IJIEMR Transactions, online available on 26th Jan 2022. Link :http://www.ijiemr.org/downloads.php?vol=Volume-11&issue=Issue 01

10.48047/IJIEMR/V11/ISSUE 01/37

TITLE: THE STUDY OF DEVELOPMENT OF RELIABILITY METRICS

Volume 11, ISSUE 01, Pages: 232-243

Paper Authors **PRAVEEN KUMAR, DR. KAILASH PATIDAR**

THE STUDY OF DEVELOPMENT OF RELIABILITY METRICS

PRAVEEN KUMAR
Research Scholar, Dr. A.P.J. Abdul Kalam University, Indore
DR. KAILASH PATIDAR
Research Supervisor, Dr. A.P.J. Abdul Kalam University, Indore

ABSTRACT

In a CBSD environment, component selection is crucial. The collection has a wide selection of potential parts. Analyzing several software metrics and pinpointing their respective sub-factors is the focus of this research.

KEYWORDS: structural, approach, the object-oriented, complexity, dependability.

INTRODUCTION

Quality metrics, such as those assessed by object-oriented software metrics, have a direct impact on a software product's performance, size, complexity, and reliability. It is impossible to overstate the importance of software metrics when comparing the results of various methods for creating software and their implications for continuing software maintenance. In modern software engineering environments, software metrics are more important. A metric in software development is an objective, quantifiable evaluation of some aspect of the project. For example, if a software is being developed in-house, the members of the development team can use a variety of metrics to assess the nature of any problems they're having with the creation of the software and the progress of any related projects, all within the context of the CBSD, and all of which can be used to perform a very significant evaluation. To make better forecasts and finish the right solution, software evaluation metrics are helpful.

There are two main schools of thought when it comes to developing new software programs: the structural approach and the object-oriented approach. The development of complex or large software applications is best done using an object-oriented approach as opposed to a structured one. There are numerous advantages to using an object-oriented approach, but there are also significant drawbacks, particularly in the areas of security and privacy, that must be addressed. The cost of developing a new information-based legacy system from scratch is more than that of developing a system based on the idea of reuse rather than coding each individual feature from scratch. Since then, a new methodology has emerged called component-based development [CBD], which is predicated on the idea of reusability. Narasimhan investigated comparing the three sets of metrics in a methodical manner.

There are a variety of metrics and models that may be used to assess the complexity, reliability, and maintainability of the CBS. Nevertheless, a different approach is necessary when both OSS and In-House components are involved. It is important to evaluate the complexity and reliability of a part before incorporating it into a larger system. This study is the initial stage in confirming or estimating the complexity of the component using the given method and the reusable process approach. The software complexity and reliability metrics of component-based systems have been the subject of several studies, but the difficulties of putting CBS into practice have received less attention. The term "coupling" refers to the degree of interdependence between two software parts. Having a better understanding of software dependencies may help improve program readability, maintainability, and usability. As far as we can tell, there is a dearth of research on the use of a properly adapted complexity measure in CBSE to enhance system dependability via the use of the concept of cohesiveness. Component complexity measurements in the context of CBSD are still lacking, with most measurements being extrapolations from previous methodologies.

There is a lot written about software metrics, but CBSE measurements are required to make up for some of the deficiencies of the more common metrics. The chidamber and Kemerer debuted the first suite in 1991. Nevertheless, further improvements to these measures were achieved by researchers Li and Henry in 1993. Since then, various new measures, including as CPC, CSC, CDC, DSC, NOH, and ANA, have evolved, all of which are based on design and connectivity between components. In 2003, Martin developed a package-based metric for measuring program complexity.

The following are some of the sources of motivation for the proposed research, all of which point to the need of using a refined metric for component selection in CBSD.

The static source code of the component has only been subjected to a small subset of the metrics available in CBSE to evaluate its reliability.

It is crucial to establish and develop new static software metrics in order to measure the dependability of components in CBSE. The urgency of our situation is motivating us to take this action.

Although several metrics exist for gauging the complexity of a component-based system, none of them can be used for optimal component selection on the basis of the common hierarchical relationship across packages, classes, and methods.

Most current metrics are based on object-oriented notions like inheritance, polymorphism, and constructor, but they don't account for the packages between classes and methods when calculating coupling and cohesion.

LITERATURE AND REVIEW

Luis F. Mendivelso et.al (2018) Software engineers consider application maintenance a crucial responsibility. There is a high cost since the user must read and understand the source code because there is typically no existing abstraction or documentation to help in this activity. Extracting architectural views of software from source code is the goal of a number of commercial and research tools. 1) their dependency on the language/technology upon which the application is constructed, and (ii) their availability of pre-defined perspectives which are too sophisticated to customize to particular needs for software comprehension, are the key downsides of such tools. In this paper, we provide a Technology-neutral approach flexible enough to enable annotated architectural viewpoint construction by programmers. These viewpoints provide a unified picture of architectural elements whose visual representation corresponds to software measurements. Working side-by-side with commercial customers that have an immediate need to update their old code is where we gained our experience in Oracle Forms, Java EE, and Ruby on Rails. In the following, we describe the specific applications of our concept in various projects and compare the results to those obtained by following industry norms.

Durga Patel, Pallavi (2016) A high degree of software dependability is essential to ensure the 24/7 availability of mission-critical business applications. For something to be reliable, there must be a reasonable chance of it happening. The program must be mistake free and 100% reliable. Errors in the code make the product completely untrustworthy. The need for complicated systems is rising quickly. Early in the 1970s, software became an issue for businesses because of the steadily rising cost of software compared to hardware during both the setup and maintenance phases. A critical component of any instrument designed to convert a discrete set of inputs into a discrete set of output is its software. Due to the human nature of its creators, software is prone to having flaws. Hence, it's crucial to have a method of monitoring software reliability in order to spot any signs of failure. Quite a bit of research has been done in the area of predicting how reliable software will be.

Yan-Fu Li et al (2021) Both energy (through power grids) and information (by telecommunications networks) are transported via networks. Both of these buildings provide essential functions for human civilization. In this research, we look at the evolution of dependability measures for both power grids and telecommunications networks. The primary goal of this analysis is to encourage and facilitate the development of dependability indicators for communication networks with respect to the power grid. We divide the metrics of the electricity grid into those of distribution reliability and those of generation/transmission reliability, and we divide those of the telecommunications networks into those of connection, performance, and status. Then, we show how the two systems' dependability measures are different in different scenarios and talk about why that matters. Finally, we suggest several avenues for further exploration and improvement in the area of dependability measures for telecommunications networks.

Ali Maatouk et al (2023) In this work, we examine the relationship between network dependability and a utility function that changes over time to represent the system's actual performance. The system suffers a utility loss when an anomaly arises, the magnitude of which is proportional to the length of time the abnormality persists. Taking into account exponential anomalies' inter-arrival periods and generic distributions of maintenance length, we examine the long-term average utility loss. We demonstrate that there is a simple form to which the estimated utility loss converges in probability. We then generalize our convergence findings to include additional families of non-periodic utility functions and a wider range of inter-arrival time distributions for anomalies. Data from a cellular network with over 20,000 subscribers and 660 base stations is used to back up our claims. We show that user traffic exhibits a quasi-periodic pattern, and we show that the intervals between anomaly occurrences follow an exponential distribution, enabling us to apply our findings and provide dependability ratings for the network. Moreover, we explore the influence of non-stationarity on our convergence findings, the interaction between the various network characteristics, and the convergence pace of the long-term average utility loss.

Gurpreet Kaur, Kailash. bahl (2014) The purpose of this work is to investigate the measures of software dependability. One of the most crucial but elusive qualities of any piece of software is its reliability. "Software Reliability is defined as the likelihood of failure-free software execution for a specific amount of time in a particular environment," states the American National Standards Institute. There is a distinction between hardware reliability and software reliability. The enormous complexity of software makes it difficult to achieve dependability. Modeling, measuring, and improving software reliability may be thought of as the three main aspects of software reliability. It is challenging to strike a balance between development time and budget and software dependability, but there are several ways to increase software reliability. The optimal method, however, is to produce high-quality software over the whole software life cycle. Metrics for measuring the stability of software are the focus of this work. Early usage of metrics may help find and fix flaws in requirements, which in turn can help avoid problems later in the software life cycle. Measurements of Software Reliability are discussed in this article.

TYPES OF SOFTWARE METRICS

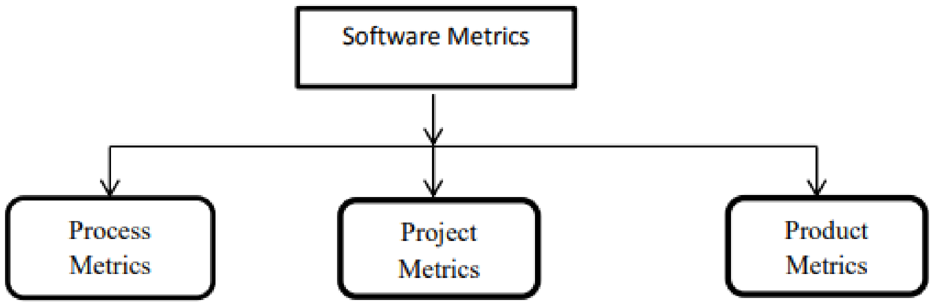

Fig 1 Classification of software metrics

PROPOSED FRAMEWORK FOR THE DIFFERENT METRICS

Many keyword-based searches and hybrid strategies using the genetic algorithm approach have been developed to address the component selection problem in the context of component-based development. The suggested method differs significantly from the evolutionary algorithm in that it is centered on the software rather than the end user. This is so because we prioritize simplicity when selecting components from the repository. Here, we describe some of the prior efforts that have prepared the way for our own. Several approaches have been proposed to solve the problem of component selection in the CBSD, such as keyword-based searches, hybrid methods, and the genetic algorithm. In contrast to the genetic algorithm, which was developed with the end user in mind, the Optimal Component Selection (OCS) algorithm and the suggested strategy are primarily concerned with the functionality of the program. That's because we're picking just the most crucial pieces of the system to include in the final build as part of the component selection from the repository, which helps keep the program as simple as possible. As a result, the proposed technique has verified the individual components' reusability ratings. Priority will be given to selecting the most functional components, taking into account factors like their potential for reuse.

Here is a detailed strategy for selecting components, along with three distinct methods for selecting the optimal combination of components from the repository.

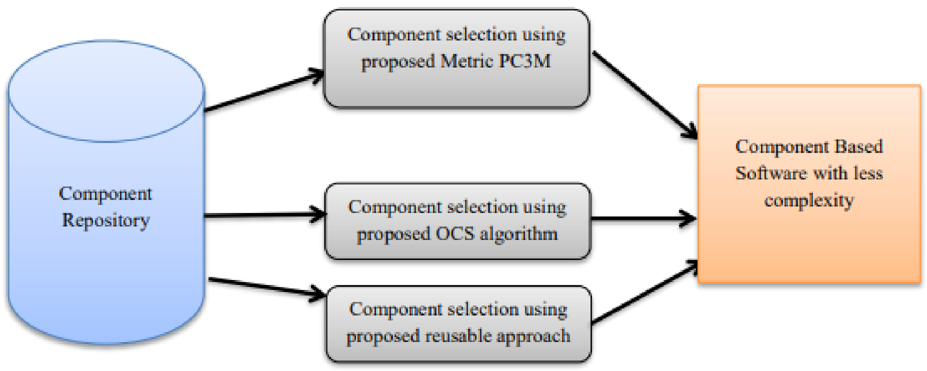
Component Repository
Component selection using proposed Metric PC3M
Component selection using proposed OCS algorithm
Component selection using proposed reusable approach
Component Based Software with less complexity

Fig 2 Complete proposed strategy for component selection

The aforementioned flowchart depicts the three primary options available when choose which component from the repository to employ in software development. In the first case, you may just use the revised measurement PC3M (Pulse Code Modulation(sampling, quantization and coding) that was proposed. In the second, you can use an OCS algorithm. In the third, you can use a reusable process technique.

IDENTIFYING SOFTWARE QUALITY SUB-FACTORS

All of the aforementioned characteristics of high-quality software are subjective in nature. Subjective considerations can't be evaluated using a qualitative scale since they lack a numeric value. Quality must be broken down into smaller, more controllable parts. Each software quality criteria may be broken down into its component parts. Instead, you may think of the components of quality as individual characteristics. Using the perspective of a sub-factor study of software quality, a quality assurance framework is provided. Below, we dissect each of these standards into its constituent pieces, discussing the amount of studies supporting it, the general agreement among academics, and the author's personal evaluation:

Efficiency

The phrase "efficiency factor" describes a product's ability to maximize the use of available resources while still meeting the demands of its consumers in a particular context. The discovered efficiency-related characteristics are broken out in detail in Table.

Table 1: Efficiency Sub-Factor

S. No	Sub-factor	Description
1	Time Behaviour	Product's ability to response time for a given throughput.
2	Resource Behaviour	Ability to use resource optimally to complete the task in terms of i.e. memory, CPU, disk, network usage, etc.
3	Efficiency Compliance	Maturity to obey standards and regulations regarding efficiency issues in specified environment.
4	Reply time	Ability to respond with output
5	Processing speed	Rate at which the data is converted into information.
6	Execution efficiency	Product's run time efficiency of the software.
7	Hardware independence	Degree to which the software is dependent on the underlying hardware.
8	Robust	It is the degree to which an executable work product continues to function properly under the abnormal condition or circumstances.

Maintainability

Maintainability refers to a product's adaptability to changes in the market and the ease with which it may be repaired and improved. Table lists the discovered maintainability factor sub-factors along with short explanations of each.

Table 2: Maintainability Sub-Factors

S.No	Sub-factor	Description

1	Analyzability	The capability of the software product to be diagnosed for deficiencies or cause s of failures in the software or for the parts to be modified to be identified.
2	Changeability	The capability of the software product to enable a specified modification to be implemented.
3	Stability	The capability of the software to minimize unexpected effects from modifications of the software.
4	Testability	The capability of the software product to enable modified software to be validated.
5	Correct ability	The capability of the software product to enable modified software to be validated.
6	Extensibility	The ease with which minor defects can be corrected between major releases while the application or component is in us e by its users.
7	Reusability	It is the ease with which an application or components can be enhanced in the future to meet the changing requirements.
8	Modularity	The rate to which the used components of the product can be reused on another product or system.
9	Adaptiveness	The rate to which the product is built from separate components so that change to one component has minimal impact on the other components of the product.
10	Perfectiveness	It is the ability of the product to accept the new environment, new hardware, new operating system, new supporting software.
11	Preventiveness	It is the ability of the product to anticipate future problems.
12	System age	It is the period since the first release of the product.
13	Understandability	The capability of the software product to enable the user to understand whether the software is suitable and how it can be used for particular tasks and conditions of use.
14	Documentation	Provision of programmer's manual that explains implementation of components.
15	Error debugging	It is the meantime to debug, find and fix errors.
16	Maintainability Compliance	The rate of how well product adhere s the standards and regulations regarding maintainability.

Portability

The portability of a product is evaluated by how readily it may be transferred from one location to another. All the identified determinants for mobility are summarized and described in Table.

Table 3: Portability Sub-Factors

S.No	Sub-factor	Description
1	Adaptability	The capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered.
2	Install ability	The capability of the software to be installed in a specified environment.
3	Coexistence	The capability of the software to coexist with other independent software in a common environment sharing common resources.
4	Replace ability	The capability of the software to be used in place of other specified software in the environment of that software.
5	Portability Compliance	The rate of how well product adheres the standards and regulations regarding portability.
6	Conformance	It is the rate to which the product meets the requirement defined in the SRS and design specification.
7	Reusability	It is the ability of the product to be used more than once and also to be used in different environments.
8	Transferability	It is the effort to transfer the product from one to another hardware and also from one to another operating system.
9	Flexibility	It is the products ability to be usable in all possible conditions for which it was designed.

Reliability

The term "reliability" refers to the frequency with which a product or component performs as expected under certain conditions and within a specified time period. The identified components of dependability are summarized in Table along with illustrative examples.

Table 4: Reliability Sub-Factors

S.No	**Sub-factor**	**Description**
1	Maturity	The capability of the software to avoid failure a s a result of faults in the software.
2	Fault Tolerance	The capability of the software to maintain a specified level of performance in case of software

		faults or of infringement of its specified interface.
3	Accuracy	Precision of computations and output.
4	Completeness	Degree to which a full implementation of the required functionalities has been achieved.
5	Recoverability	The capability of the software to reestablish its level of performance and recover the data directly affected in the case of a failure.
6	Survivability	It is the degree to which the essential services continue to be provided in spite of either accidental or malicious harm.
7	Consistency	It is the us e of uniform design and implementation techniques and notation throughout a project.
8	Simplicity	It is the ease with which the software can be understood.
9	Error tolerance	It is the degree to which a product continues to function properly despite the presence of erroneous input.
10	Statistical behaviour	The portability that the software will operate a s expected over a specified time interval.
11	Availability	The rate to which the component or system is operational and accessible for us e when required.
12	Integrity	The rate with which the component prevents the unauthorized modification of or access to system data.
13	Reliability Compliance	The rate of how well product adhere s the standards and regulations regarding reliability.

Usability

One aspect of a product's quality is its usability, or how well it works in the hands of its intended audience to do the specified task in the specified environment. The usability factor's subfactors are listed and briefly described in Table.

Table 5: Usability Sub-Factors

S.No	**Sub-factor**	**Description**
1	Understandability	The capability of the software product to enable the user to understand whether the software is suitable and how it can be used for particular tasks and conditions of use.
2	Learn ability	The capability of the software product to enable

		the user to learn its applications
3	Operability	The capability of the software product to enable the user to operate and control it.
4	Attractiveness	The capability of the software product to be liked by the user.
5	Ease of Use	The rate to which the user finds the product easy to operate and control.
6	Communicativeness	Ease with which inputs and outputs can be assimilated.
7	User friendly	Ease with which the component can be operated.
8	Accessibility	It is the degree to which the user interface enables users with common or specified disabilities to perform their specified task.
9	Customer satisfaction	It is the degree of the user's contentment in the usage of the component.
10	Documentation	It is the availability of manuals and other supporting documents for support of the user in its operation
11	Training	Ease with which the new users can use the system.
12	Usability Compliance	The rate of how well product adheres the standards and regulations regarding usability issues in specified environment.

CONCLUSION

These metrics are now referred to as the object-oriented metrics. It is possible that the software developer will make use of the framework since it facilitates enhancing the quality of the software component in response to the values of the software metrics. The quality assurance framework was developed after a thorough mapping of quality factors, sub-factors, and software metrics. The metrics may be calculated with a high degree of precision and understood with little time and effort investment at any time throughout the development process. Since the framework shows a correlation between metrics and quality characteristics, it may help to enhance software quality.

REFERENCE:

1. Mendivelso, L.F., Garcés, K. & Casallas, R. Metric-centered and technology-independent architectural views for software comprehension. J Softw Eng Res Dev 6, 16 (2018). https://doi.org/10.1186/s40411-018-0060-6

2. Durga Patel, Pallavi (2016) Software Reliability: Metrics. International Journal of Computer Applications (0975 – 8887) Volume 156 – No 5, December 2016

3. Li, Yan-Fu & Jia, Chuanzhou. (2021). An overview of the reliability metrics for power grids and telecommunication networks. Frontiers of Engineering Management. 8. 10.1007/s42524-021-0167-z.

4. Maatouk, Ali & Ayed, Fadhel & Biao, Shi & Li, Wenjie & Bao, Harvey & Zio, Enrico. (2023). A Framework for the Evaluation of Network Reliability Under Periodic Demand.

5. Kaur, Gurpreet and Kailash. bahl. “Software Reliability, Metrics, Reliability Improvement Using Agile Process.” (2014).

6. Tariq Hassain Sheakh, Vijaypal Singh, “Taxonomical Study of Software Reliability Growth Models”,International Journal of Scientific Research Publications, Vol.2, Issue 5, pp-1-3 May 2012

7. Baldwin, C. et al. 2014. Hidden Structure: Using Network Methods to Map System Architecture.

8. Curtis, B. et al. 2012. Estimating the Principal of an Application’s Technical Debt. IEEE Software. 29, 6 (Nov. 2012), 34–42. DOI:https://doi.org/10.1109/MS.2012.156.

9. Ernst, N.A. et al. 2015. Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (New York, NY, USA, 2015), 50–60.

10. Fenton, N. and Bieman, J. 2014. Software Metrics: A Rigorous and Practical Approach, Third Edition. CRC Press, Inc.

11. Ferenc, R. et al. 2014. Software Product Quality Models. Evolving Software Systems. T. Mens et al., eds. Springer Berlin Heidelberg. 65–100.

12. Jabangwe, R. et al. 2014. Empirical evidence on the link between objectoriented measures and external quality attributes: a systematic literature review. Empirical Software Engineering. 20, 3 (Mar. 2014), 640–693. DOI:https://doi.org/10.1007/s10664-013-9291-7.

13. Kazman, R. et al. 2015. A Case Study in Locating the Architectural Roots of Technical Debt. Proceedings of the 37th International Conference on Software Engineering - Volume 2 (Piscataway, NJ, USA, 2015), 179–188.

14. Kruchten, P. et al. 2012. Technical Debt: From Metaphor to Theory and Practice. IEEE Software. 29, 6 (Nov. 2012), 18–21. DOI:https://doi.org/10.1109/MS.2012.167

15. MacCormack, A. and Sturtevant, D.J. 2016. Technical debt and system architecture: The impact of coupling on defect-related activity. Journal of Systems and Software. 120, (Oct. 2016), 170–182. DOI:https://doi.org/10.1016/j.jss.2016.06.007.