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

Nadezhda Yarushkina 1,†, Gleb Guskov 1,†,∗ and Pavel Dudarin 1,†

1 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].
Nowadays there are many available pre-trained word2vec models [4,5], so there only thing to do is additionally tune word2vec model for specific task.

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

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

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

2. Formulation of the problem

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

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

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

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

The search for project structural similarity is part of the project comparison method. Another part of this method is projects filtering base on subject areas. As long as information about software project collected from open sources and there is no common tag system or any commonly used classifier,
the only way to group project is to perform clustering procedure. This part is performed by using a combination of approaches suggested in papers [19,20] for short text clustering based on semantic similarity measure obtained from word2vec pre-trained model.

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

3. Software projects preliminary filtering

3.1. Fuzzy hierarchical classifier constructing

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

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

Figure 1. Extract from hierarchical classifier of software project terms
Table 1. A set of sentences for software projects preliminary filtering

<table>
<thead>
<tr>
<th>Project name</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>VKCOM/vk-java-sdk</td>
<td>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.</td>
</tr>
<tr>
<td>dewarder/HoldingButton</td>
<td>Button which is visible while user holds it. Main use case is controlling audio recording state (like in Telegram, Viber, VK).</td>
</tr>
<tr>
<td>korobitsyn/VKOpenRobot</td>
<td>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</td>
</tr>
<tr>
<td>gleb-kosteiko/vkb</td>
<td>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.</td>
</tr>
<tr>
<td>PhoenixDevTeam/Phoenix-for-VK</td>
<td>First open-sourced VK client for Android inspired by Material Design.</td>
</tr>
<tr>
<td>petersamokhin/vk-bot-java-sdk</td>
<td>Comfortable and simple library for creating bots for VK</td>
</tr>
<tr>
<td>strelnikovkirill/VKPhotoApp</td>
<td>Android OS + VK Api. VK application for a surfing in user news feed, but this news feed build only on posted photos.</td>
</tr>
<tr>
<td>vladgolubev/nowplayingVk</td>
<td>This app broadcasts currently playing song from last.fm account to vk.com status</td>
</tr>
<tr>
<td>shavkunov/vk-analyzer</td>
<td>Application used to analyze wall of VK user or community and save results to internal database.</td>
</tr>
<tr>
<td>asaskevich/VK-Small-API</td>
<td>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</td>
</tr>
<tr>
<td>akveo/cordova-vk</td>
<td>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.</td>
</tr>
<tr>
<td>MLSDev/DroidFM</td>
<td>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.</td>
</tr>
<tr>
<td>Try4W/VKontakteAPI</td>
<td>Simple, light weighted binding VK API for Java based on official Android SDK</td>
</tr>
</tbody>
</table>
3.2. Feature construction

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

\[ F : SL \rightarrow \mathbb{R}^n, \forall s \in SL(s) = (s_1, s_2, ..., s_n), \quad s_i = R(s, v_i) \ast \frac{1}{\omega(v_i) \ast \sigma \ast \sqrt{2 \ast \pi}} \ast e^{-\frac{(\text{ln} \omega(v_i) - \mu)^2}{2\sigma^2}}, \]

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

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

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

3.3. Projects clustering

To perform clustering procedure an HDBScan [21] algorithm has been chosen. This clustering algorithm combined with features constructed from hierarchical classifier hierarchical classifier has been chosen for its ability to return quite accurate and pure clusters, mainly at the expense of precision.

Metrics for precision, accuracy and purity are defined for each class as follows:

\[ \text{Precision}_i = \frac{\max_j CM_{i,j}}{\sum_j CM_{i,j}}, \quad \text{Accuracy}_i = \frac{\max_j CM_{i,j}}{\sum_k \sum_j CM_{k,j}^\text{max}}, \quad \text{Purity}_i = \frac{\max_j CM_{j,i}}{\sum_j CM_{j,i}}, \]

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

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

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

During the clustering process 15 clusters we determined, main clusters with general description are presented on the Figure 2. HDBScan algorithm was performed with parameter \( \text{min}_{\text{cluster size}} = 5 \). Algorithm HDBScan as a true clustering algorithm always forms ‘noise cluster’ with number -1, where all the samples that could not be grouped are moved to.

![Figure 2. Clustering results](https://github.com/PavelDudarin/sentence-clustering)
In our example the ‘noise cluster’ is quite big, the main reason for this are too poor projects’ description and high dimensional clustering features. More detailed discussion for this could be found in [19]. Future studies will be dedicated to solving this problem.

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

Table 2. Sample of clustering results

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

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

4. Software design ontology

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

4.1. UML meta-model based ontology

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

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

Ontology contains concepts that describe the most basic elements of the class diagram, but it could be expanded if necessary. During translation of the UML meta-scheme the following notations were applied.
Formally, the ontology of project diagrams is represented as a set:

\[ O_{prj} = (C_{prj}, R_{prj}, F_{prj}) \]

where:

- \( C_{prj} = \{c_{prj}^1, ..., c_{prj}^i\} \) – is a set of concepts that define main UML diagram elements such as: "Class", "Object", "Interface", "Relationship" and others;
- \( R_{prj} \) – the set of connections between ontology concepts. These relationships allow to describe correctly rules of UML notation.
- \( F_{prj} \) – is the set of interpretation functions defined on the relationships \( R_{prj} \).

4.2. Design patterns as structural parts of software projects

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

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

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

- **SimpleClass**: Builder_Client, Builder_Director, Builder_ConcreteBuilder, Builder_Product.
- **AbstractClass**: Builder_AbstractBuilder.
- **Association**: Builder_Client_AbstractBuilder, Builder_Client_Director, Builder_Client_IProduct, Builder_ConcreteBuilder_Product.
- **Generalization**: Builder_ConcreteBuilder_AbstractBuilder.
- **Realization**: Builder_Product_IProduct.

Ontological representation of the design pattern:

\[ O_{prj}^{tmp} = \{\text{inst}(C_{prj}^1), ..., \text{inst}(c_{prj}^i), ..., r_{\text{sameAs}}\} \]

In fact, the ontological representation of a single design pattern is a set of individuals of concepts 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:

\[ \mu_{dc_{\gamma}, dc_{\delta}} = \bigvee_{\text{tmp} \in (dc_{\gamma} \cap dc_{\delta})} \mu_{dc_{\gamma} \cap dc_{\delta} (\text{tmp})} \]

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

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

\[ \mu_{dc_{\gamma}, dc_{\delta}} = \left( \sum_{\text{tmp} \in (dc_{\gamma} \cap dc_{\delta}) \geq 0.3} \mu_{dc_{\gamma} \cap dc_{\delta}} (\text{tmp}) \right) / N, \]

where \( N \) is the total number of design patterns in both classes diagrams.
Figure 3. “Builder” design pattern ontology presentation in Protege editor.

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

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

$$
\mu_{dc_1,dc_2} = \left( \sum_{tmp \in (dc_1 \cap dc_2) \geq 0.3} \mu_{dc_1 \cap dc_2} \right) / N,
$$

(5)

where $\mu_{dc_1 \cap dc_2}$ - weighted measure of expression.

5. The results of searching structurally similar software projects

5.1. Searching design patterns in projects

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

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

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

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

<table>
<thead>
<tr>
<th>Project name / Design pattern name</th>
<th>Delegator (3)</th>
<th>Adapter (8)</th>
<th>Builder (12)</th>
<th>Abstract superclass (3)</th>
<th>Interface (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-MVP</td>
<td>1.0</td>
<td>0.875</td>
<td>0.83</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>cordova-social-vk</td>
<td>1.0</td>
<td>0.875</td>
<td>0.83</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>cvk</td>
<td>1.0</td>
<td>0.875</td>
<td>0.83</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>DroidFM</td>
<td>1.0</td>
<td>0.875</td>
<td>0.92</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>VK-Small-API</td>
<td>1.0</td>
<td>0.875</td>
<td>0.83</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>VKontakteAPI</td>
<td></td>
<td>0.75</td>
<td>0.58</td>
<td>0.66</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 4. Similarity between projects by second metric

<table>
<thead>
<tr>
<th>Project 1 / Project 2</th>
<th>android-mvp</th>
<th>cordova-vk</th>
<th>cvk</th>
<th>droidfm</th>
<th>vk-small-api</th>
<th>vkontakte-api</th>
<th>vk_test</th>
</tr>
</thead>
<tbody>
<tr>
<td>android-mvp</td>
<td>–</td>
<td>0.96</td>
<td>0.96</td>
<td>0.98</td>
<td>0.78</td>
<td>0.96</td>
<td>0.78</td>
</tr>
<tr>
<td>cordova-vk</td>
<td>0.96</td>
<td>–</td>
<td>0.96</td>
<td>0.94</td>
<td>0.85</td>
<td>1</td>
<td>0.83</td>
</tr>
<tr>
<td>cvk</td>
<td>0.96</td>
<td>1</td>
<td>–</td>
<td>0.94</td>
<td>0.85</td>
<td>1</td>
<td>0.83</td>
</tr>
<tr>
<td>droidfm</td>
<td>0.98</td>
<td>0.94</td>
<td>0.94</td>
<td>–</td>
<td>0.78</td>
<td>0.94</td>
<td>0.78</td>
</tr>
<tr>
<td>vk-small-api</td>
<td>0.78</td>
<td>0.85</td>
<td>0.85</td>
<td>0.78</td>
<td>–</td>
<td>0.85</td>
<td>0.96</td>
</tr>
<tr>
<td>vkontakte-api</td>
<td>0.96</td>
<td>1</td>
<td>1</td>
<td>0.94</td>
<td>0.85</td>
<td>–</td>
<td>0.83</td>
</tr>
<tr>
<td>vk_test</td>
<td>0.79</td>
<td>0.83</td>
<td>0.83</td>
<td>0.78</td>
<td>0.95</td>
<td>0.83</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 5. Similarity between projects by third metric

<table>
<thead>
<tr>
<th>Project 1 / Project 2</th>
<th>android-mvp</th>
<th>cordova-vk</th>
<th>cvk</th>
<th>droidfm</th>
<th>vk-small-api</th>
<th>vkontakte-api</th>
<th>vk_test</th>
</tr>
</thead>
<tbody>
<tr>
<td>android-mvp</td>
<td>–</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.64</td>
<td>0.96</td>
<td>0.77</td>
</tr>
<tr>
<td>cordova-vk</td>
<td>0.96</td>
<td>–</td>
<td>0.99</td>
<td>0.93</td>
<td>0.67</td>
<td>0.99</td>
<td>0.80</td>
</tr>
<tr>
<td>cvk</td>
<td>0.97</td>
<td>0.99</td>
<td>–</td>
<td>0.93</td>
<td>0.67</td>
<td>0.99</td>
<td>0.80</td>
</tr>
<tr>
<td>droidfm</td>
<td>0.97</td>
<td>0.93</td>
<td>0.93</td>
<td>–</td>
<td>0.61</td>
<td>0.93</td>
<td>0.74</td>
</tr>
<tr>
<td>vk-small-api</td>
<td>0.64</td>
<td>0.67</td>
<td>0.68</td>
<td>0.61</td>
<td>–</td>
<td>0.67</td>
<td>0.87</td>
</tr>
<tr>
<td>vkontakte-api</td>
<td>0.97</td>
<td>0.99</td>
<td>0.99</td>
<td>0.93</td>
<td>0.67</td>
<td>–</td>
<td>0.80</td>
</tr>
<tr>
<td>vk_test</td>
<td>0.77</td>
<td>0.80</td>
<td>0.80</td>
<td>0.74</td>
<td>0.87</td>
<td>0.80</td>
<td>–</td>
</tr>
</tbody>
</table>

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

The results for the second and third metrics are also quite high. Design patterns with an expression degree less than 0.3 were excluded from consideration. All projects that participated in the comparison are downloaded from the open repository Github and realize interaction with the public API of the Russian well-known social network Vkontakte or with its music or mobile API.

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

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

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

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

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

Abbreviations

The following abbreviations are used in this manuscript:

- OWL: Ontology web language
- UML: Unified modeling language
- NLP: Natural Language Processing

References


