UDC 004.415.2

DEVELOPMENT OF A CLASS STORAGE REPOSITORY FOR OBJECT-ORIENTED SOFTWARE DEVELOPMENT TECHNOLOGIES

Kungurtsev O. B. – PhD, Professor of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine. ROR: https://ror.org/05xaz0w84. ORCID: http://orcid.org/0000-0002-3207-7315.

Novikova N. O. – PhD, Associate Professor of the Department of Technical Cybernetics and Information Technologies named after professor R. V. Merct, Odessa National Maritime University, Odessa, Ukraine. ROR: https://ror.org/05qze6v15. ORCID: https://orcid.org /0000-0002-6257-9703.

Buhaeva I. G. – PhD, Associate Professor of the Department of Technical Cybernetics and Information Technologies named after Professor R. V. Merct, Odessa National Maritime University, Odessa, Ukraine. ROR: https://ror.org/05qze6v15**.** ORCID: https://orcid.org /0000-0002-2839-9266.

Vytnova A. I. – Master of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine. ROR: https://ror.org/05xaz0w84. ORCID: https://orcid.org /0000-0002-4224-3006.

ABSTRACT

Context. Using previously developed code, particularly software classes and class groups related through inheritance, aggregation and composition in object-oriented technologies, significantly reduces software design time.

Objective. Problems arise when it is necessary to store classes for different purposes. In this case, the class name cannot serve as a characteristic for searching. Creating a special structure linking the class name with its purpose will significantly complicate the class repository design. Besides, if relations connect some classes with other classes, there is a need to store their relations along with the class itself, which can significantly complicate both the placement of the class in the repository and its further search. This paper aims to create a special repository of software classes, in which a class is represented by a model that defines its purpose, possible relations with other classes, and role in these relations.

Method. A mathematical model of a software class has been developed, which allows for determining the class designation and possible inter-class relationships. A method of automated placement and search of individual classes and class groups in the class repository is proposed. A storage model is proposed for placing both individual classes and groups of classes connected by inheritance, aggregation and composition relationships. A mechanism has been developed for the automated addition of a separate class or group of classes to the repository, as well as for searching and deleting a separate class or group of classes.

Result. The *ClassCall programme* has been developed to test the proposed solutions. Experiments were conducted to determine the time and quality of placement operations and search of classes and class groups in the repository. The results showed a significant reduction in time for class search compared to known libraries.

Conclusions. The proposed method of automated placement and search of classes based on the class model allows for maintaining class versions, significantly reducing the time to search for the required class and related classes. The method can be used for classes in various object-oriented languages.

KEYWORDS: object-oriented programming, class, class search, composition, inheritance, aggregation, class groups, syntactic analysis.

ABBREVIATIONS

OOP – object-oriented programming.

NOMENCLATURE

abstract is an abstract class
aggreg is a class transformed for aggregation;
attrName is an attribute identification;
attrPurpose is an attribute purpose/designation;
attrType is an attribute type;
cCode is a class code;
cHead is a class header;
cName is a class name;
cNameMain is a main class name;
cNameObject is an object class name;
cNameSource is a source object class name;
cNew is a new introduced class;
compos is a class transformed for composition;
cPurpose is a purpose of class usage;
DictEntry is a dictionary entry;
endAddr is an address of end of method in class code;
fName is a function (method) name;
fPurpose is a function (method) purpose;
fText is a function (method) text;
KeyDict is a key words dictionary;
keyword is a key word;
lang is a programming language;
mC is a set of separate software classes;
mCharacter is a set of necessary characteristics;
mCr is a set of relevant classes;
mGrC is a set of class groups related by some relations;
mGrCr is a set of relevant class groups related by some relations;
mModelC is a model for representing a class;
ordinary is an ordinary class;
queue is a class implementing a queue for aggregation;
Rel is a relation that unites a group;
relation is a relations with other classes;
RepositClasses is a class repository;
rText is a request text;
returnVal is a return value of a function;
Role is a role of class in relation;
sAggred is a set of class names used to implement aggregation;
sArgs is a set of method arguments;

DOI 10.15588/1607-3274-2026-1-13

sAttr is a set of class attributes;

sCharacter is a set of desired characteristics for a searched class;

sClassCode is a set of program class codes;

sClient is a set of client class names;

sCompos is a set of class names used by a given class to implement composition;

sCr is a discovered set of relevant classes;

sGrCr is a discovered set of relevant class groups;

sMClass is a set of class models;

sMeth is a set of functions (methods) of a class;

sResponseClasses1 is a preliminary set of candidate classes for answering a query;

sRKeyword is a set of keywords extracted from a query;

sRsArgs is a set of arguments that return the result of a calculation;

sSynonym is a set of synonyms of a keyword;

startAddr is an address of the start of the method in the class code;

type is a class type;

version is a class version.

INTRODUCTION

When creating object-oriented software, it is possible to significantly reduce design time by using previously developed classes. However, it often turns out that the time to find the required class exceeds the time of its creation. Existing class libraries [1, 2] are usually limited to a specific programming language and a narrow class specialisation (strings, graphical interface, working with files, etc.). Therefore, even within the same design organisation, projects in different subject areas can be carried out, and corresponding class sets can be created. Creating a class for a certain subject area does not mean that it cannot be successfully used in another subject area. Building a multi-subject repository for software classes gives rise to the problem of placing and searching for classes in such a repository.

To solve a certain design problem, interaction relationships of varying degrees of stability are established between classes.

The inheritance relationship (Fig. 1) implies that the derived class always “knows” its parent [3]. When using a class repository, a link from the parent class to the derived class may be needed (for example, to select a class with some additional capabilities), which is not supported by programming languages.

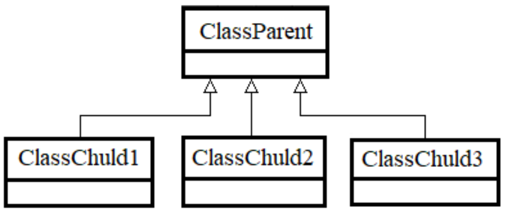


Figure 1 – The inheritance relationship

Moreover, there should be some means of identifying possible inheritance relations among some subsets of repository classes [3]. A composition relation arises if it is necessary to extend the functionality of some class *ClassMain* by using another class *ClassAttr* as an attribute, which *ClassMain* manages alone (Fig. 2a)). In this case, some modification of the attribute class [4] may be required, which is shown in Fig. 2b). Here there may be a need to indicate the availability of this relation in the main class, to place the transformed class-attribute (*ClassAttrConv*) in the repository along with the original class-attribute (*ClassAttr*) and for the latter to specify the class it serves.

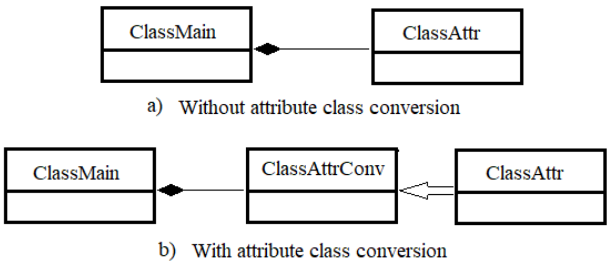

ClassMain
ClassAttr
a) Without attribute class conversion
ClassMain
ClassAttrConv
ClassAttr
b) With attribute class conversion

Figure 2 – Composition relationship

The aggregation relation also allows some *ClassMain* to use another class as an attribute (Fig.3a)). However, the *ClassMain* does not create the *ClassAttr* object. This may lead to the use of the *ClassAttr* by multiple *ClassMain* classes. Such a problem is solved by modernising *ClassMain* classes and using a queue class (*ClassQueue*) [5] (Fig. 3b)). Thus, the representation of an aggregation relation in a repository requires placing a number of classes with certain roles in it and fixing the relations between them.

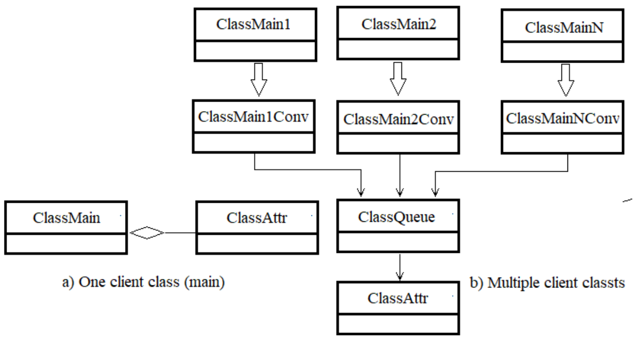


Figure 3 – Aggregation relationship

Given all the above-mentioned, there is a problem of placing and searching in the repository of programme classes intended for solving problems in different subject areas and such class groups connected by certain relations.

The object of study is the processes of defining relationships between software classes, placing and searching for groups of classes connected by relationships in repositories.

DOI 10.15588/1607-3274-2026-1-13

The subject of study is models and repositories of software classes for OOP technologies.

The research aims to reduce the time spent on placement, search, and transformation of classes within the class repository. For this purpose, the following tasks should be solved:

– to create a class model;

– to develop a mechanism for placing a class (class group connected by a certain relation) in the class repository;

– to develop a mechanism for searching a class (class group) by specified characteristics.

1 PROBLEM STATEMENT

Let assume there be a set of separate software classes represented by their codes

$$mC=\{cCod_1, \dots , cCod_n\},$$

as well as a set of class groups linked by some relations

$$mGrC=\{GrC_1, \dots , GrC_k\}.$$

Each group has the form:

$$GrC_i = <Rel,\ \{<cCod_1,\ Role>, \dots, <cCod_m,\ Role>\},$$

where *Rel* is the relation uniting the group,

– *Role* is the role of each class in the relation.

mC and *mGrC* should be placed in a storage repository:

$$(mC,\ mGrC) \rightarrow Storage.$$

To ensure the placement and search of elements in the storage, it is proposed to use the *ModelC* model for each class, containing its characteristics *sCharacter* . Then the storage can be represented by a tuple

$$Storage = <mModelC,\ mC,\ mGrC>.$$

This makes it possible to organize the search for the required class by a request to the repository storage of the form:

$$Request\ (sCharacter) \rightarrow mModelC = sCr,$$

where *sCharacter* is a set of desired characteristics for the sought class,

– *sCr* is a discovered set of relevant classes.

A similar query has a form for finding a group of classes related by the *Rel* relation:

$$Request\ (sCharacter) \rightarrow mModelC = sGrCr,$$

where *sGrCr* is the discovered set of relevant class groups.

2 REVIEW OF THE LITERATURE

Identifying its type is the first task of organising a class repository. Traditionally, program classes are stored in libraries. Such libraries provide ready-made, tested classes for solving everyday tasks like working with text strings, files, graphical interfaces, etc. [2]. They also implement basic functionality that can be used in different projects, and are usually well documented and have a stable API, which makes them easy to use. However, libraries are focused on a specific programming language, so their reuse is limited to a specific platform [1]. In addition, the classes in such libraries are usually highly specialised and cannot always be adapted to new needs, and the relationships between classes in libraries are usually limited to inheritance, which makes it challenging to integrate them into complex systems.

The paper [6] considers the possibility of storing heterogeneous information in a library, but it does not provide for storing groups of related elements with a specific role in the group belonging to the same programming language. The task of evaluating a library from the user's point of view is analysed in [7]. However, among the proposed criteria, there is no evaluation from the point of view of the ability to structure information.

The paper [8] focuses on the issue of creating metadata for storing UML diagrams in a repository. We believe that in the framework of our research, such metadata can be a model of a software class.

Some authors [9, 10, 11] analyse the capabilities of known repositories for software. Undoubtedly, it is possible to organise a repository of classes and class groups within the repository capabilities. However, such a solution seems to be excessively costly.

The second task of building a class repository is the mechanism of placement and search of stored elements (classes, class groups). The efficiency of searching code in natural language using different methods is investigated in the paper [12]. The task can be significantly simplified if, when loading an element into the repository, we use a class model supplemented with natural language text about the purpose of the class and its functions (methods) [13]. This solution can be taken as a basis for searching for the required class in the repository.

However, given the use of classes in different subject areas, the problem of matching the query and class description texts arises. In the study [14], an adaptive text comparison method is proposed, but it is applicable when the degree of overlap between the elements of the compared texts is high. A more universal solution, based on the pre-definition of terms, is proposed in [15]. The study [16] solves the problem by identifying lexically similar words. A solution based on clustering and fuzzy string comparison is proposed in [17]. Such solutions can be adopted if the subject domain is defined and there are enough texts for preliminary analysis. In our case, the set of subject areas is a condition of the problem. In our opinion, a simplified term extraction procedure and an improved method for calculating the Levenshtein distance

DOI 10.15588/1607-3274-2026-1-13

for string comparison, as a universal and quite effective one, should be addressed [18].

When placing and searching classes in the repository, the task of code fragment extraction and comparison may arise. The task of code retrieval has recently become very popular [19]. In [20], an approach to cross-language code search is proposed, which is based on building and scanning a unified abstract syntax tree (UAST) to automate code comparison partially. The method is quite universal, but it seems to be too complicated for our task.

Deep learning methods for determining the semantic properties of programs are analysed in [21]. The methods can give a general characteristic of a program, whereas a class repository needs information about class elements. Comparison of code fragments based on functions, control statements and data types is considered in [22]. A similar task, but in relation to analysing the class structure and selecting its separate elements, was solved in [4], which can be taken as a basis for developing a class repository.

An interesting idea is proposed in [23], where the efficiency of software reuse is determined based on software metrics. However, with respect to the class repository, the metrics-based definition of class inclusion in the repository seems to be too risky.

3 MATERIALS AND METHODS

Repository and class models. The repository model contains a set of class representations (models), a set of terms, and a set of class codes. The class model creates a class representation (class metadata). The purpose of the model is to ensure the class is loaded into the repository and searched for upon request (Fig. 4). The proposed model is a further development of the class model proposed in [13]. A significant development of the proposed model is the inclusion of information about relationships with other classes and the role of the class in relationships. The set of terms ensures a preliminary search for classes in the storage. The extraction of terms (keywords) from the text is discussed in detail in [24].

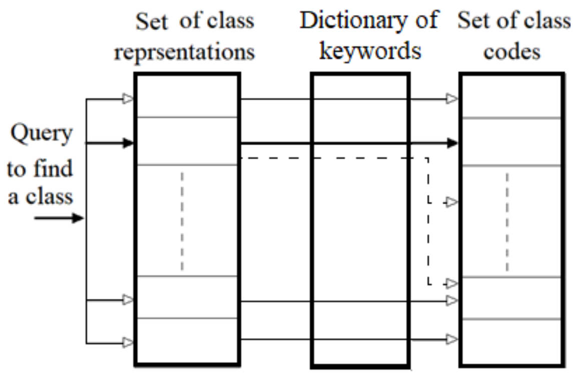


Figure 4 – Class (class group) search scheme)

The class repository is represented as a tuple

$$RepositClasses = <sMClass, KeyDict, sClassCode>, \quad (1)$$

where *sMClass* – set of class models; *KeyDict* – dictionary of keywords; *sClassCode* – set of program class codes.

The *KeyDict* dictionary contains many entries

$$KeyDict = \{DictEntry_1, \dots, DictEntry_k\}.$$

Each entry is represented as a tuple:

$$DictEntry_i = <keyword, sSynonym>,$$

where *keyword* is a key word and *sSynonym* is a set of synonyms for this keyword.

To find the required class, each class is represented by its general purpose of use and the purpose of using each method. Information about the class's purpose, the class type and the relationships will be placed in the class header. Information about the class methods will be placed in the methods view. Class attributes are not planned to be used in the class search procedure, but it makes sense to enter them into the model as additional information for the user when choosing the best class if the search result is a group of classes.

A class is represented by a tuple:

$$c = < cHead, sMeth, sAttr, sKeyword >, \quad (2)$$

where *cHead* – class header; *sMeth* – set of functions (methods) of a class; *sAttr* – set of attributes of a class; *sKeyword* – set of keywords used in the preliminary stage of class search.

Class header. It is proposed to include the general purpose of using the class in the class header as a separate sentence to compare classes.

Thus, the class header is represented as a tuple:

$$cHead =< cName, cPurpose, type, relation, lang, version>, \quad (3)$$

where *cName* – class name; *cPurpose* – purpose of using the class; *type* – class type; *relation* – relationship with other classes: *lang* – programming language; *version* – class version.

The class type can take the following values: *abstract* – abstract class; *ordinary* – ordinary class; *compos* – class transformed for composition; *aggreg* – class transformed for aggregation; *queue* – class implementing a queue for aggregation.

The type of the *relation* element depends on the type of the class.

DOI 10.15588/1607-3274-2026-1-13

For *abstract* and *ordinary* type classes, the relation has the following form:

$$relation =< sParent, sCompos, sAggred>, \qquad (4)$$

where *sParent* – set of parent class names for a given class; *sCompos*– set of class names used by a given class to implement composition (object classes can be ordinary classes or classes transformed for composition); *sAggred* – set of class names used by a given class to implement aggregation (object classes can be ordinary classes or classes transformed for aggregation).

For a *compos-type* class, the relation has the following form:

$$relation =< cNameMain, cNameSource>, \qquad (5)$$

where *cNameMain* – main class name; *cNameSource* – source object class name

For a *queue-type* class, the relation has the following form:

$$relation =< sClient, cNameObject>, \qquad (6)$$

where *sClient* is a set of names of client classes (classes willing to access the object class); *cNameObject* – object class name.

Class attributes. To understand the essence of a class attribute, it is proposed that the concept of the purpose (designation) of the attribute and its type be introduced into the model.

As a result, each attribute from the set *sAttr* will be represented as:

$$Attr=< attrName, attrPurpose, attrType>, \qquad (7)$$

where *attrName* – attribute identifier; *attrPurpose* – attribute purpose; *attrType* – attribute type.

Class methods. The main information for searching classes is in the class methods. Therefore, it is proposed that each method's designation be formulated as a short phrase, for example, “calculate the cost of the order”. For the method's arguments, the earlier attribute rules should be used.

As a result, each method from the set *sMeth* (2) will take the form as follows:

$$\begin{array}{c} func=< fName, fPurpose, fText, sArgs, returnVal, \\ sRsArgs, startAddr, endAddr>, \end{array} \qquad (8)$$

where *fName* – function (method) name; *fPurpose* – function (method) purpose; *fText* – function (method) text.

sArgs – set of method arguments; each argument is represented by: an identifier *id*, a type *argType*, and a purpose *argPurpose*; *returnVal* = <*retType*, *purpose resp*> – the return value of a function (represented by the value type and target); *sRsArgs* – set of arguments that return the result of a calculation; *startAddr* – address of the start of the method in the class code; *endAddr* – address of the end of the method in the class code.

Repository functions. The following procedures are defined for the classes in the repository:

– Add a class;
– Add a class group;
– Search a class;
– Search a class group;
– Upgrade a class;
– Delete a class;
– Delete a class group.

Add a class. After entering a new class *cNew*, its code is compared with the codes of classes in the repository. If a match is found between the codes $cNewCode=cCode_i$, then the input operation is cancelled. If a match of class names is found $cNew.cName=c.cName_i$, then the specialist must change the name of the loaded class.

The methods *cNew.sMeth* are automatically extracted from the class code.

The specialist must formulate the purpose of the class *cNew.cHead.cPurpose* within one short phrase. The purposes of each method $cNew.func_i.fPurpose$, implementing the main purpose of the class, should be formulated in a similar way. Based on the purpose of the class and its methods, the program creates keywords for the class *cNew.sKeyword* and replenishes the dictionary of keywords of the *KeyDict* storage repository.

$$KeyDict.\ sKeyword \cup cNew.sKeyword.$$

Add a class group. The entry operation is cancelled if a class group is entered and all the codes match the group in the repository.

If a class group related by an inheritance relationship is entered, and a match is observed for the code of the parent class, then the derived classes are entered according to the rules for entering a separate class.

If a class group related by a composition relationship is entered, and the codes of the main classes match, but the codes of the attribute classes do not match, then a specialist is called in to correct the error.

If a class group related by an aggregation relationship is entered, and a mismatch in the code is detected for any class in the group, then the entire group is entered under a new name.

If a match in the class names is detected, then the specialist must change the name of the loaded class.

For each class in the group, the operations of describing the class purpose and its methods are performed. The program, based on the purpose of each class and its methods, creates keywords for the class.

Search a class**.** To search for a class, the user must formulate a query/request in the form of text that formulates the purpose of using this class

$$Request = rText,$$

rText– request text.

The system forms a list of keywords from the query nouns

$$Request \rightarrow < rText,\ sRKeyword >,$$

where *sRKeyword* – set of keywords extracted from a query.

The system detects the occurrence of query keywords in the repository's *KeyDict* keyword dictionary. If some query keyword is detected in the dictionary,

$$RKeyword_j = DictEntry_i$$

then *sRKeyword* is being updated with synonyms for this keyword.

$$sRKeyword \rightarrow sRKeyword \cup DictEntry_i.sSynonym_i.$$

Based on the comparison of *sRKeyword* with the lists of class keywords, a preliminary set of candidate classes for the response to the request *sResponseClasses1* is formed.

The next step is to compare the text (fuzzy string comparison) of the *rText* request with the texts of the purpose of the $cPurpose_i$ class and its $fPurpose_{i,j}$ methods for each class from *sResponseClasses*1.

As a result, a response to the request is received in the following form:

$$sResponseClasses1 \rightarrow sResponseClasses2 = \{ClassCode_1, \dots, ClassCode_q\},$$

where q is set by user.

The resulting list of candidate classes is ranked by the degree of compliance with the query. The user can view the class code from the list.

Additional restrictions can be added to your query, for example, by specifying the class language: $Request = < rText,\ lang >$.

Search a class group. Searching for a class group starts with searching for a single class.

For example, the parent class. Unlike searching for a single class, the relationship of that class to other classes in the group should be added to the query:

$$Request = < rText,\ relation, type>.$$

The search for a class is performed in the previously specified sequence. Then, at the user's request, it is possible to see the entire group of classes connected by the specified relationship.

4 EXPERIMENTS

In accordance with the proposed model, a program for conducting experiments, *ClassCell*, was developed. Fig. 5 shows the process of loading a class into the repository. The *Composition converter* block [4] was used as a code analyser, and the *TerEx* [24] software product was used to extract keywords.

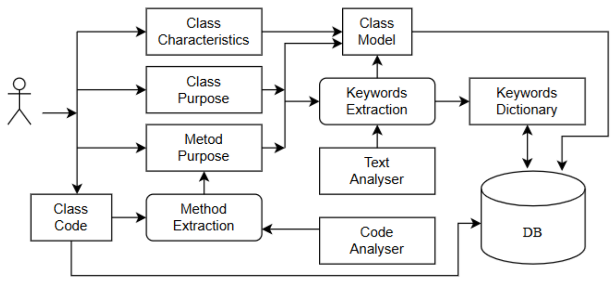


Figure 5 – Scheme of loading a class into a repository

Figure 6 shows the process of searching for a class in the repository. Based on a keyword search, a set of classes, *sClasses1*, is formed at the first stage. At the second stage, as a result of comparing the query texts and class models, the set of proposed classes is reduced to *sClasses2*.

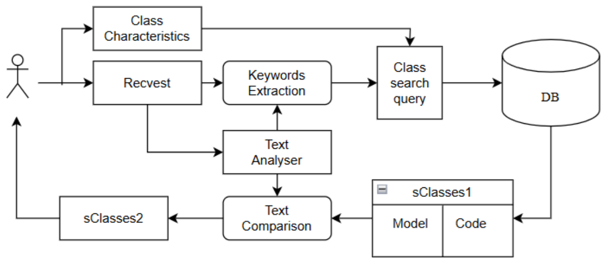


Figure 6 – Searching for classes in the repository

Figure 7 shows the first step of loading a new class into the repository. The user can copy the class text or enter it as a separate file. At the same time, the purpose of the class should be entered.

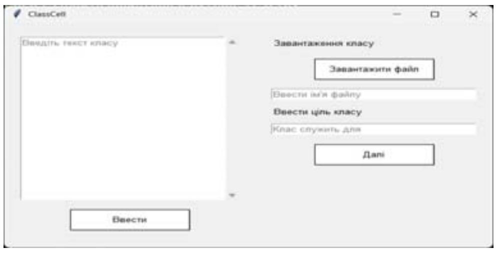
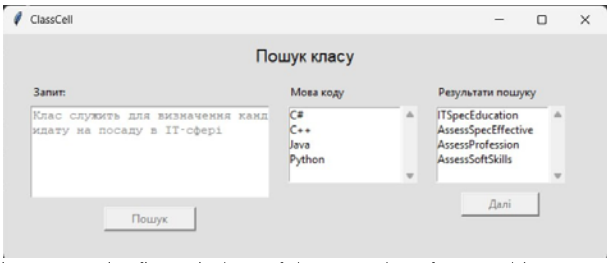


Figure 7 – The first window of the procedure for loading a class into the repository

Fig. 8 shows the window that the user sees after entering a class search query. The program offers a list of classes that were found. If the user wishes, he can see class codes, class methods, and class and method assignments.class codes, class methods, and class and method assignments.

DOI 10.15588/1607-3274-2026-1-13



Клас служить для визначення канд
идату на посаду в IT-сфері

C#
C++
Java
Python

ITSpecEducation
AssessSpecEffective
AssessProfession
AssessSoftSkills

Figure 8 – The first window of the procedure for searching a class in the repository

5 RESULTS

The *ClassCell* program allowed us to conduct a series of experiments to determine the time of loading and searching for classes (100 classes in total). The study's results were the time characteristics and completeness of query responses. Loading a class requires time from 3 to 20 minutes. The average time was 7.9 minutes. At the same time, loading the code was performed in an average of 20.1 seconds. The rest of the time was spent on formulating the purpose of the class and its methods.

The time to search for a class in the repository is from 2 to 7 minutes. Basically, this is the time for composing the query text and viewing the detected classes, since no more than 10 seconds were spent directly searching for a class in the repository.

The experiment showed that increasing the number of classes virtually has no effect on the time needed to search for a class. If we assume that the time for a class search in an unindexed library is about 5 minutes per class, then from the graph (Fig. 9) it follows that even with 10 classes in the repository, the time for a class search in it is 2.5 times less than in the library.

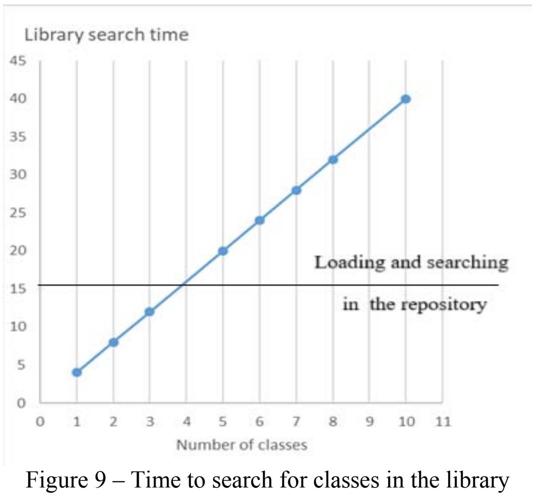


Figure 9 – Time to search for classes in the library and repository

In all cases, the set of classes representing the search result contained all classes relevant to the query subject area, unless a limit was imposed on the number of classes in the query response.

6 DISCUSSION

We assume that the user who uploads a class to the repository is the developer of this class. Then the formation of information about the class designated purpose and methods will not take much time. In addition, we do not limit the volume of this information and do not require precise wording. The more text there is in the class description and in the request for its search (of course, from the subject area under consideration), the more terms will be highlighted, the more likely it is that all the "targeted classes" will be found. At the same time, we are studying the possibility of using artificial intelligence to formulate the purpose and capabilities of a class based on its code.

It should be noted that existing libraries do not offer mechanisms for working with class groups, while they can be partially or entirely reused.

The rationale for limiting the number of classes in a query response remains an open issue.

CONCLUSIONS

A model of a software class has been developed which, unlike existing ones, allows for the organisation of groups of classes linked by inheritance, composition and aggregation relationships.

A mechanism for placing classes in a repository has been developed, providing for automatic replenishment of the dictionary of terms, separate storage of the code and class model with keywords and possible links to other classes involved in the relationship.

A mechanism for class searching based on a natural language query has been developed, providing a two-stage process of searching for relevant classes. At the first stage, it was done by keywords, at the second stage – by comparing the query texts and the class description, the third stage (if necessary) implied determining all classes associated with the desired, specific relationship

The software was created to perform experiments. The experiments showed that, with 10 classes already in the repository, a significant reduction in the time for loading and searching for classes was observed. In all cases, the set of classes representing the search result contained all classes related to the subject area of the query, if no limit was imposed on the number of classes in the response to the query.

The conducted research can serve as a basis for constructing repositories of software classes in design organisations using object-oriented technologies.

ACKNOWLEDGEMENTS

We express our gratitude to the students Bondar Valeriia R. and Gratilova Kateryna O. for their active participation in the development of the Composition Converter software product, as well as to Master's student Mileiko I.I. for her outstanding contribution to the development of the *TerEx* program. The use of these software solutions significantly reduced the time for creating a program for *ClassCell* experiments.

DECLARATIONS

Conflict of interest: The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Authors' contributions: Oleksii Kungurtsev: development of repository and class models; Nataliia Novikova: developing procedures for adding and deleting classes from the repository; Iryna Buhaieva: developing procedures for searching for a class and a group of classes in a repository; Alina Vytnova: experimental study of methods.

Data availability: All data is provided in the article.

Software availability: Software can be provided upon request.

Use of artificial intelligence tools: The authors confirm that they did not use artificial intelligence technologies in creating the submitted work.

REFERENCES

1. Microsoft Corporation. .NET Class Libraries. Microsoft Learn [Electronic resource], 2022. Mode of access: https://learn.microsoft.com/en-us/dotnet/standard/classlibraries. free (date of the application: 01.12.2022). Header from the screen.
2. Krill P. 12 top-notch libraries for C++ programming [Electronic resource]. Electronic text data, InfoWorld, 2022. Mode of access: https://www.infoworld.com/article/2265967/12-top-notch-libraries-for-c-plus-plus-programming.html. free (date of the application: 14.10.2022). Header from the screen.
3. Lee G. Modern Programming: Object Oriented Programming and Best Practices. Packt Publishing, 2019, 266 p.
4. Kungurtsev O. B., Bondar V. R., Gratilova K. O. et al. Method Automated Class Conversion for Composition Implementation, *Radio Electronics, Computer Science, Control,* 2024, №2, pp. 142–149. DOI: 10.15588/1607-3274-2024-2-14
5. Kungurtsev O., Komleva N. Implementation of class interaction under aggregation conditions, *Eastern-European Journal of Enterprise Technologies,* 2024, №2 (128), pp. 20–30. DOI: 10.15587/1729-4061.2024.301011
6. Perhac P., Simonak S. Algorithms and data structure libraries for JAVA, *Acta Electrotechnica et Informatica,* 2020, Vol. 20, No. 1, pp. 39–48. DOI: 10.15546/aeei-2020-0006
7. Tanzil M. H., Uddin G., Barcomb A. "How do people decide?": A Model for Software Library Selection, 1*7th International Conference on Cooperative and Human Aspects of Software Engineering, June 2024.* DOI:10.1145/3641822.3641865
8. Di Felice P. Paolone G., Paesani R. et al. Design and Implementation of a Metadata Repository about UML Class Diagrams. A Software Tool Supporting the Automatic Feeding of the Repository, *Electronics*, 2022, № 11, P. 201. DOI: 10.3390/electronics11020201
9. Dobrzyński B., Sosnowski J. Text Mining Studies of Software Repository Contents, *18th International Conference on Evaluation of Novel Approaches to Software Engineering,* 2023, pp. 562–569. DOI: 10.5220/0011970100003464 ISBN: 978-989-758-647-7; ISSN: 2184-4895
10. Polaczek J., Sosnowski J. Exploring the software repositories of embedded systems: An industrial experience, *Information and Software Technology*, 2021, Vol. 131, P. 106489. DOI: 10.1016/j.infsof.2020.106489
11. Moya J. Repository Software in Software Development [Electronic resource]. Electronic text data. Mode of access: https://www.wearecapicua.com/blog/software-repository-engineering-development. free (date of the application: 25.10.2024). – Header from the screen.
12. Yan S., Yu H., Chen Y. et al. Are the Code Snippets What We Are Searching for? A Benchmark and an Empirical Study on Code Search with Natural-Language Queries, *27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Singapore Management University, Singapore, 2020, pp. 344–354. DOI: 10.1109/SANER48275.2020.9054840
13. Kungurtsev O. B., Vytnova A. I. Determination of Inheritance Relations and Restructuring of Software Class Model in the Process of Developing Information Systems, *Radio Electronics, Computer Science, Control,* 2022, № 4, pp. 98–107. DOI: 10.15588/1607-3274-2022-4-8.
14. Kaufman A. R., Klevs A. Adaptive Fuzzy String Matching: How to Merge Datasets with Only One (Messy) Identifying Field, *Published online by Cambridge Universit. Political Analysis*, 2022, Vol. 30 , Issue 4, pp. 590–596.
15. Smirnov A., Shilov N., Evers K. et al. Free Text Customer Requests Analysis: Information Extraction Based on Fuzzy String Comparison, *17th IFIP International Conference on Product Lifecycle Management (PLM)*. Rapperswil, Switzerland, Jul 2020, pp. 193–202. DOI:10.1007/978-3-030-62807-9_16
16. Kalyanathaya K. P., Akila D., Suseendran G. A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP, *Journal of Computational Information Systems,* 2019, Vol. 15(3), pp. 26–32.
17. Kostanyan A., Harmandayan A. Fuzzy Segmentations of a String [Electronic resource], Electronic text data, Cornell University, 2022. Mode of access: https://doi.org/10.48550/arXiv.2201.13427. free (date of the application: 31.01.2022). Header from the screen.
18. Pikies M., Ali J. Analysis and safety engineering of fuzzy string matching algorithms, *ISA Transactions*, 2020, Vol. 113, P. 1. DOI:10.1016/j.isatra.2020.10.014.
19. Liu C., Xia X., Lo D. et al. Opportunities and Challenges in Code Search Tools [Electronic resource]. Electronic text data, 2020. Mode of access: https://arxiv.org/abs/2011.02297. free (date of the application: 4.11.2020). – Header from the screen.
20. Gorchakov A. V., Demidova L. A. Methods and Algorithms for Cross-Language Search of Source Code Fragments, *2024 International Conference on Information Technologies (InfoTech)*. Sofia, Bulgaria, 11–12 September 2024. DOI: 10.1109/InfoTech63258.2024.10701403
21. Han S., Wang D., Li W., Lu Xuesong et al. A Comparison of Code Embeddings and Beyond [Electronic resource]. Electronic text data. Cornell University,2021. Mode of access: https://doi.org/10.48550/arXiv.2109.07173. free (date of the application: 15.09.2021). Header from the screen.
22. Sudhamani M., Rangarajan L. Code similarity detection through control statement and program features, *Expert Systems with Applications,* 2019, Vol. 132, pp. 63–75. –DOI: 10.1016/j.eswa.2019.04.045
23. Shatnawi R. A Classification of Software Modules into Library and Application Components in the Open-Source Field, *International Journal of Software Engineering and Its Applications*, 2016, Vol. 10(3), pp. 179–190. DOI: 10.14257/ijseia.2016.10.3.16

24. Kungurtsev O. B., Mileiko I. I., Novikova N. O. Technology for Automated Construction of Domain Dictionaries with Special Processing of Short Documents, *Radio Electronics, Computer Science, Control,* 2024, № 4, P. 148. DOI: 10.15588/1607-3274-2023-4-14

Received 09.07.2025.
Accepted 15.01.2026.
Published 27.03.2026.

УДК 004.415.2

РОЗРОБКА СХОВИЩА КЛАСІВ ДЛЯ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ СТВОРЕННЯ ПРОГРАМНИХ ПРОДУКТІВ

Кунгурцев О. Б. – канд. техн. наук, професор кафедри інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна. ROR: https://ror.org/05xaz0w84. ORCID: http://orcid.org/0000-0002-3207-7315.

Новікова Н. О. – канд. техн. наук, доцент кафедри Технічна кібернетика й інформаційні технології ім. професора Р. В. Меркта Одеського національного морського університету, Одеса, Україна. ROR: https://ror.org/05qze6v15. ORCID: https://orcid.org /0000-0002-6257-9703.

Бугаєва І. Г. – канд. техн. наук, доцент кафедри Технічна кібернетика й інформаційні технології ім. професора Р. В. Меркта Одеського національного морського університету, Одеса, Україна. ROR: https://ror.org/05qze6v15. ORCID: https://orcid.org /0000-0002-2839-9266.

Витнова А. І. – магістр кафедри інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна. ROR: https://ror.org/05xaz0w84. ORCID: https://orcid.org /0000-0002-4224-3006.

АНОТАЦІЯ

Актуальність. Використання раніше розробленого коду, зокрема, програмних класів та груп класів, пов'язаних відносинами спадкування, агрегації та композиції в об'єктно-орієнтованих технологіях, істотно скорочує час проектування програмного забезпечення.

Мета роботи. Проблеми виникають при необхідності зберігання класів різного призначення. У цьому випадку найменування класу не може бути його характеристикою для пошуку. Створення спеціальної структури, що пов'язує ім'я класу з його призначенням, істотно ускладнить сховище класів. Крім цього, якщо деякий клас пов'язаний відносинами з іншими класами, то необхідно зберігати поряд з класом його зв'язки, що може суттєво ускладнити як розміщення класу у сховищі, так і надалі його пошук. Метою роботи є створення спеціального сховища програмних класів, в якому клас представлений моделлю, що дозволяє визначити його призначення, можливі зв'язки з іншими класами та роль у цих зв'язків.

Метод. Розроблено математичну модель програмного класу, яка дозволила визначити призначення класу та можливі зв'язки з іншими класами. Запропоновано метод автоматизованого розміщення та пошуку окремих класів, а також груп класів у сховищі класів. Запропоновано модель сховища для розміщення як окремих класів, так і груп класів, пов'язаних відносинами спадкування, агрегації та композиції. Розроблено механізм автоматизованого додавання окремого класу або групи класів до сховища, а також пошуку та видалення окремого класу чи групи класів.

Результати. Для апробації запропонованих рішень розроблено програму *ClassCell*. Проведено експерименти для визначення часу та якості виконання операцій розміщення та пошуку класів та груп класів у сховищі. Результати показали суттєве скорочення часу на пошук класів порівняно з відомими бібліотеками.

Висновки. Запропонований метод автоматизованого розміщення та пошуку класів на основі моделі класу дозволяє підтримувати версії класів, суттєво скоротити час на пошук потрібного класу та пов'язаних з ним класів. Метод може бути використаний для класів на різних об'єктно-орієнтованих мовах.

КЛЮЧОВІ СЛОВА: об'єктно-орієнтоване програмування, клас, пошук класу, композиція, успадкування, агрегація, групи класів, синтаксичний аналіз.

ЛІТЕРАТУРА

1. Microsoft Corporation. .NET Class Libraries. Microsoft Learn [Electronic resource]. – 2022. – Mode of access: https://learn.microsoft.com/en-us/dotnet/standard/classlibraries. free (date of the application: 01.12.2022). – Header from the screen.
2. Krill P. 12 top-notch libraries for C++ programming [Electronic resource] / Paul. Krill. – Electronic text data. – InfoWorld, 2022. – Mode of access: https://www.infoworld.com/article/2265967/12-top-notch-libraries-for-c-plus-plus-programming.html. free (date of the application: 14.10.2022). – Header from the screen.
3. Lee G. Modern Programming: Object Oriented Programming and Best Practices / G. Lee. – Packt Publishing, 2019. – 266 p.
4. Method Automated Class Conversion for Composition Implementation / [O. B. Kungurtsev, V. R. Bondar, K. O. Gratilova, et al.] // Radio Electronics, Computer Science, Control. – 2024. – № 2. – P. 142–149. DOI: 10.15588/1607-3274-2024-2-14
5. Kungurtsev O. Implementation of class interaction under aggregation conditions / O. Kungurtsev, N. Komleva // Eastern-European Journal of Enterprise Technologies. – 2024. – №2 (128). – P. 20–30. DOI: 10.15587/1729-4061.2024.301011
6. Perhac P. Algorithms and data structure libraries for JAVA / P. Perhac, S. Simonak // Acta Electrotechnica et Informatica. – 2020. – Vol. 20, No. 1. – P. 39–48. DOI: 10.15546/aeei-2020-0006
7. Tanzil M. H. "How do people decide?": A Model for Software Library Selection / M. H. Tanzil, G. Uddin, A. Barcomb // 17th International Conference on Cooperative and Human Aspects of Software Engineering, June 2024. DOI:10.1145/3641822.3641865
8. Di Felice P. Design and Implementation of a Metadata Repository about UML Class Diagrams. A Software Tool Supporting the Automatic Feeding of the Repository / [P. Di Fe-

DOI 10.15588/1607-3274-2026-1-13

lice, G. Paolone, R. Paesani et al.] // Electronics. – 2022. – № 11. – P. 201. –DOI: 10.3390/electronics11020201

9. Dobrzyński B. Text Mining Studies of Software Repository Contents / B. Dobrzyński, J Sosnowski // 18th International Conference on Evaluation of Novel Approaches to Software Engineering. – 2023. – P. 562–569. DOI: 10.5220/0011970100003464 ISBN: 978-989-758-647-7; ISSN: 2184-4895
10. Polaczek J. Exploring the software repositories of embedded systems: An industrial experience / J. Polaczek, J. Sosnowski // Information and Software Technology. – 2021. – Vol. 131. – P. 106489. DOI: 10.1016/j.infsof.2020.106489
11. Moya J. Repository Software in Software Development [Electronic resource] / J. Moya. – Electronic text data. – Mode of access: https://www.wearecapicua.com/blog/software-repository-engineering-development. free (date of the application: 25.10.2024). – Header from the screen.
12. Are the Code Snippets What We Are Searching for? A Benchmark and an Empirical Study on Code Search with Natural-Language Queries / [S. Yan, H. Yu, Y. Chen et al.] // 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). – Singapore Management University, Singapore, 2020. – P. 344–354. DOI: 10.1109/SANER48275.2020.9054840
13. Kungurtsev O. B. Determination of Inheritance Relations and Restructuring of Software Class Model in the Process of Developing Information Systems / O. B. Kungurtsev, A. I. Vytnova // Radio Electronics, Computer Science, Control. – 2022. – № 4. – P. 98–107. DOI: 10.15588/1607-3274-2022-4-8.
14. Kaufman A. R. Adaptive Fuzzy String Matching: How to Merge Datasets with Only One (Messy) Identifying Field / A. R. Kaufman, A. Klevs // Published online by Cambridge Universit. Political Analysis . – 2022. – Vol. 30, Issue 4. – P. 590 – 596.
15. Free Text Customer Requests Analysis: Information Extraction Based on Fuzzy String Comparison / [A. Smirnov, N. Shilov, K. Evers et al.] // 17th IFIP International Conference on Product Lifecycle Management (PLM). – Rapperswil, Switzerland, Jul 2020. – P. 193–202. DOI: 10.1007/978-3-030-62807-9_16
16. Kalyanathaya K. P. A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP / K. P. Kalyanathaya, D. Akila, G. Suseendran // Journal of Computational Information Systems. – 2019. – Vol 15(3). – P. 26–32.
17. Kostanyan A. Fuzzy Segmentations of a String [Electronic resource] / A. Kostanyan, A. Harmandayan – Electronic text data. – Cornell University, 2022. – Mode of access: https://doi.org/10.48550/arXiv.2201.13427. free (date of the application: 31.01.2022). – Header from the screen.
18. Pikies M. Analysis and safety engineering of fuzzy string matching algorithms / M. Pikies, J. Ali // ISA Transactions. – 2020. – Vol. 113. – P. 1. DOI:10.1016/j.isatra.2020.10.014.
19. Opportunities and Challenges in Code Search Tools [Electronic resource] / [C. Liu, X. Xia, D. Lo et al.] // – Electronic text data.– 2020. – Mode of access: https://arxiv.org/abs/2011.02297. free (date of the application: 4.11.2020). – Header from the screen.
20. Gorchakov A. V. Methods and Algorithms for Cross-Language Search of Source Code Fragments/ A. V. Gorchakov, L. A. Demidova // 2024 International Conference on Information Technologies (InfoTech), Sofia, Bulgaria, 11–12 September 2024. DOI: 10.1109/InfoTech63258.2024.10701403
21. A Comparison of Code Embeddings and Beyond [Electronic resource] / [S. Han, D. Wang, W. Li, Xuesong Lu et al]. – Electronic text data. – Cornell University,2021. – Mode of access: https://doi.org/10.48550/arXiv.2109.07173. free (date of the application: 15.09.2021). – Header from the screen.
22. Sudhamani M. Code similarity detection through control statement and program features/ M. Sudhamani, L. Rangarajan // Expert Systems with Applications. – 2019. – Vol. 132. – P. 63–75. DOI: 10.1016/j.eswa.2019.04.045
23. Shatnawi R. A Classification of Software Modules into Library and Application Components in the Open-Source Field / R. Shatnawi // International Journal of Software Engineering and Its Applications. – 2016. – Vol. 10(3). – P. 179–190. DOI: 10.14257/ijseia.2016.10.3.16
24. Kungurtsev O. B. Technology for Automated Construction of Domain Dictionaries with Special Processing of Short Documents / O. B. Kungurtsev, I. I. Mileiko, N. O. Novikova // Radio Electronics, Computer Science, Control. – 2024. – № 4. – P. 148. DOI: 10.15588/1607-3274-2023-4-14