Enhancing Bayesian Network Model for Integrated Software Quality Prediction

Łukasz Radliński
Institute of Information Technology in Management
University of Szczecin
Szczecin, Poland
lukrad@uoo.univ.szczecin.pl

Abstract—**A Bayesian network model for integrated software quality prediction, proposed in earlier study, has potential in supporting decision makers in software projects. However, it also has some disadvantages limiting its use. The aim of this paper is to overcome these limitations by enhancing the original model in three ways: (1) incorporating project factors, (2) adding subnets with detailed process factors, and (3) modeling integration of software components or sub-systems. These enhancements significantly improve the analytical usefulness of this predictive Bayesian network model.**

Keywords-Bayesian network; decision support; process factors; project factors; quality factors; software quality.

I. Introduction

The quality of software is a very important aspect of software project. Thus, software quality have been extensively studied since the turn of 1960's and 1970's [1][26]. While most of these studies have been focused on software defectiveness [6], some researchers also investigate selected features of software quality such as reliability [16][18], maintainability [25] or usability [2]. Although such studies are very useful contributions to software engineering discipline, they typically focus on a single feature of software quality.

Project decisions related to software quality require support from analytical and predictive models. It is possible to make decisions based on output from models focusing on a single quality feature. The majority of existing approaches involving techniques, such as case-based reasoning, decision trees, multiple regression, are not feasible for this purpose because they focus on a single output. Important decisions, influencing the whole project and its environment, should be made after deeper analyses of possible effects involving multiple outputs. Performing such analyses can be supported by a simulation model that can handle multiple outputs and various types of relationships. In our experiences with using empirical data in software companies, we found that the companies do not have data of required volume and granularity to automatically generate/learn the model purely from data. Therefore, we propose using expert-driven Bayesian networks (BNs) as a formal representation for such simulation model. Section III provides more details on motivations for using BNs.

Earlier studies [20][21] proposed a BN model for integrated software quality prediction. Preliminary experiments revealed that this model may be a useful simulation tool for decision makers in software projects. The main aim of this paper is to develop an enhanced version of this predictive model. The main contributions of this paper are the following enhancements of the original model:

- Incorporating project factors that describe the nature of a project – as a result, an enhanced model can be reused for different types of software projects, rather than for a single project type defined upfront;
- Adding subnets with detailed process factors influencing overall process quality – this may be useful where direct assessment of the level of process quality is difficult or where it is useful to perform simulations using detailed process factors;
- Modeling an integration of software components or sub-systems into larger software products – this extends the usability of the model for different parts of a software product and their integration.

This paper is organized as follows: Section II defines software quality and its factors according to ISO standards. Section III discusses related work. Section IV summarizes original BN model. Section V presents proposed enhancements to the original model. Section VI provides plans for model calibration and validation. Section VII draws conclusions and discusses future work.

II. Software Quality Factors

Detailed analysis of software quality requires investigating a variety of quality factors. This paper is based on the breakdown of software quality proposed in ISO 250xx series of standards [11][12], which superseded an older 9126 standard [13]. On the first level there are 11 quality features: compatibility, flexibility, functional suitability, maintainability, operability, performance efficiency, portability, reliability, safety, security, and usability. Then each feature is decomposed into a set of sub-features. For example, reliability has five sub-features defined: availability, fault tolerance, maturity, recoverability, and reliability compliance. On the third level there are measures describing specific sub-features. These measures should be carefully selected depending on the purpose of analysis and environment where such model will be used. In this paper, a term 'quality factors' refers to all levels of software quality, i.e., features, sub-features and measures.

III. Related Work

In previous research, a variety of statistical and machine learning techniques have been used for quality prediction. The most popular are: multiple regression (MR), case-based reasoning (CBR), decision trees (DT), random forests (RF), rule induction (RI), support vector machines (SVM), system dynamics (SD), neural networks (NN), and Bayesian networks (BN). We have investigated various features of popular and well established techniques. This analysis helped in selecting the technique that would be the best suited for our model for software quality prediction.

Table I illustrates how various features of modeling, simulation and prediction correspond to different techniques. This comparison has been developed based on the extensive literature survey, involving the investigation of inherent features of these techniques [17], applications of these techniques in software engineering area [5][7][28][30][32], our own experiments – both published [24] and unpublished. With this comparison we do not attempt to produce a general ranking of techniques, since it is very difficult and probably not possible [14][23] or feasible [29], because the technique selection should involve context-specific features. In this comparison we do not consider the accuracy of predictions for these techniques. Earlier studies showed that the accuracy is varying significantly depending on particular dataset used in analysis [14][17][24][28][32].

Most of these techniques are data-driven, which means that the prediction is provided almost entirely based on empirical data. Thus, these techniques fail when such data is not available. We were aware that, due to availability of the data, our model would have to be based in larger extent on expert knowledge rather than on empirical data.

Additionally, only some of these techniques enable providing prediction for multiple dependent variables. Such functionality is crucial because we attempt to develop a model where software quality is reflected not by a single variable but a range of interrelated variables.

The main use of the model is to provide decision support through the ability of performing various simulations. To make these simulations more realistic, the model should have the ability of defining causal relationships by domain experts. Only very few techniques enable this feature.

TABLE I. Features of Popular Modelling Techniques[a]

Feature	Technique								
	MR	***CBR***	***DT***	***RF***	***RI***	***SVM***	***SD***	***NN***	***BN***
expert knowledge	L	H	H	L	H	L	H	M	H
multiple dependent variables	L	L	L	L	H	L	H	H	H
causal relationships	M	L	M	M	H	L	H	M	H
explit uncertainty	M	L	L	L	L	L	L	L	H
intuitiveness	H	M	H	L	H	L	H	L	H
ease of adaptation	H	M	M	M	M	H	M	M	M

a. 'L' – low, 'M' – medium, 'H' – high

Given the context of our research, we have selected BN as a formal representation for our predictive model, because this technique enables the required functionality. BN is a very powerful modeling technique that has already been widely used in various studies on software engineering [22]. Pfautz at al. say that they are "well-suited to capturing vague and uncertain knowledge" [19]. BNs have a unique set of features such as ability to incorporate expert knowledge and empirical data, explicit modeling of causal relationships, probabilistic definition of variables reflecting uncertainty of modeled system, no need to declare in advance a list of input and output variables, ability to run with incomplete data, and visual representation. BNs can also take a form of time-series models called dynamic Bayesian networks. Detailed analysis of the motivations for using BNs can be found in [8][23].

BNs have been used in earlier studies to model software quality. However, most of these studies have been focused on a single aspect of software quality. We found three references, where the authors model multiple features of software quality.

Beaver [4] developed a BN model to reflect software quality according to the ISO 9126 standard. However, the author does not provide enough details on model structure, variable definitions and model validation. Thus, it is difficult to assess the correctness and usability of this model.

Wagner [31] proposed BN models for predicting software quality using activity-based software quality models. In contrast with the current study, the author focused on modeling selected features from ISO 9126 standard, i.e., maintainability and security, and not the relationships between these features.

Fenton at al. [10] developed a BN model for the trade-off between development effort, project scope and software quality. In this model software quality is reflected by two variables, defect rate and customer satisfaction.

IV. Original Bayesian Network Model

The main aim of the BN model is to deliver useful information to project managers and support their decisions. Proposed BN model enables performing various types of analyses:

- 'What-if' analysis - investigating how different actions may influence specific quality factors. For example, an impact of increased amount of *specification effort* on *functional suitability*, *maintainability* or *operability*.
- 'Goal-seeking' - answering a question: how to achieve a specific target? For example: how much better a testing process is required to achieve a higher level of reliability (with other constraints entered to the model).
- 'Trade-off' analysis - investigating the degree at which a quality factor that has to be traded for another quality factor (given other constrains). For example: an architectural trade-off between *performance efficiency* and *maintainability*, where efficient software may be difficult in maintenance.

The initial structure of this model has been discussed in [20][21]. This model is too large to be presented in detail here. Thus, this paper only briefly summarizes its main concepts illustrated in Figures 1 and 2. The main parts, i.e., quality factors are modeled as hierarchical Naïve Bayesian Classifiers where variables reflecting a detailed level of software quality are the children of the more general factors. For example, usability has four sub-features modeled as its children (Figure 1).

Such structure enables easy adjustments, e.g. adding a new sub-feature requires only a definition of this newly added variable without a need to change other parts of the model. Such structure works well even when relationships between children variables exist in reality but have not been included in the model [27].

Quality features are linked with other. These links have been defined according to a knowledge base that contains results from a literature survey [20][21]. Figure 2 illustrates some of these links.

The model also contains some basic process variables describing *effort*, *process quality* and *process effectiveness*. Since this is a part of the model that was significantly enhanced more details on process variables have been provided in Section V.A.

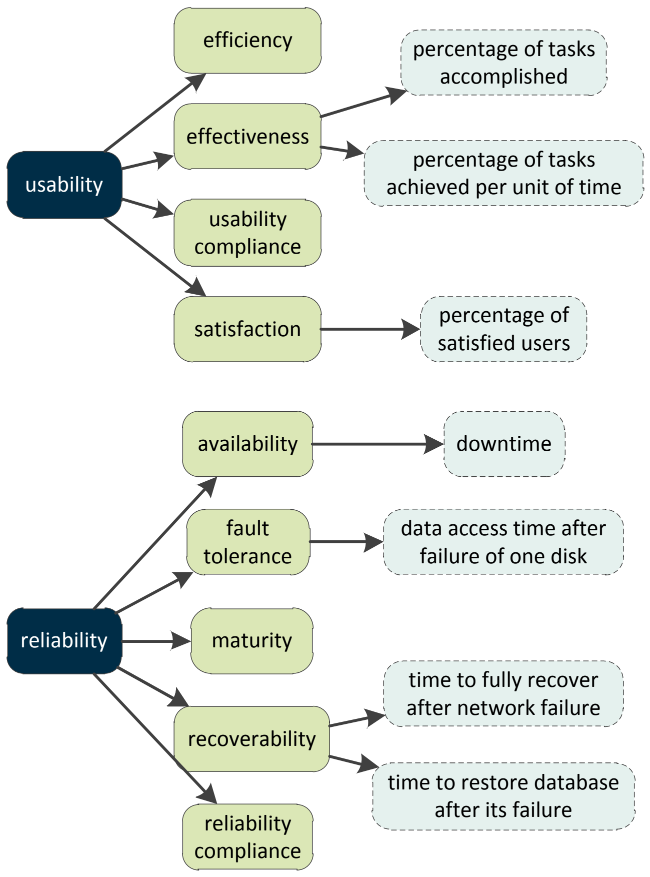
efficiency
percentage of tasks accomplished
effectiveness
usability
percentage of tasks achieved per unit of time
usability compliance
satisfaction
percentage of satisfied users
availability
downtime
fault tolerance
data access time after failure of one disk
reliability
maturity
time to fully recover after network failure
recoverability
time to restore database after its failure
reliability compliance

Figure 1. Three levels of software quality: features (left), sub-features (center) and examples of measures (right).

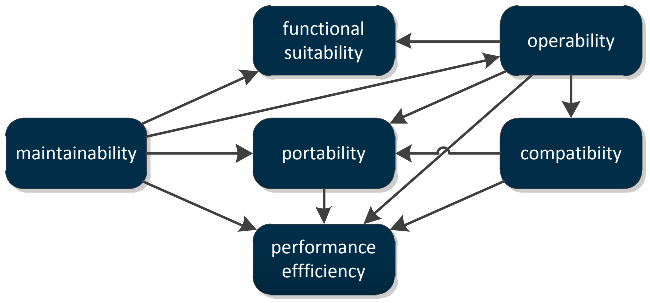

Figure 2. Selected links between quality features.

Proper definition of the quantitative part, i.e., probability distributions for variables, is a challenging step in the process of building a BN. All variables in this model are defined using a 5-point ranked scale from ‘very low’ to ‘very high’ level of intensity. Probability distributions are not defined by manually filling probability tables but using a set of expressions such as weighted mean, weighted max and weighted min [9]. For example, the following expression:

$$proc\_eff = \mathrm{N}(\mathrm{wmean}(3, effort, 4, process\_q), 0.001) \quad (1)$$

means that *process effectiveness* is defined by a Normal distribution as a weighted mean of *effort* and *process quality* with weights 3 and 4, respectively; 0.001 is a value of variance and represents the level of uncertainty. Such types of expressions simplify the process of building a BN because they require only the values of the weights for each variable instead of the whole probability tables.

V. MODEL ENHANCEMENTS

This section considers three main enhancements of the original BN model: incorporating project factors (Subsection A), adding subnets with detailed process factors (Subsection B) and integrating software components or sub-systems (Subsection C).

A. Project Factors

Original model did not contain any project factors, i.e., factors describing the nature of developed project. Thus, it had to be calibrated separately for each project or, more generally, for each type of project. Since such calibration is time-consuming, to improve model usefulness the enhanced model contains additional project factors. These project factors reflect the nature of the project and its environment. Currently the model contains the following project factors: *architecture*, *CASE tool used*, *development platform*, *functional size*, *UI* (User Interface) *type*, *intended market*, and *used methodology*.

Figure 3 illustrates links between selected project factors and selected quality features. To simplify the definition of probability tables for quality features the model uses so called ‘partitioned expressions’, where the child node is defined using different expressions for different states of parent nodes. Figure 4 provides an example of such expressions for *operability* given selected states of *UI type,* together with visualization of probability distributions.

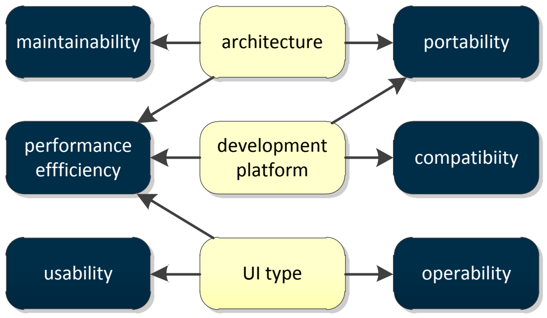

Figure 3. Example of modelling the impact of project factors.

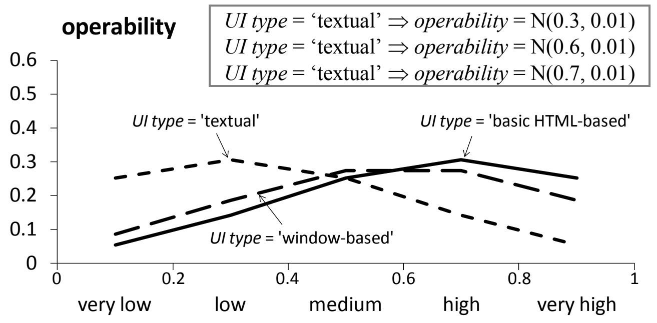

Figure 4. Example of modelling the impact of project factors.

A difficulty of this enhancement is related with the calibration stage. Some quality features, such as *performance efficiency*, have many project factors as parents. Defining probability distributions for such quality features is difficult because they have to reflect every possible combination of states of parent project factors. However, after performing such calibration the usability of the BN model is significantly improved.

B. *Process Factors*

The original model contains nine variables reflecting process of software development: *effort*, *process quality* and *process effectiveness* – separately for three main activities of software development: specification, implementation and testing. In some situations it might be sufficient to represent process quality as a single variable. However, to improve the analytical capabilities of this model, it has been enhanced by subnets with detailed process factors, separate for three main activities of software development.

Figure 5 illustrates a subnet for process factors in specification. Subnets for implementation and testing stages have similar structures – the difference is that they do not contain variables related to requirements (upper right part of Figure 5). Variables describing process factors have been mainly linked according to causal relationships.

For example, the level of *leadership quality* influences three variables: *team organization*, *defined process followed* and *appropriateness of methods and tools used*. Then, the level (quality) of *requirements management*, *defined process followed* and *appropriateness of methods and tools used* jointly determine the level of *process quality*. *Process quality*, *requirements creep* and *staff quality* influence the *overall process quality*.

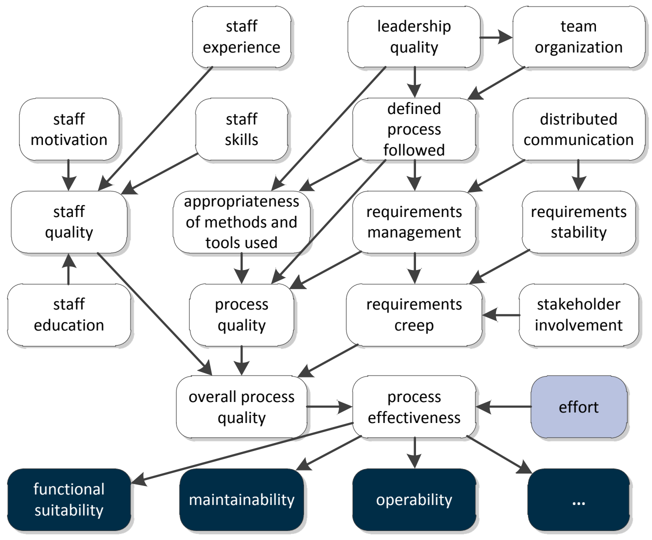

Figure 5. Subnet for process factors in specification stage.

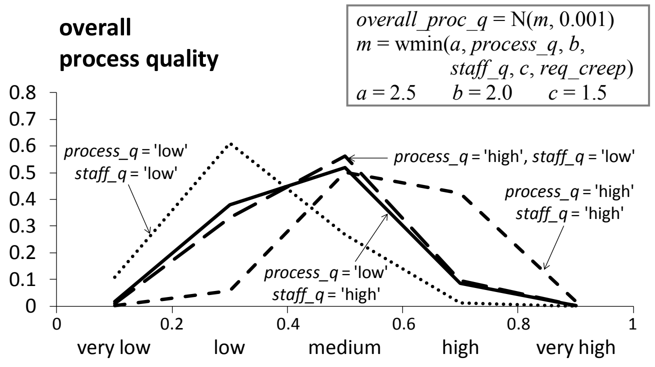

Figure 6. Example of aggregation of process factors.

Process effectiveness aggregates all process factors and is defined as a combination of *overall process quality* and the level of *effort*. Finally, *process effectiveness* variables, separately in three development stages, influence the level of selected software quality features. Variables in this subnet are quantitatively defined using weighted expressions similar to Equation 1.

Figure 6 illustrates the impact of various combinations of *process quality* and *staff quality* on *overall process quality*. The latter is defined as a weighted min (wmin) of its parents to incorporate the fact that undesirable state of one parent node may significantly decrease the value of *overall process quality*, even if other parents are at desirable states. The values of weights determine the strength of impact of particular parent on the aggregated value.

C. *Integrating components/sub-systems*

One of the challenges of building a predictive model for software quality is to properly define the level of granularity. Such model may be built for the whole software systems, sub-systems, single applications, components, modules, classes etc. To improve the flexibility of this model, as another enhancement of the original model, it now can be used at various levels of details.

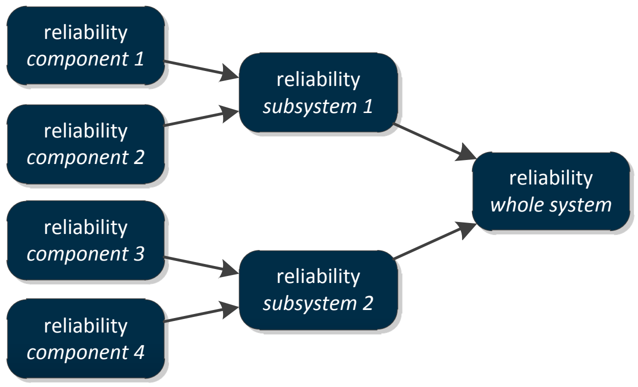

Figure 7. Example of integrating components/sub-systems.

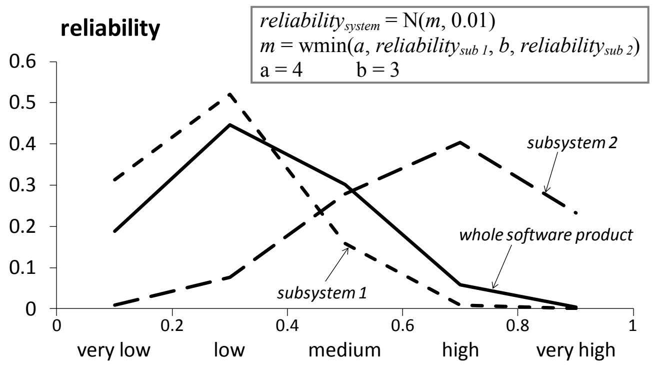

Figure 8. Definition and predictions for integrated reliability.

The basic idea, illustrated in Figure 7, is the following: First, users nominate the lowest level of details for the model. Then, they perform a calibration for this level. The higher level of details is modeled by aggregating quality factors (and possible other variables too). This aggregation can be done using expressions such as mean, max, min, weighted mean, weighted max or weighted min [9]. This procedure may be continued until the highest desired level of details has been reflected in the model.

This enhancement enables performing predictions for quality factors either at the detailed level, e.g., a class or a module, or aggregated, e.g., an application or a system. An example of such aggregation for reliability is shown in Figure 8. The reliability of the whole software product can only be as high as for the part with the lowest level of reliability. In the model it is reflected with the weighted min (wmin) function. Because some subsystems may be more frequently used than the other, their impact on overall reliability would be greater. This can be reflected by adjusting the values of weights a and b. In this hypothetical example a subsystem 1 is more reliable than subsystem 2. It also has greater impact on overall reliability (a>b). Therefore, the reliability of the whole system is between the level of reliability for subsystems 1 and 2, but much closer to reliability of system 1.

VI. Plans for Model Calibration and Validation

Currently, the process of model calibration with industrial partners is performed. In the first step, a tradeoff between model complexity, usability and clearness is investigated. It is focused on answering a question: how large the model can be so that it is clear enough for being used in industry? This calibration is performed using a customized technique of structured interviews based on repertory grid [3][15].

After investigating some patterns from this analysis, the model structure will be adjusted. Then, detailed model calibration will be performed using structured interviews. This will enable capturing relevant expert knowledge that would be difficult to express using only a predefined questionnaire.

This calibration will cover almost the whole structure of the model – except the quantitative measures assigned to quality features/sub-features. Companies that accepted to participate in the process of calibrating the model are not willing to provide such data outside their environments. On one side this is related with data protection and privacy, on the other side with time consuming process of preparing them. Calibration of the rest of the model will be performed by asking domain experts to:

- Assign weights in the weighted expressions;
- Assign the level of their uncertainty about provided data;
- Provide prior distributions for root nodes.

Results of this survey will be combined with results available in the literature and empirical analyses performed earlier.

The internal validation of the model will be focused on investigating how well the model incorporates data/knowledge gathered during the calibration stage. A variety of fitness measures will be used here. In the external validation, industrial partners will be granted access to the model to familiarize with it and assess a variety of its features, such as correctness, usability, clearness, ease of use and ease of customization/calibration.

VII. Conclusions and Future Work

Proposed BN model for software quality prediction reflects the breakdown of quality factors proposed in ISO 250xx series of standards. It contains a variety of software quality factors, together with relationships between them. It also contains process factors that influence software quality.

Obtained results lead to the following conclusions:

- Original BN model is useful in a variety of applications but suffers limitations related to the lack of details on selected software development aspects;
- Proposed enhancements, i.e., incorporating project factors, adding subnets with detailed process factors and ability of integrating software components or sub-systems overcome these limitations;
- Proposed enhancements require additional time for model calibration in target environment.

Plans for future work related to this BN model include:

- Further enhancements to the model to reflect the dynamics of software development and maintenance;
- Automated calibration of the model using the data from software repositories or knowledge base;
- Detailed model calibration and validation using software engineering literature, expert judgment, and empirical data.

ACKNOWLEDGMENT

This work has been partially supported by research funds from the Ministry of Science and Higher Education in Poland as a research grant no. N N111 291738 for years 2010-2012.

REFERENCES

[1] F. Akiyama, "An Example of Software System Debugging," in Proceedings of Federation for Information Processing Congress, vol. 71, Ljubljana, 1971, pp. 353-379.

[2] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability Meanings and Interpretations in ISO Standards," Software Quality Journal, vol. 11, pp. 325-338, 2003.

[3] H. C. Banestad, J. E. Hannay, "Comparison of Model-based and Judgment-based Release Planning in Incremental Software Projects," in Proceeding of the 33rd International Conference on Software Engineering, ACM, 2011, pp. 766-775.

[4] J. M. Beaver, "A life cycle software quality model using bayesian belief networks," Ph.D. Thesis, University of Central Florida, 2006.

[5] S. Bouktif, F. Ahmed, I. Khalil and G. Antoniol, "A novel composite model approach to improve software quality prediction," Information and Software Technology, vol. 52, no. 12, pp. 1298-1311, Dec. 2010.

[6] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications, vol. 36, no. 4, pp. 7346-7354, May. 2009.

[7] C. Catal, "Review: Software fault prediction: A literature review and current trends", Expert Systems with Applications, vol. 38, no. 4, pp. 4626-4636, Apr. 2011.

[8] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models", IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 675-689, Sep. 1999.

[9] N. E. Fenton, M. Neil, and J. G. Caballero, "Using Ranked Nodes to Model Qualitative Judgments in Bayesian Networks," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 10, pp. 1420-1432, Oct. 2007.

[10] N. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor, "Making Resource Decisions for Software Projects," in Proceedings of the 26th International Conference on Software Engineering, 2004, pp. 397-406.

[11] ISO/IEC 25000:2005, Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE, 2005.

[12] ISO/IEC CD 25010:2008, Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Software and quality in use models, Version 0.55, 2008.

[13] ISO/IEC FDIS 9126-1:2001, Software Engineering – Product quality – Part 1: Quality model, 2001.

[14] Y. Jiang, B. Cukic and T. Menzies, "Cost Curve Evaluation of Fault Prediction Models," in Proceedings of the 2008 19th International Symposium on Software Reliability Engineering, IEEE Computer Society, Washington, DC, 2008, pp. 197-206.

[15] G. Kelly, "The psychology of personal constructs," Norton, New York, 1955.

[16] M. Lyu, "Handbook of software reliability engineering," McGraw-Hill, Hightstown, NJ, 1996.

[17] C. Mair, G. Kadoda G, M. Lefley, K. Phalp, C. Schofield, M. Shepperd and S. Webster, "An investigation of machine learning based prediction systems," Journal of Systems and Software, vol. 53, no. 1, pp. 23-29, Jul. 2000.

[18] J. D. Musa, "Software Reliability Engineering: More Reliable Software Faster and Cheaper," Second Edition, Authorhouse, 2004.

[19] J. Pfautz, D. Koelle, E. Carlson, and E. Roth, "Complexities and Challenges in the Use of Bayesian Belief Networks: Informing the Design of Causal Influence Models," Human Factors and Ergonomics Society Annual Meeting Proceedings, vol. 53, no. 4, pp. 237-241, Oct. 2009.

[20] Ł. Radliński, "A Conceptual Bayesian Net Model for Integrated Software Quality Prediction," Annales UMCS Informatica, vol. 11, no. 2, 2011 (accepted).

[21] Ł. Radliński, "A Framework for Integrated Software Quality Prediction using Bayesian Nets," in Proceedings of International Conference on Computational Science and Its Applications (ICCSA 2011), vol. 6786, Springer, 2011, pp. 310-325.

[22] L. Radlinski, "A Survey of Bayesian Net Models for Software Development Effort Prediction", International Journal of Software Engineering and Computing, vol. 2, no. 2, pp. 95-109, 2010.

[23] Ł. Radliński, "Techniques for Predicting Development Effort and Software Quality in IT Projects", Research Papers of the University of Szczecin. Series: Studia Informatica, vol. 26, pp. 119-137, 2010 (in Polish).

[24] L. Radlinski and W. Hoffmann, "On Predicting Software Development Effort using Machine Learning Techniques and Local Data", International Journal of Software Engineering and Computing, vol. 2, no. 2, pp. 123-136, 2010.

[25] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in Empirical Software Engineering and Measurement, 2009, pp. 367-377.

[26] R. J. Rubey, R. D. Hartwick, "Quantitative measurement of program quality," in: Proceedings of ACM National Conference, ACM, 1968, pp. 671-677.

[27] S. Russell, P. Norvig, "Artificial Inteligence. A Modern Approach," Second Edition, Pearson Education, Upper Saddle River, 2003.

[28] M. Shepperd and G. Kadoda, "Comparing Software Prediction Techniques Using Simulation," IEEE Transactions on Software Engineering, vol. 27, no. 11, pp. 1014-1022, Nov. 2001.

[29] Q. Song, Z. Jia, Shepperd M., S. Ying and J. Liu, "A General Software Defect-Proneness Prediction Framework", IEEE Transactions on Software Engineering, vol. 37, no. 3, pp. 356-370, May-Jun. 2011.

[30] B. Stewart, "Predicting project delivery rates using the Naive–Bayes classifier", Journal on Software Maintenance and Evolution: Research and Practice, vol. 14, pp. 161-179, 2002.

[31] S. Wagner, "A Bayesian network approach to assess and predict software quality using activity-based quality models," Information and Software Technology, vol. 52, no. 11, pp. 1230-1241, Nov. 2010.

[32] D. Zhang and J. J. P. Tsai, "Machine Learning and Software Engineering," Software Quality Journal, vol. 11, no. 2, pp. 87-119, 2003.