Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.042711

ARTICLE

Software Coupling and Cohesion Model for Measuring the Quality of Software Components

Zakarya Abdullah Alzamil*

Software Engineering Department, CCIS King Saud University, Riyadh, 11495, Saudi Arabia

*Corresponding Author: Zakarya Abdullah Alzamil. Email: zakarya@ksu.edu.sa

Received: 09 June 2023 Accepted: 24 October 2023 Published: 26 December 2023

ABSTRACT

Measuring software quality requires software engineers to understand the system's quality attributes and their measurements. The quality attribute is a qualitative property; however, the quantitative feature is needed for software measurement, which is not considered during the development of most software systems. Many research studies have investigated different approaches for measuring software quality, but with no practical approaches to quantify and measure quality attributes. This paper proposes a software quality measurement model, based on a software interconnection model, to measure the quality of software components and the overall quality of the software system. Unlike most of the existing approaches, the proposed approach can be applied at the early stages of software development, to different architectural design models, and at different levels of system decomposition. This article introduces a software measurement model that uses a heuristic normalization of the software's internal quality attributes, i.e., coupling and cohesion, for software quality measurement. In this model, the quality of a software component is measured based on its internal strength and the coupling it exhibits with other component(s). The proposed model has been experimented with nine software engineering teams that have agreed to participate in the experiment during the development of their different software systems. The experiments have shown that coupling reduces the internal strength of the coupled components by the amount of coupling they exhibit, which degrades their quality and the overall quality of the software system. The introduced model can help in understanding the quality of software design. In addition, it identifies the locations in software design that exhibit unnecessary couplings that degrade the quality of the software systems, which can be eliminated.

KEYWORDS

Software coupling measurement; software cohesion measurement; quality attributes measurement; software quality measurement; software quality modeling

1 Introduction

Measuring software quality is one of the challenges in software development. It requires software engineers to understand software quality attributes that describe internal or external system features or properties, and how to measure them. However, measuring software quality attributes is a major challenge for software engineers as they are qualitative and measurement requires quantitative metrics. It has been found that quality attributes are primarily addressed in a general form without any

empirical model or specific quantifiable approach [1] that may help in integrating quality attributes in the development process.

Stevens et al. [2] introduced software coupling and cohesion as internal attributes of a software component. Since then, they have been considered as main software metrics along with complexity, inheritance, and size [3], and are being used to assess different quality attributes of software systems such as reusability and maintainability [4].

In addition, the proposed approach is not limited to object-oriented designed systems; it can be applied to different architectural design models. Furthermore, the proposed approach can be applied at different levels of system decomposition, including the subsystem level, component level, and class or module level.

In this paper, the author investigates software coupling and cohesion to propose a software model that uses a heuristic normalization of the software's internal attributes for measuring the quality of software components as well as the overall design quality of the software system. The proposed model aims to provide software engineers with practical guidelines to understand the quality of the software under development and identify places in the design that may influence the quality of the software components as well as the overall quality of the software system. Unlike most of the existing approaches, the proposed approach can be applied at the early stages of software development to different architectural design models, and at different levels of system decomposition. The paper is organized as follows; Section 2 presents the related work, Section 3 describes the proposed approach, Section 4 illustrates the experiments, and finally, Section 5 presents the conclusions.

2 Related Works

There have been many studies that investigate quality attribute measurement and evaluation of software systems at different stages of software development and for different purposes. Coupling and cohesion are among the most explored software internal quality attributes that have been investigated for different purposes such as fault prediction [5], energy consumption [6], program restructuring [7], and maintainability and testability [8].

Many approaches have been proposed to evaluate the quality of software systems using different coupling and cohesion metrics. These different approaches have been applied to software systems at the design level and source code level, and have used different coupling and cohesion metrics, such as Martin's metrics [8], which aim to measure direct coupling among software packages, i.e., classes, coupling between objects (CBO) which aims to measure coupling among object-oriented classes., and Lack of cohesion in methods (LCOM) [9] which has been extended in a series of LCOM metrics [10], which aim to measure similarity degree of methods based on the commonly used instance variables between methods to measure the inter-relatedness between class methods. In the following paragraphs, those approaches are briefly presented.

Changeability measures have been proposed in [11] using coupling among object-oriented classes to estimate the impact of the changeability of the classes by extracting associations among them. These measures identify the ripple effect to measure the change propagation, its impact on the classes, and the cost of class change. An approach to evaluate software reliability concerning correlated component failures was proposed in [12] using internal coupling among software components based on a multivariate Bernoulli distribution, in which components' dependencies are collected by static analysis and then applied to a real-time software application to illustrate the effectiveness of the software reliability evaluation. A dynamic software coupling metric has been introduced in [13] as

a weighted measurement that uses the method calls in terms of the number of interactions between classes in object-oriented software. In this dynamic approach, data is collected at runtime, and the number of method calls is compared to the static coupling dependency to compute the coupling degree statically and dynamically, in which the dynamic coupling metric complements the static ones. Aggregated coupling and cohesion metrics have been used in [14] to predict the quality of services properties of web services. This approach uses source code metrics and machine-learning techniques to automate the prediction of QoS properties and improve its efficiency.

Reference [15] used one class classification technique to evaluate the maintainability of a software system at the package level. The proposed methodology evaluates software maintainability based on four static analysis source code metrics, namely: complexity, cohesion, coupling, and inheritance. The results have shown that static analysis metrics enable the effective identification of non-maintainable components at an early stage, which represent around 50% of the software package lifecycle. Kaur et al. [4] have proposed a fuzzy model that uses internal package-level metrics, i.e., coupling and cohesion, to assess external quality attributes such as reusability, maintainability, and understandability of aspect-oriented systems at the package level. A package cohesion measure was proposed in [16] for assessing the reusability of aspect-oriented systems at the package level, which may help software developers develop high-quality software. This proposed metric is based on formal definitions and relations among the elements of the package.

A cohesion metric for classes in object-oriented software was proposed in [17] to reduce the maintenance effort of classes. The authors proposed a low-level attribute-method usage class cohesion metric that is based on the instance variable used by class methods at a source code level. This proposed cohesion metric is based on three types of relationships; received, e.g., when the instance variable is received as a parameter; manipulated, e.g., when the instance variable is used in computation within the method body; and returned, e.g., when an attribute is returned by a method. Low-level similarity-based class cohesion and class cohesion metrics are used in [18] to measure the quality of program code that uses code snippets. In this technique, the quality of the recipient class is measured before the addition of the snippet, immediately after the addition of the snippet, and at later stages of software development. The aim was to determine the impact of the addition of snippets on the program's quality, and the authors found that in 70% of the cases, the copied snippet affected cohesion, which may lead to quality deterioration.

Refactoring and re-refactoring operations on code structure were investigated in [19] to improve the code structural quality by understanding the effect on internal quality attributes and to check whether re-refactoring is more effective in improving attributes when compared to single operations. In this approach, descriptive analysis and statistical tests are used to deeply understand the effect of both refactoring and re-refactoring on internal quality attributes. This analysis aimed to understand when and how re-refactoring affects code metrics that quantify cohesion, complexity, coupling, inheritance, and size. This study has revealed that most operations improve attributes presumably associated with the refactoring type applied; the other operations keep those attributes unaffected. A non-dominated sorting genetic algorithm has been used in [7] to automate the restructuring process of object-oriented software packages. The proposed approach computes coupling and cohesion for package restructuring using different types of structural, lexical, and change history class information. The results of applying the proposed approach to restructure five object-oriented software applications indicate that such an approach may improve the design quality of the software systems under development.

The relationships between package size and internal maintainability attribute metrics such as coupling, cohesion, and complexity were analyzed in [20]. The study aims to identify the maintainability issues and metrics useful for identifying refactoring opportunities for large packages. The proposed

approach has experimented with 111 open-source Java projects to collect package-level metrics, in which higher maintainability issues in large packages are observed as indicated by the used metrics. In addition, the results have shown strong relationships between cohesion and complexity with package size, which may be used to identify large package refactoring opportunities. The influence of process and developer-related factors on design decay has been investigated in [21] by measuring internal quality attributes such as coupling, cohesion, complexity, inheritance, and size. In this study, seven software systems with an average of 45 K commits in more than six years of project history have been analyzed to identify the effects of interacting factors that cause modules to decay and observe decay patterns in these modules. The results have shown that the developers-related factors, such as first-time contributors, and process-related factors such as the size of a change, have no negative effects on the changed classes. In contrast, when both of these factors interact, a negative effect on the code that leads to decay is observed. Reference [22] proposed a dynamic approach for identifying software reusability using coupling detection among software components at the design level. It uses dynamic notions of sequence diagrams to understand the software system's behaviors. Data and control dependencies have been used to detect the dependence among different software components. The dependencies among software components are defined when one component influences the output of another component. A client-based class cohesion metric was proposed in [23] to measure class cohesion based on how its clients use its public methods. This proposed metric can be used at the design phase with the information from high-level design, in which information can be collected from the communication diagram.

A prediction model has been proposed in [24] to understand whether the existence of faults in the code indicates a quality problem in software design. This study investigates the impact of bug fixing on software internal quality attributes such as complexity, coupling, and cohesion by experimenting with the proposed model in thirteen different projects using five different classifiers. The results have shown that the prediction model using least-squares versions of support-vector machines (SVM) performs better than other techniques, which shows that more than 80% of the cases with bugs in classes that have at least one critical attribute. Reference [25] proposed a multi-objective hyper-heuristic method to improve the software maintenance process by improving the software design with better modularization. The proposed clustering model aims to minimize coupling, maximize cohesion, and enhance modularity. The results of the experiments have shown that the resulting modularized software is more optimized with lower coupling and higher cohesion. In addition, the resulting software is more robust, easier to maintain, and with better modularity. An information-theoretic software re-modularization approach has been proposed in [26] to re-modularize object-oriented software to improve its design quality. The proposed model uses entropy-based similarity measures as objective functions to optimize the internal software structure, in addition, other metrics are used as objective functions such as inter-module class change coupling. The proposed approach has experimented with seven object-oriented software systems, which shows that the approach is a good alternative for software re-modularization to improve the quality of software systems.

Reference [27] proposed an approach that aims to select the most suitable metric, e.g., complexity, cohesion, coupling, and inheritance, for detecting specific design defects. In this approach, a fuzzy decision-making trial and evaluation laboratory method has been applied to identify the detection rules. The proposed approach has been experimented with four open-source projects and has shown the efficiency of the fuzzy method in identifying the best rules to identify design defects. A usage pattern-based cohesion metric was proposed in [28] to measure cohesion at the module level of object-oriented software. This approach aims to evaluate the quality and modularity of a software system at the design level and improve overall cohesion. The proposed metric uses the frequent usage patterns

extracted from the interactions of module functions to measure the cohesiveness of the module, which is used to perform clustering of modules to maximize cohesion and minimize coupling among modules. The experiment on two Java programs has shown an improvement in the cohesiveness of the software system.

A summary of the above-mentioned related works, their used metrics, and the aim of each approach is presented in Appendix A. Although the existing measurement approaches are valuable and helpful for many software systems, they are limited to certain quality attributes, certain programming languages, or specific domains, and are qualitative, in which they depend on domain experts to predict or provide judgment to analyze the quality attributes. In addition, most of the well-known coupling and cohesion metrics, such as direct coupling between classes [11], CBO, and LCOM [9], are based on instance variables used by class methods and/or the similarity degree of methods. However, this requires very detailed design, such as method definition and instance variable usage, and may require the availability of the source code to analyze classes, methods, or parameters at the early stages of software development. However, detailed design and/or source code are not available at the early stages of software development or due to reused or outsourced components or services. Moreover, software quality may not be measured or predicted by these approaches until the late stages of software development, when software changes are costly. Another limitation of the current approaches is that most of the proposed approaches are limited to object-oriented software systems. However, most modern software systems are designed based on different architectural designs, such as service-oriented, interaction-oriented, and component-based.

In this research, the author investigates software coupling and cohesion to propose a software quality measurement model that uses a heuristic normalization of the software's internal quality attributes for measuring software components as well as the overall design quality of the software system. Unlike most of the existing approaches, the proposed approach can be applied at the early stages of software development, in which very detailed design or source code is not required. In addition, the proposed approach is not limited to object-oriented designed systems; it can be applied to different architectural design models. Furthermore, the proposed approach can be applied at different levels of system decomposition, including the subsystem level, component level, and class or module level.

3 Proposed Approach

This section describes the proposed approach for measuring the quality of software components using coupling and cohesion, in which a formal model is introduced. Among the basics of software analysis and design is divide and conquer, in which the domain problem is decomposed into smaller problems or sub-problems, and every sub-problem may focus on certain concerns or aspects of the domain problem that may be designed as a subsystem or component. Regardless of the system's architectural design and structure, system design consists of a set of components, connectors, and configurations, to which the quality attributes are related. To measure the system quality attributes, one must decompose the system into its main building blocks, i.e., main subsystems/components; understand their properties in terms of coupling, cohesion, and configuration; and identify the desired quality attributes. In this proposed approach, the author uses the notions of software coupling and cohesion as measures of the overall design quality. Software cohesion measures the internal strength of software components, and software coupling measures the connections and/or dependencies among these components. In addition, the author introduces a formal model to understand the configurations among software components for better software quality. The following paragraphs describe coupling and cohesion measures:

Software coupling is a measure of the interdependence degree between software components. Software components may exhibit different levels of interdependence, which were identified by Myers [29] from the worst to the best coupling as follows; content coupling, common coupling, external coupling, control coupling, stamp coupling, and data coupling. Data coupling occurs when simple data, i.e., a simple argument, is passed between the interconnecting components. Stamp coupling occurs when a data portion of a data structure is passed between components. Control coupling occurs when a control, such as a flag, is passed between components. External coupling occurs when the two components are tied to an environment or medium that is external to the system, such as communicating via an I/O device or file. Common coupling occurs when the interacting components reference global data. Content coupling occurs when one component uses or changes the data or control information maintained within the boundary of another component. Although coupling among software components is not desirable, in most cases it is not avoidable; therefore, the objective is to minimize it to the lowest level. Software coupling can be computed using different techniques, such as data and control flow analysis [22].

The cohesion of a software component is a measure of its relative functional strength. A cohesive component is desired, and software engineers should avoid low-level cohesion when designing software components. Cohesion has been categorized by Stevens et al. [2] from lowest to highest into seven types, namely; coincidental, logical, temporal, procedural, communication, sequential, and functional cohesion. Coincidental cohesion occurs when a component performs a set of tasks that are not related or are loosely related. Logical cohesion occurs when performed tasks, i.e., processing elements within the component, are related logically. Temporal cohesion occurs when tasks must be executed within the same span of time. Procedural cohesion occurs when the tasks must be executed in a specific order. Communication cohesion occurs when the tasks operate on one area of the data structure. Sequential cohesion occurs when the output of one processing element is used as input for another element within the component. Functional cohesion occurs when the component performs a single and well-defined task.

A software component may exhibit more than one type of cohesion, but the overall level of cohesion would be the worst type that it exhibits. The cohesion of a software component is measured by computing the relatedness of different elements or methods within the component through their interactions. Therefore, software cohesion can be computed by identifying the components' interfaces, in which the interactions are defined among the component's methods/signatures to determine the cohesion type based on the aforementioned classification.

The proposed model in this research has adopted the software interconnection model introduced by Perry [30], in which a basic input/output predicate was defined. The interconnection model defines input predicates as assumptions or preconditions that must be satisfied if a sequence of code is to execute successfully. The output predicates are defined as the results or post-conditions that are guaranteed to be true if the input predicates are satisfied. These input/output predicates, i.e., preconditions and post-conditions, represent the behavior of either the system or a system component. In the proposed model, the quality of a component is measured based on its internal strength and its coupling with other components, in which the component's behavior and/or expected results depend on the output/post-conditions of other components that the concerned component is coupled with. The coupling between different components is defined in terms of a graph connecting components by edges, where the assumptions/preconditions of one component are dependent on the output/post-conditions of another component that it is connected with, which consequently influences its output/post-conditions, and the behavior and quality attributes of the component are influenced as well.

Formally, coupling between components of a software system S is represented by a directed graph as an ordered pair $S = <C, D>$ where C is a set of n nodes representing the system's components, $C = <c_1, c_2, \ldots, c_n>$, and D is the set of ordered pairs of nodes representing coupling between these components, such that $d_{ij}(c_i, c_j)$ describes the dependency of component c_i on component c_j iff $POSTCOND(c_j)$ influences $PRECOND(c_i)$, where $PRECOND(c_i)$ is the assumption/precondition of component c_i and $POSTCOND(c_j)$ is the output/post-condition of component c_j. Consequently, if $d_{ij}(c_i, c_j)$ proves true, then $d_{ij}(POSTCOND(c_i), POSTCOND(c_j))$ is true as well, in which the coupling of component c_i with component c_j influences its behavior and consequently influences the quality attributes of the software system S.

Similarly, the cohesion of a software component SC is represented by a directed graph as an ordered pair $SC = <K, R>$ where K is a set of m functions $K = <k_1, k_2, \ldots, k_m>$ that SC performs and R is the set of interdependencies between functions of SC, such that $r_{ij}(k_i, k_j)$ describes the functions' relatedness in terms of dependency of function k_i on function k_j iff $POSTCOND(k_j)$ influences $PRECOND(k_i)$, where $PRECOND(k_i)$ is the assumption/precondition of function k_i and $POSTCOND(k_j)$ is the output/post-condition of function k_j. Consequently, if $r_{ij}(k_i, k_j)$ proves true, then $r_{ij}(POSTCOND(k_i), POSTCOND(k_j))$ is true as well, in which the cohesiveness of the software component SC influences its behavior and consequently influences its quality attributes and the software system S as well. For given $d_{ij}(c_i, c_j)$ and $r_{ij}(k_i, k_j)$, there is $dr_{ij}(POSTCOND(k_i), POSTCOND(c_j))$ that describes the influence of the coupling of component c_i with component c_j on the cohesion of component c_i by influencing function k_i of component c_i, iff $d_{ij}(POSTCOND(c_i), POSTCOND(c_j))$ and $r_{ij}(POSTCOND(k_i), POSTCOND(k_j))$ prove true. As a result, the internal strength of component c_i is reduced by the amount of coupling it exhibits with the c_j component. Fig. 1 depicts a high-level diagram of the framework of the proposed model. It consists of two parts, the parsing and graph generation part and the analysis and computation part. The parsing and graph generation part reads the software artifacts such as use cases, class diagrams, source code, etc., to build the graph, and the analysis and computation part analyzes the graph to compute coupling and cohesion to provide more insights into the system's internal quality for quality tradeoff and improvement.

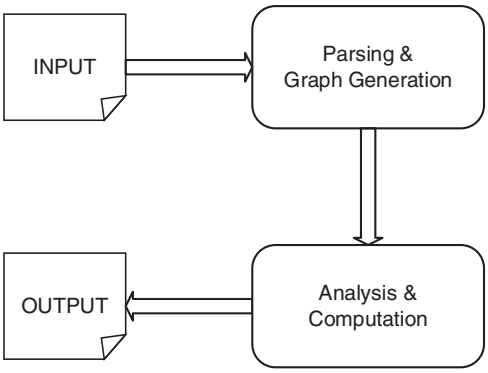

Figure 1: Proposed model framework

In this research study and for measuring coupling and cohesion, the author heuristically quantifies the qualitative description of the different types of coupling and cohesion to a normalized numerical spectrum format range from zero to one, i.e., a range of 0–1, in which zero represents the lowest and one represents the highest. This range distributes the weight evenly among the levels/types of coupling and cohesion. Tables 1 and 2 show the heuristic weighting of different types of coupling and cohesion,

respectively. As stated earlier, the best case is to have components with the highest cohesion and lowest coupling, in which the highest cohesion is represented as one and no coupling is represented as 0.01 because it is not avoidable.

Table 1: Component weighted coupling

Coupling	Interdependence	Weight of $d_{ij}(c_i, c_j)$
No coupling	No interdependence	0.01
Data coupling	Simple data is passed between components	0.15
Stamp coupling	A portion of the data structure is passed between components	0.30
Control coupling	Control flag is passed between components	0.45
External coupling	Two components are tied to an external environment or medium, i.e., I/O device or file	0.60
Common coupling	Components reference a global data	0.75
Content coupling	One component alters the data/control information within the boundary of another component	1.0

Table 2: Component weighted cohesion

Cohesion	Interdependence	Weight of $r_{ij}(k_i, k_j)$
Coincidental cohesion	No dependence/loose dependence	0.01
Logical cohesion	Control dependence	0.15
Temporal cohesion	Time dependence/tasks executed within the same span of time	0.30
Procedural cohesion	Execution order dependence/executed in a specific order	0.45
Communication cohesion	Shared data structure	0.60
Sequential cohesion	Data dependence	0.75
Functional cohesion	Single well-defined task	1.0

Coupling describes the interdependencies among n components, in which a component c_i may exhibit different types of coupling with different components. All dependencies that component c_i exhibits with n components produce the aggregated coupling of the component $WCP\,(c_i)$ based on the heuristic weights in Table 1, as follows:

$$WCP\,(c_i) = \frac{\sum_{j=1}^{n} d_{ij}(c_i, c_j)}{n} \tag{1}$$

Cohesion represents the internal strength of a component, which describes the relatedness of the functions performed by the component in the form of their interdependencies. A component may

exhibit different types of cohesion at the same time, i.e., it may perform different functions that are related logically and share the same data structure, in which it exhibits logical and communication cohesion simultaneously, which should be considered when measuring its cohesiveness. Aggregated cohesion of the component $WCH\ (c_i)$ with m functions can be computed based on the heuristic weights in Table 2 as follows:

$$WCH\ (c_i) = \frac{\sum_{t=1}^{m} CH(f_t)}{m} \tag{2}$$

where $CH(f_t)$ describes the relatedness of function f_t with other functions within component c_i, and is computed as follows:

$$CH(f_t) = \frac{\sum_{t=1}^{m} r_{tj}(k_t, k_j)}{m} \tag{3}$$

The overall cohesion of a software system S with n components $ACH(S)$ can be computed as follows:

$$ACH\ (S) = \frac{\sum_{i=1}^{n} WCH\ (c_i)}{n} \tag{4}$$

Although high cohesion and low coupling are desired for good design, they are not necessarily correlated, either positively or negatively, i.e., when one is low or high, it does not necessarily cause the other to be high or low, respectively, as it depends on the design quality, the software engineers' experience, and the followed development standards. However, cohesion measures the component's internal strength, representing its internal quality; such strength is influenced by its coupling with other component(s) regardless of their level of cohesion. In other words, the best case occurs when a component has no coupling or minimal coupling with different component(s) that are highly cohesive to avoid degrading its cohesion. However, in most cases, coupling reduces the internal strength of the coupled component(s), even if they are highly cohesive. The author proposes a heuristic approach to measure the design quality of a software component c_i in the form of the influence of the aggregated coupling $WCP\ (c_i)$, Eq. (1), on the aggregated cohesion $WCH\ (c_i)$, Eq. (2). Therefore, the influence of coupling on the cohesion of a component c_i is computed by reducing its aggregated cohesion $WCH\ (c_i)$ by a percentage equal to the aggregated coupling $WCP\ (c_i)$ it exhibits to compute its overall quality. The design quality of component c_i denoted as $Q(c_i)$ is computed as follows:

$$Q(c_i) = WCH\ (c_i) - WCH(c_i) * WCP\ (c_i) \tag{5}$$

The overall quality of a software system S with a set of n software components $C = < c_1, c_2, \ldots, c_n >$ can be computed as follows:

$$Q(S) = \frac{\sum_{i=1}^{n} Q(c_i)}{n} \tag{6}$$

The author has experimented with the proposed approach in several real-world software projects to investigate its applicability. The following section describes the experimental study.

4 Experimental Study

This section describes the experimental study that has been performed to evaluate the proposed approach. In this experimental study, several software systems from different domains and managed by different software engineering teams have been investigated using the proposed measurements in

this approach to examine the quality of these software systems. The author has experimented with the proposed approach of nine software systems to be developed by undergraduate and postgraduate students of the Software Engineering Department at King Saud University for their graduation projects. Although the selected software systems are training and/or research software projects, unlike commercial software and development teams, they are available, accessible, and willing to participate during development.

The development teams of software engineers have agreed to participate in this experiment, in which online meetings are scheduled and conducted for each team individually to introduce and describe the proposed model to them. The participants have been provided with a guideline to help them identify and measure coupling and cohesion within their software components. The teams have examined their software systems to measure their quality using the proposed measurement in this approach. However, experimental data are not open-source and unavailable to the public due to ownership and copyright issues in graduation projects. Table 3 describes the software systems used in this experiment.

Table 3: Software systems used in the experiments

No.	Software system	Description
1	CBIR system	Content-based image retrieval system that incorporates user relevance feedback to enhance the retrieved results.
2	Fatwa system	Question-answering Islamic fatwa system using machine learning and information retrieval techniques to retrieve the most relevant and accurate answers to Islamic decrees (Fatwas) for a fatwa seeker's question.
3	Habits tracking system	A software system that aims to track and analyze personal health activities to help people to increase their productivity.
4	Puzzle-solving system	Puzzle app that aims to improve people's logical thinking skills by solving puzzles using an augmented reality approach.
5	Tadreeb system	Online interns portal provides trainees with training/coop opportunities and helps companies/firms find suitable trainees for on-the-job training.
6	Restaurant rating system	A web-based system that aims to rate restaurants/cafes for individuals to choose for eating or drinking. The system uses different techniques to collect and analyze data to rate the restaurants based on specific rating criteria.
7	Video games rating system	A software system that automatically classifies and rates video games based on game reviews using a machine-learning model that employs Islamic classification standards.
8	Information security awareness system	A software system that examines published posts on social media platforms, e.g., Twitter and Facebook, to warn the users of contents that contain sensitive data such as personal information and banking data.
9	Patient companion system	A software system that provides several religious and social services to hospital patients.

The following paragraphs illustrate the coupling and cohesion measurement of these software systems using the equations described earlier to compute the overall quality of the software system. For simplicity's sake, these measurements are presented in the form of tables.

The CBIR system consists of four major components as listed in Table 4. The development team has identified the components with a sole single function, namely, *Image Processing* and *Feature Extraction* components, in which their weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, their aggregated cohesions $WCH\ (c_i)$, *Eq. (2)*, are one as well. The remaining components, i.e., S*imilarity Comparison* and *Relevance Feedback*, each have several functions, so their aggregated cohesion $WCH\ (c_i)$ is measured by the development team using the aforementioned equations based on the weights shown in Table 2. The computation of the aggregated cohesion $WCH\ (c_i)$ of the *Similarity Comparison* component is presented in Table 5, in which the aggregated cohesion $WCH\ (c_i)$ of the remaining components is computed similarly.

Table 4: Weighted coupling $d_{ij}(c_i, c_j)$, aggregated coupling $WCP\ (c_i)$, aggregated cohesion $WCH\ (c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of CBIR system

$\boldsymbol{d_{ij}\ (c_i, c_j)}$	*Image processing*	*Feature extraction*	*Similarity comparison*	*Relevance feedback*	$\boldsymbol{WCP\ (c_i)}$	$\boldsymbol{WCH\ (c_i)}$	$Q(c_i)$
Image processing	•	0.45	0.45	0.45	**0.45**	**1.0**	**55%**
Feature extraction	0.75	•	0.15	0.15	**0.35**	**1.0**	**65%**
Similarity comparison	0.75	0.30	•	0.15	**0.40**	**0.59**	**35%**
Relevance feedback	0.75	0.15	0.15	•	**0.35**	**1.0**	**65%**
			ACH(S)*, *Eq. (4)			**90%**	
			Q(S)*, *Eq. (6)				**55%**

Table 5: Weighted cohesion $r_{ij}(k_i, k_j)$ among the functions of the similarity comparison component and its aggregated cohesion $WCH\ (c_i)$

$\boldsymbol{r_{ij}(k_i, k_j)}$	*Color comparison*	*Texture comparison*	*Edge comparison*	*Combination comparison*	$CH(\boldsymbol{f_t})$, *Eq. (3)*
Color comparison	•	0.60	0.60	0.15	**0.45**
Texture comparison	0.60	•	0.60	0.15	**0.45**
Edge comparison	0.60	0.60	•	0.15	**0.45**
Combination comparison	1.0	1.0	1.0	•	**1.0**
		WCH* (c_i), *Eq. (2)			**0.59**

The development team has measured the weighted coupling $d_{ij}(c_i, c_j)$ among the system's components based on the heuristic weighted coupling shown in Table 1 to compute the aggregated coupling of the component $WCP\ (c_i)$. In addition, the development team has computed the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$ as presented in Table 4. As can be noticed, cohesion describes the internal strength

of the software system, in which it exhibits high internal strength, i.e., 90%. In addition, it shows how the amount of coupling it displays can degrade its quality, in which the quality $Q(S)$ is reduced significantly, from 90% to 55%, which indicates the overall quality of the system, which is reduced by 39%. Software engineers may use this information to localize the places in the software design that exhibit unnecessary component couplings and improve the software design for better software quality.

The Fatwa system consists of four major components as listed in Table 6. The development team has identified the components with a sole single function, namely, *Dataset Extraction*, *Fatwas Preprocessing*, and *Question Classification* components, in which their weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, their aggregated cohesions $WCH\,(c_i)$ are one as well. The development team has measured the aggregated cohesion $WCH\,(c_i)$ and aggregated coupling $WCP\,(c_i)$ of the remaining component, i.e., *Information Retrieval* which has several functions, similarly as illustrated earlier. In addition, the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$ were computed as well and presented in Table 6. It can be noticed that cohesion describes the internal strength of the software system and how the coupling it exhibits influences its quality, in which the $Q(S)$ is reduced from 85% to 79%, which indicates the overall quality of the system. Although the reduction in quality is 7%, this can help the software engineers identify the locations of couplings among software components that cause such degradation and decide how to trade off the necessity of such couplings for the amount of quality improvement that can be gained.

Table 6: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP\,(c_i)$, aggregated cohesion $WCH\,(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Fatwa system

$d_{ij}\,(c_i, c_j)$	*Dataset extraction*	*Fatwas preprocessing*	*Question classification*	*Information retrieval*	$WCP\,(c_i)$	$WCH\,(c_i)$	$Q(c_i)$
Dataset extraction	•	0.15	0.15	0.15	**0.15**	**1.0**	**85%**
Fatwas preprocessing	0.01	•	0.01	0.01	**0.01**	**1.0**	**99%**
Question classification	0.01	0.01	•	0.15	**0.07**	**1.0**	**93%**
Information retrieval	0.01	0.01	0.01	•	**0.01**	**0.38**	**38%**
ACH(S), Eq. (4)						**85%**	
Q(S), Eq. (6)							**79%**

The Habits tracking system consists of six major components as listed in Table 7. The development team has identified the components with a sole single function, namely, *Recommendation* and *Report Generator* components, in which their weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, their aggregated cohesions $WCH\,(c_i)$ are one as well. The development team has measured the aggregated cohesion $WCH\,(c_i)$ and the aggregated coupling $WCP\,(c_i)$ of the remaining components that have several functions, i.e., *Settings*, *Activity Tracker*, *Analysis*, and *Authentication*. Table 7 shows the design quality of the component $Q(c_i)$, the overall

cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. As can be seen, cohesion describes the internal strength of the software system and how the amount of coupling it exhibits reduces its quality by 24%, in which the overall quality of the system $Q(S)$ is reduced from 72% to 55%. Such quality degradation can be tracked in the software design by localizing the couplings that cause such quality degradation to avoid or minimize them.

Table 7: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Habits tracking system

$d_{ij}(c_i, c_j)$	*Settings*	*Activity tracker*	*Analysis*	*Recommendation*	*Authentication*	*Report generator*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
Settings	•	0.15	0.15	0.30	0.30	0.01	**0.18**	**0.21**	**17%**
Activity tracker	0.30	•	0.01	0.60	0.15	0.60	**0.33**	**1.0**	**67%**
Analysis	0.01	1.0	•	0.45	0.01	0.30	**0.35**	**0.10**	**7%**
Recommendation	0.01	0.01	0.75	•	0.01	0.30	**0.22**	**1.0**	**78%**
Authentication	0.30	0.01	0.01	0.01	•	0.15	**0.10**	**1.0**	**90%**
Report generator	0.01	0.01	1.0	0.30	0.01	•	**0.27**	**1.0**	**73%**
ACH(S), Eq. (4)								**72%**	
Q(S), Eq. (6)									**55%**

Puzzle-solving system consists of five major components as listed in Table 8. The development team has identified the components with a sole single function, namely, the *Player* component, in which its weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, its aggregated cohesions $WCH(c_i)$ is one as well. The development team has measured the aggregated cohesion $WCH(c_i)$ and the aggregated coupling $WCP(c_i)$ of the remaining components that have several functions, i.e., *Main Menu*, *AR View*, *Puzzle*, and *Scoreboard*. Table 8 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. As can be noticed, the quality of this system is low, as described by its cohesion. The low quality of the software system is worsened by the amount of coupling it exhibits, in which the $Q(S)$ is reduced from 42% to 31%, which indicates the overall quality of the system. Although this software system has low internal strength, which indicates very low quality, it shows how the coupling degraded its quality by 26%. This measurement provides software engineers with more insight into their software system, in which they can inspect their software design to identify the coupling among components that worsen its quality and minimize it to increase the quality to a better level.

Table 8: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Puzzle-solving system

$d_{ij}(c_i, c_j)$	*Main menu*	*AR view*	*Puzzle*	*Player*	*Scoreboard*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
Main menu	•	0.15	0.15	0.01	0.15	**0.12**	**0.05**	**4%**
AR view	0.01	•	0.60	0.60	0.01	**0.31**	**0.48**	**33%**
Puzzle	0.01	0.45	•	0.15	0.15	**0.19**	**0.38**	**31%**
Player	0.30	0.60	0.15	•	0.30	**0.34**	**1.0**	**66%**
Scoreboard	0.15	0.01	0.01	0.30	•	**0.12**	**0.21**	**19%**

(Continued)

Table 8 (continued)

$d_{ij}(c_i, c_j)$	*Main menu*	*AR view*	*Puzzle*	*Player*	*Scoreboard*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
			ACH(S), Eq. (4)				**42%**	
			Q(S), Eq. (6)					**31%**

The Tadreeb system consists of five major components as listed in Table 9. The development team has identified the components with a sole single function, namely, the *Post View* component, in which its weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, its aggregated cohesions $WCH(c_i)$ is one as well. The development team has measured the aggregated cohesion $WCH(c_i)$ and the aggregated coupling $WCP(c_i)$ of the remaining components that have several functions, i.e., *Post Controller*, *Post Model*, *HRE Model*, and *Trainee Model*. Table 9 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. The quality of this software system is very low, as described by its internal strength, the cohesion, however, such low quality is deteriorated by the amount of coupling it exhibits, in which the $Q(S)$ is reduced from 32% to 29%. Although coupling degraded the overall quality of the software system, the quality of the software is very poor and was degraded by 9%. This measurement helps software engineers understand the quality of their software systems and make some tradeoffs to decide whether to focus on improving the internal strength of their software components or minimize coupling to improve the overall quality.

Table 9: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Tadreeb system

$d_{ij}(c_i, c_j)$	*Post view*	*Post controller*	*Post model*	*HRE model*	*Trainee model*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
Post view	•	0.3	0.01	0.01	0.01	**0.08**	**1.0**	**92%**
Post controller	0.45	•	1	0.3	0.15	**0.48**	**0.15**	**8%**
Post model	0.01	0.01	•	0.01	0.01	**0.01**	**0.15**	**15%**
HRE model	0.01	0.01	0.01	•	0.01	**0.01**	**0.15**	**15%**
Trainee model	0.01	0.01	0.01	0.01	•	**0.01**	**0.15**	**15%**
			ACH(S), Eq. (4)				**32%**	
			Q(S), Eq. (6)					**29%**

The restaurant rating system consists of six major components as listed in Table 10. The development team has identified the components with a sole single function, namely, *Controller*, *Review*, and *Rating* components, in which their weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, their aggregated cohesions $WCH(c_i)$ is one as well. The development team has measured the aggregated cohesion $WCH(c_i)$ and the aggregated coupling $WCP(c_i)$ of the remaining components that have several functions, i.e., *User*, *Restaurant*, and *Admin*. Table 10 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall

quality of the system $Q(S)$. The quality of this software system is low as described by its internal strength, the cohesion, however, such low quality is deteriorated by the amount of coupling it exhibits, in which the $Q(S)$ is reduced from 53% to 38%, which indicates a bad overall quality of the system. The coupling has degraded the internal strength of this software system and reduced its quality by 28%. Software engineers can use this measurement to inspect the internal structure of the software system and decide whether to spend more time improving its internal strength or reducing its coupling, depending on which will provide better overall quality.

Table 10: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Restaurant rating system

$d_{ij}(c_i, c_j)$	*User*	*Controller*	*Restaurant*	*Review*	*Rating*	*Admin*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
User	•	0.45	0.15	0.15	0.01	0.01	**0.154**	**0.14**	**12%**
Controller	0.45	•	0.45	0.45	1.0	0.45	**0.56**	**1.0**	**44%**
Restaurant	0.15	1.0	•	0.3	0.3	0.01	**0.154**	**0.01**	**0.9%**
Review	0.3	.45	0.3	•	0.3	0.01	**0.272**	**1.0**	**73%**
Rating	0.01	0.01	0.01	0.15	•	0.01	**0.038**	**1.0**	**96%**
Admin	1.0	0.45	1.0	1.0	0.01	•	**0.692**	**0.057**	**2%**
				ACH(S), Eq. (4)				**53%**	
				Q(S), Eq. (6)					**38%**

The video game rating system consists of five major components as listed in Table 11. The development team has identified the components with a sole single function, namely, *VGames-DS*, *IGDB*, and *IGN* components, in which their weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, their aggregated cohesions $WCH(c_i)$ is one as well. The development team has measured the aggregated cohesion $WCH(c_i)$ and the aggregated coupling $WCP(c_i)$ of the remaining components that have several functions, i.e., *Naive-Bayes* and *GUI*. Table 11 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. The quality of this software system is average as described by its cohesion, however, such average quality is deteriorated by the amount of coupling it exhibits, in which the $Q(S)$ is reduced by 8%, from 74% to 68%, which indicates an average overall quality of the system. Software engineers can use this measurement to gain more insight into the software design and decide whether to focus on increasing component cohesion or reducing coupling among components, depending on which will provide better overall quality.

Table 11: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Video games rating system

$d_{ij}(c_i, c_j)$	*VGames-DS*	*Naive-bayes*	*IGDB*	*IGN*	*GUI*	$WCP(c_i)$	$WCH(c_i)$	$Q(c_i)$
VGames-DS	•	0.15	0.15	0.15	0.3	**0.188**	**1**	**81%**
Naive-bayes	0.15	•	0.01	0.01	0.01	**0.045**	**0.38**	**36%**
IGDB	0.15	0.01	•	0.01	0.01	**0.045**	**1**	**96%**

(Continued)

Table 11 (continued)

$d_{ij}(c_i, c_j)$	*VGames-DS*	*Naive-bayes*	*IGDB*	*IGN*	*GUI*	***WCP*** (c_i)	***WCH*** (c_i)	$Q(c_i)$
IGN	0.15	0.01	0.01	•	0.01	**0.045**	**1**	**96%**
GUI	0.3	0.01	0.01	0.01	•	**0.083**	**0.34**	**31%**
			ACH(S), Eq. (4)				**74%**	
				Q(S), Eq. (6)				**68%**

The information security awareness system consists of four major components as listed in Table 12. The development team has identified the components with a sole single function, namely, the *Globals* component, in which its weighted cohesion $r_{ij}(k_i, k_j)$ should be one based on the weights shown in Table 2, and as a result, its aggregated cohesions $WCH\ (c_i)$ is one as well. The development team has measured the aggregated cohesion $WCH\ (c_i)$ and the aggregated coupling $WCP\ (c_i)$ of the remaining components that have several functions, i.e., *SocialMediaAccount*, *User*, and *SenstiveText*. Table 12 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. The quality of this system is high as described by its cohesion, i.e., 99%, however, this high quality is reduced by the amount of coupling it exhibits, in which the $Q(S)$ is reduced from 99% to 78%, which indicates the overall quality of the system. Although this software system exhibits good quality with high internal strength, it shows how the coupling degraded its quality by 21%. This measurement helps software designers focus on inspecting their software design to identify avoidable coupling among components and increase the quality to a higher level.

Table 12: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP\ (c_i)$, aggregated cohesion $WCH\ (c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of Information security awareness system

$d_{ij}(c_i, c_j)$	*Social media account*	*User*	*Senstive text*	*Globals*	***WCP*** (c_i)	***WCH*** (c_i)	$Q(c_i)$
Social media account	•	0.01	0.6	0.75	**0.45**	**0.958**	**52%**
User	0.45	•	0.45	0.01	**0.30**	**1**	**70%**
Senstive text	0.01	0.01	•	0.01	**0.01**	**1**	**99%**
Globals	0.15	0.01	0.15	•	**0.10**	**1**	**90%**
			ACH(S), Eq. (4)			**99%**	
				Q(S), Eq. (6)			**78%**

The patient companion system consists of thirteen components as listed in Table 13. The development team has measured the aggregated cohesion $WCH\ (c_i)$ and the aggregated coupling $WCP\ (c_i)$ of these components, which have several functions. Table 13 shows the design quality of the component $Q(c_i)$, the overall cohesion of the system $ACH(S)$, and the overall quality of the system $Q(S)$. Although the system exhibits low coupling, it has very low quality, i.e., 34%, because of the low internal strength of its components. This low quality is further reduced by 6% because of the amount

of coupling it exhibits, in which the $Q(S)$ is reduced from 34% to 32%, which indicates the overall quality of the system. Although this software system exhibits low coupling, however, its quality is influenced by its low internal strength. This measurement helps software designers focus on inspecting their software design to identify the low internal strength of the system components to increase their cohesiveness and improve the system's overall quality.

Table 13: Weighted coupling $d_{ij}(c_i, c_j)$ and aggregated coupling $WCP(c_i)$, aggregated cohesion $WCH(c_i)$, component's quality $Q(c_i)$, and overall quality $Q(S)$ of the Patient companion system

$d_{ij}(c_i, c_j)$	*Login*	*Sign up*	*Home page*	*Settings*	*Asked questions*	*Question page*	*Manage accounts*	*Manage umra*	*Reserve umra*	*Create guider*	*AI*	*Search*	*Add fatwa*	***WCP*** (c_i)	***WCH*** (c_i)	$Q(c_i)$
Login	•	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.26	25%
Sign up	0.01	•	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.26	26%
Home page	0.15	0.15	•	0.15	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.05	0.38	36%
Settings	0.30	0.30	0.39	•	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.09	0.38	35%
Asked questions	0.15	0.15	0.01	0.01	•	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.38	37%
Question page	0.01	0.01	0.15	0.01	0.01	•	0.01	0.01	0.01	0.01	0.60	0.30	0.01	0.10	0.26	24%
Manage accounts	0.15	0.15	0.15	0.01	0.01	0.01	•	0.01	0.01	0.01	0.01	0.01	0.01	0.05	0.38	36%
Manage umra	0.15	0.15	0.15	0.01	0.01	0.01	0.01	•	0.01	0.01	0.01	0.01	0.01	0.05	0.38	36%
Reserve umra	0.30	0.30	0.15	0.01	0.01	0.01	0.01	0.01	•	0.01	0.01	0.01	0.01	0.07	0.38	35%
Create Guider	0.15	0.15	0.01	0.01	0.01	0.01	0.30	0.01	0.01	•	0.01	0.01	0.01	0.06	0.75	71%
AI	0.01	0.01	0.01	0.01	0.01	0.60	0.01	0.01	0.01	0.01	•	0.01	0.01	0.06	0.33	31%
Search	0.01	0.01	0.01	0.01	0.60	0.01	0.01	0.01	0.01	0.01	0.01	•	0.01	0.06	0.18	17%
Add fatwa	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	•	0.01	0.08	8%
								ACH(S), *Eq. (4)*							**34%**	
									Q(S), *Eq. (6)*							**32%**

Table 14 shows the summary of the results achieved for the software systems used in this experiment, which shows the overall cohesion of the system $ACH(S)$, the overall quality of the system $Q(S)$, and the quality degradation occurs as a result of the amount of coupling a software system has. Regardless of the quality a software system achieves, such quality is reduced by the amount of coupling it exhibits. As can be noticed, software systems with high quality are degraded by the amount of coupling they have, even software systems with very poor quality are further worsened because of the coupling they show.

Table 14: The overall cohesion of the system $ACH(S)$ and the overall quality of the system $Q(S)$ of software systems used in the experiment

No.	*Software system*	$ACH(S)$, *Eq. (4)*	$Q(S)$, *Eq. (6)*	*Quality degradation*
1	*CBIR system*	90%	55%	39%
2	*Fatwa system*	85%	79%	7%
3	*Habits tracking system*	72%	55%	24%
4	*Puzzle-solving system*	42%	31%	26%

(Continued)

Table 14 (continued)

No.	*Software system*	*ACH(S), Eq. (4)*	*Q(S), Eq. (6)*	*Quality degradation*
5	*Tadreeb system*	32%	29%	9%
6	*Restaurant rating system*	53%	38%	28%
7	*Video games rating system*	74%	68%	8%
8	Information security awareness system	99%	78%	21%
9	Patient companion system	34%	32%	6%

The results of this experiment have shown that measuring cohesion and coupling during software design can help software engineers understand the quality of their software systems by indicating the places in the system design that may need more attention and investigation to avoid or minimize the causes of software quality degradation. Software coupling is unavoidable in many cases; however, software engineers need to know when such coupling degrades quality and should be avoided or minimized, and when it is necessary and not avoidable, in which quality tradeoff may be considered. The proposed measurements in this approach have been investigated in this experiment on different software systems from different domains and different teams. It has shown different results, which provide more insights into the software systems under investigation that can help software engineers better understand the quality of their software systems.

The results provide software engineers with more insights into the software system's internal strength and understanding of the influence of coupling on software quality and guide them to localize the couplings that degrade the quality of the software or may be unnecessary to avoid or minimize. Software engineers can use the information provided by this approach to review their software designs and decide whether to improve the internal strength to increase cohesion or focus on identifying coupling among software components that degrades their qualities.

As illustrated in this experimental study, software engineers can perform tradeoffs to decide whether to focus on increasing cohesion or reducing coupling based on which will provide more quality improvement with less effort and/or cost. The existing techniques, as summarized in Appendix A, are limited to certain quality attributes, certain programming languages, or specific domains. In addition, most of these techniques require very detailed design and may require the availability of the source code, in which classes, methods, or parameters are analyzed at the early stages of software development. Unlike the existing approaches, the proposed approach in this research provides the software engineers with an overall picture or view of the quality of the software system under construction, which can help them to understand its quality as well as indicate the locations of the software system design that may be the source of a deficiency in which more investigation should be performed.

5 Conclusions

This paper investigated an approach that uses the coupling and cohesion among software components to measure the quality of software systems. In this approach, the quality of a component is measured based on its internal strength and dependency on other components, in which

the component's behavior and/or expected results depend on the output/post-conditions of other components. The dependencies between different components are defined in terms of a graph connecting components by edges, where the assumptions/preconditions of one component depend on the output/post-conditions of other components, which consequently influence the output/post-conditions of the concerned component. As a result, the concerned component's behavior and quality attributes are also influenced. The author has introduced a model and mathematical equations to compute the quality of the software components and the overall quality of the software system. In addition, heuristic weights for different types of coupling and cohesion have been introduced to simplify the software quality measurement.

The experimental study has shown that the internal strength of the software components is influenced by the coupling they exhibit, and as a result, the overall quality of a software system is degraded by the amount of component coupling it exhibits. The proposed approach can guide the software engineers to identify the locations in their software designs that need more attention and investigation to avoid or minimize coupling and improve the quality of their software systems. In addition, the proposed approach assists software engineers in reviewing their software designs to decide whether to increase the internal strength of software components and improve the components' internal qualities or focus on identifying coupling among software components that degrades their qualities, in which quality tradeoff may be employed.

Acknowledgement: The author would like to thank the participating students in the experiments of this research to evaluate their graduation projects using the proposed approach.

Funding Statement: The author received no specific funding for this study.

Author Contributions: The author confirms that the contribution of the whole paper by his own, and has reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data used in this research study experiment is not available due to the ownership and copyright issues of graduation projects.

Conflicts of Interest: The author declares no conflicts of interest to report regarding the presented study.

References

[1] P. Neto, G. Vargas-Solar, U. da Costa and M. Musicante, "Designing service-based applications in the presence of non-functional properties: A mapping study," *Information and Software Technology*, vol. 69, pp. 84–105, 2016.

[2] W. Stevens, G. Myers and L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[3] B. Mehboob, C. Chong, S. Lee and J. Lim, "Reusability affecting factors and software metrics for reusability: A systematic literature review," *Software: Practice & Experience*, vol. 51, no. 6, pp. 1416–1458, 2021.

[4] P. Kaur and S. Kaushal, "A fuzzy approach for estimating quality of aspect oriented systems," *International Journal of Parallel Programming*, vol. 48, no. 5, pp. 850–869, 2020.

[5] M. Rizwan, A. Nadeem and M. Sindhu, "Empirical evaluation of coupling metrics in software fault prediction," in *IEEE 17th Int. Bhurban Conf. on Applied Science and Technology*, Islamabad, Pakistan, pp. 434–440, 2020.

[6] D. Kim, J. Hong and L. Chung, "Investigating relationships between functional coupling and the energy efficiency of embedded software," *Software Quality Journal*, vol. 26, no. 2, pp. 491–519, 2018.
[7] A. Prajapati, A. Parashar and J. Chhabra, "Restructuring object-oriented software systems using various aspects of class information," *Arabian Journal for Science & Engineering*, vol. 45, no. 12, pp. 10433–10457, 2020.
[8] S. Almugrin, W. Albattah and A. Melton, "Using indirect coupling metrics to predict package maintainability and testability," *Journal of Systems and Software*, vol. 121, pp. 298–310, 2016.
[9] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.
[10] H. Izadkhah and M. Hooshyar, "Class cohesion metrics for software engineering: A critical review," *Computer Science Journal of Moldova*, vol. 25, no. 1, pp. 44–74, 2017.
[11] A. Parashar and J. Chhabra, "Assessing impact of class change by mining class associations," *International Arab Journal of Information Technology*, vol. 16, no. 1, pp. 98–107, 2019.
[12] X. Li, Y. Yin, L. Fiondella and Y. Zhou, "Software reliability analysis considering correlated component failures with coupling measurement framework," *Journal of Systems Engineering and Electronics*, vol. 26, no. 5, pp. 1114–1126, 2015.
[13] H. Schnoor and W. Hasselbring, "Toward measuring software coupling via weighted dynamic metrics," in *Proc. of 2018 ACM/IEEE 40th Int. Conf. on Software Engineering: Companion*, Gothenburg, Sweden, pp. 342–343, 2018.
[14] S. Rangarajan, H. Liu and H. Wang, "Web service QoS prediction using improved software source code metrics," *PLoS One*, vol. 15, no. 1, pp. e0226867, 2020.
[15] M. Papamichail and A. Symeonidis, "A generic methodology for early identification of non-maintainable source code components through analysis of software releases," *Information and Software Technology*, vol. 118, pp. 106218, 2020.
[16] P. Kaur, S. Kaushal, A. Sangaiah and F. Piccialli, "A framework for assessing reusability using package cohesion measure in aspect oriented systems," *International Journal of Parallel Programming*, vol. 46, pp. 543–564, 2018.
[17] A. Gosain and G. Sharma, "A new metric for class cohesion for object oriented software," *International Arab Journal of Information Technology*, vol. 17, no. 3, pp. 411–421, 2020.
[18] M. Ahmad and M. Cinneide, "Impact of stack overflow code snippets on software cohesion: A preliminary study," in *IEEE/ACM 16th Int. Conf. on Mining Software Repositories (MSR)*, Montreal, Canada, pp. 250–254, 2019.
[19] E. Fernandes, A. Chávez, A. Garcia, I. Ferreira, D. Cedrim *et al.,* "Refactoring effect on internal quality attributes: What haven't they told you yet?" *Information and Software Technology*, vol. 126, pp. 106347, 2020.
[20] H. Mumtaz, P. Singh and K. Blincoe, "Identifying refactoring opportunities for large packages by analyzing maintainability characteristics in Java OSS," *Journal of Systems & Software*, vol. 202, pp. 111717, 2023.
[21] D. Coutinho, A. Uchoa, C. Barbosa, V. Soares, A. Garcia *et al.,* "On the influential interactive factors on degrees of design decay: A multi-project study," in *IEEE Int. Conf. on Software Analysis, Evolution and Reengineering (SANER), Software Analysis, Evolution and Reengineering (SANER)*, Hawaii, USA, pp. 753–764, 2022.
[22] Z. Alzamil, "Software components' coupling detection for software reusability," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, pp. 320–328, 2018.
[23] M. Alzahrani and A. Melton, "Defining and validating a client-based cohesion metric for object-oriented classes," in *IEEE 41st Annual Computer Software and Applications Conf.*, Torino, Italy, pp. 91–96, 2017.
[24] L. Kumar, S. Tummalapalli and L. Murthy, "An empirical framework to investigate the impact of bug fixing on internal quality attributes," *Arabian Journal for Science & Engineering*, vol. 46, no. 4, pp. 3189–3211, 2021.
[25] H. Alshareef and M. Maashi, "Application of multi-objective hyper-heuristics to solve the multi-objective software module clustering problem," *Applied Sciences*, vol. 12, no. 11, pp. 5649, 2022.

[26] A. Prajapati and J. Chhabra, "Information-theoretic remodularization of object-oriented software systems," *Information Systems Frontiers*, vol. 22, no. 4, pp. 863–880, 2020.
[27] M. Maddeh, S. Al-Otaibi, S. Alyahya, F. Hajjej and S. Ayouni, "A comprehensive MCDM-based approach for object-oriented metrics selection problems," *Applied Sciences*, vol. 13, no. 6, pp. 3411, 2023.
[28] A. Rathee and J. Chhabra, "Improving cohesion of a software system by performing usage pattern based clustering," *Procedia Computer Science*, vol. 125, pp. 740–746, 2018.
[29] G. Myers, "Module Coupling," in *Reliable Software through Composite Design*, 1st ed., New York, USA: Van Nostrand Reinhold Company, pp. 33–54, 1975.
[30] D. Perry, "Software interconnection models," in *Proc. of IEEE 9th Int. Conf. on Software Engineering*, Monterey, CA, USA, pp. 61–69, 1987.

Appendix A: Summary of the related works

No.	Approach by	Metric used	Aim
[4]	Kaur et al. 2020	Package coupling and cohesion	Assessing external quality attributes such as reusability, maintainability, and understandability of aspect-oriented systems at the package level
[7]	Prajapati et al. 2020	Coupling and cohesion	Automating the restructuring process of object-oriented software packages to improve its design quality
[9]	Chidamber et al. 1994	Coupling and cohesion between classes, i.e., CBO and LCOM metrics	CBO represents the number of classes to which a class is coupled, in which methods of one class use methods or instance variables of another class LCOM uses the similarity degree of methods based on the commonly used instance variables between methods to measure the inter-relatedness between class methods
[11]	Parashar et al. 2019	Class coupling	Estimating the impact of the changeability of the class to maintain the software system
[12]	Li et al. 2015	Components coupling	Evaluating software reliability
[13]	Schnoor et al. 2018	Class dynamic and static coupling	Computing the coupling degree statically and dynamically
[14]	Rangarajan et al. 2020	Aggregated coupling and cohesion	Predicting the quality of services properties of web services

(Continued)

Appendix A (continued)

No.	Approach by	Metric used	Aim
[15]	Papamichail et al. 2020	Complexity, cohesion, coupling, and inheritance	Evaluating software maintainability based on four static analysis source code metrics
[16]	Kaur et al. 2018	Cohesion at the package level	Assessing the reusability of the aspect-oriented system at the package level
[17]	Gosain et al. 2020	Class cohesion	Reducing maintenance effort of classes
[18]	Ahmad et al. 2019	Low-level similarity-based class cohesion and class cohesion	Determining the impact of the addition of snippets on the program quality
[19]	Fernandes et al. 2020	Cohesion complexity, coupling, inheritance, and size	Applying refactoring and re-refactoring operations on code structure to improve its quality by understanding the effect on internal quality attributes
[20]	Mumtaz et al. 2023	Coupling, cohesion, and complexity	Identifying the maintainability issues and metrics useful for identifying refactoring opportunities for large software packages
[21]	Coutinho et al. 2022	Coupling, cohesion, complexity, inheritance, and size	Investigating the influence of process and developer-related factors on the design decay of software modules
[22]	Alzamil, 2018	Components coupling	Identifying software reusability
[23]	Alzahrani et al. 2017	Class cohesion	Measuring class cohesion during the design phase with the information from high-level design
[24]	Kumar et al. 2021	Complexity, coupling, and cohesion	Understanding whether the existence of faults in the code indicates a quality problem in software design
[25]	Alshareef et al. 2022	Coupling and cohesion	Improving the software maintenance process by improving the software design with better modularization
[26]	Prajapati et al. 2020	Coupling	Re-modularizing object-oriented software to improve its design quality

(Continued)

Appendix A (continued)

No.	Approach by	Metric used	Aim
[27]	Maddeh et al. 2023	Complexity, cohesion, coupling, and inheritance	Selecting the most suitable metric, e.g., complexity, cohesion, coupling, and inheritance, for detecting specific design defects
[28]	Rathee et al. 2018	Coupling and cohesion	Evaluating the quality and modularity of a software system at the design level to improve its overall cohesion