



Article

# An Approach to Determining Software Projects with Similar Functionality and Architecture Process Based on Artificial Intelligence Methods

Nadezhda Yarushkina <sup>1,†</sup> , Gleb Guskov <sup>1,†,\*</sup>  and Pavel Dudarin <sup>1,†</sup> 

<sup>1</sup> Severny Venets 32, Ulyanovsk, Russia, 432027, Ulyanovsk State Technical University; jng@ulstu.ru

\* Correspondence: guskovgleb@gmail.com; Tel.: +7-927-987-2636

† These authors contributed equally to this work.

**Abstract:** Software engineers from all over the world solve independently a lot of similar problems. In this condition the problem of code or even better architecture reusing becomes an issue of the day. In this paper two phase approach to determining the functional and structural likenesses of software projects is proposed. This approach combines two methods of artificial intelligence: natural language processing techniques with a novel method for comparing software projects based on ontological representation of their architecture automatically obtained from the projects source code. Additionally several similarity metrics are proposed to estimate similarity between projects.

**Keywords:** ontology; conceptual model; natural language processing; engineering design; fuzzy hierarchical classifier; clustering

## 1. Introduction

Well known that human resources in modern software development are the most valuable. Nevertheless often happens that the same tasks are solved by independent engineers multiple times, this leads to an ineffective software development process organization. There are approaches for reusing source code at various stages of development. Basically, these approaches allow to reuse certain functions and classes only and do not allow reveal the projects similarity on the basis of their subject area.

Knowledge of architecture gained from already implemented projects in the same subject area could allow borrowing and reusing much larger parts of projects and avoiding conceptually incorrect solutions in the future. Quite often with a change of project developers team the implemented software solutions are forgotten and not reused.

A tool capable of determining similarities between projects can be very useful in software development. This tool can not be replaced by any version control system, because it has completely different purpose. Version control system provides storage of all the project versions with comments and ability to compare file versions. Meanwhile proposed tool could store, explain and compare projects structure. Such an instrument could be able to work not only with projects from single organization, but also with projects from open repositories owned by authors from all over the world.

Search on open repositories is carried out on the basis of keywords. This search could result in returning several thousand projects, which can not be handled by hand. The choice of projects set based on project subject matter and inner architectural solutions is a promising approach. Project subject matter could be obtained by analyzing project description which is usually done in "readme.txt" file in versions control system repository and by scanning issue forums of the project. This task requires implementation of natural language processing (NLP) which is widely used in artificial intelligence and data science world and there are a lot of tools to perform NLP procedures for such a popular programming languages as Python, R, Java, C#. An interesting approach to forum analysis could be found in [1]. The state of the art technique in this area is word2vec models of natural language [2,3].

36 Nowadays there are many available pre-trained word2vec models [4,5], so there only thing to do is  
37 additionally tune word2vec model for specific task.

38 Further, to implement projects comparison based on their structure a tool for architectural concept  
39 extraction is needed. Commonly, the software project architecture is built at the design stage, which is  
40 prior to the development one. The UML language were developed to describe the project architecture  
41 with the required abstraction level. Based on the results of our previous research [6], could be concluded  
42 that developers use a lot of different types of structural elements. That is why ontology as a knowledge  
43 storage system could well act as a reference for the project analysis tool. Attempts to integrate  
44 ontologies into software development were carried out at different levels: technical documents [7–11],  
45 maintenance and testing of the source code [12], UML diagrams [13–15].

46 The minimal structural UML diagram elements, such as classes, interfaces, objects themselves,  
47 weakly convey the semantics and architectural solutions of the project. But combination of such an  
48 elements is much better describes the architecture. Stable combinations of structural elements are  
49 known as design patterns, this term exists in information technology for a long time but it is still  
50 relevant. Design patterns are actively used by the developer community, thus representing a reliable  
51 benchmark in the software project analysis. In addition, it makes sense to create local design patterns  
52 that solve specific task in a given subject area. A design pattern based on a specific subject area loses its  
53 main advantage - universality, but its greater semantic weight becomes more important characteristic  
54 for solving the problem of tool construction for searching and measure similarities between projects.  
55 There are many works devoted to the integration of software development with ontologies. There is a  
56 complete approach to development based on a domain known as development based on the subject  
57 area [16–18].

58 The rest of this paper is organized as follows. In section 2 the detailed problem formulation is  
59 presented. Section 3 dedicated to the preliminary filtering of software projects. Then in section 4 the  
60 construction of software design ontology technique is presented step by step. The experimental results  
61 are discussed in Section 5. Section 6 concludes the paper.

## 62 2. Formulation of the problem

63 The results presented in this paper are based on the work described in research [6]. The system  
64 described in that research made it possible to extract information from conceptual models and save  
65 it as an ontology of a certain format. But the life cycle and development practices of IT companies  
66 show that conceptual models are created mostly once at the beginning of the project and in rare cases  
67 updated at the beginning of each stage of the project.

68 The best description of the project state is its source code. Developers try to maintain the source  
69 code in a good condition, create documentation, provide comments and perform code refactoring.  
70 Another advantage of using source code as a source of project state information is the fact of wide  
71 using of version control systems. Tracking all the versions of software products allows effectively  
72 manage the software development process and generate a huge amount of information available for  
73 analysis.

74 Comparison of information obtained from conceptual models of a new project at the design stage  
75 and information obtained from the source code of projects that have already been implemented could  
76 allow to determine projects structural similarity.

77 In order to be able to analyze and measure the projects structural similarity, it is necessary to  
78 transform information about projects from different sources to a single format. The most convenient  
79 way of presentation of the extracted information is a form of ontology using OWL format. OWL  
80 ontology format allows to preserve semantics of complex architectural solutions, to modify already  
81 existing data and to perform logical operations on statements.

82 The search for project structural similarity is part of the project comparison method. Another part  
83 of this method is projects filtering base on subject areas. As long as information about software project  
84 collected from open sources and there is no common tag system or any commonly used classifier,

85 the only way to group project is to perform clustering procedure. This part is performed by using a  
 86 combination of approaches suggested in papers [19,20] for short text clustering based on semantic  
 87 similarity measure obtained from word2vec pre-trained model.

88 If a comparison is made for projects of the same enterprise in the same subject area, then the  
 89 comparison should be performed at the level of the processes and the components of the subject area.  
 90 In case when project comparison is carried out among projects hosted on any open repository, the  
 91 structural similarity of the projects could be more important metric than specific subject area, that is  
 92 why filtering based on NLP used just for preliminary selection of possibly relevant software projects.  
 93 In such a condition the precision of filtering method is not very important but the convenience of  
 94 the result presentation way is vital for further expert analysis. The detailed explanation of suggested  
 95 approach is done in the next section.

### 96 3. Software projects preliminary filtering

#### 97 3.1. Fuzzy hierarchical classifier constructing

98 In order to construct fuzzy hierarchical classifier an approach from paper [20] was adopted. This  
 99 algorithm takes as an input a set of sentences (short text fragments), for this task project descriptions,  
 100 forum discussions of the project, set of comments from source code, etc. could be taken as sentences.  
 101 For the experiment we have selected projects with a set of keywords ("api", "java", "mobile", "sdk") from  
 102 source code repositories like GitHub, GitLab and others. In total there were 490 projects in the input  
 103 dataset, and general project description has been taken as sentences. As long as there is not possibility  
 104 to show all the input data, in Table 1 could be found the most demonstrative samples translated into  
 105 English. Some of them are obviously appropriate, and after the clustering procedure will be searched  
 106 for design patterns. While the others are obviously are not appropriate and will be moved to different  
 107 clusters and thus filtered out.

108 Each sentence was tokenized and lemmatized. Resulting terms were organized in fuzzy graph  
 109 based on semantic similarity measure obtained from pre-trained word2vec model. And, finally, with a  
 110 help of hierarchical fuzzy graph  $\epsilon$ -clustering algorithm a fuzzy hierarchical classifier was obtained,  
 111 see Figure 1. On this figure only a two pieces of hierarchy are presented for the reason of space and  
 112 clear visibility. These two sub-hierarchy shows two semantically related groups of words, the first one  
 113 for the programming and API and the second one for the music. In the clustering result part below  
 114 will be shown that these two groups lead to forming clusters dedicated to VK APIs and VK Players  
 115 respectively.



Figure 1. Extract from hierarchical classifier of software project terms

**Table 1.** A set of sentences for software projects preliminary filtering

| Project name                      | Sentence  |
|-----------------------------------|---|
| VKCOM/<br>vk-java-sdk             | Java library for VK API interaction, includes OAuth 2.0 authorization and API methods. Full VK API features documentation can be found here. This library has been created using the VK API JSON Schema. It can be found here. It uses VK API version 5.69.   |
| dewarder/<br>HoldingButton        | Button which is visible while user holds it. Main use case is controlling audio recording state (like in Telegram, Viber, VK).  |
| korobitsyn/<br>VKOpenRobot        | VK Open Bot is a library for bot creation for VK social network. Main features: mass friends collection, mass group searching and aggregation, user detailed information, user status detection   |
| gleb-kosteiko/<br>vkb             | Script allows you to automate the searching and participation in random reposts competitions in vk.com. Check friends walls for competition posts and repost these posts (also joined all needed communities and added all needed users to friends). Do reposts for simulation of the real user behavior. Search competition posts in VK.                           |
| PhoenixDevTeam/<br>Phoenix-for-VK | First open-sourced VK client for Android inspired by Material Design.   |
| petersamokhin/<br>vk-bot-java-sdk | Comfortable and simple library for creating bots for VK   |
| strelnikovkirill/<br>VKPhotoApp   | Android OS + VK Api. VK application for a surfing in user news feed, but this news feed build only on posted photos.  |
| vladgolubev/<br>nowplayingVk      | This app broadcasts currently playing song from last.fm account to vk.com status  |
| shavkunov/<br>vk-analyzer         | Application used to analyze wall of VK user or community and save results to internal database.   |
| asaskevich/<br>VK-Small-API       | Small Java API used for work with VK. Example of using is in VK Example.java. Authorization Counters of new messages, friends, answers and groups Total count of friends. Loading basic data about friends id, name, photo. Friends' status. Send private message to chat or user. Load list of dialogs. Load list of groups  |
| akveo/<br>cordova-vk              | You can use this plugin to authenticate user via VK application rather than via webview. It makes use of official VkSDKs for iOS and Android. This project is based on another github project <a href="https://github.com/DrMoriarty/cordova-social-vk">https://github.com/DrMoriarty/cordova-social-vk</a> . But Api was made a bit more generic to fit our needs. |
| MLSDev/<br>DroidFM                | This application shows how you can integrate the RxJava, Realm, LastFM API, VK API for information on popular artists, their songs and albums. Additionally this app can stream and download any tracks that are available in VK.   |
| Try4W/<br>VKontakteAPI            | Simple, light weighted binding VK API for Java based on official Android SDK  |

### 116 3.2. Feature construction

Obtained classifier could be treated as a data source to construct feature vectors. The exact algorithm was described in [19]<sup>1</sup>. A function described below was used to transform sentences into vector form.

$$F : SL \rightarrow \mathbb{R}^n, \forall s \in SLF(s) = (s_1, s_2, \dots, s_n), \quad s_i = R(s, v_i) * \frac{1}{\omega(v_i) * \sigma * \sqrt{2 * \pi}} * e^{-\frac{(\ln \omega(v_i) - \mu)^2}{2 * \sigma^2}},$$

where  $SL$  is a lemmatized set of sentences,  $R$  is a membership degree of sentence in current fuzzy classifier vertex.

$$\omega(v_i) = ||s \in SL | R(s, v_i) > 0||, n \in [1, ||HV||], v_i \in HV$$

117 with  $HV$  is an obtained classifier, parameter settings are:  $\sigma = 1, \mu = 2.4$  in order to make function  
 118 global maximum equal to 4 repetition of the word in the dataset. The next step is to find the most  
 119 appropriate groups of similar software projects.

### 120 3.3. Projects clustering

121 To perform clustering procedure an HDBScan [21] algorithm has been chosen. This clustering  
 122 algorithm combined with features constructed from hierarchical classifier hierarchical classifier has  
 123 been chosen for its ability to return quite accurate and pure clusters, mainly at the expense of precision.  
 124 Metrics for precision, accuracy and purity are defined for each class as follows:

$$Precision_i = \frac{\max_j CM_{i,j}}{\sum_j CM_{i,j}} \quad Accuracy_i = \frac{\max_j CM_{i,j}}{\sum_k CM_{k,j_{\max}}} \quad Purity_i = \frac{\max_j CM_{j,i}}{\sum_j CM_{j,i}},$$

125 where  $CM = [cm_{i,j} = (\omega_i \cap c_j), \omega_i$  - class with number  $i, c_j$  - cluster with number  $j$ ].

126 General precision, accuracy and purity are calculated as average values.

127 This method has shown quite a good performance results for a similar task which is discussed in  
 128 paper [19].

129 During the clustering process 15 clusters we determined, main clusters with general description  
 130 are presented on the Figure 2. HDBScan algorithm was performed with parameter  $min\_cluster\_size =$   
 131 5. Algorithm HDBScan as a true clustering algorithm always forms 'noise cluster' with number -1,  
 132 where all the samples that could not be grouped are moved to.

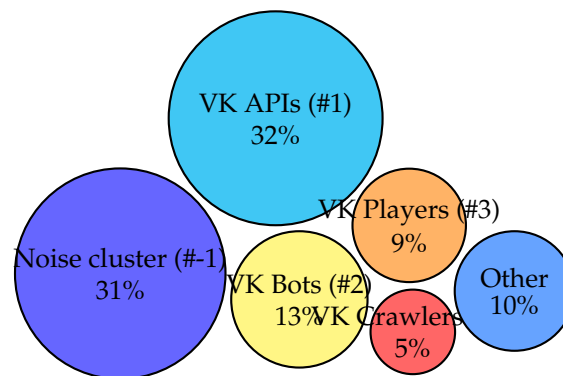


Figure 2. Clustering results

<sup>1</sup> Software implementation hosted on <https://github.com/PavelDudarin/sentence-clustering>

133 In our example the 'noise cluster' is quit big, the main reason for this are too poor projects'  
 134 description and high dimensional clustering features. More detailed discussion for this could be found  
 135 in [19]. Future studies will be dedicated to solving this problem.

136 Clustering labels for previously shown sentences could be found in Table 2. As it could be  
 137 seen projects are grouped quite accurate. Calculated quality metrics based on expert evaluation are  
 138 following:  $precision = 0.7$ ,  $accuracy = 0.98$ ,  $purity = 0.98$ . High level of accuracy and purity quite  
 139 important in cases when expert evaluation follows the clustering process and homogeneous clusters  
 140 are preferred to precise ones. In our case clustering has been made to facilitate the process of software  
 141 projects filtering, but the final decision is on expert.

**Table 2.** Sample of clustering results

| Project Name                   | Cluster |
|--------------------------------|---------|
| VKCOM/ vk-java-sdk             | 1       |
| dewarder/ HoldingButton        | -1      |
| korobitsyn/ VKOpenRobot        | 2       |
| gleb-kosteiko/ vkb             | 2       |
| PhoenixDevTeam/ Phoenix-for-VK | -1      |
| petersamokhin/ vk-bot-java-sdk | 2       |
| strelnikovkirill/ VKPhotoApp   | 1       |
| vladgolubev/ nowplayingVk      | 3       |
| shavkunov/ vk-analyzer         | 4       |
| asaskevich/ VK-Small-API       | 1       |
| akveo/ cordova-vk              | 1       |
| MLSDev/ DroidFM                | 3       |
| Try4W/ VKontakteAPI            | 1       |

142 A software projects selection strategy for the next phase could be different. The first strategy is to  
 143 choose one example from each cluster in order to have a good variety of possible architectures. The  
 144 opposite strategy is to chose a few entire clusters in case of getting some clusters that generally satisfy  
 145 business purposes of a new software project. In the experiment part of this paper the second strategy  
 146 has been chosen. For the further processing sentences from clusters with numbers 1, 2 and 3 have been  
 147 chosen.

#### 148 4. Software design ontology

149 If class diagram for software was built during design stage the structure analysis could be done.  
 150 To complete this task ontology design approach was used and described below.

##### 151 4.1. UML meta-model based ontology

152 As a target for storing knowlege from UML class diagrams has been chosen an OWL ontology  
 153 format. OWL was chosen because this format is the most expressive in terms of representation of  
 154 knowledge for complex subject areas. The class diagram elements should be translated into ontology  
 155 as concepts with their semantics consideration. Semantics of the whole diagram is being formed from  
 156 the semantics of the diagram elements and the semantics of their relations. That is why the ontology  
 157 was built on the basis of the UML meta-scheme, and not as a formal set of translated elements.

158 To solve the problem of intellectual analysis of project diagrams that ware included in the project  
 159 documentation, it is necessary to have knowledge about formalized diagrams constructing.

160 Ontology contains concepts that describe the most basic elements of the class diagram, but it  
 161 could be expanded if necessary. During translation of the UML meta-scheme the following notations  
 162 were applied.



Formally, the ontology of project diagrams is represented as a set:

$$O^{prj} = \langle C^{prj}, R^{prj}, F^{prj} \rangle, \quad (1)$$

163 where :  $C^{prj} = \{c_1^{prj}, \dots, c_i^{prj}\}$  – is a set of concepts that define main UML diagram elements such as :  
 164 "Class", "Object", "Interface", "Relationship" and others;  
 165  $R^{prj}$  – the set of connections between ontology concepts. These relationships allow to describe correctly  
 166 rules of UML notation.  
 167  $F^{prj}$  – is the set of interpretation functions defined on the relationships  $R^{prj}$   
 168

#### 169 4.2. Design patterns as structural parts of software projects

170 Design patterns are inserted into ontology as a set of individuals based on the ontology concepts  
 171 described above. Semantic constraints and properties of design patterns are specified by the  
 172 ObjectProperties and DatatypeProperties of OWL ontology. Since many design patterns are stored in  
 173 the ontology at the same time, it is necessary to define naming convention for their elements to avoid  
 174 names duplication. Name of the design pattern element begins with the design pattern name, and then  
 175 if the element is a class, its name is written. If the element is a relationship, then names of connecting  
 176 elements are written with underscore symbol between them. One of the most commonly used design  
 177 patterns is the Builder [22].

178 Builder is a creation desing pattern. This desing pattern separates the algorithm for the  
 179 step-by-step construction of a complex object from its external representation to make it possible  
 180 to have different representations of this object using the same algorithm.

181 In order to preserve this design pattern in the developed ontology, the following individuals  
 182 belonging to relevant concepts are required.

- 183 • SimpleClass: Builder\_Client, Builder\_Director, Builder\_ConcreteBuilder, Builder\_Product.
- 184 • AbstractClass: Builder\_AbstractBuilder.
- 185 • Association: Builder\_Client\_AbstractBuilder, Builder\_Client\_Director, Builder\_Client\_IProduct,  
 186 Builder\_ConcreteBuilder\_Product.
- 187 • Generalization: Builder\_ConcreteBuilder\_AbstractBuilder.
- 188 • Realization: Builder\_Product\_IProduct.

189 Ontological representation of the design pattern:

$$O_{tmp_i}^{prj} = \{inst(C_1^{prj}), \dots, inst(r_1^{prj}), \dots, r_{sameAs}\}, \quad (2)$$

190 In fact, the ontological representation of a single design pattern is a set of individuals of concepts  
 191 and relations from the ontology of project diagrams.

To calculate the structural similarity of projects based on developed ontology, the following  
 evaluation functions were proposed. The first metric gives priority to the maximum single expressed  
 design pattern in both diagrams:

$$\mu_{dc_\gamma, dc_\delta} = \bigvee_{tmp \in (dc_\gamma \cap dc_\delta)} \mu_{dc_\gamma \cap dc_\delta}(tmp), \quad (3)$$

192 where  $dc_\gamma$  and  $dc_\delta$  is projects class diagrams presented as UML metamodel ontology Abox expressions,  
 193  $\mu_{dc_\gamma, dc_\delta}(tmp)$  - measure of expression the design pattern in project diagram.

194 The second metric considers the coincidence of all design patterns in equal proportions and does not  
 195 considers design patterns with a measure of expression less than 0.3:

$$\mu_{dc_\gamma, dc_\delta} = \left( \sum_{tmp \in (dc_\gamma \cap dc_\delta) \geq 0.3} \mu_{dc_\gamma \cap dc_\delta} \right) / N, \quad (4)$$

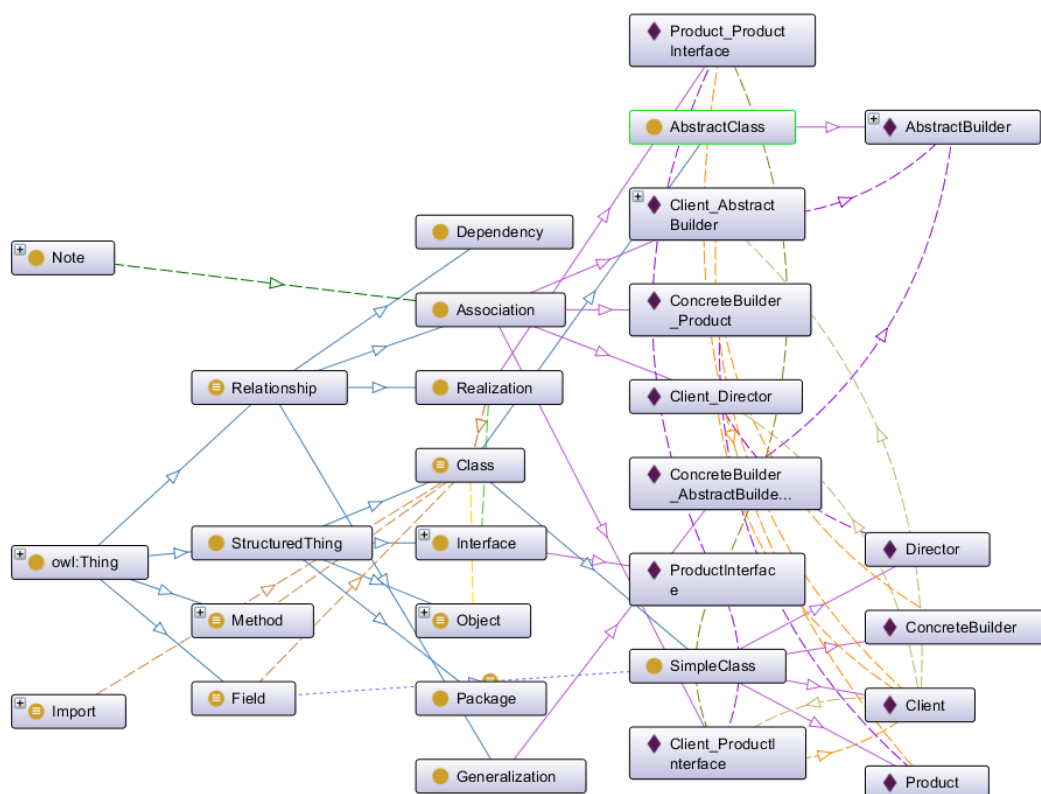


Figure 3. "Builder" design pattern ontology presentation in Protege editor.

196 where  $N$  - count of design patterns with a measure of expression greater than 0.3 for both of  
 197 projects.

198

199 The third metric works in the same way as the second one, but the contribution to the evaluation  
 200 by design patterns depends on the number of elements in the design pattern (the design pattern with  
 201 20 elements means more than a design pattern with 5 elements):

$$\mu_{dc_\gamma, dc_\delta} = \left( \sum_{tmp \in (dc_\gamma \cap dc_\delta) \geq 0.3} \tilde{\mu}_{dc_\gamma \cap dc_\delta} \right) / N, \quad (5)$$

202 where  $\tilde{\mu}_{dc_\gamma \cap dc_\delta}$  - weighted measure of expression.

203

## 204 5. The results of searching structurally similar software projects

### 205 5.1. Searching design patterns in projects

206 To determine the measure of similarity between two projects, it is necessary to calculate an  
 207 expression degree for each design pattern in each project. The expression measure of the design pattern  
 208 in the project can be calculated by mapping a project ontology Abox on a design pattern ontology  
 209 Abox. The Table 3 contains expression degree for each design pattern in each project.

### 210 5.2. Results of searching structurally similar software projects by different metrics

211 This estimations are normalized from 0 to 1. For the first metric estimations are always equal to  
 212 1. This could be easily explained because first metric chooses the most expressed design pattern in



**Table 3.** Expression of design patterns in projects

| Project name / Design pattern name | Delegator (3) | Adapter (8) | Builder (12) | Abstract superclass (3) | Interface (5) |
|------------------------------------|---------------|-------------|--------------|-------------------------|---------------|
| 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       | 0.83         | 1.0                     | 0.8           |
| DroidFM                            | 1.0           | 0.875       | 0.92         | 1.0                     | 1.0           |
| VK-Small-API                       | 1             | 0.625       | 0.42         | 0.33                    | 0.6           |
| VKontakteAPI                       | 1.0           | 0.875       | 0.83         | 1.0                     | 0.8           |
| VK_TEST                            | 1             | 0.75        | 0.58         | 0.66                    | 0.6           |

**Table 4.** Similarity between projects by second metric

| Project 1 / Project 2 | android-mvp | cordova-vk | cvk  | droidfm | vk-small-api | vkontakteapi | vk_test |
|-----------------------|-------------|------------|------|---------|--------------|--------------|---------|
| android-mvp           | –           | 0.96       | 0.96 | 0.98    | 0.78         | 0.96         | 0.78    |
| cordova-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 5.** Similarity between projects by third metric

| Project 1 / Project 2 | android-mvp | cordova-vk | cvk  | droidfm | vk-small-api | vkontakteapi | vk_test |
|-----------------------|-------------|------------|------|---------|--------------|--------------|---------|
| android-mvp           | –           | 0.96       | 0.96 | 0.96    | 0.64         | 0.96         | 0.77    |
| cordova-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         | –       |

213 both projects. Among the design patterns participating in the test there design patterns with a small  
 214 number of elements, for example: Abstract superclass, interface and delegator. Such design patterns  
 215 were have measure of experssion equal 1 by first metric at both compared project. The estimations of  
 216 similarity calculation between projects by second and third metrics are presented in the Tables 4 and 5,  
 217 respectively.

218 The results for the second and third metrics are also quite high. Design patterns with a expression  
 219 degree less than 0.3 were excluded from consideration. All projects that participated in the comparison  
 220 are downloaded from the open repository Github and realize interaction with the public API of the  
 221 russian well-known social network vkontakte or with it's music or mobile API.

222 High estimations of similarity by second and third metrics are explained by applying the results  
 223 of research first part an NLP procedures. Selection of projects was carried out by NLP procedures  
 224 based on information obtained from: readme files, source code comments, version control system  
 225 comments and issues excludes projects from other subject areas.

## 226 6. Conclusions

227 In this paper two phase approach of artificial intelligence to determining the functional and  
228 structural similarity of software projects is presented. NLP analysis and ontology construction allow  
229 to find and investigate projects with similar purposes and architecture. In the experimental part the  
230 proposed method was applied to different projects and proposed several similarity metrics to measure  
231 similarity between projects. Moreover, the work presented in this paper have great potential for further  
232 research. Number of projects could be expanded. It is possible to include new design patterns in  
233 consideration. Ontologies obtained in the intermediate stages could be used separately in Protege  
234 editor. The results of this research correspond to the artificial intelligence and can be used to create  
235 intellectual systems. Expanding the system by using ontologies of subject areas can significantly  
236 increase the relevance of the similar projects selection.

237 **Acknowledgments:** The study was conducted with the support of the Ministry of Education and Science of the  
238 Russian Federation within the framework of the theme 2.1182.2017 / 4.6.

239 **Author Contributions:** All authors have made significant contributions to the paper. Nadezhda Yarushkina  
240 carried out a literature survey. Pavel Dudarin proposed the method of projects clustering by subject area keywords.  
241 Guskov Gleb suggested an ontology scheme for software projects and implemented an import algorithm from  
242 source code and conceptual models.

243 **Conflicts of Interest:** The authors declare no conflict of interest.

## 244 Abbreviations

245 The following abbreviations are used in this manuscript:

|     |     |                             |
|-----|-----|-----------------------------|
| 246 | OWL | Ontology web language       |
| 247 | UML | Unified modeling language   |
|     | NLP | Natural Language Processing |

## 248 References

- 249 1. Han, X.; Ma, J.; Wu, Y.; Cui, C.; A novel machine learning approach to rank web forum posts. In *Soft*  
250 *Computing* **2008**, Springer:Berlin/Heidelberg Volume 18, Issue 5, pp. 941–959.
- 251 2. Tomas Mikolov , Ilya Sutskever , Kai Chen , Greg Corrado , Jeffrey Dean *Distributed representations of words*  
252 *and phrases and their compositionality*, Proceedings of the 26th International Conference on Neural Information  
253 Processing Systems, p.3111-3119, December 05-10, 2013, Lake Tahoe, Nevada
- 254 3. Quoc Le, Tomas Mikolov *Distributed Representations of Sentences and Documents*, Proceedings of the 31st  
255 International Conference on Machine Learning, PMLR 32(2):1188-1196 (2014)
- 256 4. Kutuzov, A.; Kuzmenko, E.; WebVectors: A Toolkit for Building Web Interfaces for Vector Semantic Models.  
257 In Proceedings International Conference on Analysis of Images, Social Networks and Texts (AIST 2016),  
258 Springer:Cham, Moskow, Russian Federation, 27-29 July 2017, Volume 661. pp 155-161.
- 259 5. Pelevina, M.; Arefyev, N.; Biemann, C.; Panchenko, A.; Making Sense of Word Embeddings. In Proceedings  
260 of the 1st Workshop on Representation Learning for NLP co-located with the ACL conference. Berlin,  
261 Germany, 10 August 2017; arXiv:1708.03390.
- 262 6. Guskov, G.; Namestnikov, A.; Yarushkina, N.; Approach to the Search for Similar Software Projects Based  
263 on the UML Ontology. In *Proceedings of the Second International Scientific Conference "Intelligent Information*  
264 *Technologies for Industry" (IITI 2017)*, Springer:Cham, Varna, Bulgaria, 14-16 September 2017; Volume 680, pp  
265 3-10.
- 266 7. Namestnikov, A.; Filippov, A.; Avvakumova, V.; An ontology based model of technical documentation  
267 fuzzy structuring. In *proceedings CEUR Workshop. SCAKD 2016*, Moscow, Russian Federation, 18-22 July 2016;  
268 Volume 1687, pp. 63-74.
- 269 8. Koukias, A.; Kiritsis, D.; Rule-based mechanism to optimize asset management using a technical  
270 documentation ontology. In *15th IFAC Symposium on Information Control Problems in Manufacturing 2015*,  
271 IFAC-PapersOnLine; Volume 48, Issue 3, pp. 1001 – 1006.

- 272 9. Zagorulko, U.; Ahmadeeva, I.; Mouromtsev, D.; Ontology-Based Information Extraction for Populating the  
273 Intelligent Scientific Internet Resources, In *International Conference on Knowledge Engineering and the Semantic*  
274 *Web*, Springer:Cham, Prague, Czech Republic, 21-23 September 2016; Volume 649, pp. 119-128.
- 275 10. Namestnikov, A.; Guskov, G.; Ontological Mapping for Conceptual Models of Software System, In  
276 Proceedings of conference Open Semantic Technologies for Intelligent Systems, Minsk, Republic of Belarus,  
277 16-18 February 2017, pp. 111-116.
- 278 11. Yarushkina, N.; Filippov, A.; Moshkin, V.; Development of the Unified Technological Platform for  
279 Constructing the Domain Knowledge Base Through the Context Analysis. In *Proceedings of the Conference*  
280 *on Creativity in Intelligent Technologies and Data Science (CIT&DS 2017)*, Springer:Cham, Volgograd, Russian  
281 Federation, 12-14 September 2017; Volume 754, pp 62-72.
- 282 12. Hossein, S.; Sartipi, K.; Dynamic analysis of software systems using execution pattern mining. In *Proceedings*  
283 *14th IEEE International Conference on Program Comprehension, ICPC*, IEEE Computer Society, Athens, Greece,  
284 14-16 June 2006; pp. 84-88.
- 285 13. Ma, Z.; Zhang, F.; Yan, L.; Cheng, J.; Representing and reasoning on fuzzy UML models: A description logic  
286 approach. *Expert Systems with Applications* **2011**, Volume 38, 2536-2549.
- 287 14. Zedlitz, J.; Jorke, J.; Luttenberger, N.; From UML to OWL 2. In *Proceedings of Third Knowledge Technology Week*  
288 *(KTW 2011)*, Springer:Berlin/Heidelberg, Kajang, Malaysia 18-22 July 2011; Volume 295, pp 154-163.
- 289 15. Bobillo, F.; Straccia, U.; Representing Fuzzy Ontologies in OWL 2. In *Proceedings International Conference on*  
290 *Fuzzy Systems*, Barcelona, Spain, 18-23 July 2010; pp. 2695-2700.
- 291 16. Wongthongtham, P.; Pakdeetrakulwong U.; Marzooq, S.; Ontology annotation for software engineering  
292 project management in multisite distributed software development environments. In *Software Project*  
293 *Management for Distributed Computing*. Zaigham Mahmood; Springer:Cham, 2017; pp. 315-343,  
294 978-3-319-54325-3.
- 295 17. Emdad, A.; Use of ontologies in software engineering. In *Proceedings of 17th International Conference on*  
296 *Software Engineering and Data Engineering*, Los Angeles, California, USA, 30 June - 2 July 2008; pp. 145-150.
- 297 18. Dillon, T.; Chang, E.; Wongthongtham, P.; Ontology-based software engineering-software engineering 2.0.  
298 In *Proceedings of Australian Software Engineering Conference*, IEEE Computer Society, 26-28 March 2008; pp.  
299 13-23.
- 300 19. Dudarin, P.; Yarushkina, N.; Features construction from hierarchical classifier for short text fragments  
301 clustering. *Fuzzy systems and soft computing* **2017**, *12*, 87-96.
- 302 20. Dudarin, P.; Yarushkina, N.; An Approach to Fuzzy Hierarchical Clustering of Short Text Fragments Based  
303 on Fuzzy Graph Clustering. In Proceedings of the Second International Scientific Conference "Intelligent  
304 Information Technologies for Industry" (IITI 2017), Springer:Cham, Varna, Bulgaria, 14-16 September 2017;  
305 Volume 679, pp 295-304.
- 306 21. McInnes L., Healy J., Astels S., hdbscan: Hierarchical density based clustering In: *Journal of Open Source*  
307 *Software*, The Open Journal, 2017, Volume 2, number 11.
- 308 22. Grand, M.; *Java Enterprise Design Patterns: Patterns in Java*. Wiley 2001