A Clustering-Based Approach for Tracing Object-Oriented Design to Requirement

Xin Zhou[1] and Hui Yu[2]

[1] IBM China Research Lab, China
zhouxin@cn.ibm.com
[2] Peking University, China
yuhui04@sei.pku.edu.cn

Abstract. Capturing the traceability relationship between software requirement and design allows developers to check whether the design meets the requirement and to analyze the impact of requirement changes on the design. This paper presents an approach for identifying the classes in object-oriented software design that realizes a given use case, which leverages ideas and technologies from Information Retrieval (IR) and Text Clustering area. First, we represent the use case and all classes as vectors in a vector space constructed with the keywords coming from them. Then, the classes are clustered based on their semantic relevance and the cluster most related to the use case is identified. Finally, we supplement the raw cluster by analyzing structural relationships among classes. We conduct an experiment by using this clustering-based approach to a system – Resource Management Software. We calculate and compare the precision and recall of our approach and non-clustering approaches, and get promising results.

Keywords: Object-oriented software development, Requirement Traceability, Use Case, Class, Clustering.

1 Introduction

Keeping the traceability between software requirements and other artifacts generated by each software development phase is well recognized as significant in multiple areas, including software maintenance, software evolving and software reuse [1][2]. Such traceability can be obtained from two approaches: one is to record the tracing relationships during the development duration by developers. The other one is to identify the un-recorded tracing relationships by analyzing the requirements and the artifacts.

In current object-oriented software development, use case [3] is widely used to model the user requirement, and class model [4] is the dominant means used to describe the static aspect of software design that realizes requirement. If the corresponding relationships between elements in class model and use cases are well understood, the designers can more easily analyze how requirements have been satisfied by their design, can more precisely locate to and modify the design elements impacted by changing requirements, and so forth. Although designers can accurately record and maintain such relationships while designing, they are usually reluctant to do this due to the heavy overload brought to them.

The paper addresses the problem of object-oriented design to requirement traceability by automatically identifying the potential tracing relationships. As the designers only need to verify the generated potential tracing relationships, they will not feel much troubled. Our approach leverages the ideas and technologies from Information Retrieval (IR)[5][6] and Text Clustering[7][8] area. A premise of our work is that system analyzers and designers use meaningful identifiers and descriptions in use cases and class model so as to describe their semantic exactly. According to the approach, the use case and the classes are represented as vectors in a vector space constructed with the identifiers coming from them. Then, the semantic relevance among class vectors is calculated, and those closely related classes identified are grouped into clusters. The centroid vector of each cluster is generated and its similarity with the use case vector is evaluated so that the cluster most closely related to the use case is identified. Considering some classes might be missed by the clustering process, we finally supplement some classes that have generalization or association relationships to the selected cluster.

The following part of this paper is organized as below: we illustrate our tracing approach in section 2. In section 3, we experiment the approach with a case. Related works is introduced in section 4. Section 5 concludes and forecasts future works.

2 Approach Description

Given one use case UC and a set of classes $\{C_1, C_2, \ldots, C_n\}$, the approach for identifying the classes that realize UC out of the whole class set includes four major steps. As shown in Figure 1, the first step is to extract keywords from the use case and classes, use these keywords to construct a vector space and represent the use case and each class as vectors in the vector space. This step is a prerequisite for further clustering and matching. The second step is to build class clusters according to their

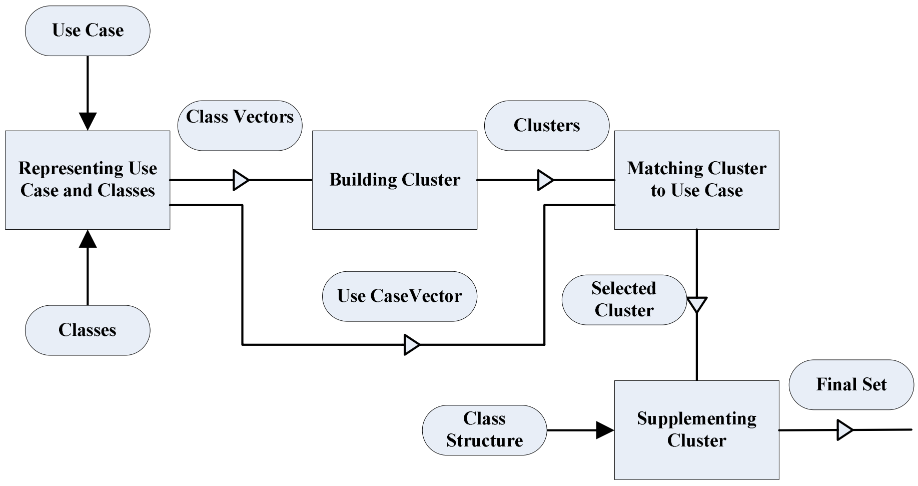


Fig. 1. Approach Overview

semantic relevance since we believe that classes collaboratively realizing the feature described by a use case should have closer semantic relevance. The third step is to generate the centroid vector that summarizes and characterizes a cluster and evaluate its similarity with the use case vector. The cluster with the highest similarity is the post possible one that realizes such use case. As we might have missed some actually relevant classes while constructing the cluster, in the final step we supplement the raw cluster by analyzing generalization and association relationships among all classes. We will introduce each step in detail in the following sub-sections.

2.1 Representing Use Case and Classes

Representing the use case and all classes in vector format is for the convenience of further clustering and matching, which consists of three activities including keyword extraction, vector space building and vector generation.

We take the names and description of the use case, classes, class attributes, and class methods as the source to extract the keywords that implicate the use case's and classes' semantics. The standard practice in information retrieval should be followed while extracting keywords. Namely, only nouns and verbs are extracted and they should be transformed to the original form (i.e. the single form of nouns and the infinitive form of verbs). After the extraction, we can get two keyword sets: one set includes all of the keywords from the use case, and the other set includes all of the keywords from all classes.

An intuitive way for building the vector space is to use the keywords from the union of use case keyword set and class keyword set to build the vector space. The number of dimensions of the resulting vector space will be scores or even hundreds, which causes large computational cost of clustering and matching. Considering that we only care the common part of the use case's semantic and the classes' semantic while conducting the clustering and matching, the keywords from the intersection of use case keyword set and class keyword set are used instead to build the vector space. This way reduces computational cost by decreasing the overall dimensions of the vector space without losing useful information.

With the built vector space, representing the use case and classes as vectors is simple. If the keyword on the x-th dimension of the vector space is one keyword of the use case, then the x-th dimension of the use case's vector is assigned as "1", otherwise is assigned as "0". Similarly, the vector of each class can be generated.

2.2 Building Cluster

We believe that a use case is realized by a group of classes collaboratively and the classes in a group have higher semantic relevance. So in this step, we group classes into clusters according to their semantic relevance, and the resulting clusters will be further processed in next step so that the one closest to the given use case can be identified.

One straightforward approach for building cluster can be: firstly calculating the semantic relevance between any two classes using the cosine of the angle between their vectors, then creating a graph that takes all classes as its nodes and adding one edge between two classes if the relevance value between them is larger than a given threshold value, finally identifying all the complete sub-graphs that cannot be contained by other complete sub-graphs. The classes in each identified complete sub-graph compose a cluster. However, the exponential time complexity of finding maximal complete sub-graph prevents the approach to be practical. With the increasing graph size, the calculating may exhaust all computation resource.

In this paper, we modify the approach above by changing the relevance calculating criterion and cluster creation method, which reduces the computation complexity of clustering. We firstly choose a constant I ($1 \le I \le N$), which is the number of dimensions that we will consider while calculating the class vector relevance. Given a determined constant I, there are totally (C_N^I) possible combinations of dimensions. For each dimension combination $\{D_{j1}, D_{j2}, \ldots, D_{jI}\}$ ($1 \le j \le C_N^I$), all those classes, whose vector has value '1' on all these I dimensions, are grouped to one cluster. So totally, we can get C_N^I clusters. By merging the same clusters, and removing nested clusters, the candidate class clusters are finally determined for further matching with the given use case.

2.3 Matching Cluster to Use Case

In this step, the cluster that has the closest semantic relevance to the given use case will be found out. We calculate the centroid of a cluster, which summarizes and characterizes all classes in the cluster. So, the semantic relevance between all classes in the cluster and the use case can be gotten by calculating the relevance between the cluster centroid and the use case.

In our approach, we map each class and each use case onto a vector, and each element of the vector corresponds to a key word extracted from the class and use case. We use the simplest method to represent the vector, either 1 if the i-th key word occurs in the class or use case, or 0 otherwise. This representation helps us simplify the process of building cluster. But when we match cluster to use case, we have to consider the frequency of a word in the use case. The frequency of the key word reflects the importance of the word in representing the semantics of the use case.

We use a well known IR metric called *tf-idf* [9]. According to this metric, we derive the vector element of cluster centroid from the term frequency tf_i, and the inverse use case frequency iuf_i (represented as *idf* in IR area). We use tf_i to reflect the importance of the *i*-th key word within a use case description; it is the ratio of the number of the occurrences of the *i*-th key word to the total occurrences that all keywords in the vector space appearing in use case description. And iuf_i is a global weight that reflects the overall value of the *i*-th keyword as an indexing term for the entire collection, and the inverse use case frequency iuf_i is defined as the ratio of total number of use cases to the number of use cases containing the *i*-th key word. For example, considering a very common word like "class", it is likely to be contained in most use case descriptions. It is not import in our retrieval process; sometimes it even makes our retrieval result inaccurate. So a small value of global weight *iuf* should be appropriate.

Given $\{C_1, C_2 \ldots\ldots C_k\}$ are the classes in a cluster, and each C_i is represented by a vector $\{c_{i1}, c_{i2} \ldots\ldots c_{iN}\}$, the centroid G=$\{g_1, g_2 \ldots\ldots g_N\}$ of the cluster can be calculated following the formula given below:

$$g_i = tf_i * \log(iuf_i) * \frac{1}{n}\sum_{j=1}^{n} c_{ij} \qquad (1)$$

And we use a similar way to represent the query vector Q, the vector element qi in Q is:

$$q_i = tf_i * \log(iuf_i) \qquad (2)$$

The semantic relevance between each cluster and the given use case is computed as the cosine of the angle between the cluster centroid vector G and the query vector Q:

$$Revelance(G,Q) = \frac{\sum_{i=1}^{N} g_i q_i}{\sqrt{\sum_{i=1}^{N}(g_i^2) * \sum_{i=1}^{N}(q_i^2)}} \qquad (3)$$

The classes in the cluster with the highest relevance to the given use case are regarded as the most probable realization of the use case.

2.4 Supplementing Cluster

Although we have generated the closest cluster, some actual relevant classes still might be missed from the cluster. This is caused by several reasons. The first one is that semantic information may be lost while keywords are extracted from natural language sentences. The second reason is that description of some use cases or classes may be not complete or accurate enough as expected. The third reason is that our cluster building requires all classes in a cluster to be relevant to each other, which is a strong precondition. Actually, there are some "common" classes in design, such as the ones realizing database accessing, human-computer interaction, and so forth. Although they actually contribute to the realization of some use cases, they might be excluded from the cluster since its semantic relevance to the classes in the identified cluster is very low. In a word, we might lose some potential relevant classes if no extra supplement is conducted.

In order to make our approach generate more accurate result, we use relationships in class diagram to supplement the raw cluster built. These relationships can help us find classes missed due to the insufficiency of the semantic search algorithm, the strong precondition of clustering approach, or lack of information in the use case description. Three rules are followed while supplementing the raw cluster, as stated below:

- If class A is in the raw cluster and class B is the super class of A, then B should be added to the cluster.

- If class A is in the raw cluster and class B is a subordinate class of A, then B should be added to the cluster.
- If class A is in the raw cluster and A has a unidirectional association to class B, then B should be added to the cluster.

The major reason for supplementing cluster at the final step instead of performing it before matching cluster to use case is that supplemented classes, especially the common classes, might dilute the semantic correlation of the original cluster and then cause imprecise cluster matching.

3 A Case Study

We use the approach introduced above to trace the design to requirement of Resource Management Software developed by Electrical Engineering and Computer Science Department of Milwaukee School of Engineering. This software schedules and visualizes the availability of specific resources, like consultants, office spaces, and equipment. The detailed description for each use case and the design is provided by [10], which is the basis for our traceability analysis. Requirement of the software is represented by 26 use cases, and the design consists of 45 classes located in five major packages: View, Controller, Entity, Database and Util. In the case study, we only deal with 11 use cases and 22 classes that closely relate to the business logic.

We respectively follow our clustering-based approach and other existing approach to identify the relative classes for each use case. Then, we carefully understand the software's pertinent documents and manually identify the relative classes for each use case, which is used as the correct answer for comparing the results got by those approaches. Two metrics, Precision and Recall, are used to quantitatively denote an approach's completeness and correctness. For each use case, precision is the ratio of actual relevant classes retrieved a certain approach over all the relevant classes regarded by this approach, and recall is calculated as the ratio of actual relevant classes identified by a certain approach over the total actual relevant classes.

3.1 Experiment Clustering-Based Approach

We experiment our clustering-based approach by assigning the constant I mentioned in sub-section 2.2 as 1 and 2 respectively. Table 1 illustrates the experiment result. The first column lists all the use cases to be traced, the second and third columns list the result of the approach with the constant I assigned as 1, and the fourth and fifth columns list the result of the approach with the constant I assigned as 2. We can find that if the constant I is assigned a higher value, the precision will increase while the recall will decrease. That means we can balance the precision and recall by tuning the value of the constant I. Usually, we prefer a higher recall value to higher precision value, since we believe the time spent on discarding a non-relevant class will be lower than time required for recovering a missed class.

Table 1. Precision and recall of the clustering-based approach

Use Case	Clustering Method *(I = 1)*		Clustering Method *(I = 2)*	
	Precision	*Recall*	*Precision*	*Recall*
Add Client	66.67%	100.00%	80.00%	66.67%
Modify Client	60.00%	100.00%	80.00%	66.67%
Delete Client	85.71%	100.00%	80.00%	66.67%
Add Resource	75.00%	100.00%	100.00%	66.67%
Modify Resource	83.33%	100.00%	100.00%	10.00%
Delete Resource	83.33%	100.00%	100.00%	60.00%
Add Project	63.64%	100.00%	80.00%	57.14%
Modify Project	81.82%	100.00%	87.50%	77.78%
Delete Project	70.00%	100.00%	100.00%	87.50%
Add Resource Allocation	72.72%	72.72%	72.72%	72.72%
Delete Resource Allocation	72.72%	100.00%	72.72%	100.00%

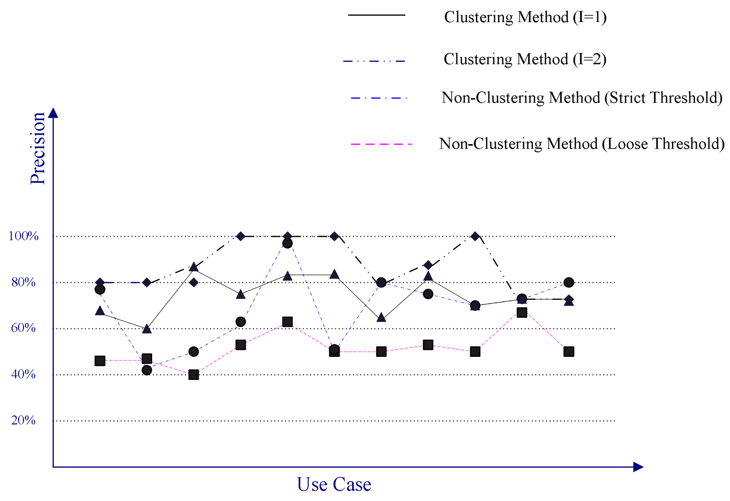
3.2 Comparing Clustering-Based Approach with Non-clustering Approach

IR technologies, especially the probabilistic model [11] and the Vector Space Model (VSM) [12], are widely used in tracing code to requirement documents [13][14]. The basic idea for the VSM-based approaches is: code elements and requirement are represented by corresponding vectors, and the requirement vector is used as a query condition to search relevant code element vectors. The code elements, whose semantic similarity with the query larger than a given threshold, are regarded as the implementation of the requirement. We experiment to use such kind of non-clustering approach to identify relevant classes for use cases in Resource Management Software. The first time, we select a loose threshold so that higher recall can be obtained. The second time, we select a more strict threshold so that higher precision can be obtained. The result is illustrated in Table 2.

As illustrated in Figure 2, we compare the results of clustering-based approach and non-clustering approach. It shows that: if a strict threshold is used, the non-clustering approach's precision values are as good as those of the clustering-based approach, but its recall values are much worse, and if a loose threshold is used, the non-clustering approach's recall values are almost as good as those of the clustering-based approach, but its precision values are much worse. Thus, we can conclude that in this experiment our clustering-based approach wins over the non-clustering approach.

Table 2. Precision and recall of the non-clustering approach

Use Case	Non-Clustering Method (Loose Threshold)		Non-Clustering Method (Strict Threshold)	
	Precision	Recall	Precision	Recall
Add Client	46.15%	100.00%	75.00%	50.00%
Modify Client	46.15%	100.00%	42.86%	50.00%
Delete Client	40.00%	100.00%	50.00%	50.00%
Add Resource	52.94%	100.00%	61.54%	88.89%
Modify Resource	62.50%	50.00%	100.00%	40.00%
Delete Resource	50.00%	100.00%	50.00%	50.00%
Add Project	50.00%	100.00%	80.00%	57.14%
Modify Project	52.94%	100.00%	75.00%	66.67%
Delete Project	50.00%	100.00%	70.00%	100.00%
Add Resource Allocation	68.75%	100.00%	72.72%	72.72%
Delete Resource Allocation	50.00%	100.00%	80.00%	100.00%



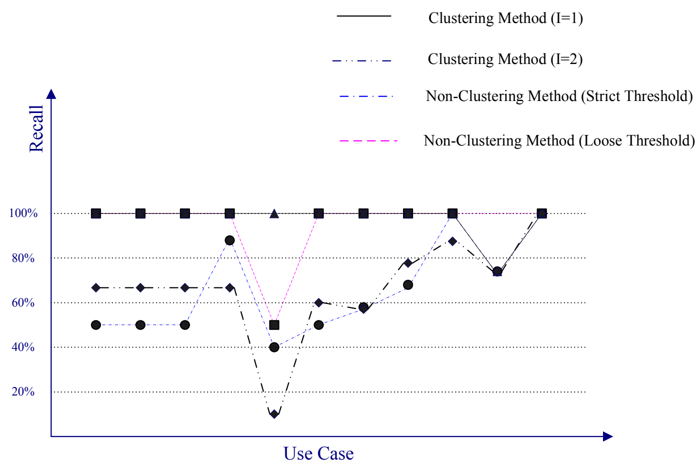


Fig. 2. Comparing clustering-based approach and non-clustering approach

4 Related Work

4.1 IR-Based Traceability Identification

Current literature provides partial solutions to the problem of traceability in various fields of computer science. The proposed solutions are related to areas of Information Retrieval[10][11], Database Management[15], Rule-based System[16][17], Concept-based approach[18], etc.

Some most popular solutions are related to the area of Information Retrieval. Papers [13][14] presented by Antoniol G. introduced a semi-automated process to recover traceability between code and documentation. Their process starts with processing the artifacts, then using a classifier to generate a ranked document list. Their solution uses two IR methods as classifiers. The probabilistic method and the vector space method are used to recover traceability links between code and documentation. Their solution also assumes that the artifacts follow the OO paradigm and the use of common vocabulary between code and documentation.

It's important to note that when we use the approach mentioned above to trace from requirement to classes, it doesn't have high precision and recall. In our approach, presented in section 2, we use a clustering-based approach to get the traceability from requirement to classes. Our work uses the potential semantic relationship between classes to get clusters, which allows our approach to lead to a more accurate result. Moreover, in order to avoid the incompleteness and imprecision of nature language which is used to describe use case, we use a supplementing process after cluster retrieving process based on the IR technology. Thus, our approach has higher precision and recall when tracing from use case to classes.

4.2 Clustering-Based IR Technology

One of the common Information Retrieval (*IR*) technologies is clustering-based approach. In practice of information retrieval, it is not possible to match each analyzed document with each analyzed search request because the time consumed by such operation would be excessive. Various solutions have been proposed to reduce the number of needed comparisons between information items and requests, among which, clustering technique is an effective way to do this, and therefore it is imported to the area of information retrieval.

There is a basic hypothesis in Clustering-based Information Retrieval: closely associated documents tend to be relevant to the same requests. This inspires us to introduce clustering technique into our research. We do clustering by analyzing the inherent semantic relevance between classes. The experiment reveals that such a mechanism brings a higher recall for the traceability process.

5 Conclusions and Future Works

In this paper, we have presented an approach for identifying the group of classes in object-oriented software design that realizes a given requirement represented in a use case. We use the Information Retrieval (*IR*) and text clustering technologies as the basis of our approach to retrieve relevant classes of a given use case. Our premise is that system analysts and designers use common meaningful keywords for use cases, classes, attributes, methods, parameters, etc. We represent the use case and the classes as vectors in a vector space constructed with the keywords coming from them. Then, we cluster the classes based on their semantic relevance and find out the cluster most related to the use case. Finally, we supplement the raw cluster by analyzing generalization, association relationships among all classes. An experimental study is also reported in this paper. This clustering-based approach is used to create traceability between the classes and use case from a system – Resource Management Software. We calculate and compare the precision and recall of our approach and other non-clustering approaches, which reveals that our approach works well on most occasions.

At present, we still cannot claim that we will get similar results on all other systems although it might be quite imaginable. In the future, we will focus on enhancing the supplement process in order to get a more precise result. For instance, we will consider the weight of each inter-class relationship while supplementing a class to a cluster. Also, we will experiments more on other real systems to evaluate and enhance the approach.

References

1. M. Jarke. “Requirements Tracing”. Communications of the ACM, Vol. 41, No. 12, pp. 32-36, December 1998.
2. Pohl, K. PRO-ART: Enabling Requirements Pre- Traceability, In Proceedings of the Second International Conference on Requirements Engineering, IEEE Computer Society Press, 1996 pp. 76-84.

3. I. Jacobson. Object-Oriented Software Engineering. Addison-Wesley Publishing Company, 1992.
4. R. Brooks. Towards a theory of the comprehension of computer programs. International Journal of an-Machine Studies, 18:543–554, 1983.
5. C.J.van Rijsbergen. Information Retrieval. Butterworths, 1979.
6. Frakes, W. and Baeza-Yates, R. Information Retrieval: Data Structures & Algorithms. Prentice Hall, 1992.
7. Bhatia, S.K., and Deogun, J.S. 1998. "Conceptual clustering in information retrieval", IEEE Transactions on Systems, Man and Cybernetics, Part B, 427-436.
8. Kaski, S. 1998. "Dimensionality reduction by random mapping: fast similarity computation for clustering", Proceedings of the 1998 IEEE International Joint Conference on Neural Networks Proceedings, 413-418.
9. G.Salton, C.Buckley. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513–523, 1988.
10. http://people.msoe.edu/~barnicks/courses/cs400/199900/teamrpts.htm
11. KS Jones, S Walkerr, SE Robertson. A probabilistic model of information retrieval: development and status. Information Processing and Management, 2000
12. D.Harman. Ranking algorithms. In Information Retrival: Data Structures and Algorithms, pages 363-392. Prentice-Hall, Englewood Cliffs, NJ, 1992.
13. G. Antoniol, B. Caprile, A. Potrich, and P. Tonella, "Design-Code Traceability for Object Oriented Systems," The Annals of Software Eng., vol. 9, pp. 35-58, 2000.
14. G. Antoniol, G. Canfora, G. Casazza, A. DeLucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," IEEE Transactions on Software Engineering, vol. 28, no. 10, pp. 970-983, Oct.2002.
15. P.K. Garg, W. Scacchi. SODOS: a software documentation support environment-its definition. IEEE Transactions on Software Engineering, August 1986
16. A. Zisman, G. Spanoudakis, E. Perez-Minana, P. Krause. Tracing software requirement artifacts. The 2003 International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA. 2003.
17. C. Nentwich, W. Emmerich, A. Finkelstein, E. Ellmer, Flexible Consistency Checking. ACM Transactions in Software Engineering and Methodology, 12(1), 28-63, 2003.
18. M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, and S. Eherer, TMConceptBase–A Deductive Object Base for Meta Data Management,º Int'l J. Intelligent Information Systems, vol. 5, no. 3, pp. 167-192, 1995.