







Approach to the Search for Software Projects Similar in Structure and Semantics Based on the Knowledge Extracted from Existed Projects

Filippov Aleksey Alekundrovich , Guskov Gleb Yurevich ^(✉) ,
Namestnikov Aleksey Michailovich , and Yarushkina Nudezhda Glebovna 

Ulyanovsk State Technical University,
32, Severny Venetz Street, 432027 Ulyanovsk, Russia
{al.filippov,jng@ulstu.ru}, guskovgleb@gmail.com,
am.namestnikov@gmail.com
<http://www.ulstu.ru/?design=english>

Abstract. This article presents a new effective model, algorithms, and methods for representing the subject area of an information system. The subject area is presented in the form of fragments of the knowledge base of the design support system. The knowledge base is formed in the process of analyzing class diagrams in UML notation and project source code. The proposed approaches can reduce the time of the design process and increase the quality of the obtained information system through the use of successful information system design solutions used in other projects. Search for successful design solutions is carried out using the developed metrics for determining the similarity of software systems projects. The metrics allow calculating the match of pattern in OWL ontology format with the source code of the project.

Keywords: Knowledge base · Information system · Class diagram · Information systems design

1 Introduction

The modern approach to the development of information systems involves the use of intelligent automated tools to support the design process, allowing to search for and reuse successful architectural solutions based on a common semantic representation of subject and design knowledge. The knowledge in such automated tools is currently usually presented in the form of ontologies. Ontology development is a long and resource-intensive process that requires the involvement of specialists with competencies in ontological engineering and software development.

This work was supported in part by the Russian Foundation for Basic Research (Projects No. 19-47-730003, No. 18-47-730019, 19-47-730006, 19-47-730005).

Developers of information systems as a rule do not have sufficient knowledge about the subject area of the project. The documents governing the subject area do not always record all the accepted semantic meanings of entities and relationships. Creating a knowledge base that allows take into account and reuse successful design solutions will reduce the time of design and development, as well as the number of semantic errors. The main problem of using ontologies in the process of developing information systems is the high requirements for knowledge of the internal structure of ontologies.

The importance of formalizing concepts of the subject area for the development of an information system has led to the emergence of special design languages, which include the formalization of domain concepts (entities). The most common design tools are based on the Unified Modeling Language (UML) [1].

Class diagrams in UML notation and Java source code are used as input data for the design support system. UML diagrams are applicable to the description of specific features of information systems, for example:

- classes that make up the architecture of the information system;
- tables and relationships of database schema ;
- properties and characteristics of user’s computer servers to create a physical deployment, etc.

The implementation and use of knowledge-based intelligent systems are relevant in all problem areas nowadays [9,10]. At the moment, a lot of researchers use the ontological approach for the organization of the knowledge bases of expert and intelligent systems: M. Gao, C. Liu [11], D. Bianchini [12], N. Guarino [3], G. Guizzardi [13], R.A. Falbo [14], G. Stumme [15], N.G. Yarushkina [18], T.R. Gruber [16], A. Maedche [17].

Fernando Bobillo Umberto Straccia proposed a fuzzy ontology description format [2]. At this stage in the development of information technology, a large number of open source software systems have been created in various subject areas. Reuse of modules of open software systems will significantly reduce the time spent on software development.

Currently, the strong influence of the characteristics of the problem area, within which the development of software systems is carried out, leads to the frequent use of domain-based development methodology (Domain Driven Development — DDD). This methodology is based on an object-oriented programming paradigm and involves various types of testing that allow performing the function of checking the quality of the source code. But this methodology does not take into account the correctness of the model from the point of view of the features of the subject area. This type of error control is carried out by project managers and leading developers. The formation of the ontological representation of the model allows detecting errors in the perception of the subject area. Knowledge is captured in the ontology in the OWL (Web Ontology Language) [4] format with a predefined *TBox*.

As input to the design support system, class diagrams in UML notation and Java source code are used. The solution a problem of design support information systems consists of executing the following tasks:

1. build a model for representing information system design as the content of a knowledge base;
2. method of ontological indexing of class diagrams in UML notation and project by source code;
3. search methods for effective design solutions in the content of the knowledge base.

2 Software Product Design Model for Representation in the Knowledge Base

Project documentation includes diagrams formalized in UML notations. To solve the problem of the intellectual analysis of design diagrams, it is necessary to formalize the rules of notation in the knowledge base. Such knowledges allow the identification of design patterns and architecture software solutions used in various projects. This allows to search for projects with similar architectural solutions and approaches to the implementation of modules of information systems.

An ontology in OWL format is used as the basis of the knowledge base of the design process support system. The W3C consortium recommends using the $\mathcal{SHOIN}(\mathcal{D})$ formalism [5–7] for the OWL language group (OWL Light, OWL DL, OWL Full) as the logical basis of the ontology description language.

In the context of the $\mathcal{SHOIN}(\mathcal{D})$ description logic, the ontology is a knowledge base of the following form [8]:

$$KB = \{TBox, ABox\}, \quad (1)$$

where $TBox$ — a set of terminological axioms representing general knowledge about the concepts of the knowledge base and their relationships;
 $ABox$ — a set of statements (facts) about individuals.

2.1 Tbox Axioms of Information Systems Design Ontology

The terminology of project diagrams is divided into the logical representation of the UML notation and the logical representation of design patterns.

$$\begin{aligned} Relationship &\sqsubseteq \top; \\ Dependency &\sqsubseteq Relationship; \\ Association &\sqsubseteq Relationship; \\ Generalization &\sqsubseteq Relationship; \\ Realization &\sqsubseteq Relationship \sqcap \exists \\ startWith.Class &\sqcap \exists endWith.Interface; \end{aligned} \quad (2)$$

where *startWith* *endWith* — name of roles *comes from* and *coming to*, respectively.

The main classes can be represented:

$$\begin{aligned}
 & Thing \sqsubseteq \top \forall hasAName.String; \\
 & StructThing \sqsubseteq Thing; \\
 & AnnotThing \sqsubseteq Thing; \\
 & Note \sqsubseteq AnnotThing \sqcap \exists connectedTo.Association; \\
 & Class \sqsubseteq StructThing; \\
 & Object \sqsubseteq StructThing \sqcap \exists isObjectOf.Class \sqcap \forall isObjectOf.Class; \\
 & Interface \sqsubseteq StructThing; \\
 & SimpleClass \sqsubseteq Class; \\
 & AbstractClass \sqsubseteq Class;
 \end{aligned} \tag{3}$$

where *hasAName*, *isObjectOf*, *connectedTo* — roles in relationship; *String* — concrete domain.

Class attributes and methods are represented as follows:

$$\begin{aligned}
 & Attribute \sqsubseteq \top \sqcap \exists \\
 & hasAAttrName.String \sqcap \exists isAPartOf.Class \\
 & Method \sqsubseteq \top \sqcap \exists \\
 & hasAMethName.String \sqcap \exists isAPartOf.Class,
 \end{aligned} \tag{4}$$

where *hasAAttrName* and *hasAMethName* — relationships “has attribute/method name”.

Consider the terminology of design patterns associated with the logical representation of design diagram notation (using the UML class diagram as an example):

$$\begin{aligned}
 & Template \sqsubseteq \top \sqcap \exists \\
 & hasATempName.String \sqcap \exists hasAExpValue.Double \\
 & SomeTemplate \sqsubseteq Template,
 \end{aligned} \tag{5}$$

where *hasATempName* — role “has a design pattern name”;

hasAExpValue — role “has value of expression”;

Double — concrete domain.

Each design pattern in each specific project has a certain degree of expression.

The hierarchy of concepts of the developed ontology is presented in the Fig. 1.

Hierarchy of properties (DataTypeProperty and ObjectProperty) in developed ontology is presented in the Fig. 2.

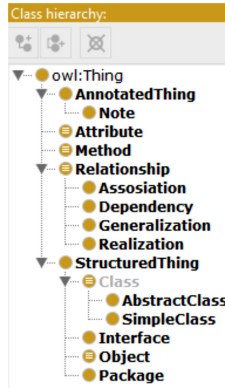


Fig. 1. The hierarchy of concepts of the developed ontology in the editor Protege

2.2 Abox Axioms of Information Systems Design Ontology

The Fig. 3 shows an example of the design pattern “Delegate”, which in the form of a set of *ABox* facts:

$$\begin{aligned}
 & class1: SimpleClass \\
 & class2: SimpleClass; \\
 & attribute1: Attribute \\
 & object1: Object; \\
 & method1: Method \\
 & method2: Method; \\
 & relation1: Association; \\
 & (method1, name1: String) : hasAMethName; \\
 & (method2, name2: String) : hasAMethName; \\
 & (attribute1, class1) : iaAPartOf \\
 & (object1, class2) : isObjectOf; \\
 & (object1, attribute1) : owl : sameAs \\
 & (method1, class1) : iaAPartOf; \\
 & (method2, class2) : iaAPartOf \\
 & (relation1, class1) : startWith \\
 & (relation1, class2) : endWith.
 \end{aligned} \tag{6}$$

In the knowledge base ontology, the *ABox* fact set includes all the facts about the design patterns used. Then, in the indexing process, the facts from *ABox* are compared with the facts extracted from the design diagrams and the degree of expression for each ontology template is determined.

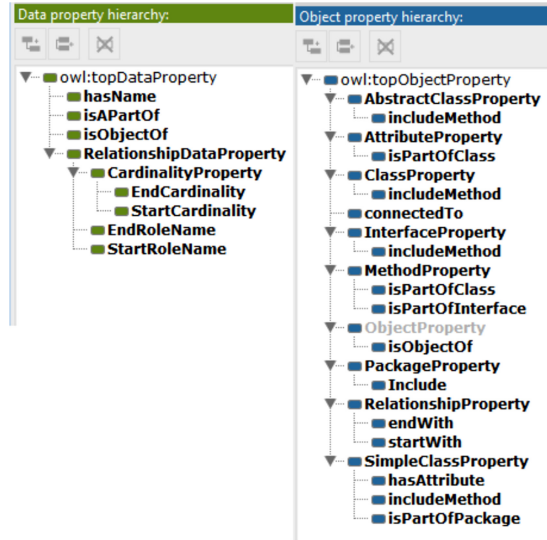


Fig. 2. Hierarchy of DataTypeProperty and ObjectProperty of the developed ontology in the editor Protege

3 Ontological Representation of Design Patterns

Formally, the ontological representation of the design pattern can be represented as follows:

$$OV_i^{tmp} = \{C, D, R^{same_as}\}, \tag{7}$$

- where C — set of individuals in knowledge base;
- D — set of relationship between elements of i – th design patterns, presented as knowledge base individuals;
- R — set of equivalence relationships knowledge base individuals.

Design pattern “Builder” is one of the most commonly used patterns in industrial software development. “Builder” is a generic design pattern and allows to create complex composite objects. Figure 4 shows the UML class diagram of the Builder design pattern in the Visual Paradigm.

Representation of the “Builder” design pattern as a fragment of a knowledge base ontology is defined by the following concept instances:

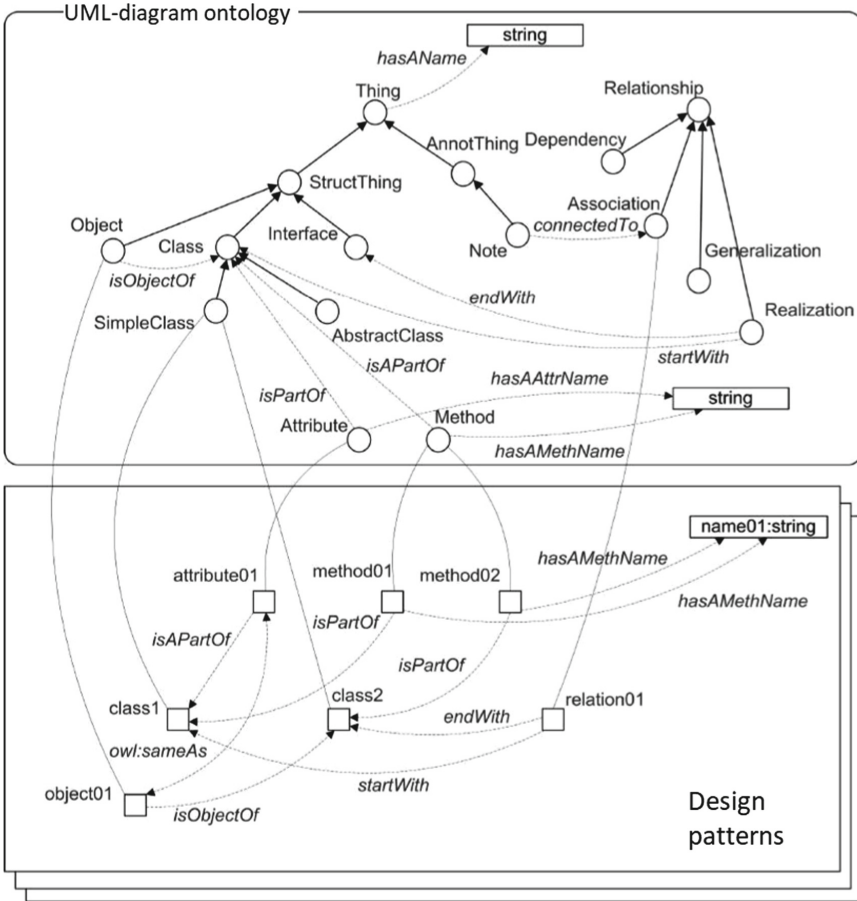


Fig. 3. Ontology structure (including design pattern example)

- Builder.Client*: *SimpleClass*
 - Builder.Director*: *SimpleClass*
 - Builder.ConcreteBuilder*: *SimpleClass*
 - Builder.Product*: *SimpleClass*
 - Builder.AbstractBuilder*: *AbstractClass*
 - Builder.Client_AbstractBuilder*: *Association*
 - Builder.Client_Director*: *Association*
 - Builder.Client_IProduct*: *Association*
 - Builder.ConcreteBuilder_Product*: *Association*
 - Builder.ConcreteBuilder_AbstractBuilder*: *Generalization*
 - Builder.Product_IProduct*: *Realization*
- (8)

The ontology fragment presented above has the form shown in the Fig. 5.

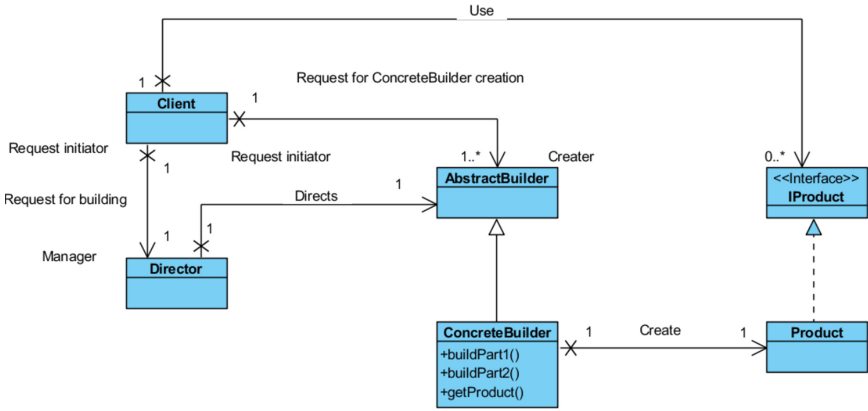


Fig. 4. Class diagram of design pattern “Builder”.

Realization of the similar project search function in the knowledge base use the metric to calculate the similarity between designed (in UML notation) and already implemented software projects.

4 Determining the Design Pattern Expression in the Information System Project

To calculate the similarity measure of software projects, the following method is proposed for calculating the measure of expression the design pattern in a software project:

$$\mu^{prj}(tmp_i) = \frac{|C^{prj} \cap C^{tmp_i}| + |R^{prj} \cap R^{tmp_i}|}{|C^{tmp_i}| + |R^{tmp_i}|}, \tag{9}$$

where $|C^{prj} \cap C^{tmp_i}|$ — number of matching individuals in an ontological representation $i - th$ knowledge base design pattern and ontological representation of a software project;
 $|R^{prj} \cap R^{tmp_i}|$ — number of matching relationships in an ontological representation $i - th$ knowledge base design pattern and ontological representation of a software project;
 $|C^{tmp_i}|$ — number of individuals in an ontological representation $i - th$ knowledge base design pattern and ontological representation of a software project;
 $|R^{tmp_i}|$ — number of relationships in an ontological representation $-th$ knowledge base design pattern and ontological representation of a software project.

If the number of facts ($|C^{tmp_i}|$ and $|R^{tmp_i}|$) ontological representation $i - th$ design pattern tmp_i determined by summing up the number of facts. To calculate number of facts ($|C^{prj}|$ $|R^{prj}|$) in ontological representation of a software project it is necessary to use the following developed algorithm:

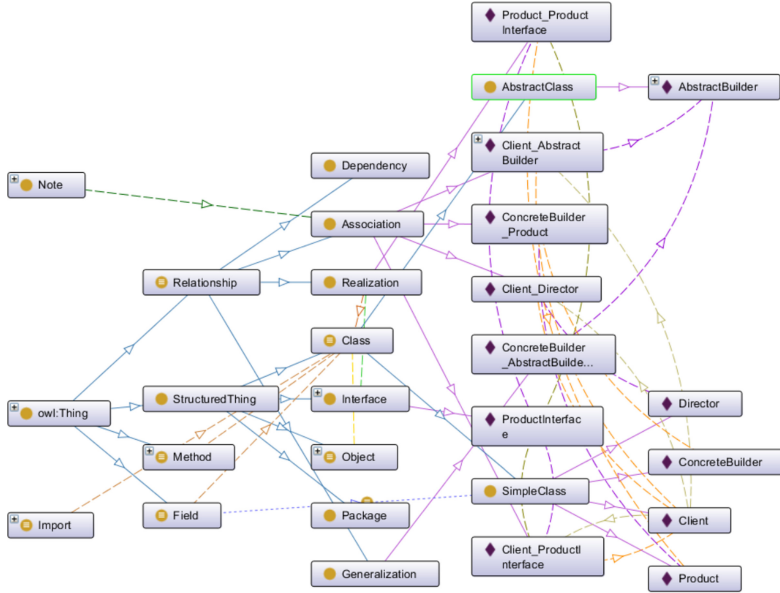


Fig. 5. An example of an ontological representation of a design pattern Builder

Step 1. Convert UML class diagram project $proj_j$ to set of facts $ABox^{prj}$:

$$\begin{aligned}
 & elem_k^{prj} : Concept \\
 & (elem_k^{prj}, elem_s^{prj}) : Role,
 \end{aligned}$$

where *Concept* — concept of the knowledge base, defined at $TBox$;

Role — role, defined at $TBox$;

$elem_k^{prj}$ — k – th individual of ontology concept, extracted from diagram.

Step 2. Defining a set of main classes from $ABox^{prj}$ regarding i – th design pattern tmp_i .

The base class will be such an individual $elem_k^{prj}$ of the concept *Class* (or its child concept *Subclass*) of $ABox^{prj}$, which corresponds to some individual $elem_k^{tmp}$: *Class* from $ABox^{tmp_i}$ for which a number of facts coinciding with the tmp_i pattern are maximum:

$$elem_k^{prj} : Concept (elem_k^{prj}, *) : Role \quad (*, elem_k^{prj}) : Role. \quad (10)$$

Step 3. Calculation of the number of true facts. The fact is true if there is a correspondence between i – th class individuals of the design pattern tmp_i and class diagrams of the project prj_j :

$$\forall k: elem_k^{tmp} \leftrightarrow elem_k^{prj}.$$

This algorithm of the class diagram indexing is performed for each design pattern available in the knowledge base ontology.

After calculating the measure of the expression of each selected design pattern in each of the considered software projects, it becomes possible to calculate the measure of similarity between software projects using one of three metrics.

5 Metrics Measures Architectural and Semantic Similarity of Software Projects

The first metric allows to calculate the measure of similarity by the most expressed pronounced design pattern in each of the projects:

$$\mu^1(prj_i, prj_j) = \bigvee_{tmp_k \in (prj_i \cap prj_j)} \mu^{prj}(tmp_k), \tag{11}$$

where prj_i, prj_j — UML class diagram ontological representation $i - th$ and $j - th$ projects respectively;
 $\mu^{prj}(tmp_k)$ — measure of expression $k - th$ design pattern in project (expression 9).

This metric demonstrates good results for a relatively small number of complex combined design patterns. Such design patterns are based on the subject area and, to a lesser extent, correspond to design patterns in the usual sense of industrial programming.

The second metric extends the first (the expression 11) and takes into account the degree of expression of design patterns that exceeds a certain threshold value. A threshold value of 0.3 was chosen experimentally. If the measure of the expression of the design pattern is less than 0.3, we can conclude that there is no design pattern in the software project, and as a result, such a design pattern should be excluded from consideration:

$$\mu^2(prj_i, prj_j) = \frac{\bigvee_{tmp_k \in (prj_i \cap prj_j) \geq 0.3} \mu^{prj}(tmp_k)}{N}, \tag{12}$$

where N — number of design patterns with expression measure more than 0.3 for each project.

The third metric is similar to the second metric (expression 12), but imposes an additional condition on the contribution of the measure of expressiveness of the design pattern ($\tilde{\mu}^{prj}$) to the measure of architectural similarity of projects:

$$\mu^3(prj_i, prj_j) = \frac{\bigvee_{tmp_k \in (prj_i \cap prj_j) \geq 0.3} \tilde{\mu}^{prj}(tmp_k)}{N}, \tag{13}$$

where $\tilde{\mu}^{prj}(tmp_k)$ — the weighted measure of expression design pattern tmp_k in software project prj .

The weighted measure of expression $\tilde{\mu}^{prj}(tmp_k)$ $k - th$ design pattern tmp_k in project prj is measure of expression (expression 9), normalized by number of elements, included in design pattern with maximum set of element:

$$\tilde{\mu}^{prj}(tmp_i) = \frac{|C^{prj} \cap C^{tmp_i}| + |R^{prj} \cap R^{tmp_i}|}{\sqrt{\sum_{tmp_k \in ABox} (|C^{tmp_k}| + |R^{tmp_k}|)}}, \quad (14)$$

This modification allows taking into account the complexity of the internal structure of the design pattern when calculating the similarity measures of software projects.

Design pattern consist from 20 elements that has full expression by $\mu^3(prj_i, prj_j)$ in projects $proj_i$ and $proj_j$, will have 4 times more weight than the design pattern, consisting of 5 elements and also having a degree of expression equal to 1.

6 Experiment in Finding Design Patterns in Public Projects from Github

To test the proposed approach to highlighting design patterns in projects, an experiment was conducted, the purpose of which was to search for design patterns in projects located in the GitHub repository. To conduct the experiment, information on 10 design patterns was added to the ontology: Delegation, Interface, Abstract superclass, Builder, Adapter, Singleton, Bridge, Faade, Decorator, Observer.

As a result of the “vk api” request, 108 projects were received related to the set of projects working with the social network API “VKontakte”. VKontakte is the most common social network in Russia.

The query by “design patterns” resulted in a sample of 6516 projects. This experiment is necessary to verify the operation of the system in conditions of increased content of design patterns in projects.

For testing, the sample was limited to the first 100 projects for both requests. Search results for design patterns are presented as bar graphs in the Figs. 6 and 7.

Selected design patterns differ in the number of elements and the relationships between them. The number of elements varies from 3 to 20.

In this experiment, only projects developed using the Java programming language were also considered.

Since the total number of projects in the GitHub repository developed in the Java language is very large, it is necessary to limit the selection of projects for the experiment. As a result, the following results were obtained: high frequency of use of the Delegation, Interface, Abstract superclass and Facade templates. This result is explained by the simple structure of these patterns — a relatively small number of structural elements and, as a result, relationships. These design patterns may have been used unconsciously by developers, or they may coincide in structure with part of a more complex pattern.

There were relatively few design patterns for Builder, Adapter, Bridge, Decorator, and Observer in the control group of projects. The rarity of these patterns is due to their complex structure — the content of a large number of elements.

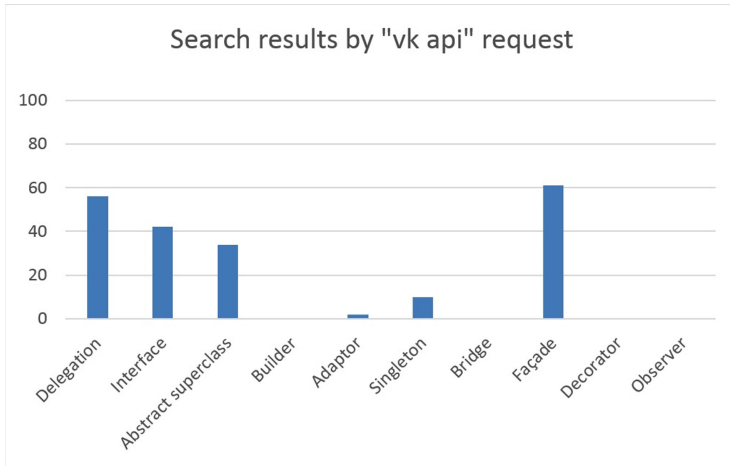


Fig. 6. Search results for templates among projects received by request “vk api”

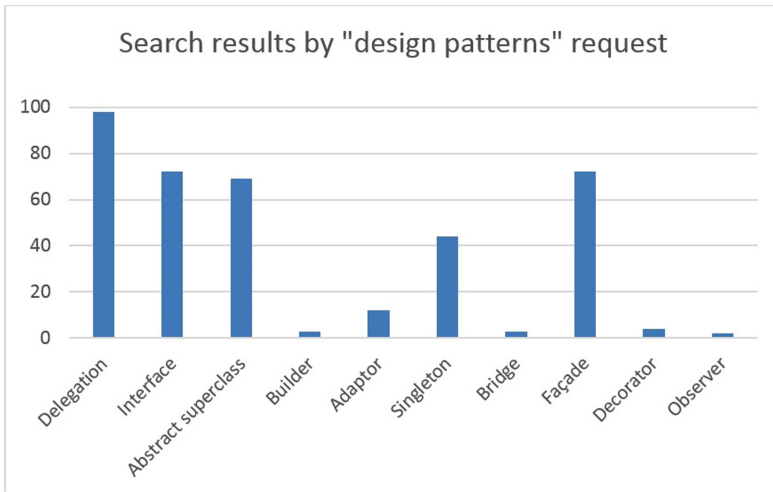


Fig. 7. Search results for templates among projects received by request “design patterns”

7 Search Experiments Results for Structurally Similar Software Projects

To determine the measure of structural similarity between two projects, it is necessary to calculate the measure of severity of each design pattern in both projects (expressions 9 and 14).

In this experiment, all projects were downloaded from the GitHub open repository. All projects were selected by the following keywords: “public API”,

“social network”, “vkontakte”, which allows to determine whether the projects belong to the subject area — work with the social network API “VKontakte”: Android-MVP, cordova-social-vk, cvk, DroidFM, VK-Small-API, VKontakteAPI, VK_TEST.

The Table 1 shows the severity of each considered design pattern in all projects of the experimental sample. The similarity score ratings are normalized from 0 to 1. The estimates of the similarity measure for the first metric

Table 1. The expression measure of design patterns in projects

Project name	Delegator	Adapter	Builder	Abstract superclass	Interface
Android-MVP	1.0	0.875	0.83	1.0	1.0
cordova-social-vk	1.0	0.875	0.83	1.0	0.8
cvk	1.0	0.875	0.92	1.0	1.0
DroidFM	1.0	0.875	0.92	1.0	1.0
VK-Small-API	1.0	0.42	0.92	0.33	0.6
VKontakteAPI	1.0	0.83	0.92	1.0	0.8
VK_TEST	1.0	0.75	0.58	0.66	0.6

(expression 11) are always equal to 1, because this metric selects a design pattern with the maximum measure of expression for each of the two compared projects. And since, for example, the Abstract superclass, Interface, and Delegator design pattern consists of a relatively small number of elements, this leads to a high degree of expression of such patterns in a large number of projects.

The estimates of the projects similarity measure by second (expression 12) and third (expression 13) metric presented at 2 and 3 tables respectively.

Table 2. Measures of structural similarity of software products in the second metric

Projects	Android-MVP	cordova-social-vk	cvk	Droid-FM	VK-Small-API	VKontakteAPI	VK_TEST
Android-MVP	—	0.96	0.96	0.98	0.78	0.96	0.78
cordova-social-vk	0.96	—	1	0.94	0.85	1	0.83
cvk	0.96	1	—	0.94	0.85	1	0.83
DroidFM	0.98	0.94	0.94	—	0.78	0.94	0.78
VK-Small-API	0.78	0.85	0.85	0.78	—	0.85	0.96
VKontakteAPI	0.96	1	1	0.94	0.85	—	0.83
VK_TEST	0.79	0.83	0.83	0.78	0.95	0.83	—

Table 3. Measures of structural similarity of software products in the third metric

Projects	Android-MVP	cordova-social-vk	cvk	Droid-FM	VK-Small-API	VKontakteAPI	VK_TEST
Android-MVP	—	0.96	0.96	0.96	0.64	0.96	0.77
cordova-social-vk	0.96	—	0.99	0.93	0.67	0.99	0.80
cvk	0.97	0.99	—	0.93	0.67	0.99	0.80
DroidFM	0.97	0.93	0.93	—	0.61	0.93	0.74
VK-Small-API	0.64	0.67	0.68	0.61	—	0.67	0.87
VKontakteAPI	0.97	0.99	0.99	0.93	0.67	—	0.80
VK_TEST	0.77	0.80	0.80	0.74	0.87	0.80	—

The degree of similarity of the projects obtained in these experiments are very high, which can be explained by two features of this experiment. Design patterns with a severity measure of less than 0.3 were excluded in at least one of the compared projects. In this experiment, it is assumed that the design pattern, expressed with a measure of expression less than 0.3, is not found in the project. Accounting design patterns with a small degree of severity will lead to a significant decrease in the value of the similarity indicator of any projects with an increase in the number of design patterns.

The considered metrics for calculating project similarity are based on a single computational principle and represent its consistent development. The third metric is the most universal for projects and design patterns of different sizes but much more parametrized.

Design patterns can be implemented in projects in various ways. This problem can be solved in two ways:

1. Using the corporate standard of the company
2. To use projects from open sources, it is worthwhile to form two or more alternative representations of the design pattern in an ontology and consider them equivalent

8 Conclusion

In the course of this research, the following results were obtained:

1. Ontologically-oriented model of the UML diagram language and ontological model of the design pattern.
2. Architectural similarity measures for software projects; measures of expressiveness of the design pattern in the considered software products.
3. An algorithm for transforming a class diagram in UML notation into an ontology of the OWL format.
4. An algorithm for transforming source code in the Java programming language into an ontology of the OWL format.

Thus, the proposed approach to supporting the design process allows the use of successful design solutions in the development of new software project, thereby reducing the design process time and increasing the quality of the resulting solutions.

References

1. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide, 2nd edn. Addison-Wesley Object Technology Series, p. 496, New York (2005)
2. Bobillo, F., Straccia, U.: Fuzzy ontology representation using OWL 2. Approximate Reasoning **52**(7), 1073–1094 (2010)
3. Guarino, N., Musen, M.A.: Ten years of applied ontology. Appl. Ontol. **10**, 169–170 (2015)
4. OWL 2 Web Ontology Language. <https://www.w3.org/TR/owl2-overview/>
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, Peter F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
6. Bonatti, P.A., Tettamanzi, A.G.B.: Some complexity results on fuzzy description logics. In: Di Gesù, V., Masulli, F., Petrosino, A. (eds.) WILF 2003. LNCS (LNAI), vol. 2955, pp. 19–24. Springer, Heidelberg (2006). https://doi.org/10.1007/10983652_3
7. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: the making of a web ontology language. J. Web Semant. **1**(1), 7–26 (2003)
8. Groszof, B., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logics. In: Proceedings of WWW 2003, Budapest, Hungary, pp. 48–57. ACM, May 2003
9. Golenkov, V., Guliakina, N., Davydenko, I.: Methods and tools for ensuring compatibility of computer systems. Open Semant. Technol. Intell. Syst. **3**, 25–53 (2019)
10. Golenkov, V., Shunkevich, D., Davydenko, I.: Principles of organization and automation of the semantic computer systems development. Open Semant. Technol. Intell. Syst. **3**, 53–91 (2019)
11. Gao, M., Liu, C.: Extending OWL by fuzzy description logic. In: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2005), pp. 562–567 (2005)
12. Bianchini, D., de Antonellis, V., Pernici, B., Plebani, P.: Ontologybased methodology for e-service discovery. Inf. Syst. **31**, 361–380 (2005)
13. Guizzardi, G., Guarino, N., Almeida, J.P.A.: Ontological considerations about the representation of events and Endurants in business models. In: International Conference on Business Process Management, pp. 20–36 (2016)
14. Falbo, R.A., Quirino, G.K., Nardi, J.C., Barcellos, M.P., Guizzardi, G., Guarino, N.: An ontology pattern language for service modeling. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 321–326 (2016)
15. Hotho, A., Staab, S., Stumme, G.: Ontologies improve text document clustering data mining. ICDM **2003**, 541–544 (2003)

16. Gruber, T.: Ontology. <http://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf>. Accessed Dec 2019
17. Maedche, A., Staab, S.: Ontology learning for the Semantic Web. <https://www.csee.umbc.edu/courses/771/papers/ieeeIntelligentSystems/ontologyLearning.pdf>. Accessed Dec 2019
18. Guskov, G., Namestnikov, A., Yarushkina, N.: Approach to the search for similar software projects based on the UML ontology. In: Abraham, A., Kovalev, S., Tarassov, V., Snasel, V., Vasileva, M., Sukhanov, A. (eds.) IITI 2017. AISC, vol. 680, pp. 3–10. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-68324-9_1