

Quality Modelling for Software Product Lines

Adam Trendowicz, Teade Punter

Fraunhofer IESE, Sauerwiesen 6, D-67661 Kaiserslautern, Germany,
<trend, punter>@iese.fhg.de

Abstract. In today's embedded software systems development, non-functional requirements (e.g., dependability, maintainability) become more and more important. Simultaneously the increasing pressure for developing software in shorter time and at a lower cost pushes software industry towards product lines solutions. To support product lines for high quality embedded software, quality models are needed. In this paper we investigate to which extent existing quality modelling approaches facilitate high quality software product lines. First, we define several requirements for an appropriate quality model. Then, we use those requirements to review the existing quality modelling approaches. We conclude from the review that no single quality model fulfils all of our requirements. However, several approaches contain valuable characteristics. Based upon those characteristics we propose the Prometheus approach. Prometheus is a goal-oriented method that integrates qualitative and quantitative approaches to quality control. The method starts quality modelling early in the software lifecycle and is reusable across product lines.

1. Introduction

More and more software is developed as a part of embedded systems. Embedded software already covers the majority of software market and its production explosively increases. In consequence quality demands, especially in terms of non-functional requirements (e.g., dependability, maintainability), are of high importance and continuously rise. At the same time, software engineers are increasingly under pressure to develop new systems in less time and at lower cost.

Software product lines paradigm (SPL) supports software organizations with solutions to reduce the cost and time of software development. SPL approach is based on the observation that most of new software systems are similar to already existing ones. They are either different versions or next releases of the same baseline product. SPL provides efficient mechanisms to develop libraries of software components and to reuse them in the creation of similar software systems with many variations. Nevertheless, SPL leaves software developers without effective means to assure that software meets certain non-functional requirements.

Over the years researchers proposed and successfully brought into practice a considerable number of variety quality assurance techniques applicable for software product lines. Yet, such methods like component certification and regression testing

are applicable only in late phases of the software lifecycle where cost of every modification is very high (ten times higher every later phase). In order to lower the cost of quality, a methods for early evaluation and control of quality have to be employed. Approaches based on quality models cope with this requirement. According to [21] the term **quality model** is defined as “the set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality”.

In this paper we investigate to which extent existing quality modelling approaches facilitate high quality (in terms of non-functional requirements) of software product lines. First, we define several requirements for an appropriate quality model. Then, we use those requirements to review the existing quality modelling approaches. The conclusion from the review is that no single quality model fulfils all of our requirements. However, several approaches have valuable characteristics. Based upon those characteristics we propose the *Prometheus* approach. Prometheus is a goal-oriented method that integrates qualitative and qualitative approaches to quality control. The method starts quality modelling early in the software lifecycle and is reusable across product lines.

In the following section we define requirements towards quality model for software product lines. Section 3 presents the review of existing quality models from the perspective of defined in section 2 requirements. Section 4 presents the Prometheus approach. Finally, section 5 concludes the paper with a summary of the results and future research perspectives.

2. Properties of quality model for SPL

Characteristics of software product lines as well as experience with several existing quality modelling approaches have guided us to define three main requirements for an appropriate quality modelling: flexibility, reusability and transparency.

Flexibility - A quality model should be flexible because of the context dependency of software quality. There are several quality contexts: company context, project context and process context. *Company context* includes unique characteristics of specific software company where the model is used. Flexible quality modelling approach should be applicable across different companies. However, employment of the approach in different companies should result with unique quality models that reflect unique characteristics of each single company. *Project context* combines unique characteristics of particular software project like its domain (e.g., web application, embedded system) or different views on the quality represented by different project stakeholders. For example, system end-user can think about software reliability in terms of failure density whereas software developer can also notice the relation between the reliability and software design complexity. Flexible quality modelling approach should be applicable for any project domain and incorporate views (on quality characteristics and their relationships) of all relevant project stakeholders. *Process context* reflects characteristics of software development process like its stability or availability of measurable objects in different process phases. Flexible quality model should not assume stabile process. Modelling approach should

allow creating the model tailored to company-specific characteristics of the development process. Postulating stable process would make a modelling approach inapplicable in most of software companies due to lack of stable processes.

Very important issue in quality modelling are phases of software lifecycle to which the model is applicable. In essence, the more effective is the quality modelling the earlier in the software lifecycle it could start and the more phases it embraces. From the perspective of controlling the quality, early quality evaluation allow in time identification and elimination of potential quality problems. For instance, elimination of design defect during software operation could cost a hundreds as much as the defect would be identified and removed already in the design phase.

In early phases of software lifecycle hardly few measurable items are available. Therefore the flexible approach should integrate all characteristics of software project environment that influences quality of software product. Those could be product characteristics (e.g., design complexity), process characteristics (e.g., inspection efficiency) as well as resource characteristics (e.g., designer experience). To improve the model accuracy it should also take advantage from the people experience and besides quantitative (measurement-based) data, it should cope with qualitative input e.g., expert's assessments.

During subsequent phases of development both a software system as well as the whole software project environment are the subjects of continues change. As the project evolves, new products are developed, new processes are applied and the more measurable artefacts are available. In order to control the quality of software product the quality model should evolve in parallel to software changes. The modelling approach should cope with missing data as well as allow easy re-estimation of quality evaluations, as the new and more precise data appear.

Reusability – The need for profiting from past experience has led software development in the direction of product lines. In order to assure the development of high-quality products, quality models should also follow this paradigm. Instead of “re-inventing the wheel” every next software project, the quality models should allow the reuse of quality experience packaged in the existing quality models across other projects. Dependent on the projects similarity-level quality model should support the reuse of measurements data as well as quality characteristics and their relationships. On the one hand reusable quality modelling will reduce time and cost of quality assurance. On the other hand it will improve accuracy and efficiency of quality evaluation as subsequent experience can contribute to improving existing models. For example, the same model could be reused every new release of the same software product and experience from previous release could be incorporated into improved model used in the next release.

Transparency - A quality model should provide the rationale of how certain characteristics are related to others and how to identify their sub-characteristics. Transparency of a quality model does also mean that the meaning of the characteristics and relationships between them are clearly (unambiguously) defined. People involved in model development and application should understand it to gain knowledge from it as well as to identify redundancies or contradictions among quality characteristics. An example of contradiction could be modularisation in object-oriented software. It improves software reliability but usually at some cost in

efficiency. The model should also allow the project stakeholder directly interfering in the model structure to modify it if needed.

3. Review of existing quality models

According to [13] there are two kinds of approaches to model product quality: *fixed-model* and *define-your-own-model*. Fixed-model solution provides fixed set of qualities so that identification of customer-specific characteristics results with a subset of those in a published fixed model. To control and measure each quality characteristic, the characteristics, measures, and relationships associated to the fixed model are used. Examples of such models are presented in: [2], [27], [4], [20], [9], [16], [29] and [18]. They contrast the define-your-own-model approach where not a specific set of quality characteristics is defined, but rather – in cooperation with the user – a consensus on relevant quality characteristics is identified for a particular system. These characteristics are then decomposed (possibly guided by an existing quality model) to measurable quality characteristics and related metrics. The relationships between quality characteristics and sub-characteristics could be then defined either directly by project stakeholders (*directly-defined model*) or automatically generated (*indirectly-defined model*). Directly-defined models have the form of dependency graphs and examples of such approaches are presented in: [15], [3], [1], [19], [26] and [11].

Indirectly-defined models result from application of various techniques, so that software project stakeholders can influence the quality model by choosing the technique and its parameters. However they have no direct influence on the output quality model. Quality relationships represented in such models are often so complex that project stakeholder has difficulties to understand them. The main domains from which such models mainly come are mathematics and artificial intelligence. There are also some known attempts to employ other approaches like multi criteria decision aid [12]. The examples of mathematical models are: multiple regression models [7], Alberg diagrams [28] and logistic regression models [23]. Typical artificial intelligence approaches are: decision and classification trees [24], genetic algorithms [6], neural networks [22], case-based reasoning [10], data mining [25], and fuzzy expert systems [32].

Fixed-model approaches miss the *flexibility* requirement. They define a constant set of quality characteristics and relationships between them. However it is unrealistic to assume that it is possible to define a prescriptive view of necessary and sufficient quality characteristics to describe quality requirements at every company, for every project and every stakeholder. Probably there is some amount of quality characteristics and relationships universally true for all organizations and projects but most of them differ from organization to organization and from project to project. Horgan [17] tries to identify such universal characteristics to compare quality across projects. He introduces Key Quality Factors (KQFs) as common for every project and every company. However, KQFs are high-level quality characteristics like maintainability or correctness already known from ISO9126 [20] or McCall's [27] models. Such approach limits the comparability of project quality to only high-level

characteristics. Furthermore, the level of reusability of quality experience gained in the past projects and stored in such universal models depends on the level of projects similarity and is usually limited by the lack of indicators of similarity.

The common problem of fixed-model approaches is that they are limited to quantitative (measurement-based) and product-related data, whereas in early stages of software life cycle hardly few measurable products are available. Some latest fixed-model methods (e.g., [18]) broaden the scope of measurement on processes and resources but are still unable to profit from qualitative data like for example expert's assessment.

Fixed-model approaches miss transparency in a way that they impose the model architecture without providing the logic behind it and without describing how the higher-level characteristics are decomposed to lower level sub-characteristics and metrics. They also do not provide guidelines how to use measurement results to evaluate software product quality. Some models seem not to be even consistent as how the characteristics are decomposed. In addition the distinction between particular quality characteristics according to their definitions is not clear. For example, the average developer will not be able to distinguish between characteristics like interoperability [27], adaptability [20], and configurability [16] as they might be regarded as being identical.

Define-your-own-model approaches address some of transparency and flexibility weaknesses of fixed-model approaches. They do not impose any prescriptive set of characteristics so that product-, process- and resource-related characteristics could be combined. Directly-defined models like SQUID [26] provide description how to decompose high-level quality characteristics into lower-level sub-characteristics and metrics. However, they say hardly anything about how to compose measurement data and propagate it into quality prediction. The exception is approach presented in [11] that uses Bayesian Belief Nets (BBNs) to propagate quality assessments from the graph-based model.

Indirectly-defined models also cope with combining measures into quality evaluations, however they face some transparency and flexibility problems. For example, statistical approaches deal with the problem of composing metrics into quality prediction using for instance regression equations. Nevertheless, Ohlson and Alberg [28] claim that character of measurement data allows only ordering modules according to their quality rather than giving objective quality assessments. They also point that many statistical models that assume normal distribution are applied to model software quality whereas in many cases such assumption cannot be made. As far as the reusability of statistical models is concerned only general conclusions coming from multiple applications of the model are usually reused as guidance within other projects. For instance, Briand and Wuest [7] state that coupling between software modules indicates quality risks, but the same conclusions cannot be made regarding cohesion.

The quantitative character of the input for statistical models limits their application in early stages of software life cycle where only few measurement objects are available. Companies missing measurement programs can have difficulties with efficient application of mathematical models in any stage of software lifecycle. However statistical models, unlike directly-defined ones, do cope with redundant and contradicting quality characteristics. The Principal Component Analysis could be

employed to identify non-redundant set of characteristics and metrics. Some more interactive solution could be found in one of recent approaches to model non-functional requirements: QARCC [5] and NFR-Framework [8]. In those approaches, user supported by the automated tool identifies overlaps in the graph-based model.

Recent experiments with artificial intelligence approaches have not also brought any breakthrough solutions. Despite possibility of combining qualitative and quantitative data for more exact early evaluations their ability to reuse quality experiences across projects is still limited by projects similarity. Machine learning models like decision trees or neural networks require significant amount of training data to achieve satisfactory accuracy of quality estimations. Even then the result of evaluation could be of very low accuracy when evaluated project differ substantially from the past ones. In addition, the structure of a neural network lacks transparency.

Important problem, common for all kinds of quality models, is still the lack of comprehensive guidelines how to produce a consensus view of quality characteristics and their relationships [17], as well as the inability to reuse quality experiences across different projects and companies to improve efficiency of quality estimation.

Last issue is the tool support. As the quality models are to support software practitioners and minimize quality assurance effort, automated tools are required. Most of the existing quality approaches includes dedicated software tools. However, some of those tools likewise SQUID toolset can only provide limited assistance. For the approaches that employ well-known techniques, like Bayesian Belief Networks or decision trees, tools supporting those methods could be reused for the purpose of quality modelling.

Based upon presented above review we conclude that despite the variety of existing quality models, many of them hardly cover the requirements defined in section 2. Most of them just replicate the weaknesses and deficiencies of the others. We perceive the SQUID approach as an improvement with regard to flexibility, reusability and transparency of quality modelling. However, the specification phase (how to achieve the actual quality model for the product line) of SQUID can be improved by the Goal Question Metric (GQM) paradigm. Besides, Bayesian Belief Nets (BBN) would add the possibility to quantify relationships as well as integrate within the model qualitative and quantitative data. BBN provides also an efficient mechanism for combining measures (packaged in the model) into quality estimates.

4. Prometheus: an approach to model quality in SPL

This section presents the Prometheus¹ approach to quality modelling. Prometheus combines characteristics of several existing models to meet requirements defined in section 2.

The Prometheus approach consists of three phases: specification, application and packaging. During *specification* a quality model is developed. In *application* phase the model is employed to evaluate specified non-functional requirements. The *packaging* phase serves for gathering experience gained during model application in

¹ Probabilistic Method for early evaluation of NFRs

order to improve the model and reuse it in other projects. The concepts for packaging and reuse are presented in detail in [31].

The focus of the current paper is on the specification phase. The specification of Prometheus model consists of the following activities:

- define quality goals
- specify quality characteristics (model content)
- specify relationships (model structure)
- review the model
- operationalize the model.

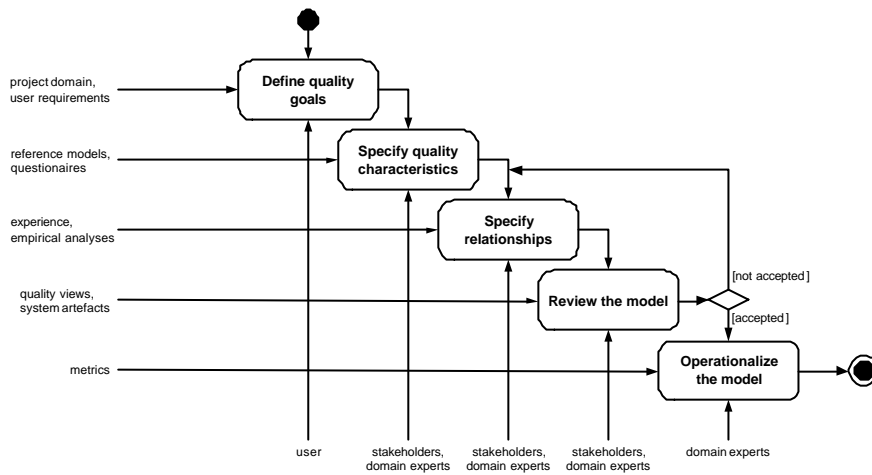


Figure 1 The activities during the Specification phase

Define quality goals – At this stage system user specifies software quality goals. Our experience shows that implicit reasons like “to judge the maintainability of the code” are often applied to start modelling and evaluation of software quality. However, the only way to keep such statements verifiable is to make them operational i.e. measurable. To support operationalization of quality goals we propose the measurement goal template (MGT).

Dimension	Definition	Example
Object	What artefact is analysed?	Analyse the software system
Purpose	Why is the object analysed?	for the purpose of evaluation
Quality Focus	What characteristics of the object are analysed?	with respect to maintainability
Viewpoint	Who will use the data collected?	from the viewpoint of system user
Context	In which environment does the analysis take place?	in the context of company X and development project Y

Table 1 Measurement goal template

The measurement goal template (table 1) does not only address the characteristics that are to be covered by the evaluation but also other subjects of quality modelling like object (artefact) and stakeholder (viewpoint). MGT supports flexibility of the quality model by adjusting it to the context of a particular software project.

Our experience in applying the Goal Question Metric method shows that although careful formulation of the goals at the beginning is necessary, they may have to be refined during subsequent GQM activities (definition of questions and metrics) due to new insights. The goal formulation is therefore conducted iteratively and it works as a baseline for the evaluation. The quality goals are defined by the system users. However, our experience show that also other stakeholders that are related to the development project should be involved to ensure the acceptance of the evaluation.

Specify quality characteristics – The second specification activity is the refinement of the quality goals into a set of quality characteristics and sub-characteristics. This procedure goes on as long as there is a set of measurable sub-characteristics defined. A sub-characteristic is measurable when it is possible to attach it to particular component of product line and to define one or more corresponding metrics. For example, size can be sub-characteristic of maintainability and it could be attached to source code, and measured with LOC (lines of code) metric.

Our activity of refining quality characteristics has similarities with the six-step methodology by Franch and Carvallo [14]. However, we do not reuse one of the existing fixed models (e.g., ISO 9126). Instead we organize an interview sessions with domain experts. During these sessions we talk to the experts about defined quality goal and quality characteristics that might influence it. As a support we use questionnaires, case studies as well as existing quality models. Nevertheless, we try not to present those materials to the experts explicitly in order to avoid suggesting any particular solutions. They should only drive the discussion. This process is inspired by our experiences in applying the GQM-approach [30].

Refining the quality goal into quality characteristics is influenced by the perception of the *stakeholders* involved in the product line. For example, system end-user can think about software reliability in terms of failure density whereas software developer can also notice the relation between the reliability and software design complexity. Stakeholders are already regarded when specifying the goal according to the measurement goal template. However, their specific perception has to be considered during the refinement process.

Specify relationships – After defining quality characteristics and measures, relationships among them are defined. The relative importance of the characteristics is determined as well as contradictions and redundancies among them are detected. Two types of relationship are to be distinguished. First is the *decomposition* relationship that specifies how high-level characteristics can be decomposed into more detailed sub-characteristics. For example, dependability can be decomposed (consists of) reliability and performance. Second is the *influence* relationship that defines which sub-characteristics have influence on the value of other characteristics. For instance, the complexity of source code influences software maintainability.

Review model – After defining the content (characteristics) and the structure (relationships) of the model we review it with respect to implementation feasibility. First, we determine the consensus of all involved stakeholders about defined model (see [30]). Then we check if specified characteristics can be measured in practice and

if the cost matches to the expected results. For instance, the availability of appropriate information sources and measurement tools is assessed. We also make sure that the model is not too complex. The trade-off between effort on model development and results of its application requires simplicity of the model. Therefore, only the most influential and easy to measure characteristics should be included. Asking experts for weighting the relative importance of quality characteristics is one way to identify the set of most relevant ones. Moreover, Bayesian Belief Nets imposes additional limitations on the complexity of model structure. Due to the cost of relationships quantification each characteristic should be influenced by (or decompose to) no more than three sub-characteristics.

Operationalize model – This step consists of model quantification. It regards both model content (characteristics) as well as model structure (relationships). Despite the term “quantification”, metrics assigned to characteristics and relationships could also have a qualitative character. In order to have possibility to quantify relationships as well as combine within the model qualitative and quantitative data we apply Bayesian Belief Nets (BBN).

A *Bayesian Belief Net* [11] is a graphical network, similar to the quality model (structure as well as content) that contains a set of probability tables, which together represent probabilistic relationships among variables (characteristics).

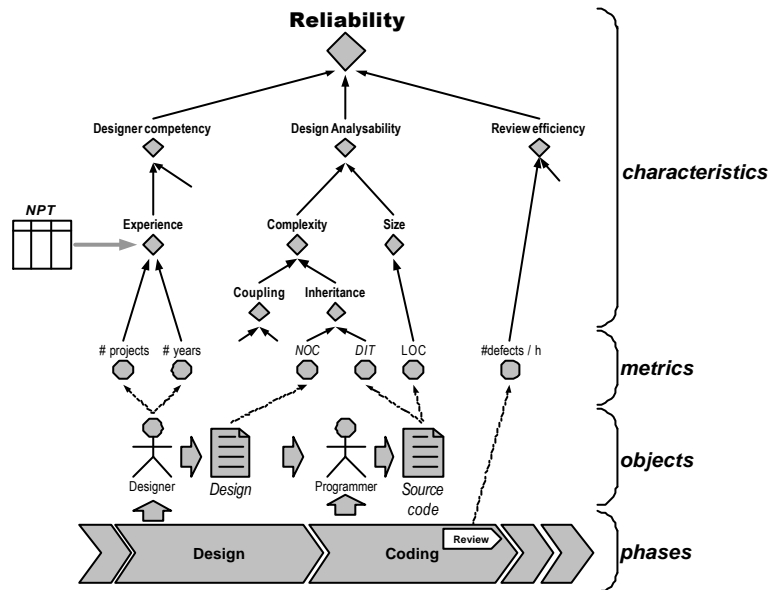


Figure 2 Application of Bayesian Belief Network to model software reliability

Figure 2 provides an example of how to apply BBN in combination with the quality model for the purpose of software reliability evaluation. We describe designer experience by discrete node “Experience” that can have three values “low”, “medium” and “high” (in contradiction, node describing size of code might be continuous). Dependent on the capabilities and needs of specific company it is

possible to represent as discrete the variables, which are in principle continuous (e.g., by dividing the variable scale into ranges and next into categories). For example, we might represent size of code as “small”, “medium” and “large”.

Each BBN node could be described either with a Node Probability Table (NPT) or mathematical formula. The NPT covers conditional probabilities of the discrete node’s state basis on the states of sub-nodes that influence this node. A mathematical formula is used for continuous nodes and calculates the node state from the actual states of sub-nodes. The NPT and formulas quantify relationships among characteristics and are defined either based on experience or some formal analysis (e.g., regression analysis). Such combination of quantitative and qualitative data is one of benefits of BBN. Table 2 presents the exemplary Node Probability Table.

# projects		few			many			lot		
# years		few	many	lot	few	many	lot	few	many	lot
experience	Low	0,70	0,50	0,33	0,50	0,33	0,20	0,20	0,33	0,70
	Med	0,20	0,30	0,33	0,30	0,33	0,30	0,30	0,33	0,20
	High	0,10	0,20	0,33	0,20	0,33	0,50	0,50	0,33	0,10

Table 2 Example of Node Probability Table

The probabilities in NPT are provided by experts and could be just subjective estimations or results of the analysis based on past projects. In the example NPT the bolded value of *0.20* means that if designer has performed *few projects* similar to the current one and has been working a designer for *many years*, then the probability that he has *high experience* to perform current project is equal to *0.2*. After finishing the project, if it occurs that the estimation was wrong (designed was very professional), the NPT could be improved (probabilities are modified) for the future reuse.

Besides combining different kinds of data, BBN facilitates also merging more than one view in one model. When stakeholders propose different characteristics, adding new nodes related to them solves the problem. In case when stakeholders differ regarding probabilities we can combine their assessments using weighted average or using random sampling method like for instance Monte Carlo.

The Bayesian probability has an additional advantage that tool support can be easily found (e.g., Analytica, Hugin, Netica, MSBNx).

5. Conclusions

In this paper we have argued that insufficient applicable quality modelling approaches are available to evaluate non-functional requirements (NFRs) of software product lines. Therefore we need quality models that are: *flexible*, to be tailored to specific organization and project; *transparent*, to allow clear insight into their rationale as well as the meaning of the characteristics and relations among them; *reusable*, to be reused and improved across project lines.

After reviewing existing quality modelling approaches we concluded that no single quality model copes with all of our requirements. However, several approaches meet some of those requirements.

Base on those models we propose a *Prometheus* approach, to model quality within software product lines. Prometheus is basically a way of modelling quality that integrates quantitative (measurement-base) and qualitative approaches to quality modelling and combines different contexts of software quality like: individual views on quality (developer, user, etc.) or different evaluation objects (processes, products, resources).

Our approach is particularly interesting for projects and organizations that want to perform early evaluation in a context of SPL even if they do not already have measurement data. Prometheus enables them to start quality evaluations early in the development process, learn effectively over several product variances/releases and refine the quality model through subsequent projects.

We now apply the approach during demonstrator work packages in the ITEA EMPRESS² project. The subject of our future research is empirical evaluation and improvement of the Prometheus approach.

References

1. J.D. Arthur, R.E. Nance, *A Framework for Assessing the Adequacy and Effectiveness of Software Development Methodologies*, Proceedings of the 15th Annual Software Engineering Workshop, Greenbelt, Md, 1990.
2. M. R. Barbacci, M. H. Klein, T. Longstaff, C. Weinstock, *Quality Attributes*, Technical Report CMU/SEI-95-TR-021, SEI CMU, Pittsburgh, 1995
3. V.R. Basili, *Software modeling and measurement. The Goal-Question-Metric paradigm*, Computer Science Technical Report Series NR: UMIACS-TR-92-96, 1992
4. B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, M.J. Merritt, *Characteristics of Software Quality*, North Holland Publishing Company, 1978
5. B.Boehm, H.In, *Identifying Quality-Requirement Conflicts*, IEEE Software, 1996
6. S. Bouktif, B. Kégl, H. Sahrroui, *Combining and Adapting Software Quality Predictive Models*, 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002), 2002
7. L.Briand, J.Wuest, *Empirical Studies of Quality Models in Object-Oriented Systems*, Advances in Computers, 56, 2002
8. L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000
9. R.G. Dromey, *A Model for Software Product Quality*, IEEE Transactions on Software Engineering, 21:146-162, 1995,
10. K. El Emam, S.Benlarbi, N. Goel, *Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components*, Journal of Systems and Software, 2001
11. N.E.Fenton, P.Krause, M.Neil, *A Probabilistic Model for Software Defect Prediction*, accepted for publication IEEE Transactions Software Eng., 2001,
12. N.E. Fenton, M. Neil, *Making Decisions: Using Bayesian Nets and MCDA*, Knowledge-Based Systems 14:307-325, 2001,
13. N.E. Fenton, S.L.O. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, published by International Thomson Computer Press, 1996,

² Empress stands for: Evolution Management and Process for Real-time Embedded Software System; <http://www.empress-itea.org/index.html>

14. X. Franch, J. P. Carvallo, *A Quality-Model-Based Approach for Describing and Evaluating Software Packages*, IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany, pp. 104-111, 2002
15. T.Gilb, *Principles of Software Engineering Management*, Addison Wesley, Reading MA, 1988,
16. Robert B. Grady and Deborah L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987,
17. G. Horgan, S. Khaddaj, P. Forte, *An essential views model for software quality assurance*, from Project Control for Software Quality, Editors, R. Kusters, A. Cowderoy, F. Heemstra, E. van Veenendaal. Shaker Publishing, 1999,
18. L.E. Hyatt, L.H.Rosenberg, *A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality*, European Space Agency Software Assurance Symposium and the 8th Annual Software Technology Conference, 1996,
19. IEEE Standard for Software Quality Metrics Methodology, 1998,
20. ISO/IEC 9126 International Standard, Software Engineering - Product quality - Part 1: Quality model, 2001,
21. ISO/IEC 14598 International Standard, Standard for Information technology - Software product evaluation - Part 1: General overview
22. T.M. Khoshgoftaar, E.B. Allen, *Neural networks for software quality prediction*, In W. Pedrycz, J.F. Peters, editors, Computational Intelligence in Software Engineering, 16:33-63 of Advances in Fuzzy Systems - Applications and Theory. World Scientific, 1998,
23. T.M. Khoshgoftaar, E.B. Allen, *Logistic regression modelling of software quality*, International Journal of Reliability, Quality and Safety Engineering 6 (4):303-317, 1999,
24. T.M. Khoshgoftaar, E.B. Allen, *Predicting fault-prone software modules in embedded systems with classification trees*, In Proceedings: 4th IEEE International Symposium on High-Assurance Systems Engineering, 1999
25. T.M. Khoshgoftaar, E.B. Allen, W.D. Jones, J.P. Hudepohl, *Data mining for predictors of software quality*, International Journal of Software Engineering and Knowledge Engineering 9 (5): 547-563, 1999,
26. B.Kitchenham, S.Linkman, A.Pasquini, V. Nanni, *The SQUID approach to defining a quality model*, Software Quality Journal 6:211-233, 1997,
27. J.A.McCall, P.K.Richards, G.F.Walters, *Factors in Software Quality*, RADC TR-77-369, Vols I, II, III, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055, 1977
28. N. Ohlsson, H. Alberg, *Predicting fault-prone software modules in telephone switches*, IEEE Transactions on Software Eng. 22 (12): 886-894, 1996,
29. P. Oman, J. Hagemester, D. Ash, *A Definition and Taxonomy for Software Maintainability*, Software Engineering Test Laboratory Report #91-08-TR, University of Idaho, ID 83843, 1992,
30. T. Punter, *Experiences in specifying perceived software quality*, in: Proceedings of Workshop on Empirical Studies in Software Engineering, Stuttgart, Fraunhofer IRB Verlag, 2003.
31. T.Punter, A.Trendowicz, P.Kaiser, *Evaluating Evolutionary Software Systems*, 4th International Conference on Product Focused Software Process Improvement PROFES 2002,
32. Z. Xu, T.M. Khoshgoftaar, E.B. Allen, *Application of fuzzy expert systems in assessing operational risk of software*, Information and Software Technology, 2003.