International Journal of Engineering & Technology, 7 (4.31) (2018) 135-139

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Towards Employing Metrics in Measuring the Quality of Software Safety Critical Systems and Managing their Development

Jamilah Din[1]*, Rodziah Din[2], Yusmadi Yah Jusoh[3], Muhammed Basheer Jasser[4]

[1234]*Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia*
**Corresponding author E-mail: jamilahd@upm.edu.my*

Abstract

Failures in safety critical systems have a high impact on the environment in which they are used. For that, developers usually follow specific development standards and techniques to avoid and reduce failures in such systems. Developers have to build safety critical systems considering the quality, complexity, and size factors in all the development phases beginning with the requirement gathering and ending with the maintenance and verification. In this paper, some quality models and some metrics for measuring the quality, complexity, and size of systems are explored. The metrics are classified to the development phases so that the classification indicates the degree of metrics usage in measuring the quality, complexity, and size factors in relation to the development phases. We are now working on relating the studied metrics in this paper with their impact on the development of the software safety critical system. We expect to use these metrics in the management of the development phases and processes of software safety critical systems so that a decision making could be performed based on the measurements of these metrics.

Keywords: *Safety Critical Systems; Quality Management; Quality Metrics*

1. Introduction

Safety critical systems [1, 2, 3] use software to meet their functionalities. Failures in these software lead to a very high impact on the environment in which the safety critical systems are used. For that developers usually follow specific development standards and techniques to avoid or reduce failures. Developers need to build safety critical systems meticulously taking into account all required procedures for avoiding failures.
Software engineering developers improve safety critical systems development by practicing appropriate processes, specific techniques, and domain expertise. Furthermore, software engineers have invented some development metrics to measure certain part of development processes seeking more qualified products and processes. Quality is an important factor in building systems. Software quality has many definitions [4, 5]. In [4], quality is defined as: "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs".
Quality becomes more important in the development of safety software critical systems for avoiding failures [6]. Several quality models, practices, attributes and metrics have been introduced in the literature to control and measure the quality of developed software and the development process.
Also, the complexity of a system is one aspect of the system quality. In this paper, some models and metrics for measuring quality and complexity of systems are explored. The metrics are classified to the development phases so that the classification indicates the degree of metrics usage in measuring the quality, complexity, and size factors in relation to the development phases beginning with the requirement gathering phase and ending with the maintenance and verification phase. We expect to use these metrics in the management of the development of software safety critical systems so that a decision making could be performed based on the measurements of these metrics.
The structure of this article is as follows. Section 2 exhibits some literature works, which define the quality and safety factors and criteria. Section 3 reviews some metrics used to measure the quality and complexity of systems. The metrics employed in measuring the quality and complexity of object oriented systems are in Section 3.1. Section 3.1.1 presents the CK metrics suite. Section 3.1.2 presents the MOOD metrics suite. Section 3.1.3 presents the Lorenz and KIDs metrics suite. Section 3.1.4 presents the QMOOD model and metrics suite. Some metrics, used in measuring the quality and complexity of traditional systems, are explored in Section 3.2. The metrics, employed in the management of software maintenance in general, are presented in Section 3.3. Section 3.4 presents the classification of the metrics into the development phases. Section 4 concludes the work.

2. Quality and Safety Criteria in Safety-Critical Systems

Quality becomes vital in safety critical systems in which failures lead to loss of lives and assets. Quality is defined in [7] as: "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software".
Software quality is represented by the set of quality factors,

attributes, and metrics that could be measured to assess the quality level of the related system. Several quality attributes and metrics are defined in the literature to measure and define the quality criteria.
Safety is defined by some terms [8] such as: correctness, efficiency, reliability, and security. Some of the quality factors and criteria have been selected in relation to software safety [8]. This is based on two assumptions: first, the system characteristics that are critical to safety are related to real time systems, human lives, and hazardous material handling, and second, safety is a sub-set of reliability so that safety concerns the failures that are only related to hazards, whereas reliability is related to all failures. Based on these assumptions, the quality criteria and metrics, which correspond to the selected quality factors, are defined. The identified safety factors are: correctness, efficiency, responsiveness, and testability. The criteria of these factors are defined as well. For example, the criteria for correctness are defined as: completeness, consistency and traceability. Metrics are also defined for the criteria. For example, the consistency criteria is defined as: the metric of the number of contradicting functions, interfaces, inputs, outputs, and data.

3. Some Metrics for Measuring Systems Quality and Complexity

Metrics are introduced in the literature to measure the quality and complexity of systems. We present in this section the metrics used for measuring the object oriented and traditional systems and some other metrics for managing and measuring the quality of software maintenance in specific.

3.1. Some Metrics for measuring Quality and Complexity of Object Oriented Systems

Object oriented analysis and design methods gain so much popularity in building systems. For that, several metrics are introduced to measure object oriented systems in relation to quality.

3.1.1. CK Metrics Suite

CK metrics suite [9, 10, 11, 12] has been introduced to be basically applied to the notions of classes, their methods, coupling between classes, and inheritance between classes.
Weighted Methods per Class (WMC): This metric denotes the sum of cyclomatic complexities of the methods implemented or defined within a class. This metric helps measuring the time and effort required to implement the class.
Depth of Inheritance Tree (DIT): This metric denotes the maximum length from the class node to the tree root within the inheritance hierarchy.
Number of Children (NOC): This metric denotes the number of immediate subclasses subordinated to a class in the class hierarchy.
Coupling between object classes (CBO): This metric denotes the count of the number of other classes to which a specific class is coupled.
Response for a Class (RFC): This metric denotes the set of methods in a class that can be invoked/executed in response to a message to an object of this class.
Lack of Cohesion in Methods (LCOM): This metric denotes the similarity between methods of a class in terms of the shared common instance variables among the methods.

3.1.2. MOOD Metrics Suite

MOOD metrics [12, 13, 14] concern measuring the encapsulation, inheritance, polymorphism, and coupling.
Method Hiding Factor (MHF): This metric denotes the ratio of the sum of all the invisibilities of all the methods of all classes to the number of all defined methods. The method invisibility is defined by the classes percentage from which the method is not visible.
Attribute Hiding Factor (AHF): This metric denotes the ratio of the sum of all the invisibilities of all the attributes of all classes to the number of all defined attributes. The attribute invisibility is defined by the classes percentage from which the attribute is not visible.
Method Inheritance Factor (MIF): This metric denotes the ratio of the sum of all the inherited methods of all classes to the number of all defined methods.
Attribute Inheritance Factor (AIF): This metric denotes the ratio of the sum of all the inherited attributes of all classes to the number of all defined attributes.
Polymorphism Factor (PF): This metric denotes the ratio of the possible various polymorphic cases to the maximum number of possible various polymorphic cases.
Coupling Factor (CF): This metric measures the coupling degrees between classes that are seen as clients and suppliers. This coupling is related to the references from client classes to supplier ones.

3.1.3. Lorenz and KIDS Metrics Suite

Lorenz and KIDS Metrics Suite [15, 12] includes ten metrics that are: number of public methods NPM, number of methods NM, number of public variables per class NPV, number of variables per class NV, number of methods that are inherited by a sub-class NMI, number of methods overridden by a sub-class NMO, number of methods added by a sub-class NMA, average method size AMS, number of times a class is reused NCR, and number of friends of class NF.
The number of public methods PM metric counts the methods of a class that are defined as public methods. This metric is useful to estimate the required effort to build the class.
The number of methods NM metric counts the number of methods of a class that are defined either as: public, private, or protected. This metric is useful to indicate the classes that may perform too much functionality on their own.
The number of public variables per class NPV metric counts the number of public attributes/variables in a class as a measure of the class size.
The number of variables per class NV metric counts the number of variables/attributes in a class including the public, private, and protected attributes. The ratio of the number of private and protected attributes of a class to the number of all attributes of the class indicates the effort, which is required by the class to introduce information to other classes.
The number of methods that are inherited by a sub-class NMI metric counts the number of methods, which are inherited by a sub-class.
The number of methods, which are overridden by a sub-class, NMO metric measures the number of methods overridden via redefinition while keeping the same name.
The number of methods, which are added by a sub-class, NMA metric counts the number of methods that are added in a sub-class and do not exist in the super-class. A method is considered as added one if the super-class does not contain a method with the same name.
The average method size AMS measures the number of source lines, which are not comments or blank lines, divided by the number of the class methods.
The number of times a class is reused NCR metric, as described in [12], counts the number of times a class is references by other classes.
The number of friends of class NF metric measures the number of friend classes of a class that allow encapsulation to be violated.

3.1.4. QMOOD Quality Model and Metrics

QMood [16, 17] is a quality model introduced to measure and assess the high level design quality attributes like: flexibility, reusability, and complexity in object oriented designs. This includes assessing behavioural and structural design properties of objects, classes, and their relationships like: modularity, encapsulation, cohesion, and coupling via a suite of object oriented design metrics.
The model has four levels. The first level L1 is to identify the design quality attributes. The second level L2 is to define the object-oriented design properties. The third level is to identify the object-oriented design metrics. The fourth level L4 is identify the object-oriented design components of interest.
Identifying the design quality attributes: This is to identify the design quality attributes: effectiveness, functionality, extendibility, understandability, flexibility and reusability.
The effectiveness denotes to the ability of designs to perform the intended functionality. Functionality refers to the responsibilities of the design classes that are allowed by their public interfaces. The extendibility denotes to the design properties that allow adding new requirements to the design. The understandability refers to the design properties that are easy to be understood. The flexibility refers to the properties, which permit modifications to be incorporated in the design. The reusability reflects the existence of design properties, which permit the design to be reused without much effort.
Defining the object-oriented design properties: This includes defining the design size, abstraction, hierarchies, coupling, cohesion, encapsulation, inheritance, polymorphism, messaging, composition, and complexity.
The design size measures the number of the design classes. The abstraction is a measurement of the design classes that have one or more descendants. The hierarchies measures the number of non-inherited classes, which have children in the design. The coupling measures the number of other objects, which are required to be accesses by an object for that object to function properly. The cohesion measures the level in which the methods and attributes of a class are related to each others. The encapsulation refers to designing classes preventing the access to their attributes by making their as private attributes. The inheritance measures the inheritance relationships among the classes in the inheritance hierarchy. The polymorphism measures the ability for the object services to take more than one specialized form during run-time. The messaging measures the number of public methods of class, which could be accessed by other classes in the design. The composition measures the aggregation relationships in the design. The complexity measures the difficulty degree in understanding the structure of classes and their interconnecting relationships.
Identifying the object-oriented design metrics: In this level, the object-oriented design metrics are identified to measure the object-oriented design properties that are defined in the previous level L3. This includes defining the following metrics: Design size in classes DSC, number of hierarchies NOH, average number of ancestors ANA, data access metric DAM, direct class coupling DCC, cohesion of methods of classes CAM, measure of aggregation MOA, measure of functional abstraction MFA, number of polymorphic methods NOP, class interface size CIS, number of methods NOM.
The design size in classes metric DSC measures the total number of classes in a design.
The number of hierarchies metric NOH measures the number of class hierarchies in the design.
The average number of ancestors metric ANA measures the average number of classes from which a class inherits from by specifying the number of classes from the root class to all other classes in the inheritance hierarchy.
The data access metric DAM measures the ratio of the private attributes to all the attributes of a class.
The direct class coupling metric DCC measures the count of the number of the classes that are directly related to a class.
The cohesion of methods of class metric CAM measures the level of relatedness of methods of a class.
The aggregation measurement metric MOA measures the count of data declarations that have types as user defined classes.
The functional abstraction measurement metric MFA measures the ratio of the number methods inherited by a class to the total number of methods that are accessed by the member methods of the class.
The number of polymorphic methods metric NOP measures the number of methods, which could employ a polymorphic behaviour.
The class interface size metric CIS measures the number of public methods in a class.
The number of methods metric NOM measures the number of all methods of a class.

Identifying the object-oriented design components: This level is for identifying the object-oriented design components that are: classes, objects, relationships between classes and objects, class hierarchies, class methods, and class attributes

3.2. Some Traditional Metrics and Measurements for Measuring Quality, size, and Complexity

Several traditional metrics are proposed to measure the size, complexity, and quality of systems [18, 19, 20, 21, 22, 23, 24, 25]. Some of these metrics are the McCABE’s cyclomatic complexity [18] and the function point [20, 21, 22, 25], and the specification weight (function bang) [25].
The McCABE’s cyclomatic complexity is based on the graph theory, and it calculates the number of linearly independent paths of a source code.
The function point is a measurement to estimate the amount of function that a software performs in terms of the data it uses as input and produces as outputs.
The specification weight (function bang) is a measurement for the product size after classifying the related system specification into: function-strong systems, data-strong systems, and hybrid systems. This classification is based on the ratio of the number of relationships in the data model to the number of functional primitives in the data flow diagram.
Some other metrics are introduced based on the information flow between systems components [23, 24]. They are to measure procedure complexity, module complexity, and module coupling. The procedure complexity is measured based on the procedure code complexity and procedure connections to its related environment. Using the information flow allows early measurements of software quality and complexity in the development process and it provides a quantitative evaluation of structural properties of large systems.

3.3. Some Quality Metrics for Managing and Measuring the Software Maintenance

Several metrics are introduced in the literature to manage the maintenance phase in developing system software. In [26], a set of metrics are introduced to manage and understand the software maintenance effort. These metrics are based on the Goal/Question/Metric paradigm [27].
Among these metrics, we are only interested in those related with the quality of software. These metrics are: current change backlog, software reliability and change cycle time from data approved and from data written. These metrics are related to the goal of maximizing customer satisfaction and they are to answer the questions: ”how many problems affect the customer?” and ”how long does it take to fix an emergency or urgent problem?”.The current change backlog metric is the number of the complaints of the customer that remain to be solved. If this metric value is zero, then this does not mean that there are no problems that affect the users, but this means that there are

enough or more staff to tackle these problems in a way making the current change backlog metric having the zero value.
The software reliability is another metric to track the problems that affect the customer. This metric is defined by the software failure rate. A failure is considered in the software if the system does not meet the customer requirement. The system manager could use this metric to manage the maintenance of software in many ways. A threshold may be determined by the number of failures occurring within specific working hours. If the system at hand operates below this threshold, then the system manager may decide to implement more difficult changes in the next software release. However, if the system operates above the threshold, then the system manager may decide to revert to the earlier version release of software or remain at the same current release until the necessary corrections are performed so that the software operates below the threshold.
The metric, change cycle time from data approved and from data written, is to answer the question: ”how long does it take to fix a priority, urgency or emergency problem ?” and this metric is also related to customer satisfaction. In other words, this metric is to measure the cycle time of the priority changes, which are delivered within a specific number of days, from the time at which the changes have been written by the customer. This metric is also to measure: the cycle time of the priority changes, which are delivered within a specific number of days, from the time at which the changes have been validated and approved by the user board.

3.5. Classification of Metrics into the Development Phases

In this section, we classify the different metrics, considering the (quality, complexity, and size) criteria for both (traditional and object-oriented) systems, into different phases, as in Table 1.
The focus is on the quality criteria since it is one of the most important factors in safety critical systems. The metrics that are covered in this work are related to the quality measurement and management of the systems within the development phases: requirement definition, analysis and design, implementation, maintenance and verification.
The metrics, which are classified to the requirement definition, analysis and design and implementation phases, are also classified to the related traditional or object-oriented systems. This is because; each of these metrics is targeted for measuring specific type of systems (traditional/object-oriented). However, the metrics, which are classified to the maintenance phase, are not classified to traditional or object-oriented systems. This is because the metrics in the maintenance phase are targeted to measure all types of systems without specifying its type necessarily.

Table 1: Classification of Metrics to Development Phases

(System,Metric) /Phase	Requirements Definition	Analysis and Design	Implementation	Maintenance and Verification
Traditional	Function point	Function point Cyclomatic Complexity (CC) Information Flow Specification weight (function bang)	Information Flow Line of code (LOC)	Current change backlog(the number of customer complaints that remain to be solved) Software reliability (software failure rate) Change cycle time from data approved and from data written
Object-Oriented		CK suite (WMC, DIT, NOC, CBO, RFC, LCOM) MOOD suite (MHF, AHF, MIF, AIF, PF, CF) Lorenz and Kidd Suite(NPM, NM, NPV, NV, NMI, NMO, NMA, AMS, NCR, NF) QMOOD suite (DSC, NOH, ANA, DCC, CAM, MOA, MFA, NOP, CIS, NOM)		

It is expected that this classification should help focusing on each phase, and how these classified metrics help in the management of the various development phases of the safety critical systems in relation with the measured and estimated quality, size and complexity of the systems that are being dealt with.

4. Conclusion

Quality management is an important factor in reducing failures in systems, especially the software safety critical systems. In this paper, a review of some quality and complexity models, attributes, and metrics, which are used in the management and measurement of both object-oriented and traditional systems, are explored.
These metrics are classified to the various development phases of both traditional and object-oriented systems.
For object-oriented systems, in specific, Chidamber and Kemerer CK metrics suite, Abreu MOOD metrics suite, Lorenz and KIDD metrics suite, and QMOOD model and metrics suite are explored. These are for the measurement and management of these systems in the analysis and design phase.
For traditional systems, in specific, cyclomatic complexity, information flow, and specification weight (function bang) are the explored metrics, which are for the measurement and management of these systems in the analysis and design phase. Function point is the metric employed in the measurement of these traditional systems in the phases (requirement definition and analysis and design).
Some other metrics, which are for measuring and managing the quality of systems in general in the maintenance phase, are also explored. These are basically related with the customer satisfaction. These metrics may be employed in both traditional and object-oriented systems.
We are now working on relating the studied metrics in this paper with their impact on the development of software safety critical system. We expect to use these metrics in the management of the development phases and processes of software safety critical systems so that a decision making could be performed based on the measurements of these metrics.

Acknowledgement

Thank you to the Ministry of Higher Education (MOHE) and Research Management Center, Universiti Putra Malaysia (UPM) for the financial supports through FRGS Vote No: 08-01-15-1726FR.

References

[1] M. Rausand, Reliability of safety-critical systems: theory and applications. John Wiley & Sons, 2014.
[2] L. E. G. Martins and T. Gorschek, “Requirements engineering for safety-critical systems: A systematic literature review,” Information and software technology, vol. 75, pp. 71–89, 2016.
[3] E. S. Grant, V. K. Jackson, and S. A. Clachar, “Towards a formal approach to validating and verifying functional design for complex safety critical systems,” GSTF Journal on Computing (JoC), vol. 2, no. 1, 2018.
[4] Iso and I. Std, “9126 software product evaluation–quality characteristics and guidelines for their use,” ISO/IEC Standard, vol. 9126, 2001.
[5] N. Bevan, “Quality in use: Meeting user needs for quality,” Journal of systems and software, vol. 49, no. 1, pp. 89–96, 1999.
[6] N. Silva and M. Vieira, “Towards making safety-critical systems

safer: Learning from mistakes," in Software Reliability Engineering Work shops (ISSREW), 2014 IEEE International Symposium, pp. 162–167, IEEE, 2014.

[7] R. S. Pressman, Software engineering: a practitioner's approach. Pal- grave Macmillan, 2005.

[8] R. Singh, "A systematic approach to software safety," in Software Engineering Conference, 1999.(APSEC'99) Proceedings. Sixth Asia Pacific, pp. 420–423, IEEE, 1999.

[9] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on software engineering, vol. 20, no. 6, pp. 476–493, 1994.

[10] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design", vol. 26. ACM, 1991.

[11] S. R. Chidamber and C. F. Kemerer, "Moose: Metrics for object oriented software engineering," in Workshop on Processes and Metrics for Object Oriented Software Development, OOPSLA, vol. 93, 1993.

[12] R. Harrison, S. Counsell, and R. Nithi, "An overview of object-oriented design metrics," in Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on incorporating Computer Aided Software Engineering, pp. 230–235, IEEE, 1997.

[13] F. B. Abreu,"Design metrics for oo software system," in ECOOP '95, Quantitative Methods Workshop, 1995.

[14] F. B. Abreu, M. Goula˜o, and R. Esteves, "Toward the design quality evaluation of object-oriented software systems," in Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, pp. 44–57, 1995.

[15] M. Lorenz and J. Kidd, Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., 1994.

[16] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," IEEE Transactions on software engineering, vol. 28, no. 1, pp. 4–17, 2002.

[17] J. Bansiya, A hierarchical model for quality assessment of object-oriented designs. The University of Alabama in Huntsville, 1997.

[18] T. J. McCabe, "A complexity measure," IEEE Transactions on software Engineering, no. 4, pp. 308–320, 1976.

[19] J. P. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M. A. Adler, "Software complexity measurement," Communications of the ACM, vol. 29, no. 11, pp. 1044–1050, 1986.

[20] J. Albrecht, "Measuring application development productivity," in Proc. of the Joint SHARE/GUIDE/IBM Application Development Symposium, pp. 83–92, 1979.

[21] J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," IEEE transactions on software engineering, no. 6, pp. 639–648, 1983.

[22] D. R. Jeffery, G. C. Low, and M. Barnes, "A comparison of function point counting techniques," IEEE Transactions on Software Engineering, vol. 19, no. 5, pp. 529–532, 1993.

[23] S. M. Henry, Information flow metrics for the evaluation of operating systems' structure. PhD thesis, 1979.

[24] S. Henry and D. Kafura, "Software structure metrics based on in-formation flow," IEEE transactions on Software Engineering, no. 5, pp. 510–518, 1981.

[25] R. Rask, P. Laamanen, and K. Lyyttinen, "Simulation and comparison of albrecht's function point and demarco's function bang metrics in a case environment," IEEE Transactions on Software Engineering, vol. 19, no. 7, pp. 661–671, 1993.

[26] G. E. Stark, "Measurements for managing software maintenance.," in icsm, p. 152, 1996.

[27] V. R. Basili and H. D. Rombach, "Tailoring the software process to project goals and environments," in Proceedings of the 9th inter-national conference on Software Engineering, pp. 345–357, IEEE Computer Society Press, 1987.