Al-Rafidain Journal of Computer Sciences and Mathematics (RJCM)

www.csmj.mosuljournals.com

Stability Metrics for Object-Oriented Design in Software Engineering: A Literature Review

Anfal A. Fadhil[1, *], Taghreed Riyadh Alreffaee[2] and Asmaa' H. AL_Bayati[3]

[1,2,3]*Department of software, College of Computer Sciences and Mathematics, University of Mosul, Mosul, Iraq*
Emails: anfalaaf@uomosul.edu.iq, taghreed_reyad@uomosul.edu.iq, asmashade77@uomosul.edu.iq

Article information

Article history:
Received:1/5/2024
Accepted:29/7/2024
Available online:15/12/2024

Abstract

One of the best qualities for any software design is stability. Any initial adjustment to a design will have a significant effect on it if it lacks stability. And for greater software quality, software stability is crucial. Stable classes typically result in lower effort and expense for software maintenance. Thus, in the case when developing software, achieving class stability is a crucial quality aim. If designers could use certain predictors for predicting class stability before it happens, they will be able to make better decisions for improving class stability. This work presents the most significant metrics that scholars have utilized to quantify stability in object-oriented (OO) design, together with the most well-known studies that have employed these metrics.

Keywords:
Stability, Object-Oriented (OO) Design, Software Engineering.

Correspondence:
Author: Aanfal A. Fadhil
Email: anfalaaf@uomosul.edu.iq

I. Introduction

Any software system must have stability as a long-term attribute throughout its entire life cycle, from implementation and design to actual operation, maintenance, management, and evolution. A system might offer a solid foundation for facilitating technological advancements as well as affordable evolution and maintenance if it is designed and created with stability in mind [1]. Software longevity is going to become a highly sought-after attribute as dependence on software systems and services grows. A system with a long lifespan can withstand changes and still function mostly unchanged (e.g. maintenance, evolution and runtime changes). Since stability addresses the effects of changes, it is generally acknowledged as a property for reflecting these concerns. The quality problem is really two sides of the same coin, and longevity and stability are interdependent. Given that longevity heavily depends on the ability to maintain stability, a stable basis serves as foundation for building of high-quality, long-lasting systems [2]. Given that software maintenance is an expensive procedure, maintainability is a crucial component of software quality, and listed five sub-characteristics for maintainability, stability being one of them. Unstable software might result in significant maintenance expenses and workloads. Stable classes might help lower the costs and efforts of software maintenance because they are the fundamental building blocks regarding the software architecture in object-oriented systems [3]. Generally, there are three main perspectives on software stability. The first definition of stability is the ability of a system to withstand change. This means that if the software doesn't change between two versions, it will be referred to as stable. According to the second definition, software is considered stable if it does not experience any ripple effects after an addition or alteration. Therefore, stable entities are object-oriented entities (such as packages and classes) that do not have unintended consequences when they are modified. The software achieves its highest level of stability, according to the third criterion, provided that the content stays unchanged. In the case when the addition of

new content has no effect on the content that already exists, the software is still stable [4]. A variety of measures were established by researchers to assess stability quality of OO apps. Class-level stability metrics consider number of methods, percentage of added and altered methods, lines of code, and various class attributes in order to assess the stability of individual classes between versions [5]. Contributions: in this study discuss the state-of-the-art in OO design for software engineering with regard to stability in this study. The specific contributions in this study are as follows:

1- From the present software engineering literature, a characterization taxonomy for the idea of stability in the OO design for software engineering evolved as a software characteristic.

2- Aanbieding and providing an example of object-oriented design stability measures that are documented in the literature

3- Show shed stability metrics in relation to other quality factors for object-oriented design.

4- An overview and demonstration of the key findings for the study of stability in object-oriented software engineering design

This is how the rest of the paper is structured. The most significant object-oriented design stability measures are provided in Section 2. Section 3 outlines the study's objective. Section 4 provides the literature review. The work is concluded in Section 5, which suggests possible directions for future research.

II. Stability Metrics of Object-Oriented Design

At several levels, software stability was analyzed. This study focuses on stability at object-oriented design (class) level; the subsequent subsections provide an explanation of the suggested metrics to gauge class stability.

A. Class stability measurement with the use of lines of code (loc)

Class implementation instability (CII) is a statistic that Li et al. [6] devised to quantify the evolutionary change in a class's implementation. It calculates the proportion that modifications to design N + 1 have an impact on the class loc in design N. It has the following definition:

CII represents class loc change percentage.

CII is calculated for class A as described below:

In the design N, Q stands for loc of class A.

In design N + 1, loc of class A is represented by R.

$$CII = \left[\frac{R-Q}{Q}\right] \times 100 \quad \ldots (1)$$

According to Eq. (1), the calculation of CII involves dividing loc of class A in design N by class A loc in design N + 1, which is the difference between loc of class A in design N and design N. Since CII measures class's instability, a lower CII number is preferable. The main problem with CII is that even when a class has undergone several changes, CII metric still indicates that the class is stable if There are n lines added and n lines subtracted.

B. Class stability measurement by the use of percentage of changed and added approaches

A technique for determining if a class is stable when its interface does not change between versions was proposed by Grosser et al. [7]. For example: C is a class, and l(c_i) is the version I interface (method signatures) for C. The stability regarding class c could be computed through the comparison of $l(c_i)$ to $l(c_{i+1})$. It denotes percentage of $l(c_i)$ which is included in $l(c_{i+1})$

$$NS(c_{i\to i+1}) = {\#\cap(c_i, c_{i+1})}\Big/{\#l(c_i)} \quad \ldots (2)$$

Where

$$\#\cap(c_i, c_{i+1}) = l(c_i,) \cap l(c_{i+1})$$

Equation (2) illustrates how NS is computed: it is the product of the quantity of approaches in version i divided by the quantity of approaches shared by versions i and i+ 1. The NS value ranges from 1 (complete stability) to 0 (total instability). In spite of how many modifications were made to the methods' bodies, the NS metric just examines the methods' signatures. Thus, the metric indicates that the class is stable if the signatures of methods do not change even when the methods' bodies do.

C. Class stability measurement with the use of Number of Methods

A method for computing class stability depending on number of methods was suggested by Ratiu et al. [8]. With regard to class stability, they established the Stabi measurement, which takes into account that a class is altered if at least one approach is removed or added. Ratiu etal. therefore, concluded that if Number of Methods (NOM) remained unaltered, a class was stable between versions I–1 and I. They employed Eq. (3) in one version to determine class stability.

$$Stab_i(C, M) = \begin{cases} 1, M_i(c) - M_{i-1}(c) = 0 \\ 0, M_i(c) - M_{i-1}(c) \neq 0 \end{cases} \quad \ldots (3)$$

M is a measurement (NOM). According to (3), Stab metric can have a value of 0 or 1. When the value of a class is 1, it indicates stability, meaning that the number of methods hasn't changed between the two versions. A value of 0 indicates that the class is unstable and that the number of

methods had changed. One of the biggest drawbacks of Stab metric is that it indicates class stability even when the number of methods was updated, added, or removed.

D. Class Stability measurement with the use of multiple factors

A Class Stability Metric (CSM) has been introduced by Al-shayeb etal. [9] to quantify class stability for OO. 8 class attributes that impact class stability are taken into account by CSM. These characteristics are:

Inherited class name, Class interface name, Class variable access-level, Class access-level, Class variable, Method access-level, Method signature and Method body (code).

The purpose of CSM is measuring the version's stability in relation to the base version. Other class modifications among the other version and the base version were disregarded in favor of stable or unchanged class attributes for calculating CSM. If a property hasn't changed between version n and the base version, it's deemed unchanged. Every attribute is computed as follows:

$$Stab_{\text{Property}} = \frac{\text{Unchanged}_{\text{Property}}}{Nnmber_{\text{Property}}} \quad \ldots (4)$$

In which $Stab_{\text{Property}}$ represent stability in any class of property, $\text{Unchanged}_{\text{Property}}$ represent the quantity of the property's unaltered items and $Nnmber_{\text{Property}}$ represent the number of items regarding the property. For the purpose of calculating the stability regarding an OO class with the use of CSM, it is expected that each class properties have the same weight. and all of the properties is separately handled as can be seen in Eq. (4). Each property's stability is determined, and the total of the calculations yields the OO class stability, as Eq. (5) illustrates. A number of 1 indicates full stability for the class, whereas a value of 0 indicates full instability. (see Eq. (5))

$$Stability_{class} = \left(\begin{matrix} stab_{classAl} + stab_{interface} + stab_{inhr} + stab_{Mthd} \\ + stab_{var} + stab_{varAl} + stab_{MthdAl} + stab_{Body} \end{matrix}\right) / \text{properties}_{class} \quad \ldots (5)$$

In which $stab_{classAl}$ represent the percentage of the stability of the class access-level property, $stab_{interface}$ represent stability percentage of the class interface property, $stab_{inhr}$ represent stability percentage of the inheritance class name property, $stab_{Mthd}$ represent stability percentage of the class method signature property, $stab_{var}$ represent stability percentage of the regarding property of class variable, $stab_{varAl}$ represent stability percentage of the regarding access-level property of class variable, $stab_{MthdAl}$ represent stability percentage of access-level property for the class method, $stab_{Body}$ represent stability percentage of the class method body property and $\text{properties}_{class}$ represent number of the class properties.

E. Measuring Class Diagram Stability

in this method address every one of the classifier properties independently and determining change in relation to base version in order to quantify class diagram stability. By taking the subsequent actions, the class diagram stability may be measured [9]:

1- Create a metric for property changes in the classifier which quantifies the variations in each property. Eq. (6) and Eq. (7), which denote changes in classifier type and classifier relationship, respectively, are used for computing property changes.

2. As indicated by Eq. (8), Divided the total number of property change metrics by the sum of unique classifier properties. One plus number of unique classifier associations (where one represents classifier type) equals unique attributes. The value of classifier change can be normalized to be between 0 and 1 by dividing by total number of the unique classifier properties. A value of 1 is an indication means that from version I to version I + 1, every classifier property was altered.

3. Divide the total number of the class diagram base version classifiers by sum of classifier change metrics. This will result in normalizing result value to be between 0 and 1, in which one indicates that every class diagram classifier had been changed from versions i to i + 1. The result is divided by number of the base version classifiers.

4. Equation (9) is used to calculate overall class diagram stability metric. Furthermore, the final result has been normalized. Zero indicates that from versions I to I + 1, all of the classifiers have changed. Version I + 1 is hence unstable. One, on the other hand, indicates that version I + 1 is entirely stable since nothing has changed. In this method need to first ascertain if the classifier is existing in version i + 1 in order to count the potential modifications to the classifier attributes. The chosen identifier is used to accomplish this. The classifier change value will be maximal, or one, if the identification is removed. If not, will use Eq. (6) and Eq. (7) to calculate each classifier property change. Classifier type modifications are seen in Equation 6. Zero denotes that the type of the classifier had changed, either from class to interface or the other way around, while one indicates that the classifier type stays unchanged.

$$Ch(CT) = \begin{cases} 0, class\ type\ changed \\ 1, class\ type\ unchanged \end{cases} \quad \ldots (6)$$

Changes in classifier relationships are reflected in Equation 7. If the relation is changed to a different type or deleted, the change counts as zero in both scenarios. In the case when the relation stays unchanged, the change will be one.

$$Ch(CR) = \begin{cases} 0, Relationship\ deleted \\ 0, Relationship\ change \\ 1, Relationship\ unchange \end{cases} \quad \dots (7)$$

$$UCC = \frac{Ch(CT(i, i+1) + \sum_{CR=1}^{NUCR} Ch(CR(i, i+1))}{NUP} \quad \dots (8)$$

In which the classifier's UCC was unchanged. Each classifier's unchanged attributes are calculated using this metric. In class diagram classifier, NUCR stands for Number of Unique Classifier Relationships. The Classifier Relationship is denoted by the CR, Classifier Type is denoted by the CT, the Number of Unique Properties = (NUCR + 1) is denoted by NUP, where the type of classifier in the class diagram version is represented by 1.

$$SSM(i+1) = \frac{\sum_{C=1}^{NC} UCC(C)}{NC} \quad \dots (9)$$

The Structural Stability Metric is SSM. The class diagram's stability is calculated by this metric. Classifier C. UCC refer Unchanged Classifier. NC refer Number of Classifiers in first version.

The methods mentioned in section (2.1-2.5) are considered the most important equations for calculating Stability for Object-Oriented Design, as these equations contain factors that are considered basic keys to calculating Stability for Object-Oriented Design. As the mentioned equations (in section (2.1-2.4) calculate stability at the code level, it one final equation without integrating it with the rest of the equations according to published research.

The method mentioned in Section (2.5) is the currently widespread method for calculating Stability for Object-Oriented Design in diagrams level.

III. Goal of Study

With regard to object-oriented design, stability was measured in a number of studies and researches in literature. The goal of this study is to provide a Literature Review which includes the most important and recent research in stability for object-oriented design based on metrics. The strategy of this work includes research published in the period (2000-2022). the articles featured in this literature review in the field of object-oriented design, it have been published in conferences, journals, book chapters.

The following research topics have been addressed through a state-of-the-art literature review in order to be accessible to researchers who will study in this field in the future:

1- Which measures are most commonly used to assess the stability of object-oriented design?

2- Does object-oriented design stability refer to code or class diagram (UML) stability?

3- Which set of measurements are used for properties?

The need of the present work is to provide an overview of developments in the stability metrics of object-oriented design.

IV. Literature Review

This section contains stability measurements for object-oriented systems at the code and class diagram levels. Aside from the methods outlined in section 2, the following studies have been conducted:

In 2000, W.Li et al [6], the evolution of the object-oriented (OO) software was measured by the authors using 3 metrics, which are: System Design Instability (SDI), Class Implementation Instability (CII), and System Implementation Instability (SII). In addition to doing research of design instability, which looks at how a class's implementation may impact its design, such metrics have been utilized in order to follow the development of OO system in empirical investigation.

In 2002, David Grosser et al [10], the authors suggested using a case-based reasoning approach for predicting the stability of software items from relevant metric data. The technique presented here considers each item as a point in a multi-dimensional space, one dimension per metric, in which a distance function is defined. The stability of each new item is computed with respect to the nearest known case in the case base. The resulting predictive model fits well realistic situations in which the available data is neither of sufficient size nor representative enough to develop universally valid models. Indeed, the similarity-based prediction avoids the pitfalls of over-generalization of logical classification models. In this respect, our preliminary results show that a very straightforward CBR classifier (1-nearest neighbor, equal metric weights, no domain theory) can perform significantly better than a decision tree drawn from the same dataset.

In 2003, David Grosser etal [7], have suggested the use of stress test results and relevant metric data for the prediction of the stability of software item using case-based reasoning method. The method looks at structural similarities across classes and uses software metrics to estimate the likelihood that each may become unstable. The stability model links the stress factor—a measure of how much the degree of class responsibility increases across versions—to the effects of changing requirements.

In 2003, Mahmoud O. Elish et al [11], the authors aim of this study was to find out whether The Chidamber and Kemerer metrics indicate positive outcomes logical stability of class indicators. The findings of the experiment indicated that CBO,WMC, RFC,DIT, and LCOM metrics are inversely connected to the classes' logical stability.Furthermore, it has been discovered that the CBO and RFC Measures are reliable predictors of logicality Layer stability.But no association was discovered. between the logical stability of classes and the NOC measure.The initial phase of developing a collection of logical stability metrics for object-oriented systems is represented by this work.

In 2004, Haohai Ma et al [12], this paper presents a quantitative

approach to evaluate UML meta-models' stability and design quality. The technique uses modified object-oriented metrics to perform objective evaluations of UML meta models. Most OO measurement solutions use a class as the primary measuring target, hence all metrics use meta-classes in the UML meta-models as their foundation. Two different types of extent-of-changes across versions are computed using metric values. In UML evolution, the degree of change is thought to be a reliable predictor of stability.

In 2005, M. Al-shayeb, W. Li [13], have conducted a research to determine whether SDI metric could be utilized to analyze system design evolution in Agile software process and for estimating and re-planning software projects in agile approach similar to XP. They offer an empirical analysis of the SDI measure and class growth in two OO systems that were created via an agile methodology comparable to Extreme Programming (XP).

In 2005, Nikolaos Tsantalis et al. [14], the study suggested using a probabilistic technique to estimate how much an object-oriented design will change. This approach involved the calculation of possibility that every system class will be affected by the changes or additions to the existing functionality. There is a possibility for tracking support maintenance and stability evolution with the help of extracted change probabilities.

In 2006, Hector M., etal, [5], have studied a new SDI form that is referred to as the SDIe, which provides a more precise software stability measure due to the fact that it is based upon highest entropy in the system. They have deployed maintenance data from commercial software project that has been produced under agile process for the purpose of testing new metric. The results that have been obtained after the case study suggest that the new SDIe metric is one of the useful tools for the gauging of system design stability.

In 2007, P. Greenwood etal,[15], have presented quantitative case study that had developed an actual application for the evaluation of several design stability aspects of the implementations that are object- and aspect-oriented. They concentrated on many system modifications that are usually carried out as part of software maintenance procedures.

In 2010, Azar et al [16], Three distinct heuristics—genetic algorithms (GA), tabu search (TS) and simulated annealing (SA), —were provided by the authors as part of their heuristic methodology, to improve models for estimating software quality. Experiments addressing stability of classes in OO system were carried out, which rely on adaptation and re-combination of previously constructed predictive models to new, unseen program. The method is evaluated on stability of classes in an OO software system, additionally, every predictive model is build using rules. and verified method utilizing stability of the software quality characteristic.

In 2010, M. Alshayeb et al [9], for measuring stability of the OO classes, the authors have suggested a class stability metric. Comparing the proposed metric to the current class stability metrics, more aspects are taken into account. Eight criteria related to class properties were shown to have an impact on class stability. The stability of those attributes impacts stability of the entire class because they are the fundamental components of the class structure. Utilizing these characteristics, a metric was proposed to assess the general stability of the class. They verified the newly suggested metric hypothetically. Additionally, two Java systems were used to empirically validate CSM, and results showed a strong negative correlation between CSM and maintenance effort.

In 2011, Dith Nimol et al, [17], the authors suggested a method for utilizing artificial neural networks (ANNs) to measure a class's logical stability. Through selecting a multilayer Perceptron method, an ANN is a technique utilized for estimating the value of class logical stability from the historical data in repository. There are numerous steps in this process: first, class logical stability was measured, and after that multiple regression was used to estimate class logical stability. Following such procedures, they developed a novel method to estimate class logical stability with the use of ANN. They after that compared the estimated CLS using ANN and Multiple Regression to the actual class logical stability. The outcome of the experiment had demonstrated the effectiveness of the ANNs for estimate.

In 2011, Mohammad Alshayeb [18], the authors evaluated the effect of the refactoring on architecture and class stability and suggested classification for the refactoring approaches based upon this impact. Refactoring methods affecting "class-level" stability were shown to have the most effect. It was discovered that refactoring techniques that primarily influence the "method-level" had the least effect on class stability, whereas techniques that primarily change fields and are used within the methods had the least effect.

In 2011, D. Azar et al [19], the authors devised an adaptive method that modifies pre-existing predictive models to fit new data. During the adaption process, they employed an ant colony optimization method. Stability of the classes in OO software systems is used to validate the strategy.

In 2012, Alshayeb [20], the authors offer a methodology for examining the connection between class stability and refactoring effort. Software designers can use this technique to determine whether or not refactoring efforts are beneficial while maintaining the stability of their software design.

In 2013, Alshayeb [3], to assess the connection between class stability and maintainability, the authors have carried out an empirical investigation. The number of the hours that have been spent on maintenance tasks and number of lines of code changed are two ways that the author measures maintainability effort, and they are correlated with class stability. The findings indicate that classes with greater class stability metric (CSM) stability values also have lower perfective maintenance effort values in terms of hours worked. In addition, when calculated for system classes that are cumulatively concatenated throughout all iterations rather than individually, CSM corresponded with all maintenance forms (i.e., corrective, adaptive, and perfective). Additionally, the author discovered that when maintainability is measured by the quantity of lines of code changed, none of the stability metrics exhibit any correlation with it.

In 2014, S. Bouktif et al [21], the authors suggested a novel method for creating stability prediction models that maintain prediction interpretability by combining classifiers. To produce a more precise composite classifier that maintains interpretability, they suggested a specific method for merging Bayesian classifiers. This method is applied in OSS large-scale system context, specifically standard Java API, and is developed with the use of a GA.

In 2014, Alshayeb et al [22], the authors suggested a stability prediction methodology employing an ANN and SVM to construct various prediction models after examining the relationship between a few known design measures and class stability across versions. In this method contrasted these prediction models' accuracy, and the studies show that object-oriented class stability can be accurately predicted using ANN and SVM prediction models.

In 2015, Ahmed and Ebad [23], a new set of ASMs that measure inter-package calls was introduced by the authors. Generally speaking, structural relationships between packages are taken into account by the available ASMs, yet message passing is not. In actuality, a good design is one in which the structural and message carrying linkages between packages are kept to a minimum, allowing evolution's effects to be localized. Maintainability might suffer if changes were applied to numerous packages. ASM was theoretically validated by the authors using a number of well-known mathematical properties. JHotDraw and the abstract window toolkit are two open-source projects that the authors used to empirically evaluate the metric. It was demonstrated that the ASM measurements matched the lines of code changes between the 2 projects' versions.

In 2015, Chhabra and Chawla [24], the four quality attributes—Changeability, Analyzability, Testability, and Stability—can be quantified using the authors' new quality model (SQMMA) that provides ready-to-use mathematical formulas as a weighted summation of a collection of software code metrics. Those qualities also serve as criteria for assessing the software's "maintainability" feature. After that, four different Apache Tomcat versions are used in order to implement the intended model, and the outcomes are shown. Ultimately, the results were verified by trend analysis and extra comparison with bug/change data.Chhabra and Chawla used five metrics for calculating stability (Coupling, Subclasses, EntExt, Hierarchies and Communication) where:

Subclasses: the number of sub-classes.

Coupling: the coupling among objects.

Hierarchies: The depth in inheritance tree.

EntExt: number entry and exit points.

Communication: directly invoked components.

In 2016, Baqais et al [25], the correlation between stability and maintainability was quantified by the authors. A stability metric and a maintainability metric were selected as potential candidates, utilized CSM in stability because of its broad coverage and great precision. MI was selected for maintainability because it is purely based on source code, is easy to understand, and is straightforward. The experimentation demonstrates that there is variability in the correlation behavior between these two measurements, making the conclusion of a direct causal relationship impossible. Nevertheless, a thorough examination and a step-by-step tracking of such experiments show encouraging outcomes. Those findings could help researchers determine the proper way to measure the association between CSM and MI.

In 2018, Goyal et al [26], as an effort to demonstrate a meaningful relationship between stability and design attributes, the authors concentrated on the necessity and significance of assessing stability during the design phase. Multiple linear regressions were established with regard to evaluate the stability of object-oriented design and development. Ultimately, an experimental test was used for validating the developed model.

In 2019, Baig et al [4], depending on ideas of change between intra-package connections, package contents, and inter-package relationships, the authors presented PSM: a new package stability metrics. and provide empirical as well as package contents, support for PSM. An analysis of metrics' mathematical characteristics forms bases of theoretical validations, 5 open-source software applications had been utilized for empirical validations, and a comparison to similar packages for current stability metrics has been provided as well. The researcher had utilized prediction analyses, principal component analyses, and correlation analyses for empirical validations. Based on correlation study, the proposed metrics have negative correlation with the effort of maintenance and provide more precise package stability indication in comparison with current stability measurements. Based on the PCA, the proposed measures served to improve maintenance prediction accuracy through the capturing of additional package stability aspects. It found that existing metrics for the stability of the package, based upon class name changes and lines of code, had shown positive correlation with maintenance efforts and negative correlation with the proposed metric.

In 2019, Alshayeb [27], to determine the connection between stability and code similarity class-level, the authors conducted an empirical investigation. PCA was used to identify the class stability measurements that have strongest correlation with class similarity, and clustering was used to categorize metrics of stability and similarity into various related groups. Furthermore, he developed a prediction model utilizing class stability indicators to forecast class similarity. The CSM has the highest association with the similarity of code, according to data, which indicate as well a substantial relationship between the four metrics of stability under investigation and similarity. Additionally, High stability classes are also probably to have a high degree of similarity, according to the clustering data. Furthermore, it was found that 74.023% of class similarity could be revealed by both CSM and the CII metric. The researchers came to the conclusion that stability measures are a useful gauge of how similar a class is.

In 2022, Skladannyi [28], the authors expanded the quantifiable qualities of the source code utilized in the Delta Maintainability Model (DMM). Comparative analysis of source code changes is a crucial step in proving stability and efficacy of OO software change measurement, as it allows one to measure the

maintainability in processes with a continuous delivery and uninterrupted integration methodological methods. additionally, made modifications to the maintainability. The new characteristic "modification" took the place of the sub characteristics "variability" and "stability." All of the literature reviews listed above are represented in **Table 1**.

Table 1. Summary of Literature Review.

No	Article	Artifact	Metric's Name	Metric's Description
1	W. Li, et al [6]	Code	Loc,SDI, CII and SII Metrics	These metrics give indications of project progress.
2	David Grosser et al[10]	Code	CBR Approach	considered every one of the items as location where the distance function is defined in a multidimensional space with one dimension per metric
3	David Grosser et al [7]	Code	CBR Approach	This method had explored structural similarities between the classes, which have been represented as software metrics, to estimate the probability that they may become unstable.
4	Mahmoud O. Elish et al [11]	Code	Chidamber and Kemerer Metrics	OO design metrics that have been presented by Chidamber and Kemerer were chosen as potential markers of OO designs' logical stability.
5	Haohai Ma et al [12]	Code	Object _Oriented Metrics	adapt object-oriented design metrics and criterions as an approach for assessment of the UML meta-models. It carries out the of stability assessment and quality of design to the UML meta-models
6	Alshayeb, Mohammad, and Wei Li. [13]	Code	System Design Instability (SDI) Metric	used SDI metrics for the estimation of and re-planning of the software projects in conventional process and in XP-like agile process as well
7	Nikolaos Tsantalis et al. [14]	Code UML	Probabilistic approach	Assessment of probability that every one of the classes will be changed in a future generation.
8	Hector M., et al [5]	Code	SDIe Metrics	developed improved original SDI metric measure. It's class-based, OO metric of evolution utilizing the Shannon entropy yielding one score for the stability of the system design
9	Phil Greenwood, et al [15]	Code	-	reports a quantitative case study evolving real-life application for the assessment of a variety of the facets of design stability of object-oriented and AO implementations
10	Azar et al[16]	Code	*Lines percentage of *comments in the source code *Calls (number of statements that include method calls)	heuristic approach to optimize software quality and conducted experiments on stability of classes in OO
11	M. Alshayeb et al[9]	Code	* Class access-level * Class interface name * Inherited class name * Method signature * Method access-level *Method body * Class variable * Class variable access-level	Used these metrics to measure overall class stability
12	Dith Nimol et al [17]	Code UML	Multiple Regression,Artificial Neural Netwok	estimate value of class logical stability from the historical data through selection of multilayer Perceptron approach
13	Mohammad Alshayeb [18]	Code	-	proposed classification for the refactoring approaches based upon refactoring impacts on the class stability
14	D. Azar et al [19]	Code	An Ant Colony Algorithm	Have suggested ant-colony based predictive method for the prediction of syntactic class stability in the OO software systems at early stages of development.
15	Alshayeb [20]	Code	-	The correlation between the effort of refactoring and the stability of classes
16	Alshayeb [3]	Code	CSM	Stability of evaluation with the use of 8 metrics and evaluate correlation between the stability and maintainability of the classes
17	Bouktif et al [21]	Code	-	Building Prediction model for the open-source software systems utilizing Bayesian classifier combination, which had allowed class stability interpretations
18	Alshayeb et al [22]	Code	CSM	proposed a stability prediction model using an ANN and SVM
19	Ebad and Ahmed [23]	Code	ASM	measure the inter-package communication stability
20	Chawla and Chhabra [24]	UML	*Subclasses *Coupling,	Using these metrics to calculate stability

			*Herarchies, *EntExt, *Communication.	
21	Baqais et al[25]	Code	CSM	measured the correlation between stability and maintainability
22	Goyal et al [26]	code	-	development stability evaluation by establishing multiple linear regressions
23	Baig et al [4]	Code	PSM	Metric of stability for 3 dimensions: content, internal and external package connections
24	Alshayeb[27]	Code	CSM CII	the correlation between the similarity and the stability of the code at class level
25	Skladannyi [28]	Code	-	introduced changes to maintainability The sub characteristics "variability" and "stability" have been replaced by new characteristic "modification

A. Analysis of the Literature Review

From the previous studies that were presented in Table (1), it was shown that Stability Metrics for Object-Oriented Design in Software Engineering is calculated at two levels, the code level and the diagrams level, and that the researches published on the code level is much more than the research published on the diagrams level, and there are few researches that have combined the two levels. Also, the research published at the planning level is the most recent.

The oldest metric used to calculate stability is LOC Which is considered a basic and subsidiary metrics of modern standards, Also, the most widely used metrics to compute Stability for Object-Oriented Design in this study is the metric CSM, the rest of the metrics are shown in Figure 1 in terms of the number of uses in the research presented in this study

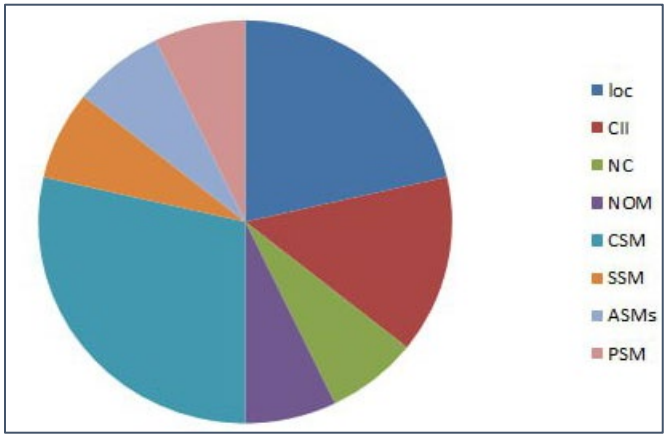

Fig 1. represent how much research used for each metrics.

V. Conclusion

In this study aims to provide a set of measures for assessing the software's object-oriented design stability. Stable software typically requires less work and costs for maintenance. Stakeholders and organizations are becoming more and more concerned with software lifespan. One of the main criteria for reaching it might be stability. With a particular emphasis on OO Design in Software Engineering, the paper has discussed reviewed research on stability as a software property in this publication. And The results reached by this study are as follows:

class stability metrics (structural stability metrics (SSM), architectural stability metrics (ASMs), and package stability are the most significant metrics used to measure stability. The SDI and SII metrics can give indications of project progress (how complete the design and implementation is). This information can in turn be used to adjust a project plan in real time, if the absolute value of the SDI metric stays relatively high and has not shown a downward trend, all the activities that depend on a stable design should be postponed in the project schedule. If the absolute values of the CII and SII metrics remain relatively high, it is better to postpone the formal and systematic testing of the classes and the system. As the preliminary results show that a very straightforward CBR classifier can perform significantly well when the stress estimation is properly fed. And that WMC, DIT, CBO, RFC, and LCOM metrics are negatively correlated with the logical stability of classes. In addition, CBO and RFC metric were found to be good indicators of the logical stability of classes. However, no correlation was found between NOC metric and the logical stability of classes. The SDIe gives a more accurate indication of software stability and maturity since it suffers less from data spikes. CSM was a better indicator of the maintainability measures and showed significant correlation at system level. In Stab and NS, this may mainly be because of the limitations in the way stability, and classes with higher values of stability measured by CSM are associated with lower values of perfective maintenance effort measured by hours. CSM was also found to be correlated with all types of maintenance if measured by hours for the overall cumulative class data rather than for each iteration and the most widely used in literature review. As for the PSM measure, it found a positive relationship between the current package stability metrics, which depend on changes in lines of code and class names, while shows that Artificial Neural Network is effective in the estimation. With this result, the estimation of class logical stability is accepted because the prediction value at level 0.25 is more than 0.75. At the same time, code was the primary focus of most metrics utilized to determine stability instead of UML diagrams. The researchers suggest

investigating new metrics in subsequent work for predicting the stability of other software engineering phases, such as object-oriented design.

Acknowledgement

The research would first like to express our gratitude to Allah, the Creator, for giving me the willpower and perseverance to complete this task. Second, the facilities supplied by the University of Mosul/College of Computer Sciences and Mathematics are greatly appreciated by the authors as they enhanced the caliber of this study.

References

[1] R. Capilla, E. Yumi Nakagawa, U. Zdun and C. Carrillo, "Toward architecture knowledge sustainability: Extending system longevity," IEEE Software 34.2 (2017): 108-111.

[2] S. Maria, R. Bahsoon and P. Lago, "Stability in software engineering: Survey of the state-of-the-art and research directions," IEEE Transactions on Software Engineering, 47.7 (2019): 1468-1510.

[3] M. Alshayeb, "On the relationship of class stability and maintainability," IET software 7.6 (2013): 339-347.

[4] J. Javed Akbar Baig, S. Mahmood, M. Alshayeb and Mahmood Niazi ," Package-Level stability evaluation of object-oriented systems," Information and Software Technology, 116 (2019): 106172.

[5] H. Olague, L. H. Etzkorn, W. Li and G. Cox, ""Assessing design instability in iterative (agile) object-oriented projects," Journal of Software Maintenance and Evolution: Research and Practice 18.4 (2006): 237-266.

[6] W. Li , L. Etzkorn , C. Davis and J. Talburt," An empirical study of object-oriented system evolution," Information and Software Technology 42.6 (2000): 373-381.

[7] G. David, H. A. Sahraoui, and P. Valtchev, "An analogy-based approach for predicting design stability of Java classes," Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717). IEEE, 2004.

[8] D. Rapu, S. Ducasse, T. Girba and R. Marinescu "Using history information to improve design flaws detection," Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.. IEEE, 2004.

[9] M. Alshayeb ,M. Naji ,M. Elish andJ. Al-Ghamdi ," Towards measuring object-oriented class stability," IET software 5.4 (2011): 415-424.

[10] G., David, Houari A. Sahraoui, and Petko Valtchev, "Predicting software stability using case-based reasoning," Proceedings 17th IEEE International Conference on Automated Software Engineering,. IEEE, 2002.

[11] E. Mahmoud and D. Rine, "Investigation of metrics for object-oriented design logical stability," Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.. IEEE, 2003.

[12] Ma, H., Shao, W., Zhang, L., Ma, Z., Jiang, Y , "Applying OO metrics to assess UML meta-models," «UML» 2004—The Unified Modeling Language. Modeling Languages and Applications: 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings 7. Springer Berlin Heidelberg, 2004.

[13] M. Alshayeb and W. Li. ," An empirical study of system design instability metric and design evolution in an agile software process," Journal of Systems and Software 74.3 (2005): 269-274.

[14] T. Nikolaos, A. Chatzigeorgiou, and G. Stephanides ," "Predicting the probability of change in object-oriented systems," IEEE Transactions on Software Engineering 31.7 (2005): 601-614

[15] G. Phil, et al., "On the impact of aspectual decompositions on design stability: An empirical study," ECOOP 2007–Object-Oriented Programming: 21st European Conference, Berlin, Germany,July 30-August 3, 2007. Proceedings 21. Springer Berlin Heidelberg, 2007.

[16] A. Danielle, H. Harmanani, and R. Korkmaz, "Predicting stability of classes in an object-oriented system," Journal of Computational Methods in Sciences and Engineering 10.s1 (2010): S39-S49.

[17] N. Dith, S. Prakancharoen and P. Muenchaisri, "Estimating Software Logical Stability using ANN from Class diagram," Information Technology Journal 7.1 (2011): 58-63.

[18] M. Alshayeb, "The impact of refactoring on class and architecture stability," Journal of Research and Practice in Information Technology 43.4 (2011): 269-284.

[19] A. Danielle, and J. Vybihal, "An ant colony optimization algorithm to improve software quality prediction models: Case of class stability," Information and Software Technology 53.4 (2011): 388-393

[20] M. Alshayeb ,"Investigating the relationship between refactoring activities and class stability," 21st International Conference on Software Engineering and Data Engineering, SEDE 2012.

[21] B. Salah, H. Sahraoui and F.Ahmed , "Predicting stability of open-source software systems using combination of Bayesian classifiers," ACM Transactions on Management Information Systems (TMIS) 5.1 (2014): 1-26.

[22] M. Alshayeb, E. Yagoub, and A. Moataz , "Object-oriented class stability prediction: a comparison between artificial neural network and support vector machine," Arabian Journal for Science and Engineering 39 (2014): 7865-7876.

[23] E. Shouki, and M. Ahmed, "Measuring stability of object-oriented software architectures," IET Software 9.3 (2015): 76-82.

[24] M. Chawla, and I. Chhabra,"Sqmma: Software quality model for maintainability analysis," Proceedings of the 8th Annual ACM India Conference. 2015.

[25] B. Abdulrahman, M. Amro, and M. Alshayeb, "Analysis of the correlation between class stability and maintainability," 2016 7th International Conference on Computer Science and Information Technology (CSIT). IEEE, 2016.

[26] G. Nidhi, and R. Srivastava, "Stability evaluation model for object oriented software," International Journal of Advanced Research in Computer Science 9.2 (2018).

[27] M. Alshayeb, "An Empirical Study on Using Class Stability as an Indicator of Class Similarity," Arabian Journal for Science and Engineering 44.11 (2019): 9413-9426.

[28] P. Skladannyi , O. Nehodenko , S. Shevchenko , O. Zolotukhina and V. Nehodenko, "Modified Delta Maintainability Model of Object-Oriented Software," Cybersecurity Providing in Information and Telecommunication Systems 2022 3288.1 (2022): 117-124.