An Approach to Software Assets Reusing

Olena Chebanyuk[1,2(✉)]

[1] Department of Informatics, New Bulgarian University, Sofia, Bulgaria
Chebanyuk.olena@gmail.com
[2] Software Engineering Department, National Aviation University, Kyiv, Ukraine

Abstract. The modern software development methodologies require systematic reuse of software assets. It is expected that reuse becomes a cause of reducing efforts. From the other hand, nowadays reuse procedure is connected with plenty of problems and risks, for example, how to choose the best software asset from the set of available for reuse? What criterion should be considered to estimate internal structure of software asset? How to perform such an operation quickly and effectively? Paper proposes the approach allowing predicting estimation of effectiveness for further reuse of software asset. Approach is based on matching software assets to requirement specification considering their semantic attributes, namely OCL expressions. Software asset is associated with some problem domain process through keywords. Points of requirement specification are associated with the same keywords and completed by OCL expressions. Similarity of requirement specification and software assets from repository is defined by means of comparing corresponding keywords and OCL expressions. Model for approximate comparison of OCL expressions and its estimation are proposed. Evaluation of the proposed approach according to IBM reuse maturity model is represented.

Keywords: Software product lines · Constraint modeling and languages · AGILE · Model-driven development · OCL · Domain engineering

1 Introduction

Successful software development process today needs following to many business requirements, for example low code, speed development, reducing of development costs, high quality of code and others. Answer to it – organization of development process involving techniques that satisfy to the next requirements: (i) software assets reuse; (ii) high levels of software development processes maturity; (iii) automatization of routine activities in software development processes.

One of the ways to answer to these business requirements is involving activities of asset-based development into modern software development methodologies, such as Test-Driven Development, Behavioral-Driven Development, and Software Product Line approaches [4].

Approaches and tools for static code analysis and generation of static UML diagrams from source code aimed to discover only structure of the source code.

T. Zlateva and R. Goleva (Eds.): CSECS 2022, LNICST 450, pp. 73–83, 2022.
https://doi.org/10.1007/978-3-031-17292-2_6

Questions about semantic are solved mentally and require much time in case of big amount of software assets under search area. Procedure of searching software assets may take much time and, then to become unsuccessful. In addition, absents of strict criteria how to estimate results of searching may be cause for different developers to give different answers to question: “Is found software asset suitable for reuse?”.

Further results of reuse search are also may be unpredictable. Similar situations are related to reuse problems for other types of assets, such as test suites test cases, interface prototypes, database models, UML (or BPMN) diagrams [15].

2 Review of Papers

Strategy of related papers review is based on matching papers to Reuse Maturity Model, proposed by IBM [4]. Graphical representation of Reuse Maturity Model looks like a matrix. Each row corresponds to maturity level of CMMI model [6], and its cells explain activities of stakeholders [4].

Researches aimed to perform software assets analysis according to different levels of CMMI, are concentrated on several areas:

– analysis of requirement specification by means of natural languages analysis techniques [9, 15];

There are reuse approaches containing:

1. Ontology-based approach to search similar requirements analyzing texts of requirement specification [16];
2. Investigating of human factor influences in considering requirement specifications for specific areas [12].

– analysis of the software assets’ structure with the aim to discover their functionality [14];
– techniques and approaches to describe software asset semantic using analytical apparatus [1].

There are reuse approaches containing:

1. System of elements to represent assets’ features (their structure and semantic);
2. A set of requests to recognize assets’ features [7].
3. Complex rules, expressions, and patterns for representation of reuse asset structure [8, 11].
4. Approaches related to using ontologies

– flexible approaches that are based on using intermedia analytical languages to explain semantic of software asset [2].

There are reuse approaches containing:

1. Designing of new reuse approaches that are based on new transformation rules containing specific facts about concrete domains [3];

2. Modification of software designing technologies

Literature review gives the ground to formulate research task – to propose an approach for software assets reuse based on analysis both semantic attributes of requirement specification and software assets.

3 Task and Research Questions

Task: to propose the approach for estimating effectiveness of software assets reuse based on comparison of semantic attributes of software assets and requirement specification. Keywords in natural language and OCL expressions [13] are chosen as semantic attributes. In forward engineering activities, semantic attributes are prepared in requirement analysis. In reverse engineering activities, semantic attributes are prepared when software asset repository is composed.

Research Questions (RQs)

RQ1: Propose a structure for software assets repository. Aim of repository is to systemize information about assets for further reuse. Assets that are gathered in repository related to concrete problem domain.
RQ2: Propose a structure of requirement specification, which summarizes requirements description and semantic attributes of requirements.
RQ3: Develop a comparison algorithm for matching semantic attributes of repository assets and software requirement specification.
RQ4: Propose a model for approximate comparison of semantic attributes allowing comparing semantic of requirement specification and assets from repository (define criterion for full and approximate matching of OCL expressions).
RQ5: Perform an experiment, verifying proposed approach and comparison model.

4 Proposed Approach

Proposed approach is divided into three stages:

– domain analysis - when repository of software assets is designed;
– requirement analysis - when semantic attributes of requirement specification are composed;
– comparison of semantic attributes - when decision about reuse of software module is performed.

1. Domain analysis.

    1.1. A problem domain tree, containing description of problem domain processes, sub-processes and keywords, is designed. See first two rows of the Table 1.

Table 1. Structure of repository tree for software assets

Problem domain processes and sub-processes	Keywords (kw)	Software asset name and links of software assets textual description	OCL expressions
Name of the process1	*kw1,* *kw2,* *kwn*		
Name of the sub-process1 (process1.1)	*kw1.1,* *,…,* *kw1.m*		

1.2. A repository tree of assets for considering problem domain is designed. Structure of repository tree is represented in the Table 1.

2. Requirement analysis.

2.1. A requirements specification with detailed description of the software project is designed.
2.2. Semantic attributes of requirement specification are prepared performing the next steps:

- expressions containing limitations in natural language are defined;
- OCL constraints for these limitations are designed.
- proper keywords from problem domain tree are selected;

Table 2 proposes the structure of requirement specification table with semantic attributes of requirements.

Table 2. Structure of requirement specification table with semantic attributes of requirements

Requirement specification (RQ) code	RQ description and limitations in natural language	Selected keywords from problem domain tree	OCL expressions
…	…	…	…

3. Comparison of semantic attributes from requirement specification and assets repository.

3.1. The common keywords for repository tree and requirement specification are defined. (Comparing proper rows from Table 1 and Table 2).

3.2. Corresponding OCL expressions are compared using a model for approximate comparison of OCL expression.
3.3. Comparison results are analyzed. Then decision about selection software assets from repository for the further reuse is made.

5 Model for Approximate Comparison of OCL Expressions

5.1 Graph Representation of OCL Expression

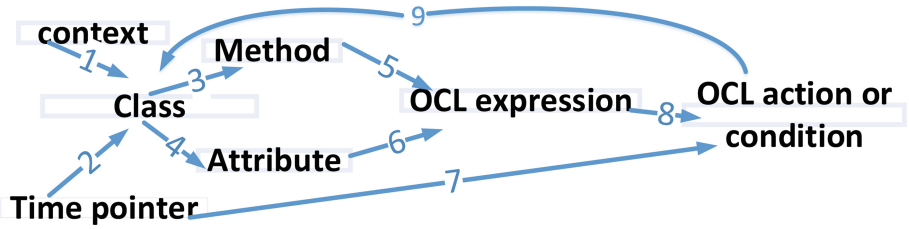

Fig. 1. Graph representation of OCL expressions

In order to describe rules for approximate comparison of OCL expressions, OCL expression is represented as a directed graph (Fig. 1). Developed scheme is based on graph representation of UML diagrams [3].

Possible variants of OCL graph vertexes meanings (Fig. 1) according to OCL standard [13].

Time pointer = *{pre, post, inv}* - keywords from [11].
Class = *{self* (keyword from [13], *class name}*.
OCL expressions = *{select, exists, forAll,* other keywords from [11]}.
OCL action or condition = *{or, and, true, false,* other keywords from [11]}.
Method – a signature of class method.
Attribute – a name and type of class attribute.

As a result of OCL expressions analysis (Fig. 1) several types of OCL were defined. Aim of the proposed classification is to define *key vertexes of OCL graph* that are important for approximate comparison. Due to limited size of paper, *only several types of OCL expressions* are considered.

OCL Expressions of the First Type

General graph representation of the OCL expression is defined by the following:

$$(context, 1, class), \ (class, 3, method) \tag{1}$$

Graph representation of the OCL expression, marked by green color, is defined by following (Fig. 1):

(context, 1, sensor), (sensor, 3, isActive(b:action):boolean)

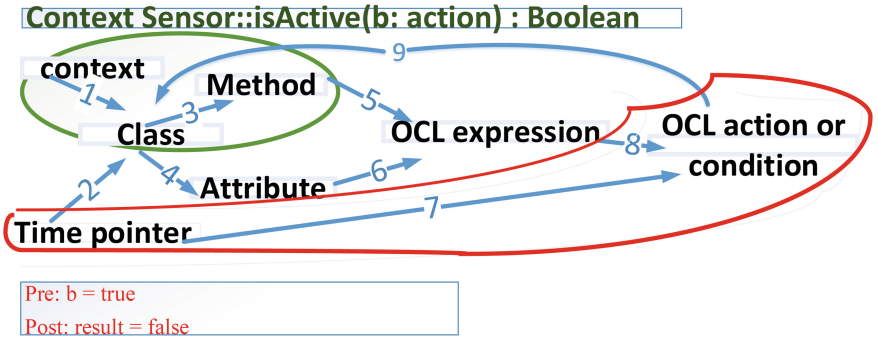

Fig. 2. Examples of OCL expressions representation

OCL Expressions of the Second Type

General graph representation of the OCL expressions, marked by red color, is defined by the following (Fig. 2):

$$(Time\ pointer, 7,\ OCL\ action\ or\ condition) \tag{2}$$

Graph representation of the OCL expression, marked by red color, is defined by following (Fig. 2):

(pre, 7, b = true),(post, 7, result = false)

Complex OCL Expressions

Lets' start from the example considering RQ description in natural language –

If sensor is active it is turned off.

Context Sensor::isActive(b: action): Boolean

pre: b = true.
post: result = false.

OCL expression is composed of OCL expressions of the first and second types (1) and (2).

If one OCL expression contains more than one string or represent different types of OCL expressions, it is a *complex OCL expression.*

OCL Expressions of the Third Type

Lets' start from the example considering RQ description in natural language –

If current user is registered, he obtains an access to database through network.

context Network

inv:self.dbAccess.users- > includes(self.currentUser)

General graph representation of these OCL expressions is defined by the following:

$$(Context, 1, class),\ (Class, 4, Attribute), (Attribute, 6, OCL\ expression),\\ (OCL\ expression,\ 8, OCL\ action) \tag{3}$$

OCL Expression of the Fourth Type

Lets' start from the example considering RQ description in natural language.

In order to obtain access to network resource user must get a proper permissions before.

context Network::authenticate (c:Credentials)

pre: self.dbAccess.users- > contains(u - > u.c = c);

It is complex OCL expression. The first sting is OCL expression of the first type (1). The second string represents OCL expression of the fourth type. General graph representation of the OCL expressions is defined by following:

$$(Time\ pointer, 2, Class), (Class, 4, Attribute), (Attribute, 6, OCL\ expression),\\ (OCL\ expression, 8, OCL\ condition) \tag{4}$$

Analysis of other types of OCL expressions is made by the same principle.

Proposed graph model (Fig. 1) is extensible. In order to consider different types of OCL expressions new vertexes and edges may be added.

In addition, graph representation idea may be used to describe other string-based query expressions (For example LINQ, SQL queries, predicate logic expressions or other types of query expressions).

5.2 Model for Approximate Comparison of OCL Expressions

1. Two OCL expressions are parsed to components in accordance with graph represent tation (1)–(4).
1. Types of these OCL expressions are defined. In order to perform this task graph representation of the considering OCL expression are compared with defined types of OCL expressions (1)–(4).
2. If two OCL are the same type, their graph representations are compared.
3. Comparison results are estimated for equality using proposed comparison model.

Comparison model is based on calculating values of weight coefficients that aimed to answer the question – What is the distance between two OCL expressions?

Value of weight coefficient shows "the importance" of some part of OCL expressions for approximate comparison. Those parts of OCL expressions that play more important role in comparison (from the authors' experience) are assigned to greater values of weight coefficients.

Proposed *principles of weight coefficients* assigning and criteria of approximately equality for OCL expressions are also *flexible*. Recommendations are grounded on empirical experience of authors. They were gathered while working in different projects performing semantic analysis of different assets. Area of authors' activities is monitoring of local networks security and game development area.

Estimation of the Proposed Model. Performing authors' projects, more than 50 OCL expressions were prepared manually to test proposed approach. In order to perform approximate comparison, activities similar to described in this chapter were done. For all OCL expressions analysis of human comparison and formal calculations are matched. It is the ground of idea that comparison model is valid and may be used for other OCL experiments.

6 Evaluation of the Proposed Approach

On order to estimate an effectiveness of the represented approach and measure maturity of reuse process it is proposed to use asset reuse strategy [3, 5]. Asset reuse strategy proposes to estimate effectiveness of reuse approaches calculating parameters important for return of investment (ROI) activities.

Estimation process is based on calculating time coefficients measuring "time ratio" between ROI operations of the represented approach and ad-hoc level processes (the first level of reuse maturity model).

In order to make evaluation process more accurate our team have chosen one more approach for comparison. It corresponds to the five maturity reuse level [7]. Part of our team tried to design domain models, represented as class diagrams, performing activities similar to [11]. It was an attempt to design new domain models reusing some entities of existing software assets (sending QVT-R requests to existing elements of class diagrams).

In order to explain the idea of evaluation process ROI parameter: "amount of time spent to learn information about asset" from asset reuse strategy is considered [5]. Table 3 represents time coefficients for different components of this parameter (The first row of the Table 3). Numbers in the title of the Table 3 (1, 4, and 5) correspond to maturity levels of reuse model.

Time coefficients (rows 4 and 5) show the percent of time economy for proposed approach (Sect. 3) and in comparison with the first maturity level.

Table 3. Time coefficients for estimation parameter "amount of time spent to learn information about asset"

Level of maturity	t(i)	1	4	5
Is asset about problem domain?	*t1*	1	-	0.1 0.2
What is the solution that is provided?	*t2*	1	0.1	0.07
What other assets can be used in conjunction with this asset?	*t3*	1	-	0.1 0.2
What constraints are placed on your project after you have used the asset?	*t4*	1	0.1	0.08

Cells, marked by green, correspond to maturity levels of time coefficients (t1–t4) for the proposed approach.

Cells, marked by yellow correspond to maturity levels of time coefficients (t1–t4) for the ad-hoc approach.

7 Conclusion

Paper proposes flexible approach for estimation of effectiveness for software assets reuse by means of comparison of semantic attributes for requirement specification and software assets. Keywords from natural language and OCL expressions play role of semantic attributes. Keywords serve as pointers allowing performing quick matching of problem domain processes and requirement specification. They help to reduce a number of OCL expressions to be compared (see case study, Table 6). OCL expressions play roles of «semantic hashes» representing characteristics of software requirements (or software assets from repository tree) and expressed by means of formal language.

Proposed approach extends traditional using of OCL expressions as part of meta-models for class diagrams analysis [5]. OCL is chosen for the proposed approach because it is widespread language of OMG standard and OCL plug-ins are integrated into many software tools used for software designing [8].

Consider other flexibilities of the proposed approach.

Area - *changing of semantic attributes.* The answer is that matching procedure may be changed in several directions:

– graph representation of OCL expression may be modified (Fig. 1);
– constituents of OCL expressions and weight coefficients' may be changed (Table 3) [13];
– calculation schemas for estimation for approximate equality may be modified;
– OCL may be changed to other graph-compatible notations for representation of semantic attributes (Table 1 and Table 2) (LINQ, SQL, predicate logic expressions) [10].
– instead of keywords in natural languages their hashes, or vector of synonyms may be used.

Main idea of represented approach (see Sect. 4) is to propose a flexible "way of thinking" in mentioned directions aimed to organize reuse of software assets considering their semantic attributes.

Proposed approach is related to Software Product Line techniques and requires extra efforts to perform domain analysis activities (Table 2). Analysis of procedures aimed to process software assets repositories shows that for large repositories search of information takes a long time and requires large amount of resources. Comparison of keywords allows saving search time for large repository. (For example, when repository contains more than 1000 software assets. Such a repository is used in authors' case study). Software assets are reused with optimized search and estimation procedures.

References

1. Abbas, M., Rioboo, R., Ben-Yelles, C.B., Snook, C.F.: Formal modeling and verification of UML Activity Diagrams (UAD) with FoCaLiZe. J. Syst. Archit. **1**(14), 101911 (2021)
2. Bjørner, D.: Domain engineering. In: Boca, P., Bowen, J., Siddiqi, J. (eds.) Formal Methods: State of the Art and New Directions, pp. 1–41. Springer, London (2010). https://doi.org/10.1007/978-1-84882-736-3_1
3. Chebanyuk, O., Palahin, O., Markov, K.: Domain engineering approach of software requirement analysis. Проблеми програмування **2–3**(2020), 154–172 (2020)
4. Gnatyuk, S., Kinzeryavyy, V., Stepanenko, I., Gorbatyuk, Y., Gizun, A., Kotelianets, V.: Code obfuscation technique for enhancing software protection against reverse engineering. In: Hu, Z., Petoukhov, S.V., He, M. (eds.) AIMEE2018 2018. AISC, vol. 902, pp. 571–580. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-12082-5_52
5. DeCarlo, J., et al.: Strategic Reuse with Asset-Based Development, vol. 15. IBM Corporation, Riverton (2008)
6. Validating and exploring characteristics of UML model elements (2015). https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=models-validating-uml-model-elements
7. Ferdinansyah, A., Purwandari, B.: Challenges in combining agile development and CMMI: a systematic literature review. In: 2021 10th International Conference on Software and Computer Applications, pp. 63–69 (2021)
8. Habeh, O., Thekrallah, F., Salloum, S.A., Shaalan, K.: Knowledge sharing challenges and solutions within software development team: a systematic review. In: Al-Emran, M., Shaalan, K., Hassanien, A.E. (eds.) Recent Advances in Intelligent Systems and Smart Applications. SSDC, vol. 295, pp. 121–141. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-47411-9_7
9. Kopetzki, D., Lybecait, M., Naujokat, S., Steffen, B.: Towards language-to-language transformation. Int. J. Softw. Tools Technol. Transf. **23**(5), 655–677 (2021). https://doi.org/10.1007/s10009-021-00630-2
10. Lami, G., Gnesi, S., Fabbrini, F., Fusani, M., Trentanni, G.: An automatic tool for the analysis of natural language requirements. Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre (2004). https://openportal.isti.cnr.it/data/2004/150653/2004_150653.pdf
11. OMG standard Object Constraint Language 2.3.1 Access mode (2011). https://www.omg.org/spec/OCL/2.3.1/About-OCL/
12. Nestererenko, К., Rahulin, S., Sharabaiko, A.: Human factor in the quality improvement system of aircraft maintenance. Системи управління, навігації та зв'язку. **1**(59), 41–45 (2020). https://doi.org/10.26906/SUNZ.2020.1.041

13. Pérez, B., Porres, I.: Reasoning about UML/OCL class diagrams using constraint logic programming and formula. Inf. Syst. **81**, 152–177 (2019)
14. Silva, A.R., Savić, D.: Linguistic patterns and linguistic styles for requirements specification: focus on data entities. Appl. Sci. **11**(9), 4119 (2021). https://doi.org/10.3390/app11094119
15. Quinton, C., Vierhauser, M., Rabiser, R., Baresi, L., Grünbacher, P., Schuhmayer, C.: Evolution in dynamic software product lines. J. Softw.: Evol. Process **33**(2), e2293 (2021)
16. Tkachenko, O., Tkachenko, K., Tkachenko, O.: Designing complex intelligent systems on the basis of ontological model. In: Proceedings of the Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020), Zaporizhzhia, Ukraine, 27 April 27–1 May 2020, P. 266–277 (2020)