Quality Evaluation of Structural Design in Software Reverse Engineering: A Focus On Cohesion

CHUNG-YANG CHEN\(^1\), KUANG-YEN TAI\(^2\), AND SIN-SIAN CHONG\(^1\)

\(^1\)Department of Information Management, National Central University, Taoyuan City 320317, Taiwan
\(^2\)Department of Information Management, National Taiwan University, Taipei City 10617, Taiwan

Corresponding author: Kuang-Yen Tai (d07725001@ntu.edu.tw)

This work was supported in part by Taiwan Ministry of Science and Technology under Project 106-2221-E-008-062.

ABSTRACT Software reverse engineering (SRE) plays a crucial role in contemporary software environments. Software developers may implement a system first then use SRE tools to generate design content such as the Unified Modeling Language (UML) diagrams. In the literature of SRE, studies majorly focus on how precisely the conversion can reflect the system; there is, however, little or no research that further looks into the quality of the converted results. Therefore, this paper presents an online knowledge-based ontological SRE system, OntRECoh, for quality evaluation of converted UML structural models. OntRECoh features a domain-specific knowledge base that focuses on cohesion design and a rule-based inference engine for computing the cohesion scores of Java-based implemented systems and providing improvement recommendations through its Web-based interface. Furthermore, OntRECoh includes both static and dynamic cohesion measures from both the design and the implementation aspects, for the evaluation to be more comprehensive and synthetic in the SRE context.

INDEX TERMS Cohesion, design quality, UML structural design ontology, software reverse engineering.

I. INTRODUCTION

Because of the highly uncertain and evolutionary nature of contemporary development environments, software projects are expected to be implemented in a shorter time so that users can see actual results rapidly and provide timely feedback for corrections \([1],[2]\). Therefore, software developers may adopt flexible and empirical process approaches (such as agile) that prioritize workable software over planning-based analysis and design. These approaches prevent the work for tedious documentation due to subsequent system revisions \([1]–[3]\). However, scholars have argued that the lack of system design documentation leads to adverse effects; numerous systems cannot undergo practical and affordable maintenance and reconstruction because the maintainers do not understand the systems sufficiently \([4]–[6]\).

Given the importance of both system design and rapid development, a software project may implement a system first and then utilize software reverse engineering (SRE) to help trace back and generate the system’s design-related content. In the context of modern system design language such as UML \([7]\), software projects may use SRE to generate graphical UML designs that explain the software. This use of SRE effectively reduces the time and cost of repeated documentation work \([5],[8],[9]\). Therefore, existing SRE studies mostly report how to automatically and precisely convert implemented code into specific UML diagrams. Commercial or open-source SRE tools (e.g., ModelGoon, Object-Aid, and easyUML) have also been developed to realize this feature.

These studies and tools aim at specific object-oriented programming languages, such as Java, to generate UML diagrams to reflect the system’s implemented contents. However, if the original code’s quality is poor at the time of programming, the automatically generated UML graphical content can be inherently incorrect. However, in the literature of SRE, little or no research has addressed this challenge; there is a lack of a mechanism to assist in assessing the quality of the converted contents. Because software design verification (e.g., UML quality assessment) is a knowledge-intensive task \([3],[10],[11]\), it requires continuous accumulation of organizational knowledge to help in performing the quality assessment. In the knowledge engineering area, ontology
is a crucial research topic and has been applied to various domains. The application of ontology to software development is receiving more attention as it provides a robust feature to facilitate communication and knowledge transfer of organization members in the development tasks [12].

With the aforementioned environment, this paper aims at developing a system for online assessment of design quality in SRE. To achieve this goal, this paper applies ontology and focuses on the structural models, e.g., UML class diagrams, of system design for evaluating implemented systems written in Java code. In the quality assessment of structural design, we focus on cohesion because it is a critical quality concern in system design to help understand, maintain, and modify software more effectively [13]–[18].

The system prototype, named OntRECoH, has the following features. In terms of architecture and concept, OntRECoH comprises a cohesion-centric quality ontology model and a rule-based inference engine to evaluate the cohesion of a tested system and provide improvement recommendations. In terms of cohesion assessment, the proposed work innovatively combines static cohesion, which measures the structure of a system’s classes, and dynamic cohesion, which focuses on the runtime behavior of the classes, to assess cohesion quality from both the design and the implementation aspects in the SRE context. OntRECoH also features a Web-based interface to facilitate real-time collaboration in a decentralized development environment. The remainder of this paper is organized as follows. Section 2 reviews related literature; Section 3 presents the design of the proposed work; Section 4 demonstrates and validates the implemented results; Section 5 evaluates the proposed work; Section 6 concludes the study.

II. LITERATURE REVIEW
A. SOFTWARE REVERSE ENGINEERING AND QUALITY
SRE entails analyzing a software system’s implemented content to identify and establish a high-level abstraction of the system [19]. Traditional software development follows the process of planning, analysis, design, testing, and implementation. The use of SRE is contrary to the traditional development sequence by allowing developers to implement a software system first and then generate the associated design content from the implemented code [13]. SRE can effectively reduce the time and labor costs of a system’s design documentation if the system has frequent updates or changes. In terms of the abstract overview of system design, Unified Modeling Language (UML) is a widely used diagramming approach of graphical system representation in software industries. In object-oriented system development, SRE research has focused on transforming low-level system code into high-level UML diagrams. For example, Keschenau developed an SRE tool for converting Java bytecode into class diagrams that can reflect multiple associations and combined relationships in classes [20]. Shatnawi et al. proposed an approach for reengineering object-oriented APIs into component-based contents [21]. Sarkar et al. and Sunitha have developed methods that can automatically generate UML class diagrams for Java-based systems [22], [23].

SRE has gained increasing attention because of its usability in modern empirical and rapid software development. Despite this, a risk of SRE is the degree to which its produced output can precisely reflect the code. In this regard, few studies have investigated the quality of SRE outputs. For instance, Stepan et al. compared the quality of the SRE outputs from Java code and bytecode [24]. Gahalaut and Khandnor focused on Java code and integrated the two perspectives from developers and analysts in defining SRE performance. Although studies have investigated SRE performance and output quality, most of them have considered how to accurately convert and reflect the code in the form of produced UML diagrams. However, if the code itself is of poor quality, the poor quality can inherently affect the quality of the converted output. This challenge becomes critical in rapid software development when developers are encouraged to implement their systems rapidly without the guidance of methodical software and system design [25].

B. SOFTWARE DESIGN QUALITY: COHESION
In the design of software systems, cohesion is a massive quality concern. It refers to the correlation between members in a software system [16], [18]. A high degree of cohesion maximizes the understandability and maintainability of the system [13], [17]. Cohesion is a critical indicator to measure the quality of software design, and it contributes to the development of other software activities such as software failure prediction [15], [26], software modularization [14], identifying reusable components, and numerous other activities. In object-oriented programming languages, the class is often the design unit, and the members of a class refer to the methods and attributes in the class. In this context, Al Dallal and Briand define cohesion by looking at three types of relationships among class members: (1) method–method, (2) method–attribute, and (3) attribute–attribute. A class exhibits a higher degree of cohesion when it has at least two of the above three relationships [18]. Many studies have developed metrics for measuring cohesion. For instance, Briand et al. suggested four quantitative properties for class cohesion measures in object-oriented systems [27]. In particular, the non-negativity and normalization properties denote that the cohesion measure must belong to a specific interval (e.g., [0, 1]); the monotonicity property means that a system’s cohesion does not decrease when the cohesion increases in its modules; the property of cohesive modules means that the overall cohesion is not increased when two unrelated modules are combined into one. Al Dallal and Briand also proposed two cohesion measures, namely High-Level Design (HLD) and Low-Level Design (LLD), for different stages of measurement. In particular, HLD is utilized to identify potential cohesion problems in the initial design stage to improve them proactively and reduce the development cost. The LLD cohesion metric can be used for refactoring after
system implementation; LLD can produce more system information than HLD to identify less cohesive members for refactoring. Because this study handles implemented system contents in the SRE context, we focus on the LLD measurement [18], [28].

C. STATIC AND DYNAMIC MEASURE OF COHESION

The measurement of LDD cohesion has two categories: static and dynamic. Concerning static cohesion, most measures include two properties of software: (1) model-level information such as classes and their members (i.e., methods and attributes) and relationships (e.g., inheritance and association) between classes; and (2) code-level information regarding message passing, which can be obtained by looking into the code content of the model constructs. One of the earliest static measures, LCOM1 (Lack of Cohesion of Methods), can be traced back to Chidamber and Kemerer’s work that calculates the number of paired methods without shared variables. Scholars have extended such work to develop LCOM-related measures [29]. For example, Chidamber and Kemerer proposed LCOM2, which measures the gap between the number of paired methods without shared variables and the number of paired methods with shared attributes [30]. LCOM3, proposed by Li and Henry, expresses LCOM in the form of a graph, in which each method is a vertex, and any instance of attribute sharing is an edge between methods [31]. LCOM4, proposed by Hitz and Montazeri, is similar to LCOM3, except that edges represent the calls between methods [32]. Briand et al. proposed LCOM5, which only considers direct use between methods and attributes to calculate the number of methods used for each attribute [33].

The above cohesion measures focus mainly on the method–attribute relationship in a class. Bieman and Kang proposed Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC); these computed the number of direct or indirect associations between attributes and pairwise methods in the calculation of cohesion [34]. Badri and Badri extended TCC and LCC to propose DC_D (Degree of Cohesion-Direct) [35]. Later, Al Dallal and Briand proposed LSCC (LLD Similarity-based Class Cohesion), which uses similarity analysis and comprehensively considers the four quantitative properties to compute the degree of cohesion of a class [28], [33]. Qu et al. proposed MCC (Method Community Cohesion) and MCEC (Method Community Entropy Cohesion) [36], which establish a Call Graph to capture method calls and lay out the community structure of classes by the method calls. Correlation studies have demonstrated that MCC and MCEC can obtain cohesive measurement dimensions different from other cohesion measures. However, the measurement of MCC and MCEC only focuses on the method–method relationship; it does not consider the attribute members of a class.

Unlike static cohesion (which measures the structure of classes), dynamic cohesion measures the runtime behavior of classes (i.e., the activated cohesive linkages in classes during system execution) [37]. Dynamic cohesion is useful in various contexts. For example, it can empirically identify unnecessary linkages or calls among class members; and the measurement of polymorphic methods can be obtained only by dynamic cohesion measurements because they are determined during system runtime. The literature has several reports on dynamic cohesion measurement. For example, Mitchell & Power modified the static measurement LCOM into Dynamic LCOM (DLCOM) and Dynamic Call-weighted LCOM (DCLCOM) for measuring class cohesion under a dynamic execution environment [38]. Later, Gupta & Chhabra proposed Dynamic Class Cohesion (DCC). DCC identifies four different types of associations of class members with four corresponding dynamic measures: (1) DC_MA (Dynamic cohesion due to the ‘read’ dependency of methods on attributes); (2) cohesion measure DC_AM (Dynamic Cohesion due to the ‘write’ dependency of attributes on methods); (3) DC_MM (Dynamic Cohesion due to the ‘call’ dependency between methods); and (4) DC_AA (Dynamic Cohesion due to the ‘reference’ dependency between attributes). Their study also experimentally showed that the accuracy of DCC is better than DLCOM and DCLCOM [37].

D. ONTOLOGY AND SOFTWARE ENGINEERING

Ontology is derived from the field of philosophy; it is used to describe entities or things that exist in the real world and infer their implicit associations. The definition of ontology varies, and it is widely known as a clear specification of conceptual models [39], [40]. Ontological modeling is suitable for describing complex knowledge management [41]. To establish an ontology in a specific domain, one must address the fundamental elements, including concepts, properties, instances, etc. Web Ontology Language (OWL) is a widely used language for implementing practical ontologies [42]. With the emergence of the semantic web, rule-based inference has drawn more attention in ontological applications. Among many rule-based languages, Semantic Web Rule Language (SWRL) is the major W3C rule-based inference language that can effectively make associations with ontological models implemented with OWL [41]. The inference rules in SWRL and SQWRL (Semantic Query-enhanced Web Rule Language) comprise two parts: head and body. In particular, the header refers to the inference result, and the body is the premise of the inference. In describing the body of a rule with SWRL/SQWRL, the notation “” refers to a variable and can represent an instance of a concept or class.

Ontology has been applied to various domains (including artificial intelligence, medical engineering, library science, and software engineering) because ontology can help to formalize a conceptual model of a specific domain to establish and share explicit and implicit knowledge in this field between people and machines [43], [44]. Software engineering and development is a complex and knowledge-intensive process, and research on applying ontology to the software engineering domain has drawn much attention for effectively sharing domain-related knowledge in the collaborative development of software [45]. Numerous software engineering
ontologies have been developed. Borghini et al. divided existing software engineering ontologies into several categories: software process ontology, software document ontology, software maintenance ontology, and software quality ontology. In applying ontology to the SRE quality assessment in this study, the subsequent reviews the ontological works in the software quality category further [46].

Regarding software quality ontology, Kayed et al. surveyed 80 research reports related to software quality and extracted a list of terminologies of quality attributes to establish an ontology to provide a general semantic framework [47]. Chen and Tsai developed a requirement quality ontology (OWQFunc) to help identify the hidden requirements that require experiences and knowledge to unveil [48]. Ciancarini et al. established an ontological model (SQuAPOnt) to define software quality relational factors for financial information systems. In summary, these works provide useful references for modeling the quality of knowledge products in various stages of software development [49]. For the software development that applies SRE to generate UML-based design contents that are highly subjective and knowledge-intensive [50], researchers can use ontology effectively to ensure the quality of the produced design.

III. RESEARCH METHODOLOGY

A. FRAMEWORK

This section presents the design of the proposed work, Online Ontological Cohesion Assessment for SRE Design Quality, abbreviated as OntRECoh. As a research prototype with a focus on cohesion, OntRECoh has the following features. First, the cohesion measurement innovatively comprises both the static and the dynamic aspects for the quality assessment to be highly comprehensive. Second, OntRECoh is designed for operating in the client-server Internet environment so that users to use it without temporal and geographical restrictions. Third, to enable knowledge sharing, OntRECoh features an ontology model with the code-first and cohesion design concepts and a rule-based inference engine for providing improvement recommendations for refactoring. Fourth, users can review the generated improvement recommendations and provide feedback for updating OntRECoh to increase the correctness and completeness of its knowledge body.

The design of OntRECoh comprises three main functions: (a) loading data, (b) instance creation, and (c) results demonstration. Fig. 1 illustrates the operational workflow associated with the three functions. Specifically, the function of loading data refers to the first three steps in the figure, including the collection of the static and dynamic cohesion data of a tested software system. That is, OntRECoh obtains (1) static cohesion data by converting the structural diagramming information that is from an existing SRE tool and (2) dynamic cohesion data that is from the local development environment. Then, instance creation refers to Step 4 that instantiates the ontological model based on the above-captured cohesion information. Finally, results demonstration refers to Step 5, which consists of two parts: (i) the computation of static and dynamic cohesion scores and (ii) the recommendation for the structural reconstruction of the tested system to improve cohesion.

B. DATA COLLECTION: CAPTURE OF STATIC AND DYNAMIC COHESION INFORMATION

The first step of the proposed work is to collect the static and dynamic cohesion information in SRE contexts. After converting the implemented code of a system into the UML structural design outputs. For the consistency of the collected data with the structural content of the cohesion assessment, this paper first defines the structure of to-be-assessed software systems into the following levels: project, package, class, and class member. The class member level further consists of attributes and methods. By following the three relationships suggested by Al Dallal and Briand [18], cohesion among class members is formed based on the following scenarios: method call, attribute call (e.g., a method retrieves the data from an attribute), and assignment between two attributes. The cohesion between methods and attributes further includes two subtypes: reading data from attributes and assigning data to attributes.

With the structure mentioned above, static cohesion information is captured from the structural UML class diagrams generated by an SRE tool. The information includes the data regarding the members of a tested system as well as the data of method calls, attribute calls, and assignments between two attributes. As for the dynamic cohesion information, it is captured from the execution of the tested system. For doing so, we apply AOP (Aspect-Oriented Programming) to record the following four system behaviors of point-cuts: method call, method execution, reading values from attributes, and writing values to attributes. For instance, as the simple program in Fig. 2, indicates, the a.display() method call in
main_class represents a pointcut. Therefore, a set of dynamic cohesion information can be obtained: main_class.main() and Aspect.display().

C. MODEL CONSTRUCTION AND INSTANCE CREATION

After the server receives the data for assessment, OntRECoH creates the instance content for the tested system based on the established ontological cohesion model and then evaluates the design quality through rule-based inference. The establishment of the ontological model comprises two steps: building concepts and defining rules [40], [48]. After the concepts and rules are set, instances can be created accordingly. To facilitate the establishment process, this study utilized Protégé, an ontological model editor developed by Stanford University, and then used Jess, a plugin module associated with Protégé, for the development of the rule-based inference engine [3].

Specifically, in this study, four ontology concepts represent the static design structure of a system: Project, Package, Class, and Member. The Member concept further consists of two sub-concepts: Attribute and Method. Fig. 3 shows the structural relationships between these concepts and highlights the associated properties (the black dots in the figure) to be used in the proposed inference process.

![FIGURE 3. Static and dynamic cohesion structure in the proposed ontology model.](image)

In particular, static cohesion can be identified based on a call between methods and attributes and the attribute reference. Moreover, because the classes of an object-oriented system form an inheritance relationship, OntRECoH identifies the inheritance relationship through rule inference. The three inference properties is_Descendant, can_Refactoring, and is_Special are further explained. First, the is_Descendant property denotes whether the underlying class or the member in a class inherits from another class. Because it is difficult to know whether they inherit from a parent class (if any) when recording the members for a class, it is necessary to infer the inheritance feature of the members through the is_Descendant property. The can_Refactoring property denotes whether the inference method is relocatable. According to Tsantalis and Chatzigeorgiou’s study, refactoring is not available to the static or inherited methods [49]. The is_Special property denotes if the inferred method is of a special type (e.g., constructors and service methods) that should be exclusive in computing cohesion [37].

As for the dynamic cohesion, the model defines the following properties for collecting dynamic cohesion information: Invoked (Read/Write), is_Executed, and Referenced. Specifically, the two properties, Invoked_Read and Invoked_Write, are used to identify if a method during the runtime reads data from or writes data to an attribute. The is_Executed property denotes the actual usage of a method to justify the necessity of the linkage of method-call. The Referenced property indicates whether two attributes are interconnected, including assigning the value of one variable to another. Such an interconnection between two attributes is an indirect relationship and is identifiable through the method in which the interconnection occurs. Hence, such a property is also an inferred property, and the inference starts with finding out the attributes used by the methods whose is_Executed property is true and then sets the Referenced property to be true for the paired attributes.

To perform the inference and to set the properties, a set of rules are defined for the ontology model to generate instances and then infer the values of the properties stated above. Through the ontological model and the associated rules, we can obtain implicit knowledge that hides in the relationships [50], [51]. Table 1 presents the rules in SWRL and the associated functionality for the ontological inference in this study.

D. RESULTS DEMONSTRATION: COHESION SCORES AND IMPROVEMENT RECOMMENDATION

This study evaluates both the static and dynamic aspects of the cohesion design of an implemented system. Besides the static measures that evaluate the structural design of a tested system, dynamic cohesion measures the performance of the design by looking at the real-time operation of the system. Hence, the work of evaluation in OntRECoH comprises the computation of static and dynamic measures and the generation of improvement suggestions based on the measurement.

The proposed static cohesion measurement is based on LSCC [28] so that it not only satisfies the four quantitative measurement requirements but considers the implicit relationship due to inheritance and the special types of methods [33]. However, because LSCC is limited to a single class only, the static cohesion measure in this study extends to measure the cohesion for the entire system.

In particular, the measurement of static cohesion in this study begins with looking into the shared attributes between
TABLE 1. SWRL rules.

| Rules and semantics                                                                 | Rule association                  |
|------------------------------------------------------------------------------------|----------------------------------|
| Class(?x)^ Class(?y)^ Inheritance(?x,?y)^                                        | Static cohesion.                 |
| Class Method(?x,?m1)^ Method(?m1)^ has_Name(?m1,?n1)^ is_Private(?m1,?false) => | Static cohesion.                 |
| Method(?m2)^ has_Name(?m2,?n2)^ Class Method(?y,?m2)^ is_Descendant(?m2,true) ▷ | Inference this method’s is Descendant attribute. |
| Static cohesion.                                                                  |                                   |
| Static cohesion.                                                                  |                                   |
| Static cohesion.                                                                  |                                   |
| Dynamic cohesion.                                                                 |                                   |
| Dynamic cohesion.                                                                 |                                   |
| Dynamic cohesion.                                                                 |                                   |

In a MAR matrix, the similarity value between two columns quantifies the cohesion between a pair of methods (i and j). Al Dallal and Briand (2012) define ns(i, j) as the number between i and j rows. The degree of similarity is indicated in formula (2), where Y is the number of columns of the matrix. The degree of similarity is used here to highlight the cohesion between the two methods. The higher the degree of similarity, the greater is the cohesion.

\[
ns(i, j) = \frac{\sum_{x=1}^{Y} (m_{ix} \land m_{jj})}{Y} \quad (2)
\]

In LSCC, the formula, \( \frac{2 \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} m_{ij}}{l(k-l)} \) that computes the average of the similarity for all of the paired methods is incorporated with formula (2) as \( \frac{2 \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} m_{ij}}{l(k-l)} \). Moreover, the pairwise method similarity of the \( i^{th} \) column can be expressed as \( \frac{x_i(x_i-1)}{2} \), where \( x_i \) represents the number of 1s in the MAR matrix; therefore,

\[
\frac{2 \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} (m_{ij} \land m_{ji})}{l(k-l)} = \frac{\sum_{i=1}^{l} x_i(x_i-1)}{l(k-l)} \quad (3)
\]

As for the dynamic measure, it is developed to comprehend the performance of the structural linkages and present a system’s actual cohesion during its runtime. This study extends \( DCC \) to establish \( DOC_e \), an overall dynamic cohesion measure of a class for comprehensively considering the four cohesive properties of the data and method members in the class [37]. In particular, the first property refers to dynamic cohesion due to the write-in dependency of attributes on methods, namely \( DC_{AMx} \). Suppose that \( m \) and \( n \) are the number of attributes and methods in a class. As indicated in formula (4) below, the parameter \( r^R_{ij}(e^R_i, e^R_j) \) represents the attributes that are accessed in a method of class \( c \) by writing data to the attributes, and \( O \) is the collection of all classes in the system. The value of \( DC_{AMx} \) is between zero and one. In the case that has a method with no attributes or an attribute but no method, the value for such a write-in property is zero.

\[
DC_{AMx}(o) = \{0 \text{ when } m=0 \text{ or } n=0, \ \sum_{i=1}^{m} \sum_{j=1}^{n} r^R_{ij}(e^R_i, e^R_j) \} \quad (4)
\]

where \( e^R_i \in A(o) \land e^R_j \in M_N(o) \land o \in O \)

The second property is dynamic cohesion due to the read dependency of methods on attributes, and is denoted as \( DC_{MAX} \). As indicated in formula (5) below, the parameter \( r^R_{ij}(e^R_i, e^R_j) \) represents the attributes in a class that has been read by a method. The value of \( DC_{MAX} \) is between zero and one. The zero value denotes that the read behavior does not occur during the system’s execution time.

\[
DC_{MAX}(o) = \{0 \text{ when } m=0 \text{ or } n=0, \ \sum_{i=1}^{m} \sum_{j=1}^{n} r^R_{ij}(e^R_i, e^R_j) \} \quad (5)
\]

where \( e^R_i \in M_N(o) \land e^R_j \in A(o) \land o \in O \)

The third property, dynamic cohesion due to the call dependency between methods and denoted as \( DC_{MMx} \), is expressed in the following formula (6). The expression \( r^R_{ij}(e^R_i, e^R_j) \) represents the methods in this class that are actually called. The value of \( DC_{MMx} \) ranges from 0 to 1, and
the values of zero or one occur when there exist no methods or only one method in the class.

\[ DC_{MM}(o) = \begin{cases} 0 & \text{when } n = 0 \\ 1 & \text{when } n = 1 \\ \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} |e_i^R \wedge e_j^R| & \text{otherwise} \end{cases} \]

where \( e_i^R \in M_N(o) \wedge e_j^R \in M_N(o) \wedge o \in O \)  \hspace{1cm} (6)

The last property, dynamic cohesion due to the reference dependency between attributes, is denoted as \( DC_{AA}(o) \). As indicated in formula (7), \( r_{RR}(e_i^R, e_j^R) \) represents the number of paired attributes that are referenced in the same method. In Java code, the linkage between two variables is realized through the operation of an assignment inside a method. If the class has no attributes, such a linkage would not occur and thus the value of this property is 0. If the class has only one attribute, the cohesion is the maximum value, i.e., 1.

\[ DC_{AA}(o) = \begin{cases} 0 & \text{when } m = 0 \\ 1 & \text{when } m = 1 \\ \frac{1}{m(m-1)/2} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} |e_i^R \wedge e_j^R| & \text{otherwise} \end{cases} \]

where \( e_i^R \in A(o) \wedge e_j^R \in A(o) \wedge o \in O \)  \hspace{1cm} (7)

Taken together, formula (8), shown at the bottom of the page, denotes that the dependence relationship mentioned above and (10) respectively. In computing the degree of dispersion of cohesion score for \( n \) classes in a system, the smaller the value of \( LSCC_{MAD} \) or \( DOC_{MAD} \), the lower the degree of dispersion is.

\[ LSCC_{MAD} = 1 - \frac{\sum_{i=1}^{n} |LSCC(o) - m(LSCC)|}{n} \]
\[ DOC_{MAD} = 1 - \frac{\sum_{i=1}^{n} |DOC(o) - m(DOC)|}{n} \]

After computing the cohesion value, OntRECoh provides the improvement recommendation for refactoring, i.e., changing the internal structure to improve cohesion [52]. To provide improvement recommendations, OntRECoh applies the suggestion of Al Dallal [53], that is, to identify candidate classes for remodeling and relocating certain members in the classes. In consideration of class members for relocation, the method members with lower cohesion were selected. Such an identification process may repeat until \( LSCC_{MAD} \) and \( DOC_{MAD} \) meet the designated values.

IV. SYSTEM IMPLEMENTATION

A. OPERATING ENVIRONMENT

This section presents the implementation of OntRECoh. First of all, Fig. 4 shows the screenshots when entering the system. The following content demonstrates the system by following the workflow in Fig. 1. Specifically, in the implementation of Step 1, this study uses Eclipse, a freely IDE for Java software development environment. To capture the static information, i.e., Step 2 in the workflow, OntRECoh is associated with ModelGo, an open-sourced SRE tool that is suitable for the academic and research purpose of prototype development. For the function that captures the dynamic information, the system utilizes the AspectJ package that is from Eclipse to help record the execution traces when executing a tested software system.

![System webpage screen of OntRECoh](image)

FIGURE 4. System webpage screen of OntRECoh.

We also developed an Eclipse plugin tool named OntRE-Coh_plugin that is operated in the local Java development’s environment. OntRE-Coh_plugin converts the extracted static and dynamic information into the JavaScript Object Notation format through HTTP and then transfers the converted data to OntRECoh. At the OntRECoh server-side, this study implements Step 4 by utilizing Apache Jena to formalize the
TABLE 2. Members of the ATM case.

| Class       | Method               | Attribute            | F  |
|-------------|----------------------|----------------------|----|
| ATM         | performTransaction() | currentAccountNumber | 3  |
| ATM         | getAccount()         | bankDatabase         | 2  |
| ATM         | createTransaction()  | userAuthenticated    | 2  |
| ATM         | run()                | cashDispenser        | 0  |
| Deposit     | Deposit()            | accountNumber        | 1  |
|             | promptForDepositAm   | CANCELED             | 2  |
|             | count()              | bankDatabase         | 0  |
|             | execute()            | amount               | 1  |
| Transaction | Transaction()        | accountNumber        | 1  |
|             | Execute()            | bankDatabase         | 1  |
| Withdrawal  | Withdraw()           | bankDatabase         | 1  |
|             | execute()            | accountNumber        | 1  |
|             |                      | cashDispenser        | 1  |
|             |                      | amount               | 1  |
|             |                      | CANCELED             | 2  |
| BankDatabase| credit()             | accounts             | 1  |
| BankDatabase| getAccount()         |                     |    |
|             | getAvailableBalance()|                     |    |
|             | displayMenuOfAmounts()|                    |    |
| BalancelInquiry | execute()   | accountNumber        | 1  |
|              |                     | bankDatabase         | 0  |
| Account     | Account()           | availableBalance     | 2  |
|             | debit()             | pin                  | 1  |
|             | credit()            | accountNumber        | 1  |
|             | totalBalance        |                     | 3  |
| CashDispenser| isSufficientCashAvailable() | INITIAL_COUNT          | 0  |
|             | dispenseCash()      | Count                | 2  |
|             | validatePIN()       |                     |    |

The results shown on the screen are further verified. Specifically, for the static cohesion scores, they are inspected with formula (3) as follows:

\[
\text{LSCC (BankDatabase)} = \frac{1(0)}{10(105)} = 0
\]
\[
\text{LSCC (Account)} = \frac{2(1)+1()+1()+1()0+1(3)2}{10(105)} = 0.3333
\]
\[
\text{LSCC (Withdrawal)} = \frac{1(0)+1()+1()+1()+1(0)+2(1)1}{5(2)} = 0.2
\]
\[
\text{LSCC (Transaction)} = 1
\]
\[
\text{LSCC (CashDispenser)} = \frac{0+2(1)}{2(2)(1)} = 0.5
\]
\[
\text{LSCC (Deposit)} = \frac{1(0)+2(1)}{4(2)(1)} = 0.25
\]
\[
\text{LSCC (ATM)} = \frac{3(2)+2(1)+2(1)}{3(4)(3)} = 0.21
\]

The results shown on the screen are further verified. Specifically, for the static cohesion scores, they are inspected with formula (3) as follows.
limitations of space, only the analyses for Account and BalanceInquiry are shown as follows:

\[
\begin{align*}
DC_{AMx}(\text{Account}) &= \frac{3}{4} = 0.75 \\
DC_{MAx}(\text{Account}) &= \frac{4}{3} = 1.333 \\
DC_{MMx}(\text{Account}) &= 0 \\
DC_{AAx}(\text{Account}) &= \frac{1}{3(4-1)/2} = 0.056 \\
DOCx(\text{Account}) &= \frac{4(0.25) + 3(0.3333) + 0 + 0.056}{10} = 0.21 \\

DC_{AMx}(\text{BalanceInquiry}) &= \frac{0}{1/2} = 0 \\
DC_{MAx}(\text{BalanceInquiry}) &= \frac{2}{1/3} = 1 \\
DC_{MMx}(\text{BalanceInquiry}) &= 1 \\
DC_{AAx}(\text{BalanceInquiry}) &= 0 \\
DOCx(\text{BalanceInquiry}) &= \frac{(0) + 3(1) + 2(1) + 4}{10} = 0.2 
\end{align*}
\]

V. EVALUATION AND RESULTS

In this section, the proposed work is evaluated. The evaluation in this section focuses on three parts, i.e., the design, the expected output, and the performance of the proposed work. Specifically, the assessment of the methodological design focuses on the proposed cohesion metric, the validation of the expected output is performed by walking through a sample case, and the performance evaluation uses several actual cases for statistically analyzing the performance of the work.

A. EVALUATION OF THE METRIC DESIGN

In evaluating the design of the measures, two approaches were employed, i.e., property-based evaluation and principal component analysis [18], [28], [55], [56]. In the property-based evaluation, we utilize the six property requirements for evaluating cohesion metrics [56]. As Property 1 states, a usable metric should be calculated without having complex mathematical functions. In OntRECoh, LSCCMA and DOCMAD extend two existing metrics, i.e., LSCC and DOC, which are scoped to a single class only, to compute the overall cohesion score for the entire system. Based on the inherited property, the mathematical body of the metrics is understandable, and the extended part of the proposed metrics is not
complicated. For Property 2, i.e., the measurement should be language-independent, although OntRECOh measures Java-based systems, it evaluates the cohesion of the systems by looking at the class diagrams that are the abstract-level of system representation and is language independent.

Property 3 and 4 state that a measure should be developed on a proper scale and the metrics in a metric suite should be consistent. In this study, the measurement scale of the proposed metrics produces a rational range that is from zero to one, and the resulting values of both metrics indicate that the higher the value, the better the cohesion design of the tested system. Therefore, the two properties are satisfied. For Property 5, i.e., the foundation of a metric should be able to be explained and visualized. In the proposed work, LSCC-MAD and DOCMAD are derived from LSCC and DOC. The theoretical foundation of the computation is therefore traceable and verifiable. For visualization, the computed scores of two metrics are presented along with the cohesion values of the comprising classes of the tested system. Lastly, for Property 6, i.e., the metric should give a cohesion as a positive number, is satisfied since the values of the metrics proposed are positive or zero.

In addition, we performed principal component analysis (PCA), as it is commonly used to compare and verify newly developed metrics with the existing ones [18], [28], [57]. PCA is a statistical analysis method that linearly transforms the observations of a series of values for possible related variables (i.e., the compared metrics in this study) to analyze their main components [18], [28]. It can effectively reduce the complexity of the observed variables to reveal the data’s internal structure and scrutinize the variability of the data [58]. Thus, in this study, we utilize PCA to observe the differences between the compared metrics and then conclude the more representative measurement effect [18]. The analysis comprises two levels, the product, and its component level. At the product level, OntRECOh is the first measurement that comprehensively includes both the static and dynamic aspects in evaluating the cohesion of implemented systems in the context of SRE. Looking into the component level, the static aspect of the proposed cohesion measurement is analyzed, and the existing metrics LCOM5, DCC, TCC, and SCOM [18], [59] are selected for the comparison since they are highly related to the proposed metric suite of OntRECOh.

To report the PCA analysis, we continue to use the ATM case. The comparison is analyzed using a correlation matrix and loading matrix [18]. Table 4 below shows the correlation matrix calculated by the nonparametric Spearman correlation coefficient, in which the justification of a p-value (0.05) indicates that the components are higher correlated. The correlation values of DCC and LCOM5, i.e., 0.53 and 0.04, are closer to the p-value, meaning that the two metrics correlate to OntRECOh and thus are considered as the same group. As for the correlation among OntRECOh, TCC, and SCOM, the values are -0.33 and -0.42, indicating that they are not significantly related to OntRECOh.

Next, we utilize the results in Table 4 to compute further the loading matrix of the PCA experiment. The loading matrix helps us to group the observed components and identify the differences effectively. We obtain the principal components (PCs) by applying the varimax rotation technique [28], [60]. In this technique, the eigenvectors and eigenvalues are calculated and used to form the PCA loading matrix that transforms the original variables into weights for calculating the correlations between the variable and the principal components and highlighting the feature of the proposed metric. Based on the analyses of three different principal component groups (i.e., PC1~PC3), Table 5 shows the results for the three analyses, in which each PC indicates the influential metrics contributing to the captured dimension.

In this section, the process and the output of the proposed work are evaluated. To do so, we use walkthrough [60], [61],

### Table 4. Correlation matrix of PCA in this study.

| Metrics    | OntRECOh | LCOM5 | DCC  | TCC  | SCOM |
|------------|----------|-------|------|------|------|
| OntRECOh   | 1.00     | 0.04  | 0.53 | -0.33| -0.42|
| LCOM5      | 1.00     | 0.32  | -0.41| 0.07 |
| DCC        | 1.00     | 0.21  | -0.08|      |
| TCC        | 1.00     | -0.18 |      |      |
| SCOM       | 1.00     |       |      |      |      |

According to Hair [61], when the coefficient value is above 0.4 and below 1 in the loading matrix, the factor loading is significant and requires verifying the commonality of the variables of the group, i.e., each column of Table 5. Specifically, in the table, for PC1, the values of OntRECOh, LCOM5, DCC, and TCC are 0.72, 0.8, 0.53, and 0.42 respectively, thus they are classified as one group. These metrics similarly consider the number of attributes or methods and their interactive calls in classes as the basis in measuring cohesion. For PC2, the values of OntRECOh and DCC are 0.64 and 0.41. The commonality of these two metrics is that they both have dynamic cohesion for measuring the runtime behavior of classes. For PC3, the value of OntRECOh is 0.81 while others are insignificant, indicating that OntRECOh is structurally independent of the other metrics. In this study, OntRECOh innovatively includes both static and dynamic cohesion measures, and such an independence characteristic is identified and highlighted by PC3. In other words, the measurement information of cohesion captured by OntRECOh is more comprehensive and is not accomplished by other cohesion metrics, thus confirming its innovativeness.

### Table 5. Loading matrix of eigenvectors and eigenvalues.

|          | PC1  | PC2  | PC3  |
|----------|------|------|------|
| OntRECOh | 0.72 | 0.64 | 0.81 |
| LCOM5    | 0.80 | 0.32 | 0.23 |
| DCC      | 0.53 | 0.41 | 1.65 |
| TCC      | 0.42 | 1.33 | 0.01 |
| SCOM     | 2.36 | 0.15 | 0.11 |

### B. EVALUATION OF THE SYSTEM AND THE OUTPUTS

In this section, the process and the output of the proposed work are evaluated. To do so, we use walkthrough [60], [61],
FIGURE 7. Suggested class diagram after OntRECoH.

Table 6. Cohesion scores after the movement of three methods.

| Class Name   | LSCC Score (Static) | DOC Score (Dynamic) |
|--------------|---------------------|---------------------|
| BankDatabase | 0.6                 | 0.23                |
| BalanceInquiry | 1                   | 0.2                 |
| Account      | 0.67                | 0.3                 |
| Withdrawal   | 0.67                | 0.33                |
| Transaction  | 1                   | 0.2                 |
| CashDispenser | 0.5                 | 0.25                |
| Deposit      | 0.25                | 0.19                |
| ATM          | 0.21                | 0.22                |
| System MAD   | 0.8723              | 0.9875              |

C. EVALUATION OF ACCUMULATED SYSTEM PERFORMANCE

In this section, the system’s performance is empirically evaluated. In software and reliability engineering, the experimental approach [62] is often used to examine statistically the performance of a proposed software work through a number of test cases. Therefore, we conducted an experiment with hypotheses to examine the performance of the proposed work. In doing so, six real software cases were utilized to examine the accumulated performance of OntRECoH. The cases were from a known open-source Java empirical project on Github. These cases were selected due to different sizes, for the purpose to examine the empirical performance of OntRECoH. Notably, the largest case in the case list, Elasticsearch, is one of the most popular Java projects on Github, with 53,000 GitHub stars and 18.4K GitHub forks. The primary function of Elasticsearch is a distributed search engine built for the cloud. Table 7 shows the information of these examples.
The analysis comprises two steps. The first step began with performing OntRECoh to compute the cohesion scores for the six cases and generate the corresponding improvement suggestions. Then we modified the structural contents for each of the cases according to the improvement suggestions. After that, OntRECoh was performed again to evaluate the modified cases to obtain the improved cohesion values. In the second step, the values generated by OntRECoh were manually inspected and computed. The verification and the results are illustrated in Fig. 8 and are discussed in the following.

As the figures show, cohesion is improved for all of the cases tested. Specifically, after refactoring by following the improvement suggestions, the cohesion values of $LSCCMAD$, $LSCCAverage$, $DOCMAD$, and $DOCAverage$ were increased by 28%, 44%, 21.5%, and 63%, respectively. To further verify this result, a null hypothesis set was established as follows:
TABLE 7. The real cases in this analysis.

| Application case     | Source                                   | Software Size |
|----------------------|------------------------------------------|---------------|
| CheckerGameServer    | “Java: How to program” [54]             | 2 classes     |
| Junit4               | https://github.com/junit-team/junit4     | 22 classes    |
| Photoview            | https://github.com/chrisbanes/Photoview  | 30 classes    |
| Jenkins              | https://github.com/jenkinsci/jenkins     | 48 classes    |
| Eventbus             | https://github.com/greenrobot/EventBus   | 87 classes    |
| Elasticsearch        | https://github.com/elasticsearch        | 106 classes   |

$H_0$: OntREChoh does not significantly help improve system cohesion

$H_1$: OntREChoh significantly improves system cohesion

The Paired Sample T-test was employed to test the hypotheses. Specifically, the samples of the cases are pair-dependent, i.e., to evaluate the effect of the proposed system by comparing the cohesion scores of the cases before and after using the system. Therefore, Paired Sample T testing is eminently suitable for testing the $p$-value, and if the $p$-value is <0.05, the null hypothesis is rejected. According to the result of the test for static cohesion $LSCC$, the two-tailed test $p$-values for the cases tested are 0.003, 0.0012, 0.005, 0.003, 0, and 0.023, which are all <0.05. In terms of the dynamic cohesion $DOC$, the two-tailed test $p$-values are 0, 0.001, 0.013, 0.03, 0, 0.011, and 0.017, which are all <0.05 also. Therefore, the null hypothesis of $H_0$ is rejected and $H_1$ is accepted, indicating that the system (OntREChoh) has a statistically significant effect on the improvement of software cohesion.

D. VALIDITY ANALYSIS OF THE EVALUATION

Because the evaluation involved multiple cases, this paper addressed four types of validity threats [63]–[65] with the following treatments. Construct validity relates to the research design, i.e., how we create the measurement instruments to measure the effects [63], [66]. In the evaluation, the composition of the test cases was reviewed to help ensure the validity of the design. Specifically, the five cases had a comprehensive composition of implemented Java classes that realized the three principal structural associations for effective cohesion assessment such that the examination can completely cover the scope of the proposed work. External validity is addressed in two aspects. First, when operating a new system, familiarity is critical to the stability of system performance. In the evaluation, the testers who operated the system were trained to be familiar with the selected SRE tool, the Java IDE, and the proposed system. As the system is widely used, we suggest providing a proper user manual instructing the cooperation of the software components in OntREChoh to minimize this threat. The second aspect refers to the generalizability of system output [64], [65]. Although the evaluation demonstrated a good result, it was conducted with the cases scoped to Java applications; this scoping constituted a limitation concerning the application of the performance result.

VI. CONCLUSION

A. RESEARCH CONTRIBUTION

This paper has presented a knowledge-based ontological system for online quality evaluation on the converted UML structural models in the SRE environment. Several research contributions are summarized as follows. First, this study contributes to extant SRE literature by innovatively adding quality assurance to the automated reverse engineering process for the work of SRE to be more comprehensive. As existing SRE research majorly focuses on the automation of conversion from software code to design contents, this study extends to provide an online quality evaluation approach and implemented system to ensure the design quality of the reversed contents. Second, in evaluating the design quality in terms of cohesion, this research contributes to including both static and dynamic cohesion and extends to examine the structure of a system from not only the design (static) perspective but also the runtime performance (dynamic) aspect.

Third, in measuring the cohesion, the proposed work extends the existing metrics that focus on a single class to compute the degree of dispersion of the classes for the entire system, for showing the mutual relationship among classes when considering the system as a whole in the attempt to improve the overall performance of the system. Fourth, the assessment of software design quality is knowledge-intensive, and it requires continuous feedback and updates about newer experiences and knowledge. This study has provided an ontology for not only assimilating and exploiting an explicit knowledge model of cohesion design in the assessment of software design quality but also enabling case-based implicit knowledge inference for future cohesion improvement.

B. RESEARCH LIMITATIONS AND FUTURE DEVELOPMENT

Based on the current development of this study and the proposed system, we highlight several research limitations for future research and development. First, the dynamic cohesion metric in this study includes the four parameters in observing the runtime activation of the linkages among class members. Future studies can extend to allow users to customize the weights of the parameters. Second, some classes are inherently less cohesive, such as abstract classes or interfaces, which may intrinsically affect the validity of the quality assessment result. In the future, the formulation of the computation can further allow users to assign weights for various classes to avoid this potential confounding effect and for the outcome of cohesion scores to be more explanatory. Third, due to a research prototype, the proposed work is evaluated with an experimental approach. Readers can expect that the further development of the work will extend to include an empirical investigation. Besides, the improvement suggestion
information is text-based in the current version of the prototype; further research can extend to present the information on the displayed UML diagram.

Lastly, the proposed design quality evaluation in the SRE context focuses on cohesion, which is also a limitation of this study. However, as system design quality includes other factors, such as security, complexity, and coupling, future research can extend to include these quality concerns and integrate with current work for establishing a more comprehensive scheme of design quality evaluation in SRE.

REFERENCES

[1] M. Brhel, H. Meth, A. Maedche, and K. Werder, “Exploring principles of user-centered agile software development: A literature review,” Inf. Softw. Technol., vol. 61, pp. 163–181, May 2015.

[2] T. Ding, S. S. N. Der, V. Balijepally, and N. B. Moe, “A decade of agile methodologies: Towards explaining agile software development,” J. Syst. Softw., vol. 85, no. 6, pp. 1231–1231, Jun. 2012.

[3] C. Y. Chen and T. H. Hsu, “The development of online and ontological quality evaluation on UML structural design in software reverse engineering,” J. Inf. Manage., vol. 26, no. 4, pp. 379–412, 2019.

[4] R. Chari, C. A. Gonzalez, and J. Cabo, “Smart bound selection for the verification of UML/OCL class diagrams,” IEEE Trans. Softw. Eng., vol. 45, no. 4, pp. 412–426, Apr. 2019.

[5] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, D. Caivano, and I. Ramos, “Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance? A family of experiments,” Inf. Softw. Technol., vol. 57, pp. 644–663, Jan. 2015.

[6] G. Canfora, M. Di Penta, and L. Cerulo, “Achievements and challenges in software reverse engineering,” Commun. ACM, vol. 54, no. 4, pp. 142–151, Apr. 2011.

[7] M. Ozkaya and F. Erata, “A survey on the practical use of UML for different software architecture viewpoints,” Inf. Softw. Technol., vol. 121, pp. 1–27, Mar. 2020.

[8] U. Sabir, F. Azam, S. U. Haq, M. W. Anwar, W. H. Butt, and A. Amjad, “A model driven reverse engineering framework for generating high level UML models from Java source code,” IEEE Access, vol. 7, pp. 158931–158950, 2019.

[9] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot, “MoDisco: A model driven reverse engineering framework,” Inf. Softw. Technol., vol. 56, no. 8, pp. 1012–1032, 2014.

[10] R. S. Bashir, F. Lee, S. U. R. Khan, V. Chang, and S. Farid, “UML models consistency management: Guidelines for software quality manager,” Int. J. Inf. Manage., vol. 36, no. 6, pp. 883–899, Dec. 2016.

[11] M. Genero, A. M. Fernández-Sáez, H. J. Nelson, G. Poels, and M. Piattini, “Research review: A systematic literature review on the quality of UML models,” J. Database Manage., vol. 22, no. 3, pp. 46–70, Jul. 2011.

[12] J. Morente-Molinerà, R. Al-Hmouz, A. Morfeq, A. Balamash, and M. Genero, “A domain ontology for software requirements change management in global software development environments,” in IEEE Access, vol. 8, pp. 53374–53393, 2020.
[46] S. Borghini and L. Mazzieri, “On the mass of static metrics with positive cosmological constant: II,” *Commun. Math. Phys.*, vol. 35, no. 12, 2020, Art. no. 125001.

[47] A. Kayed, N. Hirzalla, A. A. Samhan, and M. Alfayoumi, “Towards an ontology for software product quality attributes,” in *Proc. 4th Int. Conf. Internet Web Appl. Services*, 2009, pp. 200–204.

[48] C. Y. Chen and H. J. Tsai, “Applying ontology to online functional deployment of latent and knowledge-intensive IS quality requirements,” *J. Inf. Manage.*, vol. 21, no. 1, pp. 45–82, 2014.

[49] N. Tsantalis and A. Chatzigeorgiou, “Identification of move method refactoring opportunities,” *IEEE Trans. Softw. Eng.*, vol. 35, no. 3, pp. 347–367, May 2009.

[50] W. Hao, J. Chen, G. Chen, R. Kang, Z. Zhang, and A. Zou, “Ontology alignment repair through 0–1 programming,” *IEEE Access*, vol. 7, pp. 155424–155436, 2019.

[51] Y. Wand and R. Weber, “Research commentary: Information systems and conceptual modeling—A research agenda,” *Inf. Syst. Res.*, vol. 13, no. 4, pp. 363–376, Dec. 2002.

[52] B. B. Mayvan, A. Rasoolzadegan, and A. J. Jafari, “Bad smell detection using quality metrics and refactoring opportunities,” *J. Softw. Evol. Process*, vol. 32, no. 8, Aug. 2020, Art. no. e2255.

[53] J. A. Dallal, “Identifying refactoring opportunities in object-oriented code: A systematic literature review,” *Inf. Softw. Technol.*, vol. 58, pp. 231–249, Feb. 2015.

[54] P. Deitel and H. Deitel, *Java: How to Program*. Upper Saddle River, NJ, USA: Prentice-Hall, 2018.

[55] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, “Using structural and semantic measures to improve software modularization,” *Empirical Softw. Eng.*, vol. 18, no. 5, pp. 901–932, Oct. 2013.

[56] G. Rakić, M. Tóth, and Z. Budimac, “Toward recursion aware complexity metrics,” *Inf. Softw. Technol.*, vol. 118, Feb. 2020, Art. no. 106203.

[57] M. O. Cinnéide, I. H. Moghadam, M. Harman, S. Counsell, and L. Tratt, “An experimental search-based approach to cohesion metric evaluation,” *Empirical Softw. Eng.*, vol. 22, no. 1, pp. 292–329, Feb. 2017.

[58] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th ed. Oxford, U.K.: Blackwell, 1989.

[59] G. Bavota, A. De Lucia, and R. Oliveto, “Identifying extract class refactoring opportunities using structural and semantic cohesion measures,” *J. Syst. Softw.*, vol. 84, no. 3, pp. 397–414, Mar. 2011.

[60] T. Dingöy, N. B. Moe, T. E. Fagri, and E. A. Seim, “Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation,” *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 490–520, Feb. 2018.

[61] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Berlin, Germany: Springer, 2007.

[62] R. S. Pressman and B. Maxim, *Software Engineering: A Practitioner’s Approach*, 9th ed. New York, NY, USA: McGraw-Hill, 2019.

[63] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson, and R. L. Tatham, *Multivariate Data Analysis*, 6th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

[64] B. Kitchenham, L. Pickard, and S. L. Pfleeger, “Case studies for method and tool evaluation,” *IEEE Softw.*, vol. 12, no. 4, pp. 52–62, Jul. 1995.

[65] R. Malhotra, *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. Boca Raton, FL, USA: CRC Press, 2016.

[66] D. A. Broniatowski and C. Tucker, “Assessing causal claims about complex engineered systems with quantitative data: Internal, external, and construct validity,” *Syst. Eng.*, vol. 20, no. 6, pp. 483–496, Nov. 2017.

[67] M. Staron, “Validity evaluation,” in *Action Research in Software Engineering*. Cham, Switzerland: Springer, 2020, pp. 169–189.

**CHUNG-YANG CHEN** is currently a Distinguished Professor with the Department of Information Management, National Central University, Taiwan. His research can be seen in many journals, such as *Information Systems Journal, Journal of System and Software, Information Technology and People, European Journal of Operational Research, Computers in Human Behavior, Journal of Enterprise Information Management, Journal of Engineering and Technology Management, International Journal of Project Management, IEEE Transactions, Total Quality Management and Business Excellence, IEEE TRANSACTIONS ON EDUCATION, and Assessment & Evaluation in Higher Education*. His research interests include enhancement of information systems, such as software development and process improvement, project management, and quality management systems, such as data/information quality, software quality, and software engineering education.

**KUANG-YEN TAI** received the M.S. degree from the Department of Information Management, Tunghai University, Taiwan. He is currently pursuing the Ph.D. degree with the Department of Information Management, National Taiwan University, Taiwan. He is currently a Lecturer with the Center for General Education, Tunghai University, Taiwan. His main research interests include software engineering, artificial intelligence, information security, and parallel computing.

**SIN-SIANG CHONG** received the M.S. degree from the Department of Information Management, National Central University, Taiwan. He is currently served as a Software Engineer with TITANSOFT. His research interests include software engineering and project management.