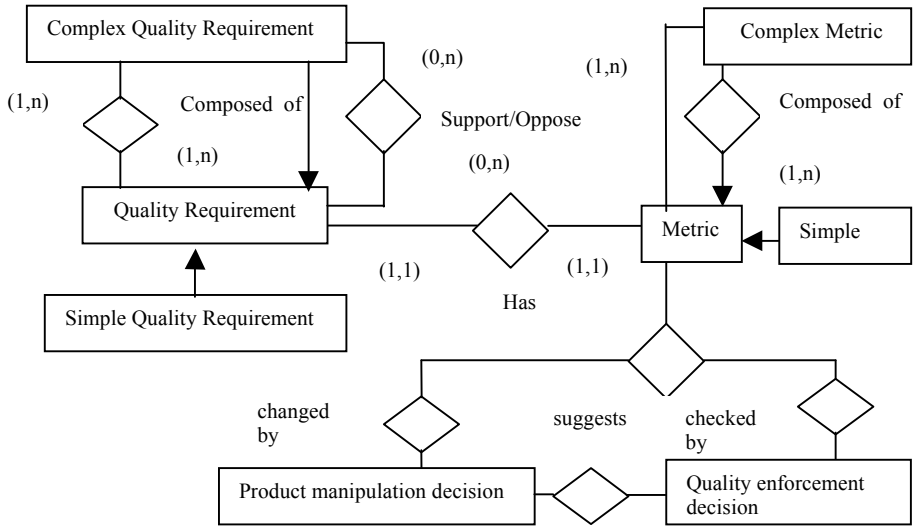

Fig. 1. The Quality Model

4 Using the Quality Model : An Example

In this section, the notions developed in the previous sections will be illustrated through an example. Let us say that we wish to build a DFD. The design heuristics, for DFDs [11] and the function headers are as follows :

1. (PRODUCT, P_cnt, < , 7) Process_count(PRODUCT, P_cnt, 7)
2. (Process, outpt_cnt, > , 0) Output_count(Process, outpt_cnt, 0)
3. (Process, inpt_cnt, >, 0) Input_count(Process, inpt_cnt, 0)

Consider the following product manipulation and heuristic enforcement decisions:

Product manipulation
<Process, create>, <Input, create>, <Output, create>
< Process, Input, couple>, < Process, Output, couple>

Heuristic enforcement
<PRODUCT, Process_count(PRODUCT,P_cnt,7), enforce_Process_count, 7)>
<Process, Output, Output_count(Process, outpt_cnt, 0), enforce_Output_count>
<Process, Input, Input_count(Process, inpt_cnt, 0), enforce_Input_count>

The requirement dependencies between these are as follows:
1.<Process, create>----REQ----<PRODUCT, Process_count(PRODUCT, P_cnt, 7), enforce_Process_count>
2.<Process, Input, couple>---REQ----<Process, Input, Input_count(Process, inpt_cnt, 0), enforce_Input_count>
1. < Process, Output, couple>---REQ---<Process, Output, Output_count(Process, outpt_cnt, 0), enforce_Output_count>

Let the sequence of product manipulation decisions to build a DFD be as follows:
1.<Input(Cash withdrawal request), create>
2.<Process(Verify account no.), create>
3. <Output(Valid accnt. no.), create>
4.<Process(Verify account no.), Input(Cash withdrawal request), couple>
5.<Process(Verify accnt. no.), Output(Valid accnt. no.), couple>

After the execution of the first decision the Input, Cash withdrawal request, is created. There is no heuristic applicable to this concept. However, when the second decision is executed and a Process is created then the first requirement dependency shown above comes into play. The metric P_cnt is computed as unity.If the application engineer were to use the first heuristic enforcement decision above then the value true would be returned. However, the application engineer has chosen to execute another product manipulation decision (the third one) to create an output. This again does not involve any heuristic. After the fourth decision is executed, the second requirement dependency also comes into play, inpt_cnt is now incremented by one. It can be seen that the application engineer can obtain product quality information while the product is under development.

5 Conclusion

The proposed quality model is generic enough to handle both constraints and heuristics. A prototypical version of the quality enforcement mechanism developed in this paper which shall handle both, constraints and heuristics is under development. The difficult part of our proposals is the generation of method heuristics. We treat this as a part of the method engineering problem and will, in future follow the rule based approach[12] to address it.

References

1. Boehm B., Brown J., Kaspar J., Lipow M., MacCleod G., & Merrit M., „Characteristics of Software Quality". North Holland.
2. McCall J., Richards P., & Walters G., „Factors in Software Quality". Vols I, II, III, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055.
3. Gilb T., „Principles of Software Engineering Management", (addison wesley, 1988).
4. Kitchenham B., „Software quality assurance", Microprocessors and Microcomputers, vol 13, no. 6, 373-381.
5. Inglis J., „Standard Software Quality Metrics", AT&T Technical Journal, 1986, vol 65, (2), pp 113-118.
6. Daskalantonakis, MK: „ a Practical View of software Measurement and Implementation experiences Within Motorola" IEEE Transaction on Software Engineering, 1992, vol 18, (11), pp 998-1010.
7. Delen, GPAJ, Rijsenbrij, DBB: „The Specification Engineering, and Measurement of Information Systems Quality", Journal of systems and Software, 1992, Vol 17, (3), pp 205-217.
8. Krogstie J., Lindland O., Sindre G., „Towards a deeper understanding of Quality in Requirements Engineering", in „Advanced Information Systems Engineering", Springer 1995.
9. Prakash N., „Towards a Formal Definition of Methods", Requirements Engineering Journal, Springer.
10. Prakash N., & Sibal R., „Computer Assisted Quality Engineering : A CASE for Building Quality Products", in First International Workshop on the Many Facets of Process Engineering, Tunis, September, 1997.
11. Yourdon E., „Modern Structured Analysis", Prentice-Hall.
12. Prakash N., & Daya Gupta., „An Architecture for a CAME TOOL", in Proceedings of the 8th European-Japanese Conference on Information Modelling and Knowledge Bases, pp 147-179.