УДК 629.7.054
UDC 629.7.054
DOI: 10.33744/2308-6645-2025-1-60-127-132

ТИПИ ТА ДОЦІЛЬНІСТЬ ВИКОРИСТАННЯ АРХІТЕКТУР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Ходаков Д.В., кандидат технічних наук, Державний університет «Київський авіаційний інститут», Київ, Україна, hodakovdv@gmail.com, orcid.org/0009-0000-7970-151X

TYPES AND FEASIBILITY OF SOFTWARE ARCHITECTURE USAGE

Hodakov D.V., candidate in technical sciences, State University «Kyiv Aviation Institute», Kyiv, Ukraine, hodakovdv@gmail.com, orcid.org/0009-0000-7970-151X

Вступ. Моделювання має багату історію застосування в усіх галузях інженерії та наукових дослідженнях. Неможливо налагодити випуск нових літаків чи автомобілів, не випробувавши свій проект на моделях: від комп'ютерних моделей і креслень до фізичних моделей в аеродинамічній трубі та повномасштабних прототипів. Якісна модель завжди включає елементи, які суттєво впливають на результат і не включає ті, які малозначні на даному рівні абстракції [2]. Кожна система може бути описана з різних точок зору, для чого використовуються різні моделі, кожна з яких є семантично замкненою абстракцією системи. Модель може бути структурною, що підкреслює організацію системи, або моделлю поведінки, що відображає її динаміку.

Для швидкого й ефективного розроблення програмного забезпечення (ПЗ) потрібно залучати високопрофесійну команду розробників, вибрати правильні методи та інструменти роботи. Необхідно, щоб процес розроблення був ретельно продуманий і адаптований до мінливих потреб користувачів, технології розроблення та життєвого циклу (ЖЦ) системи [1]. Основою створення якісного ПЗ є моделювання, що дозволяє:

– наочно продемонструвати бажану структуру і поведінку системи;

– здійснювати візуалізації та керування її архітектурою;

– забезпечити краще розуміння створюваної системи, що найчастіше призводить до її спрощення й забезпечує можливість повторного використання;

– мінімізувати ризики.

Постановка проблеми. Одним з найпродуктивніших напрямків у проектуванні ПЗ стало використання візуальних схем, змальованих на папері й екрані комп'ютера, що відображають ті чи інші вигляди ПЗ.

Аналітична модель – це стисла й точна абстракція того, що саме повинна робити система. Аналітична модель не повинна містити ніяких рішень щодо реалізації. Наприклад, клас Window в аналітичній моделі системи керування вікнами для робочої станції повинен бути описаний у термінах видимих атрибутів і операцій. Аналітична модель складається з двох частин:

1) моделі предметної області (domain model) – опису об'єктів реального світу, що відображає система,

2) моделі програмного додатка (application model) – опису видимих користувачеві частин самого додатка.

Наприклад, для додатка біржового маклера об'єктами предметної області можуть бути акції, облігації, торги й комісія. Модель предметної області, у свою чергу, складається з моделі класів та моделі взаємодії. Об'єкти моделі додатка можуть керувати здійсненням торгів і відображати результати. Гарна модель повинна бути доступною для розуміння та критики з боку експертів, які не є програмістами.

В залежності від складності домену, що в першу чергу задається початковим рівнем його розуміння, процес може трохи відрізнятись, але в середньому можна прийняти наступний порядок дій.

- Занотувати своє розуміння того, яку проблему має вирішувати ПЗ.
- Перерахувати основних користувачів або персон.
- Продумати ключові сценарії використання або скористатися сценаріями замовника.
- Визначити процеси, які необхідні для забезпечення діяльності компанії замовника.

Це, по суті, і буде контекст, який вже можна використовувати для нарощування обсягу інформації: пошуку у Google чи запитів до ChatGPT, чи, в ідеалі, питань до замовника та SME (експерта предметної області).

Цей процес займає від декількох годин до днів і напряму залежить від якості та кількості вхідної інформації.

Аналіз останніх досліджень. Вибір архітектури залежить від багатьох факторів: масштабу проєкту, кількості користувачів, необхідності масштабування та розширення, і навіть технологічних можливостей команди [2, 3]. Кожен тип архітектури має свої плюси і мінуси, і найкраща архітектура – це та, що відповідає конкретним вимогам вашої системи.

Архітектурні шаблони – це готові рішення для повторюваних проблем, які зустрічаються при розробці програмного забезпечення і шаблони допомагають організовувати структуру і взаємодію компонентів в кожному типі архітектури, забезпечуючи гнучкість, масштабованість та ефективне управління ресурсами. Кожен тип архітектури має свої специфічні шаблони, які використовуються для організації взаємодії між компонентами системи [3].

Мова UML створювалася як мова моделювання загального призначення для застосування в таких «дискретних» галузях, як програмне забезпечення, апаратні засоби й цифрова логіка [1]. Структури UML дозволяють фіксувати різноманітні рішення з відображення:

1) функціональності системи;
2) динамічної та статичної структури системи;
3) організації елементів системи;
4) реалізації системи.

Популярності набуває використання UML при проектуванні баз даних. Завдяки відкритості (наявності в мові механізмів розширення) він надає потужний інструментарій для вирішення завдань інших галузей, наприклад, бізнесу-моделювання [1].

Мова UML призначена для вирішення таких завдань.

1. Надати в розпорядження користувачів готову до використання виразну потужну мову візуального моделювання, що дозволяє розробляти осмислені моделі й обмінюватися ними.

2. Передбачити внутрішні механізми розширення й спеціалізації базових концепцій мови.

3. Забезпечити максимальну незалежність проекту створення програмного забезпечення від конкретних мов програмування й процесів розробки.

4. Забезпечити формальну основу для однозначної інтерпретації мови.

5. Стимулювати розширення ринку об’єктно-орієнтованих інструментальних засобів створення програмного забезпечення.

6. Інтегрувати кращий практичний досвід використання мови йреалізації програмних засобів його підтримки.

Основна частина. З розвитком ІТ сфери і продуктів, формувались і підходи до побудови архітектури ПЗ.

Розглянемо основні типи архітектури ПЗ, які на сьогодні активно використовуються.

Монолітна архітектура означає, що вся програма складається з єдиного блоку коду, де всі частини взаємодіють прямо одна з одною (рис. 1). Якщо це велика програма, у ній є всі можливості та функції в одному місці.

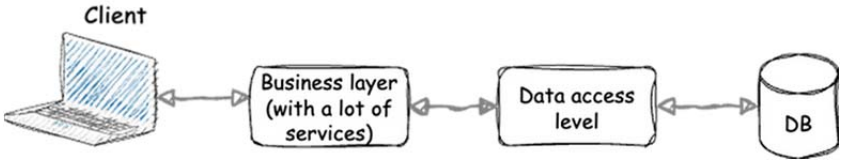

Рисунок 1 – Монолітна архітектура
Figure 1 – Monolithic Architecture

Мікросервісна архітектура поділяє систему на багато дрібних автономних частин (мікросервісів), де кожен виконує окрему функцію (рис. 2). Це фактично міні-програми. Ці частини взаємодіють між собою через спеціальні протоколи (наприклад, через API).

Подійно-орієнтована архітектура (Event-driven architecture). Ця архітектура побудована навколо подій – маленьких повідомлень, що відправляються, коли щось відбувається в системі (рис. 3). Різні компоненти (сервіси) реагують на ці події.

Сервіс-орієнтована архітектура (SOA). SOA – архітектура, де всі частини системи представлені у вигляді незалежних сервісів, які можуть обмінюватися даними один з одним (рис. 4). Подібна до мікросервісної архітектури, але орієнтоване на більші сервіси.

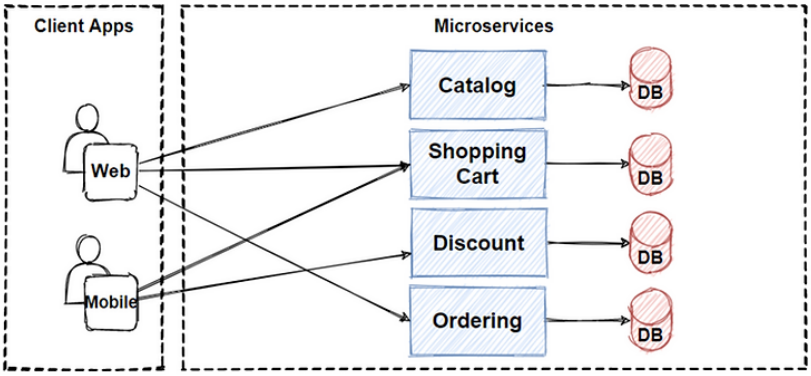

Рисунок 2 – Мікросервісна архітектура
Figure 2 – Microservice Architecture

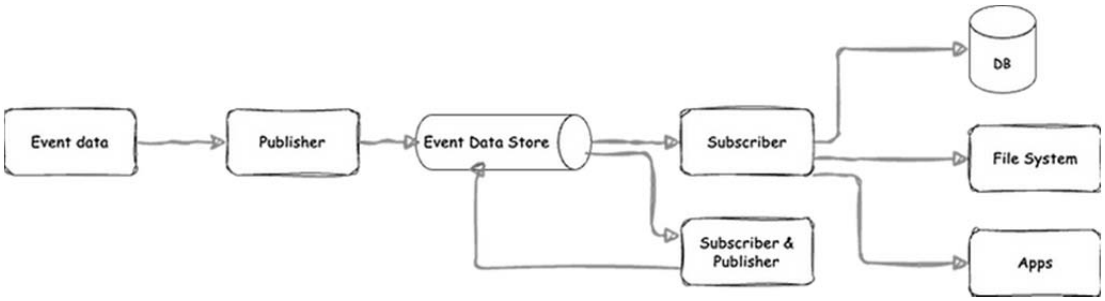

Рисунок 3 – Подійно-орієнтована архітектура
Figure 3 – Event-Driven Architecture

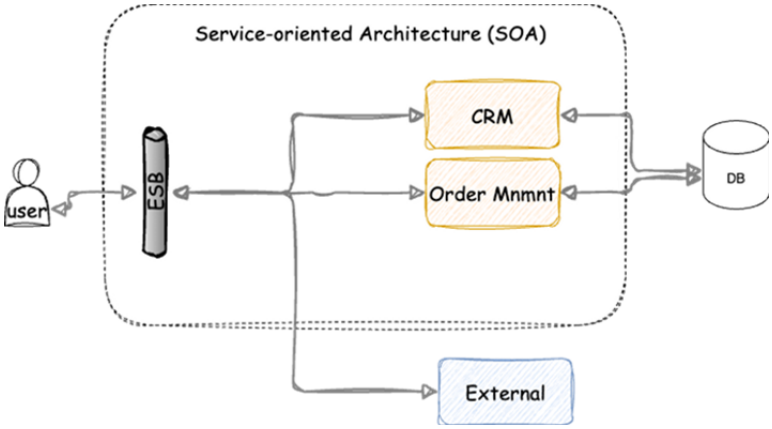

Рисунок 4 – Сервіс-орієнтована архітектура
Figure 4 – Service-Oriented Architecture

Для прикладу візьмемо задачу з реального проєкту, але значно спрощену: «Побудувати систему для морського вантажного терміналу, що буде відповідати за розвантаження контейнерів із судна та розміщення їх у контейнерному депо».

Не залежно від нотації, наступним кроком моделювання буде запис всіх можливих сутностей (об’єктів) у домені (рис.5). Для концептуальної моделі здебільшого це будуть фізичні об’єкти, які потім будуть використовуватися (або ні) у майбутній інформаційній системі.

Ідентифікувати основні об’єкти можна, розглядаючи процеси та сценарії використання, це щось, над чим виконується операція (фізичні об’єкти, документи й т.д.), і сутність яка виконує ці операції.

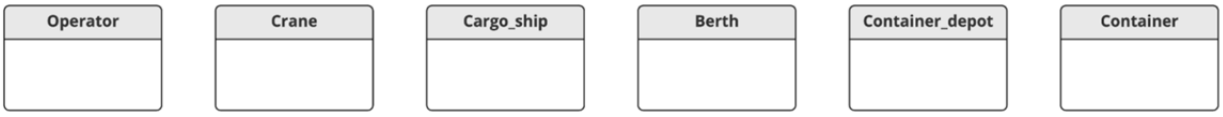

Рисунок 5 – Сутності об’єктів
Figure 5 – Entity Objects

Далі потрібно записати атрибути – характеристики обраних сутностей. Відрізнити атрибут від сутності (об’єкту) дуже просто: атрибути самі ніякої роботи не виконують, вони тільки можуть щось описати. Також в атрибутів є значення, наприклад (рис.6).

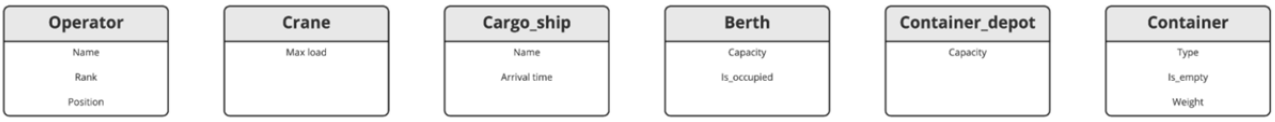

Рисунок 6 – Об’єкт – атрибут – значення
Figure 6 – Object – Attribute – Value

Тепер можна переходити до зв’язків. Зв’язок – це те, яким чином сутності асоціюються одна з одною. Зв’язок можна формулювати: дією, що поєднує дві сутності, або співвідношенням сутностей одна до одної.

Зв’язок в основному встановлюють, керуючись загальною логікою, та іноді його можуть продиктувати атрибути (рис. 7).

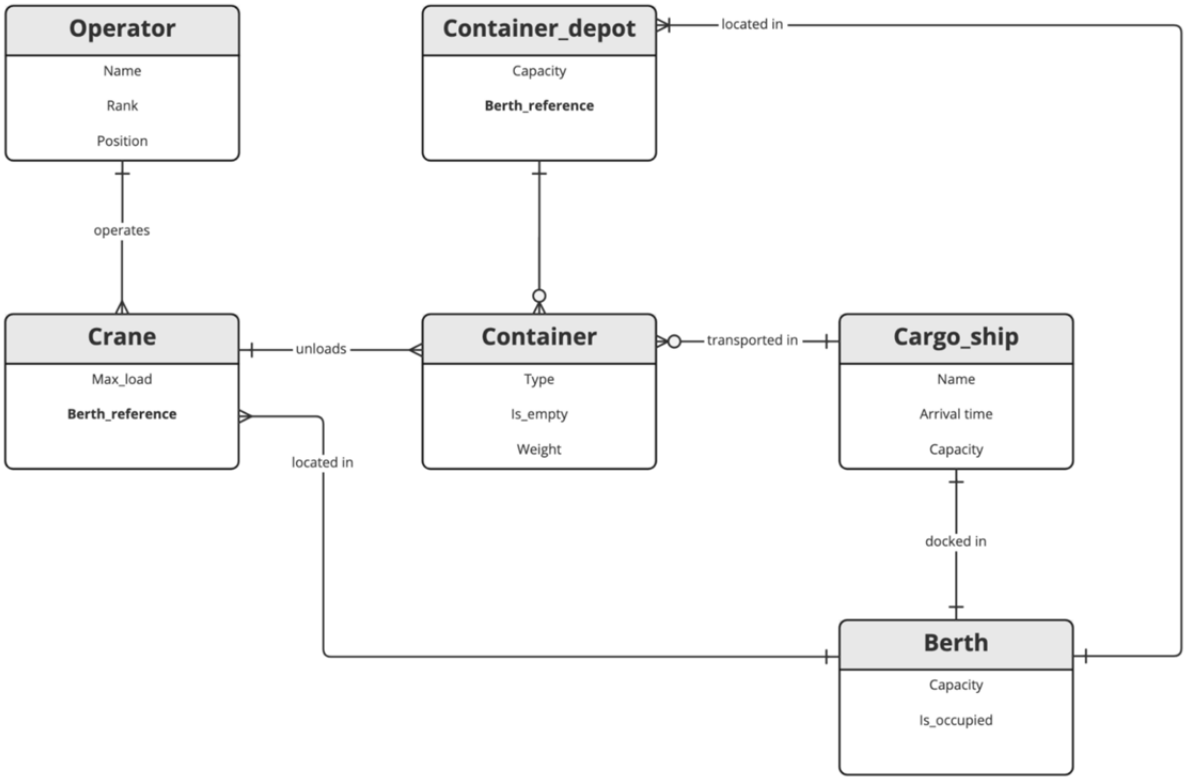

Рисунок 7 – Зв’язки між атрибутами
Figure 7 – Attribute Relationships

У наведеному прикладі:

- один оператор маніпулює декількома кранами,
- декілька кранів розташовані на одному причалі,
- щонайменше одна зона розміщення контейнерів розташована на одному причалі,
- один кран може розвантажити декілька контейнерів,
- багато або жодного контейнера може бути перевезено на кораблі,
- один корабель може бути пришвартовано в одному доці.

Жирним виділено приклад атрибута, який є посиланням на іншу сутність. Кардинальність не відіграє великої ролі при моделюванні концепту. Щоб засвоїти предметну область на рівні, достатньому для проведення пресейлу, це не завжди потрібно, скоріше бажано.

Немає, загально прийнятих конкретних метрик, щоб визначити, коли пора завершувати процес. Поки що, не придумано кращого ніж «має бути достатньо». В залежності від масштабу (галузь знань в цілому, бізнес замовника, конкретний продукт) кількість сутностей і деталізація будуть різними.

Відмінності між типами архітектури програмного забезпечення.

Моноліт – це один великий блок коду, в той час як мікросервіси складаються з дрібних автономних частин. Мікросервіси дозволяють легше розділяти та масштабувати програму, тоді як моноліт простіший на початкових етапах.

Клієнт-сервер має чітке розділення між тим, що бачить користувач (клієнт), і тим, що працює на задньому плані (backend, сервер). Мікросервісна і SOA архітектури більше орієнтовані на поділ сервісів, але SOA орієнтується на великі системи, тоді як мікросервіси – на менші й незалежні.

Архітектура обробки подій фокусується на реакції на події, а не на постійному обміні даними між частинами системи, як це відбувається в інших архітектурах.

Висновок

Таким чином, у більшості випадків ідеї, на яких ґрунтуються нові системи, є продовженням уже існуючих ідей. Існує кілька способів пошуку концепцій нових систем:

– нова функціональність – можна додати функціональність в існуючу систему;

– модернізація – зняття обмежень або універсалізація роботи системи;

– спрощення – надання звичайним людям можливості займатися тим, чим раніше займалися тільки фахівці;

– автоматизація ручних процесів;

– інтеграція – об’єднання функціональності різних систем;

– аналогії – пошук аналогій в інших предметних областях і дослідження їх на наявність корисних ідей;

– глобалізація – вивчення культури й ділової практики інших країн та впровадження їх досвіду.

Вибір архітектури залежить від багатьох факторів: масштабу проєкту, кількості користувачів, необхідності масштабування та розширення, і навіть технологічних можливостей команди.

ПЕРЕЛІК ПОСИЛАНЬ

1. Петрик М.Р., Моделювання програмного забезпечення: науково-методичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.

2. Табунщик Г.В. Проектування та моделювання програмного забезпечення сучасних інформаційних систем: навчальний посібник/ Г.В. Табунщик, Т.І. Каплієнко, О.А. Петрова − Запоріжжя: Дике Поле, 2016. – 250 с.

3. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler – Longman (Pearson Education) – 2003. – 300 P.

REFERENCES

1. Petryk M.R. Modeliuvannia programnogo zabezpechennia: naukovo-metoduchnyi posibnyk/ M.R. Petryk, O.Iu. Petryk – Ternopil: Vud-vo TNTU imeni Ivana Puliuia, 2015. – 200 s.

2. Tabunshchyk G.V., Proektuvannia ta modeliuvannia programnogo zabezpechennia suchasnuh informaciinuh sustem: navchalnyi posibnyk/ G.V. Tabunshchyk, T.I. Kaplienko, O.A. Petrova, – Zaporizhzhia: Dyke Pole, 2016. – 250 s.

3. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler – Longman (Pearson Education) – 2003. – 300 P.

РЕФЕРАТ

Ходаков Д.В.Типи та доцільність використання архітектур програмного забезпечення / Д.В. Ходаков // Вісник Національного транспортного університету. Серія «Технічні науки». Науковий, науково-виробничий журнал. – К.: НТУ, 2025. – Вип. 1 (60).

Кожна система може бути описана з різних точок зору, для чого використовуються різні моделі, кожна з яких є семантично замкненою абстракцією системи. Модель може бути структурною, що підкреслює організацію системи, або моделлю поведінки, що відображає її динаміку. Основою створення якісного програмного забезпечення є моделювання.

Немає, загально прийнятих конкретних метрик, щоб визначити, коли пора завершувати процес. Поки що, не придумано кращого ніж «має бути достатньо». В залежності від галузі знань в цілому, кількість сутностей і деталізація будуть різними. Архітектура обробки подій фокусується на реакції на події, а не на постійному обміні даними між частинами системи.

Аналітична модель не повинна містити ніяких рішень щодо реалізації. Вибір архітектури залежить від багатьох факторів: масштабу проєкту, кількості користувачів, необхідності масштабування та розширення, і навіть технологічних можливостей команди. Кожен тип архітектури

має свої плюси і мінуси, і найкраща архітектура – це та, що відповідає конкретним вимогам вашої системи.

Запропоновано процес моделювання програмного забезпечення за допомогою візуальних схем. Робота над системою починається з розроблення концепції, для цього використовуємо готові архітектурні шаблони. Представлені найбільш пристосовані архітектурні шаблони програмного забезпечення.

Результати статті можуть бути використані під час розробки і впровадження програмного забезпечення. Також результати роботи можуть бути впроваджені в освітній процес під час викладання навчальних дисциплін циклу професійної підготовки фахівців з інженерії програмування.

КЛЮЧОВІ СЛОВА: АРХІТЕКТУРНІ ШАБЛОНИ, МОДЕЛЮВАННЯ, ПРОЕКТУВАННЯ.

ABSTRACT

Khodakov D.V., Types and feasibility of software architecture usage. Visnyk National Transport University. Series «Technical sciences». Scientific, scientific and industrial journal. – K.: NTU, 2025. – Issue 1 (60).

Every system can be described from various perspectives using different models, each being a semantically closed abstraction. A model may emphasize the structure of the system (structural model) or focus on its dynamics (behavioral model). Consequently, modeling forms the foundation for developing high-quality software.

An analytical model must not include implementation details. The choice of architecture depends on multiple factors: the project scale, number of users, scalability and extensibility needs, and even the technological capabilities of the team. Each architectural type has its pros and cons; therefore, the best architecture is the one that meets the specific needs of your system.

There are no generally accepted specific metrics to determine when it is time to end a process. So far, nothing better than “should be enough” has been invented. Depending on the field of knowledge as a whole, the number of entities and the level of detail will vary. Event processing architecture focuses on responding to events, rather than on the constant exchange of data between parts of the system.

This paper proposes a software modeling process using visual schemes. The development process starts with the concept, often built using pre-defined architectural templates. As a result, the most adaptable architectural software patterns are presented.

The results of the article can be used during software development and implementation. Also, the results of the work can be introduced into the educational process during the teaching of educational disciplines of the cycle of professional training of software engineering.

KEY WORDS: ARCHITECTURAL TEMPLATES, MODELING, DESIGNING.

АВТОР:

Ходаков Д.В., кандидат технічних наук, Державний університет «Київський авіаційний інститут», доцент кафедри інженерії програмного забезпечення, e-mail: hodakovdv@gmail.com, тел. +380444067098, Україна, 03058, м. Київ, пр. Любомира Гузара, 1, к.6-303, orcid.org/0009-0000-7970-151X.

AUTHOR:

Hodakov Daniil V., candidate in technical sciences, National Aviation University, associate professor of department of Software Engineering , e-mail: hodakovdv@gmail.com, tel. +380444067098, Ukraine, 03058, Kyiv, Lubomyr Huzar Avenue 1, of. 6-303, orcid.org/0009-0000-7970-151X.

РЕЦЕНЗЕНТИ:

Гізун А.І., кандидат технічних наук, доцент, Державний університет «Київський авіаційний інститут», професор кафедри інженерії програмного забезпечення, Київ, Україна.

Волкогон В.О., кандидат технічних наук, доцент, Державний університет «Київський авіаційний інститут», доцент кафедри інженерії програмного забезпечення, Київ, Україна.

REVIEWER:

Gizun A.I., candidate of technical sciences, associate professor (PhD), State University «Kyiv Aviation Institute», professor of Department of Software Engineering, Kyiv, Ukraine

Volkogon V.O., candidate of technical sciences, associate professor (PhD), State University «Kyiv Aviation Institute», associate professor of Department of Software Engineering, Kyiv, Ukraine