Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 23 Issue 1 Version 1.0 Year 2023
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Application of Meta-Programming Techniques for Accelerating Software Development and Improving Quality

By Amirali Kerimovs

Annotation- A contemporary software tool has been devised to evaluate software quality through metric analysis techniques. This tool calculates pertinent metrics utilizing quality indicators and establishes a composite quality indicator value for software products. The intricacies of software quality assessment processes have been elucidated, including the examination of software quality's standardization as well as the presentation level of its model. This enables the potential for enhancement through the formulation of suitable criteria for quality assessment, refining models for metric analysis, and quantitatively measuring quality across all phases of project implementation. Notably, the use of metric analysis to gauge software quality reveals a lack of standardized metrics, resulting in varying assessment methods and metrics from different measurement system providers. Interpreting metric values also proves challenging for most software users due to a lack of clarity and informativeness. Furthermore, it has been discovered that while decisions based on cost, development duration, and designer company reputation influence software implementation choices, they do not always guarantee optimal software quality.

Keywords: *software engineering, project management, software project, quality assessment criteria, software quality indicators, comprehensive quality indicator.*

GJCST-C Classification: *ACM Code: D.2.11*

Strictly as per the compliance and regulations of:

Application of Meta-Programming Techniques for Accelerating Software Development and Improving Quality

Amirali Kerimovs

Annotation- A contemporary software tool has been devised to evaluate software quality through metric analysis techniques. This tool calculates pertinent metrics utilizing quality indicators and establishes a composite quality indicator value for software products. The intricacies of software quality assessment processes have been elucidated, including the examination of software quality's standardization as well as the presentation level of its model. This enables the potential for enhancement through the formulation of suitable criteria for quality assessment, refining models for metric analysis, and quantitatively measuring quality across all phases of project implementation. Notably, the use of metric analysis to gauge software quality reveals a lack of standardized metrics, resulting in varying assessment methods and metrics from different measurement system providers. Interpreting metric values also proves challenging for most software users due to a lack of clarity and informativeness. Furthermore, it has been discovered that while decisions based on cost, development duration, and designer company reputation influence software implementation choices, they do not always guarantee optimal software quality.

Keywords: software engineering, project management, software project, quality assessment criteria, software quality indicators, comprehensive quality indicator.

I. Introduction

Considering the multi-faceted nature of software quality, a combination of these metrics is used for evaluation. Weighting factors, established by experts, are applied to individual metrics based on the dominant quality criteria. These combined indicators provide a comprehensive assessment of software quality. Extensive complexity metrics are particularly relevant during the design phase, while subsequent stages refine the value metrics.

In accordance with ISO [1] standards, quality pertains to the extent of alignment between relevant attributes and stipulated requirements. As defined by [10], quality signifies the entirety of features and traits within a product, process, or service, ensuring the capability to fulfil anticipated or declared needs. In accordance with [3], software quality refers to the extent of its possession of the requisite combination of attributes. Essentially, software quality reflects the degree to which software aligns with specified requirements.

The challenge is to ensure the desired software quality while recognizing that an unknown number of errors and defects persist within complex software systems, necessitating their containment or reduction to an acceptable level. Consequently, a pivotal objective within the modern software life cycle is the assurance of software product quality [4].

II. Literature Review

Software quality is contingent upon the quality of methods and tools employed throughout its complete life cycle. Practical assessment of program quality is crucial not only upon completion but also during the design and development phases. The predicted or estimated quality of a software product comprises attributes evaluated or addressed at each life cycle stage, grounded in process quality and technological support [6].

The Software Development Life Cycle (SDLC) embodies a model depicting software creation and usage across various stages, commencing from the point of need identification and culminating in its retirement from user utilization. Numerous SDLC models exist, with three classified as foundational by international standards [4]: waterfall, incremental, and spiral.

During the design phase, establishing a set of quality requisites is vital: structure requirements for the software system (PS); air navigation specifications; user interface design prerequisites; multimedia component requisites for aircraft; usability demands; and technical prerequisites. The design stage formulates the response to the question, "How will the software system realize the imposed requirements?" Information flows during the software design stage [9] encompass software requirements portrayed through informational, functional, and behavioral analysis models. The information model outlines the data the software must process as per the customer's specifications. The functional model delineates a roster of information processing functions and software system modules. The behavioral model captures the desired system dynamics (operational modes). Concluding the design phase entails data

Author: Independent Researcher, Riga, Latvia.
e-mail: kerimovsoftdev@gmail.com

development, architecture formulation, and procedural software development.

Various approaches are employed for evaluating quality indicators, as outlined in standard [6]: measurement, recording, computation, expert assessment, and their combinations. Measurement involves specialized software tools to gather data on software characteristics such as volume, lines of code, operators, branches, entry/exit points, and more. Recording tracks factors like execution time, failures, and software start/end instances. Computation relies on statistical data collected during testing, operation, and maintenance to estimate indicators like reliability, accuracy, and stability. Expert assessment involves a panel of experienced evaluators who rely on intuition and experience rather than direct calculations or experiments. This method is used for reviewing programs, codes, documentation, and software requirements to assess factors like analyzability, documentation quality, and structured design [11].

In this context, the spiral life cycle model allows for the early assessment of software quality using a combination of calculation and expert evaluation techniques during the design phase.

1. *The Purpose of the Article:* Is to develop an adequate tool for determining the quality of software using the methods of metric analysis, which will make it possible to calculate the appropriate metrics with the help of quality indicators and determine the value of a complex indicator of the quality of a software product.
2. *Presentation of the Main Material:* The valuation of software can take the form of its monetary cost or be expressed through alternative means. Typically, clients hold their own notions regarding the maximum investment they're willing to make and the subsequent returns they expect, contingent on the software achieving its core objectives. The client's perspective might also encompass the software's functionality and specific expectations concerning its quality.

Typically, a client's initial focus revolves around the functional capacities of the software, often overlooking quality considerations, let alone the associated development costs. Consequently, during the initial phases of a software project, the focus may shift towards ensuring the client comprehends both the benefits of software utilization and the developmental expenses tied to attaining a particular level of software quality. Ideally, these crucial determinations should primarily occur when establishing user requirements for the software. Nonetheless, these considerations remain pertinent throughout the entirety of the software's development process. While standardized decision-making protocols might not exist, systems engineers must possess a clear understanding of the diverse avenues leading to specific levels of software quality and the corresponding developmental costs. This clarity aids in the anticipation of the overall expenditure associated with executing the software project.

To visually illustrate the correlation between the implementation costs of a software project and the level of software quality, we delve into the particulars of an information protection system's (ISI) development. Specifically, we analyze its functional model while bearing in mind its inherent intricacies. This model omits the depiction of information's inherent value- the object of confidentiality (e.g., bank deposit accounts or access codes), as such information retains its value over time. To facilitate understanding, the diagram introduces specific notations:

- *P:* Probability level indicating the extent of information protection (approximately $0.6 \leq P < 1.0$).
- *Z (P):* Permissible costs associated with safeguarding information as a function of the required level of protection. These costs rise as the demands for higher levels of information protection increase.

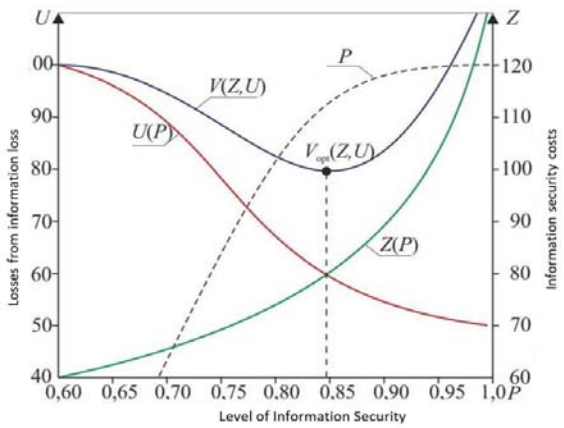

Fig. 1: The Main Features of the Process of Evaluating the Quality of SHI

The aspiration to achieve an exceedingly high level of information protection often ushers in a substantial escalation in expenses, potentially surpassing the intrinsic value of the information being safeguarded. The conceivable losses, or damages, borne by the information owner U(P), stemming from an insufficient level of protection, form a direct correlation with the extant level of protection, denoted as P. The diagram illustrates how the sum of Z(P) and U(P) collectively shapes the overall costs V(Z, U) associated with ensuring information security. Within this context, the optimal threshold for safeguarding, marked as $V_{opt(Z, U)}$, corresponds to the point where the combined costs of protection (Z(P)) and potential losses (U(P)) are minimized. This equilibrium signifies the balance between investing in protection measures and the potential losses due to inadequacies in protection, effectively preventing both excessive expenditures and heightened risks.

Striving to surpass this equilibrium point inevitably triggers a sharp escalation in Z(P), the expenses tied to information protection. Conversely, lowering the level of protection would lead to an escalation in potential losses, U(P), stemming from the compromised functionality of the system handling the safeguarding of information.

Consequently, the notion of software quality is intrinsically relative, gaining true comprehension within the context of real-world application scenarios. Therefore, the quality requirements established by relevant standards must be carefully aligned with the circumstances of the software's use and its specific domain of application.

Software quality embodies several critical components, notably:

1. *Quality of Software Development Processes:* This pertains to the efficacy, efficiency, and adherence to best practices during the creation of the software.
2. *Quality of Software Project Products:* Referring to the final software products themselves, encompassing attributes like functionality, reliability, and performance.
3. *Quality of Software Support or Implementation:* Addressing the competence and effectiveness of the software's implementation, utilization, and ongoing support.

This multi-faceted perspective illustrates how software quality is a nuanced and multifarious concept, emerging as a result of intricate interplays between development processes, product attributes, and the operational support environment.

The element concerning software development processes plays a pivotal role in gauging the extent of formalization and the inherent reliability of these processes across every stage of software evolution. This facet is intricately interwoven with the critical activities of verification and validation (abbreviated as V & V), which entail scrutinizing and endorsing the interim outcomes generated during these processes. The diligent pursuit of error detection and eradication within the finalized software is facilitated through rigorous testing methodologies. These approaches serve to diminish the occurrence of errors, thereby elevating the overall quality of the forthcoming software product.

Fostering excellence in the software project's products is underpinned by the meticulous application of procedures that govern the oversight of intermediate project deliverables at all developmental stages. These steps encompass meticulous checks to ascertain the attainment of the requisite quality standards. Furthermore, modern methodologies and resources dedicated to supporting the software product are harnessed to bolster this quality pursuit. The efficacy of software implementation hinges upon a symbiotic combination of factors, including the expertise of service personnel, the functional prowess of the software product, and the meticulous adherence to well-defined implementation protocols.

The framework for software quality is structured across four distinct levels of representation, as expounded by [7].

1. *First Level:* This pertains to the delineation of software quality's inherent attributes or indicators. Each of these indicators offers a unique vantage point from an end-user's perspective, encapsulating diverse facets of software quality. Established standards such as ISO/IEC 9126, DSTU 2844-1994, DSTU 2850-1994, and DSTU 3230-1995 elucidate a comprehensive quality model comprising six key characteristics or quality indicators for software: functionality, reliability, usability, maintainability, efficiency, and portability.
2. *Second Level:* Subsequent to the first tier, the focus shifts to expounding software quality attributes germane to each distinctive characteristic. This intricate articulation delves into the finer nuances and multifaceted features that contribute to each attribute. This assemblage of attributes subsequently underpins the metric analysis of software quality, enabling a comprehensive assessment across a spectrum of dimensions.

Therefore, a comprehensive understanding of the intricacies involved in assessing software quality has been elucidated. This endeavor encompasses a meticulous exploration of the very essence of software product quality, a subject subjected to the tenets of standardization. Concurrently, an in-depth investigation into the strata of the software quality model's representation has taken place. This discerning analysis has not only unveiled latent dimensions for refinement but also paved the way for the construction of judicious requisites tailored to the assessment of quality criteria.

Furthermore, it has facilitated the enhancement of the metric models used for the analysis of software quality and the calibration of quantitative measurement methods across every juncture of software project implementation.

The empirical landscape reveals a significant proportion of software errors manifesting during the critical phase of requirement formulation, accounting for 10-23% of the entire spectrum. A conspicuous trend emerges whereby the magnitude of software intricacy is positively correlated with the prevalence of conceptual errors within this stage (Hrytsiuk, 2018). It is noteworthy that as the complexity of software augments, the propensity for conceptual discrepancies becomes more pronounced. This phenomenon often arises due to the inherent challenges of grappling with extensive and multifaceted requirements.

Moreover, the formulation of software requirements engenders a vulnerability to information losses, primarily stemming from the interplay of incomplete articulation and variances in comprehending customer needs and the contextual milieu within the requirements specification. This predicament is particularly acute within software projects traversing the intersections of diverse domains of knowledge. It is unequivocally established that software endeavors marred by incomplete requirements and ill-prepared specifications invariably confront hurdles impeding successful realization.

Consequently, in light of such circumstances, the judicious recourse of subjecting the software requirements specification to rigorous analysis by impartial experts assumes paramount significance. This proactive measure serves as a pivotal bulwark against errors cascading through successive stages, encompassing requirement formulation, software architecture design, and subsequent construction phases [3].

Informed by the data presented in Table 1, a salient revelation surfaces wherein errors originating from requirement formulation and architectural design precipitate as a substantial portion, accounting for 25-55% of the overall error spectrum. It is compelling to note that this proportion is notably exacerbated as the magnitude of software complexity escalates, signifying a heightened susceptibility to errors during the nascent stages of development.

Table 1: Distribution of Errors Assumed at Different Stages of Software Development [2]

Software development stage	Volume of Software/Share of Errors,%				
	2K	8K	32K	128K	512K
Formulation of requirements	10	15	20	22	23
Architecture design	15	19	25	28	32
Designing	75	66	55	50	45

Consequently, we hold the conviction that an imperative avenue for further exploration lies in investigating the potential of harnessing metric analysis to ascertain software quality through insights gleaned from software requirements specifications. As a decisive stride towards this objective, we have conceived a bespoke software tool (depicted in Figure 2) meticulously architected to evaluate software quality via metric analysis. More specifically, it capitalizes on the utilization of quality metrics replete with both precise and prognostic values. A salient distinction of our tool, differentiating it from established counterparts, resides in its adeptness to dissect software based on ascertained metric values, prognosticating the trajectory of its developmental trajectory. Furthermore, the tool orchestrates a sequence of computations culminating in the generation of a comprehensive dataset, which in turn enables an extrapolation of metric outcomes. This inductive methodology endows the capacity for a quantitative assessment of the project's product quality and engenders the anticipation of developmental software quality attributes.

To orchestrate a systematic software development risk management paradigm, a project manager assumes the pivotal role of foretelling the precursors to potential predicaments, the emergence of adversities, or the occurrence of unfavorable events. This endeavor unfolds as an art of forecasting, grounded in empirically substantiated inferences regarding plausible trajectories of software project management execution, juxtaposing alternative courses and temporal dynamics. The interplay of forecasting management decisions intersects intimately with strategic and tactical contours delineating the risk landscape of program project implementation.

The development of the aforementioned software tool was steered within the contours of Microsoft Visual Studio. NET 2017 development environment. Significantly, this tool operates autonomously, devoid of any tether to internet connectivity. The commencement of the task hinged upon an intricate process of prototyping the user interface, progressively infusing augmentative functionalities into the software tool's architecture. The outcome of this endeavor, culminating in the software tool's user interface, is prominently featured in Figure 5.

A cornerstone of the software's architecture is encapsulated within the MetricsQualitySoftware.cs

class, an abstract entity that encapsulates pivotal functionalities essential for metric evaluation. This class is equipped with a suite of cardinal methods that underpin its operational dynamics. These include functions such as modifying metric parameter values (ChangeValue_OfParameter), accessing parameter names (GetNameOfParameter), furnishing fundamental metric information (SetInformation_OfMetric), illuminating metric definitions (ShowDescription_Of Metric), establishing metric parameter value functionality (SetAllParameters), ascertaining metric values (FindMetric), and facilitating metric parameter reference information display (ClearAllParameters_OfMetric).

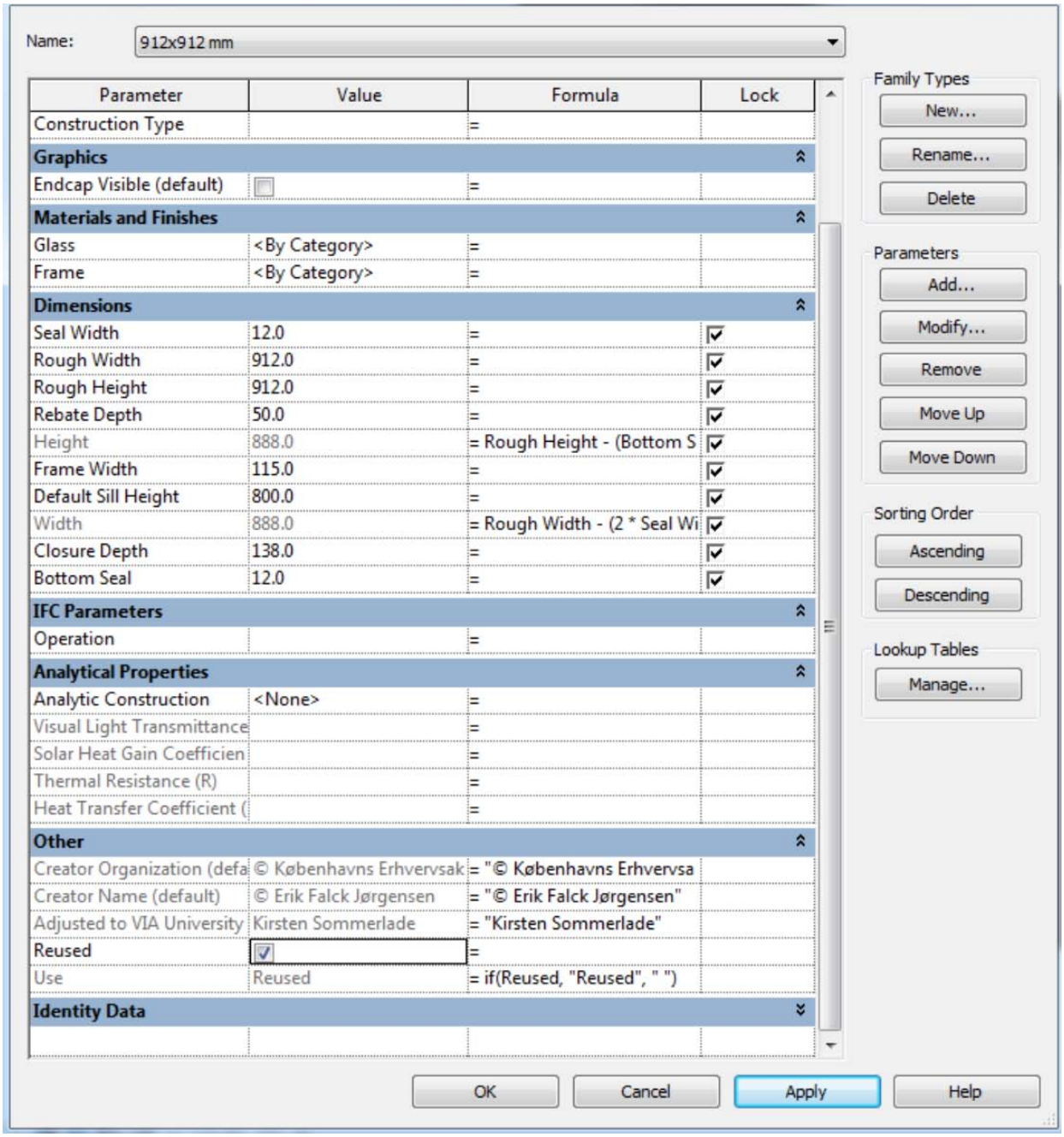

Fig. 2: Windows of the Software Tool for Determining Software Quality by Metric Analysis Methods

To facilitate the seamless manipulation, input, and retrieval of data within specific cells of the DataGrid table, the software employs the DataGridHelper.cs class. This crucial class encompasses key methods that empower efficient data handling: first, the ability to retrieve the value of a designated cell by specifying the row and column indices (GetCell), and second, the capability to retrieve data based solely on the row index (GetRow).

The architecture encompasses a series of distinct metrics classes, namely CHPmetric.cs, CPP metric.cs, MBQmetric.cs, MMTmetric.cs, RUPmetric.cs,

CCCmetric.cs, CPTmetric.cs, SCCmetric.cs, SCTmetric.cs, SDTmetric.cs, SQCmetric.cs, FPmetric.cs, LCmetric.cs, DPmetric.cs. Each of these classes is crafted to inherit from the abstract MetricsQualitySoftware.cs class, thereby inheriting its foundational structure, while also seamlessly overriding its methods to align with the specific requisites of their respective contexts.

The design also embraces auxiliary model classes such as MyTableInfo_OfAllMetrics.cs, MyTable Info_OfAllParameters.cs, and MyTableInfoCharacteristic _forMetricFp.cs. These model classes are meticulously sculpted to serve as repositories for recording the data harvested from distinct DataGrid tables. They also boast the capacity to efficiently dispense the synthesized tabular information.

Illustrating the software tool in action, let's delve into an illustrative scenario that underscores its operational prowess. In a bid to engender a comprehensive understanding of the tool's underlying mechanics, a meticulous examination is undertaken to ascertain both the quality and overarching forecasted assessment of the developmental process. This exploratory analysis culminates in the extraction of essential input data pertinent to the metrics, as delineated in Table 2. Following the meticulous input of all pertinent metrics' parameters and their subsequent calculation utilizing the software tool, a comprehensive dataset is curated, pivotal for constructing an informed forecast concerning the software's quality attributes.

The software tool instantiates the delivery of diverse representations of the culled information. Foremost, it furnishes an all-encompassing tabular display of metric values (Figure 4), thereby proffering a succinct overview of the analytical outcome. Furthermore, it leverages graphical illustrations to visually convey the insights, employing both pie charts and histograms (Figure 3) to distill the intricacies of the analysis. This holistic visualization augments the clarity and interpretability of the results. Conclusively, the software tool culminates in the holistic assessment of the software's quality, synthesizing the intricate array of metrics and their concomitant implications.

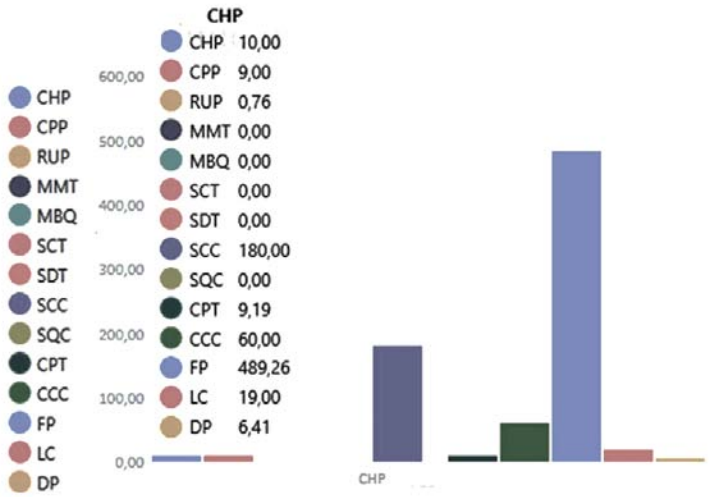

Fig. 3: Graphic Presentation of Results in the form of a Histogram

Table 2: Input Data for the Software Tool

No. for/p	Parameter name	Value
1	How many times will the module actually access the global variable	265
2	How many times a module could access a global variable	348
3	The number of lines of program code	4670
4	The duration of the implementation of the software project	126
5	Part of the software architecture design stage in the process of its development	2
6	Number of module errors	108
7	Number of modules	345
8	Expected number of lines of function source code	54, 34, 28, 58, 6
9	Estimated cost to develop a feature line	1
10	Part of the stage of verification , validation and testing of software in the process of its development	1
11	Part of the product quality control stage of the project at the verification , validation and testing stages	2
12	The expected number of lines of source code in a similar function	45, 30, 25, 50, 5

13	Productivity of the process of developing a similar function	2
14	Predicted performance of the software development process	3
15	The number of external inputs to the function that affect the executed function differently	5, 11, 6, 5, 34
16	The number of external outputs of the function for significantly different algorithms and non-trivial functionality	8, 56, 7, 7, 12
17	Number of external requests	3, 3, 10, 2, 4
18	Number of internal logical files or unique logical groups of user data	1, 1, 53, 5, 7
19	Number of external logical files or unique logical groups of user data	4, 1, 1, 8, 2
20	Connectivity level	functional
21	Clutch type	by content
22	Number of functions	5

№	Result	ID	Metric name
1	10		SNA Connectivity metric
2	9		CPP Clutch metric
3	0.761		Rup metric for accessing global variables
4	0		MMT model modification time metric
5	0		MBQ metric for the total number of errors found when checking models and D
6	0		SCT metric for predicting total software development time
7	0		SDT software design process completion time metric
8	180		SCC metric for the expected cost of software development
9	0		SQC metric for predicting the cost of software quality control
10	9.193		CPT metric for predicting software development performance
11	59.999		CCC metric for predicting the cost of implementing program code
12	489.255		FP metric for predicting function size
13	19		LC metric for predicting Labor Cost Estimates based on the Boehm model
14	6.414		DP metric for predicting project duration estimates based on the Boehm model

Fig. 4: Obtained Results of Metrics

In the realm of software engineering, a sophisticated and advanced software tool has been meticulously crafted with the explicit purpose of ascertaining the quality of software through the adept utilization of metric analysis methodologies. This innovative tool transcends mere analysis, extending its reach into the realm of forecasting the prospective efficacy of the software development process. A notable feature of this software is its intrinsic capability to curate a comprehensive dataset that plays a pivotal role in the determination of a multifaceted indicator encapsulating the quality of the resultant software product. To concretize the tool's operational essence, an illuminating example elucidating its function is presented. Moreover, a comprehensive research endeavor has been undertaken to scrutinize and discern the quality of select software entities, culminating in a holistic prognostic-cation concerning the triumphant trajectory of their developmental journey.

This contemporary software marvel, meticulously fashioned to evaluate software quality, harnesses the power of metric analysis paradigms, enabling the seamless translation of quality indicators into precise metrics. Through this harmonious synergy, the intricate fabric of software quality is meticulously woven, ultimately manifesting in the articulation of a multifaceted metric indicative of software excellence. An in-depth examination of the research findings precipitates several salient conclusions, shedding luminous insight into the complex tapestry of software quality assessment.

The labyrinthine path of software quality assessment is unveiled, wherein the fundamental tenets of this process are dissected with precision. The concept of software product quality, assuming a central role in standardization, undergoes profound analysis. Simultaneously, the stratification of the software quality model is scrutinized, thereby establishing a robust framework conducive to iterative enhancements. This involves the meticulous refinement of quality assessment criteria, augmentation of metric analysis models, and the development of methods for quantitative measurement. Consequently, this holistic approach encompasses all facets of software project realization.

III. Conclusion

For gauging software quality during the design phase, the spiral model of the software life cycle emerges as the most fitting approach. Examining the methods of assessing quality indicators (metrics) reveals that solely calculation and expert measurement techniques are viable at this stage. This is due to the inability to measure characteristics of software that hasn't been developed and the impracticality of recording execution moments for non-existent software.

1. The bedrock of successful software project implementation is unveiled through meticulous exploration. The crux of this revelation lies in the ardent aspiration of project managers to engender software solutions that bear inherent value. This value is both a catalytic agent in solving intricate challenges and a cornerstone in accomplishing tactical and strategic objectives. A nuanced understanding of this value leads to the discernment that it can be encapsulated either in monetary terms or via alternative metrics. This profound insight is fortified by the recognition that customers harbour their distinct perception of maximum investment thresholds, intertwined with the anticipated returns rooted in the attainment of overarching objectives through software deployment. Moreover, this discernment extends to the articulation of software functionality and the quality paradigm, encapsulating the customer's discerning expectations.
2. The unique contours of metric analysis as a conduit for assessing software quality come to the fore. A pivotal observation is the absence of homogenous standards for metrics, resulting in diverse methodologies proposed by individual system providers to gauge software quality. The enigmatic interpretation of metric values surfaces as an additional challenge, as these values often elude the comprehensive grasp of the majority of users. The interplay of these facets underscores the complexity inherent in selecting a software implementation route. As a corollary, pivotal determinants in this selection process include financial viability, temporal dynamics, and the reputation of the design company. Notably, however, these determinants do not inexorably guarantee the desired software quality outcome.
3. A groundbreaking feat materializes in the form of a bespoke software tool architected to gauge software quality by harnessing the potential of metric analysis methodologies. This innovative tool ingeniously extends its functionality beyond analysis, adroitly projecting the future efficacy of the developmental process. The hallmark of this innovation is its adeptness in formulating a dataset of paramount importance, intricately intertwined with the determination of a comprehensive quality indicator encompassing the software product's inherent excellence.
4. A culmination of insightful observations culminates in the crystallization of pertinent recommendations, offering guidance in the employment of the developed information visualization technique. This technique augments the interpretability and efficacy of software quality assessment, paving the way for enhanced decision-making and informed trajectories in software development endeavors.

References Références Referencias

1. Bozic, Velibor. (2023). Methods and Techniques of Software Development. 10.13140/RG.2.2.27516.00645.
2. Bozkurt, Erkam. (2022). The usage of cybernetic in complex software systems and its application to the deterministic multithreading. Concurrency and Computation: Practice and Experience. 34.10.1002/cpe.7375.
3. Cheng, Kwok Sun & Huang, Pei-Chi & Ahn, Tae-Hyuk & Song, Myoungkyu. (2023). Tool Support for Improving Software Quality in Machine Learning Programs. Information. 14. 53. 10.3390/info14010053.
4. Hong, Sirui & Zheng, Xiawu & Chen, Jonathan & Cheng, Yuheng & Zhang, Ceyao & Wang, Zili & Yau, Steven & Lin, Zijuan & Zhou, Liyang & Ran, Chenyu & Xiao, Lingfeng & Wu, Chenglin. (2023). MetaGPT: Meta Programming for Multi-Agent Collaborative Framework.
5. Kovari, Attila & Katona, Jozsef. (2023). Effect of software development course on programming self-efficacy. Education and Information Technologies. 1-27. 10.1007/s10639-023-11617-8.
6. Luo, Ke & Deng, Wei. (2023). Software engineering database programming control system based on embedded system. Applied Mathematics and Nonlinear Sciences. 10.2478/amns.2023.1.00473.
7. Nagalakshmi, S. (2023). Software Development Techniques In Current Scenario. Data Analytics and Artificial Intelligence. 3. 50-53. 10.46632/cllrm/3/2/10.
8. Romli, Rohaida & Nordin, Noorazreen & Omar, Mazni & Mahmod, Musyrifah. (2018). A Review on Meta-Heuristic Search Techniques for Automated Test Data Generation: Applicability Towards Improving Automatic Programming Assessment. 896-906. 10.1007/978-3-319-59427-9_92.
9. Shafiq, Muhammad & Alghamedy, Fatemah & Jamal, Nasir & Kamal, Tahir & Daradkeh PhD., P. Eng, Dr. Yousef & Shabaz, Dr. Mohammad. (2023). Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality. IET Software. 17. n/a-n/a. 10.1049/sfw2.12091.
10. Stuikys, Vytautas & Damasevicius, Robertas. (2013). A Background of Meta-Programming Techniques. 10.1007/978-1-4471-4126-6_3.
11. Tietz, Vanessa. (2021). Development of a Meta-language and its Qualifiable Implementation for the Use in Safety-critical Software.