**DOI: https://doi.org/10.15276/ict.03.2026.36**
**UDC 004.41**

# Effort estimation for complex business logic migration in legacy systems: a case study of hospital information system modules in Ukraine

**Mykhailo O. Lytvynov**[1)]
ORCID: https://orcid.org/0009-0000-9765-1501; litvinovma0@gmail.com
**Volodymyr V. Gerasimov**[1)]
ORCID: https://orcid.org/0000-0002-1366-715X; gerasimov@dnu.dp.ua. Scopus Author ID: 57191378683
[1)] Oles Honchar Dnipro National University, 72, Nauky Ave. Dnipro, 49000, Ukraine

## ABSTRACT

The object of research is the migration of a legacy hospital information system to a Domain-Driven Design architecture. The problem is accurate effort estimation under changing conditions, since use case-based methods rely on global, project-wide adjustment factors that are too abstract to capture localized architectural, logical, and technological anomalies within individual functional blocks. A modified Use Case Size Points method is proposed, integrating a multidimensional Complexity Amplification Factor across the Infrastructure, Logic, and Technology domains, each scored on calibrated indicator scales. The method was calibrated and validated on real functional blocks of the Unified Clinico-Statistical Classification of Disease subsystems of a Ukrainian surgical clinic. Incorporating the Complexity Amplification Factor increased the accuracy of prediction for query-oriented tasks by approximately seven point six times, and for command-oriented tasks by approximately three point six times. In both cases, the resulting error falls within the range generally considered acceptable for reliable project planning, whereas the baseline method's errors were large enough to cause systematic effort underestimation and budget overruns. This improvement occurs because the Complexity Amplification Factor captures non-standard business logic and hidden infrastructural and technological effort that purely structural size metrics overlook. As a result, project managers obtain a more realistic and trustworthy basis for scheduling, cost control, and risk assessment when planning similar legacy migration projects. The method applies to effort estimation, planning, budgeting, and auditing of legacy migration projects, provided the Complexity Amplification Factor weights and the man-hour coefficient are recalibrated per task class and organizational context.

***Keywords:*** Complexity factor; computational experiments; effort estimation; hospital information system; legacy software migration; optimization; software

***For citation:*** **Lytvynov M. O., Gerasimov V. V., "Effort estimation for complex business logic migration in legacy systems: a case study of hospital information system modules in Ukraine". *Informatics. Culture. Technology*. 2026; Vol. 3 No. 1 (3): 461–481. DOI: https://doi.org/10.15276/ict.03.2026.36**

## INTRODUCTION

**Problem Statement.** Modernizing legacy software remains one of the most challenging problems that engineers face when maintaining information systems. Domain-Driven Design (DDD) and its architectural derivatives are widely recognized as effective industrial patterns for handling domain complexity [1], yet their practical execution within legacy infrastructure migration remains highly unpredictable. This unpredictability is especially acute when migrating complex, domain-specific business functions, such as clinical diagnosis classification, resource planning, and specialized medical analytics, whose structural originality produces complexity anomalies that traditional use case-based effort-estimation frameworks, in particular the standard Use Case Points (UCP) method, fail to capture [2].

As an E-type system that must continuously adapt to a changing environment, HIS software carries an evolutionary pressure that existing estimation methods fail to anticipate. In the Ukrainian healthcare sector, this pressure has intensified sharply since the 2018 launch of national digital-transformation initiatives and the eHealth ecosystem, which forced medical institutions to adapt their legacy infrastructures to cloud-based Medical Information Systems (MIS) [3], [4]. However, Ukrainian hospitals operate under extreme constraints: frequent blackouts, cyberattacks, and network instability severely degrade the Quality of Service of purely cloud-oriented solutions [5]. Consequently, many municipal and specialized clinics continue to rely on locally hosted legacy software to guarantee 24/7 operational resilience, and a full transition to a new certified MIS is regarded as highly risky, since such transitions cause major disruptions to healthcare delivery [6]. Moreover, certified cloud MIS cannot cover institution-specific clinical logic: a critical part of it

relies on custom; domain-specific classifications such as the Unified Clinico-Statistical Classification of Diseases (UCSCD) [7], a frame-based model that lets clinicians encode a structured final diagnosis sufficient to trigger targeted clinical protocols. The obsolescence of the legacy platforms hosting such logic makes migrating these functions into modern service-oriented DDD architectures an urgent necessity, and estimating the effort of that migration the central engineering problem.

**Analysis of Recent Research and Publications.** Among the approaches to software effort estimation, methods based on requirements specifications (use cases) occupy a special place, since they do not require detailed component or algorithmic specifications as input. The base method, Use Case Points (UCP), and its size-oriented variant, Use Case Size Points (USP), are known to suffer from four recurring problems, systematized in [2]: subjective complexity definition within use case models (PRB1); subjectivity of the Technical and Environmental Adjustment Factors, TAF/EAF (PRB2); the absence of a standardized use case classification (PRB3); and the difficulty of converting size into person-hours without historical datasets (PRB4).

Recent modifications of UCP confirm that the adjustment stage remains the weakest link.

The traditional UCP method provides two adjustment factors: Technical Adjustment Factors (TAF) and Environmental Adjustment Factors (EAF).

The primary disadvantage of this method is that the TAF and EAF coefficients remain constant for the entire project, effectively applying the same weight to all use cases. This occurs because Technical and Environmental factors are overly abstract. For instance, 'programming language difficulty' (E8) is a highly specific variable that depends on individual team members and the unique nature of their tasks. Additionally, the pre-defined weights might be significantly underestimated for such tasks. A second critical issue is the lack of standardized criteria for measuring these factors on the [0; 5] scale. The unreliability of these adjustment factors is confirmed by a convergence of empirical evidence.

In [8], the AUCSE method begins with six candidate variables: counts of simple, average, and complex actors and use cases, the latter classified by the number of transactions in their scenario. Then the method applies stepwise regression to select significant predictors: for the dataset used, only the counts of the most complex actors and use cases (Actor 3, Case 3) were retained, and the authors report that the technical and environmental adjustment factors (TCF/ECF) had a small practical impact on accuracy and were excluded from the model entirely. While this yields a substantial accuracy gain over classical UCP and independently corroborates the unreliability of TAF/EAF, the resulting model reduces every use case to a single complexity class derived from its scenario's step count, discarding any step-level content. This design makes the method sensitive to the maturity of the use case model at the moment of estimation because a use case's complexity class depends on how many steps its scenario already documents: a use case that is genuinely complex but still only partially elaborated at an early analysis stage can be misclassified into a lower class, directly distorting the model's inputs. The authors acknowledge this risk and recommend calibrating on stage-matched local data, which mitigates the resulting bias statistically but does not remove its structural cause. The authors further claim that AUCSE scales naturally because both small and large projects are processed identically. However, this presumes that the proportional distribution of actors and use cases across complexity classes remains stable as project size grows – an assumption the paper does not test.

In [9], UCP was integrated with Activity-Based Costing (UCPabc) to allocate development cost across life-cycle activities more precisely than the Adjusted Function Point method (cost deviation 23.52 % vs 33.35 %, on a single case-study web application). The method retains Karner's full, unmodified set of twenty-one technical and environmental complexity factors as its size-estimation basis and does not revisit their reliability, localization, or granularity in any way; Activity-Based Costing is layered entirely on top of the resulting UCP figure to redistribute cost across activities, leaving the underlying TAF/EAF-based adjustment – and its project-wide, non-step-level character – completely untouched. The reported accuracy gain is therefore not evidence against PRB2 at all, but a demonstration that a more granular costing layer can be added without addressing the

adjustment mechanism beneath it; moreover, the result stems from a single project and has not been validated across a broader dataset.

According to the results in [10], the two variants of UCP with and without unadjusted actor weights (UAW) provided similar prediction accuracy; the UCP calculation should be based on steps instead of transactions (a set of activities in use-case-scenarios, which is either performed entirely or not at all, e.g., condition-action). A large-scale cross-validation study found that the twenty-one technical and environmental adjustment factors exert only a minor influence on estimation accuracy, and the number of factors may be reduced to six (four technical, two environmental) without loss of accuracy.

In [11], an empirical study of 110 projects tested whether productivity is proportionally influenced by each of the eight environmental factors; the hypothesis was rejected because only five of the eight factors exhibited the expected relationship. Building on this, the study partitions projects into locally homogeneous subsets by environmental-factor levels and trains ensemble ML models on each subset, showing that locality-based models outperform models trained on the full, heterogeneous dataset – and, notably, that conformance to the productivity hypothesis is not itself a requirement for a factor to yield a useful partition. However, this locality is defined at the level of the project dataset, requires a large, heterogeneous historical corpus (such as ISBSG) to populate each local subset, and offers no mechanism for adjusting an individual use case.

The problem remains open in [12], which reports that Karner's original TCF/ECF systematically deviate from observed project outcomes and proposes the UCPRisk method: project risk-register items are clustered, scored by weight, probability, and loss magnitude, and aggregated into a single average risk coefficient per project. This coefficient yields one project-wide correction factor RS, which is applied to the summary project UCP. Consequently, if one use case requires distributed processing (F2) and another does not, this distinction is invisible to the model: both are already merged into the same UCP total before the single, project-averaged RS is applied. UCPRisk therefore does not resolve the project-wide rigidity of TCF/ECF.

The most recent and methodologically most elaborate attempt in this direction is [13], which applies LASSO regression to automatically select and re-weight technical and environmental factors (yielding a dataset-level correction pair LaTF, LaEF) and further stacks seven statistical and ML models on top of the resulting UCP-based variables. Despite its sophistication, LaTF and LaEF remain scalars calibrated once per dataset and applied uniformly to every project and every use case within it – an even coarser granularity than Karner's original per-project TCF/ECF. Notably, the LASSO-based factor selection independently reproduces the redundancy Ochodek observed in 2011: several of the same technical and environmental factors are consistently discarded. These works have improved how the project-wide multiplier is computed, but none has relocated the adjustment to the individual use case.

In prior work [14], a method resolving PRB1, PRB3, and PRB4 was proposed for evolutionary migration between DDD variations, where, unlike revolutionary migration, no technology change occurs and environmental/technological factors therefore play a comparatively minor role, which is why it was left outside that method's scope. The method is grounded in a strict, frame-based definition of use cases: single-responsibility and rigorous step identification, explicit definition of alternative flows, use case decomposition to eliminate repeated logic, and accounting for the number of domain entities at the use case level. This rule set transforms an informal use case description into a strictly defined model that fully reflects the system's architectural specifics. To resolve PRB3, use cases are further classified into frame-classes (e.g., addition, editing, deletion, and retrieval), each with an ordered set of slots representing step types and their facets (e.g., capacity), enabling both controlled use case description and precise sizing in use case size points. Effort in person-hours (PRB4) was predicted using a coefficient calibrated from the historical average of prior estimates that converts the resulting Fuzzy Use Case Size Points (FUSP) metric into labor effort, achieving a substantially lower error than the classical FUSP baseline (MMRE = 0.0343 vs. 0.1094). However, direct application of this method to revolutionary legacy system business-logic migration is not feasible due to the absence of an effective mechanism for accounting

for negative factors connected to the environment (operational infrastructure, technologies, domain logic).

In work [15], the authors proposed using CAF (complexity amplification factor) and CCF (complexity control factor) factors to resolve the problem PRB4 for evolutionary migration of a data-intensive system from monolith to microservice architecture. However, the method cannot be directly applied to resolve the task of legacy system migration because of a different type of migration, which requires a different structure of CAF components; secondly, because of different classes of use cases (the presented work is focused on standard data management systems with rich business logic instead of high-performance data-intensive systems).

**Actuality of research.** Thus, none of these works, except [15], relocates the adjustment to the individual use case. [10] and [8] discard the technical and environmental factors outright, reducing them via factor analysis or excluding them from a stepwise-regression model built solely on actor and use case counts. In [11], the author leaves the factors themselves untouched and merely clusters projects by their values to build local productivity models. [9] goes further in the opposite direction, retaining Karner's full, unmodified twenty-one-factor set and layering an activity-based costing model on top of it without ever revisiting the factors' reliability. Is [12] and [13] both attempt genuine recalibration, aggregating a project's risk register into a single corrective coefficient or fitting factor weights via LASSO regression and a stacked ensemble, respectively – yet in both cases the result is still one scalar (or scalar pair) computed once per project or per dataset and applied uniformly to every use case within it. Across this entire body of work, the adjustment is variously discarded, clustered on, left untouched, or recalibrated, but never computed below the level of the whole project, and none of it is sensitive to the infrastructural complexity that dominates legacy HIS business logic migration. Is [14] and [15], oriented to evolutionary DDD project migration, left this gap unaddressed.

**The Aim of the Article.** The aim of the research is to develop a method based on the principles described in [14], [15] for estimating the effort of migrating complex, non-standard business logic of legacy hospital information systems to Domain-Driven Design architectures, which replaces the project-wide Technical and Environmental Adjustment Factors (TAF/EAF) with a Complexity Amplification Factor (CAF) linked to each use case, thereby increasing the objectivity of effort estimation and resource planning under unstable infrastructure conditions.

To achieve this foundational objective, the following specific research tasks must be resolved:

1. Formalization of the migration effort estimation method, which implies establishing a mathematical framework extending the Use Case Size Points (USP) approach by introducing and formalizing multi-domain Complexity Amplification Factors (CAF) components to quantify localized architectural uncertainty for legacy system business logic migration.

2. Experimental verification based on conducting estimation experiments utilizing functional blocks from real-world Unified Clinico-Statistical Classification of Diseases (UCSCD) software subsystems in order to validate the predictive accuracy of the developed method against the baseline TAF/EAF-based estimation.

## MAIN PART

**Method for Effort Estimation of Complex Business Logic.** The proposed approach is based on the modified UUSP (hereafter xUUSP) method [16] based on granular, weighted accumulation of structural elements.

The total complexity of a Use Case is defined as:

$$\mu(s_k) = b_k \cdot w_b + p_k \cdot w_p + r_{s,k} \cdot w_{s,r} + r_{a,k} \cdot w_{a,r} + r_{c,k} \cdot w_{c,r} + d_k \cdot w_d , \quad (1)$$

where $w_b$ is the weight of a functional step (in case of presence $b_k = 3.0$ for condition-action pairs, $b_k = 1.0$ otherwise, $b_k = 0$ for empty slots and when $r_{s,k} + r_{a,k} + r_{c,k} > 0$), which is normally equal to 1.

For example, $b_k \cdot w_b$ for the checking object version step, it is equal to 3, because it is connected with alternative handling: checking the condition and generating an exception (this metric remains

from the basic USP method), which gives us $b_k = 3.0$ multiplied by $w_b = 1$, and it brings in result 3; $w_p$ is the informational capacity weight ($p_k$ denotes the number of fields-properties engaged in the $k$-th slot);

$w_{s,r}$ represents the weight of the simple rule, while $r_{s,k}$ indicates the number of simple rules connected to the $k$-th slot. Similarly, $w_{a,r}$ and $r_{a,k}$ represent average rules, $w_{c,r}$ and $r_{c,k}$ correspond to complex ones;

$w_d$ is the weight of dependency ($d_k$ represents the number of dependent entities engaged in the operation of the $k$-th slot).

In general form, this can be defined as

$$\mu(s_k) = \sum_{j}^{m} w_j \mu_j, \tag{2}$$

where $\mu_j$ denotes the $i$-th metric and $w_j$ is the weight of that metric.

$$\mu(UC_i) = \sum_{k=1}^{s} \mu(s_k) + ent_i, \text{ where } \; s_k \in UC_i, \tag{3}$$

where $s$ denotes the number of slots in the use case frame to which the instance belongs; $ent_i$ denotes the number of entities as it is proposed by the base USP approach.

**Effort prediction accuracy.** Traditional software engineering benchmarks rely on the Mean of Magnitude of Relative Error (MMRE) and the Percentage of Prediction within x% (PRED(x)) as the two most common metrics for evaluating effort estimation accuracy [2]. Both parameters are based on the quantity called Magnitude of Relative Error (MRE), which is described by Equation 4. MMRE and PRED(x) are shown in Equation 5 and Equation 6 – respectively.

$$MRE_i = \frac{|\hat{y}_i - y_i|}{y_i}, \tag{4}$$

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n}, \tag{5}$$

$$PRED(x) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1, if MRE_i \le x \\ 0, otherwise \end{cases}, \tag{6}$$

where $\hat{y}_i$ is denotes the predicted value; $y_i$ is known real value; $i$ is iteration; $n$ is number of iterations; $x$ is value is considered to be 0.25. According to [17], an accurate effort prediction model should have an MMRE $\le$ 0.25 (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).

**Effort estimation method.** The baseline step-size estimation formula and the general architecture of the multiplicative amplification factor applied to it are adapted from our companion work on monolith-to-microservice migration [15], which addresses a structurally different domain (evolutionary high-performance data-intensive software migration).

The present study departs from that framework in two aspects: it introduces a dedicated Logic complexity domain absent from the original model, reflecting the code-level (rather than infrastructure-level) nature of legacy business-logic migration, and it replaces the original unweighted CAF components aggregation with an AHP-weighted combination of indicators within each domain.

Based on [15], the uncertainty-aware version $\mu^+(U_i)$, which considers CAF, is defined by Equation (7).

$$\mu^+(U_i) = \sum_{k=1}^{p} \mu^+(s_k) + we_i \cdot ent_i, \tag{7}$$

where $\mu^+(s_k) = \alpha_M(s_k)\mu(s_k)$; $we_i$ is the weight of the $ent_i$ parameter for the $i$-th use case.

Engineering inputs into data entities/structures are already captured by the CAF factors, primarily the Data management complexity factor ($\alpha_D$). Therefore, by default, $we_i = 0$ avoids double counting of entities' contribution, but the $ent_i$ parameter remains manageable in case the CAF does not cover the direct per-entity contribution.

$$\mu^+(s_k) = \left(1 + \sum_j \left(\alpha_j(s_k) - \beta_j(s_k)\right)\right)\mu(s_k), \tag{8}$$

where $\mu(s_k)$ is the unadjusted baseline size of the use case step $s_k$ computed via the xUUSP method; $\alpha_j(s_k)$ represents the complexity-amplifying metric, where $j$ is the index of the technical domain within the defined set; $\beta_j(s_k)$ represents the corresponding complexity-controlling dampening values derived from pre-stabilized architectural components within the matching CAF sub-domain.

The subtractive interplay $\alpha_j(s_k) - \beta_j(s_k)$ ensures that the deployment of pre-existing, mature technical solutions suppresses environment friction, thereby establishing stability within the prediction process.

In the current method, we define $we_j$=1, because we don't have CAF factors to capture this. Secondly, we omit $\beta_j(s_k)$ because the originality of tasks often makes control factors less influential.

Thus, the general formula (7) can be rewritten in a simple form adapted for legacy system migration:

$$\mu^+(U_i) = \sum_{k=1}^{p} \mu^+(s_k) + ent_i, \tag{9}$$

where $\mu^+(s_k) = \alpha_M(s_k)\mu(s_k)$ and $ent_i$ is the number of entities connected to $U_i$; $\alpha_M(s_k)$ is composed of measurable factors.

In this work, we select three domains relevant to legacy system migration:

Infrastructure complexity factor ($\alpha_I$): For example, because we lacked an out-of-the-box ORM for FoxPro, we had to manually implement TransformSlot and TransformFrame. The code shows effort in manual data mapping. Infrastructure includes testing.

Logic complexity factor ($\alpha_L$): For example, some methods are recursive. Migrating recursive logic from a data-centric approach to DDD is a non-trivial operation that increases effort exponentially, not linearly.

Technology complexity factor ($\alpha_T$): For example, using DocumentFormat.OpenXml requires low-level manipulation of the XML DOM, which is much more effort-intensive than a standard UI migration.

The resulting formula can be presented as follows

$$\mu^+(s_k) = (1 + \alpha_I(s_k) + \alpha_L(s_k) + \alpha_T(s_k))\mu(s_k). \tag{10}$$

It is notable that the provided tripartite decomposition corresponds directly to the Cognitive Load Theory (CLT) [18]. Infrastructure Complexity maps to Extraneous Cognitive Load, because environmental obstacles, monitoring, verification, and validation issues are unrelated to the application's core logic, introducing noise from a business logic perspective, forcing engineers to expend working memory on environmental state-reconstruction. Logic Complexity maps to Intrinsic Cognitive Load, because the domain rules define the fundamental internal complexity of the task itself, setting the baseline mental demands required to process the business logic. Technology Complexity maps to Germane Cognitive Load, responsible for schema construction and automation, because framework operations and technological paradigm shifts represent the constructive mental effort required to transform obsolete realizations into modern ones.

To enhance the objectivity and repeatability of the metrics, firstly, we proposed to introduce component-indicators for each factor; secondly, we tried to align the indicators [1-10] scales with contemporary software engineering measurement frameworks.

Inspired by the material presented in [19], we developed specialized task-based and structural rating scales to approximate cognitive strain. The main concept we use is Cognitive Load, which is the mental effort applied to resolve a task [18]. The cognitive workload level can be used to indicate the level of learning and to promote task recommendations based on their cognitive workload [18].

To identify the workload level, researchers (e.g., [20]) use various biometric features such as brain waves, pupil size, etc. However, they admit that the results are affected by the level of experience and can vary across individuals. Instead of relying on physical factors, we propose to use Bloom's Taxonomy of Learning revised for computing and software development. For instance, in [21], Bloom's Revised Taxonomy is adapted to structure computing competencies, mapping technical operations across six distinct cognitive depth levels ranging from remembering to synthesizing and creating new knowledge.

Thus, the infrastructure complexity factor model is defined as follows:

$$\alpha_I^{'}(s_k) = -\theta_I + w_{tc}\nu_{tc,k} + w_{dc}\nu_{dc,k} + w_{mc}\nu_{mc,k}, \tag{11}$$

$$\nu_{tc,k} = \eta\left(\frac{1}{t}\sum_{j=1}^{t} tc_j\right),\ \nu_{dc,k} = \eta\left(\frac{1}{m}\sum_{p=1}^{m} dc_p\right),\ \nu_{mc,k} = \eta\left(\frac{1}{l}\sum_{j=1}^{l} mc_j\right), \tag{12}$$

$$\alpha_I(s_k) = max(0,\ \alpha_I^{'}(s_k)), \tag{13}$$

$$\eta(x) = min\left(10,\ \ x/\max\left(1, med(x)\right)\right), \tag{14}$$

where $tc_j$ is approximate cognitive load (hereafter ACL) of the test; $dc_p$ is ACL of the data access component; $mc_j$ – approximate monitoring complex infrastructure; $-\theta_I$ is a baseline offset ensuring $\alpha_I = 0$ for typical steps (all metrics at project median); $med(x)$ is the median of ordered components, e.g., for $tc_j$. The scale for the components is presented in Table 1.

While cognitive load is the mental effort spent to resolve a task, in our case, Approximate Cognitive Load is defined more precisely as the total mental effort required to synchronize the new system's logic with the legacy environment's constraints. For instance, $tc_j$ measures not the cyclomatic complexity of the test code itself, but the state-reconstruction effort necessary to validate a use case step. High values within the test verification complexity scale formalize the phenomenon of Cognitive Saturation. This state occurs when a single test case forces the developer to validate the state and execution paths of the complex logic. Empirically, this constraint is directly associated with the proliferation of architectural anomalies known as "test smells", specifically *General Fixture* and *Eager Test* [22]. To ensure practical relevance, the descriptions and parameters of the indicators are mapped directly to the specific project migration tasks analyzed in the experimental section of this study.

$\theta_I = 1$ represents the normalized baseline, not anomalous complexity. It means that each class has a number of tests and logs, but it cannot be regarded as anomalous complexity. The expression $\max(0, \alpha_I^{'}(s_k))$ prevents the impact of negative values.

Applying the method that has been used for defining $\alpha_I^{'}$ for infrastructure to the Logic and Technology axes, we got the following formulas:

$$\alpha_L^{'}(s_k) = -\theta_L + w_{cc}\nu_{cc,k} + w_r\nu_{r,k} + w_{cr}\nu_{cr,k}, \tag{15}$$

$$\alpha_L(s_k) = max(0,\ \alpha_L^{'}(s_k)), \tag{16}$$

$$\nu_{cc,k} = \eta\left(\frac{1}{t}\sum_{i=1}^{t} cc_i\right),\ \nu_{r,k} = \eta\left(\frac{1}{m}\sum_{p=1}^{m} r_p\right),\ \nu_{cr,k} = \eta\left(\frac{1}{l}\sum_{j=1}^{l} cf_j\right), \tag{17}$$

where $-\theta_L$ is analogically to $\theta_I$, baseline offset; $cc_i$ is Approximate Cognitive Load (Table 2); $r_p$ is number of recursion calls; $cf_j$ is coupling factor (Table 3).

Cognitive Load indicators define the mental effort required by a developer to comprehend, trace, and migrate a legacy functional block to the target platform. From the perspective of Cognitive Load Theory [18], restructuring logic flows and executing structural modifications significantly increase the intrinsic cognitive load, forcing the engineer to allocate critical working memory resources to maintain an active mental model of the system execution path. Furthermore, an architectural paradigm shift introduces an additional layer of complexity during legacy code modernization, accelerating working memory saturation due to the misalignment between source and target software abstractions.

*Table 1.* **Assessment Scale for Cognitive Complexity of Test ($tc_i$)**

| Score | Complexity Level | Description and Indicators |
|---|---|---|
| Cognitive Complexity of Test ($tc_j$) | | |
| 1-2 | Baseline | Minimal Friction. Standard state verification with clearly defined inputs and outputs. No external environment setup is required |
| 3-5 | Medium | Environmental Friction. Requires specific state preparation (e.g., legacy FoxPro tables, specific SQL dialects, data sets). Demands deep knowledge of legacy data structures [23] |
| 6-8 | High | Testing complex object graphs or moderately nested structures, which requires partial reconstruction of system state to verify a single test case. |
| 9-10 | Extreme | Cognitive Saturation. Testing recursive structures or complex object graphs where a single test forces the verification of the entire hierarchy [22] |
| Cognitive Complexity of Data Access ($dc_p$) | | |
| 1-2 | Baseline | Standard Access. Direct mapping using modern ORM/ADO.NET providers |
| 3-5 | Medium | Environmental Friction. Working with legacy SQL dialects (e.g., FoxPro-specific SQL) or flat tables without relational integrity [23] |
| 6-8 | High | Data access through low-level file operations with predictable, documented behavior |
| 9-10 | Extreme | Structural Entropy. Accessing data through low-level file manipulations, proprietary drivers with side effects |
| Monitoring Complexity ($mc_j$) | | |
| 1-2 | Baseline | High Observability. Integrated logging frameworks provide full stack traces. State can be inspected via standard debugger without side effects |
| 3-5 | Medium | Limited Visibility. Requires manual insertion of logging entries to track data flow through legacy components. Debugging is hindered by proprietary drivers or asynchronous boundaries [24] |
| 6-8 | High | Limited observability, requiring custom diagnostic tools outside standard tooling |
| 9-10 | Extreme | Black Box. System state is opaque. Developer must write custom visualization tools or use high-density debug and logging points to verify complex side effects [24] |

***Source:*** **compiled by the authors**

*Table 2.* **Approximate Cognitive Load (CC)**

| Score | Level | Description | Example (Migration Context) |
|---|---|---|---|
| 1 | Trivial | Direct pass-through of business rules. No change in data structure or logic flow | Simple logic, trivial tasks |
| 2-3 | Linear | Logic remains flat and predictable and can be mapped onto target platform with minimal efforts | Validation of intermediate business rules |
| 4-5 | Functional | Transition requires restructuring the logic flow. Cannot be easily mapped from source platform onto the target one | Implementation of non-trivial algorithmic transformations |
| 6-7 | Structural | Transition requires significant effort related to structural modifications (e.g., shift in data topology) | Implementing recursive mapping for ICD classes tree with up to 4 levels of nesting |
| 8-9 | Complex | Original complex logic which requires significant effort to realize the task. Paradigm shift | Working with hierarchical knowledge structures |
| 10 | Extreme | Architectural Gravity: Unpredictable flow due to extreme complexity of the task or legacy side effects | We don't have such logic in the presented projects |

***Source:*** **compiled by the authors**

*Table 3.* **Coupling factor scale (CF)**

| Score | Level | Description | Example |
|---|---|---|---|
| 1-2 | Isolated | Method works with primitive types or local DTOs. No dependency on legacy state | Math utility or internal string formatter |
| 3-5 | Domain Bound | Dependent on modern domain entities but isolated from the legacy database | ICD classification hierarchy management without direct SQL calls |
| 6-8 | Environment Bound | Tight coupling with legacy database schemas | Frame collection retrieval |
| 9-10 | Global State Bound | Dependency on side effects, shared legacy buffers, or non-deterministic external states | We don't have such logic in the presented projects |

***Source:*** **compiled by the authors**

The Coupling Factor evaluates the degree of structural and informational interdependence between the target legacy functional block and its operational environment. As demonstrated in recent studies [25], in contemporary software engineering Coupling Between Objects (CBO) presents severe limitations, failing to differentiate between a desirable dependency on a clean abstraction and an architectural boundary violation involving a legacy implementation, which seems critical during incremental reverse migration.

Within our framework, an elevated coupling factor signifies that a legacy component cannot be isolated or encapsulated without forcing the modern domain-driven layer to cross layer boundaries and bind directly to historical infrastructure layouts, significantly multiplying the modernization effort.

For the Technology factor, we have the following formula:

$$\alpha_T'(s_k) = -\theta_T + w_{pif}\nu_{pif,k} + w_{mit}\nu_{mit,k} + w_{mis}\nu_{mis,k}, \quad (18)$$

$$\alpha_T(s_k) = max(0,\ \alpha_T'(s_k)), \quad (19)$$

where $-\theta_T{}^{\theta}$ is analogically to $\theta_I$, baseline offset. We use $\theta_I = 1$, which represents the normalized baseline, not anomalous complexity; $\nu_{pif,k}$ is a complex characteristic that defines team unfamiliarity with the product (Table 4); $\nu_{mit,k}$ is inverse of maturity means the level of unfamiliarity with the target technologies (the greater the maturity inverse, the greater the anomaly factor); $\nu_{mis,k}$ is inverse of maturity means the level of unfamiliarity with the source technologies.

*Table 4.* **Team unfamiliarity with the product (PIF, MIT, MIS)**

| Score | Level | Description | Example |
|---|---|---|---|
| 1-2 | Native | Full support by modern frameworks. The team is acquainted with the technology | Standard operations, known tasks |
| 3-5 | Adapters, average complexity of domain | Requires wrappers or adapters. Synthetic solution | FoxPro-specific SQL queries for .NET developers |
| 6-8 | Complex domain or unknown technology | Requires a deep understanding of the domain, source and target technological stacks | Unknown OpenXML technology |
| 9-10 | Incompatible or unknown complex technologies applied to very high complexity tasks | Requires significant effort to find a proper solution to the task. Always connected with research and experiments | We don't have such tasks in the presented projects |

***Source:*** **compiled by the authors**

As demonstrated by Idris et al. [26], traditional static metrics fail to capture the true understandability of architectural transformations because they omit the operational profile of the engineer. By validating the Experience-Weighted Cognitive Complexity Metric (EWCCM), contemporary research proves that cognitive workload must be cross-calibrated with human experience factors [26]. This approach aligns with empirical software engineering research [18], which establishes that a developer's professional experience and background directly modulate the mental effort expended during source code comprehension and task resolution.

Following this principle, our technology complexity factor ($\alpha_T$) integrates team unfamiliarity thresholds to scale the structural divergence, transformation depths, and paradigm shifts from a rigorous, cognitive-informatics perspective.

The specific indicators populating each domain in this study, for instance, recursion depth within the Logic domain or FoxPro-specific coupling within the Infrastructure domain, are project-specific instantiations rather than an exhaustive or universally fixed set. This mirrors the structure of the SEI software development risk taxonomy used as the basis in [12], which similarly organizes a small number of stable top-level risk classes into a much larger, extensible set of elements and attributes, from which researchers and practitioners select the subset relevant to their specific project context [27]. Analogously, the Infrastructure, Logic, and Technology domains defined here are intended as stable top-level categories, while their constituent indicators (e.g., $v_{tc}$, $v_{dc}$, $v_{mc}$) represent one context-specific instantiation; migrations characterized by different architectural patterns, legacy technologies, or team compositions may require different or additional indicators within the same three-domain structure.

**Effort prediction.** Two baseline prediction variants are considered. The average-based (static calibration) variant computes a single coefficient from a completed reference block of use cases (window) belonging to the same task class and then applies this fixed value to the entire subsequent block.

$$\Gamma_{W_{t+1}} = \frac{\sum_{i=1}^{k-1} E(U_i)}{\sum_{i=1}^{k-1} \mu(U_i)}, \left[1; k-1\right] \in W_t, \tag{20}$$

$$\hat{E}_k = \Gamma_{W_{t+1}} \cdot \mu(U_k), \left[k; k + \left|W_{t+1}\right| - 1\right] \in W_{t+1}, \tag{21}$$

where $\hat{E}_k$ is calculated for every use case in the new block $W_{t+1}$ using $\Gamma_{W_{t+1}}$ is the calculated coefficient for the $t$+1 window, with no further adjustment as new actual results become available. The size of the window depends on the size of the project and the specifics of the tasks, risks, etc. It is obvious that the method is inapplicable to the very first window of a migration project, since no completed reference window exists yet.

The cumulative (knowledge-accumulating) variant used in [14] treats estimation as a sequential process. The coefficient $\Gamma_i$ is recomputed before every use case $i$ using only the actual effort and use case size in UUSP for all previous use cases [1; $i$-1], so $\Gamma_i$ is continuously updated as the migration progresses rather than fixed in advance.

$$\Gamma_k = \frac{\sum_{i=1}^{k-1} E(U_i)}{\sum_{i=1}^{k-1} \mu(U_i)}, \tag{22}$$

$$\hat{E}_k = \Gamma_k \cdot \mu(U_k). \tag{23}$$

The former assumes the calibration block is representative and stable, while the latter adapts step-by-step but is inherently more sensitive to abrupt complexity changes late in the sequence, since it has not yet “seen” them.

**Project selection and methodology of experiment.** Considering the specifics of Ukrainian hospitals, including public and private clinics, we can make the following conclusion.

This study follows a case study research design [28], in which the migration effort estimation process constitutes the case, and each functional block (nested use case) of the Unified Clinico-Statistical Classification of Disease subsystems serves as an embedded unit of analysis, situated in the real-life context of an ongoing hospital information system migration in a Ukrainian surgical clinic.

– The Big Bang strategy of migration cannot be applied in most cases, because it requires rigorous preparation and testing before its overall implementation within the clinic. If clinics encompass multiple departments and services that rely on information systems, immediate migration seems impossible.

– Forward migration is not applicable because this approach can only be applied to software where the Data-access layer is decoupled from the upper layer by the interfaces. In most cases, Ukrainian hospital systems don't meet this condition.

– The main drawback of the Butterfly Migration methodology [29], in our opinion, is that the system cannot be used by the end-users until the whole migration process is completed. The switch of end-users to the Target System is not a continuous process, and additional efforts are needed to train the personnel.

Based on these considerations, the most applicable way is to use an incremental, reverse-engineering-informed migration strategy with service-oriented legacy logic encapsulation [30]. This approach establishes a reverse migration gateway, where contemporary domain-driven components are forced to consume and synchronize state with historical, unabstracted storage layouts (such as FoxPro table spaces). This architectural decision explains the direct tight coupling with legacy data constraints and justifies the necessity of analyzing specialized cognitive complexity overheads within the infrastructure domain.

In this part, we provide the results of migrations based on the approach described for the Hospital Information System of Garvis Clinic [31].

The critical and fundamental part of the business that gives the Clinic a competitive advantage is using custom classifications such as the Unified Clinico-Statistical Classification of Disease, using original methods and approaches for managing and optimizing the resources, and developing clinical standards (especially surgical standards, because the main activity of the clinic is surgery). The classifications allow us to define protocols-standards of treatment and surgery more precisely, to analyze the groups of patients, to detect the most effective ways of treatment, and to optimize the resources.

The Unified Clinico-Statistical Classification of Disease (UCSCD) was developed by a group of Ukrainian scientists under the leadership of prof. Bereznitsky Y.S., and it was rather compact and easy to use in comparison with its foreign analogues [7]. UCSCD is based on the standard International Statistical Classification of Diseases and Related Health Problems ($10^{th}$ Revision) [32] and can be thought of as its compact and flexible extended version based on a frame-based knowledge representation model, which enables the definition of a final clinical diagnosis of a patient. A final diagnosis in this context is a formal, compact description of a disorder that is sufficient to initiate treatment.

According to the UCSCD model, each frame used to represent a set of clinical diagnoses is connected to a definite ICD-10 code. As a rule, each frame is composed of several nonterminal slots, each of which is connected to a clinical characteristic (such as Location, Complexity, Phase, etc.). An example is shown in Fig.1. The structure of the slot is represented as a composition of facets, which can be divided into two categories: semantical and syntactical. Semantical facets are as follows: capacity of the slot allowed describing how many values of the same characteristic can be used simultaneously within the clinical diagnosis description; type of characteristic (axis) allowed describing the class of characteristics. While semantical facets are used to build a semantically correct representation allowing statistical and analytical processing of clinical diagnosis information, syntactical facets are used for building a syntactically correct and comprehensible textual representation of a clinical diagnosis (Fig.2).

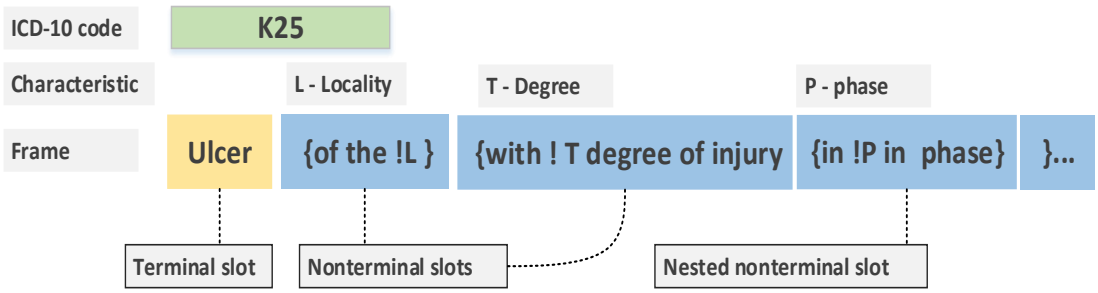


*Fig. 1.* **Unified Clinico-Statistical Classification of Disease frame structure**
***Source:*** **prepared by the authors [7]**

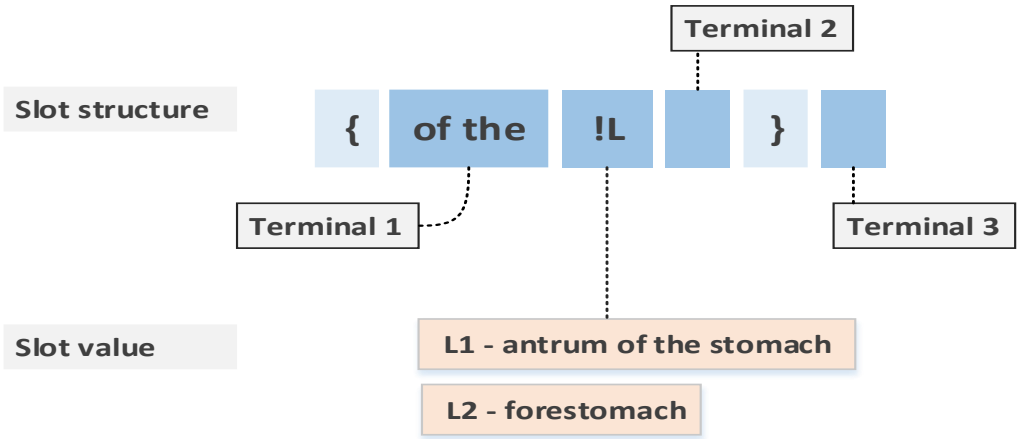


*Fig. 2.* **Unified Clinico-Statistical Classification of Disease slot structure**
***Source*: prepared by the authors [7]**

Thus, according to the frame structure mentioned above, the following code of clinical diagnosis K25 L2T1P1 can be transformed into “Ulcer of the forestomach with moderate degree of injury in exacerbation phase”.

A primary challenge with the legacy software was the difficulty in managing this classification (e.g., updating and printing). This specific task was selected as a pilot project to define the technological stack, architecture, and migration methods for the entire system, as well as to validate the effort estimation model.

The migration methodology adopted in this study includes a preparatory zero phase, conducted before the main migration effort. Its purpose is to resolve foundational infrastructure and technology questions: building a reusable platform baseline and settling integration constraints with the legacy environment (e.g., FoxPro-specific access) so that the team is not forced to address such foundational issues reactively while simultaneously handling complex business-logic migration.

**Effort estimation experiment.** We initiated the experiment with the UCSCD printing module. This task was chosen because the legacy application required an unacceptable amount of time, often tens of minutes, to generate a Word document and could not function as a remote service. From a software engineering perspective, the core problem involved transforming the legacy model into a more flexible object model based on the Composite pattern (details can be found in [7], [33]). The application was developed in 2 weeks. A significant amount of time was spent on tuning the DocumentFormat. Openxml technology, which was used for printing the prepared part of the classification.

The complex use case for printing a classification is formalized through the following sequential steps.

1. Input Range Specification. The analytics expert inputs the specific segment of the classification to be printed in a format such as 'C11-C45; C24;' using the ICD-10 classification codes, where a hyphen denotes a continuous range of diagnoses.

2. SQL Condition Generation. The system parses the input string and translates it into SQL conditions combined by the OR operator, converting hyphenated ranges into SQL.

3. Legacy Frame Retrieval. The system fetches domain frames from the legacy frame database table using the constructed SQL condition. Notably, some legacy frames may lack associated slots.

4. Legacy Slot Retrieval. The system retrieves individual attribute slots from the legacy slot table using the same generated SQL condition.

5. Object Graph Synthesis. The system dynamically maps and binds the slots to their aggregate frames, ordering them based on the presence of the slot definition within the frame description.

6. Hierarchical Mapping. The system attaches the synthesized frame entities to the respective ICD classes within the structural taxonomy.

7. Document Generation. The system translates the resulting hierarchical ICD classes and domain models into an external Microsoft Word document.

As can be observed, this specific use case cannot be directly mapped to a standard class of use case frames proposed in [14]. The operational steps are heavily influenced by unexpected environmental, architectural, and technological factors, causing the actual implementation effort to diverge from standard baseline estimations. This exceptional behavior necessitates a detailed classification of the steps by their complexity amplification levels, as presented in Table 5.

*Table 5*. **The classification of the Print UCSCD steps by CAF levels**

| No. | Use Case Step | Complexity Amplifier Factor |
|---|---|---|
| 1 | Input range (C11-C45) | Low: Involves routine string parsing and basic input query validation |
| 2 | Translate to SQL | Low: Relies on Regular Expressions and standard string splitting algorithms to construct SQL queries |
| 3-4 | Get Frames & Slots | Medium: Complicated by the need to map semi-structured frame-slot models and handle legacy Visual FoxPro-specific SQL dialect constraints |
| 5 | Connect Slots to Frames | High: Effort diverges due to the overhead of building the aggregate object graph. Requires gathering and ordering relational slots based on unstructured textual frame descriptions |
| 6 | Get ICD tree and attach Frames to Icd diagnoses | High: Transforming a flat relational table structure into a deeply hierarchical data structure using the Composite architectural design pattern. The mapping logic requires handling two interdependent recursions |
| 7 | Translate to Word | Extreme: Requires mapping of the hierarchical ICD structure containing UCSCD clinical diagnosis frames into low-level OpenXML considering specifics of the technology |

***Source:*** **compiled by the authors**

The results of the use case effort estimation using the baseline method $\mu(s_k)$ and its modified version $\mu^+(s_k)$ (formulas (7) and (10)) are presented in Table 6, where we organize steps in blocks by the specifics of the task, representing these blocks as nested use cases. This decomposition is necessary because actual effort is incurred and measured at the granularity of steps, not at the level of an undecomposed parent use case.

When calculating the baseline estimation $\mu(s_k)$, the specific structural weights are assigned as $w_b = w_p = 1$, $w_{s,r} = 3$. It should be noted that the operational weights $w_d$, $w_{a,r}$, $w_{c,r}$ are omitted for this specific migration task, as the document printing process does not involve data integrity validation routines.

The weights of the components ($w_{tc}$, $w_{dc}$, $w_{mc}$ for $\alpha_I$, $w_{cc}$, $w_{cr}$, $w_r$ for $\alpha_L$, and $w_{pif}$, $w_{mit}$, $w_{mis}$ for $\alpha_T$) were elicited through pairwise comparison in the style of the Analytic Hierarchy Process (AHP) [34], informed by the zero-phase pilot conducted on the ICD database.

*Table 6.* **Results of the use case effort estimation using the baseline method $\mu(s_k)$ and its modified version $\mu^+(s_k)$**

| Nested use case | μ | $\mu^+$ | Complexity amplifier factor | | | | Approx. Effort (m-h) | $\frac{E_j}{\mu_j}$ | $\frac{E_j}{\mu^+_j}$ |
|---|---|---|---|---|---|---|---|---|---|
| | | | $\alpha_M$ | $\alpha_I$ | $\alpha_L$ | $\alpha_T$ | | | |
| 1-2 | 14 | 14 | 1.0 | - | - | - | 2 | 0.14 | 0.14 |
| 3-4 | 15 | 42 | 2.8 | 1 | - | 0.8 | 4 | 0.26 | 0.1 |
| 5 | 25 | 87.5 | 3.50 | 1 | 1.5 | - | 8 | 0.32 | 0.09 |
| 6 | 25 | 108.75 | 4.35 | 1.25 | 1.9 | 0.2 | 10 | 0.4 | 0.09 |
| 7 | 40 | 398 | 9.95 | 1.25 | 2.5 | 5.2 | 46 | 1 | 0.12 |
| Total | 119 | 650.25 | Avg($\alpha_M$)=4.32 | | | | 70 | 0.58 | 0.11 |

***Source:*** **compiled by the authors**

For defining $\alpha_I$ coefficients, calibration depends on the specific project. In our case, we use $w_{tc}$ = 0.25, $w_{dc}$ = 0.5, $w_{mc}$ = 0.25 based on experiments conducted at the zero phase of migration devoted to experiments and infrastructure development, which showed that the most complex tasks were linked to legacy database adaptation. Database components took approximately two times more than other tasks because of legacy database structure specifics (FoxPro-specific SQL, database structure). Thus, $w_{dc}$ was adjusted. Approximate values are shown in Table 6 ($\alpha_I$ column). The formula captures non-uniform complexity distributions.

To illustrate the application of the infrastructure factor $\alpha_I$ = 1 calculated in accordance with formula (13), we examine Steps 3-4 (Data Extraction and Selection). Based on our empirical observations, the infrastructure complexity (based on the scale presented in Table 1) for these steps is calculated as follows.

$\alpha_I'(s_{3-4})$ = -1 + 0.25×1 + 0.5×3 + 0.25×1 = -1 + 2 = 1

$\nu_{tc,3-4} = 1$ – The test logic (GetTest method) relies on raw semicolon-delimited string queries such as "M16; M17; M20; M232; M23.5; S72.0;…"

$\nu_{dc,3-4} = 3$ – While the underlying database is FoxPro, this specific step involves a standard collection fetch without complex conditions, but using the FoxPro SQL dialect.

$\nu_{mc,3-4} = 1$ – The visibility remains at Baseline (1) as the developer can easily perform a control against the console log.

The weights of the Logic-domain components ($w_r$, $w_{cc}$, $w_{cr}$) were likewise elicited through pairwise comparison in the style of AHP (Analytic Hierarchy Process). Recursion was judged the dominant source of logic complexity ($w_r$ = 0.5): the number of recursive invocations directly governs the effort required to correctly construct and test recursive logic, such as the Composite-pattern-based traversal used to migrate the hierarchical ICD classification in this study – recursive structures of this kind are inherently harder to reason about and validate than linear control flow, and migrating them from a data-centric approach to DDD increases effort exponentially rather than linearly. Domain-comprehension complexity was assigned an intermediate weight ($w_{cc}$ = 0.3), reflecting the effort required to correctly interpret domain-specific semantics (e.g., diagnosis-classification rules); this weight is explicitly project-specific and is expected to require re-tuning for domains with a different semantic density. Coupling with legacy structures was assigned the lowest weight ($w_{cr}$ = 0.2): although structural ambiguity in the legacy database schema does increase logic complexity, it does so largely through the recursive traversal it necessitates, meaning its effect is already partly captured by $w_r$, and assigning it a comparatively smaller independent weight allows us to avoid double-counting.

The weights of the Technology-domain components ($w_{pif}$, $w_{mit}$, $w_{mis}$) were likewise elicited through pairwise comparison in the style of AHP. All three components are defined as inverse-maturity measures: the lower the team's familiarity with a given technology, the higher the corresponding factor. The target-technology maturity inverse ($w_{mit}$ = 0.4) was assigned the highest

weight, since migrating to the new platform required the team to work with unfamiliar frameworks and tooling, and greater unfamiliarity with the target technology directly increases anomalous effort. The source-technology maturity inverse ($w_{mis}$ = 0.3) received a lower weight, reflecting the team's substantial prior familiarity with the legacy source platform. The product-unfamiliarity factor ($w_{pif}$ = 0.3) was assigned the same weight as $w_{mis}$, since the team's prior familiarity with the product itself was comparably strong. The individual component values and the resulting aggregated CAF metrics for each step are provided in Table 7.

*Table 7*. **The individual component values and the resulting aggregated CAF metrics for each step**

| Nested use case | Complexity amplification factor | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\alpha_M$ | $\alpha_I$ | | | $\alpha_L$ | | | $\alpha_T$ | | |
| | | *tc* | *dc* | *mc* | *cc* | *r* | *cr* | *pif* | *mit* | *mis* |
| 1-2 | 1.0 | 1 | - | - | - | - | - | - | - | - |
| 3-4 | 2.8 | 1 | 3 | 1 | - | - | - | 2 | - | 4 |
| 5 | 3.50 | 1 | 3 | 1 | 5 | - | 5 | - | - | - |
| 6 | 4.35 | 2 | 3 | 1 | 5 | 2 | 2 | 3 | 0 | 1 |
| 7 | 9.95 | 2 | 3 | 1 | 6 | 1 | 6 | 7 | 8 | 3 |

***Source:*** **compiled by the authors**

The individual component values and the resulting aggregated CAF metrics for each step are provided in Table 7. The prediction results for both methods (CAF-enhanced method and the standard one) are shown in Table 8. Here we use $\Gamma_{W_{t+1}}$ = 0.11 and $\Gamma_{W_{t+1}}$ = 0.58 as if it was calculated for the same block of previous use cases, which doubles the structure of the current one; other variants of window selection, as well as the materials, are available at [33].

*Table 8*. **The prediction results for both methods (CAF-enhanced method and the standard one)**

| Nested use case | $E_j$ | $\mu^+$ | Predicted $\hat{E}_j^+$ ($\Gamma_{W_{t+1}}$=0.11) | Relative Error (*MRE*$^+$) | $\hat{E}_j^+$ using $\Gamma_i$ | *MRE*$^+$ | Predicted $\hat{E}_j$ ($\Gamma_{W_{t+1}}$=0.58) | Relative Error (*MRE*) | $\hat{E}_j$ using $\Gamma_i$ | *MRE* |
|---|---|---|---|---|---|---|---|---|---|---|
| 1-2 | 2 | 14 | 1.5 | 0.25 | - | - | 8.12 | 3.06 | - | - |
| 3-4 | 4 | 42 | 4.5 | 0.12 | 6 | 0.5 | 8.7 | 1.18 | 2.14 | 0.46 |
| 5 | 8 | 87.5 | 9.4 | 0.18 | 9.38 | 0.17 | 14.5 | 0.81 | 5.17 | 0.35 |
| 6 | 10 | 108.75 | 11.7 | 0.17 | 10.61 | 0.06 | 14.5 | 0.45 | 6.48 | 0.35 |
| 7 | 46 | 398 | 42.7 | 0.07 | 37.87 | 0.18 | 23.2 | 0.5 | 12.15 | 0.74 |
| | | | h = 69.8<br>*MMRE*$^+$ = 0.158 | | *MMRE*$^+$ = 0.227 | | h = 69.02<br>*MMRE* = 1.199 | | *MMRE* = 0.476 | |

***Source:*** **compiled by the authors**

Based on the results in Table 8, CAF reduces Mean Magnitude of Relative Error (MMRE) from 119.9 % to 15.8 % with the flat coefficient, and from 47.6 % to 22.7 % with the cumulative $\Gamma_i$ – the benefit of CAF holds regardless of the estimation strategy. If we use the first three use cases (1-2, 3-4, 5) to calibrate the coefficient, trying to predict the effort for 6 and 7, we get the following: for $\mu^+$, $\hat{E}_6$=10.61 and $\hat{E}_7$=38.83, yielding *MMRE*$^+$=10.9 %; for μ, $\hat{E}_6$=6.48 and $\hat{E}_7$=10.37, yielding *MMRE*=56.3 %. In terms of PRED (0.25), the CAF-enhanced model reaches 100% with the flat coefficient and 75% with the cumulative $\Gamma_i$, compared to 0% for the baseline model under both strategies, meeting the acceptability threshold from [17] that the baseline fails to reach.

The next project was the UCSCD editor [7]. The main challenge for the migrated version was using a textual editor conception as the core idea of the project instead of a widespread semantic tree-based variation. The new version of UCSCD is more robust to noise of modifications, such as adding new functions and the ability to switch among different database providers and schemes, which significantly improves system maintainability. The switching is based on using Repository and Abstract Factory variation [35] patterns. Each set of repositories working with one database provider is placed in a different assembly. The main use cases are shown in Table 9.

*Table 9.* **The main use cases of the UCSCD editor project**

| Use case | Abbreviation | xUUSP | Anomaly |
|---|---|---|---|
| GetIcdClassification | GIC | 115 | Middle |
| RegisterIcdClass | RIC | 42 | Middle |
| RegisterIcdDiagnosis | RID | 13 | Middle |
| ChangeIcdClassName | CICN | 15 | Low |
| ChangeIcdDiagnosisName | CIDN | 7 | Low |
| GetAxisCollection | GAC | 16 | Low |
| GetFrame | GF | 68 | High |
| SaveFrame | SF | 25 | High |
| RemoveFrame | RF | 5 | Low |

***Source:*** **compiled by the authors**

The functions with CAF > 0 are as follows: GetIcdClassification (GIC), RegisterIcdClass (RIC), RegisterIcdDiagnosis (RID), GetFrame (GF), SaveFrame (SF). The results are shown in Table 10. The individual component values and the resulting aggregated CAF metrics for each use case are provided in [33].

*Table 10.* **The prediction results for the functions with CAF > 0**

| Steps Block | $\mu(s_k)$ | $\mu^+(s_k)$ | CAF | | | | Appr. Effort (m-h) | $\frac{E_j}{\mu_j}$ | $\frac{E_j}{\mu^+_j}$ |
|---|---|---|---|---|---|---|---|---|---|
| | | | $\alpha_M$ | $\alpha_I$ | $\alpha_L$ | $\alpha_T$ | | | |
| GIC | 115 | 184 | 1.6 | - | 0.6 | - | 8 | 0.07 | 0.044 |
| RIC | 42 | 212.1 | 5.05 | 1.25 | 1.2 | 1.6 | 8 | 0.19 | 0.038 |
| RID | 13 | 51.35 | 3.95 | 0.75 | 0.9 | 1.3 | 2 | 0.154 | 0.039 |
| GF | 68 | 255 | 3.75 | 1.25 | 1.5 | 0 | 12 | 0.176 | 0.047 |
| SF | 25 | 158.75 | 6.35 | 1.25 | 2.1 | 2.0 | 10 | 0.400 | 0.063 |
| Total | 263 | 861 | | | | | 40 | 0.198 | 0.046 |

***Source:*** **compiled by the authors**

Using the calibrated average coefficient $k$ = 0.046 (effort $E_j$ per weighted $\mu^+(UC_j)$), we calculate the prediction accuracy for each original use case. The results are shown in Table 11.

*Table 11.* **Prediction accuracy for use cases (Table 10)**

| Nested use case | $E_j$ | $\hat{E}_j^+$ ($\Gamma_{W_{t+1}}$=0.046) | Relative Error ($MRE^+$) | $\hat{E}_j^+$ using $\Gamma_i$ | $MRE^+$ | $\hat{E}_j$ ($\Gamma_{W_{t+1}}$=0.2) | Relative Error ($MRE$) | $\hat{E}_j$ using $\Gamma_i$ | $MRE$ |
|---|---|---|---|---|---|---|---|---|---|
| GIC | 8 | 8.5 | 0.063 | - | - | 23 | 1.88 | - | - |
| RIC | 8 | 9.8 | 0.225 | 9.22 | 0.15 | 8.4 | 0.05 | 2.92 | 0.64 |
| RID | 2 | 2.4 | 0.2 | 2.07 | 0.04 | 2.6 | 0.3 | 1.32 | 0.34 |
| GF | 12 | 11.7 | 0.025 | 10.26 | 0.15 | 13.6 | 0.13 | 7.2 | 0.4 |
| SF | 10 | 7.3 | 0.27 | 6.78 | 0.32 | 5 | 0.50 | 3.15 | 0.69 |
| | | h = 39.7<br>$MMRE^+$ = 0.157 (15.7 %) | | $MMRE^+$ = 0.164 | | h = 52.6<br>$MMRE$ = 0.572 | | $MMRE$ = 0.514 | |

***Source:*** **compiled by the authors**

Based on the results in Table 11, CAF reduces MMRE from 57.2 % to 15.7 % with the flat coefficient, and from 51.4 % to 16.4 % with the cumulative $\Gamma_i$ – again confirming that the CAF benefit holds regardless of the estimation strategy. Unlike Case 1 (print classification use case), the cumulative and flat variants perform almost identically here (16.4% vs 15.7%). If we use the first three use cases (GIC, RIC, RID) to calibrate the coefficient, trying to predict the effort for GF and SF, we get the following: for $\mu^+$, $\hat{E}_{GF}$=10.26 and $\hat{E}_{SF}$=6.39, yielding $MMRE^+$=25.3 %; for $\mu$, $\hat{E}_{GF}$=7.20 and $\hat{E}_{SF}$=2.65, yielding *MMRE*=56.8 %. In terms of PRED (0.25), the CAF-enhanced model reaches 80% with the flat coefficient and 75 % with the cumulative $\Gamma_i$, compared to 40% and 0%, respectively, for the baseline model, again meeting the acceptability threshold from [17] that the baseline fails to reach.

Key scientific and practical insights from the results of the experiments include:

– Task class-Specific Calibration (The Context Boundary). The research highlights a critical constraint: coefficients are not interchangeable across different classes of functional tasks. The discrepancy between the Query-oriented model ($k$ = 0.11, Table 6) and the Write-oriented model ($k$ = 0.046, Table 10) of the UCSCD classification reflects the inherent structural differences between data presentation and data persistence logic. Using a “reporting” coefficient to estimate "data entry" forms would lead to an overestimation of resources, while the reverse would result in a critical project deficit. It is worth noting that in both cases, we work only with original tasks related to complex business logic rules with a high value of CAF, excluding simple ones.

– Prediction Accuracy. By integrating multi-domain Complexity Amplification Factors (CAF), the developed model achieved a highly stable MMRE of 15.8 % (against 120% in the case without CAF introduction) for query-oriented tasks and 15.7% (against 57.2%) for complex operation-oriented tasks. The empirical results confirm that accounting for infrastructure, logic, and technology domains significantly reduces estimation errors compared to baseline functional metrics, validating that structural sizing alone cannot yield high-fidelity estimates in legacy migration environments without proper cognitive complexity amplification.

– Estimation Strategy Choice. A flat, pre-calibrated coefficient outperforms a cumulative, self-updating one when the sequence contains a late complexity spike (15.8 % vs 22.7 % MMRE), but the two converge under gradual complexity growth (15.7 % vs 16.4 %), making the cumulative variant a viable fallback when no prior calibration window exists.

The proposed method provides a robust instrument for the audit and evaluation of legacy system migration, integrating localized architectural uncertainty analysis.

## CONCLUSIONS

This paper introduces an adjustment framework designed to refine functional software sizing and effort estimation within complex, non-standard application domains. The specific scientific and practical conclusions, aligned with the stated research tasks, are as follows.

Regarding the formalization of the migration effort estimation method, this work introduces a modified Use Case Size Points (USP) approach, which quantifies Complexity Amplification Factors (CAF) across Infrastructure, Logic, and Technology domains. The proposed three-domain CAF structure builds on the amplification-factor architecture introduced in our companion work on infrastructure-oriented migration [15], extended here with a dedicated Logic domain and AHP-derived indicator weighting to address the code-centric character of legacy business-logic migration. The application of this three-domain framework enables the identification of hidden implementation dependencies that traditional size-based metrics and TAF/EAF-oriented methods fail to capture.

Experimental verification on real functional blocks of the Unified Clinico-Statistical Classification of Diseases (UCSCD) software subsystems confirmed the accuracy and stability of the developed method: the Mean Magnitude of Relative Error (MMRE) was reduced to 15.8 % (against 120 % in the case without CAF introduction) for query-oriented tasks and 15.7 % (against 57.2 %) for operation-oriented tasks, which is well within acceptable software engineering limits.

The study established stable baseline calibration coefficients, specifically $k$=0.11 for UCSCD printing tasks and $k$=0.046 for complex UCSCD management tasks, confirming that the two task classes require distinct calibration. A comparison between a flat, pre-calibrated coefficient and a cumulative, self-updating one further showed that the former remains preferable when a representative prior calibration window exists, while the latter offers a viable fallback when no such history is available.

These calibrated coefficients and low error rates demonstrate that the developed mathematical framework can be applied for life-cycle planning, budgeting, and automated auditing of other high-uncertainty functions, transforming legacy business-logic migration estimation from an intuitive, high-risk activity into a manageable engineering process.

**Conflict of interests:** The authors declare that they do not have a conflict of interests, in particular financial, personal, copyright or any of a different nature, which could have influenced the research, as well as the results published in this article

**Financing:** The research was partially funded by the Ministry of Foreign Affairs of the Czech Republic within the framework of the project 25-PKVV-UM-004 "Development of Education, Research, and International Cooperation at Oles Honchar Dnipro National University (DNU)", implemented by Charles University and Oles Honchar Dnipro National University

**Accessibility data:** Data model implementations, generalized results, experiment parameters, and formed graphic materials are available in the open data repository Zenodo: https://doi.org/10.5281/zenodo.21536777

**Using artificial intelligence tools:** During preparation of the manuscript, the authors used the ChatGPT 5.5 tool of artificial intelligence (AI) for grammatical and stylistic text checks, translation assistance, and identification of sources for the literature review (including DOI and bibliographic data retrieval). All fragments processed by it were checked and edited by the authors. The reliability and relevance of the suggested sources were verified via ISBN Search, Crossref, and Scopus Sources. AI was not used to generate text, formulas, or tables, formulate scientific provisions, to obtain results, or to draw conclusions. The scientific provisions, results, and conclusions are the authors' own intellectual contribution

## REFERENCES

1. Lytvynov, O. & Hruzin, D. “On the migration of domain-driven design to CQRS with Event sourcing software architecture”. *Information Technology Computer Science Software Engineering and Cyber Security*. 2024; 1 (1): 50–60. DOI: https://doi.org/10.32782/IT/2024-1-7.

2. Nhung, H. L. T. K., Hoc, H. T. & Van Hai, V. “A review of use case-based development effort estimation methods in the system development context”. *software engineering and computer systems: international conference*. 2019. p. 484–499. DOI: https://doi.org/10.1007/978-3-030-30329-7_44.

3. Baeva, O., Kovalenko, O. & Chemerys, N. “Formation of digital competence in the training of future doctors”. *Health of the Nation*, 2025; 3: 135–144. DOI: https://doi.org/10.32782/2077-6594/2025.3/16.

4. Malakhov, K. S. “Insight into the digital health system of Ukraine (eHealth): Trends, definitions, standards, and legislative revisions”. *International Journal of Telerehabilitation*. 2023; 15 (2). DOI: https://doi.org/10.5195/ijt.2023.6599.

5. Soroka, I. M., Kizim, A. V., Mochalov, Yu. O., Bobelskyi, V. V. & Komisar, A. V. “Analysis of the problems of interaction of operators of medical information systems with the systems themselves at the global level (literature review)”. *Bulletin of Social Hygiene and Health Care Organization of Ukraine*, 2025; 4: 128–138. DOI: https://doi.org/10.11603/1681-2786.2025.4.15909.

6. Sittig, D. F., Lakhani, P. & Singh, H. “Applying requisite imagination to safeguard electronic health record transitions”. *Journal of the American Medical Informatics Association*. 2022; 29 (5): 1014–1018, https://www.scopus.com/pages/publications/85128489000. DOI: https://doi.org/10.1093/jamia/ocab291.

7. Litvinov, A. A. & Litvinov, M. A. “On redesign of unified clinico-statistical classification of disease information system”. *System Technologies.* 2021; 2 (133): 3–11. DOI: https://doi.org/10.34185/1562-9945-2-133-2021-01.

8. Silhavy, R., Silhavy, P. & Prokopova, Z. “Using actors and use cases for software size estimation”. *Electronics*. 2021; 10 (5): 592, https://www.scopus.com/pages/publications/85101907677. DOI: https://doi.org/10.3390/electronics10050592.

9. Haryono, A. F., Farhan, A., Khairani, D., Razak, S. & Mintarsih, F. “Use case point activity-based costing and adjusted function point for software cost estimation”. *Jurnal Teknik Informatika*. 2025; 18 (2): 316–326. DOI: https://doi.org/10.15408/jti.v18i2.46669.

10. Ochodek, M., Nawrocki, J. & Kwarciak, K. “Simplifying effort estimation based on Use Case Points”. *Information and Software Technology*. 2011; 53 (3): 200–213, https://www.scopus.com/pages/publications/79251644866.
DOI: https://doi.org/10.1016/j.infsof.2010.10.005.

11. Azzeh, M., Nassif, A. B. & Martín, C. L. “Empirical analysis on productivity prediction and locality for use case points method”. *Software Quality Journal*. 2021; 29: 309–336, https://www.scopus.com/pages/publications/85103656135. DOI: https://doi.org/10.1007/s11219-021-09547-0.

12. Dewi, R. S., Sarno, R. & Astuti, E. S. “A risk-integrated perspective on the influence of technical and environmental complexity in software effort estimation”. *Engineering, Technology & Applied Science Research*. 2025; 15 (6): 30617–30623. DOI: https://doi.org/10.48084/etasr.12404.

13. Nhung, H. L. T. K., Van Hai, V., Silhavy, P., Prokopova, Z. & Silhavy, R. “Incorporating statistical and machine learning techniques into the optimization of correction factors for software development effort estimation”. *Journal of Software: Evolution and Process*. 2024; 36 (5): e2611, https://www.scopus.com/pages/publications/85169480450. DOI: https://doi.org/10.1002/smr.2611.

14. Lytvynov, O. A., Khandetskyi, V. S. & Lytvynov, M. O. “Estimation of effort of migration among domain-driven design architectural variations”. *Radio Electronics, Computer Science, Control*. 2026; 1: 159–175. DOI: https://doi.org/10.15588/1607-3274-2026-1-14.

15. Lytvynov, O. & Lytvynov, M. “High-performance data-intensive software migration effort estimation”. *Preprints.org*. 2026. DOI: https://doi.org/10.20944/preprints202607.1501.v1.

16. Lytvynov, M. & Gerasimov, V. “Operation-oriented model and method for effort estimation of architectural migration among domain-driven design variations”. *EngrXiv*. 2026. DOI: https://doi.org/10.31224/7649.

17. Mahmood, Y., Kama, N., Azmi, A., Khan, A. S. & Ali, M. “Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation”. *Software: Practice and Experience.* 2022; 52(1): 39–65, https://www.scopus.com/pages/publications/85108156511. DOI: https://doi.org/10.1002/spe.3009.

18. Gonçales, L. & Farias, K. “Towards the measurement of mental effort in software engineering: A research agenda”. *International Journal of Computer Applications.* 2020; 177: 1–8. DOI: https://doi.org/10.5120/ijca2020919825.

19. Gonçales, L. J., Farias, K. & da Silva, B. C. “Measuring the cognitive load of software developers: An extended systematic mapping study”. *Information and Software Technology.* 2021; 136, https://www.scopus.com/pages/publications/85104947604.
DOI: https://doi.org/10.1016/j.infsof.2021.106563.

20. Ahsan, Z. & Obaidellah, U. “Eye-Tracking indicators of novice programmers’ proficiency: A machine learning approach”. *ACM Transactions on Computing Education*, 2025; 26 (1): 12, https://www.scopus.com/pages/publications/105029752021. DOI: https://doi.org/10.1145/3773898.

21. Geissler, M., Servin, C., Tang, C., Koumadi, K. & Tucker, C. “Bloom's for computing: enhancing bloom's revised taxonomy with verbs for computing disciplines”. *Association for Computing Machinery (ACM)*. 2023. DOI: https://doi.org/10.1145/3587276.

22. Pontillo, V., Amoroso d'Aragona, D., Pecorelli, F., Di Nucci, D., Ferrucci, F. & Palomba, F. “Machine learning-based test smell detection”. *Empirical Software Engineering*, 2024; 29: 55, https://www.scopus.com/pages/publications/85186768320.
DOI: https://doi.org/10.1007/s10664-023-10436-2.

23. Ramos-Vidal, D., Cortiñas, A., Luaces, M. R., Pedreira, O., Saavedra Places, Á. & Assunção, W. K. G., “Seamless data migration between database schemas with DAMI-Framework: an empirical study on developer experience”. *Proceedings of the 29th International Conference on*

*Evaluation and Assessment in Software Engineering (EASE)*, 2025. p. 453–464, https://www.scopus.com/pages/publications/105027150400.
DOI: https://doi.org/10.1145/3756681.3756947.

24. Adkins, H., Beyer, B., Blankinship, P., Lewandowski, P., Oprea, A. & Stubblefield, A. "Building secure and reliable systems: Best practices for designing, implementing, and maintaining systems". *O'Reilly Media, Inc.* 2020, 558 p. – Available: https://books.google.co.uk/books/about/Building_Secure_and_Reliable_Systems.html?id=93HnDwAAQBAJ. – [Accessed: 24 July 2025].

25. Ortiz-Fuentes, J. D., Herranz, Á. & Colomo-Palacios, R. "Quest for a better coupling metric". *SSRN Preprint*. 2026. DOI: https://doi.org/10.2139/ssrn.6549853.

26. Idris, H. S., Isah, O. M., Fasola, O. O. & Onwudebelu U. "Experience-weighted cognitive complexity metric for software understandability: A cognitive-informatics perspective". *ITM Web of Conferences.* 2026; 81: 01005. DOI: https://doi.org/10.1051/itmconf/20268101005.

27. Menezes, J., Gusmão, C. & Moura, H. "Risk factors in software development projects: a systematic literature review". *Software Quality Journal*. 2019; 27 (3): 1149–1174, https://www.scopus.com/pages/publications/85056268064. DOI: https://doi.org/10.1007/s11219-018-9427-5.

28. Wohlin, C. "Case Study Research in Software Engineering – It is a Case, and it is a Study, but is it a Case Study?". *Information and Software Technology*. 2021; 133: 106514. DOI: https://doi.org/10.1016/j.infsof.2021.106514.

29. Romero-Organvidez, D., Benavides, M., Jordan, M. & Gómez-López, T. "Variability in data transformation: towards data migration product lines". *Proceedings of the 18th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS)*. New York, NY, USA: ACM. 2024. p. 113–121. DOI: https://doi.org/10.1145/3634713.3634724.

30. Wolfart, D., Assunção, W. K. G., da Silva, I. F., Domingos, D. C. P., Schmeing, E., Villaca, G. L. D. & Paza, D. do N. "Modernizing legacy systems with microservices: A roadmap". *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE).* 2021. p. 149–159, https://www.scopus.com/pages/publications/85108906702. DOI: https://doi.org/10.1145/3463274.3463334.

31. "Garvis. Official website". – Available from: https://garvis.com.ua/. – [Accessed: 23 July 2025].

32. "World Health Organization. International statistical classification of diseases and related health problems, 10th Revision (ICD-10)". *2019 ed. Geneva: World Health Organization*. 2019. – Available from: https://icd.who.int/browse10/2019/en. – [Accessed: 25 July, 2025].

33. Lytvynov, M. "Working materials for the article effort estimation for complex business logic migration in legacy systems: A case study of hospital information system modules in Ukraine". *Zenodo*. 2026. DOI: https://doi.org/10.5281/zenodo.21536777.

34. Mu, E. & Pereyra-Rojas, M. "Practical decision making using super decisions v3: An introduction to the Analytic Hierarchy Process (AHP)". Cham: *Springer*. 2018. DOI: https://doi.org/10.1007/978-3-319-68369-0.

35. Litvinov, A. A. "On the variation of abstract factory pattern". *System Technologies.* 2021; 1 (132): 107–115. DOI: https://doi.org/10.34185/1562-9945-1-132-2021-09.

Received 06.07.2026
Received after revision 27.08.2026
Accepted 03.09.2026

**DOI: https://doi.org/10.15276/ict.03.2026.36**
**УДК 004.41**

# Оцінка трудовитрат міграції складної бізнес-логіки в застарілих інформаційних системах: дослідження випадку міграції модулів медичної інформаційної системи в Україні

**Литвинов Михайло Олександрович**[1)]
ORCID: https://orcid.org/0009-0000-9765-1501; litvinovma0@gmail.com
**Герасимов Володимир Володимирович**[1)]
ORCID: https://orcid.org/0000-0002-1366-715X; gerasimov@dnu.dp.ua. Scopus Author ID: 57191378683
[1)] Дніпровський національний університет імені Олеся Гончара, пр. Науки, 72. Дніпро, 49000, Україна

## АНОТАЦІЯ

Об'єктом дослідження є міграція застарілої медичної інформаційної системи до архітектури на основі предметно-орієнтованого проєктування. Проблема полягає у точному оцінюванні трудовитрат в умовах змінних вимог, оскільки методи на основі прецедентів використання застосовують глобальні коригувальні коефіцієнти, однакові для всього проєкту, які є надто абстрактними для врахування локалізованих архітектурних, логічних та технологічних аномалій у межах окремих функціональних блоків. Запропоновано модифікований метод Use Case Size Points, що інтегрує багатовимірний фактор підсилення складності в областях інфраструктури, логіки та технологій, кожна з яких оцінюється за відкаліброваними шкалами індикаторів. Метод відкалібровано та верифіковано на реальних функціональних блоках підсистем Єдиної клініко-статистичної класифікації хвороб української хірургічної клініки. Врахування фактору підсилення складності підвищило точність прогнозування для задач, орієнтованих на запити, приблизно в сім цілих шість десятих рази, а для задач, орієнтованих на виконання команд, – приблизно в три цілих шість десятих рази. В обох випадках отримана похибка перебуває в межах, що вважаються прийнятними для надійного планування проєктів, тоді як похибки базового методу були настільки значними, що призводили до систематичного заниження трудовитрат та перевитрати бюджету. Це покращення зумовлено тим, що фактор підсилення складності враховує нестандартну бізнес-логіку та приховані інфраструктурні й технологічні трудовитрати, які суто структурно-розмірні метрики не відображають. Як результат, керівники проєктів отримують більш реалістичну та достовірну основу для планування графіку, контролю витрат і оцінювання ризиків під час планування аналогічних проєктів міграції застарілих систем. Метод застосовується для оцінювання трудовитрат, планування, формування бюджету та аудиту проєктів міграції застарілих інформаційних систем за умови, що вагові коефіцієнти фактору підсилення складності і коефіцієнт людино-годин перекалібровано для кожного класу завдань та з урахуванням організаційного контексту.

**Ключові слова:** фактор складності; обчислювальні експерименти; оцінювання трудовитрат; медична інформаційна система; міграція застарілого програмного забезпечення; оптимізація; програмне забезпечення

## ABOUT THE AUTHORS

**Mykhailo O. Lytvynov** - PhD Student, Department of Electronic Computing. Oles Honchar Dnipro National University, 72, Nauky Ave. Dnipro, 49000, Ukraine
ORCID: https://orcid.org/0009-0000-9765-1501; litvinovma0@gmail.com
***Research field**:* computer engineering, information technologies, computer science

**Литвинов Михайло Олександрович - а**спірант кафедри Електронних обчислювальних машин. Дніпровський національний університет імені Олеся Гончара, пр. Науки, 72. Дніпро, 49000, Україна
***Галузь досліджень:*** комп’ютерна інженерія, інформаційні технології, комп’ютерні науки

**Volodymyr V. Gerasimov** - Candidate of Engineering Science, Associate Professor, Head of Department of Computer Science and Information Technologies. Oles Honchar Dnipro National University, 72, Nauky Ave. Dnipro, 49000, Ukraine
ORCID: https://orcid.org/0000-0002-1366-715X; gerasimov@dnu.dp.ua. Scopus Author ID: 57191378683
***Research field**:* information technologies, computer science, computer engineering

**Герасимов Володимир Володимирович** - кандидат технічних наук, доцент, завідувач кафедри Комп'ютерних наук та інформаційних технологій. Дніпровський національний університет імені Олеся Гончара, пр. Науки, 72. Дніпро, 49000, Україна
***Галузь досліджень:*** інформаційні технології, комп’ютерні науки, комп’ютерна інженерія