UDC 614.2+574/578+004.38

ESTIMATION OF EFFORT OF MIGRATION AMONG DOMAIN-DRIVEN DESIGN ARCHITECTURAL VARIATIONS

Lytvynov O. A. – PhD, Associate Professor, Faculty of Physics, Electronics and Computer Systems, Oles Honchar Dnipro National University, Dnipro, Ukraine. ROR: https://ror.org/00qk1f078. ORCID: 0000-0001-7660-1353.

Khandetskyi V. S. – Dr. Sc., Professor, Head of the Department of Electronic Computing Machinery, Oles Honchar Dnipro National University, Dnipro, Ukraine. ROR: https://ror.org/00qk1f078. ORCID: 0000-0002-6386-4637.

Lytvynov M. O. – Master, Post-graduate student, Faculty of Physics, Electronics and Computer Systems, Oles Honchar Dnipro National University, Dnipro, Ukraine. ROR: https://ror.org/00qk1f078. ORCID: 0009-0000-9765-1501.

ABSTRACT

Context. The article addresses the issue of effort estimation of migration among variations of DDD architecture using a method based on specifications of requirements to increase the predictability of the software migration process.

Objective. The goal of the work is to propose an effective method of effort estimation based on Use Case analysis.

Method. First, a set of rules for rigorous Use Case description adapted for software effort estimation needs are provided. Second, the modified Use Case metamodel and the method of use cases classification based on frame-based knowledge representation model are also suggested. The rigorous description allows to make the estimation of the use cases more precise, using FUSP method, and to build individual predictors for each class of use cases. Thirdly, the method uses historical data taken from previous iterations of the same project, and is based on three trends (optimistic, pessimistic and mean based trend).

Results. The result is the collection of functions used to predict the effort required for the next iteration (measured in person-hours) for each class of use cases.

Conclusions. FUSP method was adapted for task of gaining greater prediction accuracy of effort estimation for migration among variations of DDD architecture using a methodology based on specifications of requirements. The set of conditions to form the Use Case description rules adapted for software migration effort estimation needs is developed. The modified Use Case metamodel and the method of use cases classification based on frame-based knowledge representation model are suggested. It was proposed algorithm for building the individual predictors of each class and for corresponding effort estimation. The coefficient of FUSP person-hours transformation is based on three trends achieved and updated considering the results from previous iterations: the most pessimistic prediction is based on the upper bound, the lower bound predictor plays the role of the optimistic predictor, and the main trend is the meaning among these. The coefficients are used to predict the effort in person-hours required for the next iteration for each class of use cases. The results of experiment, conducted using the test RTP project for this class of software, showed that MMRE for the proposal method is 0.0343, and for the standard method – 0.1094. The obtained results evidence that the classification of use cases along with their rigorous description according to provided rules, and modification of the method by separating prediction logic in accordance with the use case classes makes the prediction more accurate and can be effectively used for effort estimation for DDD architectural variations migration.

KEYWORDS: Use case point, Software Effort Estimation, Fuzzy Use Case Size Point, Domain-Driven Design.

ABBREVIATIONS

DDD is a Domain Driven Design;
FUSP is a fuzzy use case size points;
SDE is a software development effort;
CMMI is a capability maturity models integration;
LOC is a lines of code metric;
FP is a functional points metric;
UCP is an Use-case point;
USP is an Use-case Size Point;
TAF is a Technical Adjustment Factor;
EAF is an Environmental Adjustment Factor;
NL is a natural language;
DTO is a data transfer object;
RTP is a Representative Test Project;
MVP is a minimum valuable product;
CRUD is a set of create, read, update and delete operations;
API is an Application Programming Interface.

NOMENCLATURE

MRE is a magnitude of relative error;
$\hat{y}_i$ is a predicted value;
y_i is a known real value;
i is an iteration;
n is a number of iterations;
x is a value which is considered to be 0.25;
$MMRE$ is a mean magnitude of relative error;
$PRED(x)$ is a prediction criteria within x%;
TPA is a total complexity of actors;
$TPPrC$ is a total complexity of the preconditions;
PCP is a complexity of the main scenario;
$TPCA$ is a total complexity of the alternative scenarios;
TPE is a total exceptions complexity;
$TPPoC$ is a total complexity of postconditions;
$UUSP$ is a numer of Unadjusted Use-case size points;
w_i is a weight of the i-th Technical / Environmental Adjustment Factor;
v_i is a degree of domination of the i-th TAF / EAF;
$\mu_A(x)$ is a degree of membership of element x in fuzzy set A, which in our case is defined by a trapezoidal membership function;

DOI 10.15588/1607-3274-2026-1-14

$FUSP$ is a number of fuzzy use-case size points;

E is an effort in person-hours;

k is an empirical coefficient used to translate FUSP to person-hours;

t is an index of the current iteration;

$t-1$ is an index of the previous iteration;

k_R^t is an actual ratio of effort E_R^t (person-hour) to FUSP^t for t-th iteration;

$E_{\max}^{t+1}$ is a predicted maximum effort needed to realize FUSP^{t+1} in the next iteration;

$E_{\min}^{t+1}$ is a predicted minimum effort needed to realize FUSP^{t+1} in the next iteration;

E_{avg}^{t+1} is a basic prediction.

INTRODUCTION

Today information systems have become an integral part of modern business, the increasing complexity and agility of which require advanced system flexibility, functionality, scalability to provide complete business processes automation and maintainability. One of the basic approaches focused on tackling business domain complexity is DDD [1] approach which encompasses a wide range of architectural variations. Some of the variations are well known [1–3], some appear as a result of their adaptation to pursue specific tasks and undertake certain projects. The situation is complicated by the fact that the requirements changes can cause architectural evolution or even migration of the project to new architecture. According to [4] architecture should be ready to react to changing requirements and end-users and developers feedback by the guided, controllable evolution in the ways to implement more effective solutions and disallow it evolving in a way that harms architectural concerns. In the case of DDD architectural variations it means that managers and developers need to ensure that the architectural modifications are objectively necessary and to estimate the efforts needed to perform the evolution. Accurate estimation of SDE has a strong impact on cost, schedule, functionality and quality of the software [5] and this work is devoted to estimating an effort for DDD architectural variations migration on the example of a project based on the application service-oriented variation of DDD to the Use Case focused variation.

The object of study is evolutionary architecture and architectural migration of information systems.

The subject of study is the problem of the effort estimation of the migration among DDD variations.

The purpose of the work is to provide a method of migration effort estimation of DDD architectural variations based on modified FUSP method.

1 PROBLEM STATEMENT

This work is devoted to the development of methodology for estimation of the complexity of migration among DDD architectural variations using modified FUSP method. In particular, we consider a migration of project based on the application service-oriented variation to the Use Case oriented variation).

The research questions answered by this study are as follows:

RQ1: Which factors can improve effort estimation accuracy using FUSP method based on specifications of requirements in the case of software migration assessment?

RQ2: What strategy of estimation tuning should be used in case of iterative development process?

For the effort estimation accuracy we used the *MMRE* and Percentage of Prediction within x% [6,7], which are the two most common metrics used in software engineering. Both parameters are based on the quantity called *MRE*, which is described by Equation 1.

MMRE and *PRED*(x) are shown in Equation 2 and Equation 3 – respectively.

$$MRE_i = \frac{\left|\hat{y}_i - y_i\right|}{y_i}, \tag{1}$$

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n}, \tag{2}$$

$$PRED(x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1, \text{if } MRE_i \le x;\\ 0, \text{otherwise.}\end{cases} \tag{3}$$

According to [8], an accurate effort prediction model should have an $MMRE \le 0.25$ (to ensure that the mean estimation error should be less than 25%) and $PRED(0.25) \ge 0.75$ (meaning no less than 75% of the predicted values with *MRE* are lower than 0.25. Some authors [9] also take into consideration $PRED(0.3)$.

2 REVIEW OF THE LITERATURE

The goal of software effort estimation is to make software development and maintaining process more manageable, and predictable, which is necessary to increase the maturity of the company up to 4-th level of CMMI [10].

According to [11] the software effort is directly proportional to a project size and complexity and inversely proportional to team productivity (4):

$$\text{Effort} = \frac{\text{Size} \cdot \text{Complexity}}{\text{Productivity}}. \tag{4}$$

The question of measuring software size is not trivial, and different metrics can be used, e.g. LOC, FP, Halstead Metrics, McCabes cyclomatic complexity etc.

Among several categories of methods used to estimate the effort (e.g. expert judgement, top-down, bottom-up, design to cost, parametric models), methods of estimation based on specifications of requirements stand out with

DOI 10.15588/1607-3274-2026-1-14

their relative simplicity (they don't require providing software components and algorithms specifications as input data) and ability to estimate software size and effort at early phases of software development.

Evolutionary migration among DDD variations can be regarded as a software development process which allows us to take Requirements Specifications as the main source of information and use appropriate software effort estimation methods.

The basic method of estimation based on specifications of requirements is UCP [12] which has a number of modifications. For example, USP [13], FUSP [14], fuzzy-UCP [15], and ANFUSP [16]. The complete list of methods can be found in [17, 6].

The reason for such variety of modifications relates to four main problems [6] connected to the UCP method foundation.

PRB1. Defining of the complexity in the Use Case models.

PRB2. Adjustments of the Technical Complexity Factors and Environmental Complexity Factors.

PRB3. Classification of Use Cases. It is complicated to define the metrics, e.g., many variations of use case specification style and formality measure the length of a use case. For example, in [10] author argued that defining the UCP model's elements (actors and use cases), and assigning weights to them are the main problems to obtaining a reasonably accurate estimate.

PRB4. The fourth is how to make the right choice in a productivity factor whose units are person-hours for the estimated effort from UCP size measurements – especially when the historical dataset is not available.

In [18] all UCP based Effort Estimation models are classified into three main groups of approaches that focus on UCP factors: adding more complexity levels for the use case/actor weight; discretizing existing complexity levels into more detail options; empirically calibrating complexity weights to the different complexity levels.

FUSP is the extension of UCP which combines two approaches. It is focused on the details of the use-case structure: measures the functionality by counting the number and weight of scenarios, actors, precondition and post conditions [13]. To determine the USP, the sections of the use-case are analyzed, and the following steps are made to calculate USP [19]:

Step 1. *TPA*, *TPPrC*, *PCP*, *TPCA*, *TPE* and *TPPoC* are calculated using Table 1. *UUSP* value is calculated using formula (5)

$$UUSP = TPA + TPPrC + PCP + TPCA + TPE + TPPoC \quad . \qquad (5)$$

Table 1 – Unadjusted Use-case Size Point components classifications

Actor complexity (TPA)		
Complexity	Data (provided to or received from the Use Case)	USP
Simple	≤ 5	2
Average	[6;10]	4
Complex	≥ 10	6
Precondition complexity (TPPrC)		
Complexity	Expressions tested by the condition in precondition	USP
Simple	≤ 1	1
Average	[2;3]	2
Complex	≥ 4	3
Main and alternative scenarios (PCP, TPCA)		
Complexity	Entities + Steps	USP
Very simple	≤ 5	4
Simple	[6;10]	6
Average	[11;15]	8
Complex	[16;20]	12
Very complex	≥ 21	16
Exceptions (TPE)		
Complexity	Tested expressions	USP
Simple	≤ 1	1
Average	[2;3]	2
Compex	≥ 4	3
Postconditions		
Complexity	Entities related with postcondition	USP
Simple	≤ 3	1
Average	[4;5]	2
Complex	≥ 6	3

DOI 10.15588/1607-3274-2026-1-14

Step 2. *TAF* can be calculated using Table 2 by Equation (6)

$$TAF = 0.65 + \left(0.01 \cdot \sum_{i=1}^{14} w_i \cdot v_i \right). \quad (6)$$

Step 3. *EAF* can be calculated using Table 3 by Equation (7)

$$EAF = 0.01 \cdot \sum_{i=1}^{5} w_i \cdot v_i . \quad (7)$$

Step 4. The final value for a use-case is given by Equation (8):

$$USP = UUSP \cdot (TAF - EAF) . \quad (8)$$

The use of the classification tables does not allow a gradual change from one complexity category to another. The issue can be resolved by Fuzzification which allows us to solve this problem of classification which is made through the generation of a trapezoidal fuzzy number to each complexity category found on the classification tables [19, 20].

A fuzzy set is represented by a membership function (formula (9)). Each element will have a grade of membership that represents the degree to which a specific element belongs to the set (Fig. 1):

$$Fz[x \in A] = \mu_A(x) : \mathbb{R} \to [0;1], \quad (9)$$

Table 2 – Technical factors

Factor	Requirement	Influence	Weight
F1	Data communication	I1	2
F2	Distributed processing	I2	1
F3	Performance	I3	1
F4	Equipment utilization	I4	1
F5	Transaction Capacity	I5	1
F6	On-line input of data	I6	0.5
F7	User efficiency	I7	0.5
F8	On-line update	I8	2
F9	Code reuse	I9	1
F10	Complex processing	I10	1
F11	Easiness of deploy	I11	1
F12	Easiness operation	I12	1
F13	Many places	I13	1

Table 3 – Environmental factors

Factors	Description	Influence	Weight
E1	Formal development process existence	I1	1.5
E2	Experience with the application being developed	I2	0.5
E3	Experience of the team with the used Technologies	I3	1
E4	Presence on an experienced analyst	I4	0.5
E5	Stable requirements	I5	1
E6	Part time workers	16	2
E7	Motivation	17	–1
E8	Difficulty in programming language	18	–1

DOI 10.15588/1607-3274-2026-1-14

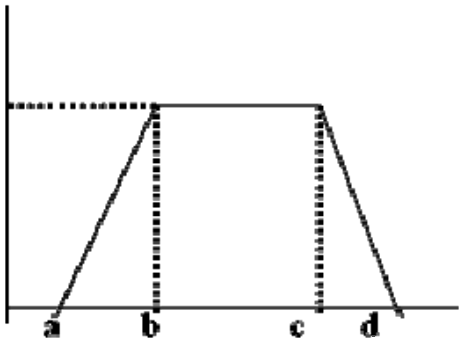

Figure 1 – Trapezoidal Membership function

$$\mu_A(x)=\begin{cases}0, x\le a,\\ 0, x\ge d,\\ 1, b<x\le c,\\ \dfrac{x-a}{b-a}, a<x\le b,\\ \dfrac{d-x}{d-c}, c<x\le d.\end{cases}, \quad (10)$$

Then the effort can be calculated using formula (11)

$$E = FUSP\cdot k\ . \quad (11)$$

In our case of migration among architectural variations of DDD, the method won't be effectively applied to the selected task. The reasons are as follows.

1. To predict the effort, we need to know k, which cannot be based on the historical data of the previous projects, e.g. we cannot use the coefficient applied to realized projects for the estimation of migration purposes, considering that the relationship between effort and size cannot be described by linear function [21].

2. Some factors such as Actor Complexity, Precondition and Postcondition Complexity don't affect the business logic layer and can be omitted. They are required to consider mainly during the development process, not for migration.

3. Technical factors and most environmental factors are oriented to whole project realization tasks and do not cover the specific of business layer and tasks of migration.

Thus, this paper is focused on adaptation of the FUSP method [14, 13] for estimation the efforts of the project migration among DDD variations. The main attention is given to the problems PRB1, PRB3, and PRB4 described above.

3 MATERIALS AND METHODS

The first problem (PRB1) and partially the third one (PRB3) are connected to Use Case definition rules.

The use case model documents the majority of software and system requirements and serves as a contract between stakeholders about the envisioned system behavior [22]. This model consists of two parts: a diagram which represents the relationships among the use cases and use cases descriptions represented as a structured document written in NL. The usage of NL can lead to poor descriptions such as ambiguous, inconsistent and/or incomplete descriptions which lead to missing requirements and eliciting incorrect requirements as well as less comprehensiveness of produced use case models [23]. In works [23, 24] authors proposed a catalog of symptoms of poor descriptions. However, this method cannot solve the problem of effort estimating the requirements development maxim states that requirements shall not consider design aspects.

In [25, 26] noted that to build automated verification of use cases the textual description is not sufficient. In [25] authors provide seven different types of steps: atomic, choice, concurrent, goto, include, success, failure. In [26] use-cases are instrumented by annotations, which are represented by tags (semantical markers) appended to a particular use-case step.

In [10] author asks the main question connected with UCP method issue: How should actors and use cases be defined if the use case model will be used for estimation? Answering the question, the author argued that we cannot ignore the existing architecture and some aspects of the project if we want to obtain accurate enough estimates.

Based on [10] proposition and considering the approaches and strategies mentioned above we provide the following set of rules to the Use Case definition.

Single responsibility of steps. Each step of the Use Case should be defined in accordance with Single Responsibility principle, i.e. responsible for atomic actor or system activity.

Rigorous steps identification. The architectural solution involves use case definition by introducing some additional steps to reflect the specifics of system's behavior. E.g. if the system uses event-driven notification mechanism the steps connected with notification preparation, publishing and handling cannot be omitted or absorbed by other steps.

Alternative flows definition. All the alternatives are represented at least by two steps (condition check and exception handling in case of <abort>).

Use cases decomposition. Some use cases should be split in order to omit repeated logic. However, before the estimation the steps connected to other use cases invocation should be represented as a sequence of atomic steps.

Entity consideration. We provide fixing the relationship between steps and entities which correspond to basic FUSP method.

Implicit use cases. Architecture can involve the requirements by introducing additional use cases. For example, when bounded contexts interact using events mechanism, it causes the introduction of integrated events handling use cases.

Steps and Rules connection. The step responsible for checking business rules should be accompanied by only one alternative. If it is connected to several alternatives, it should be split.

In result informal Use Case definition is transformed into rigorously defined Use Case considering architectural specifics. Modified metamodel of Use Case presented in [27] is shown in Fig. 2.

DOI 10.15588/1607-3274-2026-1-14

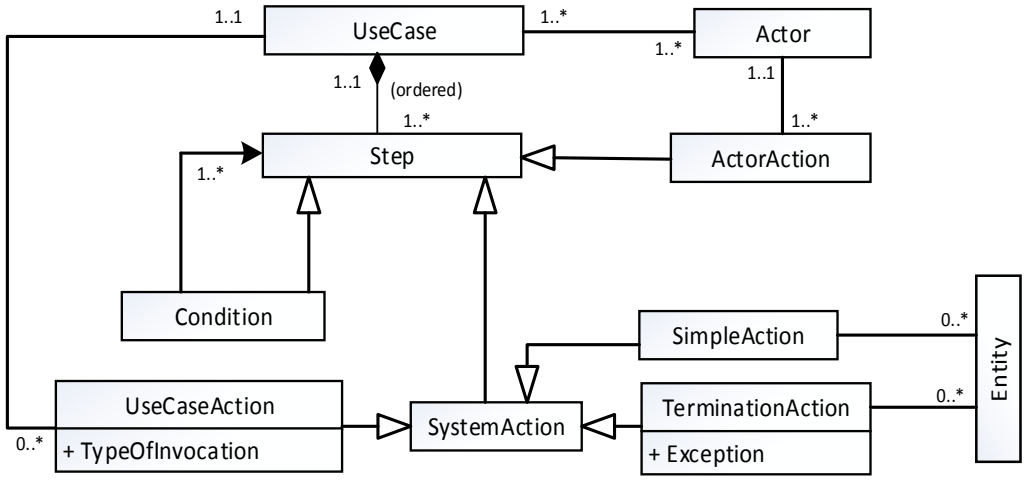

Figure 2 – Metamodel of Use Case

TerminationAction defines the termination of the actor request handling (in the most of cases it means exception handling). Alternatives are resolved using Conditional Steps which are often connected with business rules testing and termination actions (the case when before the termination some actions should be executed is also considered by the model). The step may trigger conditional (extension relationship case) and unconditional (include relationship case) invocation of another use case (see UseCaseAction). Some actions can be connected with entities.

Then we provide to classify the use cases and define the frames (knowledge description structures) connected with the classes of use cases. Each frame is comprised of several slots which represent types of steps, some of which are required, and some are optional which can be represented by the capacity of the slot. Required steps have the capacity equal to 1 or 1..* (one or many). Optional steps have the capacity equaled to 0 or 0..*. For example, the frame which describes a set of use cases of Modify type considering DDD architecture is shown in Table 4.

Table 4 – Modify Use Case Frame

Baseline Step	Alternative		Capacity	Number of steps	Entity
	Condition	Action			
1.Command validation	1a: Validation failed	1a1: Notify	1	3	0
2.Fetch data	2a: Object doesn't exist	2a1: Notify	1	3	1..*
3.Check timestamp	3a: Timestamp is not valid	3a1: Notify	1	3	0
4. Check simple rules	4a: Check simple rules failed	4a1: Notify	0..*	0..*	0
5. Check average rules	5a: Check average rules failed	5a1: Notify	0..*	0..*	0..*
6. Check complex rules	6a: Check complex rules failed	6a1: Notify	0..*	0..*	1..*
7. Check data integrity	7a: Check data integrity failed	7a1: Notify	0..*	0..*	1..*
8. Modify data			1..*	1..*	1..*
9. Prepare events			0..1	0..1	0
10. Publish events			0..1	0..1	0
11.Return response			1	1	0

It means that all the instances of Modify Use Case Type have at least 5 slots which will include at least 11 steps. Some authors [15] proposed to count the number of transactions, omitting the Condition, i.e. an alternative represented as condition-actions would only be counted by the number of actions. Of course, the number of use case types may not be restricted by the create, update, delete and select use cases.

Command validation step addresses input command validation logic performed by the request handler. Fetch data step means entity or several entities retrieval from persistence storage or from cache. Check timestamp step is connected to checking the version of the object to prevent race conditions issues. Checking simple rules means checking simple business rules in the scopes of the fetched data. For example, check the required fields are filled out, validate the data, checking the data on consistency. Checking average rules means checking business rules, involving additional data sources to perform the task. For example, checking the uniqueness of the modified object implies the additional request to the data source or checking the collection of entities. Checking complex rules means using a third-party service to perform checking. For example, checking a credit card balance using bank service. Checking data integrity means checking if the requested operation may violate data integrity. For example, referential integrity. It is worth noting that each rule should be represented as a separate step with the appropriate alternative. For example, when we have several rules connected with data checking, they should not be represented within one step, because it violates single responsibility principle. In our opinion, the main driver for such separation is the exceptions. Modify data slot is responsible for data modification that is the

DOI 10.15588/1607-3274-2026-1-14

core logic of the Use Case. In some cases, it could be connected to transactional modification of several entities, which may reside in several data sources and even perform distributed transactions. Prepare events slot means the preparation of the integration events to publishing (in some cases it may be extraction of domain events from an aggregate after performing the modification, publishing them and translating into integration events; in some cases, creating integration events at the application service level). Publish events slot means delivering events to subscribers (of course, here we primarily mean actors). Return response means creating and returning a response DTO to customer.

Example of fuzzification is shown in Fig. 3. The linguistic variables are not restricted by the 5 basic classes presented in Fig. 3 and could be extended to make the prediction more precise.

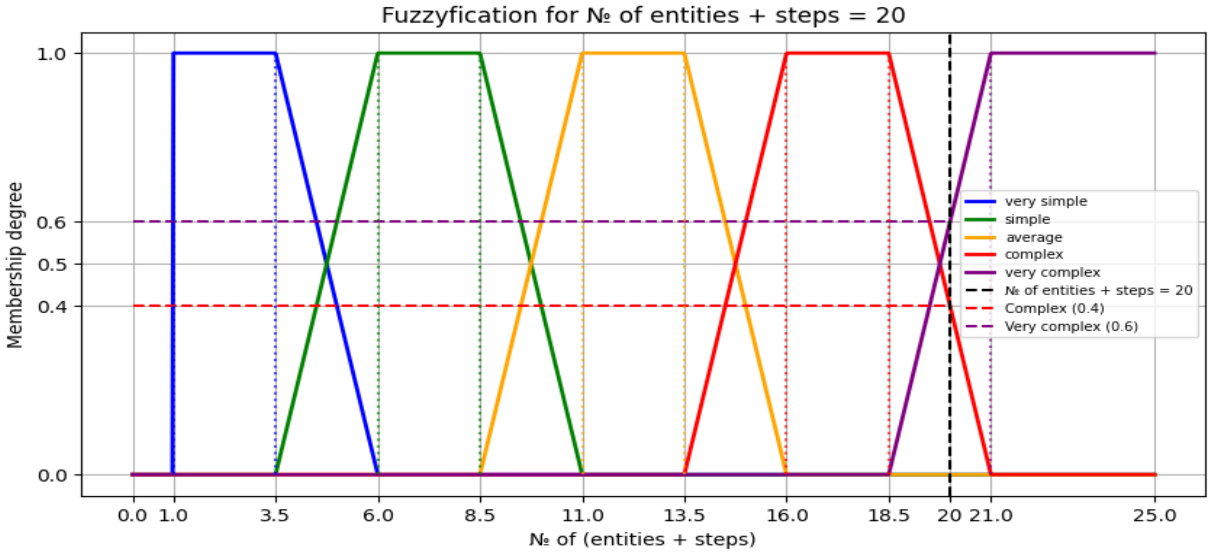

Figure 3 – Example of fuzzification

We propose a methodology which is purposed to predict the effort needed to perform every next iteration of the migration process.

The effort prediction is based on three coefficients used to transform *FUSP* to person-hours: minimal effort estimation, maximum effort estimation, basic effort estimation. While the minimum and maximum effort estimation functions establish bounds of the prediction, the basic effort estimation considers the history of the estimation (in our case we use mean of the values). The definition of the coefficients is represented by the formulas (12)–(15)

$$k_R^t = \frac{E_R^t}{FUSP^t}, \tag{12}$$

$$k_{\max}^t = \begin{cases} k_{\max}^{t-1}, k_R^t \le k_{\max}^{t-1}; \\ k_R^t, \quad k_R^t > k_{\max}^{t-1}. \end{cases} \tag{13}$$

$$k_{\min}^t = \begin{cases} k_{\min}^{t-1}, k_R^t \ge k_{\min}^{t-1}; \\ k_R^t, \quad k_R^t < k_{\min}^{t-1}, \end{cases} \tag{14}$$

$$k_{\text{avg}}^t = \frac{1}{t}\sum_{i=0}^{t-1} k_R^{t-i}. \tag{15}$$

The composition of indices R and t indicates real effort obtained at t-th iteration while indices min, max, avg in combination with t show that the coefficients are used to predict the effort (optimistic, pessimistic, average) for t-th iteration.

The prediction of the effort needed to perform the next iteration is based on the above coefficients as follows (16)–(18)

$$E_{\max}^{t+1} = k_{\max}^t \cdot FUSP^{t+1}, \tag{16}$$

$$E_{\min}^{t+1} = k_{\min}^t \cdot FUSP^{t+1}, \tag{17}$$

$$E_{\text{avg}}^{t+1} = k_{\text{avg}}^t \cdot FUSP^{t+1}. \tag{18}$$

4 EXPERIMENTS

The experiment is based on RTP [28], which is technical station management system, derived from real projects, considering preferable, basic scenarios occurred in real projects of DBB software company [29]. The method of obtaining RTP is as follows. Software projects of the company are classified by architectural solutions and technologies. Each project within the class can be represented by MVP [30] which is used to validate developers' hypotheses about customer needs serving the fastest way to get through the Build-Measure-Learn feedback loop with the minimum amount of effort and the least amount of development time. MVP is regarded as the core of the specific project which can be evaluated by the end-user. MVP projects of the same class, in turn, can be developed on the base of the same prototype. In accordance with [31] RTP relates to evolutionary incremental prototype.

DOI 10.15588/1607-3274-2026-1-14

RTP is not valuable to the customer, because it is primarily focused on typical task resolution in order to help produce MVP as quickly as it is possible, serves as a foundation needed for quick start. In daily practice this concept is known as labs, test projects or sample projects. Figuratively speaking, RTP is a seed from which MVP projects can grow. The connection of RTP and MVPs of several projects of class A is schematically shown in Fig. 4.

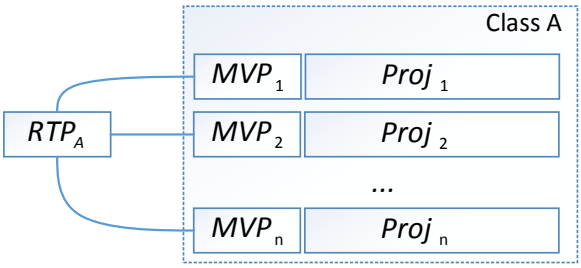

Figure 4 – Representative test project definition

We believe that there is not only one way to extract RTP from a set of projects, but the way we propose is based on use case classifications with mapping the use case classes onto architectural components. In result we get a collection of frames (the example is shown in Table 4). Then we examine the entities and the specifics of their interrelation trying to find a pattern which covers other variants. The domain used for RTP does not matter; the entities may not have all fields the end-use is interested in. The main purpose is to obtain the structural and behavioral pattern of the project.

The metamodels [32] of source and target architectural DDD variations are presented in Fig. 5 and Fig. 6.

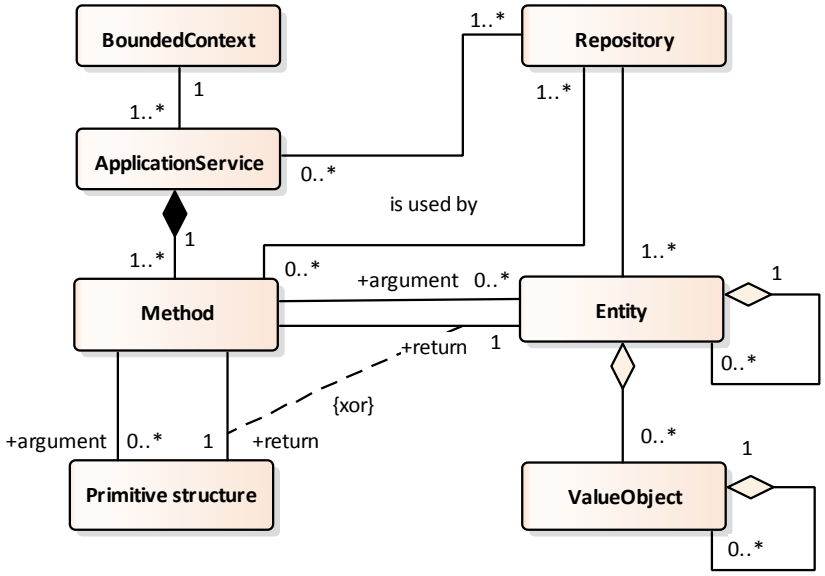

Figure 5 – Application service oriented architectural variations

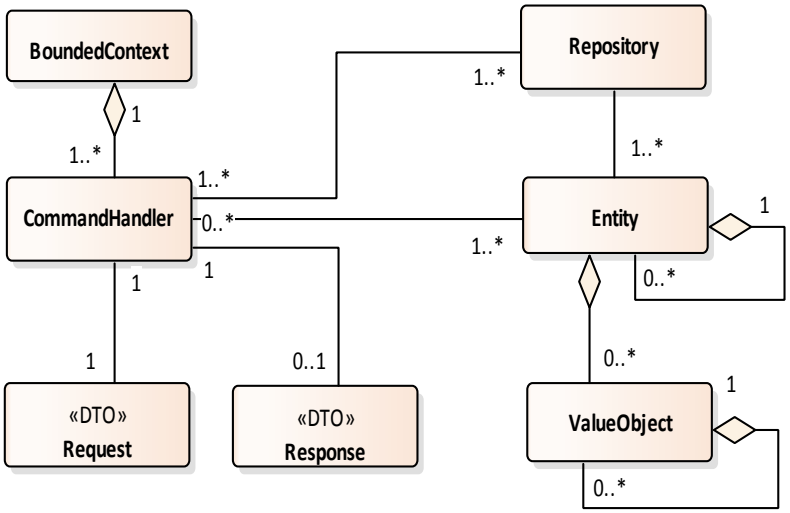

Figure 6 – Use case oriented architectural variations

DOI 10.15588/1607-3274-2026-1-14

The specifics of the source architectural solution are as follows. First, CRUD oriented application services are traced to multiple use cases connected with a certain entity. CRUD operations form four fundamental classes to project information management. According to [28] many use cases start their life as simple CRUD use cases, which later might grow into a full-fledged business use case with rich behavior, but significant part of use cases might remain simple. CRUD orientedness does not contradict DDD, which allows using CRUD for bounded contexts with low complexity [33]. Second, anemic domain model entities can be passed to the application services methods as arguments. These entities obtained by the mapping of DTO at the API controller side. The main reason for that is excluding the introduction of an additional layer of DTOs (which practically double the anemic entities), responsible only for passing the request to the application service methods from the higher system layer. Thirdly, the services are injected into API controllers or hubs using dependency injection mechanism. Controllers are responsible for input, which is represented as an API level DTO, validation, transforming the DTO into domain objects and passing them to appropriate application services. It is worth noting that API level DTO cannot be used as objects to business logic layer, because of several reasons, e.g. supporting several different API with different object representations and naming conventions.

The advantages of the above variation are as follows. Because of the excluding business logic level DTO objects the number of classes is reduced, consequently saving the time and efforts of the developers. The methods of Application services can share the same logic, e.g. validation, check integrity logic etc. In the case of CRUD operations, the testing infrastructure is simple, and the main logic of Test Fixtures may be represented as a generic class. In addition, we can say that the application services and anemic domain entities can be effectively generated using code generator with structure-oriented description model [34–35].

The main disadvantage of the solution is a certain distance from the use case definition (referring to the requirements, business rules) and their code implementation. Use case, which describes the interaction between the actor and the system, is triggered by an actor's request, includes different scenarios-paths comprised of several steps and finishes with a kind of response. The request can be represented as a DTO object or as an event (in case of event-driven architecture using pub-sub pattern) and these DTO structures could not be mapped directly to Entities. The response can be provided as a DTO object and several events used to notify the internal and external (i.e. actors) system components (for example using Event Bus).

Considering the agility of the business which causes the requirements changes, the problems connected with the completeness of the requirements, developers face the challenge of finding an effective process and mechanisms able to respond to functional requirements uncertainty. One of such mechanisms is to make software realization closer to use case definition, mapping the use case to separated class. This idea is implemented using several approaches (e.g. transaction script pattern, CQRS command handlers), including DDD variations, e.g. [12]. The metamodel of the target architectural DDD variation is presented in Fig. 6.

CommandHandler is a separate class which is responsible for task realization connected with a certain use case. The realization of the CommandHandler can be based on Command Bus pattern which provides a mechanism for executing commands in an application in a decoupled and extensible way. By the Use Case we mean the projection of the conventional Use Case onto the Business Logic. It is worth noting that the logic of the methods is comprised of definite blocks of operations which can be classified, and which can be thought of as slots of the frame which represents Method, in case of the first variation, or the CommandHandler in the case of the second one. Some of these slots can be required for definite types of use cases, some are optional. In most cases these slots are connected to the appropriate use case steps or transactions which can also be classified. At this point we face the necessity of the normalization of the use cases in order to make them more rigorously defined.

We restrict the RTP to only one Bounded Context because the logic connected to Bounded Contexts interaction remains the same. We don't consider any technological and environmental changes. In our case, we have 5 entities (Fig. 7) and four basic Use Cases (get, add, modify, remove) connected to them. Totally we have 20 Use Cases. Why can the above combination of entities be considered as a pattern of entity relationships for other projects?

The core of the schema is Worker-Work-Order relationship. Customer and Car entities represent an infrastructure which at first glance may be omitted. But the Customer-Car and Car-Order relationships cannot be covered by the Worker-Work-Order relationship. It seems that Car-Order relationship is similar to Worker-Work relationship, but they are not: Work entity depends on the Order as well.

The combination of entities is common for multiple Bounded Contexts: there could be a number of 1–0..* Car-Order like and several Order-Work-Worker like relationships. Of course, there could be variations of relationships, for example, an entity which depends on more than two entities, but it can be considered as an extension of the Order-Work-Worker relationship type. In Fig. 8 represented relationships examples: a) is covered by the extended Order-Work-Worker relationship; b) taken from [2] is covered by three Customer-Car-Order relationships.

DOI 10.15588/1607-3274-2026-1-14

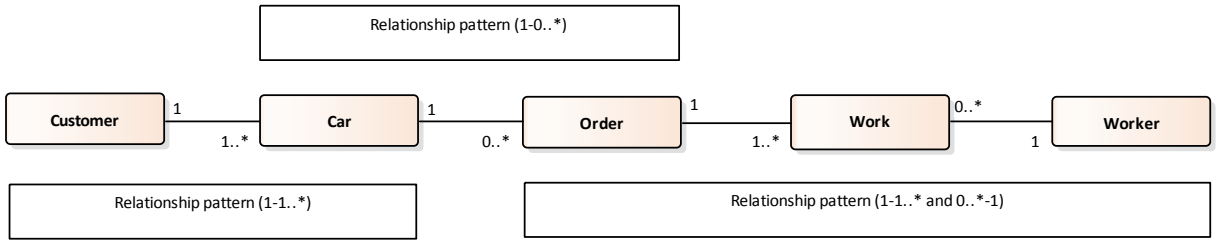

Figure 7 – Main entities of RTP and their relationships

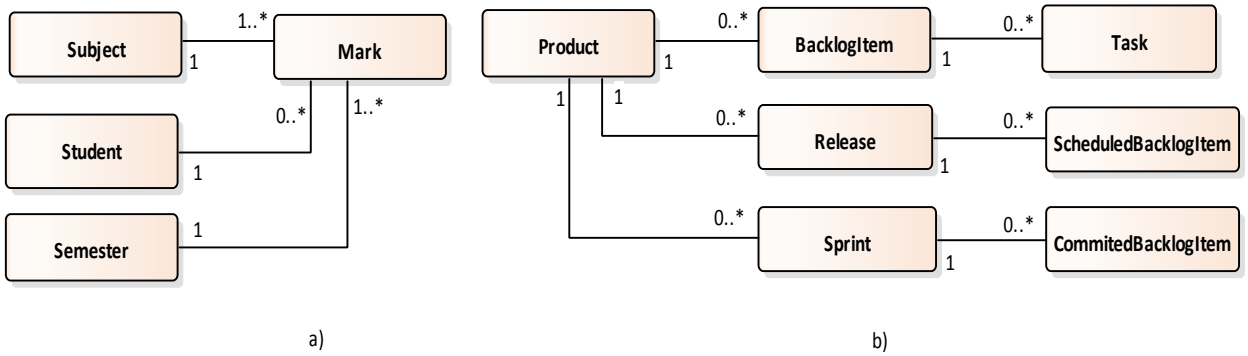

Figure 8 – The relationships covered by the Order-Work-Worker relationship

5 RESULTS

To estimate the effort required to migrate from source architectural variation into the destination one we used the modification of use case size points method [13] without using *TCF–ECF* correction. To check our hypothesis, we calculated *TAF* and *EAF* for DDD architecture as follows. The technical factors (Table 2) in case of selected projects class are: On-line update = 2; Data-communication = 2; Code reuse = 1; Easiness to deploy = 1. Environmental factors are: Stable requirements = 1; Experience with the technologies = 1; Formal development process = 1.5; Experience with the application being developed = 0.5

$$TAF = 0.65 + \left(0.01 \cdot \sum_{i=1}^{14} w_i \cdot v_i \right) = 0.65 + 0.01 \cdot (2+2+1+1) = 0.71,$$

$$EAF = 0.01 \cdot \sum_{i=1}^{5} w_i \cdot v_i = 0.01 \cdot (1+1+1.5+0.5) = 0.03 .$$

Thus, *TAF – EAF* adjustment is equaled to 0.68.

The provided modification of FUSP method is as follows. Firstly, we provide a method of Use Case classification and normalization based on single responsibility pattern applied to steps classifying them in accordance with the slot types of the operation frame (e.g. Method, CommandHandler). In result we acquire 4 basic Use Case patterns. Secondly, we separate effort estimations into four different predictors with their own coefficients used to translate obtained use case size points into manhour. Thirdly, we propose an incremental method of the coefficients correction which considers project realization dynamic. The strategy of coefficients correction is based on different bounds estimation: upper, average, minimal bounds. In tables and charts bellow we can see the application of the method to 20 use cases connected to 5 entities.

Let's see the specific of the evaluation on the example of Modify Use Case type. The evaluation of the modify type use cases in FUSP for 5 entities is shown in Table 5. The structure of the modify use cases is defined by the frame presented in Table 4: all required slots are defined, all capacity rules are satisfied.

Let us consider the above description in more detail. Command validation for all the entities is 3-step slot, because the validation is connected to one alternative scenario (condition and notification). Fetching data, checking timestamp and simple rules (testing required fields and fields values conditions) are also estimated as 3 step slots, because checking the conditions implies the existence of an alternative. In case of Customer entity checking average rules slot filled out with the logic of testing the request against the rule "There could not be two customers with the same name in the system" which requires querying the Customer entities collection using repository. In case of Car entity, it is also necessary to check the rule of entity uniqueness which states "There could not be two different cars with the same number". The situation with Worker entity is the same as with Customer entity. There is no checking complex rules logic which implies the connection with third-party services (e.g. banking service), using API etc. Check data integrity means checking referential integrity. In our case Car entity modification requires checking CustomerId reference, whether it is valid or not (in the case when one customer sells the car to another customer it changes). Order has a reference to the Car (CarId field), and Work entity has two references to check: WorkerId (who performs the work), OrderId

DOI 10.15588/1607-3274-2026-1-14

(what order the work belonged to). All slots starting from Modify data and finishing with Publish events one has only one step. The number of entities depends on Fetch data and Check data integrity slots. Thus, in case of Car entity to check the validity of CustomerId the handler requires to interact with the additional Customer entity collection. The same situation with Order (additional Car entity collection) and Work (additional Worker, Order entity collections) entities.

Fuzzification process for Customer Modify Use Case is shown in Fig. 9. The continuous classification is presented in Table 6, considering [19], where p_i – lower value of the linguistic ter T_i in the classification (Table 1, section Main and alternative scenarios); $n_i = (p_i + p_{i+1})/2$; $a_i = n_{i-1}$; $b_i = p_{i+1}$.

Table 5 – Evaluation for Modify type use cases

	Entity				
Step	Customer	Car	Order	Worker	Work
Command validation	3	3	3	3	3
Fetch data	3	3	3	3	3
Check timestamp	3	3	3	3	3
Check Simple rules	3	3	3	3	3
Check Average rules	3	3	0	3	0
Check Complex rules	0	0	0	0	0
Check data integrity	0	3	3	0	6
Modify data	1	1	1	1	1
Prepare Integration Events	1	1	1	1	1
Return response	1	1	1	1	1
Publish events	1	1	1	1	1
Entities	1	2	2	1	3
Summary (UUSP)	20	24	21	20	25
FUSP (unadjusted)	14.4	16	16	14.4	16
USP (UUSP*0.68)	13.6	16.3	14.3	13.6	17
FUSP (adjusted)	8.2	8.5	9.2	8.2	9.6

Table 6 – Main and alternative scenarios classification (see Table 1, section Main and alternative scenarios)

Complexity	Entities + Steps	p	n	a	b	USP
Very simple	$[1;5]$	1	3.5	–	6	4
Simple	$[6;10]$	6	8.5	3.5	11	6
Average	$[11;15]$	11	13.5	8.5	16	8
Complex	$[16;20]$	16	18.5	13.5	21	12
Very complex	$[21;\infty)$	21	23.5	18.5	–	16

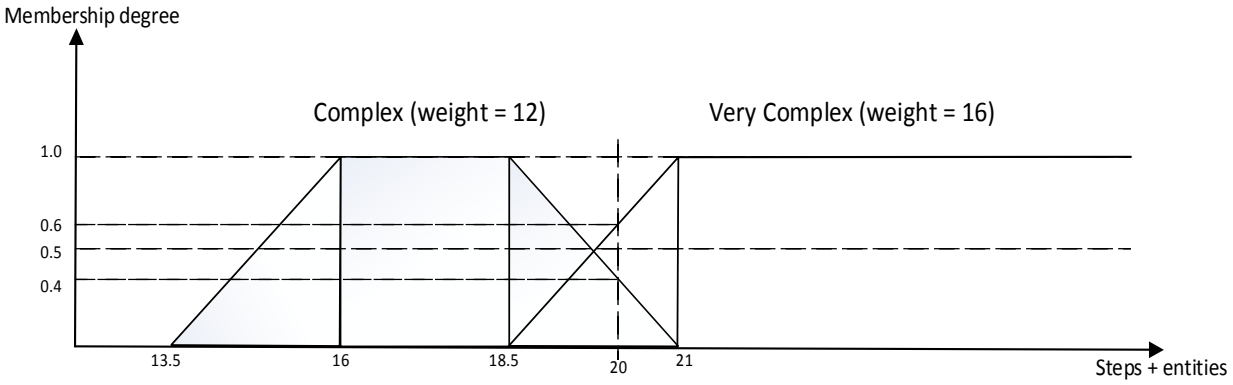

Figure 9 – Example of fuzzification for Customer Modify Use Case

According to FUSP method Modify Customer Use Case (20 steps) belongs to two classes of use cases Complex (weight = 12) and Very Complex (weight = 16) (Table 6). Thus, *FUSP* for this use-case will be evaluated using Formula (10) as follows.

$$FUSP(x) = \frac{b_i - x}{b_i - n_i} \cdot w_i + \frac{x - n_i}{b_i - n_i} \cdot w_{i+1}$$

$$= \frac{21-20}{21-18.5} \cdot 12 + \frac{20-18.5}{21-18.5} \cdot 16 = 14.4.$$

DOI 10.15588/1607-3274-2026-1-14

As we can see from Table 1, in accordance with the core FUSP Method the highest complexity of the use case is 21 steps and more (weight = 16).

The prediction subsequence shown in Table 7 represents 5 iterations. First iteration is devoted to getting initial data which will be used for next iteration prediction. For example, at the initial phase for Modify Use Case type 14.4 FUSP had been realized in 6 manhours, which sets the coefficient (manhours per FUSP) to 0.417. This coefficient is used to predict effort for the next iteration. According to the prediction the next iteration in case of Modify Use Case type would take 6.67 $(16 \cdot 0.417)$ manhours. But in fact, it took 7 manhours, which shifted the max effort coefficient to 0.438 and affected the average FUSP/manhours conversion coefficient: $(0.417 + 0.438)/2 = 0.4275$.

If we use *TAF–EAF* correction the prediction becomes worse (see Table 8). Using the MMRE criteria to estimate the accuracy of prediction we get: $MMRE = 0.0322$ and $MMRE_{(TAF-EAF)} = 0.0831$.

The estimations of the remaining three use case classes are presented in Tables 9–11. The structure of the tables includes three additional columns which indicate basic slot features: capacity, steps and connection to entity (similarly to Modify Use Case Frame description presented in Table 4). Unfortunately, because of the page limits we cannot provide full description of all the use case classes and calculation details. However, the description and calculations were performed similarly to Modify Use Case class.

Resulting comparison of the predictions for proposed and standard methods is shown in Table 12.

Table 7 – Calculations example for Modify Use Cases

FUSP	Total FUSP	Fact (manhours)	Prediction (manhours)		Coefficients (manhours per FUSP) for next iteration effort estimation			
			min	max	avg	min	max	avg
14.4	14.4	6	–	–	–	0.417	0.417	0.417
16	30.4	7 (>max)	6.67	6.67	6.67	0.417	0.438	0.427
16	46.4	6.67	6.67	7.01	6.83	0.417	0.438	0.424
14.4	60.8	5.8(<min)	6	6.31	6.10	0.403	0.438	0.418
16	76.8	6.67	6.44	7.01	6.7	0.403	0.438	0.418

Table 8 – Calculations for Modify Use Cases using TAF–EAF correction

FUSP with TAF, EAF	Total FUSP	Fact (manhours)	Prediction (manhours)		Coefficients (manhours per FUSP) for next iteration effort estimation			
			min	max	avg	min	max	avg
8.2	8.2	6	–	–	–	0.732	0.732	0.732
8.5	16.7	7 (>max)	6.22	6.22	6.22	0.732	0.824	0.778
9.2	25.9	6.67 (<min)	6.73	7.5	7.15	0.725	0.824	0.760
8.2	34.1	5.8(<min)	5.95	6.76	6.23	0.707	0.824	0.747
9.6	43.7	6.67(<min)	6.78	7.91	7.17	0.695	0.824	0.737

Table 9 – Evaluation of Get type use cases

	Entity					Frame		
Step	Customer	Car	Order	Worker	Work	Capacity	Steps	Entity
Command validation	3	3	3	3	3	1	3	0
Fetch data	3	3	3	3	3	1	1	1
Check Simple rules	0	0	0	0	0	0..*	3	0
Check Average rules	0	0	0	0	0	0..*	3	0..*
Return response	1	1	1	1	1	1	1	0
Entities	1	1	1	1	1	–	–	–
UUSP	8	8	8	8	8	–	–	–
FUSP	6	6	6	6	6	–	–	–
USP (UUSP*0.68)	5.44	5.44	5.44	5.44	5.44			
FUSP (adjusted)	4.2	4.2	4.2	4.2	4.2			

DOI 10.15588/1607-3274-2026-1-14

Table 10 – Evaluation of Add type use cases

	Entity					Frame		
Step	Customer	Car	Order	Worker	Work	Capacity	Steps	Entity
Command validation	3	3	3	3	3	1	3	0
Create data	1	1	1	1	1	1	1	1
Check Simple rules	3	3	3	3	3	0..*	3	0
Check Average rules	3	3	0	3	0	0..*	3	0..*
Check Complex rules	0	0	0	0	0	0..*	3	0..*
Save data	1	1	1	1	1	1	1	1..*
Fetch data	1	1	1	1	1	1	1	1
Prepare Integration Events	1	1	1	1	1	1	1	0
Return response	1	1	1	1	1	1	1	0
Publish events	1	1	1	1	1	1	1	0
Entites	1	2	2	1	3	–	–	–
UUSP	16	17	14	16	15	–	–	–
FUSP	12	12	8.8	12	10.4	–	–	–
USP (UUSP*0.68)	10.88	11.56	9.52	10.88	10.2	–	–	–
FUSP (adjusted)	7.9	4.9	6.8	7.9	7.4	–	–	–

Table 11 – Evaluation of Remove type use cases

	Entity					Frame		
Step	Customer	Car	Order	Worker	Work	Capacity	Steps	Entity
Command validation	3	3	3	3	3	1	3	0
Fetch data	3	3	3	3	3	1	1	1
Check Simple rules	0	0	0	0	0	0..*	3	0
Check Average rules	0	0	0	0	0	0..*	3	0..*
Check Complex rules	0	0	0	0	0	0..*	3	0..*
Check data integrity	3	3	3	3	0	0..*	3	0..*
Remove data	1	1	1	1	1	1	1	1
Prepare Integration Events	1	1	1	1	1	1	1	0
Publish events	1	1	1	1	1	1	1	0
Entities	2	2	2	2	1	–	–	–
UUSP	14	14	14	14	10	–	–	–
FUSP	8.8	8.8	8.8	8.8	7.2	–	–	–
USP (UUSP*0.68)	9.52	9.52	9.52	9.52	6.8	–	–	–
FUSP (adjusted)	6.8	6.8	6.8	6.8	4.6	–	–	–

Table 12 – Comparison of Effort/FUSP prediction for proposed and standard methods

Iteration	Proposed method with classification without adjustment		Standard method without classification with adjustment		Fact Effort (manhours)
	FUSP (unadjusted)	Prediction (manhours) Classified	FUSP (adjusted)	Prediction (manhours) Unclassified	
1 (Customer)	41.2	0	27.1	0	17.84
2 (Car)	42.8	18.51	24.4	16.06	18.34
3 (Order)	39.6	17	27	19.04	16.2
4 (Worker)	41.2	17.47	27.1	18.16	16.5
5 (Work)	39.6	16.61	25.8	16.9	16.29

Thus, as a result *MMRE* for the proposal method is 0.0343, and for the standard method – 0.109. The parameter *PRED* for both methods is equal to 1.

6 DISCUSSION

Based on the approaches reviewed, the set of conditions to form rigorous Use Case description rules adapted for software effort estimation needs is developed. The modified Use Case metamodel and the method of use cases classification based on frame-based knowledge representation model are suggested. It was proposed to build individual predictors for each class of use cases using corresponding formulas for effort estimation.

DOI 10.15588/1607-3274-2026-1-14

The experiment is based on RTP derived from real projects of corresponding direction using the MVP methodology.

The initial values for estimation (i.e. FUSP to effort ratio) are acquired during the initial warm-up iteration, i.e. the prediction is driven by the RTP migration iterations and does not rely on statistics from other projects. The result is the collection of functions which are used to predict the effort required for the next iteration (measured in person-hours) for each class of use cases. The coefficient of FUSP person-hours transformation is based on three trends achieved and updated considering the results from previous iterations: the most pessimistic prediction is based on the upper bound, the lower bound predictor plays the role of the optimistic predictor, and the main trend is the meaning among these.

The experiment was conducted for migration among Application service and Use case oriented DDD architectural variations. We take 20 basic use cases related to 5 entities divided into 4 classes (Add, Modify, Get, Remove) and used two variants of prediction: with and without use case classification. We also used filtering described by the formulas (10)–(12): we excluded the old values from the calculation when the deviation of Effort/FUSP ratio was greater than the threshold (in our case it was 0.08). To compare the methods, we used MMRE and PRED criteria.

The results are as follows. Without filtering *MMRE* for the proposal method is 0.0343, and for the standard method – 0.1094. The parameter *PRED* for both methods is equal to 1. In case of filtering the proposal method showed better result: *MMRE* is 0.0309 and for standard method *MMRE* remains the same.

Thus, the obtained results allow us to say that the classification of use cases along with their rigorous description according to provided rules, and modification of the method by separating prediction logic in accordance with the use case classes makes the prediction more accurate and can be effectively used for effort estimation for DDD architectural variations migration.

An experiment was conducted, demonstrating how the proposed method can be applied in practice to describe the use cases, evaluate them, to plan the effort and compare different methods using MMRE and PRED criteria, enabling the project managers to prognose the effort more accurately and developers to determine the development issues. The results obtained can be also reused as initial and historical data when planning similar architectural variations migration in real-world projects.

CONCLUSIONS

The work is focused on providing a theoretical and experimental platform applicable to effort estimation of domain-driven architectural variations migration.

The scientific novelty. FUSP method was adapted for task of gaining greater prediction accuracy of effort estimation for migration among variations of DDD architecture using a methodology based on specifications of requirements.

The set of conditions to form the Use Case description rules adapted for software migration effort estimation needs is developed. The modified Use Case metamodel and the method of use cases classification based on frame-based knowledge representation model are suggested. It was proposed algorithm for building the individual predictors of each class and for corresponding effort estimation. The coefficient of FUSP person-hours transformation is based on three trends achieved and updated considering the results from previous iterations: the most pessimistic prediction is based on the upper bound, the lower bound predictor plays the role of the optimistic predictor, and the main trend is the meaning among these. The coefficients are used to predict the effort in person-hours required for the next iteration for each class of use cases.

The results of experiment, conducted using the test RTP project for this class of software, showed that *MMRE* for the proposal method is 0.0343, and for the standard method – 0.1094. The obtained results evidence that the classification of use cases along with their rigorous description according to provided rules, and modification of the method by separating prediction logic in accordance with the use case classes makes the prediction more accurate and can be effectively used for effort estimation for DDD architectural variations migration.

The practical significance. The experimental part of the article presents the methodology for applying the developed method to describe the use cases, evaluate them, to plan the effort and compare different methods using MMRE and PRED criteria. The results obtained can be also used as initial and historical data when planning similar architectural variations migration in real-world projects.

ACKNOWLEDGEMENTS

We sincerely appreciate DBB Software company [29] for providing their proprietary platform, which served as the foundation for our experiment. This platform offered essential capabilities for our research, ensuring the accuracy and reliability of our experimental results.

DECLARATIONS

Conflict of interest: The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Authors' contributions: Oleksandr Lytvynov: method of use cases classification based on frame-based knowledge representation model; Volodymyr Khandetskyi: the main idea behind improving estimation accuracy; Mykhailo Lytvynov: FUSP method.

Data availability: The data will be made available on reasonable request from authors by email litvinovma0@gmail.com.

Software availability: The manuscript has associated software in a repository

DOI 10.15588/1607-3274-2026-1-14

https://github.com/MykhayloLytvynov/TechnicalStation.ADMCommandBus; https://github.com/MykhayloLytvynov/TechnicalStation.Services.

Use of artificial intelligence tools: The authors confirm that they did not use artificial intelligence technologies in creating the submitted work.

REFERENCES

1. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, Addison Wesley Professional, 2003, 560 p.
2. Vernon V. Implementing Domain-Driven Design. Boston, Addison Wesley, 2013, 656 p.
3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Hong Kong, Pearson Education Asia, 2017, 432 p.
4. Ford N., Parsons R., Kua P., Sadalage P. Building Evolutionary Architectures, 2nd ed. Sebastopol, O'Reilly Media, 2022, 256 p.
5. Sharma S., Kushwaha D. S. Estimation of Software Development Effort from Requirements Based Complexity, *Procedia Technology*, 2012, Vol. 4, pp. 716–722. DOI: 10.1016/j.protcy.2012.05.116
6. Nhung H. L. T. K., Hoc H. T., Hai V. Van A Review of Use Case-Based Development Effort Estimation Methods in the System Development Context, *Software Engineering and Computer Systems : 2019 International Conference, Kuantan, 24–26 June 2019 : proceedings*. Cham, Springer, 2019, pp. 484–499. (Communications in Computer and Information Science, Vol. 1010). DOI: 10.1007/978-3-030-30329-7_44
7. Hoc H. T., Hai V. Van, Nhung H. L. T. K. AdamOptimizer for the Optimisation of Use Case Points Estimation, *Software Engineering and Computer Systems : 2020 International Conference, Kuantan, 24–26 June 2020 : proceedings*. Cham, Springer, 2020, pp. 747–756.
8. Dolado J. J. On the Problem of the Software Cost Function, *Information and Software Technology*, 2001, Vol. 43, № 1, pp. 61–72.
9. Foss T., Stensrud E., Kitchenham B., Myrtveit I. A Simulation Study of the Model Evaluation Criterion MMRE, *IEEE Transactions on Software Engineering*, 2003, Vol. 29, № 11, pp. 985–995.
10. Diev S. Use Cases Modeling and Software Estimation: Applying Use Case Points, *ACM Software Engineering Notes*, 2006, Vol. 31, № 6, pp. 1–5.
11. Galorath D. D., Evans M. W. Software Sizing, Estimation and Risk Management. Boston, Auerbach Publications, 2006, 573 p.
12. Karner G. Resource Estimation for Objectory Projects. Stockholm, Objective Systems SF AB, 1993, 20 p.
13. Braz M. R., Vergilio S. R. Software Effort Estimation Based on Use Cases, *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, 17–21 September 2006 : proceedings*. Los Alamitos, IEEE Computer Society, 2006, pp. 221–228. DOI: 10.1109/COMPSAC.2006.77
14. Braz M., Vergilio S. Using Fuzzy Theory for Effort Estimation of Object-Oriented Software, *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), Boca Raton, 15–17 November 2004 : proceedings*. Los Alamitos, IEEE, 2004, pp. 196–201.
15. Nassif A. B., Capretz L. F., Ho D. Enhancing Use Case Points Estimation Method Using Soft Computing Techniques, *Global Research in Computer Science*, 2010, Vol. 1, № 4, pp. 12–20. DOI: 10.48550/arXiv.1612.01078
16. Iraji M. S., Motameni H. Object Oriented Software Effort Estimate with Adaptive Neuro Fuzzy Use Case Size Point (ANFUSP), *International Journal of Intelligent Systems and Applications*, 2012, Vol. 4, № 6, pp. 14–24. DOI: 10.5815/ijisa.2012.06.02
17. Wen J., Li S., Lin Z., Hu Y., Huang C. Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models, *Information and Software Technology*, 2012, Vol. 54, № 1, pp. 41–59.
18. Qi K., Hira A. Calibrating Use Case Points Using Bayesian Analysis, *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018), Oulu, 11–12 October 2018 : proceedings*. New York, ACM, 2018, Article 22. DOI: 10.1145/3239235.3239247
19. Arthi B., Selvarani A. G. Simplified Software Effort Estimation Using Fuzzy Set Theory, *Australian Journal of Basic and Applied Sciences*, 2015, Vol. 9, № 23, pp. 347–353.
20. Azzeh M., Neagu D., Cowling P. I. Analogy-Based Software Effort Estimation Using Fuzzy Numbers, *Journal of Systems and Software*, 2011, Vol. 84, № 2, pp. 270–284. DOI: 10.1016/j.jss.2010.09.028
21. Nassif A. B., Ho D., Capretz L. F. Regression Model for Software Effort Estimation Based on the Use Case Point Method, *2011 International Conference on Computer and Software Modeling, Singapore, 14–16 October 2011 : proceedings*. Singapore, IACSIT Press, 2011, pp. 117–121.
22. Cockburn A. Writing Effective Use Cases. Boston, Addison-Wesley, 2001, 304 p.
23. Seki Y., Hayashi S., Saeki M. Detecting Bad Smells in Use Case Descriptions, *Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19), Jeju, 23–27 September 2019 : proceedings*. Los Alamitos, IEEE, 2019, pp. 98–108.
24. Seki Y., Hayashi S., Saeki M. Cataloging Bad Smells in Use Case Descriptions and Automating Their Detection, *IEICE Transactions on Information and Systems*, 2022, Vol. 105–D, № 5, P. 849–863.
25. Sinnig D., Khendek F., Chalin P. Partial Order Semantics for Use Case and Task Models, *Formal Aspects of Computing*, 2010, Vol. 23, № 3, pp. 307–332.
26. Simko V., Hauzar D., Hnetynka P., Bures T., Plasil F. Formal Verification of Annotated Textual Use-Cases, *The Computer Journal*, 2015, Vol. 58, № 7, pp. 1495–1529.
27. Durán A., Bernárdez B., Genero M., Piattini M. Empirically Driven Use Case Metamodel Evolution, *UML 2004 : 7th International Conference, Lisbon, 11–15 October 2004 : proceedings*. Berlin, Springer, 2004, pp. 1–11. (Lecture Notes in Computer Science, Vol. 3273).
28. Hombergs T. Get Your Hands Dirty on Clean Architecture 2nd ed. Birmingham, Packt Publishing, 2023, 278 p.
29. DBB Software's Official Company Site [Electronic resource]. Access mode: https://dbbsoftware.com/
30. Ries E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. New York, Crown Business, 2011, 336 p.
31. Floyd C. A Systematic Look at Prototyping, *Approaches to Prototyping*. Berlin, Springer, 1984, pp. 1–18. DOI: 10.1007/978-3-642-69796-8_1

DOI 10.15588/1607-3274-2026-1-14

32. Da Silva A. R. Model-Driven Engineering: A Survey Supported by the Unified Conceptual Model, *Computer Languages, Systems & Structures*, 2015, Vol. 43, pp. 139–155.
33. Milet S. Patterns, Principles, and Practices of Domain-Driven Design. Indianapolis, Wrox, 2015, 800 p.
34. Lytvynov O., Lytvynov M. On Application of the Frame-Based Modeling Language for Information System Development, *System Technologies*, 2023, Vol. 1, № 144, pp. 83–98. DOI: 10.34185/1562-9945-1-144-2023-11
35. Lytvynov O., Khandetskyi V., Lytvynov M. On Use of the Frame-Based Modeling Language for Information System Development, *VIII Vseukrainska naukovo-praktychna konferentsiia "Perspektyvni napriamky suchasnoi elektroniky, informatsiinykh i komp'iuternykh system" (MEICS-2023), Dnipro, 22–24 lystopada 2023 : proceedings.* Dnipro, DNU im. O. Honchara, 2023, pp. 11–12.

Received 03.08.2025.
Accepted 08.01.2026.
Published 27.03.2026.

УДК 614.2+574/578+004.38

ОЦІНКА ЗУСИЛЬ ДЛЯ МІГРАЦІЇ МІЖ АРХІТЕКТУРНИМИ ВАРІАНТАМИ DOMAIN-DRIVEN DESIGN

Литвинов О. А. – канд. техн. наук, доцент, Факультет фізики, електроніки та комп'ютерних систем, Дніпровський національний університет імені Олеся Гончара, просп. м. Дніпро, Україна. ROR: https://ror.org/00qk1f078. ORCID: 0000-0001-7660-1353.

Хандецький В. С. – д-р техн. наук, професор, Завідувач кафедри електронних обчислювальних машин, Дніпровський національний університет імені Олеся Гончара, м. Дніпро, Україна. ROR: https://ror.org/00qk1f078. ORCID: 0000-0002-6386-4637.

Литвинов М. О. – магістр, аспірант, Факультет фізики, електроніки та комп'ютерних систем, Дніпровський національний університет імені Олеся Гончара, м. Дніпро, Україна. ROR: https://ror.org/00qk1f078. ORCID: 0009-0000-9765-1501.

АНОТАЦІЯ

Актуальність. У статті розглядається проблема оцінки трудовитрат при міграції між варіаціями архітектури DDD із використанням методу, що базується на специфікаціях вимог, з метою підвищення передбачуваності процесу міграції програмного забезпечення.

Мета роботи – запропонувати ефективний метод оцінки трудовитрат на основі аналізу Use Case.

Метод. По-перше, запропоновано набір правил для строгої формалізації Use Case, адаптованих під потреби оцінки трудовитрат у розробці програмного забезпечення. По-друге, представлено метамодель модифікованого Use Case і метод класифікації Use Case на основі фреймової моделі подання знань. Такий строгий опис дозволяє точніше оцінювати Use Case за допомогою методу FUSP та створювати окремі предиктори для кожного класу Use Case. По-третє, метод використовує історичні дані з попередніх ітерацій того самого проєкту та ґрунтується на трьох трендах: оптимістичному, песимістичному та середньому.

Результати. Результатом є набір функцій, що використовуються для прогнозування трудовитрат (у людино-годинах), необхідних для наступної ітерації, окремо для кожного класу Use Case.

Висновки. Метод FUSP було адаптовано для підвищення точності прогнозування трудовитрат при міграції між варіаціями архітектури DDD, із застосуванням підходу, заснованого на специфікаціях вимог. Розроблено набір умов для формування правил опису Use Case, адаптованих до задач оцінки трудомісткості міграції програмного забезпечення. Запропоновано метамодель модифікованого Use Case та метод класифікації Use Case на основі фреймової моделі подання знань. Сформульовано алгоритм побудови індивідуальних предикторів для кожного класу Use Case та відповідної оцінки трудовитрат. Коефіцієнт перетворення FUSP у людино-години базується на трьох трендах, що формуються і оновлюються за результатами попередніх ітерацій: найпесимістичніше передбачення визначається верхньою межею, найоптимістичніше – нижньою, а основна оцінка – це середнє між ними. Ці коефіцієнти використовуються для прогнозування трудовитрат у людино-годинах, необхідних для наступної ітерації для кожного класу Use Case. Результати експерименту, проведеного на тестовому проєкті RTP цього класу ПЗ, показали, що середня відносна похибка запропонованого методу становить 0,0343, а стандартного – 0,1094. Отримані результати свідчать про те, що класифікація Use Case разом із їхнім строгим описом за запропонованими правилами, а також модифікація методу шляхом розділення логіки прогнозування відповідно до класів Use Case дозволяє досягти більшої точності та може ефективно використовуватись для оцінки трудовитрат при міграції архітектурних варіацій DDD.

КЛЮЧОВІ СЛОВА: Use Case Point, оцінка трудовитрат програмного забезпечення, Fuzzy Use Case Size Point, Domain-Driven Design.

ЛІТЕРАТУРА

1. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. – Boston : Addison Wesley Professional, 2003. – 560 p.
2. Vernon V. Implementing Domain-Driven Design / V. Vernon. – Boston : Addison Wesley, 2013. – 656 p.
3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. – Hong Kong : Pearson Education Asia, 2017. – 432 p.
4. Building Evolutionary Architectures / [N. Ford, R. Parsons, P. Kua, P. Sadalage]. – 2nd ed. – Sebastopol : O'Reilly Media, 2022. – 256 p.
5. Sharma S. Estimation of Software Development Effort from Requirements Based Complexity / S. Sharma, D. S. Kushwaha // Procedia Technology. – 2012. – Vol. 4. – P. 716–722. DOI: 10.1016/j.protcy.2012.05.116
6. Nhung H. L. T. K. A Review of Use Case-Based Development Effort Estimation Methods in the System Development Context / H. L. T. K. Nhung, H. T. Hoc, V. Van Hai // Soft-

DOI 10.15588/1607-3274-2026-1-14

ware Engineering and Computer Systems : 2019 International Conference, Kuantan, 24–26 June 2019 : proceedings. – Cham : Springer, 2019. – P. 484–499. – (Communications in Computer and Information Science, Vol. 1010). DOI: 10.1007/978-3-030-30329-7_44
7. Hoc H. T. AdamOptimizer for the Optimisation of Use Case Points Estimation / H. T. Hoc, V. Van Hai, H. L. T. K. Nhung // Software Engineering and Computer Systems : 2020 International Conference, Kuantan, 24–26 June 2020 : proceedings. – Cham : Springer, 2020. – P. 747–756.
8. Dolado J. J. On the Problem of the Software Cost Function / J. J. Dolado // Information and Software Technology. – 2001. – Vol. 43, № 1. – P. 61–72.
9. A Simulation Study of the Model Evaluation Criterion MMRE / [T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit] // IEEE Transactions on Software Engineering. – 2003. – Vol. 29, № 11. – P. 985–995.
10. Diev S. Use Cases Modeling and Software Estimation: Applying Use Case Points / S. Diev // ACM Software Engineering Notes. – 2006. – Vol. 31, № 6. – P. 1–5.
11. Galorath D. D. Software Sizing, Estimation and Risk Management / D. D. Galorath, M. W. Evans. – Boston : Auerbach Publications, 2006. – 573 p.
12. Karner G. Resource Estimation for Objectory Projects / G. Karner. – Stockholm : Objective Systems SF AB, 1993. – 20 p.
13. Braz M. R. Software Effort Estimation Based on Use Cases / M. R. Braz, S. R. Vergilio // Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, 17–21 September 2006 : proceedings. – Los Alamitos : IEEE Computer Society, 2006. – P. 221–228. DOI: 10.1109/COMPSAC.2006.77
14. Braz M. Using Fuzzy Theory for Effort Estimation of Object-Oriented Software / M. Braz, S. Vergilio // 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), Boca Raton, 15–17 November 2004 : proceedings. – Los Alamitos : IEEE, 2004. – P. 196–201.
15. Nassif A. B. Enhancing Use Case Points Estimation Method Using Soft Computing Techniques / A. B. Nassif, L. F. Capretz, D. Ho // Global Research in Computer Science. – 2010. – Vol. 1, № 4. – P. 12–20. DOI: 10.48550/arXiv.1612.01078
16. Iraji M. S. Object Oriented Software Effort Estimate with Adaptive Neuro Fuzzy Use Case Size Point (ANFUSP) / M. S. Iraji, H. Motameni // International Journal of Intelligent Systems and Applications. – 2012. – Vol. 4, № 6. – P. 14–24. DOI: 10.5815/ijisa.2012.06.02
17. Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models / [J. Wen, S. Li, Z. Lin et al.] // Information and Software Technology. – 2012. – Vol. 54, № 1. – P. 41–59.
18. Qi K. Calibrating Use Case Points Using Bayesian Analysis / K. Qi, A. Hira // Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018), Oulu, 11–12 October 2018 : proceedings. – New York : ACM, 2018. – Article 22. DOI: 10.1145/3239235.3239247
19. Arthi B. Simplified Software Effort Estimation Using Fuzzy Set Theory / B. Arthi, A. G. Selvarani // Australian Journal of Basic and Applied Sciences. – 2015. – Vol. 9, № 23. – P. 347–353.
20. Azzeh M. Analogy-Based Software Effort Estimation Using Fuzzy Numbers / M. Azzeh, D. Neagu, P. I. Cowling // Journal of Systems and Software. – 2011. – Vol. 84, № 2. – P. 270–284. DOI: 10.1016/j.jss.2010.09.028
21. Nassif A. B. Regression Model for Software Effort Estimation Based on the Use Case Point Method / A. B. Nassif, D. Ho, L. F. Capretz // 2011 International Conference on Computer and Software Modeling, Singapore, 14–16 October 2011 : proceedings. – Singapore : IACSIT Press, 2011. – P. 117–121.
22. Cockburn A. Writing Effective Use Cases / A. Cockburn. – Boston : Addison-Wesley, 2001. – 304 p.
23. Seki Y. Detecting Bad Smells in Use Case Descriptions / Y. Seki, S. Hayashi, M. Saeki // Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19), Jeju, 23–27 September 2019 : proceedings. – Los Alamitos : IEEE, 2019. – P. 98–108.
24. Seki Y. Cataloging Bad Smells in Use Case Descriptions and Automating Their Detection / Y. Seki, S. Hayashi, M. Saeki // IEICE Transactions on Information and Systems. – 2022. – Vol. 105-D, № 5. – P. 849–863.
25. Sinnig D. Partial Order Semantics for Use Case and Task Models / D. Sinnig, F. Khendek, P. Chalin // Formal Aspects of Computing. – 2010. – Vol. 23, № 3. – P. 307–332.
26. Formal Verification of Annotated Textual Use-Cases / [V. Simko, D. Hauzar, P. Hnetynka et al.] // The Computer Journal. – 2015. – Vol. 58, № 7. – P. 1495–1529.
27. Empirically Driven Use Case Metamodel Evolution / [A. Durán, B. Bernárdez, M. Genero, M. Piattini] // UML 2004 : 7th International Conference, Lisbon, 11–15 October 2004 : proceedings. – Berlin : Springer, 2004. – P. 1–11. – (Lecture Notes in Computer Science, Vol. 3273).
28. Hombergs T. Get Your Hands Dirty on Clean Architecture / T. Hombergs. – 2nd ed. – Birmingham : Packt Publishing, 2023. – 278 p.
29. DBB Software's Official Company Site [Electronic resource]. – Access mode: https://dbbsoftware.com/
30. Ries E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses / E. Ries. – New York : Crown Business, 2011. – 336 p.
31. Floyd C. A Systematic Look at Prototyping / C. Floyd // Approaches to Prototyping. – Berlin : Springer, 1984. – P. 1–18. DOI: 10.1007/978-3-642-69796-8_1
32. da Silva A. R. Model-Driven Engineering: A Survey Supported by the Unified Conceptual Model / A. R. da Silva // Computer Languages, Systems & Structures. – 2015. – Vol. 43. – P. 139–155.
33. Milet S. Patterns, Principles, and Practices of Domain-Driven Design / S. Milet. – Indianapolis : Wrox, 2015. – 800 p.
34. Lytvynov O. On Application of the Frame-Based Modeling Language for Information System Development / O. Lytvynov, M. Lytvynov // System Technologies. – 2023. – Vol. 1, № 144. – P. 83–98. DOI: 10.34185/1562-9945-1-144-2023-11
35. Lytvynov O. On Use of the Frame-Based Modeling Language for Information System Development / O. Lytvynov, V. Khandetskyi, M. Lytvynov // VIII Всеукраїнська науково-практична конференція «Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем» (MEICS-2023), Дніпро, 22–24 листопада 2023 : proceedings. – Дніпро : ДНУ ім. О. Гончара, 2023. – P. 11–12.

DOI 10.15588/1607-3274-2026-1-14