Towards Effectiveness Assessment of Domain Modelling Methods and Tools in SPL Development

Mykola Tkachuk[*,a,b], Rustam Gamzayev[a], Iryna Martinkus[a], Volodymyr Sokol[a], Oleh Tovstokorenko[a]

[a] Department of Software Engineering and Management Information Technologies, National Technical University "Kharkiv Polytechnic Institute", Ukraine
[b] Department of Systems and Technologies Modelling, V.N. Karazin Kharkiv National University, Ukraine

Abstract. *Domain-driven design (DDD) and especially the usage of domain modelling methods (DMM) are modern approaches to improve software quality, and a way to develop software product lines (SPL). To emphasize advantages of DDD and DMM usage, a 3-level design scheme is proposed, which reflects also variability issues in the framework. According to this metaphor the main attention is paid to the phases of logical domain modelling and physical modelling, with usage of two alternative DMM-methods: JODA and ODM approaches. The algorithmic model for an efficiency coefficient estimation of alternative DMM usage is proposed, which utilizes structured data resources, expert methods and metrics used in SPL development processes. A feasibility study for the proposed approach is provided and the obtained experimental results are discussed, which allow to make positive conclusions about this research and to outline its further steps to be done.*

Keywords. Domain Modelling Methods • Software Product Line • Variability • Structural Complexity • Efficiency Coefficient • Code Reusing • Metric

1 Introduction: Research Actuality and Goals

The largest amount of existing methodologies for software development, and in the first place modern agile-methods are supposed to achieve the following two main goals proposed by Sommerville (2011):

1. to decrease the project's costs with respect to all specified functional requirements and quality attributes to be implemented in a target software system;
2. to reduce the time needed for delivering of this software product on consumer market.

One of the most effective way to resolve this problem is reusing different project solutions (assets): domain knowledge, requirements specifications, software architectures, design patterns, and finally source (program) code. This approach is the basis of advanced concepts of software engineering such as the development of software products lines and software factories (Greenfield and Short 2004), as well as methods of software variability management (Capilla et al. 2013). In turn, in order to achieve an appropriate level of assets reuse in these processes, the methodology of domain-driven design (DDD) is widely used (Evans 2004; Tune and Millet 2015), where a concept of domain model (DM) as a core for conceptualization and reuse of expert knowledge within the given application area (universe of discuss - UoD) was elaborated: e. g. Karagiannis et al. (2016), Michael and Mayr (2015) and others. The

* Corresponding author.
E-mail. tka.mobile@gmail.com

DDD approach is successfully used to develop software system families (SSF) and software product lines (SPL), which is one of the a main trends in modern software engineering (Klaus 2016). It is to mention that these 2 notions sometimes are used as synonyms because they define a specific collection of software components, which have both general and specific functional properties, and include a special mechanism to provide software variability (Capilla et al. 2013) in order to be configured for multiple reuse for solving of different problem-specific tasks in an appropriate UoD.

But some authors, e.g. Kittlaus and Clough (2009) and Lee (2015) define an important difference between SSF and SPL, namely: the members of SSF as usually are used together to solve some tasks in an appropriate UoD, while single items of any SPL can be applied autonomously in particular software applications.

As a example of SSF the collection of service-oriented software solutions on cloud-based platform Oracle SOA Suite 11g (Oracle Corp n.d.) can be considered, and as a typical example of SPL a well-known office software package MS Office (Microsoft Corp n.d.) can be named. In the following some development problems of SPLs are examined, which are defined more correctly in the manuals of the world-wide recognized international organization in the IT-domain, the Software Engineering Institute at the Carnegie Mellon University (Software Product Line Conference n.d.), namely *«... software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way... ».* Due to these properties, the usage of SPL concepts provide, in particular, the following advantages in comparison with separate software system development, namely:

- less project costs (up to 60%),
- time decreasing for release of a software product on the market (in some cases up to 90%)
- significant reduction of IT-project's staff (up to 70%), and some others.

Taking the issues mentioned above into account, the main objective of the research presented in this article is to propose a sophisticated approach to effectiveness estimation of DMM usage in the SPL development. The remaining article is organized in the following way: Section 2 analyses briefly related work in the SPL domain, provides the classification of existing DMMs and shows the results of the comparative analysis of some CASE-tools used for support. In section 3 the proposed research methodology is outlined, which emphasizes variability issues in the domain modelling conceptual scheme, and includes a collection of heuristic assumptions combined with formalized specifications to define an efficiency coefficient for alternative DMM usage. In section 4 we present the algorithmic model for the estimation of this efficiency coefficient in different project situations, which is based on structuring and analysing of domain-specific knowledge about interconnected and complex data resources, expert methods and metrics. Section 5 presents the first implementation issues and results of the test-cases to prove the proposed approach. In section 6 the article concludes with a short summary and with an outlook on next steps to be done.

2 Software Product Lines Development with Domain Modelling: Related Work and Open Problems

2.1 Typical Structure and Features of Software Product Lines

Considering typical SPL structures (can be found in Klaus (2016)), usually its software components can be categorized into three main groups (see Figure 1), namely:

i constant components, which form so-called core of SPL;

ii variable components that already exist, and which can be customized for the specific usage with special features as a part of this SPL;

iii new components that have to be developed additionally in this SPL, taking into account new customer's functional requirements etc.

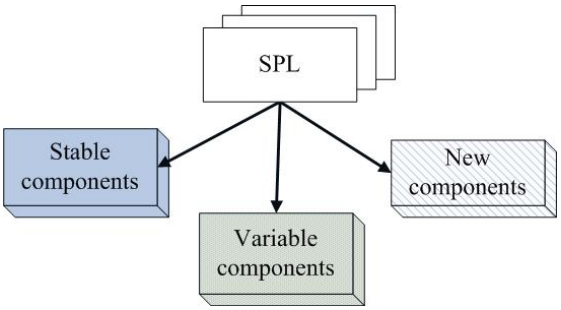

Figure 1: Reference SPL structure

Summarizing the research presented in Nagesh and Vivek (2016), components of any SPL can be divided into their three main types (i) – (iii) in the following way (see Figure 1):

- constant components (group i): they are the components, that have a number of functional changes not higher than 25%;
- variable components (group ii): these components have a number of functional changes within range of 25% - 75%;
- new components (group iii): they represent the components, that have a number of functional changes more than 75%.

Main problems of development and efficient SPL implementation are considered by many experts, e. g. in Pohl et al. (2010); Perovich et al. (2009); Bayer et al. (2004), and the relevance of those works proved by a representative international scientific and practical conference: The Software Product Line Conference (n.d.) (SPLC), which has been held annually for more than 20 years. So, in particular, among these problems the following can be identified:

- design and evaluation of SPL reference architectures,
- development of CASE-tools and code frameworks for SPL implementation and maintenance,
- advanced requirements management in SPL development,
- transformation legacy software into new SPL, and some others.

At the same time, the provided analysis of the obtained results in the SPL problem domain shows, that the following important issues still remain insufficiently elaborated, namely:

- building of quantitative metrics for SPL components complexity assessment, which have an impact on the degree of code reuse extent;
- determination of structural and functional complexity of a DM which is further used for code generation in the SPL development;
- elaboration of approaches to effectiveness estimation of alternative DMM usage in order to improve the quality of SPL construction.

These problems are considered more detailed in the following sections.

2.2 Classification of Domain Modelling Methods

During the last 10-15 years a lot of different domain modelling methods (DMM) were developed (Ferré 1999; Kelly and Tolvanen 2008). Despite of their differences from the design point of view, the most suitable way to classify DMMs, is to consider them by type of phases / artefacts to be reused in a software development process. Based on this suggestion, the following groups of these methods should be considered (see Figure 2):

1. DMM for requirements reusing;
2. DMM for architectures reusing;
3. DMM for assets reusing;
4. DMM for component reusing.

Taking our main research goal into account: to identify the effectiveness of different DMM usage in the SPL development, we have to consider the methods from two similar groups more detailed: (1) and (3) respectively. Therefore, two appropriate methods, namely JODA and ODM were chosen and briefly presented below. The JODA method (Joint integrated avionics Object oriented Domain Analysis (Ferré 1999)) uses an object -

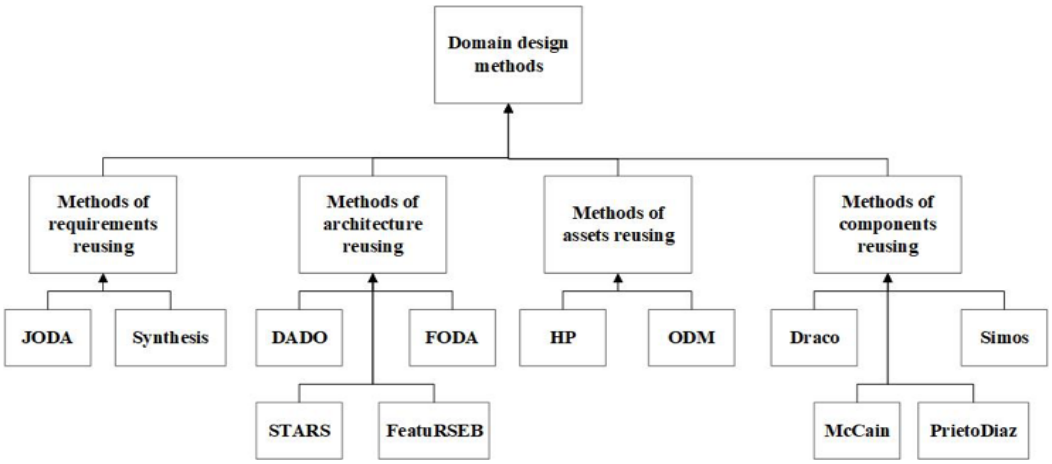

Figure 2: Taxonomy of domain modelling methods.

oriented approach to cover the domain analysis phase, and includes the following processes:

- Domain data preparation: identification and gathering of appropriate data sources, references and software artefacts, which are relevant for a given domain.
- Domain scope definition: elaboration of diagrams for higher-level entities, identify of generalization-specialization, aggregation and other relations within domain, build a domain glossary.
- Domain modelling: identification, definition and modelling several domain scenarios in order to group domain-specific objects and activities to represent them in next domain engineering process.

The ODM method (Organizational Domain Modelling (Ferré 1999)) systematically supports the mapping of domain-specific artefacts into reusable assets, that can be reused in future software development activities. This approach includes the following phases:

- Plan domain engineering: this one is focused on understanding stakeholders and defining the domain analysis scope.
- Domain modelling: it concerns collecting and documenting the domain-specific information resources which are relevant for future reusing.
- Domain assets base: the final phase of the ODM method that supposes defining the project scope, creating (choosing) system architectures and implementing a physical asset base for the given domain.

In order to support all main phases / activities with any DA&DSM method the appropriate CASE-tool has to be used, and a short overview of them is given in the next paragraph.

2.3 Comparative Analysis of Domain Modelling CASE-tools

Generally, visual modelling tools in Software Engineering have evolved a lot in recent years. One of the new trends in this domain is the transition from unified modelling environments like UML or SysML (OMG 2010), to domain-specific modelling (DSM) languages and tools, e.g. WebML, SoaML, and some others (Reinhartz-Berger 2013). These DSM - approaches allow developers to design and to analyse software in terms of a target problem domain, and finally to generate an application source code in different programming languages based on high-level requirements specifications. It is to mention that existing CASE-tools for DSM are quite varied in their capabilities,

and for comparing them, it is necessary to choose a set of criteria. Obviously there are a lot of different ways to define such criteria configurations, but generic enough and in the same time a practice-oriented one is the following list of criterion: a possibility to generate code by domain model, a possibility to build model by code, and last but not least mandatory license. The appropriate data about wide-used CASE-tools to support DMM are shown in Table 1.

Table 1: Results by comparative analysis of domain modelling CASE-tools

	Generate code by model	Build model by code	Mandatory license
Eclipse Modeling Framework	+	-	+
Rational Rose	+	+	+
FeatureIDE	+	+	-
Visual Paradigm	+	+	?
Actifsource	+	+	-

The legend here is the follows: "-" means "is not available"; "+" means “is available for free”; and "?" means “a proprietary license is needed”.

Taking into account the data presented in Table 1 as the dedicated CASE-tools for future DM implementation the Actifsource and Eclipse Modeling Framework (EMF) were chosen.

3 Research Methodology Outline

3.1 Variability Issues in Domain Modelling: A Conceptual Scheme

Nowadays DDD in considered as a recognized methodology to build a complex software in different application areas with respect to this important challenge: to provide a high level of assets reusability in a given project. Although main essential advantages and limitations of DDD are already discussed intensively in many recent publications, from our point of view the positive core of the DDD-methodology can be emphasized once again if we consider an analogy between the DDD-approach in software development and the well-known 3-level vision about data representation in database development (Batini et al. 1992) (see Figure 3).

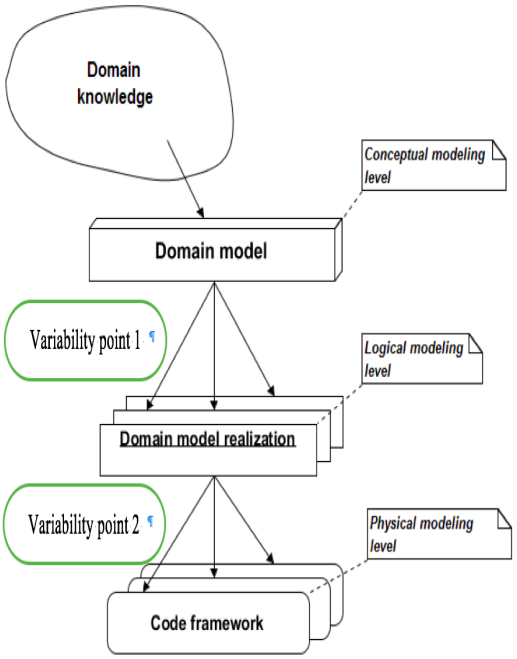

Figure 3: 3-level scheme in DDD approach.

Note that, according to this vision about the DDD approach for the one and the same domain model (at conceptual modelling level), a lot of different of its realizations (at the logical modelling level) can be constructed. And for each of them an appropriated code framework might be generated finally (at the physical modelling level) using CASE-tools. In fact, in this scheme we are facing with a variability problem (see in Figure 3 "Variability point 1" and "Variability point 2"), which is now one of the main challenges in modern software development (Capilla et al. 2013).

3.2 An Approach to Effectiveness Estimation of Domain Modelling Methods (DMMs): Heuristic Rules and Formalization

In order to evaluate an influence of different factors on effectiveness of DMM usage within a SPL development framework, it is necessary to take a

large number of complicated and poorly formalized information processes and objects into account. This is the reason why we propose to develop an appropriate algorithmic model for this purpose (Patdar 2014). The methodological basis for constructing this model is the following system of five heuristic assumptions, which were formulated based on the study of modern information sources and the generalization of our own experience in software development using the methods of domain modelling, namely:

Assumption 1. There exists a certain set of domain modelling methods (set M):

$$(DMM)_i \quad \epsilon \quad M, i = 1, 2, 3... \tag{1}$$

where DMM (DomainModelingMethod) is an identifier denoting a separate method, that can be used to construct an appropriate DM. There also exists the set of relevant technologies T:

$$(DMT)_j \quad \epsilon \quad T, j = 1, 2, 3... \tag{2}$$

where DMT (DomainModelingTechnology) is an identifier denoting a particular technology (with a CASE-tool) for implementing the appropriate DMM.

As a result of the application of a particular DMM with the corresponding DMT, the target DM from the set D can be constructed for the given UoD:

$$(DM)_{i,j} \quad \epsilon \quad D, \tag{3}$$

where $(DM)_{i,j}$ is a domain model obtained as a result of the application of the i-th DMM and j-th DMT respectively.

Assumption 2. All DM from the set D have different complexity levels, so there is such a mapping ρ:

$$p : D \rightarrow DMC, \tag{4}$$

where DMC (DomainModelComplexity) is a set of possible values of the quantitative level of structural and functional complexity of domain models.

Assumption 3. Based on each DM from a set D, an appropriate generated code framework can be obtained, so there is such a mapping φ:

$$\varphi : D \rightarrow GCF, \tag{5}$$

where GCF (GeneratedCodeFramework) is a set of program code frames that can be used for the construction of SPLs.

Assumption 4. All generated code frameworks from a set GCF have different code reusability extents (CRE), so there is such a mapping σ

$$\varphi : GCF \rightarrow CRE, \tag{6}$$

where CRE is a set of possible values of program code reusability extent.

Assumption 5. The efficiency coefficient of a certain DMM usage in SPL development can be defined as the ratio of the code reusability extent received using this DM, to the level of its structural complexity, namely:

$$(K)_{Eff}\left[(DM)_{i,j}, (GCF)_{i,j}\right] = \frac{(CRE)_{i,j}}{(DMC)_{i,j}} \tag{7}$$

where $(K)_{Eff}$ is an efficiency coefficient, and the variables $(CRE)_{i,j}$ and $(DMC)_{i,j}$ are defined by expressions (6) and (4).

3.3 Analysis of Relationship Between Software Quality Attributes, Complexity Metrics and Rate of Software Code Reuse

We have already shown, that the DDD approach for software development assumes a reuse of different project artefacts, which improves software quality attributes. A variety of projects asset types and their complex and hardly formalized relationships make quantitative analysis practically infeasible. Therefore, it defines a sophisticated and actual task in software engineering (see Sommerville 2011). In order to structure reusable artefacts and perform their qualitative analysis, the article proposes to use mind maps (Guerrero and Ramos 2015), which unlike more formalized approaches (such as UML, IDEF0, etc.) allows to represent

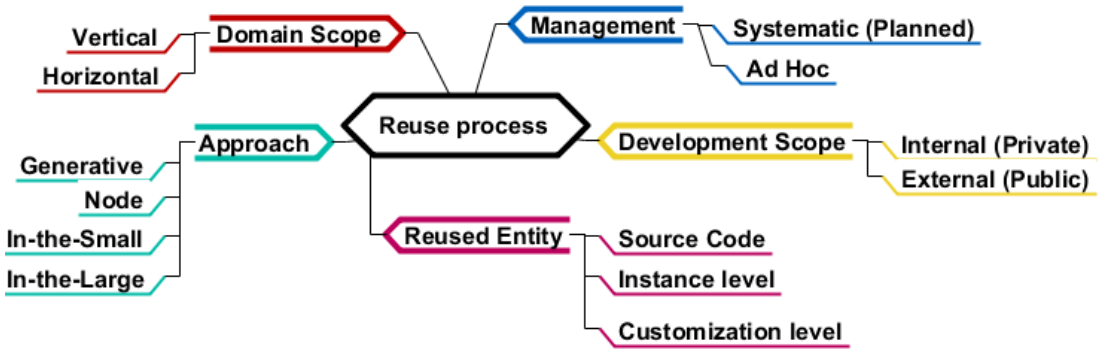

Figure 4: Software reuse artefacts.

in any UoD conceptual entities and their semantic relationships of any nature. The ideas about general classification and analysis of software reuse factors may be concluded from research performed in (Paliwal and Shrivastava 2014) and they are depicted as the mind map in Figure 4.

The following entities and their relationships, which qualitatively represent the software reuse process, are shown: *Development Scope* defines the source of reusable components (e.g., from the same project or not); *Approach* defines technical methods to be applied for the software reuse implementation; *Domain Scope* defines where software reuse takes place (e.g., within the same software family or between different ones); *Management* defines how a systematically software reuse process occurs; *Reused entity* defines the object type to be reused.

The next step in this research is to perform a qualitative analysis of relationships between software reusability and software quality attributes as maintainability, adaptability and understandability, as well as software structural complexity. The corresponding mind map for their qualitative analysis is presented in Figure 5. We may conclude that aforementioned software quality attributes, and therefore its ability for reuse, significantly depends on structural software complexity. For an object-oriented approach, it may be defined with the help of well-known metrics (Tkachuk et al. 2016a) as depicted on Figure 5.

Moreover, Preschern et al. (2014) define a correlation between the rate of software reuse and metrics such as DIT, RFC, NOC, CBO and WMC, where:

- DIT (Depth of Inheritance Tree) is defined as the longest path to the current class from the parent class in the class hierarchy.
- RFC (Responses For a Class) is the number of methods, which may be called from an object of the class.
- NOC (Number Of Children) is the number of direct subclasses of the class.
- CBO (Coupling Between Object classes) shows interaction between objects and defines a number of other related classes excluding subclasses.
- WMC (Weighted Methods per Class) is defined as a sum of all class methods, where each method is assessed by its cyclomatic number.

We have to note (Tkachuk et al. 2016a), that there is still a lack of comprehensive analysis in which way the different DMMs influence the complexity of source program code generated basing on an appropriate DM within the DDD software development. Therefore, it is crucial task to identify this correlation in order to reduce costs for DDD-oriented software projects and especially

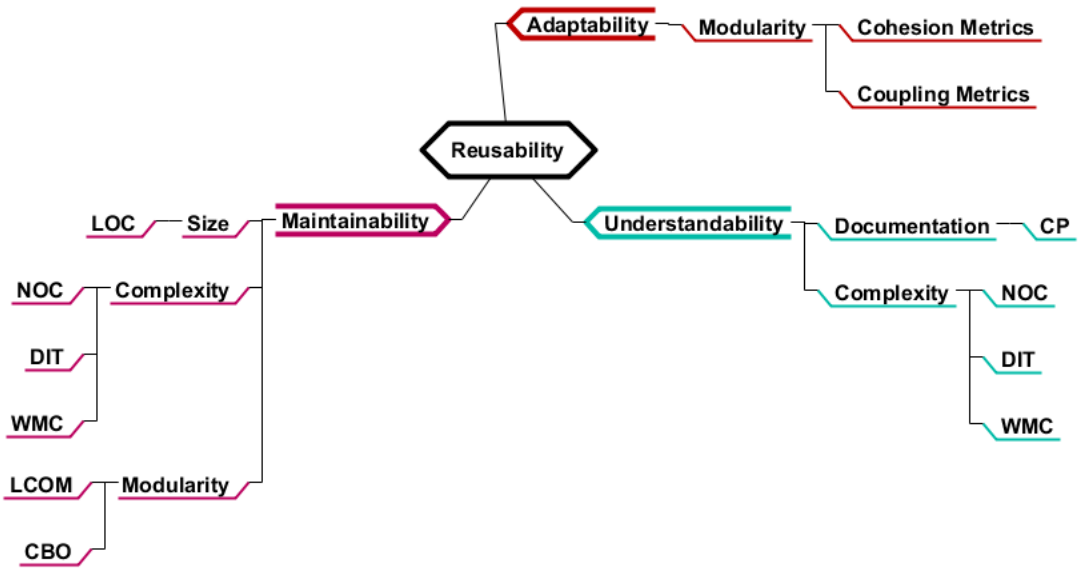

Figure 5: Quality attributes, metrics and their influence on the software reuse process

for SPL development. Consequently, taking the analysis results of relationships between quality attributes into account, complexity metrics and software code reuse artefacts shown in Figure 4 and Figure 5, we are able to provide the SPL development approach. It utilizes methods and tools for domain modelling support and uses a source code reusability extent as an effectiveness measurement criteria.

3.4 Cybernetic-based Technological Scheme for Software Products Line Development with Usage of DMM

The proposed schema for constructing a SPL, structured as an automated control system with a feedback loop, is finally given on Figure 6. Its main functional modules cooperate in the following manner:

- an initial description of a given UoD in the form of business requirements (as User Stories) to functionality of a target SPL serves as informational basis for building a DM on the conceptual level;
- DMMs (e.g., FODA, JODA, ODM, etc.) and appropriate CASE-tools or domain modelling tools (DMT), such as e.g. FeatureIDE, Actifsource etc., allow to provide a DM realization (DMR);
- a generated code framework (GCF) can be derived based on DMR, and after some changes (e.g., via code refactoring, applying code patterns, etc.) it should be used to build components of a target SPL;
- an efficiency coefficient (see formula (7)), which is used in the feedback control loop, allows us to analyse the domain modelling quality, and to make decisions how to choose the appropriate DMM and CASE-tool for the development of a target SPL.

It is worth to note, that the schema proposed in Figure 6 allows us also to use also other metrics than the proposed efficiency coefficient, in order to perform the analysis of various SPL implementations, and therefore may be seen as improvement of existing methods for solving variability issues in

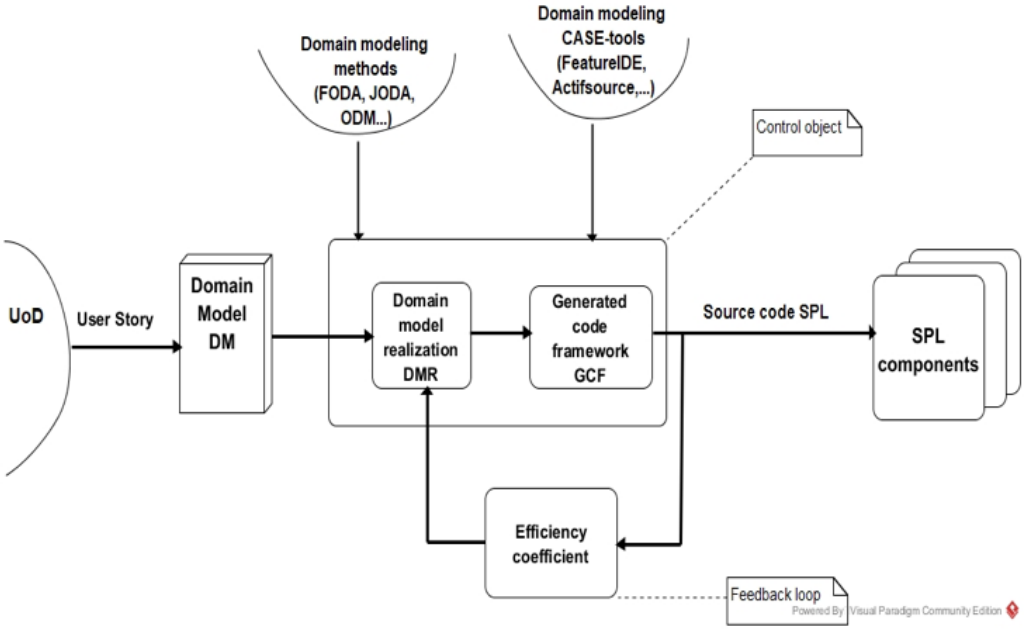

Figure 6: The proposed technological scheme.

software development with usage of an alternative DMM.

4 Algorithmic Model for Effectiveness Assessment of Domain Modelling Effectiveness

4.1 Definition of the Proposed Algorithmic Model

The functional dependence given with formula (7) can not be obtained analytically, therefore, for its investigation, it is necessary to develop a collection of information models, methods and metrics. The corresponding algorithmic model (AM) allows us to present all these components in a formalized way, and it can be presented as follows:

$$AM = \langle InfoBaseWorkFlow(Methods), Metrics \rangle \quad (8)$$

where InfoBase is an information basis of the AM, WorkFlow(Methods) is a set of algorithms (WorkFlow) that implement the appropriate methods (Methods) for assessing the effectiveness of DMM usage, and finally Metrics is a collection of metrics that are used in the corresponding algorithms of the AM model. The information basis of the algorithmic model AM can be presented in the form of the following tuple:

$$InfoBase = \langle M, T, D \rangle \quad (9)$$

where M is a set of methods of DM given in formula (1), T is a set of implementation technologies from formula (2), and D is a set of DM given in formula (3).

The collection *Methods* for evaluating the effectiveness in expression (8) includes:

- the method to determine the code reuse extent of the program code, which is proposed in Tkachuk et al. (2016a);
- the method to estimate the complexity level of a DM, which is given by the expression (4) and is discussed in more detail below.

Finally, the collection *Metrics* from expression (8) consists of:

- metrics for evaluating the structural complexity of the program code (Paliwal and Shrivastava 2014), which are used to analyse the frames generated with the help of the corresponding DM from expression (5);
- metrics for assessing the code reusability extent (Nandakumar 2016), that should be used to construct the corresponding SPL;
- metrics for complexity estimation of the DM (Preschern et al. 2014; Poels et al. 2001) from the set D given by expression(4);
- metrics to define an effectiveness coefficient of the DMM usage in the SPL development or maintenance given by expression (7).

The next step in the proposed approach is the direct development of a method for determining the complexity of DM.

4.2 Method for Estimation of Domain Model Complexity

Current publications show, that the problem of DM structural complexity assessment is very relevant both for stand-alone software systems, and for SPL and SSF development. Thus, in Preschern et al. (2014) the approach for an evaluation of DM complexity is proposed, that takes all its main structural components into account, namely:

- a number of DM objects types (#OT),
- a number of links between these objects (#ED), and
- a number of operations defined on them (#DO).

Further, the overall DM structural complexity index CD is considered as the ratio of existing associations to the number of object types: CD = (#ED) / (#OT), but at the same time the number of operations (#DO) is not taken into account while calculating CD and becomes a separate indicator of the DM complexity.

So, this approach does not allow to evaluate comprehensively the corresponding DM complexity. In Poels et al. (2001) a DM complexity problem is considered more detailed. Based on the methodology GOPPRR (Graph-Object-Property-Port-Role-Relationship) for evaluation of individual structural units of a DM, the authors offer their expressions for quantitative estimations:

$$\begin{gathered}(C)_{interface} = \#Relationship + \#Role + \\ + \#Constraints, \\ (C)_{element} = \#Objects + \#Ports, \\ (C)_{property} = \#Properties,\end{gathered} \tag{10}$$

where $(C)_{interface}$, $(C)_{element}$, $(C)_{property}$ are the quantitative indicators of complexity for structural DM elements. For a final evaluation of DM complexity the use of the following expressions is suggested:

$$\begin{gathered}(C)_{Overall} = (C)_{interface} + (C)_{element} + \\ + (C)_{property}\end{gathered} \tag{11}$$

However, at the same time this method does not take the functional DM complexity into account, but evaluates only its structure complexity.

Taking aforementioned disadvantages of existing methods for DM complexity assessment into account, it is necessary to propose an integrated approach to structural and functional model complexity estimation, which will allow to obtain a single integral index for this purpose.

In Tkachuk et al. (2016b) an approach for complexity of architectural model assessment in software systems is defined, that was developed using post object-oriented technologies (POOT). It provides a possibility to build analytical expressions to assess the common complexity of different POOT architectures based of their specific components and corresponding weight coefficients, which can be obtained with elaborated expert-oriented procedures. In our opinion, the similar approach can be applied to DM complexity assessment as well. In this way, we propose to introduce

formalized definitions for all major artefacts while constructing an appropriate DM. The small example of such a DM (as a fragment) is shown in Figure 7.

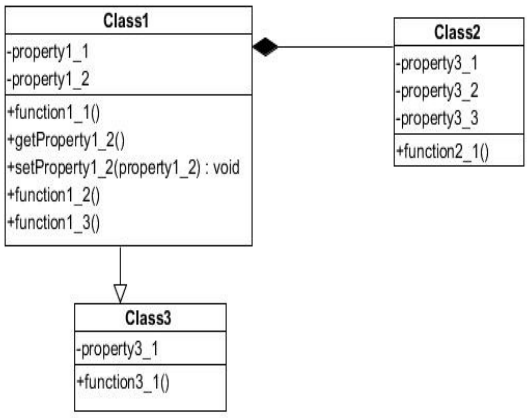

Figure 7: A fragment of a DM to be assessed

Definition 1. A Domain Model (DM) is a tuple:

$$DM = \langle C, R \rangle \tag{12}$$

where C is a set of classes, and R is a set of relationships between them.

Definition 2. Any class $c_k \epsilon C$ is a tuple:

$$c_k = \langle P^{(k)}, F^{(k)} \rangle \tag{13}$$

where $P^{(k)} = \left\{ p_i^{(k)} \right\}, i = \overline{1,n}$ is a set of class''s properties; $F^{(k)} = \left\{ f_j^{(k)} \right\}, j = \overline{1,m}$ is a set of functions (or methods) of a given class, which represent its functional content.

Definition 3. A set of links R between elements of the DM is a tuple:

$$R = \left\{ (r)_{i,j} \right\} \tag{14}$$

where $r_{i,j}$ is the connection between classes i and j in this DM. At the same time, each element of the set R refers to one possible element in the set RT.

Definition 4. A set RT defines all types of relationship which can be defined between different classes:

$$RT = \{As, Ag, Cm, In\} \tag{15}$$

where As is the name of the Association, Ag is the name for the Aggregation, Cm is the name for the Composition, and In is the name for Inheritance relationships in a given DM.

In (Kang et al. 2004) a quantitative analysis was carried out to estimate the impact of these types of relationships on an objects behaviour in the corresponding DM. Analytically, this dependence can be expressed in the following way:

$$(H)_{As} < (H)_{Ag} < (H)_{Cm} < (H)_{In} \tag{16}$$

where $(H)_{As}, (H)_{Ag}, (H)_{Cm}, (H)_{In}$ are the coefficients of the influence degree for the corresponding type of communication on the overall DM complexity. The proposed method realizes the mapping ρ (see the expression (4)), that in turn supposes to provide the following calculations:

1) the complexity value for each class included in a given DM,
2) the complexity value for all relationships between classes in this DM,
3) the overall complexity value for a given DM.

The appropriate weight coefficients for each type of DM components to be assessment can be determined in an expert-centred way, e. g. using the analytical hierarchy method (Saaty 2000). The steps (1)-(3) are presented below in more details.

Step 1. Calculation of the complexity for each class. As the influence of functional class properties on the overall DM complexity is more important, the complexity of each DM class can be obtained using the following expression:

$$CC = 0.3 \times PC + 0.7 \times FPC, \tag{17}$$

where CC (ClassComplexity) is a parameter that determines quantitatively the overall structural and functional class complexity; PC (PropertyComplexity) is a parameter that determines the

complexity of class properties (structural complexity); FPC (FuncPropertyComlexity) is a parameter that determines the complexity of functional properties of a given class (the behavioural complexity of a class). Similarly, a quantitative impact on class complexity (its atomic attributes), and their collections cen be represented by the following expression:

$$PC = 0.4 \times \#STP + 0.6 \times \#CTP, \quad (18)$$

where #STP (SimpleTypeProp) is a number of properties counted for any "simple" types (no collections), #CTP (CollectionTypePrperty) is the number of properties counted for any collection types. The definition of any DM provides its functions signature only (without their implementation details). For this reason, it is enough to calculate their number for the determination of their impact on the overall separate class complexity. By this fact, the following expression is offered:

$$FPC = \#FP, \quad (19)$$

where #FP (FunctionalProperty) is the number of functional properties of the class (operations).

Step 2. Calculation of the complexity for each relationship According to expression (14) following weighting factors for each DM class connection types can be determined:

$$(W)_{As} = 0.1, (W)_{Ag} = 0.2, \\ (W)_{Cm} = 0.3, (W)_{In} = 0.4 \quad (20)$$

Given the expression coefficients from (18) the overall connections complexity of a certain DM can be calculated using the following expression:

$$RC = (W)_{As} \times (\#As) + (W)_{Ag} \times (\#Ag) + \\ + (W)_{Cm} \times (\#Cm) + (W)_{In} \times (\#In) \quad (21)$$

where RC (RelationalComplexity) is a parameter, that determines the overall link complexity of DM; #As (AssociationRelation) is the number of association type connections; #Ag (AggregationRelation) is the number of aggregation connections, #Cm (CompoistionRelation) is the number of composition connections; and #IR (InharitanceRelation) is the number of inheritance connections.

Step 3. Structural and functional DM complexity Calculation. For the final evaluation of structural and functional DM complexity, we suggest the following expression:

$$DMC = 0.7 \times \#Class \times CC + 0.3 \times RC \quad (22)$$

where DMC (DomainModelComplexity) – is a parameter, that represents the overall structural and functional DM complexity and #Class – an amount of DM classes. The analytical expressions (12) – (22) define an algorithm for implementing a method which determines structural and functional DM complexity, which is a component of the algorithmic model AM given by expression (8).

4.3 Method for the Assessment of Code Reusability Extent

In Tkachuk et al. (2016a), an approach for the assessment of code reusability extent (CRE) is proposed, which is generated using DMM and appropriate CASE-tools. This method has three main phases, namely: 1) elaboration of a DM based on the UoD description using initial business requirements, e.g. given in form of user stories; 2) evaluation of source code complexity by using the OOP code metrics mentioned before; 3) calculation of a target CRE in form of their summarized values with weighted coefficients, which have to be defined using, e. g. the analytical hierarchy method. As a result, the following analytical expression was obtained to determine the integrated value of CRE:

$$CRE = 0.12 \times WMC + 0.04 \times RFC \\ + 0.27 \times DIT + 0.36 \times NOC + \\ + 0.21 \times CBO \quad (23)$$

where WMC, RFC, DIT, NOC and CBO are the OOP code complexity metrics mentioned before (see in section 3.3 for more details).

5 Feasibility Study of the Proposed Approach: Implementation Workflow and Experimental Results Discussion

5.1 Workflow for the Implementation of the Proposed Approach

Based on the algorithmic model represented by expression (8), taking the collection of its components defined by (9) – (14) into account, and using the method for structural and functional DM complexity evaluation, which was presented by (15) – (22), it is possible to elaborate a procedure for the determination of an effectiveness coefficient of the DMM usage given in formula (7). This procedure is presented in form of an UML activity diagram in Figure 8.

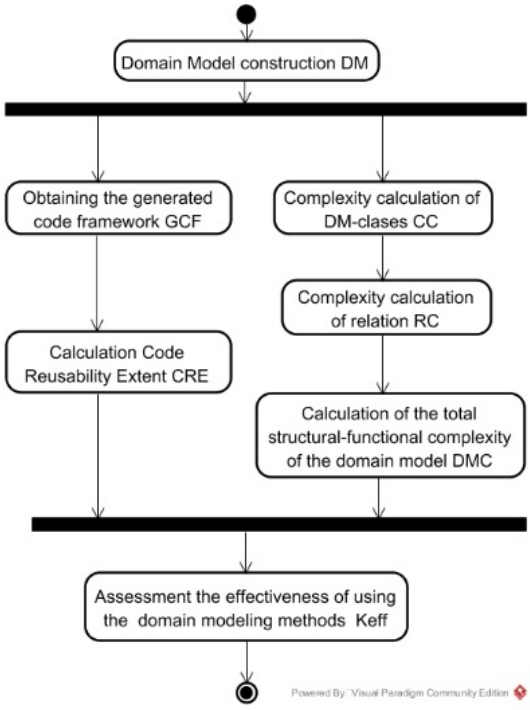

Figure 8: Workflow for the proposed approach.

According to expression (7), for the definition of $(K)_{Eff}$ it is necessary to calculate two variables: CRE and DMC respectively. The activities needed to calculate the DMC parameter is described in the previous section. The activity «Obtaining generated code framework GCF» realizes the mapping φ(see expression (5)), and it can be provided with help of appropriate domain modelling tools (see in Section 2.3). Further, the activity «Calculation of code reusability extent CRE» (see expression (6)) realizes the mapping σ, using the appropriate metrics given in formula (21).

5.2 Use case Domain Models and Experimental Results Discussion

To test the proposed approach, two use case DMs were developed for the UoD «Students Personal Data Processing in Education Management System» using two alternative DMMs: ODM (Organizational Domain Modeling) and JODA (Joint integrated avionics Object oriented Domain Analysis), which can be implemented using CASE - tools as EMF (Eclipse Modeling Framework) and Actifsource (see sections 2.2 and 2.3). The example of generated Java-source code in Figure 9, and the DM elaborated for JODA / Actifsource technology is presented in Figure 10.

This Java - code contains two packages of classes: student.javamodel (provides the implementation and interfaces) and student (realizes the additional resource classes).

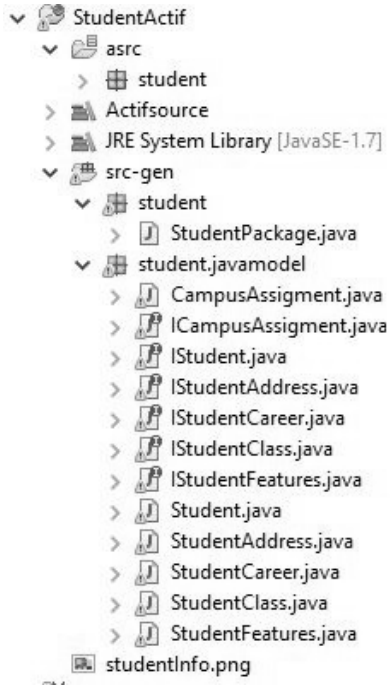

Figure 9: Generated code framework (GCF).

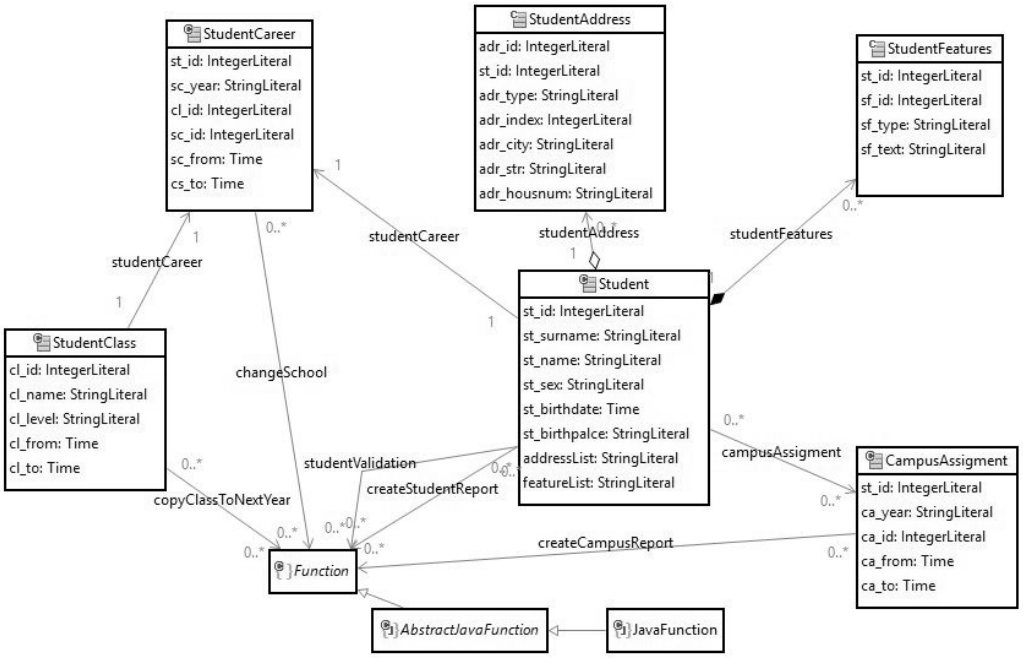

Figure 10: The example of JODA / Actifsource domain model

According to the proposed procedure for efficiency coefficient calculation (see Figure 8), it is necessary to get such DM characteristics as DMC and CRE: see formulas (22) and (23). The appropriate experimental data obtained based on the elaborated DM using the JODA / Actifsource tool are presented in Table 2 and Table 3, and for the ODM / EMF tool in Table 4 and Table 5.

Table 2: DM classes complexity assessment for JODA /Actifsource realization

	Class	#STP	#CTP	#FP
1	Student	6	2	2
2	StudentAddress	7	0	0
3	SturdentFeature	4	0	0
4	StudentCareer	6	0	1
5	StudentClass	5	0	1
6	CampusAsignement	5	0	1
7	Function	0	0	0
8	AbstractFunction	0	0	0
9	JavaFunction	0	0	0
	$\sum$	**33**	**2**	**5**

Table 3: DM relationships complexity assessment for JODA / Actifsource implementation

Association	8
Aggregation	1
Composition	1
Inheritance	2

Table 4: DM classes complexity assessment for ODM / EMF implementation

	Class	#STP	#CTP	#FP
1	Student	6	2	2
2	StudentAddress	7	0	0
3	SturdentFeature	4	0	0
4	StudentCareer	6	0	1
5	StudentClass	5	0	1
6	CampusAsignement	5	0	1
	$\sum$	33	2	5

Table 5: DM relationships complexity assessment for ODM / EMF implementation

Association	3
Aggregation	0
Composition	2
Inheritance	0

Based on the parameters presented in the Tables 2-5 and using the formulas (16) – (22) the following values for the elaborated DM were obtained:

for JODA / Actifsource realization

$PC(JODA/Actifsource) = 0.4 * 33 + 0.6 * 2 = 14.4$
$FPC(JODA/Actifsource) = 5.0$
$CC(JODA/Actifsource) = 0.3 * 14.4 + 0.7 * 5 = 7.82$
$RC(JODA/Actifsource) = 0,1 * 8 + 0.2 * 1 + 0,3 * 1 + 0,4 * 2 = 2,1$
$DMC(JODA/Actifsource) = 0.7 * 9 * 7.82 + 0.3 * 2.1 = 49,9$

for ODM/EMF realization

$PC(ODM/EMF) = 0.4 * 33 + 0.6 * 2 = 13.2 + 1.2 = 14.4$
$FPC(ODM/EMF) = 5.0$
$CC(ODM/EMF) = 0.3 * 14.4 + 0.7 * 5 = 4.32 + 3.5 = 7.82$
$RC(ODM/EMF) = 0,1 * 3 + 0.2 * 0 + 0,3 * 2 + 0,4 * 0 = 0,9$
$DMC(ODM/EMF) = 0.7 * 6 * 7.82 + 0.3 * 0.9 = 33.11$

At the same time, it is necessary to get the values of code reusability extent (CRE) for these two alternative DMs. To calculate CRE, formula (23) should be used after the appropriate OOP metrics were defined (see Tkachuk et al. 2016a for more details), and the following values of the CRE were obtained:

$$CRE(JODA/Actifsource) = 7.78$$
$$CRE(ODM/EMF) = 16.67$$

Using expression (7), the final values of efficiency coefficients for the tested DMs are calculated (see Table 6).

In graphical representation, these results are shown in Figure 11.

Table 6: Efficiency coefficients values obtained

	CRE	DMC	$(K)_{eff}$
JODA / Actifsource	7,78	49.90	0.156
ODM / EMF	16,67	33.11	0.503

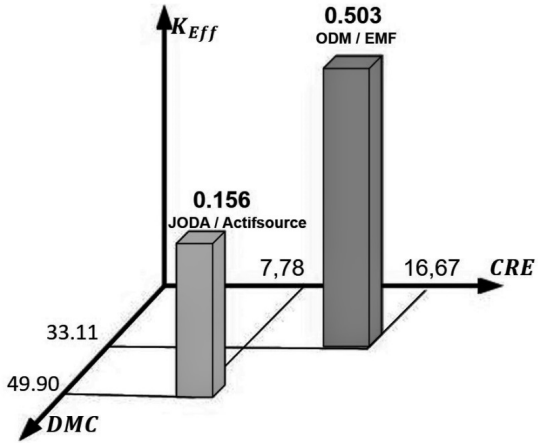

Figure 11: Graphical interpretation of the results obtained.

Based on this feasibility study we can conclude, that domain modelling with usage of ODM / EMF technology provides a higher rate of efficiency in comparison with the JODA / Actifsource tool, and these results should be taken into account for domain-driven development of a target software product line.

6 Conclusions and Future Work

In this paper we have considered essential aspects of domain-driven development (DDD), and especially, the usage of domain modelling methods (DMM) in modern software engineering with respect to the improvement of software quality, and as a way to develop software product lines (SPL). To emphasize advantages of DDD and DMM usage, a 3-level design scheme is proposed, that reflects variability issues in this framework. According to this metaphor, the main attention is paid to the phases of logical domain modelling level and to physical modelling, with usage of two alternative DMM-methods: JODA and ODM approaches. The algorithmic model for an efficiency coefficient estimation of alternative DMM

usage is proposed, which utilizes structured data resources, expert methods and metrics used in the SPL development process. A feasibility study for the proposed approach was performed, and the obtained experimental results are shown, which can be useful in the development of a target SPL.

In future, we are going to construct a more sophisticated collection of software complexity metrics, e. g. to take the metrics of relationships between packages into account, and to improve the prototype of our software tool to support the proposed evaluation approach.

Acknowledgments

The authors, former visiting researchers and students at Alpen-Adria Universität Klagenfurt in Austria, would like to express their special gratitude to Prof. Dr. Dr. h.c. Heinrich C. Mayr for stimulating discussions of problems related to the topics presented in this paper.

We would like also to thank students of National Technical University "Kharkiv Polytechnic Institute" A. Tkachuk and M. Raldugyn for their support in software implementation to test our approach.

References

Batini C., Ceri S., Navathe S. (1992) Conceptual Database Design: An Entity-Relationship Approach. Benjamin Publishing Company

Bayer J., Kettemann S., Muthig D. (2004) Principles of software product lines and process variants. Process Family Engineering in Service-Oriented Applications.BMBF-Project. In: PESOA-Report No. 03/2004.

Capilla R., Bosch J., Kang K. (2013) Systems and Software Variability Management. Springer

Evans E. (2004) Domain-Driven Design Tackling Complexity in the Heart of Software, 1st ed. Prentice Hall

Ferré X. (1999) An Evaluation of Domain Analysis Methods. In: 4th CAiSE.IFIP8.1 International Workshop in Evaluation of Modeling Methods in Systems Analysis and Design, pp. 1–13

Greenfield J., Short K. (2004) Software Factories: Assembling Application with Patterns, Models, Frameworks and Tools. Wiley, Indianapolis

Guerrero J., Ramos P. (2015) Mind Mapping for Reading and Understanding Scientific Literature. In: International Journal of Current Advanced Research. 4(11), pp. 485–487

Kang D., Xu B., Lu J. (2004) A Complexity Measure for Ontology based on UML // Distributed Computing Systems. In: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, pp. 222–228

Karagiannis D., Mayr H., Mylopoulos J. (2016) Domain-Specific Conceptual Modeling: Concepts, Methods and Tools. Springer

Kelly S., Tolvanen J. (2008) Domain-Specific Modeling: Enabling Full Code Generation. Wiley Computer Society Press

Kittlaus H.-B., Clough P. (2009) Software Products: Terms and Characteristics. In: Software Product Management and Pricing.. Springer

Klaus P. (2016) Learning and Evolution in Dynamic Software Product Lines. In: 11th Int'l Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) Carlo Ghezzi, Sam Malek (eds.)

Lee S. U.-J. (2015) An Effective Methodology with Automated Product Configuration for Software Product Line Development, Article ID 435316. In: Mathematical Problems in Engineering 2015

Michael J., Mayr H. C. (2015) Creating a Domain Specific Modelling Method for Ambient Assistance. In: International Conference on Advances in ICT for Emerging Regions (ICTer2015). IEEE, pp. 119–124

Microsoft Corp Accessed on: 15.05.2017 https://www.microsoft.com/uk-ua/download/office.aspx

Nagesh P., Vivek S. (2016) An Approach to Find Reusability of Software Using Objet Oriented Metrics. In: International Journal of Innovative Research in Science, Engineering and Technology 3(3) Tiwari K. (ed.)

Nandakumar A. (2016) Constructing Relationship between Software Metrics and Code Reusability in Object Oriented Design. In: International Journal of Advanced Computer Science and Applications 7(2)

OMG (2010) OMG Unified Modeling Language, Superstructure. Version 2.3. OMG

Oracle Corp Accessed on: 15.05.2017 http://www.oracle.com/technetwork/middleware/soasuite/overview/index.html

Paliwal N., Shrivastava V. (2014) An Approach to Find Reusability of Software Using Objet Oriented Metrics. In: International Journal of Innovative Research in Science, Engineering and Technology 3 K. T. (ed.)

Patdar S. M. (2014) Literature Survey on Algorithmic Methods for Software Development Cost Estimation. In: Int. J. Computer Technology & Applications 5(1), pp. 183–188

Perovich D., Rossel P., Bastarrica M. (2009) Feature model to product architectures: Applying MDE to Software Product Lines. In: Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture., pp. 201–210

Poels G., Dedene G., Viane S. (2001) A Quantitative Assessment of Complexity Of Static Conceptual Schemata For Reference Types Of Front-office. In: In Proceedings of the Fifth International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE2001)

Pohl K., Bockle G., Van Der Linden F. (2010) Software product line engineering: Foundations, Principles, and Techniques. Springer

Preschern C., Kajtazovic N., Kreiner C. (2014) Evaluation of Domain Modeling Decisions for two identical Domain Specific Languages. In: Lecture Notes on Software Engineering 2(1), pp. 37–41

Reinhartz-Berger I. (2013) Domain Engineering: Product Lines, Languages, and Conceptual Models. Springer

Saaty T. (2000) Fundamentals of the Analytic Hierarchy Process. RWS Publishing

Software Product Line Conference Accessed on: 15.05.2016 http://splc.net/

Sommerville I. (2011) Software Engineering. Addison-Wesley, Boston, MA

Tkachuk M., Martinkus I., Gamzayev R., Tkachuk A. (2016a) An Integrated Approach to Evaluation of Domain Modeling Methods and Tools for Improvement of Code Reusability in Software Development. In: Mayr H. C., Pinzger M. (eds.) INFORMATIK 2016. Lecture Notes in Informatics (LNI) Vol. P-259. Kollen Druck, Verlag GmbH, pp. 143–156

Tkachuk M., Nagorniy K., Gamzayev R. (2016b) Models, Methods and Tools for Effectiveness Estimation of Post Object-Oriented Technologies in Software Maintenance. In: ICTERI 2015: Revised Selected Papers, Series title: Communications in Computer and Information Science. 594 Yakovyna V. (ed.), pp. 20–37

Tune N., Millet S. (2015) Patterns, Principles And Practices Of Domain-driven Design, 1st ed. John Wiley & Sons