# Survey on Software Quality Prediction Models

Maninderpal Singh, Neha Sharma, Prabhjot Kaur, Varinderdeep Kaur

***Abstract*—T**hese Software quality assurance is an important part of any software project. The overall complexity and the average size of the software product keeps growing and at the same time, customer keep demanding that more should be done with lesser and lesser effort. The prediction of Software quality during development life cycle of software project helps the development organization to make efficient use of available resource to produce the product of highest quality. This paper presents survey on the software quality prediction models.

***Keywords*—**Software, Quality, Prediction, Models.

## I. Introduction

THIS Software quality assurance is an important part of any software project. The overall complexity and the average size of the software product keeps growing and at the same time, customer keep demanding that more should be done with lesser and lesser effort. Assuring whether the desired software quality and reliability is met for a project is an important as delivery it within scheduled budget and time. In order to appraise the quality of any software product we make use of Software quality estimation models. These quality models can be used to identify program modules that are likely to be defected [5]. It helps project manager to make efficient use of limited resources to target those modules that are defected [18]. A software quality models help development team to track and detect potential software defects during development cycle and saving lots of efforts that are later required for the maintenance of that product. A software quality model is trained using software measurement and defect data of a previously developed release or similar project [13]. A software quality model is a useful tool for meeting the objectives of software reliability and software testing initiatives of different projects [25]. The ability of software quality models to accurately identify critical components allows for the application of focused verification activities ranging from manual inspection to automated formal analysis methods [1]. Software quality models ensure the reliability of the delivered products. The trained model is then applied to modules of the current project to estimate their quality. The proposed work is a supervised clustering approach to estimate the quality of program module.

Maninderpal Singh and Neha Sharma are working as faculty with the Department of CSE/IT at GGSCMT, Kharar, INDIA.

Prabhjot Kaur and Varinderdeep Kaur are working as faculty with the Department of CSE/IT at Bhutta College of Engineering and Technology, Punjab, INDIA.

Quality assurance, or QA for short, refers to a program for the systematic monitoring and evaluation of the various aspects of a project, service, or facility to ensure that standards of quality are being met. Quality assurance is not a phase of the quality plan it is an ongoing process to ensure that the plan is being carried out according to the procedures laid down. It should also have a role in monitoring the effectiveness of procedures intended to establish a quality culture. The role of quality assurance is to ensure that the quality of the procedures and processes result in a product that fully meets the users requirements. As such it is suggested that the quality assurance function be carried out by an independent group of people whose function is solely to monitor the implementation of the quality plan, under the first three headings. It is important to realize also that quality is determined by the program sponsor. QA cannot absolutely guarantee the production of quality products, unfortunately, but makes this more likely.

Two key principles characterize QA: fit for purpose i.e. the product should be suitable for the intended purpose and right first time mistakes should be eliminated. QA includes regulation of the quality of raw materials, assemblies, products and components; services related to production; and management, production and inspection processes. It is important to realize also that quality is determined by the intended users, clients or customers, not by society in general: it is not the same as expensive or high quality. Even goods with low prices can be considered quality items if they meet a market need. QA is more than just testing the quality of aspects of a product, service or facility, it analyzes the quality to make sure it conforms to specific requirements and comply with established plans.

The prediction of Software quality during development life cycle of software project helps the development organization to make efficient use of available resource to produce the product of highest quality. Whether a module is faulty or not approach can be used to predict quality of a software module. To predict the quality we will use various quality prediction models. This requires the usage of a good quality assurance model to maintain a sufficient level of software quality. There are numbers of software quality prediction models described in the literature based upon genetic algorithms, artificial neural network and other data mining algorithms.

In context to software quality estimation, most research have focused on clustering using K-means, Mixture-of-Guassians, Self-Organizing Map, Neural Gas [16]. In all these techniques we are required a predefined structure that is numbers of neurons or clusters before we start clustering

process. To avoid the need of pre-determining the quantity of neurons and the topology of the structure to be used, other clustering techniques start with a minimal neurons structure that is incremented during training until it reaches a maximum number or a optimal number of neurons, in some sense, or until the network reaches a minimal error regarding data quantization. Examples of these algorithms include the Growing Cell Structures (GCS) and the Growing Neural Gas [7] both developed by B. Fritzke., self-organized systems.

## II. Quality Prediction Models

Software quality prediction models seek to predict quality factors such as whether a component is fault prone or not. Methods for identifying fault-prone software modules support helps to improve resource planning and scheduling as well as facilitating cost avoidance by effective verification. Such models can be used to predict the response variable which can either be the class of a module (e.g. fault-prone or not fault-prone) or a quality factor (e.g. number of faults) for a module. The former is usually referred to as classification models and while the latter is usually referred to as prediction models. Software quality models seek to predict quality factors of software components based on product and process attributes.

The basic hypothesis of software quality prediction is that a module currently under development is fault prone if a module with the similar product or process metrics in an earlier project developed in the same environment was fault prone. Therefore, the information available early within the current project or from the previous project can be used in making predictions. This methodology is very useful for the large-scale projects or projects with multiple releases.

Accurate prediction of fault-prone modules enables the verification and validation activities focused on the critical software components. Therefore, software developers have a keen interest in software quality models. It is required that fault-prone prediction models should be efficient and accurate. Fault-proneness models are models that are built from information about the code and its faults, and that relate code to faults. The existence of such classes of software would allow deriving fault-proneness models from historical data and then using such models for predicting fault-proneness of new software applications of the same class. Such models could be useful during both planning and executing testing activities. Planning testing activities could take advantage of information about software fault-proneness for anticipating costs and allocating activities, while test execution can use this information to evaluate the quality of the results.

The software Quality assurance has become a significant aspect in the software industry. Assuring whether the desired software quality and reliability is met for a project is as important as delivering it within scheduled budget and time. To achieve desired software quality, software quality models to identify high risk program modules are used. A software quality model is a useful tool for meeting the objectives of software reliability and software testing initiatives of different projects [25]. Metrics available in the early lifecycle data can be used to verify the need for increased quality monitoring during the development. Different modeling techniques can be used to identify modules as faulty or fault free modules [15]. Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Early detection of fault-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development. Early lifecycle data includes metrics describing unstructured textual requirement and static code metrics. Various researches show that use of static code metrics (such as Halstead complexity, Cyclomatic complexity, McCabe's complexity etc.) to measure quality is inefficient. The use of single features of software to predict faults is uninformative. Fenton offers an example where the same program functionality is achieved using different programming language constructs resulting in different static measurements.

Fenton uses this to argue the uselessness of static code attributes. However, where single features fail, combinations can succeed [25]. Hence combinations of static features extracted from requirements and code can be good predictors for identifying modules that actually contains fault. The ability of software quality models to accurately identify critical components allows for the application of focused verification activities ranging from manual inspection to automated formal analysis methods [1]. Software quality models ensure the reliability of the delivered products. It has become important to develop and apply good software quality models early in the software development life cycle, especially for large-scale development efforts.

One of the more renowned predecessors of today's quality models is the quality model presented by Jim McCall (also known as the General Electrics Model of 1977). This model, as well as other contemporary models, originates from the US military (it was developed for the US Air Force, promoted within DoD) and is primarily aimed towards the system developers and the system development process. It his quality model McCall attempts to bridge the gap between users and developers by focusing on a number of software quality factor that reflect both the users' views and the developers' priorities.

The McCall quality model has three major perspectives for defining and identifying the quality of a software product: product revision (ability to undergo changes), product transition (adaptability to new environments) and product operations (its operation characteristics). Product revision includes maintainability (the effort required to locate and fix a fault in the program within its operating environment), flexibility (the ease of making changes required by changes in the operating environment) and testability (the ease of testing the program, to ensure that it is error-free and meets its specification). Product transition is all about portability (the effort required to transfer a program from one environment to

another), reusability (the ease of reusing software in a different context) and interoperability (the effort required to couple the system to another system). Quality of product operations depends on correctness (the extent to which a program fulfils its specification), reliability (the system's ability not to fail), efficiency (further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage), integrity (the protection of the program from unauthorized access) and usability (the ease of the software).

Boehm's Quality Model (1978) is the second of the basic and founding predecessors of today's quality models is the quality model presented by Barry W. Boehm. Boehm addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. In essence his models attempts to qualitatively define software quality by a given set of attributes and metrics. Boehm's model is similar to the McCall Quality Model in that it also presents a hierarchical quality model structured around high-level characteristics, intermediate level characteristics, primitive characteristics - each of which contributes to the overall quality level.

The high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put – the general utility of software. The high-level characteristics address three main questions that a buyer of software has:

• As-is utility: How well (easily, reliably, efficiently) can I use it as-is?

• Maintainability: How easy is it to understand, modify and retest?

• Portability: Can I still use it if I change my environment?

The intermediate level characteristic represents Boehm's 7 quality factors that together represent the qualities expected from a software system:

• Portability (General utility characteristics): Code possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one.

• Reliability (As-is utility characteristics): Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily.

• Efficiency (As-is utility characteristics): Code possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources.

• Usability (As-is utility characteristics, Human Engineering): Code possesses the characteristic usability to the extent that it is reliable, efficient and human-engineered.

• Testability (Maintainability characteristics): Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

• Understandability (Maintainability characteristics): Code possesses the characteristic understandability to the extent that its purpose is clear to the inspector.

• Flexibility (Maintainability characteristics, Modifiability): Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined.

Dromey's Quality Model is more recent model similar to the McCall's, Boehm's and the FURPS(+) quality model, is the quality model presented by R. Geoff Dromey. Dromey proposes a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems.

## REFERENCES


[1] 1. Bellini, P. (2005), "Comparing Fault-Proneness Estimation Models", In Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), China, pp. 205-214.

[2] 2. Challagula, Bastani, B. and Yen (2006),"A Unified framework for Defect Data Analysis using the MBR Technique", In Proceedings of the 18th IEEE InternationalConference on Tools with Artificial Intelligence (ICTAI'06), Washington, pp. 39-46

[3] 3. Costa, J. A. F., and Oliveira, R. S. (2007)," Cluster Analysis using Growing Neural Gas and Graph Partitioning" International Conference on Neural Networks, Orlando, pp. 3051-3056.

[4] 4. Doherty, K. A. J., Adams, R. G. and Davey, N. (2005), "Hierarchical Growing Neural Gas." International Conference on Adaptive and Natural Computing Algorithms, Boston, pp. 324-333.

[5] 5. Emam, K. E.,Benlarbi, S.,Goel, N. and Rai, S.N. (2001), "Comparing casebased reasoning classifiers for predicting high-risk software components", Journal System Software, vol. 55, no. 3,pp. 301–320.

[6] 6. Fritzke, B. (1994), "Growing Cell Structures - A Self-organizing Network for Unsupervised and Supervised Learning. Neural Networks", IEEE transaction on Software Engineering, vol. 7, no. 9, pp. 1441-1460.

[7] 7. Fritzke, B. (1995), "A Growing Neural Gas Network Learns Topologies." Advances in Neural Information Processing Systems, Los Alamos, pp. 625-632.

[8] 8. Fenton, N. E. and Pfleeger, S. L. (1997), "Software Metrics: A Rigorous and Practical Approach", 2nd editionBoston, MA: PWS-Kent, pp. 256-260.

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

[10] 10. Gillies, A. (1996), "Software Quality", Second Edition, Cengage Learning India Private Limited, Delhi, pp. 45-58.

[11] 11. Giovanni, D. (2000), "Estimating Software Fault Proneness for Tuning testing Activities", IEEE Transactions on Computer Science,vol. 8, no.4 pp.704-706.

[12] 12. Gray, A.R. and MacDonell, S.G. (1999) "Software Metrics Data Analysis: Exploring the relative performance of some commonly used modeling techniques". Empirical Software Engineering Journal, Volume: 4, pp. 297-316.

[13] 13. Guo, L., Cukic, B. and Singh, H. (2003), "Predicting fault prone modules by the Dempster–Shafer belief networks," In Proceedings of 18th International Conference on Automated Software Engineering, Montreal, QC, Canada, pp. 249–252.

[14] 14. Goldman, S. and Zhou, Y. (2000), "Enhancing supervised learning with unlabeled data," In Proceedings of17th International Conference Mach. Learn., pp. 327–334.

[15] 15. Jiang, Y., Cukic, B. and Menzies, T. (2007), "Fault Prediction Using Early Lifecycle Data". In Proceedings of 18th IEEE Symposium on Software Reliability Engineering, IEEE Computer Society, Sweden, pp. 237-246

[16] 16. Kohonen, T. (2001), "Self-Organizing Maps", 3nd Edition, Berlim: Springer: 2001.

[17] 17. Khoshgoftaar, T. M. and Seliya, N. (2002), "Software quality classification modeling using the SPRINT decision tree algorithm", In Proceeding of 4thIEEE international conference on tools with artificial intelligence, MA, pp. 365–374.

[18] 18. Khoshgoftaar,T. M. and Seliya, N. (2003), “Analogy-based practical classification rules for software quality estimation,” Empirical Software Engineering Journal, vol. 8, no. 4, pp. 325–350.

[19] 19. Khoshgoftaar, T. M. and Seliya, N. (2004), “Comparative assessment of software quality classification techniques: An empirical case study,” Empirical Software Engineering Journal, vol. 9, no. 3, pp. 229–257.

[20] 20. Lucky, R. W. (1965), “Automatic equalization for digital communication,” Bell System Technology Journal, vol. 44, no. 4, pp. 547–588.

[21] 21. Lanubile, F., Lonigro, A and Visaggio, G. (1995), “ Comparing Models for Identifying

[22] 22. Fault-Prone Software Components”, In Proceedings of 7th International Conference on Software Engineering and Knowledge Engineering, USA, pp. 12-19.

[23] 23. Munson, J. and Khoshgoftaar, T.M. (1992),” The Direction Of Fault Prone Programs”, In IEEE transaction on Software Engineering, vol. 18,no. 5, pp. 1442-1450.

[24] 24. Seliya, N. and Khoshgoftaar, T M. (2007), “Software Quality Analysis of Unlabeled Program Modules with Semi supervised Clustering”, Software Quality Journal, Vol. 37, no. 2, pp. 201-211.

[25] 25. Seliya N., Khoshgoftaar T.M. and Zhong S. (2005), “Analyzing software quality with limited fault-proneness defect data”, In Proceedings of the 9th IEEE international Symposium on High Assurance System Engineering, Germany, pp. 89-98.

[26] 26. Schneidewind, N. F. (2001), “Investigation of logistic regression as a discriminant of software quality,” In Proceedingsof 7th Internationalconference on Software Metrics Symp., London, U.K., pp. 328–337.

[27] 27. Yaun, X., Khoshgoftaar, M. and Allen, B. (2000), “An Application of Fuzzy Clustering to Software Quality Prediction”, Information Sciences: An International Journal, Vol. 179, no. 8, pp. 1040-1058.

[28] 28. Zhong, S., Khoshgoftaar, T. M. and Seliya, N. (2004), “Analyzing software measurement data with clustering techniques”, IEEE Intell System Journal, vol. 19, no. 2, pp. 20–27.