Received: 11 August 2018 | Revised: 8 October 2018 | Accepted: 9 October 2018

DOI: 10.1002/spe.2656

SURVEY PAPER

WILEY

Software component identification and selection: A research review

Shabnam Gholamshahi | Seyed Mohammad Hossein Hasheminejad

Department of Computer Engineering, Alzahra University, Tehran, Iran

Correspondence
Seyed Mohammad Hossein Hasheminejad, Department of Computer Engineering, Alzahra University, Tehran 19938 93973, Iran.
Email: SMH.Hasheminejad@Alzahra.ac.ir

Summary

Nowadays, with the development of software reuse, software developers are paying more attention to component-related technologies, which have been mostly applied in the development of large-scale complex applications to enhance the productivity of software development and accelerate time to market. Component-based software development is well acknowledged as a methodology, which establishes the reusability of software and reduces the development cost effectively. Two crucial problems in component-based software development are component identification and component selection. The main purpose of this paper is to provide a reference point for future research by categorizing and classifying different component identification and component selection methods and emphasizing their respective strengths and weaknesses. We hope that it can help researchers find the current status of this issue and serve as a basis for future activities.

KEYWORDS

component identification, component selection, software component

1 | INTRODUCTION

Over the last decade, the extensive uses of software have placed new demands on the software industry especially for improving quality and enhancing development productivity; for this purpose, so many research studies have been done.

An increasing number of software project developers had presented component-based solutions to overcome the current software crises. Although the idea of component-based development was mooted as of 1969 by McIlroy,[1] it was not prevalent until 1918 when component technologies became popular. Having components that can be trusted and used in several projects will have a considerable impact on the speed of software development.[2] As systems get more complex, the task of finding a near-optimal solution will become more difficult for a human; hence, the process of finding an automated or a semiautomated solution to improve software architecture or resource allocation is a big dream in software engineering.[3] When you have a large software project, the appropriate component extraction process becomes more difficult, and we need an automated method to identify the components of our software systems; this case is ideal, but more existing methods have provided semiautomatic methods. Component-based software development (CBSD) created a method to deal with highly complex software systems. CBSD is the process of assembling existing software components in an application such that they interact to satisfy a predefined functionality,[4] and these components can be reused in the further system.

In the context of CBSD, components are the lego blocks or, in other words, the fundamental building block for a software system,[5,6] and there are different component definitions, which were used in component identification (CI) approaches. In general, a component is an independent software package that provides functionality via well-defined interfaces.

However, the published approaches use different component definitions as a basis for the identification or selection. These definitions range from domain-oriented component concepts to architectural viewpoints. On the basis of component definitions published in the literature, components can be classified into three categories: business component, logical component, and technical component. One of the most famous definitions of components is consistent with the overall thrust of these definitions, notably Szyperski's[7]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." In the work of Baster et al,[8] a component is defined as abstract, self-contained packages of functionality performing a specific business function within a technology framework. These business components are reusable with well-defined interfaces. Most architecture-focused software component definitions usually do not contain a statement about the domain-specific functionality and concentrate instead on logical characteristics, eg, structuring in required and provided interfaces.[9] In the end, technical component definitions concentrate on deployment and implementation aspects.[10]

There are two ways to achieve software development by using components, namely, (1) CI and (2) component selection (CS), both of which are crucial problems in CBSD. In most cases, CI has to be accomplished at the beginning of the development process, and it serves as a basis for the next design steps as well as the composition and usage of components. On the other side, CBSD is interested in the integration of preexisting software components for building software systems. The CBSD life cycle has four phases[11]: (1) domain analysis, (2) component design and development, (3) component cataloging and retrieval, and (4) component selection and application assembly.[12]

During the development of a CBSD software product, available components are assembled or adapted to an application with some development tools, eg, assemble tools, packaging tools, operating systems, a database management system, and an application server. The providers of these tools or architectures play quite distinct roles and collaborate together to deliver the final software system. These software organizations, according to the roles they play, can be classified into component developers, tool providers, environment providers, component broker, and software developers (some literature works also name them as the application assembler).[13] The overall procedures of CBSD application software and providers involved as well as their roles are given in Figure 1. The environment providers are typically operating system, database system, application server, or web server vendors. The tool provider is the company or person that creates development and assembly and is responsible for packaging tools used by component providers and software developers. The term "component provider" refers to the organization that extracts general business logic and designs, which, in turn, develops and delivers components. The term "component broker" is generally understood to mean the group that evaluates and procures components from component developers and builds a component library or store. Sometimes, a component broker also designs and develops components by himself. A software developer is defined as a company that

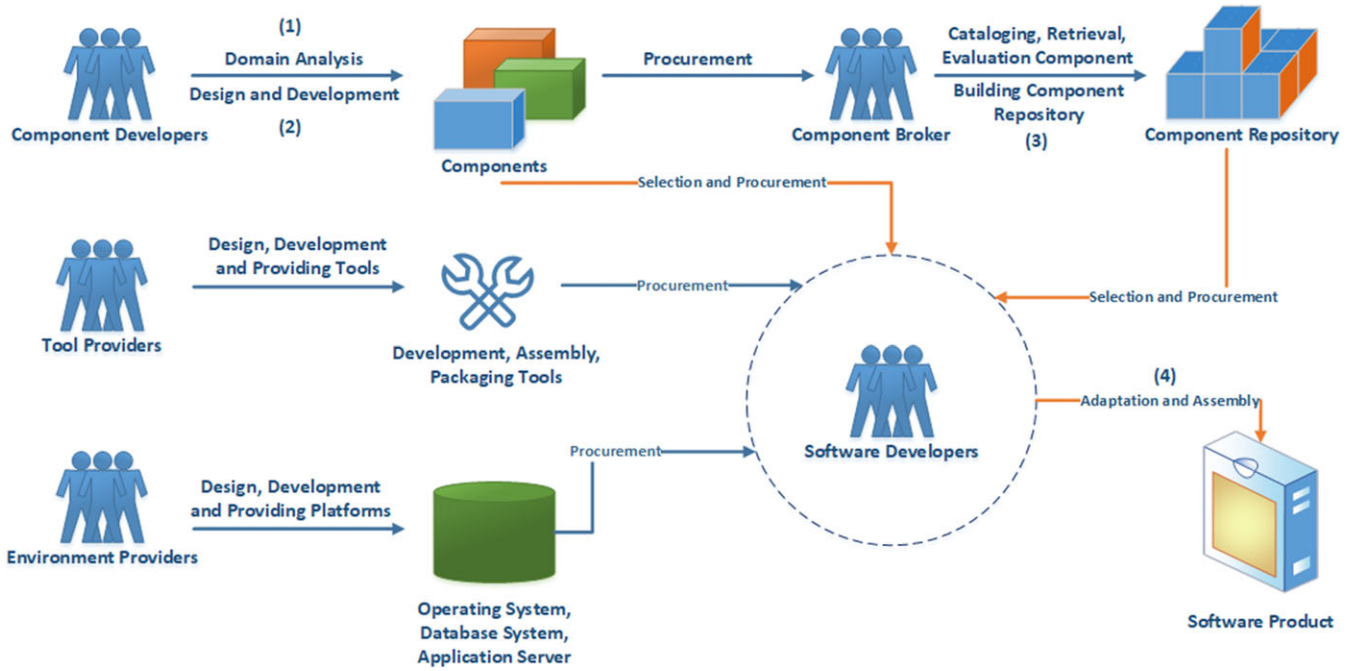

FIGURE 1 Component-based software development flowchart [Colour figure can be viewed at wileyonlinelibrary.com]

procures components from component providers or component brokers and then assembles them together to implement applications according to customer requirements. It must be noted that Figure 1 is introduced for software developers to assist them in selecting components and assembling such components into a final system.

In the following, we will try to demonstrate a review of the articles that are represented in the two areas of CI and CS. In Section 2, we illustrate how to select and extract information from articles. Section 3 is devoted to CI papers, and Section 4 is devoted to CS papers. Finally, Section 5 is dedicated to this paper's conclusion.

2 | RESEARCH METHOD

This research was conducted according to the procedure used by Kitchenham and Charters,[14] which, in turn, fits into three phases. The first phase involves planning the review and creating the research protocol to be followed. The second phase entails conducting the review, subsequently performing the search, selecting the papers, and extracting and synthesizing data. The third phase takes reporting the review and specifying the dissemination mechanisms and formats of the main report. In this sense, such a phase resulted in the elaboration of this paper. In this section, we describe how the first two phases were executed.

2.1 | Planning the review

The main goal of the review is to provide an overview of this field and to analyze the approaches' elements, such as artifacts, search methods, artifact representations, evaluation methods and results, and the like, which are used in the experimentation.

2.2 | Research question

Our review aims at answering the research questions presented in Table 1. They were defined with the goal of covering the main aspects of CI and CS approaches.

We provide answers to these questions in Sections 3 and 4. RQ1 is answered for CI in Subsection 3.1 and for CS in Subsection 4.1, RQ2 is answered for CI in Subsection 3.2 and for CS in Subsection 4.2, RQ3 is answered for CI in Subsection 3.3 and for CS in Subsection 4.3, RQ4 is answered for CI in Subsection 3.4 and for CS in Subsection 4.4, RQ5 is answered for CI in Subsection 3.5 and for CS in Subsection 4.5, RQ6 is answered for CI in Subsection 3.6 and for CS in Subsection 4.6, and RQ7 is answered for CI in Subsection 3.7 and for CS in Subsection 4.7.

2.3 | Search string

To build the search string (Table 2), we defined the main terms (CI and CS) and synonymous terms identified by analyzing search strings used in related works.

2.4 | Paper selection criteria

We established a set of inclusion and exclusion criteria adopted in selecting the primary sources presented in Table 3. First, the papers were chosen according to the inclusion criteria. Then, the exclusion criteria were applied in the selected papers. A paper was included or excluded by reading it in the following order: title, abstract, introduction, conclusion, and the entire paper; by doing so, there is no doubt in the selection.

TABLE 1 Research question

RQ No.	Research Question
RQ1	What type of artifacts/criteria is used to identify/select components?
RQ2	What methods have been used to identify/select the components?
RQ3	How are the artifacts represented?
RQ4	What are the obtained results and criteria for identifying/selecting component?
RQ5	What are the most commonly used case studies?
RQ6	What are the used evaluation methods?
RQ7	What are some best ideas for future works?

TABLE 2 Research question

Search Strings
("Softwar" OR "Logical" OR "Business") AND ("Component") AND ("Identification" OR "Identify" OR Identification") AND ("Method" OR "Evolutionary" OR "Algorithm" OR "Clustering" OR "Artificial intelligence" OR "Multi-Objective" OR "Multi Objective" OR "Formal Concept Analysis" OR "Neutral Networks" OR "Soft Computing" OR "Software Engineering")
("Software" OR "Logical" OR "Business") AND ("Component") AND ("Selection" OR "Select") AND ("Method" OR "Evolutionary" OR "Clustering" OR "Algorithm" OR "Artificial intelligence" OR "Multi-Objective" OR "Multi Objective" OR "Component Composition" OR "Software Engineering")

TABLE 3 Inclusion and exclusion criteria

Inclusion	Exclusion
Papers that use any technique to identify/select a component from an artifact is given as input	Out of Scope; Not available online; Not in English or Persian; Abstracts, posters, technical reports, thesis, keynote, doctoral symposiums, book reviews, and patents.

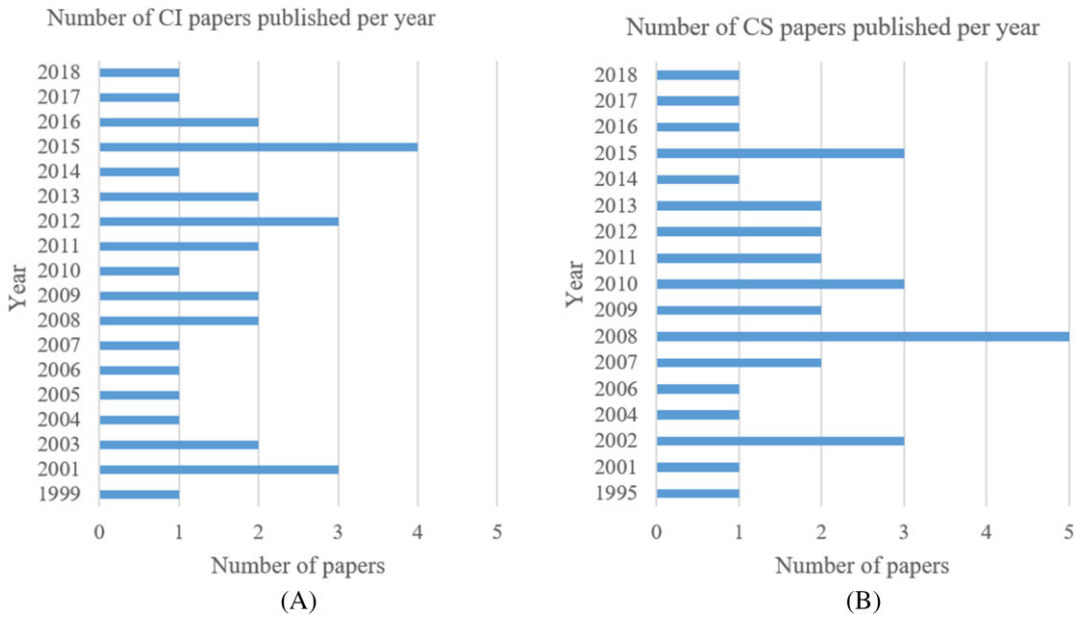

FIGURE 2 Number of (A) component identification (CI) and (B) component selection (CS) papers published per year [Colour figure can be viewed at wileyonlinelibrary.com]

2.5 | Paper selection criteria

The first step was the selection of primary sources. This can be fulfilled by using the search string in the selected engines; needless to say, the title, abstract, and keywords should be taken into account in this respect. The engines for making the search are online repositories chosen based on the fact that they win popularity and they provide many leading software engineering publications. The selected sources are as follows: (1) ACM Digital Library, (2) IEEE Xplore, (3) Springer, (4) Science Direct, (5) Web of Science, and (6) Wiley Online Library. Figure 2 illustrates the number of CI (1999-2018) and CS (1995-2018) published papers.

3 | COMPONENT IDENTIFICATION

CI is one of the most important and difficult tasks in developing component-based systems. The existing component development approaches do not provide a standard for CI, and they depend on the experience of developers. In this section, we pay attention to CI approaches' details.

3.1 | Artifacts used in the identification of a component

In this section, we answer RQ1 and describe what type of artifacts is used in CI approaches. Table 4 shows the artifacts, the number of papers (NP), and the references. As shown in Figure 3, most papers applied use cases, and some of them focus on class diagrams and source codes. Moreover, if you pay attention to Table 4, most papers used a combination of artifacts, and a few of them applied other types of artifacts, like requirements definition and static/dynamic call graphs. What is evident is that most papers that focused on CI at the early stages of software design applied more use cases and class diagrams, but when it comes to the legacy system, it is totally different.

Legacy software systems are characterized informally as old software systems that are often poorly designed and documented but still perform a useful job for the business-critical application that must be supported over the years; however, they are not easily dealt with but remain vital to our organizations.[15] Legacy systems continue to be used because of the prohibitive cost of replacing or redesigning them, despite their poor competitiveness and compatibility with modern equivalents.[16] However, these legacy software systems have suffered from lack of standardization and openness, difficulty of change, and absence of distributed architecture.[17] Instead of continually maintaining these legacy systems at a high

TABLE 4 Artifacts used by the papers

Artifacts	NP	References
Static/dynamic call graphs	1	21
Requirements definition	1	22
Relationship between classes	2	23,24
Collaboration diagram	3	23,25,26
Analysis class diagram	4	2,27-29
Business model documents	7	26,30-35
Source code	8	19,21,24,36-40
Sequence diagram	8	20,22,23,30,41-44
Class diagrams	11	20,22,23,25,41-47
Use cases	20	2,9,20-23,25,27-31,41-45,48-50

Abbreviation: NP, number of papers.

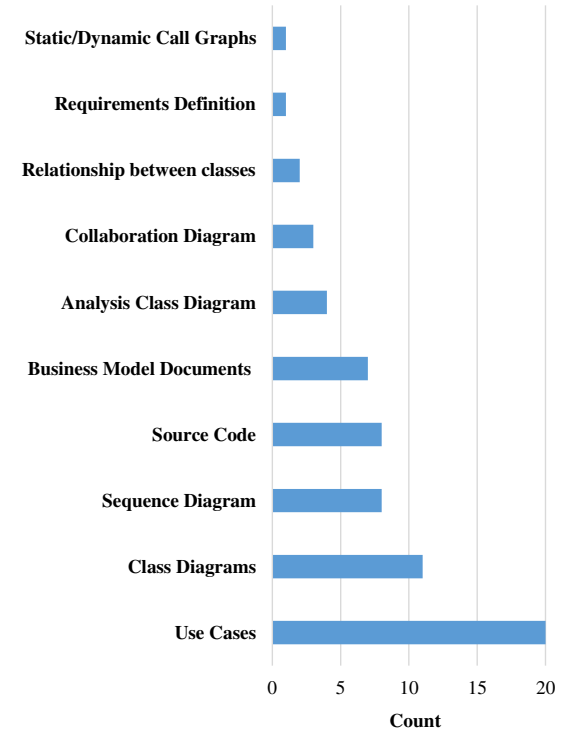

FIGURE 3 Component identification artifact usage [Colour figure can be viewed at wileyonlinelibrary.com]

cost, reengineering them to new systems with good design can improve their understandability, reusability, and maintainability. Therefore, the main issues in software reengineering consist of gathering the software entities and making meaningful (highly cohesive) and independent (loosely coupled) components. The common process of CI starts by parsing the source code of a legacy system, and then, the source code is organized into cohesive subsystems that are loosely interconnected by a particular clustering algorithm.[18] In this case, CI is a critical part of software reengineering.[19] Some papers[19-21] focused on CI for legacy systems. As shown in Table 4, use case is one of the most important artifacts among all of them.

3.2 | CI research methods

In this section, we answer RQ2, provide an overview of CI methods, and elaborate on the various approaches published in the literature. The methods are presented in Table 5 and Figure 4, along with their references. A variety of methods has been proposed for identifying components as projected in Table 5.

Here, three categories are taken into consideration. Two categories are the most widely used methods of (1) *clustering algorithms* and (2) *heuristic/metaheuristic methods*, and the third category refers to (3) *other methods*, which is evidently a set of methods less utilized than the former categories. In the case of methods that identify components in the early stages of software design, it should be noted that all these methods map functionalities of a system to a model and then try to partition the functionalities into some components. However, in the other methods that identify components in legacy systems, the goal is to classify the system classes into components. However, when the two methods are taken into consideration, we see that clustering methods have some disadvantages compared to heuristic/metaheuristic methods.

First, in most of the methods, there is a need to determine the number of components or subcomponents in advance, but approaches that used evolutionary algorithms like those in the works of Hasheminejad and Jalili[2,27] and Albani et al[34] and some other methods do not require the number of components in advance. Second, many of these methods do not consider cohesion and coupling simultaneously and consider some clustering criteria like sum of squared errors and variance ratio criterion; however, some new approaches using clustering improve these disadvantages. Third, most of these approaches need manual weighting of the features used in clustering and are highly dependent on expert opinion.

By comparison with the evolutionary approaches (EAs), clustering methods' search component space outdid the first one due to using powerful optimization search algorithms, eg, genetic algorithm (GA). The EAs cannot guarantee an

TABLE 5 The algorithms used by the papers

Methods	NP	References
Clustering Algorithms		
Hierarchical clustering	5	19,30,33,35,38
Customized clustering algorithm	4	22-24,43
K-means	1	50
Seed algorithm	1	41
Cohesion algorithm	1	41
Graph-based clustering	1	21
Heuristic/Metaheuristic Technique		
Genetic algorithm (GA)	4	2,27,28,47
Particle swarm optimization (PSO)	1	
Evolutionary programming	1	46
Harmony search	1	37
Not mentioned	1	41
Other Methods		
Set of methods	6	9,20,25,38,39,44
Graph partitioning	3	32,34,36
Neural networks	2	40,48
Formal concept analysis (FCA)	2	26,45
CRUD-based	2	42,49
Affinity analysis technique	1	31

Abbreviations: CRUD, create, read, update, delete; NP, number of papers.

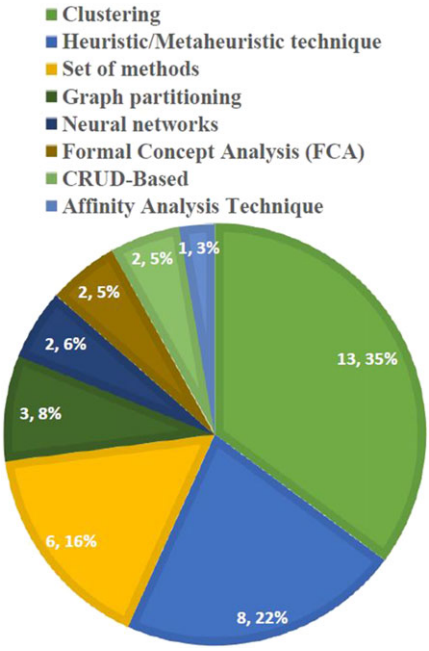

FIGURE 4 Component identification methods. CRUD, create, read, update, delete [Colour figure can be viewed at wileyonlinelibrary.com]

optimal solution because they are based on a metaheuristic method, but they achieved a near-optimal solution than clustering approaches. In addition, some approaches, like that in the work of Hasheminejad and Jalili,[27] attempt to improve the conventional GA and propose several guided and component-oriented operators to avoid getting trapped in local optima. Contrary to other methods, these approaches do not require the number of components in advance, and in this connection, they provide facilities for software architects to identify logical components according to their preferences, including deployment constraints, implementation framework constraints, and legacy components that have to be reused.

One of the issues that must be considered is the amount of artifacts to be used in the presented methods, the information on which is in Figure 5. Applying use cases is one of the similarities of the methods used to identify the component in the early stages of software design. In other words, these methods classify them based on the similarities and differences of the system performances. However, most of the methods for legacy systems used source codes, and if there is initial information on system design, they have applied them. Other information and similarities of these methods in different aspects are also presented in the next sections.

However, according to the process of using these methods, it can be said that although, in Figure 5, the class of clustering methods has the highest usage, in recent years, taking advantage of heuristic and metaheuristic methods has been increased based on the abovementioned merits. Another issue that seems to be widely used is the application of multiobjective methods to solve this problem, and due to the widespread and increasing complexity of software systems, further research studies will be undertaken in this respect.

Clustering

Most papers use clustering algorithms to identify components. Among the clustering algorithms, hierarchical clustering is the most used method. It starts from a set of individual entities that are first grouped into small clusters; these are, in turn, grouped into larger clusters until reaching final all-inclusive clustering.[37] All of the clustering methods include three common key steps: (1) determining the object features and data collection, (2) computing the similarity coefficients of the data set, and (3) executing the clustering method. One advantage of these algorithms is that they are unsupervised and do not need any extra information such as the number of expected clusters and candidate regions of the search space for locating each cluster, and produce a multilevel architectural view that helps in architectural understanding.

Heuristic/metaheuristic technique

Depending on the structure of the CI problem and its complexity in large software systems and the ineffectiveness of other methods in dealing with these systems, some papers have turned to the use of heuristic/metaheuristic methods to solve this problem. Among these methods, EAs are the most used and, consequently, the GA, which is the most used EA method.[2,27,28,47] Evolutionary computation[51] is one of the first population-based and bioinspired metaheuristics for the resolution of optimization problems. For this reason, evolutionary computation has been applied to a variety of

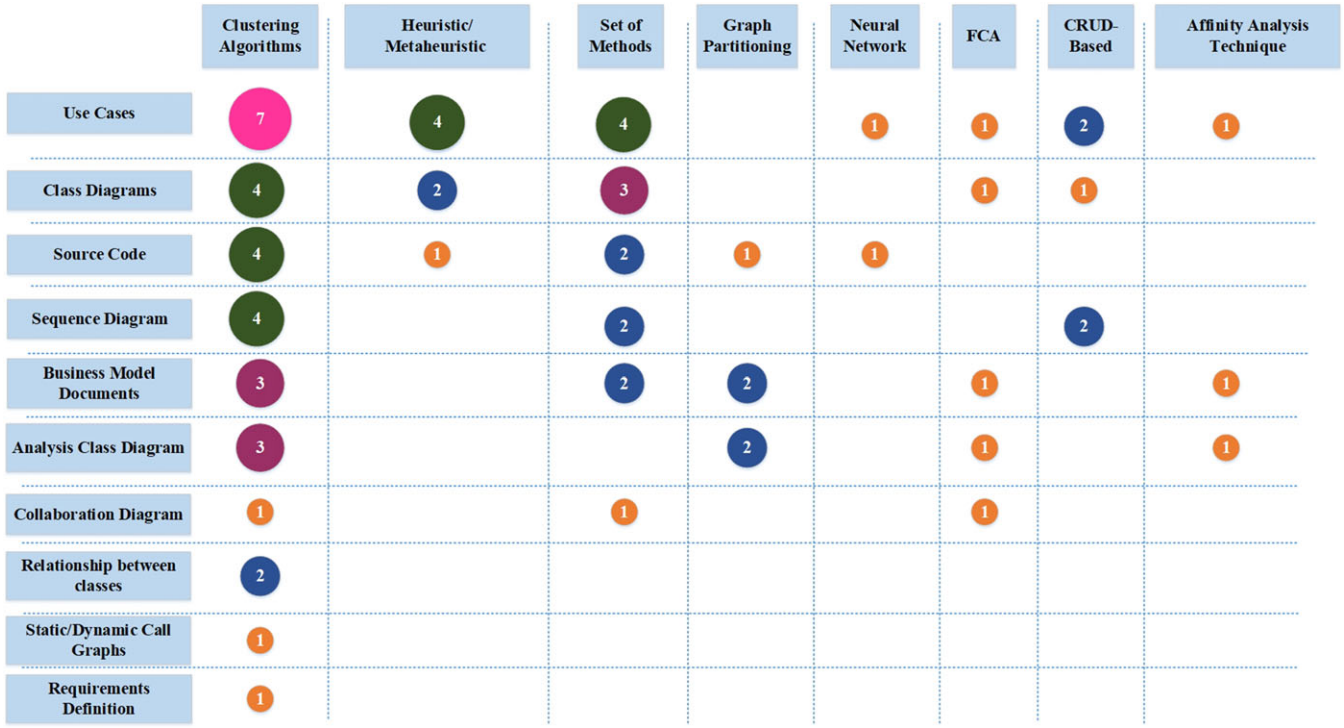

FIGURE 5 Number of used artifacts in component identification methods. CRUD, create, read, update, delete; FCA, formal concept analysis [Colour figure can be viewed at wileyonlinelibrary.com]

topics, and considerable efforts have been applied in order to propose new techniques and operators[52] to solve complex applications. Therefore, a new scope of software engineering has appeared, which is called search-based software engineering, to reformulate software problems as optimization problems. The term "search-based software engineering" was coined by Harman and Jones,[53] although there has been work on the application of search techniques to software engineering problems as early as 1992.[54] The CI problem is an NP-complete problem,[25] and the search-based approaches are crucial alternatives for solving NP-complete optimization problems that have achieved promising results.[55]

Set of methods

Some literature works have used a set of these methods. For example, in the work of Shahmohammadi et al,[25] a set of clustering algorithms has been run over the given system. According to the best results, one of these methods has been selected the best and, as a consequence, introduced to the expert. In the work of Kebir et al,[38] two algorithms have been applied in two separate sections of the component detection. In the initial steps of the CI, the *hierarchical algorithm* has been used for primary estimation, and then, the GA has been applied to improve and obtain better results. In the work of Hashemi and Shahmohammadi,[9] in the initial step, the particle swarm optimization algorithm has been used to determine the number of clusters, and then, the fuzzy algorithm has been applied with clustering to determine the number of components. In the work of Garg and Dahiya,[44] the slicing technique[56] and clustering methods have been adopted to identify the components.

Graph partitioning

Graph partitioning is a common technique in computer science, engineering, and related fields. Peng et al[32] and Albani et al[34] used this method and mapped their model (business and domain model) into a weighted graph, and then, a graph segmentation method is applied on the graph to identify components. An advantage of their heuristic is that there is no need to define the number of components in advance. This approach has two limitations: (1) weights are manually assigned to edges according to expert experiences, and (2) it requires determining the number of components in advance.

Artificial neural network

Among artificial intelligence (AI) algorithms, two approaches[40,48] used neural networks. This method is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. Artificial neural networks (ANNs), like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Formal concept analysis

Formal concept analysis (FCA) is a mathematical theory about concepts and concept hierarchies. It explicitly formalizes the extension and intension of a concept, their mutual relationships, and the fact that increasing intent implies decreasing extent and vice versa. On the basis of the lattice theory, it allows deriving a concept hierarchy from a given data set.[57] In the domain of software engineering, FCA has typically been applied to support software maintenance activities and for the identification of object-oriented structures.[58] For example, Hamza[45] proposed a framework that used the theory of FCA and the concepts of the software stability model to partition a class diagram into logical components; it was mentioned that concept analysis has several advantages over cluster analysis, including associating features to the analysis results and assigning elements to multiple groups.

CRUD-based

In this type of approaches, four relationship types—create (C), read (R), update (U), and delete (D) with the priority C>D>U>R—are used to specify the semantic relationship between elements. The relationships are visualized in a matrix. Examples of CRUD matrix methods are found in other works.[43,49] One of the disadvantages of CRUD matrix methods is not using available additional information in the domain models.

Affinity analysis

Affinity analysis is a data analysis and data mining technique that discovers co-occurrence relationships among activities performed by specific individuals or groups. In general, affinity analysis largely involves the process of finding connections between different items based on the subject's view of these items. The first step in affinity analysis is to identify the subject, which may be defined on certain conditions. The next step is to observe and then record the habits of this subject. On the recording, certain patterns start to emerge, which can be used for making connections. For example, Jang et al[31] proposed the CI method using this method.

3.3 | Representation of artifacts

In this section, we answer RQ3 and present the types of representations of artifacts used to manipulate the artifacts presented in Table 6 and Figure 6. Some works do not mention the representation, but it is possible to guess that the artifact is used.

TABLE 6 Representation of artifacts that are used in component identification

Representation	NP	References
Feature vector	25	2,9,19,20,22-25,27-31,35,38-40,42-44,46-50
Graph	7	21,32-34,36,37,41
Lattice	3	21,26,45

Abbreviation: NP, number of papers.

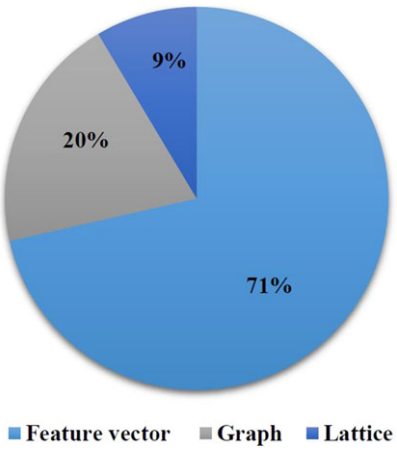

FIGURE 6 Number of used artifacts in component identification methods [Colour figure can be viewed at wileyonlinelibrary.com]

Feature vector

In most papers, the feature vector is used as an artifact representation, when a data object is described by a set of features represented as a vector. These features are as follows: quantitative or qualitative, continuous or binary, and nominal or ordinal. In Table 7, we presented a sample of the feature vector with the name "use case/property matrix" that is used in the work of Hasheminejad and Jalili,[27] and each entry denotes the relationship between use cases and the corresponding property. In this matrix, each entry is either 0 or 1, ie, the 1 entry denotes that the ith use case (UC_i) has a relationship with property j ($EntityClass_j$ or $Actor_j$) and 0 otherwise. Another sample of the feature vector known as the CRUD matrix is shown in Table 8 and represents the relationship between use cases, and classes are defined as being of four types: "create," "delete," "write," and "read." Many researchers determine some priorities as follows: create>delete>write>read. In another feature vector known as the chromosome as in Figure 7, use cases are encoded as genes in a chromosome. The value of each gene can be an integer number that represents the component number of the corresponding use case.

Graph

The second most used artifact representation is the graph, such as the business entity (object) relation graph in the works of Allier et al[21] and Fan-Chao et al[33] and the actor and object usage graph in the work of Lee et al[41] shown in Figure 8D; this graph is obtained from two sequence diagrams shown in Figure 8A,B and from the class diagram shown in Figure 8C. In Figure 8D, the nodes and "Gen" represent the classes and generalization relationship in the class diagram, respectively.

TABLE 7 A sample of the "use case/property" matrix

	Actor 1	Actor 2	...	Actor *i*	EntityClass 1	EntityClass 2	...	EntityClass *m*
UseCase 1	0/1	0/1		0/1	0/1	0/1	0/1	0/1
UseCase 2	0/1	0/1		0/1	0/1	0/1	0/1	0/1
...	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
UseCase *n*	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

TABLE 8 A sample of the CRUD (create, read, update, delete) matrix

	Class 1	Class 2	Class 3	Class 4
UseCase 1	C	R	D	R
UseCase 2	W	R	C	C
UseCase 3	R	C	W	D
UseCase 4	W	R	C	W

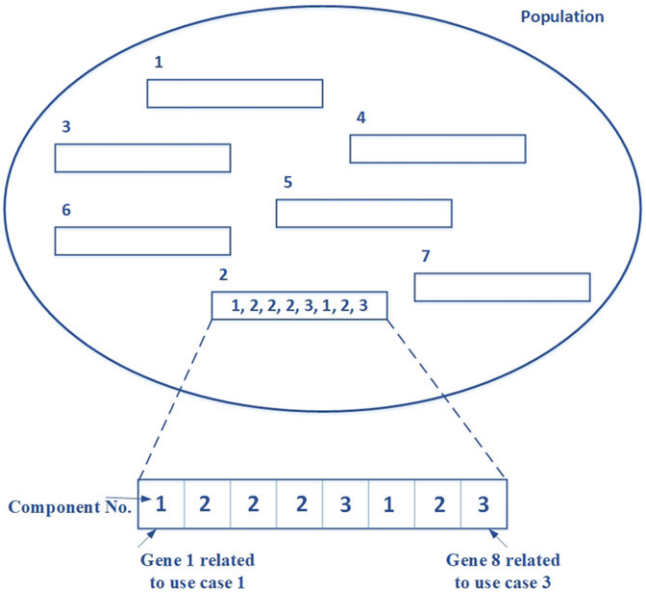

FIGURE 7 Sample population with highlighted chromosome [Colour figure can be viewed at wileyonlinelibrary.com]

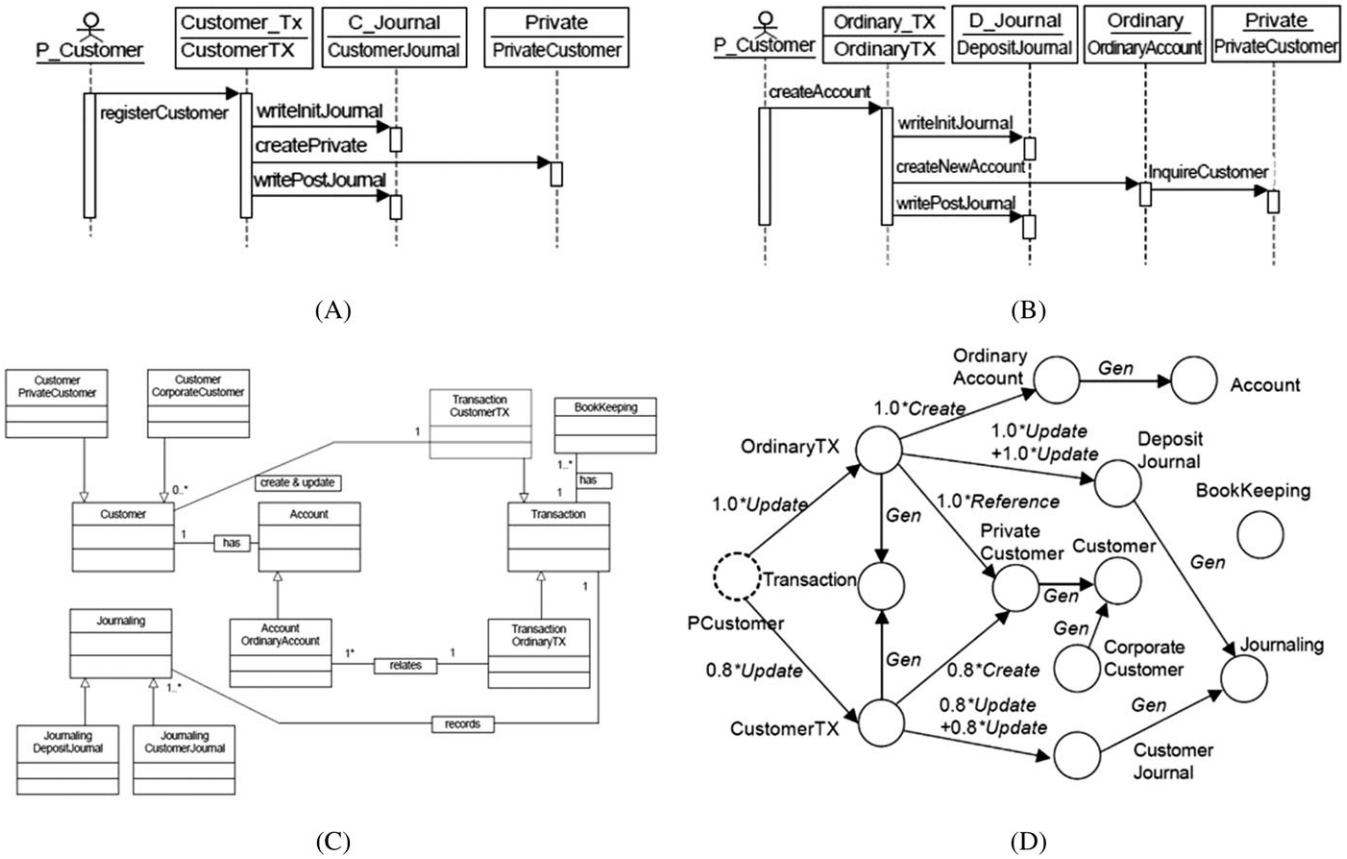

FIGURE 8 A, Sequence diagram for registering a private customer; B, Sequence diagram for creating a new account; C, Class diagram for domain objects in a banking; D, Example of the actor and object usage graph

Object usages, such as *1.0*Update* and *0.8*Update+0.8*Update*, are extracted from sequence diagrams (Figure 8A,B) on the assumption that the weights of the use case "create a new account" and the use case "register a private customer" are 1.0 and 0.8, respectively.

Lattice

The other representation is the concept lattice, which provides all the concepts and their partial order from a specific formal context. In general, a lattice is an abstract structure studied in the mathematical subdisciplines of order theory and abstract algebra, and it consists of a partially ordered set in which every two elements have a unique supremum (also called a least upper bound or join) and a unique infimum (also called a greatest lower bound or meet). There are two kinds of approaches for building the concept lattice: the batch algorithm[59] and the incremental algorithm.[60] The latter is preferable for its efficiency; however, the mentioned method proposes a new procedure for building the concept lattice. For example, in the work of Cai et al,[26] the concept is a term with extent (business elements) and intent (their properties), and each concept is a potential component candidate. The authors proposed an incremental algorithm, where all the nodes in the lattice are indexed by the layer. A sample of the concept lattice mentioned as an example is shown in Figure 9, where each o is a business object, p is the business operation, and a is the object's attributes. Needless to say, a business object used in object-oriented programming is a representation of parts of a business. A business object may represent a person, place, event, business process, or concept and exist as, for example, a manifest, a product, a transaction, or even details of a person. The business operations are executed to implement specific business functions by changing the object status. Object attributes are facts relating to the purpose of the business object.

3.4 | Results and main features

In this section, we answer RQ4 and outline the results and used criteria and features of methods. In general, it is difficult to compare the CI methods, because they have employed different criteria to identify the component. In Appendix A1, we review these criteria. Another issue that must be noticed is that not all these methods could obtain the ideal result;

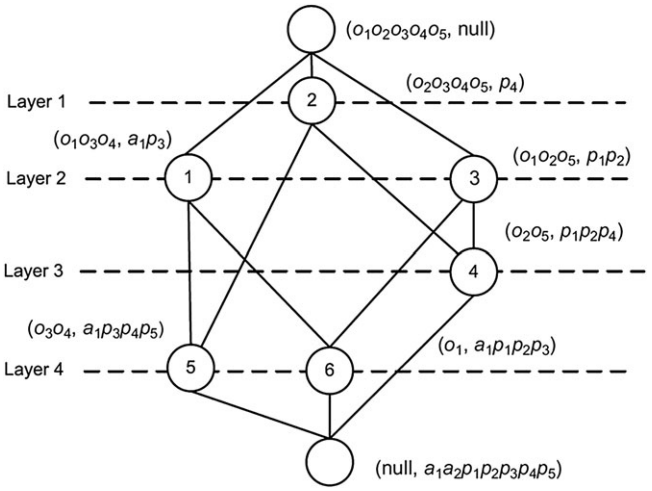

FIGURE 9 Sample of the concept lattice

therefore, an accurate evaluation is required. However, according to the information presented in the previous sections and the information given in the literature and in the Table presented in Appendix A1, it can be said that the evolutionary methods excelled over other methods. With the advent of the EA in software engineering, we are now able to automatically identify logical software components based on a powerful optimization search algorithm. However, using an evolutionary search algorithm leads to an increase in complexity in general and time complexity in particular. It should be noted that to identify logical components at an early stage of software design, it is not necessary to have a real-time method, and these approaches identify logical components during an acceptable time. Contrary to other methods, especially clustering, it can be concluded that all of the clustering-based methods used classic clustering techniques; however, they may achieve poor components due to their simple heuristics, and they have the problem of determining the best number of components in advance. In the following, samples of numerical results are given. As can be observed in Section 3.6, one way to evaluate the obtained results is to compare those based on expert opinion and those based on approach. Their results are listed in Table 9.

Table 10 shows the values of the main and important functions of the articles that provide a basis for evaluating and comparing the proposed method with other methods.

3.5 | Case studies

This section answers RQ5, and we provide the mentioned case studies in all of the reviewed papers. Many systems are used as case studies to validate the approaches proposed in the papers. Approaches are evaluated by using different numbers of case studies that can be industrial or academic. Table 11 shows the case studies used, the references, the number of papers using them, and their access links.

TABLE 9 The best results of the difference between the number of components identified by methods and the number of components identified by experts

	Best Results	
Ref	**No. of Identified Components by Approach**	**No. of Identified Components by Experts**
[28]	4	3
[9]	6	6
[27]	6	6
[2]	9	9
[25]	6	6
[42]	4	4
[33]	Not mentioned	Not mentioned

TABLE 10 Case study systems used by the papers

Best Results				
Comparison Based on the DBI Metric				
CCIC[28]	*SCI-GA*[2]	*Agglomerative Hierarchical*	*RBR*	
Without constraint handling = 0.0094	0.0103	0.0282	0.0188	
With constraint handling = 0.0297				
Comparison Based on the F-Measure				
CCIC[28]	*SCI-GA*[2]	*Agglomerative Hierarchical*	*RBR*	
Without constraint handling = 0.58	0.56	0.61	0.51	
With constraint handling = 0.94				
Comparison Based on Time				
CCIC[28]	*SCI-GA*[2]	*Agglomerative Hierarchical*	*RBR*	
Without constraint handling = 138.6 s	98-2 s	1.8 s	2 s	
With constraint handling = 1891.2 s				
Fan-Chao et al[33]	Not mentioned	Not mentioned		
Comparison Based on Cohesion and Coupling (Mixed) (Cohesion-Coupling)				
Ahmadzadeh et al[48]	*SOM*	*SCI-GA*[2]	*Shahmohammadi et al*[25]	*Kim and Chang*[22]
0.15	0.1	−0.3	−4.8	−1.6
Comparison Based on Cohesion and Coupling				
SBLCI[27]	*Shahmohammadi et al*[25]	*Cai et al*[26]	*Lee et al*[43]	*Manually*
Cohesion = 0.969	0.927	0.919	0.825	0.966
Coupling = 0.159	0.163	0.183	0.171	0.159
Comparison Based on the Fitness Function (SoftwareCohesion-SoftwareCoupling)				
SBLCI[27]	*Shahmohammadi et al*[25]	*Cai et al*[26]	*Lee et al*[43]	*Manually*
0.810	0.764	0.736	0.654	0.807
Comparison Based on the Fitness Function (Quality Metric [Q])				
SCI-GA[2]	*Shahmohammadi et al*[25]	*Cai et al*[26]	*Lee et al*[43]	*Manually*
97	83	87	60	100
Comparison Based on the Number of Components				
Choi and Cho[42]	*RUP*	*Lee et al*[23]	*Ham et al*[61]	*Manually*
4	1	4	4	4

Abbreviations: CCIC, clustering analysis classes to identify software components; DBI, Davies-Bouldin index; RBR, repeated bisections; RUP, rational unified process; SBLCI, search-based logical component identification; SCI-GA, software component identification using genetic algorithm; SOM, self-organizing map.

3.6 | Methods for evaluating the obtained results

This section answers RQ6, and we present the evaluation metrics and methods. Table 12 provides the metrics and methods that are used to evaluate the identified components. In general, more than one metric is used in these papers. Among all the metrics, cohesion and coupling are the most common for CI result evaluation, especially those with clustering approaches; some approaches[2,9,19,23,27,42,46] defined formulas for calculating cohesion and coupling. In fact, the coupling and cohesion aspects of software components are the fundamental quality attributes that can seriously affect software architecture maintenance, evolution, and reuse. Generally, cohesion is the "inner connection" between properties of a design unit; in other words, cohesion is a metric for measuring the interoperability of functionalities within a design unit, and vice versa, and coupling represents how tightly one component interacts with other components. Therefore, the main goal is to group related objects into components in such a way that the identified components have ***high cohesion*** and ***low coupling*** with others.[22,23,29,31,43] In other words, by comparison with other methods, expert opinion is a common method and is used to evaluate identified components; some approaches[2,28] used this method. In fact, this method computes the relative difference between the number of components identified by the proposed method and the number of components identified by experts. Other evaluation methods are Customize Fitness Function/Tools. Provided functions or tools are based on defined metrics that are developed to facilitate component identification and evaluation. The fitness value of a solution is used to compare the solution quality to others or to analyze it during the optimization process; Hasheminejad and Jalili[27] and Ahmadzadeh et al[48] used this method and introduced different criteria to evaluate the quality of components.

Another method, which has been widely used as the two previous methods, in comparison with other methods, whether qualitative or quantitative, is where some samples are pointed in Tables 9 and 10. Other methods given in Table 12 have been used less than others; however, they can be used to make a more accurate evaluation.

TABLE 11 Case study systems used by the papers

Case Study Systems	Ref	NP	Available On/Details	Academic/Industrial
Not mentioned	22,23,32,33,40,41,43,49	8	_____	_____
Home appliance control system (HACS)	9,25,28,29,48	5	http://www.coursehero.com/file/3666313/, /HACSFINAL/Accessed 2016 Aug	Industrial
Online stock brokerage system (OSBS)	2,25,27-29	5	http://www.isr.umd.edu/~austin/ense621.d/projects04.d/project_gouthami.html	Industrial
Agri insurance system	2,28,29	3	http://www.aiiri.gov.ir/HomePage.aspx?TabID=1&Site=aiiriPortal&Lang=en-US	Industrial
JDOM	38,39	2	http://www.jdom.org	Industrial
Apache HTTP components	38,39	2	http://hc.apache.org	Industrial
Aqualush	46,47	2	http://www.ifi.uzh.ch/en/rerg/research/aqualush.html	Industrial
Datapro4j	46,47	2	http://www.uco.es/grupos/kdis/kdiswiki/index.php/Datapro4j	Industrial
JHotDraw	21,37	2	http://www.jhotdraw.org/	Industrial
NekoHTML	46,47	2	http://nekohtml.sourceforge.net/	Industrial
Chain-store system (CSS)	25,34	2	http://ws-i.org/SampleApplications/SupplyChainManagement/2002-11/SCMUseCases-0.18-WGD.pdf	Industrial
Parking Lot System (PLS)	9,27	2	http://www.modares.ac.ir/en/Schools/ece/grp/cmp/lab/SCSLAB/Project/Project1	Industrial
JavaCC	19	1	https://javacc.org/	Industrial
Point of sale (POS) software case studies	9	1	_____	Industrial
Quantitative system business (QSB)	48	1	_____	Industrial
Library Management System	20	1	_____	Academic
Loan generation system (LGS)	29	1	https://taam.bankmellat.ir/TAAM/jsp/customer/main.jsp	Industrial
Enterprise storage management system	36	1	http://designfest.acm.org/Years/2002.htm	Industrial
Air traffic control (ATC)	27	1	http://www.modares.ac.ir/en/Schools/ece/grp/cmp/lab/SCSLAB/Project/Project1	Industrial
DefectPredictor	19	1	62	Industrial
LBFSrv	19	1	63	Industrial
BattlefieldSim	19	1	_____	Academic
Jaga	19	1	http://www.jaga.org/download.html	Industrial
MTGForge	19	1	https://www.openhub.net/p/crosis	Industrial
Rhino	19	1	https://github.com/mozilla/rhino	Industrial
Openreports	19	1	http://sourceforge.net/projects/oreports/	Industrial
SableCC	19	1	http://www.sablecc.org/	Industrial
RSSOwl	19	1	https://github.com/rssowl/RSSOwl	Industrial
Jigsaw	19	1	http://openjdk.java.net/projects/jigsaw/	Industrial
JEval	38	1	http://hc.apache.org	Industrial
Apache Commons Email	39	1	http://commons.apache.org/email/	Industrial
Restaurant Automation System (RAS)	2	1	http://www.modares.ac.ir/en/Schools/ece/grp/cmp/res/lab/SCSLAB/Project/Project2	Industrial
Jsapar	47	1	https://sourceforge.net/projects/jsapar/	Industrial
Java2HTML	47	1	http://www.java2html.de/	Industrial
Marvin	47	1	http://marvinproject.sourceforge.net/en/download.html	Industrial
Logo Interpreter	21	1	http://naitan.free.fr/logo/.	Industrial
Internet shopping mall problem in the domain of electronic commerce	42	1	_____	Academic
Auto-Insurance Claims Domain	30	1	_____	Industrial
Online marks analysis system	50	1	_____	Academic
Open Service Gateway Initiative System (OSGiS)	31	1	https://www.osgi.org	Industrial

(Continues)

TABLE 11 (Continued)

Case Study Systems	Ref	NP	Available On/Details	Academic/Industrial
OpenProj	36	1	https://sourceforge.net/projects/openproj/	Industrial
ArgoUML	37	1	http://argouml.tigris.org	Academic
JFreeChart	37	1	http://www.jfree.org/jfreechart/	Industrial
Order processing system	21	1	_____	Industrial
A B2B auction system	21	1	_____	Academic
Presta Shop	26	1	http://www.prestashop.com/en/downloads	Industrial
Fish Cart	26	1	http://fishcart.org/download.html	Industrial
Custom System	28	1	http://www.unece.org/trans/bcf/etir/welcome.html	Industrial
Inventory management system	35	1	_____	Industrial
Apache Ant	26	1	http://ant.apache.org/	Industrial
Enterprise storage management system (ESMS)	45	1	http://designfest.acm.org/Problems/Storage/Storage_02.pdf	Industrial
Online web-forum	44	1	_____	Industrial
Book, Movie tape Rental System (BMRS)	24	1	64	Industrial

Abbreviation: NP, number of papers.

TABLE 12 Component identification evaluation metrics and methods to obtain the best results

Metrics	Ref	Metrics	Ref
Cohesion	2,9,20-27,29-34,37-43,46-48	Groups/components ratio (GCR)	47
Cohesion	2,9,20-27,29-34,37-43,46-48	Mean absolute	40
Comparison with other approaches	2,9,19,20,22,25,27-29,37,42,43,48	Root mean square error	40
Comparison with expert opinion	2,9,23,25,27-29,33,42	Stability index (SI)	27
Customize Fitness Function/Tools	2,19,27,32,37,39,46-48	Coupling index	27
Not mentioned	36,44,45,49,50	Autonomy	38
Complexity	24,27,30	Composability	38
Cost	28,30,33	Sensitivity analysis	25
Precision	26,28,35	Package count index (PCI)	37
Ray-Turi index	9,25	Number of different use cases	27
Calinski-Harabasz (CH) index	9,25	Average error per criterion function (QCF)	25
Time	28,47	Variance ratio criterion (VRC)	28
Recall	28,35	Sum of squared errors (SSE)	28
Davies-Bouldin index (DBI)	28	_____	

3.7 | Future work

In this section, we answer RQ7 and provide future works in the field of CI. One of the future works in many papers that used evolutionary algorithms is the application of other optimization algorithms such as particle swarm optimization, ant colony, or hybrid algorithms to improve the search performance. In addition, some ideas as using multiobjective algorithms are appropriate for the structure of this problem, due to dealing with multiple objectives where some of them may be in conflict with others.

The topic that most papers have discussed is providing a fully automatic approach for the CI process without any human interaction; for example, they could lead to set weights based on the main criteria automatically, using automatic design pattern selection in order to design classes of each component,[2] or the development of a Component Description Language that can be used to formally specify identified stable components.[45] However, the presentations of semiautomated methods are a good improvement in this field.

The main limitation of the single-objective optimization approach is optimizing a single criterion and generating only a single solution at each run. Therefore, formulation of the CI problem as a multiobjective optimization (MOO) problem and solving using multiobjective EA can be a good idea for future works. Many papers use functional attributes as their similarity metric, but one of the issues involved in most of them is using nonfunctional attributes such as complexity, reliability, and reusability. In addition, many papers present some approaches to improve the dependency among all the

properties and extend their approaches to apply it to larger systems or multiple versions of the same system to obtain highly reusable components. As was mentioned before, use cases are used more often than the other artifacts, but many methods do not consider use case text. However, this kind of artifact can be used by some techniques like text mining.

One of the important topics that should be given more attention in the future is testing the proposed approaches in real case studies and comparing them with other methods for a better understanding of their advantages and disadvantages.

4 | COMPONENT SELECTION

The goal of CBSD is the decomposition of a software system into functional or logical components with well-defined interfaces, and with the growing need for complex software systems, CBSD has emerged as a key element in the development of complex software systems. The aim of CS methods is to select a set of components from a component repository, which can satisfy a given set of requirements and, at the same time, minimize/maximize various criteria (ie, cost, number of used components, etc); however, in a CBSD process, the selection of components is an activity that takes place over multiple life cycle phases. The CS methods are traditionally done in an architecture-centric manner, but another type of CS approaches is built around the relationship between requirements and components available for use. In general, the component set (known as the reuse library) can be either bought as commercial off-the-shelf (COTS) components from third-party vendors or developed using in-house–built software components (build-or-buy decisions). The idea of building a component in-house comes when a completely customized job is required and the technology is easily available and cheaper, or the reliability of the available COTS component cannot be trusted, and the available component may not be compatible with the proposed system architecture. There is always a tussle between build-or-buy decisions.

COTS describes software or hardware products that are ready-made or available for sale, lease, or license to the general public.[65] However, we can say that the use of COTS products has grown steadily. In both cases, one of the main goals of a development process is to select the "best" set of (COTS and in-house) components in each phase.[66] COTS products as a way of managing cost and developing time and effort[67] require less code that needs to be designed and implemented by the developers, and the vendor of the COTS supplier provides information about the cost and delivery time of the COTS products. With all these advantages of COTS, there are some risks, which are associated with using COTS products, such as functional problems, security issues, compatibility issues, integration and interoperability issues, vendor issues, procurement and licensing issues, and testing issues.

In order to clarify the CS problem, the process of building a COTS-based software system is of paramount importance. In the work of Cechich,[68] six main steps are presented to successfully implement a COTS-based system. These six steps consist of the following cases:

- analyze software requirements;
- evaluate and select COTS solution(s);
- negotiate terms with COTS vendors;
- implement the COTS-based solution;
- maintain license, subscription, and royalty fees.

On the basis of these six steps, Cechich and Piattini[69] maintained the manner of choosing the COTS component, which consists of five main steps as listed below. These steps are also shown in Figure 10.

1. Define system stability[70] and stakeholder preferences.
2. Assist in the selection process by creating a complete description of the COTS candidates.
3. Select a COTS candidate or a set of COTS candidates from several alternatives. Decisions are made based on previous definitions and measurements.
4. Ensure that the selected candidates can be integrated and supported within the required quality.
5. In this phase, a successful COTS selection procedure should communicate its practices and suggest possible corrective/adaptive actions to sustain its success.

However, the existing selection methods do not fully address the specification and evaluation of functional and nonfunctional requirements. It should also be considered that the system requirements can change over time,[64] and they depend on an uncertain, dynamic or changing environment. Moreover, their characterization supposes the use of more than one criterion,[71] and the repository containing the components can vary over time.[72]

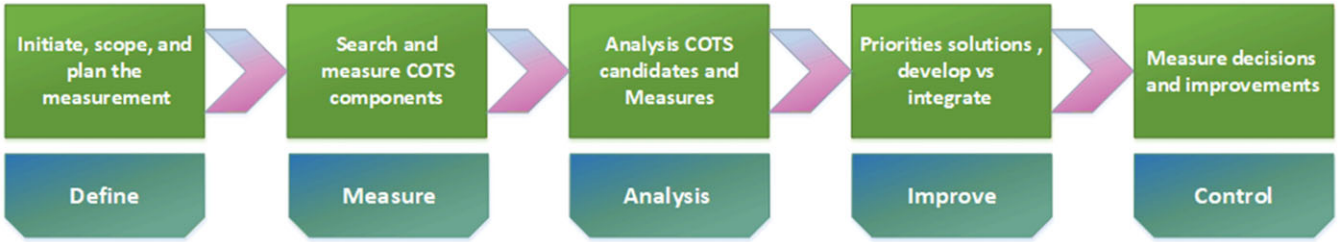

FIGURE 10 Selecting commercial off-the-shelf (COTS) components with Six Sigma (five-step process) [Colour figure can be viewed at wileyonlinelibrary.com]

4.1 | Criteria used in CS

In this section, we answer RQ1 and provide the categorization of the criteria used in CS. One of the component definitions used in the CS process is that a component is an independent software package that provides functionality via well-defined interfaces.[73] The interface may be an export interface through which a component provides functionality to other components or an import interface through which a component gains services from other components.[73] One of the formal definitions of the CS problem from this perspective is as follows: consider SR as the set of final system requirements (the provided functionalities of the final compound component) as SR = R1, R2, R3 and SC as the set of components (repository) available for selection as SC = SC1, SC2, SC3, SC4. The goal is to find a set of components that meets all the requirements of the system. Figure 11 describes the CS problem when it considered in the composition only the provided services (as satisfied requirements in the final system) of a component.

On the basis of this provided definition, one of the most widely used artifacts is system requirements, which are classified into two major classes, namely, functional requirements and nonfunctional requirements. In this respect, functional requirements have the highest application, followed by nonfunctional requirements in second place. The other class is related to the software structure. All the mentioned artifacts and applied references are given in Table 13.

However, the focal point is the criteria established in selecting the components. We have collected the most important applied criteria and their amount of usage in Table 14. As can be seen, the cost stands at a pinnacle of usage. Generally, cost is defined as the overall cost of acquisition and adaptation of that component, but it can include maintenance costs, learning costs, modification costs, etc. Some approaches[11,66,67,74-78] defined formulas for calculating the cost.

Second place belongs to reliability, and for the software to be reliable, it has to be made fault tolerant. Some publications, such as that by Jha et al,[74] used approaches like the consensus recovery block to select components and make fault-tolerant systems at the end. Fault tolerance means that in these systems, for some or all of the software modules, execution can be resumed even after failure with minimal loss of data and time. The number of components takes third place, and it may be seen that it has an important impact on the required solution time. The fourth criterion is delivery time; in general, it is the total delivery time that includes development, integration, and system testing time. Other criteria like development time, functionality, source line of code, etc, follow.

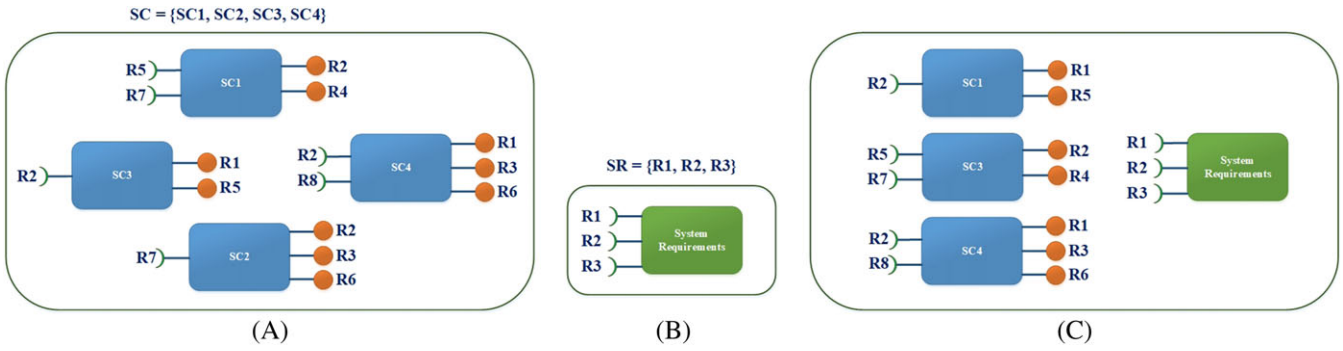

FIGURE 11 Selecting commercial off-the-shelf components with Six Sigma (five-step process) [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 13 Artifacts used by the papers

Artifact	Description	NP	Ref
Functional requirements	Include functional system requirements	49	11,64-67,72-96
Nonfunctional requirements	Include nonfunctional requirements	18	11,65,66,74,75,77-79,81,87,90-94,96,97
Software structure	Provides a set of constraints deriving from how particular applications are built; this includes components and design patterns used, communication and interface standards, and platform characteristics	10	11,65,67,79,81,83,93,95,97,98

Abbreviation: NP, number of papers.

TABLE 14 The main criteria of approaches

Criteria	Ref	Criteria	Ref
Cost	11,64-67,72-79,81,84,85,88,90,92-94,96,97	Compatibility	66,94
Reliability	65-67,74,75,78,81,88,89,91,93,97	Number of required interfaces	91,95
Number of components	11,64,72,73,76,84,86,88,92	Not mentioned	83
Delivery time	66,67,74,78,81,88	Testing time	81
Functionality	66,77-79,85,90	Adaption time	97
Number of modules in the software	65,67,75,78,93	Training time	97
Number of provided interfaces	64,72,86,91,95	Deviational time	97
Number of alternative COTS available	65,75,78,93,95	Testability	89
Coupling	66,90,92	Availability	89
Number of requirements	64,84,86	Maintainability	89
Number of initial requirements not in solution	64,72,86	Complexity	88
Build or buy	74,78,81	Security	89
Execution time	67,74,78	Safety	89
Quality	67,79,97	Performance	89
Development time	66,81,83		11
Source lines of code	67,74,78	Reliability of in-house components	74
Frequency of use of functions	65,75,78	Reliability for both in-house and COTS components	74
Provided services	77,85,95	Reliability of the CRB scheme	74
Required services	77,85,95	Average number of failures	66
Cohesion	11,66	Probability of failures	67
Reusability	77,85		

Abbreviation: COTS, commercial off-the-shelf; CRB, collaborative rateless broadcast.

4.2 | CS research methods

In this section, we answer RQ2 and refer to the classification of the methods employed in selecting the software components presented in Table 15 and Figure 12. As shown in Figure 13, optimization methods (multiobjective, mathematical, and evolutionary algorithms) and a combination of methods (set of methods) apart from the two traditional methods—weighted sum method (WSM) and analytic hierarchy process (AHP)—have more applications over other methods. However, the WSM has many disadvantages, including the summing up of differing types of data (eg, cost plus memory plus quality), the lack of a process for determining attribute weights, and the inherent problem of the formula losing dependency information between attributes (eg, conflicts and corequisites).[98] One of the techniques deployed as an alternative is the AHP, which includes a method for determining weights and component scores against attributes and, as these scores are based on pairwise comparisons, uses the same "units" even when combining qualitative and quantitative data. One of the distinctive features of this method is that it organizes the criteria into a hierarchy and scores can be

TABLE 15 Component selection methods

Method	NP	Ref
Optimization Methods		
Multiobjective Optimization (MOO)		
Evolutionary MOO based on GA	7	64,72,76,84-87
Fuzzy mathematical programming (FMP)	3	67,74,78
Customized MOO (used LINGO optimization model solver)	2	81,82
Goal programming	2	75,96
Lexicographic MOO (LMO)	1	88
Fuzzy MOO	1	66
Mathematical Optimization		
Integer Programming	2	65,89
Lagrange relaxation decomposition	1	100
Evolutionary Algorithms		
Customized GA	1	11
Set of Methods		
Both AHP and WSM	2	79,80
MCDM and multiobjective combinatorial optimization (MOCO)	1	92
Evolutionary and WSM	1	73
AHP and nonlinear optimization	1	93
Multiple-Criteria Decision-Making (MCDM)		
Analytic hierarchy process (AHP)	1	97
Weighted scoring method (WSM)	1	83
Clustering		
Fuzzy clustering	2	77,90
SIREN method[101]	1	83
Artificial Intelligence (AI)		
Neural network	1	98
C4.5	1	98
Customized Algorithms		
Ontology-based	1	91
Customized algorithm based on XML Query Language	1	94
Customized algorithm based on backtracking algorithm	1	95

Abbreviations: GA, genetic algorithm; NP, number of papers.

consistency checked. However, like any methods, the AHP has some demerits, such as the number of pairwise comparisons required, and therefore, it is considered to be time consuming. This, in turn, can be eventuated in the rank-reversal problem, which can be addressed by using a multiplicative formula for aggregation.[16]

In addition, other methods have also been applied, but due to the structure of the CS problem and its similarity to the optimization method, they have been used less than others.

The next type of methods is MOO, which has been utilized more than all other optimization methods. The structure of the CS problem is more similar to the multiobjective problem due to the involvement of different and sometimes contradictory criteria, and in this case, this type of methods has more suitable efficiency in solving this problem. By far, various methods have obtained desirable results by presenting different heuristics and applying MOO methods.

The other methods that have been used in this respect were ad hoc, and they did not become pervasive as shown in Figure 12.

Optimization methods

Regarding the structure of the CS problem and the involvement of various and sometimes contradictory criteria, in the work of Bartholet et al,[99] it is shown that the complexity of the optimal selection of adaptable components varies from polynomial to NP-complete, and even exponential depending on our assumptions. On the basis of these, the use of optimization methods will be the best choice. In the simplest way, in the proposed methods, the CS problem is converted into an optimization problem, which seeks to maximize or minimize one fitness function (or more in MOO methods).

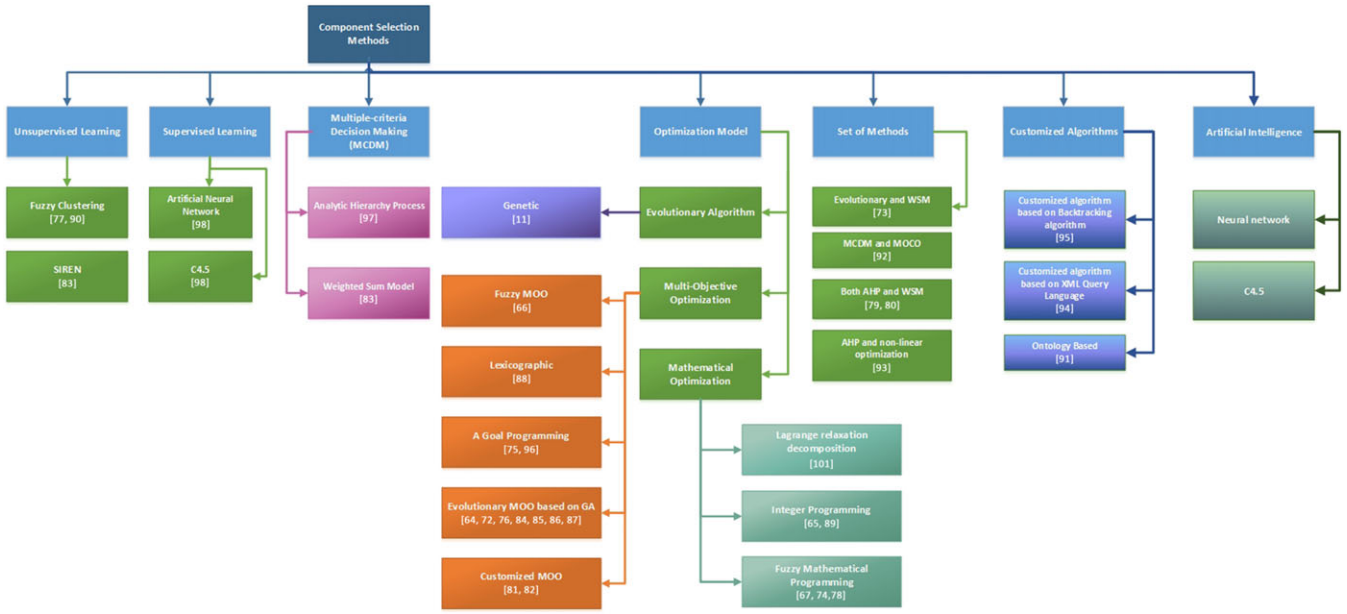

FIGURE 12 Component selection method categories. AHP, analytic hierarchy process; GA, genetic algorithm; MOCO, multiobjective combinatorial optimization; MOO, multiobjective optimization; WSM, weighted scoring method [Colour figure can be viewed at wileyonlinelibrary.com]

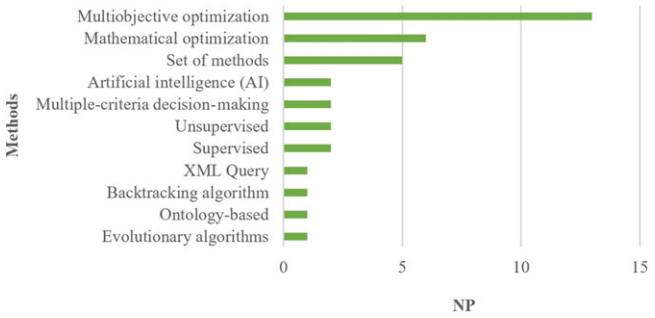

FIGURE 13 Component selection method usages. NP, number of papers [Colour figure can be viewed at wileyonlinelibrary.com]

Set of methods

Some of proposed approaches have used a combination of methods for CS; for example, in the work of Vescan and Grosan,[73] the early stages of CS have benefited from the WSM algorithm and further used evolutionary algorithms.

Multiple-criteria decision-making

Multiple-criteria decision-making or multiple-criteria decision analysis is a subdiscipline of operations research that explicitly evaluates multiple conflicting criteria in decision-making. Conflicting criteria are typical in evaluating options: cost or price is usually one of the main criteria, and some measure of quality is typically another criterion, easily in conflict with cost.

Clustering

In some approaches, clustering algorithms have been used to categorize similar components; for example, in the work of Vescan and Şerban,[90] the fuzzy clustering algorithm has been used to group similar components, and then, by providing a set of requirements, they have selected the best candidate.

Artificial intelligence

In this category, Maxville et al[98] described a technique for training AI classifiers (ANN and c4.5) to assist in the selection of software components for development projects. The authors believe that when using AI, we are able to represent dependencies between attributes, overcoming some of the limitations of existing aggregation-based approaches to CS.

Customized algorithms

In this category, some papers have presented approaches based on various concepts and theories. For example, Yessad and Boufaida[91] presented a method based on ontology with the following goals:

- describing the quality of service (QoS) of software components;
- selecting the relevant components for the developer's QoS requirements.

The authors point out the following QoS concept modeling by ontologies.

- Ontology provides a shared and common description for the QoS and, thus, reduces the semantic heterogeneity between component descriptions.
- It provides reasoning mechanisms on QoS concepts.
- It is extensible, ie, new QoS concepts and properties can be added.

Also, two other approaches[94,95] have provided methods based on XML Query Language and backtracking algorithms, respectively.

4.3 | Representation of artifacts

In this section, we answer RQ3 and provide the representation of artifacts in reviewed papers. These representations are listed in Table 16. Like the component detection problem, in this type of problem, vector also has the most application. It is also due to the greater use of MOO methods and the vector structure to display problem solutions. Then, the graph, XML, and text structure is used to define the requirements.

Vector

Feature vector is commonly used as an artifact representation in most papers on CS. In this representation, a data object is described by a set of features represented as a vector, for example, a set of operational goals or components.

XML Schema

XML Schema is commonly known as XML Schema Definition. It is used to describe and validate the structure and the content of XML data. XML Schema defines the elements, attributes, and data types. An example of this representation is presented in the work of Maxville et al,[98] as shown in Figure 14.

TABLE 16 Representation of artifacts that are used in component selection

Representation	NP	Ref
Feature vector	23	11,64-67,72-78,82,84-87,89,90,92,93,95,96
XML Schema	4	81,91,94,98
Requirements document templates	3	79,80,83
Graph	3	88,97,98

Abbreviation: NP, number of papers.

```
<?xml version="1.0"?>
<Description xmlns=http://www.scis.ecu.edu.au/swvML/1.0/
        xmlns:dc=http://purl.org/dc/elements/1.0
        xmlns:sw=http://www.scis.ecu.edu.au/swvML/1.0/>
        <dc:description type="mandatory">scientific</dc:description>
        <dc:description>calculator</dc:description>
        <swv:devStatus type="mandatory">mature</swv:devStatus>
        <swv:licence type="preferred">GPL </swv:licence >
        <swv:price type="preferred" min="25" max="50">40</swv:price>
        <swv:technical>
                <swv:devLanguage type="preferred">java</swv:devLanguage>
                <swv:devLanguage>c++</swv:devLanguage>
                <dc:operatingSystem type="mandatory">Linux</dc:operatingSystem >
                <dc:systemRequirements>
                        <dc:memory type="preferred" min="15" max="50">20</dc:memory>
                        <dc:diskSpace type="preferred" min="30" max="50">40</dc:diskSpace>
                <dc:systemRequirements>
        </swv:technical>
</Description>
```

FIGURE 14 Component specification in XML Schema format

Requirements document template

Generally, the requirement(s) document or, in other words, user requirement(s) document is a document usually used in software engineering that specifies what the user expects the software to do and must contain all relevant information about the stakeholder's requirements, for example, in the work of Alves and Castro,[80] each part of the document is explained as follows.

Req-id: each requirement has a unique identifier
Type: functional or nonfunctional requirements
Description: detailed information about the requirements
Priority: varies from 1 to 4
4—very high priority (mandatory)
3—high priority (important)
2—medium priority (would be nice)
1—low priority (do not care)

Graph

Some papers represent the problem as a graph, for example, in the work of Khan and Mahmood,[88] a graph $G = (X, E)$ with node set X and edge set E. A node x corresponds to a candidate component, and an interface between two components x and y is represented by an undirected edge (x, y).

4.4 | Results and main features

In this section, we attempt to answer RQ4 and present the main criteria of the CS methods in Appendix A1. According to the study methods and criteria of Appendix A1, among all the presented methods and among all the criteria, optimization methods and cost have been the most widely used. However, what is evident in all of the methods are the lack of study on the presented methods in real systems on one hand and, as in the given literature in the component detection section, as shown in Section 4.1, the fact that each has taken advantage of the different criteria on the other hand; this makes it difficult to evaluate and compare the methods with one another. However, it can be said that MOO methods are still the best option to solve the CS problem, just with this difference that in the future presenting method, the qualitative criteria would be used more than ever. In addition, another issue that has been considered in the literature and which seems to be useful is changing the functional and nonfunctional requirements. Given the fact that it is possible to change these requirements during the production of software and the selection of components and as, in practice, this might happen frequently, paying attention to this issue will help system designers choose the components. Furthermore, in any CS method, it is unrealistic to expect a perfect match between the components needed and the components available. A group of components that compose a system may have overlaps and gaps in the required functionality. A gap represents a lack of functionality; an overlap can cause a confusion in responsibility and degrade nonfunctional properties like size and performance.

4.5 | Case studies

This section answers RQ5, and all the case studies that are used in CS methods are presented in Table 17. Note that most of them describe their case study systems in their papers, and a limited number of real and available systems are employed.

4.6 | Methods for evaluating the obtained results

In this section, we present the methods used to evaluate the given methods in Table 18 and answer RQ6 presented in Section 2.2. Another interesting point is the lack of evaluation of the presented methods in most literature works. Most of these articles have only implemented the proposed method on a system, but they have not performed an evaluation to determine whether this method is efficient and outdoes other methods or not. However, the customized function is the most widely used evaluation method in comparison to others.

4.7 | FUTURE WORK

In this section, we answer RQ7 and provide future works in the field of CS.

Although so many methods have been presented for solving this problem, they have some constraints. For example, one of the limitations of their work is that component compatibilities and integration effort are not considered. In

TABLE 17 Case studies in component selection methods

Case Study	NP	Ref	Details
Custom system 1	13	64,66,74-78,80,85-87,95	All case study details are mentioned in the papers
Not mentioned	12	65,73,83,84,88,89,91-94,96,97	_____
Reservation system	3	37,90	Case study details are mentioned in the papers
ReMap	1	79	102
Mobile application	1	81	103
Hypertext Browser Selection	1	79	104
Mail server	1	82	105
Custom system 2	1	67	106
Garment Industry Financial System	1	11	Case study details are mentioned in the paper
Pharmaceutical Industry Financial System	1	11	Case study details are mentioned in the paper

Abbreviation: NP, number of papers.

TABLE 18 Evaluation methods for used research methods

Method/Objective Function	Ref	Method/Objective Function	Ref
Not mentioned	65,66,73,74,77,78,81,83,92,94-96	Comparison with suppliers/engineer opinion	80,97
Customize Fitness Function/Tools	75,80,84-87	Wilcoxon statistical test	72
Comparison with other approaches	11,64,76,88,90	10-fold cross validation	98
Sensitivity analysis	11,89,93	Precision and recall	91

addition, for a large number of COTS alternatives and evaluation criteria, the decision-making process can be very complex, and numerous possible situations may arise from this complexity. The field of quality attribute determination of a component-based system is extensive; therefore, more research should be performed in this field.

In this review, methods using multiobjective algorithms have increased in recent years. Therefore, the use of multiobjective algorithms is appropriate for this problem. With considering many metrics, many objective algorithms can be used for solving this problem.

In addition, some ideas like using fuzzy theories could be introduced to solve the fuzziness caused by subjective judgments and fuzzy inputs and consideration of uncertainty caused by changing system requirements, evolving of COTS products, requirements, and so on.

Another subject that should be considered is applying these approaches to more case studies and testing them in real systems to better understand their benefits and limitations. None of the mentioned methods provide a fully automatic approach for the whole CS; however, the presentations of semiautomated methods are a good improvement in this field. There is a need for a collaborative process in which both stakeholder requirements and candidate components can trade off stakeholder requirements and limitations of available components. These approaches cannot investigate a formal methodology for determining the relative weights to be assigned to the different quality metrics based on stakeholders' data, and it can be dealt with in the future.

5 | CONCLUSION

In this work, we have presented a systematic review of CI and CS. In this sense, specific aspects of the approaches were revealed, and some trends and opportunities were identified to guide future research studies in these fields. That way, researchers can identify the common characteristics of the approaches and directions for future works.

On the basis of the methods studied in CS, among all the presented methods, optimization methods have been the most widely used; however, MOO methods are still the best option to solve the CS problem just with this difference that in the future presenting method, the qualitative criteria would be used more than ever, but until now, cost has been the most widely used criteria.

According to the study of CI methods, clustering and evolutionary approaches have been the most widely used. However, using an evolutionary search algorithm leads to an increase in complexity, but some ideas as using multiobjective

algorithms is appropriate to the structure of this problem. In the end, it must be noticed that not all of these methods could obtain the ideal result; therefore, an accurate evaluation is required.

ORCID

Seyed Mohammad Hossein Hasheminejad http://orcid.org/0000-0002-7357-7906

REFERENCES

1. McIlroy MD. Mass-produced software components. In: Proceedings of the NATO Conference on Software Engineering; 1968; Garmisch, Germany.
2. Hasheminejad SMH, Jalili S. SCI-GA: software component identification using genetic algorithm. *J Object Technol*. 2013;12(2):1-34.
3. Räihä O. A survey on search-based software design. *Comput Sci Rev*. 2010;4(4):203-249.
4. Gill NS. Importance of software component characterization for better software reusability. *ACM SIGSOFT Softw Eng Notes*. 2006;31(1):1-3.
5. Capretz LF, Capretz MAM, Li D. Component-based software development. Paper presented at: 27th Annual Conference of the IEEE Industrial Electronics Society (IECON) - Vol. 3; 2001; Denver, CO.
6. Mahmood S, Lai R, Kim YS. Survey of component-based software development. *IET Softw*. 2007;1(2):57-66.
7. Szyperski C. Emerging component software technologies—a strategic comparison. *Softw Concepts Tools*. 1998;19(1):2-10.
8. Baster G, Konana P, Scott JE. Business components: a case study of bankers trust Australia limited. *Commun ACM*. 2001;44(5):92-98.
9. Hashemi SH, Shahmohammadi GR. Detection system software components using a hybrid algorithm. *ANDRIAS J*. 2015;40(2):57-63.
10. Object Management Group (OMG). Unified Modeling Language Specification, Version 2. 2005. http://www.omg.org/spec/UML/2.0/
11. Tang JF, Mu LF, Kwong CK, Luo XG. An optimization model for software component selection under multiple applications development. *Eur J Oper Res*. 2011;212(2):301-311.
12. Vitharana P, Zahedi F, Jain HK. Design, retrieval, and assembly in component-based software development. *Commun ACM*. 2003;46(11):97-102.
13. Armstrong E, Ball J, Bodoff S, et al. The J2EE 1.4 tutorial. Sun Microsystems; 2005.
14. Kitchenham B, Charters S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report; 2007.
15. Bennett K. Legacy systems: coping with success. *IEEE Softw*. 1995;12(1):19-23.
16. Triantaphyllou E. *Multi-Criteria Decision Making Methods: A Comparative Study*. Dordrecht, The Netherlands: Springer Science+Business Media Dordrecht; 2000. *Applied Optimization*; vol. 44.
17. Kim H-K, Chung Y-K. Transforming a legacy system into components. In: Gavrilova ML, Gervasi O, Kumar V, et al, eds. *Computational Science and Its Applications - ICCSA 2006: International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part III*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2006:198-205. *Lecture Notes in Computer Science*; vol. 3982.
18. Mitchell BS, Mancoridis S. On the automatic modularization of software systems using the bunch tool. *IEEE Trans Softw Eng*. 2006;32(3):193-208.
19. Cui JF, Chae HS. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Inf Softw Technol*. 2011;53(6):601-614.
20. Mishra SK, Kushwaha DS, Misra AK. Creating reusable software component from object-oriented legacy system through reverse engineering. *J Object Technol*. 2009;8(5):133-152.
21. Allier S, Sahraoui HA, Sadou S. Identifying components in object-oriented programs using dynamic analysis and clustering. In: Martin P, Kark AW, Stewart DA, eds. *Proceedings of the 2009 Conference of the Centre for Advanced Studies on Collaborative Research, November 2-5, 2009, Toronto, Ontario, Canada*. New York, NY: Association for Computing Machinery; 2009:136-148.
22. Kim SD, Chang SH. A systematic method to identify software components. Paper presented at: 11th Asia-Pacific Software Engineering Conference (APSEC); 2004; Busan, South Korea.
23. Lee JK, Seung SJ, Kim SD, Hyun WH, Ham DH. Component identification method with coupling and cohesion. In: Proceedings Eighth Asia-Pacific Software Engineering Conference; 2001; Macao, China.
24. Lee E, Lee B, Shin W, Wu C. Toward component-based system: using static metrics and relationships in object-oriented system. In: Ramamoorthy CV, Lee R, Lee KW, eds. *Software Engineering Research and Applications: First International Conference, SERA 2003, San Francisco, CA, USA, June 25-27, 2003, Selected Revised Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2004:87-101.
25. Shahmohammadi G, Jalili S, Hasheminejad SMH. Identification of system software components using clustering approach. *J Object Technol*. 2010;9(6):77-98.
26. Cai Z-G, Yang X-H, Wang X-Y, Kavs AJ. A fuzzy formal concept analysis based approach for business component identification. *J Zhejiang Univ SCI C*. 2011;12(9):707-720.
27. Hasheminejad SMH, Jalili S. An evolutionary approach to identify logical components. *J Syst Softw*. 2014;96(1):24-50.
28. Hasheminejad SMH, Jalili S. CCIC: clustering analysis classes to identify software components. *Inf Softw Technol*. 2015;57(13):329-351.

29. Hasheminejad SMH, Gholamshahi S. PCI-PSO: preference-based component identification using particle swarm optimization. *J Intell Syst*. 2018.
30. Jain HK, Chalimeda N, Ivaturi N, Reddy B. Business component identification - a formal approach. In: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing (EDOC); 2001; Seattle, WA.
31. Jang Y-J, Kim E-Y, Lee K-W. Object-oriented component identification method using the affinity analysis technique. In: Konstantas D, Léonard M, Pigneur Y, Patel S, eds. *Object-Oriented Information Systems: 9th International Conference, OOIS 2003, Geneva, Switzerland, September 2003. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2003:317-321. *Lecture Notes in Computer Science*; vol 2817.
32. Peng L, Tong Z, Zhang Y. Design of business component identification method with graph segmentation. In: Proceedings of the 2008 3rd International Conference on Intelligent System and Knowledge Engineering; 2008; Xiamen, China.
33. Fan-Chao M, Den-Chen Z, Xiao-Fei X. Business component identification of enterprise information system: a hierarchical clustering method. In: Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE); 2005; Beijing, China.
34. Albani A, Overhage S, Birkmeier D. Towards a systematic method for identifying business components. In: Chaudron MRV, Szyperski CA, Reussner RH, eds. *Component-Based Software Engineering: 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:262-277. *Lecture Notes in Computer Science*; vol. 5282.
35. Fanchao M, Dechen Z, Xiaofei X. Identifying business components from business model - a method based on feature matching. Paper presented at: International Conference e-Management and Business Intelligence; 2007.
36. Constantinou E, Naskos A, Kakarontzas G, Stamelos I. Extracting reusable components: a semi-automated approach for complex structures. *Inf Process Lett*. 2015;115(3):414-417.
37. Prajapati A, Chhabra JK. Harmony search based remodularization for object-oriented software systems. *Comput Lang Syst Struct*. 2017;47(pt 2):153-169.
38. Kebir S, Seriai A-D, Chardigny S, Chaoui A. Quality-centric approach for software component identification from object-oriented code. In: Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA-ECSA); 2012; Helsinki, Finland.
39. Kebir S, Seriai A-D, Chaoui A, Chardigny S. Comparing and combining genetic and clustering algorithms for software component identification from object-oriented code. In: Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering (C3S2E); 2012; Montreal, Canada.
40. Falamarzi A, Ahmadzadeh M, Sayadi H, Shahabi J. Identify the components of an application using neural networks. Paper presented at: Conference on Management and ICT (Information and Communication Technology); 2016.
41. Lee W-J, Kwon O-C, Kim M-J, Shin G-S. A method and tool for identifying domain components using object usage information. *ETRI J*. 2003;25(2):121-132.
42. Choi M, Cho E. Component identification methods applying method call types between classes. *J Inf Sci Eng*. 2006;22(2):247-267.
43. Lee SD, Yang YJ, Cho FS, Kim SD, Rhew SY. COMO: a UML-based component development methodology. In: Proceedings Sixth Asia Pacific Software Engineering Conference (ASPEC); 1999; Takamatsu, Japan.
44. Garg RM, Dahiya D. An aspect oriented component based model driven development. In: Zain JM, Mohd WMBW, El-Qawasmeh E, eds. *Software Engineering and Computer Systems: Second International Conference, ICSECS 2011, Kuantan, Pahang, Malaysia, June 27-29, 2011, Proceedings, Part III*. Berlin, Germany: Springer-Verlag GmbH Berlin Heidelberg; 2011:502-517. *Communications in Computer and Information Science*; vol. 181.
45. Hamza HS. A framework for identifying reusable software components using formal concept analysis. In: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations; 2009; Las Vegas, NV.
46. Ramírez A, Romero JR, Ventura S. A novel component identification approach using evolutionary programming. In: Blum C, ed. *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. New York, NY: Association for Computing Machinery; 2013:209-210.
47. Ramírez A, Romero JR, Ventura S. An approach for the evolutionary discovery of software architectures. *Inf Sci*. 2015;305:234-255.
48. Ahmadzadeh M, Shahmohammadi GR, Shayesteh M. Identification of software systems components using a self-organizing map competitive artificial neural network based on cohesion and coupling. *ANDRIAS J*. 2016;40(3):642-651.
49. Ganesan R, Sengupta S. O2BC: a technique for the design of component-based applications. In: Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS); 2001; Santa Barbara, CA.
50. Kumar S, Bhatia RK, Kumar R. K-means clustering of use-cases using MDL. In: Krishna PV, Babu MR, Ariwa E, eds. *Global Trends in Information Systems and Software Applications: 4th International Conference, ObCom 2011, Vellore, TN, India, December 9-11, 2011. Proceedings, Part II*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:57-67.
51. Boussaid I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. *Inf Sci*. 2013;237:82-117.
52. Whitley D. An overview of evolutionary algorithms: practical issues and common pitfalls. *Inf Softw Technol*. 2001;43(14):817-831.
53. Harman M, Jones BF. Search-based software engineering. *Inf Softw Technol*. 2001;43(14):833-839.
54. Xanthakis S, Ellis C, Skourlas C, Le Gall A, Katsikas S, Karapoulios K. Application of genetic algorithms to software testing. In: Proceedings of 5th International Conference on Software Engineering and its Applications; 1992; Toulouse, France.
55. Zäpfel G, Braune R, Bögl M. *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2010.
56. Rodrigues NF, Barbosa LS. Component identification through program slicing. *Electron Notes Theor Comput Sci*. 2006;160:291-304.

57. Stumme G. Formal concept analysis. *Handbook on Ontologies*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2009:177-199. *International Handbooks on Information Systems*.
58. Tilley TA, Cole RJ, Becker P, Eklund PW. A survey of formal concept analysis support for software engineering activities. *Formal Concept Analysis: Foundations and Applications*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:250-271. *Lecture Notes in Computer Science*; vol. 3626.
59. Nourine L, Raynaud O. A fast algorithm for building lattices. *Inf Process Lett*. 1999;71(5-6):199-204.
60. Godin R, Missaoui R, Alaoui H. Incremental concept formation algorithms based on Galois (concept) lattices. *Comput Intell*. 1995;11(2):246-267.
61. Ham D-H, Kim J-S, Cho J-H, Ha S-J. MaRMI-III: a methodology for component-based development. *ETRI J*. 2004;26(2):167-180.
62. Kim T-Y, Kim Y-K, Chae H-S. An experimental study of generality of software defects prediction models based on object oriented metrics. *KIPS Trans PartD*. 2009;16(3):407-416.
63. Park JG, Chae HS, So ES. A dynamic load balancing approach based on the standard RFID middleware architecture. In: Cheung SC, Li Y, Chao K-M, Younas M, Chung J-Y, eds. *Proceedings of ICEBE 2007, IEEE International Conference on e-Business Engineering and the Workshops SOAIC 2007, SOSE 2007, SOKM 2007, 24-26 October, 2007, Hong Kong, China*. Washington, DC: IEEE Computer Society; 2007:337-340.
64. Vescan A. Case study method and research design for the dynamic multilevel component selection problem. In: Norta A, Gaaloul W, Gangadharan GR, Dam HK, eds. *Service-Oriented Computing - ICSOC 2015 Workshops WESOA, RMSOC, ISC, DISCO, WESE, BSCI, FOR-MOVES, Goa, India, November 16-19, 2015, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2015:130-141. *Lecture Notes in Computer Science*; vol. 9586.
65. Jha PC, Kapur PK, Bali S, Kumar UD. Optimal component selection of COTS based software system under consensus recovery block scheme incorporating execution time. *Int J Reliab Qual Saf Eng*. 2010;17(3):209-222.
66. Jha PC, Bali V, Narula S, Kalra M. Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme. *J Comput Sci*. 2014;5(2):233-242.
67. Gupta P, Mehlawat MK, Verma S. COTS selection using fuzzy interactive approach. *Optim Lett*. 2012;6(2):273-289.
68. Cechich A. Challenges setting a process to manage COTS component selection. In: Proceedings of 26th International Conference on Software Engineering - W7S Workshop International Workshop on Models and Processes for the Evaluation of COTS Components; 2004; Edinburgh, UK.
69. Cechich A, Piattini M. Managing COTS components using a six sigma-based process. In: *Product Focused Software Process Improvement: 5th International Conference, PROFES 2004, Kansai Science City, Japan, April 5-8, 2004. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2004:553-567.
70. Cechich A, Piattini M. Defining stability for component integration assessment. In: Proceedings of the International Conference on Enterprise Information Systems; 2003; Angers, France.
71. Vescan A, Grosan C. A hybrid evolutionary multi-objective approach for the component selection problem. In: *Hybrid Artificial Intelligence Systems: Third International Workshop, HAIS 2008, Burgos, Spain, September 24-26, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:164-171. *Lecture Notes in Computer Science*; vol. 5271.
72. Vescan A. An evolutionary multiobjective approach for the dynamic multilevel component selection problem. In: Norta A, Gaaloul W, Gangadharan GR, Dam HK, eds. *Service-Oriented Computing - ICSOC 2015 Workshops WESOA, RMSOC, ISC, DISCO, WESE, BSCI, FOR-MOVES, Goa, India, November 16-19, 2015, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2016:193-204.
73. Vescan A, Grosan C. A hybrid evolutionary multiobjective approach for the component selection problem. In: *Hybrid Artificial Intelligence Systems: Third International Workshop, HAIS 2008, Burgos, Spain, September 24-26, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:164-171.
74. Jha PC, Kaur R, Bali S, Madan S. Optimal component selection approach for fault-tolerant software system under CRB incorporating build-or-buy decision. *Int J Reliab Qual Saf Eng*. 2014;20(6):273-289.
75. Kumar D, Jha PC, Kapur PK, Kumar UD. Optimal component selection problem for COTS based software system under consensus recovery block scheme: a goal programming approach. *Int J Comput Appl*. 2012;47(4):9-14.
76. Vescan A. Pareto dominance-based approach for the component selection problem. In: Proceedings of the 2008 Second UKSIM European Symposium on Computer Modeling and Simulation (EMS); 2008; Liverpool, UK.
77. Şerban C, Vescan A, Pop HF. A new component selection algorithm based on metrics and fuzzy clustering analysis. In: Corchado E, Wu X, Oja E, Herrero Ã, Baruque Bruno, eds. *Hybrid Artificial Intelligence Systems: 4th International Conference, HAIS 2009, Salamanca, Spain, June 10-12, 2009. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2009:621-628. *Lecture Notes in Computer Science*; vol. 5572.
78. Kaura R, Arora S, Jhab PC, Madan S. Fuzzy multi-criteria approach for component selection of fault tolerant software system under consensus recovery block scheme. *Procedia Comput Sci*. 2015;45:842-851.
79. Kontio J. *OTSO: A Systematic Process for Reusable Software Component Selection*. Technical Report. College Park, MD: University of Maryland at College Park; 1995.
80. Alves C, Castro J. CRE: a systematic method for COTS components selection. In: Proceedings of the XV Brazilian Symposium on Software Engineering (SBES); 2001; Rio de Janeiro, Brazil.
81. Cortellessa V, Marinelli F, Potena P. Automated selection of software components based on cost/reliability tradeoff. In: Gruhn V, Oquendo F, eds. *Software Architecture: Third European Workshop, EWSA 2006, Nantes, France, September 4-5, 2006, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2006:66-81. *Lecture Notes in Computer Science*; vol. 4344.

82. Cortellessa V, Crnkovic I, Marinelli F, Potena P. Experimenting the automated selection of COTS components based on cost and system requirements. *J Univers Comput Sci*. 2008;14(8):1228-1255.
83. Martínez MA, Álvarez JAT. COTSRE: a components selection method based on requirements engineering. Paper presented at: Seventh International Conference on Composition-Based Software Systems (ICCBSS); 2008; Madrid, Spain.
84. Vescan A, Grosan C. Two evolutionary multiobjective approaches for the component selection problem. Paper presented at: 2008 Eighth International Conference on Intelligent Systems Design and Applications; 2008; Kaohsiung, Taiwan.
85. Vescan A. A metrics-based evolutionary approach for the component selection problem. In: Proceedings of the UKSim 2009 11th International Conference on Computer Modelling and Simulation (UKSIM); 2009; Cambridge, UK.
86. Vescan A, Grosan C. A hybrid evolutionary multiobjective approach for the component selection problem. In: *Hybrid Artificial Intelligence Systems: Third International Workshop, HAIS 2008, Burgos, Spain, September 24-26, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:164-171. *Lecture Notes in Computer Science*; vol. 5271.
87. Vescan A, Grosan C. Evolutionary multiobjective approach for multilevel component composition. *Stud Univ Babes Bolyai Ser Informatica*. 2010;55(4):18-32.
88. Khan MA, Mahmood S. Optimal component selection for component-based systems. In: Sobh Tarek, Elleithy Khaled, eds. *Innovations in Computing Sciences and Software Engineering*. Dordrecht, The Netherlands: Springer Science+Business Media BV; 2010:467-472.
89. Pande J, Garcia CJ, Pant D. Optimal component selection for component based software development using pliability metric. *ACM SIGSOFT Softw Eng Notes*. 2013;38(1):1-6.
90. Vescan A, Şerban C. A fuzzy-based approach for the multilevel component selection problem. In: Martínez-Álvarez F, Troncoso A, Quintián H, Corchado E, eds. *Hybrid Artificial Intelligent Systems: 11th International Conference, HAIS 2016, Seville, Spain, April 18-20, 2016, Proceedings*. Cham, Switzerland: Springer International Publishing Switzerland; 2016:463-474. *Lecture Notes in Computer Science*; vol. 9648.
91. Yessad L, Boufaida Z. A QoS ontology-based component selection. *Int J Soft Comput*. 2011;2(3):16-30.
92. Neubauer T, Stummer C. Interactive decision support for multiobjective COTS selection. Paper presented at: 2007 40th Annual Hawaii International Conference on System Sciences (HICSS); 2007; Waikoloa, HI.
93. Verma S, Mehlawat MK. Multi-criteria optimization model integrated with AHP for evaluation and selection of COTS components. *Optimization*. 2017;66(11):1879-1894.
94. Boonsiri S, Seacord RC, Bunting R. Automated component ensemble evaluation. *Int J Inf Technol*. 2002;8(1):40-53.
95. Iribarne L, Troya JM, Vallecillo A. Selecting software components with multiple interfaces. In: Proceedings of the 28th Euromicro Conference; 2002; Dortmund, Germany.
96. Cortellessa V, Crnkovic I, Marinelli F, Potena P. Driving the selection of COTS components on the basis of system requirements. In: Stirewalt REK, Egyed A, Fischer B, eds. *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. New York, NY: Association for Computing Machinery; 2007:413-416.
97. Lozano-Tello A, Goméz-Pérez A. BAREMO: how to choose the appropriate software component using the analytic hierarchy process. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE); 2002; Ischia, Italy.
98. Maxville V, Armarego J, Lam CP. Intelligent component selection. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC); 2004; Hong Kong, China.
99. Bartholet RG, Brogan DC, Reynolds Jr PF. The computational complexity of component selection in simulation reuse. In: Proceedings of the 37th Conference on Winter Simulation; 2005; Orlando, FL.
100. Zhiqiao W, Kwong CK, Tang J, Chan JWK. Integrated model for software component selection with simultaneous consideration of implementation and verification. *Comput Oper Res*. 2012;39(12):3376-3393.
101. Toval A, Nicolás J, Moros B, Garciá F. Requirements reuse for improving information systems security: a practitioner's approach. *Requir Eng*. 2002;6(4):205-219.
102. Kontio J, Limperos K, Tesoriero R, Caldiera G. ReMap: Evaluation criteria definition. 1995.
103. Perkins C. IP mobility support for IPv4. IETF. 2002.
104. Kontio J. A case study in applying a systematic method for cots selection. In: *Proceedings of the 18th International Conference on Software Engineering*. Washington, DC: IEEE Computer Society Press; 1996:201-209.
105. Alves CF, Franch X, Carvallo JP, Finkelstein A. Using goals and quality models to support the matching analysis during COTS selection. In: *COTS-Based Software Systems: 4th International Conference, ICCBSS 2005, Bilbao, Spain, February 7-11, 2005. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:146-156.
106. Jung H-W, Choi B. Optimization models for quality and cost of modular software systems. *Eur J Oper Res*. 1999;112(3):613-619.

How to cite this article: Gholamshahi S, Hasheminejad SMH. Software component identification and selection: A research review. *Softw: Pract Exper*. 2018;1–30. https://doi.org/10.1002/spe.2656

APPENDIX

TABLE A1 Component identification (CI) and component selection (CS) methods and metrics

Ref	Type	Artifact	Method	Metrics
[48]	CI	- Use cases	Neural networks	- Cohesion - Comparison with other approaches
[28]	CI	- Analysis class diagram - Use cases	Genetic algorithm	- Comparison with other approaches - Comparison with expert opinion - Variance ratio criterion - Sum of squared errors - Time - Recall - Cost - Precision - Davies-Bouldin index
[9]	CI	- Use cases	Set of methods	- Comparison with other approaches - Cohesion and coupling - Comparison with expert opinion - Ray-Turi index - CH index
[47]	CI	- Class diagram	Genetic algorithm	- Customize Fitness Function/Tools - Cohesion and coupling - Groups/components ratio - Time
[27]	CI	- Use cases - Analysis class diagram	Genetic algorithm	- Stability index - Coupling index - Number of different use cases - Comparison with other approaches - Comparison with expert opinion - Customize Fitness Function - Cohesion and coupling
[2]	CI	- Use cases - Analysis class diagram	Genetic algorithm	- Comparison with other approaches - Comparison with expert opinion - Customize Fitness Function/Tools - Complexity - Cohesion and coupling
[39]	CI	- Source code	Set of methods	- Average error per criterion function - Cohesion and coupling - Comparison with expert opinion - CH index
[25]	CI	- Class diagrams - Use cases - Collaboration diagram	Set of methods	- Average error per criterion function - Cohesion and coupling - Comparison with expert opinion - CH index
[26]	CI	- Business model documents - Collaboration diagram	Formal concept analysis	- Precision - Cohesion and coupling
[31]	CI	- Business model documents - Use cases	Affinity analysis technique	- Cohesion and coupling
[21]	CI	- Source code - Use cases	Graph-based clustering	- Cohesion and coupling
[34]	CI	- Business model documents	Graph partitioning	- Cohesion and coupling
[32]	CI	- Business model documents	Graph partitioning	- Cohesion and coupling - Customize Fitness Function/Tools
[35]	CI	- Business model documents	Hierarchical clustering	- Precision
[42]	CI	- Class diagrams - Sequence diagram - Use cases	CRUD-based	- Comparison with expert opinion - Comparison with other approaches - Cohesion and coupling

(Continues)

TABLE A1 (Continued)

Ref	Type	Artifact	Method	Metrics
[22]	CI	- Class diagrams - Sequence diagram - Use cases - Requirements definition	Customized clustering Algorithm	- Cohesion and coupling - Comparison with other approaches
[41]	CI	- Class diagrams - Sequence diagram - Use cases - Class diagrams	Seed algorithm Cohesion algorithm	- Cohesion and coupling
[23]	CI	- Sequence diagram - Use cases - Collaboration diagram - Relationship between classes	Customized clustering Algorithm	- Cohesion and coupling - Comparison with expert opinion
[30]	CI	- Sequence diagram - Business model documents - Use cases	Hierarchical clustering	- Cohesion and coupling - Comparison with expert opinion - Complexity
[86]	CS	- Functional requirements	Evolutionary MOO based on GA	- Number of initial requirements - Number of requirements - Number of provided interfaces - Number of components
[89]	CS	- Functional requirements	Integer programming	- Security - Safety - Performance - Testability - Availability - Maintainability
[74]	CS	- Nonfunctional requirements - Functional requirements	Fuzzy mathematical Programming	- Source lines of code - Reliability of in-house components - Reliability for both in-house and COTS components - Reliability of the CRB scheme - Execution time - Delivery time - Cost - Reliability
[88]	CS	- Functional requirements	Lexicographic MOO	- Cost - Reliability - Number of components - Delivery time - Complexity
[72]	CS	- Functional requirements	Evolutionary MOO based on GA	- Cost - Number of components - Number of provided interfaces - Number of initial requirements
[77]	CS	- Nonfunctional requirements - Functional requirements	Fuzzy clustering	- Provided services - Required services - Cost
[78]	CS	- Nonfunctional requirements - Functional requirements	Fuzzy mathematical Programming	- Source lines of code - Frequency of use of functions - Build or buy - Execution time - Number of alternative COTS available - Functionality - Number of modules in the software - Reliability - Cost

(Continues)

TABLE A1 (Continued)

Ref	Type	Artifact	Method	Metrics
[66]	CS	- Nonfunctional requirements - Functional requirements	Fuzzy MOO	- Reliability - Delivery time - Functionality - Compatibility - Cost - Cohesion and coupling - Development time
[65]	CS	- Nonfunctional requirements - Functional requirements - Software structure	Integer programming	- Reliability - Number of modules in the software - Cost - Number of alternative COTS available - Frequency of use of functions
[74]	CS	- Nonfunctional requirements - Functional requirements - Software structure	AHP and WSM	- Cost - Functionality

Abbreviations: AHP, analytic hierarchy process; CH, Calinski-Harabasz; COTS, commercial off-the-shelf; CRB, collaborative rateless broadcast; CRUD, create, read, update, delete; GA, genetic algorithm; MOO, multiobjective optimization; WSM, weighted scoring method.