A Generic Model for Assessing Process Quality

Manoranjan Satpathy[1], Rachel Harrison[1], Colin Snook[2], and Michael Butler[2]

[1]School of Computer Science, Cybernetics and Electronic Engineering
University of Reading, Reading RG6 6AY, UK
{M.Satpathy, Rachel.Harrison}@reading.ac.uk
[2]Department of Electronics and Computer Science
University of Southampton, Highfield, Southampton SO17 1BJ, UK
{cfs98r, mjb}@ecs.soton.ac.uk

Abstract. Process assessment and process improvement are both very difficult tasks since we are either assessing or improving a concept rather than an object. A quality process is expected to produce quality products efficiently. Most of the existing models such as CMM, ISO 9001/9000-3 etc. intend to enhance the maturity or the quality of an organization with the assumption that a matured organization will put its processes in place which in turn will produce matured products. However, matured processes do not necessarily produce quality products [21, 6]. The primary reasons are: (i) In the process quality models, the relationship between the process quality and product quality is far from clear, and (ii) many of the process models take a monolithic view of the whole life-cycle process, and as a result, the idiosyncrasies of the individual processes do not receive proper attention.

In this paper, we first define an internal process model in a formal manner. Next, we define a generic quality model whose scope covers all the development processes and most of the supporting processes associated with the development phase. The generic quality model is a parametric template and could be instantiated in a systematic manner to produce the quality model for any individual process. We then show such a customization for the formal specification process and use this customized model to formulate a GQM-based measurement plan for the same process. We then discuss how the generic model would be useful in process assessment and process improvement.

1 Introduction

Our aim is to perform process assessment by using techniques from Empirical Software Engineering including the use of quantifiable metrics. Many methods exist for the selection of metrics; prominent among them is the Goal-Question-metric (GQM) method [1]. Under this approach, we define some goals, refine those goals into a set of questions, and the questions are further refined into metrics. Of course it is important, in relation to a certain objective, to neither measure too much nor too little. The measured values are then interpreted to answer the goals that we started with. The question that then arises is: from where should we choose the goals whose refinement will give us the metrics? The usual approach is to choose a quality model, and keeping both the model as well as the application domain of the process in mind, the goals are derived through extended consultation [17, 6]. Among the models

R. Dumke and A. Abran (Eds.): IWSM 2000, LNCS 2006, pp. 94-110, 2001.

available for process assessment, the most influential ones include: the Capability Maturity Model (CMM) [12], ISO/IEC 12207 [15], ISO 9001/9000-3 [14], the BOOTSTRAP model [3], and the SPICE model [7]. These process models are not specific enough to cater to the needs of each of the individual processes. The reasons are: (i) the nature of processes vary widely (ii) most of the models are more oriented towards enhancing the maturity of the organization and take a monolithic view of the overall development process, and finally (iii) the process models, while emphasizing on the process activities, often put too little importance on the products which are the results of the process activities. In this context, there are certain process aspects (including the duality of process attributes which we discuss later) which are not addressed by any of the existing models. In order to alleviate these problems, we define a generic process quality model that incorporates the merits of all the existing models, and takes care of their deficiencies. The generic model could be instantiated to be the customized model of any individual process.

The organization of the paper is as follows. Section 2 describes related work. Section 3 presents an internal process model. Section 4 discusses the generic model and its instantiation of individual processes. Section 5 discusses a GQM-based measurement plan. Section 6 discusses the role of the generic model as regards to process assessment and improvement, and finally, section 7 concludes the paper.

2 Related Work

ISO/IEC 9126 [13] describes a generic model for specifying and evaluating the quality of software products. The model isolates six factors, called Functionality, Usability, Reliability, Efficiency, Maintainability and Portability; and the quality of a product is defined in terms of the quality of the above factors. Each factor may in turn be defined by a set of subfactors. The FURPS+ model used by HP [10] is quite similar to ISO/IEC 9126. Focusing on product quality alone may not guarantee that an organization will deliver products of good quality. Products are created by processes. So, based on an orthogonal view that improving the quality of a process will deliver products of good quality [9], many models have been developed. Prominent among them are the CMM [12] and ISO 90001 [14]. Models like Bootstrap [3] and SPICE [7] are variants of the CMM. ISO/IEC 12207 [15] does a classification of all processes associated with the software development and offers general guidelines which can be used by software practitioners to manage and engineer software. The scope of most of these standards cover an entire organization.

The CMM deals with the capability of a software organization to consistently and predictably produce high quality products [12]. The model defines five maturity levels: from Level 1 to Level 5. A maturity questionnaire assesses the following three areas of the organization: (i) organization and resource management, (ii) software engineering process and its management and (iii) the tools and technology; and based on the assessment the model certifies that the organization is at certain maturity level. The model also identifies the weak areas and prescribes a set of guidelines; following those guidelines, an organization can attain the next higher maturity level. Measurement, quantitative feedback from the processes and continuous process improvement are some of the highlights of the model. However, it has been suggested [21] that increasing the maturity level of an organization may not necessarily lead to

improvements in the quality of processes. Further, there is no evidence of a link between a high CMM rating and products that are of high quality [6].

ISO 9001/9000-3 is a model for quality assurance in the development, supply, and maintenance of software. The key strength of ISO 9001 is in its quality system processes. Most of the basic practices of the 9001 translate to level 2 practices of CMM and some translate to level 3 practices [5]. The continuous improvement paradigm has not also been adequately addressed by the standard.

The GQM method [1, 20] proposes a measurement plan for assessing the quality of entities like products, processes or people. It starts with a set of business goals and the goals are progressively refined through questions till we obtain some metrics for measurement. The measured values are then interpreted in order to answer the goals. Existing approaches choose a quality model from those that exist so as to generate the business (or the primary) goals of the GQM formulation for any individual product or process. Application of Metrics in Industry (ami) [Pul 95] combines CMM and the GQM method, and the result is that it provides a complete framework for process improvement. The ami approach is: iterative, goal-oriented, quantitative, involves everyone in the organization and integrates necessary management commitment. It covers the whole of the process improvement cycle. CMM or other standards like Bootstrap, SPICE, ISO 9001 *etc.* is used to identify weak areas in the development process. This information along with the business and environment specific objectives is used to define some software process goals. The goals are validated and next, using the principle of GQM, they are refined into subgoals. The subgoals are further refined into metrics and a measurement plan is made to collect data. The data are then analysed and related to the original goal. Based on the data collected an action plan may be made to improve the development process. New goals are then defined and the cycle is repeated.

Focusing on either process quality or product quality alone is not sufficient. In a European-wide awareness survey 65% of the respondents agreed that certification against ISO 9000 is not enough to guarantee the quality of software products [2]. 40% of the respondents agreed that a combined certification of products and processes is necessary, and almost all of the models fail to make the relationship between process quality models and product quality clear. We will make such a relationship clear by taking the dual nature of process attributes into account. Also note that the models cannot compare lateral processes; for example, they cannot address a question like: is formal specification more suitable than informal specification for a particular organization? Two development processes following one of the approaches may have similar ratings under a process model and hence such a model cannot do such a comparative study.

The PROFES (product-focused) improvement methodology [18] follows an orthogonal approach to process improvement. It first uses ISO 9126 to identify the subfactors in relation to product quality which need to be improved. Following a PPD (Product-Process Dependency) model, it identifies the process attributes which need to be improved for achieving the desired product quality. Then an action plan is made following an ISO 15504 compliant method and the plan is executed. So far as improvement of product quality is concerned, our objective is similar to that of the PROFES methodology. However, in addition, we also address some process specific issues like process faults, process understandability and process reliability.

3 An Internal Process Model

In this paper, unless otherwise stated, by a process we will mean *any software activity associated with the development and maintenance of software*: from requirement analysis through to maintenance. A product is an entity, which a process (e.g. any software activity) produces as output. Examples are requirement documents, specifications, end-products *etc*. Products may also be fed to processes as inputs – specification is an input to design, implementation is as input to testing and so on. Each of the products has a type in the sense that it belongs to a set. Table 1 enumerates some of the products and their types. For a uniform treatment, we will assume that a requirement that exists in a user's mind is also a product. The set of pre-elicitation requirements (the requirements in the user's mind) is well-defined in the sense that an analyst is capable of extracting it in order to produce a requirements document. The set of product definitions above are not exhaustive; based on the application context, a process designer may add or modify the above sets. But once they are defined, they should be adhered to by the process executers.

We will now define our process model which will elucidate the internal structure of a process. A process is any software activity which takes product(s) as input and produces product(s) as output. Formally, it could be defined as a relation from a set of products to another set of products; the set of relations from *m* input products to *n* output products could be denoted by:

$$IP_1 \bullet IP_2 \bullet \ldots \bullet IP_m <---> OP_1 \bullet OP_2 \bullet \ldots \bullet OP_n$$

where IP_i and OP_j are the types of the i-th input product and the j-th output product respectively. Software processes can be classified into two categories:

(i) **Constructing processes:** These processes take a set of products and apply some transformations over them to generate a new set of products; e.g. formal specification, implementation, maintenance etc.

(ii) **Verifying processes:** Such processes take a set of products and do some verification or validation over them to determine whether they hold or do not hold certain properties. They are more like functions. They do not apply any transformations on the input products. The result is usually a Boolean condition, and some other information such as defect data, certain inferences from verification and/or other documentation. Examples of such processes are: formal verification, validation, code inspection, testing etc.

Our definitions of constructing processes and the verifying processes mostly correspond to the development processes and some of the supporting processes of ISO/IEC 12207 [15]. Some examples of processes are shown in Table 2.

The above definitions are not absolute but should be adapted by the designer of a process who will choose the input product set and the output product set keeping the application context in mind. For instance, the validation of formal specification may output a set of features which are found invalid; the code inspection process in addition to indicating whether code is good, bad or of moderate quality may output a set of suggestions for code improvement. But a prior definition of a process on the above lines by its designer establishes a standard, which, if adhered to by the process executers, will make the process execution more disciplined. Usually the output product of a process is fed as input to another process. So giving a type to such a

product helps in better communication between the process executers of both the processes.

Table 1. Some products with their types

req:	***PRE-ELICIT-REQ*** **// Set of pre elicitation requirements**
rd:	***RD*** **// Set of requirement documents.**
spc:	***SPEC*** **// Set of Specifications**
fs:	***FML-SPEC*** **// Set of formal specifications**
des:	***DESIGN*** **// Set of designs**
mod:	***MODULES*** **// Set of set of program modules**
fd:	***FDESIGN*** **// Set of formal designs**
impl:	***IMPL*** **// Set of implementations**
testdata:	***TESTDATA*** **// Set of test data**
defectdata:	***DEF-DATA*** **// Set of defect data**
doc:	***DOC*** **// Set of documentations**
insp-flag:	***FLAGSET*** **// Set of flags to indicate quality of code**
form-proof:	***FML-PROOF*** **// Set of formal proofs**
safety-cases:	***SAFETY-CASES*** **// Set of safety cases**

Table 2. Some processes with their types

Requirement analysis:	***PRE-ELICIT-REQ*** $<->$ ***RD***
Formal specification:	***RD*** $<->$ ***FS***
Formal design:	***RD*** • ***FS*** $<->$ ***FD***
Maintenance:	***RD*** • ***SPEC*** • ***DES*** • ***IMPL*** • ***DOC*** $<->$ ***RD*** • ***SPEC*** • ***DES*** • ***IMPL*** • ***DOC***
Validation of formal spec.:	***RD*** • ***FS*** $<->$ ***Bool***
Verification of formal design:	***FS*** • ***FD*** $<->$ ***Bool***
Code inspection:	***IMPL*** $<->$ ***FLAGSET***

To show that a process may have many type definitions, let us take the example of the formal specification process (FS process). Table 2 defines the type of the FS Process as: ***RD*** $<->$ ***FS***. A process designer may design this process so that the specifier not only consults the RD, but also interviews the users to complete the specification. Then the type of the FS process would be:

$$PRE\text{-}ELICIT\text{-}REQ \bullet RD \;<->\; FS$$

Thus a process, depending on its design, may have many types; but an organization, in relation to a specific project and for a specific set of processes, will usually stick to one type definition. However, any type definition of a process will uniquely identify a process. For instance, both the preceding type definitions correspond to the FS process.

The proof follows from the construction of process definitions, and we will not elaborate it here.

The question may arise as to whether our simplistic type definitions given to processes do really correspond to the complex nature of processes. To illustrate this point, consider a type definition of the design process as: ***SPEC*** < – > ***DESIGN***. Some requirements may arise quite late, i.e., during the design process. The given type definition then cannot handle such a situation. As mentioned earlier, the type definition of a process is not unique, but an organization should follow one type definition which suits its best. To accommodate the fact that requirements may arise during the design phase, an organization can have the type definition:

PRE_ELICIT_REQ X SPEC < – > ***DESIGN***

Further, the arrival of such requirements may demand that the specification be changed accordingly. Such a situation could be handled by the type definition:

PRE_ELICIT_REQ X RD < – > ***SPEC***

A process may be composed of subprocesses. For example, consider the testing process and let it consist of unit testing and integration testing (we will ignore system testing and acceptance testing for the time being). Both are subprocesses of the testing process. A subprocess is also a process in the sense that it takes a product set as input and gives out a product set as output. Unit testing also consists of subprocesses like black-box testing and white-box testing. Whether a process should be decomposed into subprocesses or not is decided by the process designer. For instance, the designer may decide not to decompose further the processes like the black-box testing and the white-box testing. We refer to such processes as *atomic* processes. When considered from a process point of view, the inputs and outputs of subprocesses are called intermediate product sets. Usually, the output of a process becomes the input of another process (see Figure 1). Note that the subprocesses within the scope of a process need not be homogeneous; for instance, in Figure 1, in between unit testing and integration testing, there may be an integration subprocess which is not strictly a part of the testing process. An atomic process is in turn defined as a set of steps, like the steps of an algorithm. But unlike the case of an algorithm, the steps usually do not have rigorous definitions: a process executer usually has degrees of flexibility; for instance, during the code inspection process, an inspector may pass a piece of code as OK, while another inspector may find it complex and suggest simplification. During structured design, a designer has flexibility when deciding which group of nodes in a structured chart would be put inside a module. Thus, process executers often have to use their artistic skills rather than sticking to some strict and rigorous definitions.

There are three aspects to the steps of an atomic process: *what are its process steps, who will execute those process steps* and *how they will be done*? A process is defined in terms of process steps and each process step is in turn defined by a set of flexible guidelines. Further, some resources like supporting tools may be needed to execute the process steps. All of these contribute to the ‘how’ part of the questions. Each atomic process, in addition to its input and output product sets, will have its starting and ending conditions. They correspond to the ‘what’ part of the questions. The ‘who’ part will be taken care of by a person or some automatic tool. ISO/IEC 12207 follows a similar (though less formal) approach to process modeling.

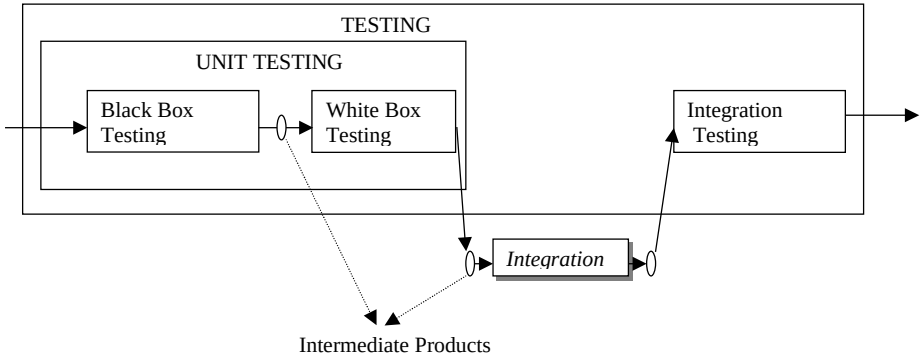

Fig. 1. Intermediate products in the testing process

Our process model is useful in many ways. The type structure offers a general view; depending on specific needs, restrictions can be imposed upon them. Consider an organization which formally verifies its code; and in order to facilitate this process, code is written in SPARK [16]. In such a case, a constraint can be imposed upon the coding process that code must be written in SPARK. Further, our process architecture takes a modular view of the process in the sense that the type of every process or subprocess is made clear at the definition level. The well-defined type structure of our process model makes the larger context of a task or activity or process step clearer.

There are three ways of looking at a process for assessment (i.e. evaluation of a process through measurable attributes). The first one is the black-box view. Under this scheme, we do not look at the internals of a process (that it may consist of subprocesses or process steps), but rather view it as a black box and try to observe its effects by analysing its input and output product sets. Under the white-box assessment scheme, we analyse the internal behaviour of the process; this may include analyzing the subprocesses that the process consists of. By subprocess analysis, we mean that we observe the intermediate products corresponding to an input product set of the main process. There are certain product independent attributes such as process cycle time, process cost, total process effort *etc.* are also very important observation entities. Process failures also need to be considered. Examples of process failures are: a supporting tool (say a testing tool in the testing process) crashing or a verifying tool failing to verify *etc.* To summarize, for assessing a process we need to have (i) its black-box view (ii) white box view and (iii) the study of the product-independent attributes discussed above.

Unlike product assessment, process assessment is non-trivial for the following reasons.

- For assessing a process, we in turn look at the input, output and the intermediate products. So assessment is *mostly* indirect in nature. (Although, products are usually assessed in this way as well, it is possible to assess software directly through code reviews, static analysis, formal verification etc., which is not the case for processes.) Further, we need to assess a number of input and output product sets and their intermediate products in order to find out something concrete about the process. In a sense, the process behaviour is the average of

the observations of a large number of input and output products and their intermediate products.

- Most of the process attributes have a dual perspective. Consider, for instance, the understandability attribute. The two perspectives are (i) the concepts of the process should itself be understandable to the process executer and further (ii) the process should make its output product sets understandable. When we consider the formal specification process, the method of creating the formal specification and the formal specification language itself must be understandable to the specifier, and further the specification process must make the formal specification understandable to its users. In conclusion, any process assessment must consider a dual perspective. Of course since we are looking at product attributes from the process quality point of view, we may miss out some important product factors which are not directly addressed by process quality factors. In order to alleviate this problem, we use major product quality models (see Section 4) as references while defining our generic process quality model.
- If a process introduces a defect in its output, then at a later period it may be discovered and the source of the defect could be traced back to the process. So, it is not only the case that time span of process assessment is long but also it may include time periods when the process is not active.

4 Our Generic Quality Model

In the last section, our process model has been described at a much higher level. In order to make it a generic quality model, its internal details need to be filled in. To do this, we take as reference the ISO 9001/9000-3 model, the ISO/IEC 12207 model, the CMM, the ISO/IEC 9126 model and the FURPS+ model. We have included the last two product quality models because while dealing with process quality we also want to put emphasis on the product aspects. Table 3 illustrates our generic model in which each factor is defined by a set of subfactors. Appendix A presents the definitions of the subfactors of the generic model. This generic model (Gmodel) could be seen as a template with three parameters:

*Gmodel(**inp-prod-set-type, out-prod-set-type, application-domain**)*

where by application domain we mean whether it is a safety-critical application, a real-time application, a business application *etc*.

The customization proceeds in two steps: (i) the substitution step and (ii) the refinement step. In the substitution step, we substitute the parameters with their actual bindings. Let us take the example of the FS process and let its type be ***RD*** < – > ***FS***. The substitution is illustrated by the expression:

Gmodel [*RD* / **inp-prod-set-type] [** *FS* / **out-prod-set-type]**

Table 3. Factors and their Subfactors in the Generic Process Quality Model

Functionality	Compliance Completeness Consistency Generality Suitability Inter-operability Security
Usability	Understandability Learnability Operability
Efficiency & Estimation	Cost/ Effort estimation Cycle Time Complexity estimation Resource Usage Schedule/ Priority estimation Process maturity
Visibility and Control	Automatic checks and feedback Progress Monitoring Improvement Measures
Reliability	Failure Frequency Fault Tolerance Recoverability
Safety	Risk avoidance
Scalability	Scalability
Maintainability	Analysability Modifiability Stability Testability Defect Trend Formal Verifiability Informal Verifiability Reusability Portability

Note that if a particular process has input product set type IP and output product set type OP, then the process takes a member of IP as input and produces a member of type OP as output. So the above expression signifies that, all occurrences of the input product set in the definitions of the factors and the subfactors in the generic model are substituted by RD (requirement document). Similarly, all occurrences of output product set are substituted by FS (formal specification). Thus, at the end of the substitution step, we have a crude definition of each of the subfactors of the process concerned.

With these assumptions, since we already know the type of the process, we also know the process name (refer to Section 2). For the *application-domain parameter*, assume that it is a safety critical application. Now, with the knowledge that we are

dealing with a FS process in a safety critical application, the refinement step refines the crude definitions that we have obtained after the substitution step. The result then will be the customized quality model for the FS process. As an illustration, the definitions of some of the subfactors of the FS process are given in Appendix B. For example, consider the first part of the subfactor 'completeness'. After the substitution step, we obtain the following definition: *the degree to which the process transforms all of the functionalities of the RD into the FS.* In the refinement step we know that it is the FS process and the application domain is the 'safety critical application'. Further FS process achieves transformation through specification; and at the RD level, a functionality is understood by a 'feature'. So the refined definition is: *the degree to which the FS process specifies all of the features (including the safety critical features) of the RD in the FS.*

5 The Measurement Plan through the GQM Paradigm

The GQM-based measurement plan starts with the specification of some goals concerning the project. Then a set of questions are formulated whose answers would in turn provide an answer to this main goal. Then one should determine what metrics should be measured so that each of the questions could be answered. Let us start with a goal such as: *assess the Functionality of the FS process.* From the quality model for FS, the functionality of the FS process is defined in terms a set of subfactors. Now the definition of these subfactors will give rise to questions in the GQM formulation, and the questions when considered in the context of formal specification, will help to formulate the metrics. Table 4 elaborates the measurement plan.

6 Benefits of the Generic Model

- The fact that certain individual processes need special attention is more or less ignored by the existing process quality models. Our generic model provides an answer to this. During the course of instantiation, the generic model takes the application-domain as a parameter. This parameter, in combination with the other parameters of a process, emphasizes those individual process attributes which need special attention. Further, we offer a systematic approach to the instantiation of the customized model.
- The relationship between the process quality model and product quality is never well-defined in existing process models. ISO 12207 suggests following ISO 9126 to ensure product quality and ISO 9001 for quality assurance [19]. Our dual perspective of process attributes makes this relationship between process quality and product quality more visible. So, our model can be used as a companion to the ISO 12207 standard.
- **The Generic Model and Process Assessment/Improvement:** the ami method [Pul 95] classifies primary goals into two categories: knowledge goals and change goals. Knowledge goals are for assessment purposes and the change goals are for improvement purposes. Under the ami approach, primary goals are transformed into a goal tree. Building the goal tree is always an art. For the refinement of a goal (or a subgoal) we must identify the products, the processes associated with the goal and also the participants who are responsible for each of

such products or processes, and further what resource the participants use or can use [Pul 95]. We claim that our quality model will help in building such a goal tree in a systematic manner. Because of the duality of the subfactors, their definitions themselves cover both the process as well as the product aspects of the goal. The context of the process easily identifies the participants associated with the subgoal. Further, the definitions of the subfactors provide enough information for identifying the associated quantitative metrics.

Table 4. A GQM based measurement plan

Goal**:**	***Object of Study*:**	FS Process
	***Purpose*:**	To assess
	***Focus*:**	Functionality
	***Points-of-view*:**	Manager/Maintainer/Specifier
Q1.	Underlying specification language?	
	***Metric*:**	Note the language
Q2.	Specification process complying with the syntax and semantics of the formal language?	
	***Metric*:**	No. of deviations
Q3.	Specification of functions matching the RD/User needs?	
	***Metric*:**	Trend of incorrectly specified features.
Q4.	Specification process uncovering all contradictions in RD? Process introducing contradictions during specification?	
	***Metric*:**	No. of contradictions found in RD No. of contradictions discovered in FS
Q5.	(i) Process specifying all features of a RD? (ii) Process failing in specifying a RD?	
	***Metric*:**	(i) No. of missing features in a FS (ii) History of failures in specifying RDs
Q6.	Process over-specifying the features of a RD?	
	***Metric*:**	No. of over-specified features in a FS (subjective assessment by the Specifier)
Q7.	Process addressing all security features in FS?	
	***Metric*:**	No. of security lapses in implementation/end-product linked to FS process.
Q8.	Process addressing all inter-operability features in RD?	
	***Metric*:**	No. of inter-operability problems in implementation/ end-product linked to FS process.

7 Conclusions and Future Work

In this paper we have presented a new process model which assigns types to the processes and the products associated with them. We have then defined a generic quality model which could be instantiated to be the quality model for any particular process, and shown how such a customization could be done in a systematic manner. One important highlight of our generic model is that it makes the relationship between product quality and process quality much clearer. This relationship is handled in an ad-hoc manner by existing models.

Like the PROFES improvement methodology, our generic model can be used to achieve better product quality. However, the PROFES methodology depends on previous experience to address process-specific issues, which can lead to neglect of issues like process defects, process scalability, process understandability etc. A detailed comparison between PROFES methodology and our generic model is a part of our future work.

The processes that we have handled in our model have the same type structure: they are relations between an input product set and an output product set. However, there are processes with complicated type structures. One such example is the *quality improvement process (QIP)* which takes a quality model and a process (say a development process), and returns a process (which should be improved). Part of our future work will be to extend our model to cover the whole spectrum of processes. We also intend to validate our generic model using industrial platforms.

Acknowledgements:

The authors wish to acknowledge the support of UK EPSRC, which has funded EMPAF (ER/L87347) and a Ph.D. studentship for this work.

References

1. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. The Encyclopedia of Software Engineering, John Wiley (1994)
2. Bazzana, G. et al.: ISO 9000 and ISO 9126: Friends or Foes? Proc. of IEEE Software Engineering Standards Symposium, Brighton, Sept (1993)
3. Bootstrap Team, Bootstrap: Europes's Assessment Method. IEEE S/W, Sept (1993)
4. Chaudron, M. et al.: Lessons from the Application of Formal Methods to the Design of a Storm Surge Barrier Control System. Proc. of FM'99, LNCS No. 1709, Springer (999)
5. Coallier, F.: How ISO 9001 fits into the Software's World. IEEE S/W, January (1994)
6. Debou, C.: Goal-Based Software Process Improvement Planning in Better Software Practice and Business Benefit (Eds. R. Messnarz and C. Tully). IEEE Computer Society (1999)
7. Dorling, A.: SPICE: Software Process Improvement and Capability Determination. Software Quality Journal, Vol. (2) (1993)
8. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous & Practical Approach. International Thomson Computer Press (1996)

9. Grady, R.B., Caswell, D.L.: Software Metrics: Establishing a Company-wide Program. Printice Hall, New Jersey (1987)
10. Grady, R.B.: Practical Software Metrics for Project Management and Process Improvement. Printice Hall, New Jersey (1992)
11. Haase, V. et al.: Bootstrap: Fine-Tuning Process Assessment. IEEE S/W, July (1994)
12. Humphrey, W.S.: Introduction to Software Process Improvement. SEI Report CMU/SEI-92-TR-7 (1992)
13. International Standard: ISO/IEC 9126, ISO, Geneva (1991)
14. ISO 9001, Quality Systems – Model for Quality Assurance in Design, Development, Production, Installation and Servicing. 2nd Ed., ISO (1994)
15. ISO/IEC 12207, Software Life Cycle Processes. 1st Ed., Aug (1995)
16. King, S., Hammond, J., Chapman, R., Pryor, A.: The value of verification: Positive experience of Industrial Proof. FM'99, LNCS No. 1709, Springer (1999)
17. Pulford, K., Kuntzmann-Combelles, A., Shirlaw, S.: A Quantitative Approach to Software management: The AMI Handbook. Addison Wesley (1996)
18. The PROFES Methodology. website http://www.profes.org
19. Singh, R.: International Standard ISO/IEC 12207 Software Life Cycle Processes, Software Process – Improvement and Practice. 2(1996) 35-50
20. Solingen, R.v., Berghout, E.: The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. Mc Graw Hill (1999)
21. Voas, J.: Can Clean Pipes Produce Dirty Water. IEEE Software, Jul/ Aug. (1999)

Appendix A: Definitions of the Subfactors of the Generic Model

(a) Subfactors of Functionality

Compliance:

(i) The degree to which the process conforms to prescribed (IEEE, ISO or company-specific) standards.

(ii) The degree to which the process conforms to its own model (i.e. it consists of a set of subprocesses and the subprocesses in turn consisting of subprocesses or sets of steps).

Consistency:

(i) The degree to which the process uncovers contradictions in the input product set.

(ii) The degree to which the process does not introduce contradictions in the output product set.

Completeness:

(i) The degree to which the process transforms all of the functionalities of the input product set into the output product set.

(ii) The degree to which the process handles all inputs with valid types.

Generality (*robustness*): The ability of the process to address by over-specifying/ over-implementing conditions which are not covered by the input product specification but are relevant to its context.

Suitability (also known as *correctness*):

(i) The degree to which the process makes the functionalities of the output product set match accurately those in the input product set.

(ii) The degree to which the implementation of the process accurately matches its own specification (any gap between what the process does and what it is supposed to do?)

Security: The degree to which the process addresses the security issues of the final product against hostile environments (too important for network applications involving critical activities such as e-commerce)

Inter-operability: The degree to which the process contributes to the inter-operability (the ability of the product to interact with specified systems) of the final product.

(b) Subfactors of Usability

Understandability:

(i) The effort with which a typical process executer (any of Analyst/ Specifier/ Developer/ Programmer/ Maintainer) understands the logical concepts of the process.

(ii) The degree to which the process contributes to the understandability of its output product set and that of the end-product.

Learnability:

(i) The effort required for a typical process executer to learn to use the process.

(ii) The degree to which the process makes the output product set/ the end-product easy to use (through informative help messages, good user interface etc.)

Operability:

(i) The effort required for a typical process executer to execute the process with a level of confidence (this could be inferred from the learning curve of the process executer).

(ii) The degree to which a process contributes to the operability of the output product set and the end-product (through informative help messages, guidelines).

(c) Subfactors of Efficiency & Estimation

Cost/ Effort Estimation: The degree to which the cost/ effort of process execution remain within a specified range and the ability of the process to support their estimations.

Cycle Time estimation: The degree to which a process meets its expected cycle time and the ability of the process to support its estimation.

Complexity estimation: The ability of the process to support the prior estimation of various forms of complexity (Computational Complexity, Communication Complexity, Structural Complexity, Cognitive Complexity etc [8, 9]), and the degree to which the estimates are accurate.

Schedule/ Priority estimation: The priority of various stages in the process and the ability of the process to support their estimations and scheduling.

Resource Estimation: The degree to which a process keeps its resource usage in a specified range, and its ability to support their estimations.

Process Maturity: The CMM maturity level and/or any ISO certification of the organization.

(d) Subfactors of Visibility & Control

Progress Monitoring: The ability of the process to facilitate monitoring of its progress at any point of time during the process execution to show that progress so far has been correct and effective. (e.g. work product analysis [10], PERT charts etc.)

Automatic Feedback: The ability of the process to provide (or support in providing) feedback data and to support corrective actions if necessary (e.g. Automatic work product analysis tools [10]).

Improvement Measures: The ability of the process to support the analysis of the feedback data in combination with the data of previous runs and improve itself, or result in the improvement of a sibling process, continuously (e.g. FS process improving the testing process [4]).

(e) Subfactors of Reliability

(i)The degree to which the process prevents failures in the final product (a process relying on unsafe unsafe features of C may lead to failures).

(ii)The degree to which the process is reliable itself. In that case, we will have the sub-factors:

- **Failure Frequency:** The number of (and the interval between) failures encountered during the execution of the process. The failures are due to defects in the design/ implementation of the process itself. (e.g. an automatic tool crashing or taking too much time)
- **Fault Tolerance:** Whether the process can still continue in presence of faults/ failures in the process itself (e.g. use of redundant tools).
- **Recoverability:** Whether the process can attain its level of operation after a process failure has been addressed.

(f) Subfactors of Scalability

(i) The degree to which a process makes its output or the end-product perform in an expected way (in terms of time and resource usage) in face of a large problem context (e.g. A robust and complete design of a database system can make it handle a large problem).

(ii) The degree to which the process maintains its efficiency level in terms of time, cost and resource usage in handling a problem of larger dimension.

(f) Subfactors of Safety

(i) Depending on the underlying application, the input product set may define safety conditions which could take the form of (a) liveness (b) degraded performance and (c) limiting damage from failures. The degree to which the process addresses them (say, through introduction of fault-tolerance).

(ii) The process must not introduce unsafe features that may result in hazards occurring (a compiler, for optimization purposes converting static memory to dynamic memory [16]).

(g) Subfactors of Maintainability

Analysability: The effort with which a typical process maintainer can analyse the cause of a fault, process failure or unexpected feedback data. A fault could be one of the following categories.

(i) the fault is discovered during the process execution and the fault may be with the process itself, or it may be with the input product set.

(ii) the fault is discovered at a later point in time, and the cause of the fault is linked to the process.

Modifiability: The effort with which a Maintainer addresses failures, detection of faults and unexpected feedback data during the process execution; or faults discovered at a later time but linked to the process.

Stability:

(i) The degree to which addressing a process fault adversely affects the process itself (say, process efficiency or properties like process consistency).

(ii) The frequency of changes done to the process (less the number of changes, more is the stability).

Formal Verifiability: Effort with which properties like consistency, correctness and invariant conditions of the transformed product could be formally verified during process execution.

Informal Verifiability: Effort with which properties like completeness, generality etc. of the product under transformation could be informally verified during process execution.

Testability:

(i) The degree to which the process could be validated itself (success history of its output products).

(ii) The degree to which the process contributes to the testability of its output or the final product (say, for generating an adequate test suite).

Defect Trend:

(i) The trend of defects that are observed in the process itself (defects linked to the process – during or after process execution).

(ii) The degree to which the process detects the defects or deficiencies in the product set under transformation so that defects in the output product set and the final product are minimal.

Reusability:

(i) The degree to which the process contributes to the reusability of its output product .

(ii) The degree to which components of the process could be reused in a different context (testing process of one organization could be reused in another organization).

Portability: The degree to which the process contributes to the portability of the end-product; i.e., the process should facilitate the migration of the product to a different environment.

Appendix B: Some Sufactors of the FS Process

Completeness:

(i) The degree to which the FS process specifies all of the features (including safety critical) of the RD.

(ii) To what extent the specification process can handle all RDs with valid types.

Consistency:

(i) The degree to which the FS process uncovers contradictions in the RD.

(ii) The degree to which the FS process does not introduce contradictions in the FS.

Security: The degree to which the FS process specifies the security issues (unauthorized access to program and data) of the final product against hostile environments (too important for network applications involving critical activities such as e-commerce, e-banking etc.)

Understandability:

(i) The degree to which the syntax and the semantics of the specification language, and the logical concepts of Specifying are understandable to the Analyst, Specifier, Developer, and the Maintainer.

(ii) The degree to which the FS process contributes to the understandability of the FS (say, through comments).

Cost/ Effort Estimation: The degree to which the cost/ effort of the FS process are kept within allowable limits, and the ability of the FS process to support their estimations.