DOI 10.31891/2307-5732-2023-321-3-222-228
УДК 004.42

ПАСІЧНЮК АНТОН
НТУУ «Київський політехнічний інститут імені Ігоря Сікорського»
e-mail: pasichniuk.anton@gmail.com
ТИХОХОД ВОЛОДИМИР
НТУУ «Київський політехнічний інститут імені Ігоря Сікорського»
ORCID ID: 0000-0002-1525-4770
e-mail: v.tikhokhod@gmail.com

МЕТОДИ ТА ЗАСОБИ ПРЕДМЕТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ СКЛАДНИХ ПРОГРАМНИХ СИСТЕМ НА ПЛАТФОРМІ .NET CORE

В роботі проаналізовано проблеми реалізації методів та вибору засобів предметно-орієнтованого проєктування складних проблемних областей, розглянуто особливості реалізації шаблонів, що використовуються для створення програмного забезпечення підтримки діяльності складних областей на платформі .NET Core.

***Ключові слова**: предметно-орієнтоване проєктування, DDD, проєктування складних проблемних областей, шаблони тактичного проєктування.*

PASICHNIUK ANTON
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
TYKHOKHOD VOLODYMYR
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”

METHODS AND TOOLS OF DOMAIN-DRIVEN DESIGN OF COMPLEX SOFTWARE SYSTEMS ON THE .NET CORE PLATFORM

Object-oriented design and approaches to software implementation of systems are relatively new concepts. Templates, tools and technologies related to the implementation of domain models are developing and taking on new forms together with other areas of information technology and software engineering, in particular, with the development of programming languages and platforms, object-relational mapping technologies, microservices, cloud applications. The paper analyzes the problems of implementing methods and choosing means of subject-oriented design of complex problem areas, considers the templates used to create software to support the activities of complex areas on the .NET Core platform.

Existing practices are summarized, methods of implementing tactical design patterns on the .NET Core platform are described. The basic concepts of tactical design templates are shown on the examples of object models made in the UML modeling language. Some examples with fragments of the implementation of subject models in the C# programming language are given, which reflect the peculiarities of the implementation of some aspects of the domain in this programming language and emphasize their peculiarities.

Technical solutions for the implementation of the main concepts of entities, aggregates and objects of values, which are the most important concepts of tactical domain design, are analyzed. The technical features of applying Entity Framework Core object-relational mapping technology for building a high-quality domain model are disclosed. The necessity of structuring the subject area into small isolated parts in order to simplify the task space by using the Module template is emphasized, and the main techniques of implementing this template are highlighted. There are several possible ways of implementing domain events that are used to transfer side effects between aggregates.

***Keywords**: domain-driven design, DDD, design of complex problem areas, tactical design patterns.*

Постановка проблеми та аналіз публікацій. Предметно-орієнтоване проєктування (domain-driven design, DDD) є досить складним напрямком програмної інженерії, що використовується переважно для проєктування систем зі складною логікою предметної області. Шаблон модель предметна область був класифікований Матріном Фаулером [1]. Засновником терміну предметно-орієнтоване проєктування є Ерік Еванс, в своїй праці [2] він описав концепції DDD та практичні аспекти їх реалізації мовою програмування Java. В подальшому теоретичні, практичні та методичні аспекти використання DDD були досліджені і описані в роботах Вона Вернона [3], Мартіна Фаулера [1], Скота Міллета [4], Джимі Нілсена [5] та інших авторів.

Незважаючи на те, що області застосування предметно-орієнтованого проєктування та підходи до програмної реалізації систем розширюються [6–8], концепції, що лежать в основі архітектури таких систем, є відносно новими. Шаблони, засоби та технології, пов’язані з реалізацією моделей предметної області розвиваються та набувають нових форм внаслідок розвитку іншими напрямків інформаційних технологій та програмної інженерії, зокрема, з розвитком мов програмування та платформ, технологій об’єктно-реляційного відображення, технологій реалізації мікросервісів, хмарних технологій тощо.

Платформа .NET Core також є відносно новою сучасною технологією розробки кросплатформних застосунків, що активно розвивається. З появою платформи .NET Core, її розвитком та розвитком мови програмування C# з'явились нові можливості, що значно спростили реалізацію концепцій DDD та покращили способи розробки моделі предметної області в порівнянні з попередньою платформою .NET Framework.

Отже, методи та технології моделювання предметної області розвиваються, внаслідок чого виникає необхідність в перегляді відповідних підходів та практик.

Цілі статті. Виходячи з вище зазначеного актуальним є аналіз та узагальнення сучасних знань щодо проблеми вибору та застосування методів та практик предметно-орієнтованого проєктування на платформі .NET Core.

Практики реалізації шаблонів тактичного проєктування. Предметно-орієнтоване проєктування — це архітектурний стиль, що призначений для створення моделі програмного забезпечення,

в якій максимально точно відображається модель предметної області (домен), що включає бізнес-процеси та правила, що діють в ній. Важливою концепцією DDD є шаблон «єдина мова», що передбачає використання в комунікаціях між розробниками та спеціалістами проблемної області єдиної множини термінів та понять предметної області, які потім відображаються на модель предметної області.

DDD включає стратегічний та тактичний рівні проєктування. На кожному з рівнів використовують шаблони стратегічного та тактичного проєктування відповідно. В результаті стратегічного проєктування проблемна область з метою приборкання складності поділяється на частини, що називаються обмеженими контекстами. Після декомпозиції предметної області з використанням шаблонів стратегічного проєктування, далі застосовують проєктування моделі предметної області в границях обмежених контекстів за допомогою шаблонів тактичного проєктування.

До шаблонів тактичного проєктування відносяться: сутність (Entity), об'єкт значення (Value Type), агрегат (Aggregate), служба (Service), подія (Event), модуль (Module), агрегат (Aggregate), фабрика (Factory), сховище (Repository). Кожен тактичний шаблон має своє призначення, але реалізація цих концепцій відрізняється в різних мовах програмування та на різних платформах.

Шаблон «сутність» використовується для моделювання певних унікальних понять предметної області. Певний набір характеристик забезпечує унікальність сутності, зазвичай це деякий ідентифікатор, що може мати натуральну або синтетичну природу. Сутності в DDD мають поведінку, на відміну від об'єктів анемічної моделі (anemic model) [1]. Тому зовні впливати на сутність можна тільки через її методи. Основні концепції сутності та прийоми їх реалізації перелічені в таблиці 1.

Таблиця 1

Техніки реалізації шаблону Сутність (Entity)

Концепції	Практики та засоби реалізації
Інкапсуляція правил та поведінки	• Властивості тільки для читання (приватний set) — захищає сутності від випадкової, неусвідомленої зміни. • Загальнодоступні методи, в назві яких використовується єдина мова домена — забезпечують усвідомлений вплив на сутність з метою виконання зрозумілої бізнес-операції; виконується перевірка бізнес-правил при виконанні метода; змінити стан сутності можна тільки через виклик методів.
Підтримка коректного (валідного) стану. Концептуальна цілісність	• Приватний конструктор без параметрів — забороняє створювати об'єкт сутності з невідомим станом. • Ініціалізація об'єкта через конструктор з параметрами — забезпечує ініціалізацію сутності з необхідним набором параметрів, що забезпечує коректність та цілісність об'єкта. • Перевірка коректності параметрів в реалізації конструктора — блокують можливість передачі некоректних значень, які впливають на правильну ініціалізацію сутності. • Одним зі шляхів перевірки інваріантів є реалізація шаблону специфікація.
Порівнюваність	• Перевизначення методів Equals базового класу Object. • Перевизначення методу GetHashCode базового класу Object. • Реалізація операторів порівняння.
Унікальність та незмінність ідентичності на протязі життєвого циклу сутності	• Заборона зміни ідентифікатора об'єкту через приватний метод set. • Можливі способи уніфікації: ○ Програмне створення ідентифікатора, наприклад, використання ідентифікатора GUID. ○ Створення ідентифікатора механізмами системи керування базою даних. ○ Використання алгоритму Hi/Lo [9].

На рисунку 1 основні концепції шаблону відображено на діаграмі класів UML моделі сутності доставки. Клас Package має приватний конструктор без параметрів; публічний конструктор з двома параметрами; закриту для змін властивість Weight, що ініціалізується в конструкторі; поле Id тільки для читання, що містить синтетичний ідентифікатор сутності, за створення ідентифікатора відповідає конструктор сутності.

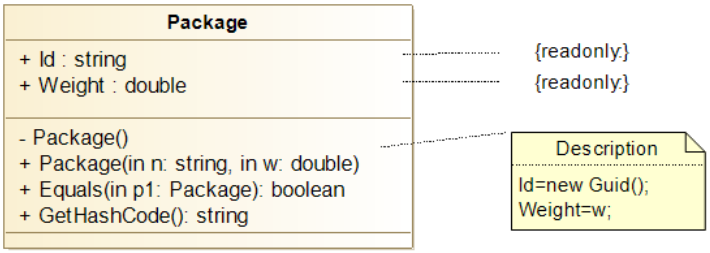

Рис. 1. Діаграма UML, що демонструє основні концепції сутності

Шаблон об’єкт-значення в архітектурі DDD має особливе призначення: він не має індивідуальності та специфікується повним набором його атрибутів, атрибути не змінюються на протязі життєвого циклу об’єкта-значення, при необхідності зміни характеристик об’єкта створюється новий об’єкт-значення з новими унікальними значеннями атрибутів. Основні концепції об’єкта-значення та прийоми його реалізації перелічені в таблиці 2.

Таблиця 2

Техніки реалізації шаблону Об’єкт-значення (Value type)

Концепції	Практики та засоби реалізації
Унікальність за повним набором атрибутів	• Методи порівняння двох об’єктів-значень містять комбінацію логічних операції порівняння атрибутів цих класів.
Відсутність ідентичності	• Відсутнє відображення на власну таблицю в базі даних. • Відсутні ідентифікатори.
Незмінність значень атрибутів	• Властивості тільки для читання (приватний set).
Концептуальна цілісність	• Заборона доступу до конструктора без параметрів. • Ініціалізація об’єкта через конструктор з параметрами.
Порівнюваність	• Перевизначення методів Equals базового класу Object. • Перевизначення методу GetHashCode базового класу Object. • Реалізація операторів порівняння. • Операції порівняння для всіх об’єктів-значень доцільно винести в абстрактний базовий клас [10] з механізмом повного перебору властивостей за допомогою ітератора
Відсутність побічних ефектів	• Методи зміни стану повертають новий об’єкт-значення

На рис. 2 відображено клас прямокутника, що є об’єктом-значенням та реалізує перелічені концепції: клас містить закриті для зміни властивості; значення атрибутів вимагаються в конструкторі; для порівняння об’єктів перевизначається метод Equals базового класу Object, в реалізації якого порівнюються значення всіх атрибутів двох об’єктів; метод Add зміни розмірів прямокутника не впливає на стан об’єкта, а створює та повертає новий об’єкт прямокутника.

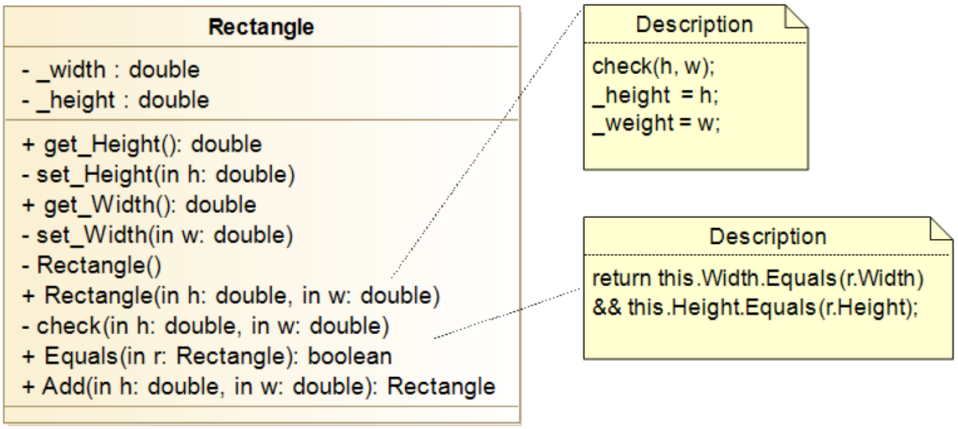

Рис. 2. UML-клас, що демонструє основні концепції об’єкту-значення

Фрагмент найпростішого об’єкта-значення на мові C# може виглядати наступним чином:

```
public class Rectangle
{
    public double Width { get; private set; }
    public double Height { get; private set; }

    private Rectangle() { }

    public Rectangle(double w, double h) { …  }

    public bool Equals(Rectangle r)
    {
         ...
         return this.Width.Equals(r.Width)
                && this.Height.Equals(r.Height);
    }
    public override int GetHashCode()
    {
         ...
         return (this.Width.GetHashCode()
                + this.Height.GetHashCode()).GetHashCode();
    }
    ...
}
```

Для реалізації об'єкта-значення на мові програмування C# доцільно використати готовий базовий універсальний клас [10], що забезпечує механізмом повного перебору властивостей за допомогою ітератора.

Пов'язані сутності об'єднуються в агрегати — ще один шаблон тактичного проєктування. Основною метою агрегатів є об'єднання сутностей в ізольовану модель, що змінюється в рамках однієї транзакції, при цьому агрегат повинен гарантувати узгодженість моделі та дійсність інваріантів в границях цієї моделі. Приблизна структура моделі агрегату доставки зображена на схемі діаграмі класів UML на рис. 3, агрегат включає два об'єкти-значення: адресу та оплату, та колекцію, що включає об'єкти посилок. Основні концепції агрегата, що відображено на схемі: клас Delivery має приватний конструктор без параметрів; публічний конструктор з двома параметрами; закриту для змін колекцію залежних об'єктів Package; публічний метод AddPackage, що забезпечує коректну ініціалізацію об'єкта Package та зв'язування його з потрібним об'єктом Delivery; поле Id тільки для читання, що містить синтетичний ідентифікатор сутності, за створення ідентифікатора відповідає конструктор сутності. Основні концепції агрегату та шляхи їх реалізації представлено в таблиці 3.

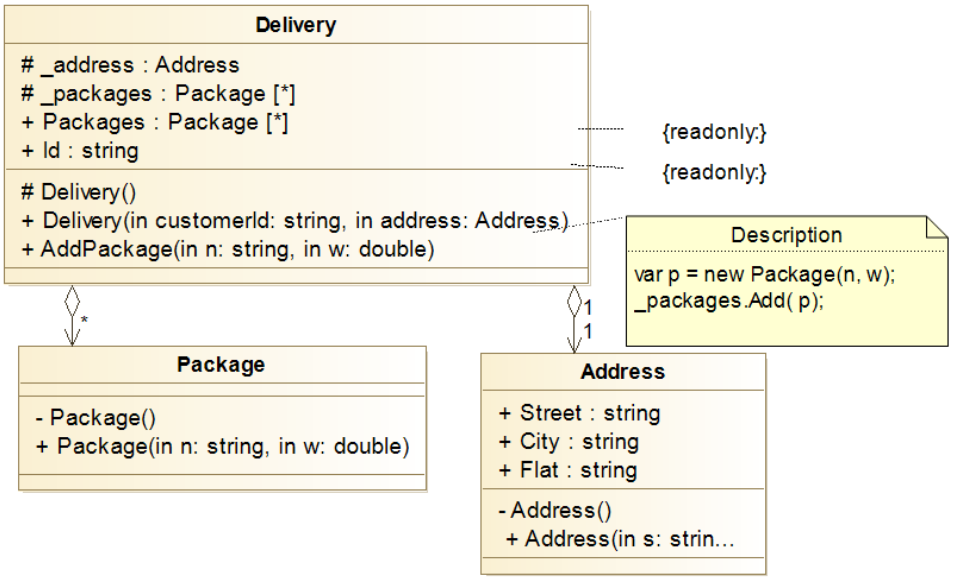

Рис. 3. Модель агрегату доставки

Проаналізуємо техніки реалізації тактичного шаблону Служба. Служби предметної області — це структури, що не мають свого стану, використовуються для виконання бізнес-операцій, що не відносяться до конкретних агрегатів та сутностей. Концептуально служби предметної області слід відрізняти від служб застосунку та служб, що розглядаються в контексті службово-орієнтованої архітектури.

Предметні служби можуть бути потрібні в таких сценаріях як виконання бізнес-операцій, трансформація об'єктів предметної області, агрегування інформації з декількох об'єктів предметної області тощо. В таблиці 4 наведено прийоми реалізації основних концепцій предметної служби.

Таблиця 3

Техніки реалізації шаблону Агрегат (Aggregate)

Концепції	Практики та засоби реалізації
Заборона прямого додавання або зміни зв'язаних сутностей	• Прихована колекція, що містить пов'язані об'єкти — забороняє зміну колекції об'єктів. • Створення IReadOnlyCollection<T> при запиті колекції пов'язаних об'єктів на основі приватної колекції — для постачання колекції залежних об'єктів без можливості зміни колекції.
Підтримка концептуальної цілісності. Визначає границі паралельних транзакцій. Узгодженість графу об'єктів.	• Перевірки інваріантів в реалізації методів. • Приховування посилання на пов'язані об'єкти від зовнішнього світу. • Зміни пов'язаних сутностей тільки через корінь агрегату через виклик методів (наприклад, ChangeAddress та ChangePayment на рис. 4). • Використання однонаправлених зв'язків.
Ізольованість агрегатів	• Різні агрегати посилаються один на одного через ідентифікатори.

Таблиця 4

Техніки реалізації шаблону Служби (Service)

Концепції	Практики та засоби реалізації
Виконання бізнес-сценаріїв предметної області, що не відносяться до відповідальності сутності або значення	• Відокремлення операції в окремий інтерфейс. • Визначається інтерфейс з операцією на мові моделі предметної області. • Ім'я операції включається до єдиної мови обмеженого контексту. • Необхідні об'єкти передаються через параметри методу, якщо операція вимагає декількох об'єктів.
Відсутність власного стану	• Відсутність членів класу, що зберігають значення.
Приховування складності предметної логіки	• Метод служби може інкапсулювати логіку трансформації об'єктів або їх комбінації
Підтримка єдиної мови обмеженого контексту	• Назви методів відображають їх відповідальність з використанням предметних понять та термінів, та також стають частиною єдиної мови.
Забезпечення принципу єдиної відповідальності	• Метод служби реалізує операцію, що не може бути віднесена до інших об'єктів домена.

Технічно предметні служби представляють собою класи, що містять методи, що виконують бізнес-операцію, викликаючи методи інших об'єктів, в яких реалізована предметна логіка, тобто виконують координацію між різними агрегатами та сховищами.

Таблиця 5

Техніки реалізації шаблону Сховище (Repository)

Концепції	Практики та засоби реалізації
Одне сховище на один агрегат	• Використовують базовий шаблонізований клас для всіх сховищ. • Конкретне сховище закривають класом агрегату.
Можливість розширення	• Використовують Шаблон Специфікація (Specification) [11] • Nuget-пакет Ardalis.Specification прискорює реалізацію специфікації [12]
Репозиторій повинен відноситись до домену	• Використовують шаблон розділений інтерфейс (Separate Interface) [1] та інверсію залежності. • Інтерфейс сховища розміщують в моделі домена. • Реалізацію інтерфейса розміщують на рівні інфраструктури.
Методи сховища включаються до єдиної мови	• Методи називають таким чином, щоб вони відображали відповідальність операцій предметної області. • В назві методів використовують терміни предметної області.
Концептуальна цілісністю	• Використання механізму рефлексії для об'єктно-реляційного відображення

Для реалізації збереження та відновлення агрегатів використовують шаблон сховище (Repository), що є проміжним рівнем між рівнем домену та рівнем доступу до даних. При цьому як правило одне сховище відповідає за один агрегат. Практично інтерфейс сховища включають в рівень домену, а класи, що

реалізують репозиторій, відносяться до рівня інфраструктури, далі служби або класи-обробники подій отримують об'єкти сховища через механізм впровадження залежності (Dependency Injection), що налаштовується в застосунку . При програмній реалізації сховищ використовують різні технології, наприклад, інструмент об'єктно-реляційного відображення (Object-Relation Mapping — ORM) Entity Framework, виконання коду SQL з допомогою ADO.NET, NoSQL тощо. В таблиці 5 відображено основні техніки реалізації концепцій сховища.

З виходом Entity Framework (EF) Core стало значно простіше технічно реалізувати відображення домену на реляційну структуру. Оскільки DDD орієнтований на поведінку, а не на дані, то для підтримки узгодженості моделі властивості приховуються для зміни зовні, а змінюються в методі. Ця особливість призводила до значної складності реалізації відновлення об'єктів на попередній платформі Entity Framework, оскільки вона дозволяла відображати тільки публічні властивості. На відміну від цього, EF Core використовує механізм рефлексії для об'єктно-реляційного відображення, що дає можливість налаштувати відображення приватних полів. Так, для класу:

```
public class Payment
{
    private int? _paymentMethodId;
    public int? PaymentMethodId
    {
        public get { return _paymentMethodId; }
        private set { _paymentMethodId = value; }
    }
    ...
}
```

конфігурація для EF Core, що вказує відображати приватне поле _paymentMethodId на колонку PaymentMethodId в таблиці бази даних, використовує для цього метод *UsePropertyAccessMode*:

```
orderConfiguration
    .Property<int?>("_paymentMethodId")
    .UsePropertyAccessMode(PropertyAccessMode.Field)
    .HasColumnName("PaymentMethodId")
    .IsRequired(false);
```

Шаблон «Модуль» в DDD принципово не відрізняється від модуля в об'єктно-орієнтованому програмуванні. Виділення такого шаблону окремо в DDD має на меті підкреслити необхідність структуризації предметної області на невеликі ізольовані частини з метою спрощення простору задач, причому підкреслюється важливість використання єдиної мови домену при моделюванні. Основні техніки реалізації шаблону наведено в таблиці 7.

Таблиця 7

Техніки реалізації шаблону Модуль (Module)

Концепції	Практики та засоби реалізації
Організація та об'єднання спільних понять	• Відокремлення просторів імен (namespace) та проєктів та розподілення в них споріднених програмних структур.
Підтримка слабкої зв'язаності та сильної зв'язності	• Мінімізація посилань між програмними структурами в різних модулях. • Об'єднання об'єктів, що мають тісні зв'язки, в один проєкт.
Використання єдиної мови домену та розширення її	• Назви проєктів та просторів імен повинні відповідати їх призначенню в моделі домену з використанням термінів єдиної мови. • Ієрархія просторів імен та проєктів .Net Core рішення повинна відповідати структурі предметної області.

Шаблон «фабрика» є загальним шаблоном проєктування, що призначений для конструювання класів, відповідальних за створення об'єктів. В архітектурі DDD він займає важливе місце, оскільки забезпечує цілісність домену за рахунок інкапсуляції логіки створення складних об'єктів домену з дотриманням бізнес-правил, інваріантів. Прості об'єкти рекомендується створювати через конструктори класів.

Важливе місце в моделюванні поведінки об'єктів домену посідають події. Події в DDD розділяють на два концептуальні типи: події предметної області та події інтеграції. Події предметної області використовуються для передачі побічних ефектів між агрегатами — коли певні зміни стану одного агрегату здійснюють вплив на інший агрегат. Для програмної реалізації розповсюдження подій часто

використовують компонент MediatR. MediatR надає контракти та автоматично сканує код проєкту, знаходить обробники та зв'язує їх з подіями. Основні технічні рішення реалізації подій предметної області представлено в таблиці 6.

Таблиця 6

Техніки реалізації шаблону Подія (Event)

Концепції	Практики та засоби реалізації
Розповсюдження побічних ефектів між агрегатами	• Події моделюються простими класами даних, що включаються в модель домена. • Nuget-пакет MediatR, що реалізує шаблон медіатр, може бути використаний для побудови інфраструктури розповсюдження подій та керування ними [13]. • Уді Дахан запропонував рішення на основі класу зі статичними методами [14], перевагою використання якого є відсутність зовнішніх залежностей. • Джимі Богарт рекомендує відкладені генерацію та відправку подій [15].

Висновки. В роботі проаналізовано основні концепції шаблонів тактичного предметно-орієнтованого проєктування складних проблемних областей, проведено узагальнення методів та механізмів реалізації розглянутих архітектурних шаблонів на платформі .NET Core, наведено міркування щодо застосування різних технологій при програмній реалізації предметно-орієнтованих моделей.

References

1. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2003. 560 p.
2. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional. 2004. 560 p.
3. Vernon V. Implementing Domain-Driven Design. Addison-Wesley Professional. 2013. 656 p.
4. Millett S., Tune N. Patterns, Principles and Practices of Domain-Driven Design. Wrox. 2015. 800 p.
5. Nilsson J. Applying Domain-Driven Design And Patterns: With Examples in C# and .NET. Addison-Wesley Professional. 2006. 528 p.
6. Oukes P., Marc van Andel, Erwin Folmer, Rohan Bennett, Christiaan Lemmen. Domain-Driven Design applied to land administration system development: Lessons from the Netherlands. Land Use Policy. Issue 104. 2021. DOI: https://doi.org/10.1016/j.landusepol.2021.105379.
7. Design microservices for drones. URL: https://learn.microsoft.com/en-us/azure/architecture/microservices/design
8. Jinsong Zhang, Yan Chen, Shengjun Qin. The Application of Domain-Driven Design in NMS. Proceedings of SPIE. The International Society for Optical Engineering. Issue 8350. 2011. DOI: 10.1117/12.920133.
9. Mihalcea Vlad. The hi/lo algorithm. URL: https://vladmihalcea.com/the-hilo-algorithm/
10. Implement value objects | Microsoft Learn. URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/implement-value-objects
11. Specification Pattern | DevIQ. URL: https://deviq.com/design-patterns/specification-pattern
12. GitHub — ardalis/Specification: Base class with tests for adding. URL: https://github.com/ardalis/Specification
13. GitHub — jbogart/MediatR: Simple, unambitious mediator. URL: https://github.com/jbogard/MediatR
14. Domain Events — Take 2. URL: https://udidahan.com/2008/08/25/domain-events-take-2/
15. A better domain events pattern. URL: https://lostechies.com/jimmybogard/2014/05/13/a-better-domain-events-pattern

.