Approach to determining the structural similarity of software projects

Gleb Guskov
Information system department
Ulyanovsk State Technical University
Ulyanovsk, Russia
guskovgleb@gmail.com

Alexey Namestnikov
Information system department
Ulyanovsk State Technical University
Ulyanovsk, Russia
am.namestnikov@gmail.com

***Abstract*—The paper proposes a method for comparing software projects on the basis of their structural similarity. An approach is proposed for obtaining an ontological representation of the project structure based on the source code. The paper considers several metrics for calculating the structural similarities applicable for different types of projects.**

***Keywords*—ontology, conceptual model, engineering design, hierahical clusstering**

I. Introduction

Human resources in modern software development are basic. Often the same tasks are solved several times, this leads to an ineffective waste of time. There are approaches for reusing source code at various stages of development. Basically, these approaches allow you to reuse only certain functions and classes. But such approaches do not allow finding the similarity of projects on the basis of their subject area.

Knowledge gained from already implemented projects in one subject area will allow borrowing and reusing much larger parts of projects and avoiding conceptually incorrect solutions. Quite often when updating the composition of developers, the implemented software solutions are forgotten and not used.

A tool capable of determining the similarities between projects can be very useful in software development. This tool can not be replaced by a version control system, because it only provides storage of all versions of the project with comments. Version control system provides comparison beetween file versions. Such an instrument will be able to work not only with projects from single organization, but also with projects from open repositories.

Search on open repositories is carried out on the basis of keywords. The results of such a search can be several thousand projects, which can not be handled by hand. The choice of projects based on their architectural solutions is a promising approach. To implement filtering on projects based on their architecture, you need to be able to analyze it. The architecture of the software project is built at the stage of its design, prior to the development. Elements of the UML language were developed to describe the architecture of the project with the required level of detail. Basing on the results of our previous research [1], we can conclude that the developers have different types of structural elements. 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 [2]–[7], maintenance and testing of the source code [8], the UML diagrams [9]–[13].

The minimal structural elements of UML, such as classes, interfaces, objects themselves, weakly convey the semantics and architectural solutions of the project. The combination of such elements is much better describes the architecture of the project. Stable combination of structural elements, known as design patterns. The design patterns appeared relatively long time in information technology and are still relevant. Design patterns are actively used by the developer community, thus representing a reliable benchmark in the analysis of the project. In addition, it makes sense to create local design patterns that solve this or that task in a given subject area. A template based on a specific subject area loses its main property - universality, but its semantic weight is a more important metric for solving the problem of constructing a tool for finding similarities between projects. There are many works devoted to the integration of software development with ontologies. There is a whole approach to development, based on a domain known as development based on the subject area [14]–[16].

II. Formulation of the problem

The results presented in this paper are a continuation of the work described in research[1]. The system described in 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 manufacturing practices of IT companies show that conceptual models are created at best at each stage of the project, and in the worst once at the beginning of the project.

The state of the project is best described by the source code of the project. Developers try to maintain the source code in good condition, create documentation, provide comments and opportunely refactor. Another advantage of the source code is the widespread introduction of version control systems. Tracking all versions of software products allows effectively

The authors acknowledge that this paper was supported by the project no. 16-47-730742 and 16-47-732033 of the Russian Foundation for Basic Research.

manage the development of software and generates a large amount of information available for processing.

Information for comparison 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 will allow to determine the structural similarity of projects.

To get the projects structural similarity, it is necessary to translate information about projects from different sources to a single format. It is most convenient to present the extracted information in the form of ontology in the language of OWL. OWL ontology will allow preserving the semantics of complex architectural solutions, allowing to make changes to already existing data and to perform logical operations on statements.

The search for structural similarity of projects is part of the project comparison method. In addition to comparing the structure of projects, it is planned to compare still subject areas of projects. If a comparison is made between projects of the same enterprise in one subject area, then the comparison should be performed at the level of the processes and the components of the subject area. If the comparison of the project is carried out among the projects placed on the open repository, the structural similarity of the projects is more important metric then general subject area.

III. UML META-MODEL BASED ONTOLOGY

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

To solve the problem of intellectual analysis of project diagrams, included in the project documentation, it is necessary to have knowledge in the area of construction of formalized diagrams.

Ontology contains concepts that describe the most basic elements of the class diagram, but it can be expanded if necessary. When translating the meta-scheme of UML, the following notations were applied.

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

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

where : $C^{prj} = \{c_1^{prj}, ...c_i^{prj}\}$ – 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 us to correctly describe the rules of UML notation.
F^{prj} – is the set of interpretation functions defined on the relationships R^{prj}

IV. DESIGN PATTERNS AS STRUCTURAL PARTS OF SOFTWARE PROJECTS

Design patterns are insert into ontology as a set of individuals based only on the ontology classes described above. Semantic constraints and properties of design patterns are specified with 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 enter rules of naming for their elements to avoid duplicate names. The name of the design pattern element begins with the design pattern name, and then if the element is the class, its name is written. If the element is a relationship, then the names of the elements that it connects are written through the underscore. One of the most commonly used design patterns is the Builder [17].

Builder is a creational pattern. The Builder pattern separates the algorithm for the step-by-step construction of a complex object from its external representation so that it is possible to obtain different representations of this object using the same algorithm.

In order to preserve this design pattern in the developed ontology, the following individuals were 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:

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

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

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

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

where dc_γ and dc_δ is project class diagrams presented as UML metamodel ontology Abox expressions,
$\mu_{dc_\gamma, dc_\delta(tmp)}$ - measure of similarity design pattern in project diagram.
The second metric considers the coincidence of all design patterns in equal proportions and does not considers design patterns with a measure of expression less than 0.3:

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

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

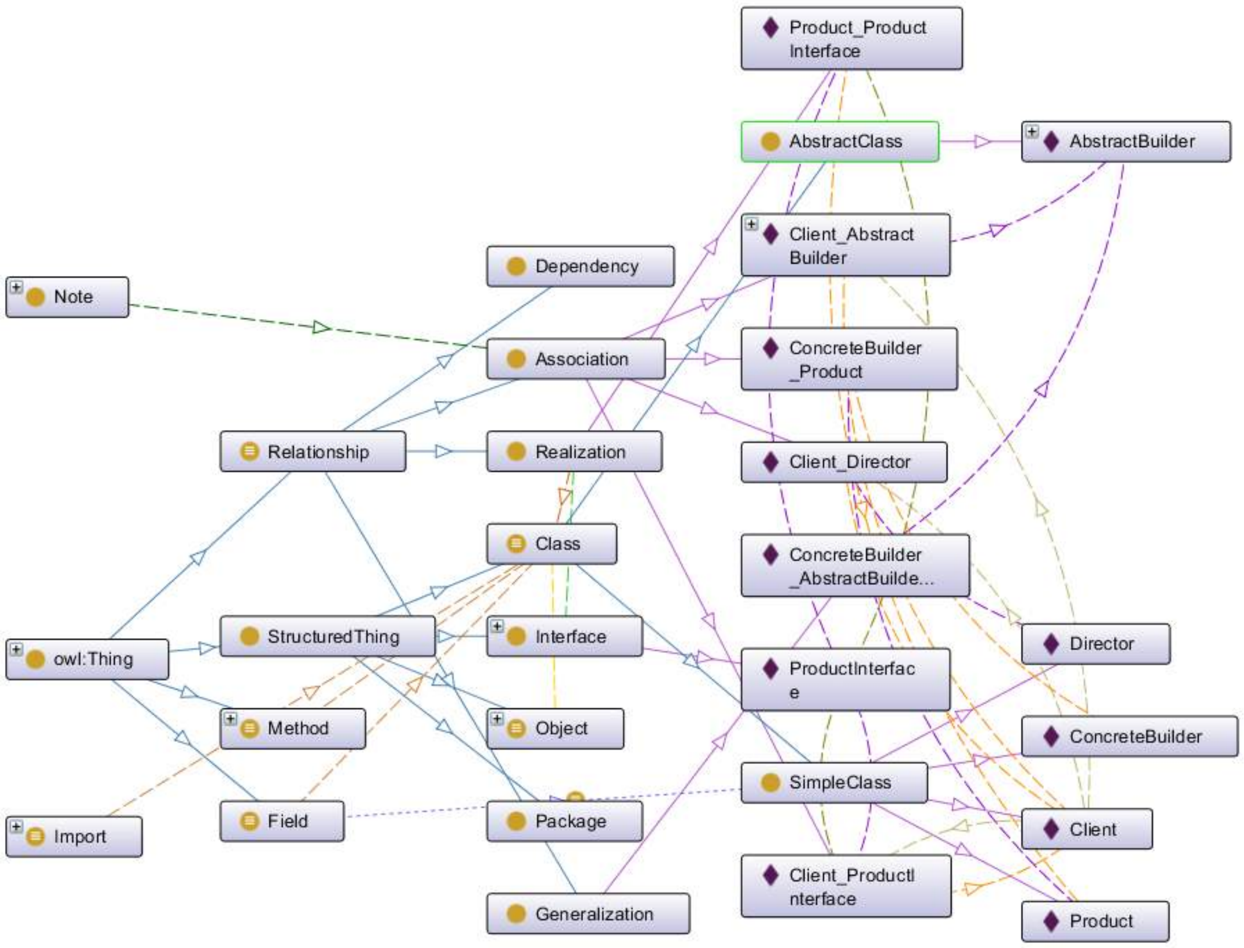

Figure 1. "Builder" design pattern ontology presentation in Protege editor.

The third metric works the same 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):

$$\mu_{dc_\gamma,dc_\delta} = \Big(\sum_{tmp \in (dc_\gamma \cap dc_\delta) \geq 0.3} \widetilde{\mu}_{dc_\gamma \cap dc_\delta} \Big) / N, \qquad (5)$$

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

V. The results of searching structurally similar software projects

A. *Searching design patterns in projects*

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

B. *Results of searching structurally similar software projects by different metrics*

The results of calculating the similarity between projects by three metrics are presented in the Table II.

Estimates based on the results of comparison projects are quite high. The estimates are normalized from 0 to 1. The estimates for the first metric are always equal to 1. This is easy to explain, because it chooses the most expressed design pattern in both projects. Among the design patterns participating in the test there are templates with a small number of elements, for example: Absatrast superclass, interface and delegator. The results for the second and third metrics are also quite high. Design patterns with a degree of expression less than 0.3 are excluded from consideration. All projects that participated in the comparison are downloaded from the open repository Github and somehow interact with the public API of the well-known social network vkontate.

Table I
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 II
SIMILARITY BETWEEN PROJECTS

First project / Second project	Android-MVP	cordova-social-vk	cvk	DroidFM	VK-Small-API	VKontakteAPI	VK_TEST
Android-MVP	–	1 | 0.96 | 0.96	1 | 0.96 | 0.96	1 | 0.98 | 0.96	1 | 0.78 | 0.64	1 | 0.96 | 0.96	1 | 0.78 | 0.77
cordova-social-vk	1 | 0.96 | 0.96	–	1 | 1 | 0.99	1 | 0.94 | 0.93	1 | 0.85 | 0.67	1 | 1 | 0.99	1 | 0.83 | 0.80
cvk	1 | 0.96 | 0.97	1 | 1 | 0.99	–	1 | 0.94 | 0.93	1 | 0.85 | 0.67	1 | 1 | 0.99	1 | 0.83 | 0.80
DroidFM	1 | 0.98 | 0.97	1 | 0.94 | 0.93	1 | 0.94 | 0.93	–	1 | 0.78 | 0.61	1 | 0.94 | 0.93	1 | 0.78 | 0.74
VK-Small-API	1 | 0.78 | 0.64	1 | 0.85 | 0.67	1 | 0.85 | 0.68	1 | 0.78 | 0.61	–	1 | 0.85 | 0.67	1 | 0.96 | 0.87
VKontakteAPI	1 | 0.96 | 0.97	1 | 1 | 0.99	1 | 1 | 0.99	1 | 0.94 | 0.93	1 | 0.85 | 0.67	–	1 | 0.83 | 0.80
VK_TEST	1 | 0.79 | 0.77	1 | 0.83 | 0.80	1 | 0.83 | 0.80	1 | 0.78 | 0.74	1 | 0.95 | 0.87	1 | 0.83 | 0.80	–

CONCLUSION

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. Expanding the system by using ontologies of subject areas can significantly increase the relevance of the similar projects selection.

REFERENCES

[1] G.Guskov A. Namestnikov, *Ontological mapping for conceptual models of software system*, OSTIS-2017 proceedings, 2017.

[2] A.Namestnikov , A. Filippov, V. Avvakumova, *An ontology based model of technical documentation fuzzy structuring.*, CEUR Workshop Proceedings. SCAKD 2016. Moscow. Russian Federation. Volume 1687, pp. 63-74 (2016).

[3] A.Goy, D. Magro, *Towards an ontology-based software documentation management - a case study.*, KMIS (Kecheng Liu and Joaquim Filipe, eds.), SciTePress, pp. 125–131 (2012).

[4] A. Koukias, D.Kiritsis, *Rule-based mechanism to optimize asset management using a technical documentation ontology*, IFAC-PapersOnLine 48, no. 3, 1001 – 1006 (2015).

[5] U. Zagorulko, I. Ahmadeeva, A. Serii, V. Shestakov, *Postroenie tematicheskih intellektualnih nauchnih internet-resursov sedstvami semantic web*, Trudi 15 nacionalnoi konferencii po iscusstvennomu intelektu KII-2016, Smolensk, vol 2, 2016, pp. 47-55.

[6] A. Namestnikov, G. Guskov, *Programnaya sistema preobrazovaniya UML-diagram v ontologii na yazike OWL* , Trudi 15 nacionalnoi konferencii po iscusstvennomu intelektu KII-2016, Smolensk, vol 3, 2016, pp. 270-278.

[7] Filippov A.A., Moshkin V.S., Shalaev D. O., Yarushkina N.G. *Edinaya ontologicheskaya platforma intellectualnogo analiza dannih//* Materiali VI mejgunarodnoi nauchno-tehnicheskoikonferencii OSTIS-2016, Minsk, Respublica Belarus, 2016.

[8] Hossein S., Sartipi K., *Dynamic analysis of software systems using execution pattern mining.*, ICPC, IEEE Computer Society, pp. 84–88 (2006).

[9] Ma, Z., Zhang, F., Yan, L., Cheng, J. *Representing and reasoning on fuzzy UML models: A description logic approach.* Expert Systems with Applications,38(3), 2536–2549 (2011).

[10] J. Zedlitz, J. Jorke,N. Luttenberger, *From UML to OWL 2*, In: Proceedings of Knowledge Technology Week 2011. Springer (2012).

[11] Almeida Ferreira D., Silva A., *UML to OWL Mapping Overview An analysis of the translation process and supporting tools.* Conference: 7th Conference of Portuguese Association of Information Systems(2013).

[12] Bobillo F., Straccia U., *Representing Fuzzy Ontologies in OWL 2*, WCCI IEEE World Congress on Computational Intellegence, Barcelona, Spain, July 18-23, pp.2696- 2700. (2009)

[13] N.Dorodnich i A. Urin, *Ispolzovanie diagram klassov UML dlya formirovaniya produccionnih baz znanii*, Programnaya ingeneriya vol. 4, 2015.

[14] Wongthongtham P., Pakdeetrakulwong U.,Marzooq S., *Ontology annotation for software engineering project management in multisite distributed software development environments*, pp. 315–343, Springer International Publishing, Cham, 2017.

[15] Emdad A., *Use of ontologies in software engineering.*, SEDE (Hisham Al-Mubaid and Rym Zalila-Wenkstern, eds.), ISCA, pp. 145–150 (2008).

[16] Dillon T., Chang E., Wongthongtham P., *Ontology-based software engineering- software engineering 2.0.*, Australian Software Engineering Conference, IEEE Computer Society, pp. 13–23 (2008).

[17] Mark Grand, *Java enterprise design patterns: Patterns in java (patterns in java)*, John Wiley and Sons, 2002

ПОДХОД К ПОИСКУ ПРОГРАММНЫХ ПРОДУКТОВ СО СХОЖЕЙ СТРУКТУРОЙ

Глеб Гуськов, Алексей Наместников
Кафедра «Информационные системы»
Ульяновский государственный технический университет, Россия, Ульяновск

В статье описан метод сравнения программных продуктов на основе их структурного сходства. Подход основан на использовании онтологических представлений структуры программных продуктов. Структура программных продуктов извлекается из исходного кода или концептуальных моделей. В статье рассматривается несколько различных метрик для расчёта структурного сходства, которые могут быть использованы для проектов различных типов.