УДК 004.057.8

Е.О. Горбачова

А.С. Васюра

Методологія предметно-орієнтованого проектування в складних системах

Вінницький національний технічний університет

Анотація

В роботі описано принципи та підходи до проектування складних програмних систем з використанням методології предметно-орієнтованого проектування (domain-driven design-DDD). Визначені основні труднощі, з якими зустрічаються розробники програмного забезпечення и способи їх вирішення в контексті даної методології. Досліджені можливості застосування потужної об'єктно-орієнтованої парадигми.

Ключові слова: предметно-орієнтоване проектування; модель; дизайн; домен; область; контекст; загальна мова; сутність; сукупність; сервіс; фабрика; репозиторій; подія домену; логіка домену; об’єкт значення; програмне забезпечення

Abstract

The paper describes the principle and approaches to the design of complex software systems using the domain-driven design-DDD methodology. The main difficulties that software developers face and the ways in which they are solved in the context of this methodology are identified. Possibilities of application of object-oriented design are investigated.
Keyword: object-oriented design, model, design, domain, context, ubiquitous language, entity, aggregate, service, factory, repository, domain event, domain logic, value object, software

Вступ

Мета розробки будь-якого програмного забезпечення зводиться до оптимізації або автоматизації процесів та явищ, які мають місце в предметній області. Зазвичай, складність процесу розробки залежить від складності меж певного контексту, якого практично неможливо уникнути, через те що він породжений самою суттю предмета. Зрозуміти достатньо глибоко предметну сферу з намаганням побудувати з першого підходу необхідну програмну архітектуру, надзвичайно важко, якщо не сказати, практично не реально, якщо розглядати достатньо великі програмні продукти та вкрай складні області дослідження. Окрім того, часто у розробників програмної архітектури не вистачає достатніх знань про досліджувану предметну сферу, аби швидко побудувати ефективну та зрозумілу для всіх структуру проекту.

Предметно-орієнтоване проектування технологій (англ. Domain-driven design, DDD) – це підхід до моделювання складного об'єктно-орієнтованого програмного забезпечення. Переваги DDD полягають в концентрації основної уваги на предметній області та створенні програмних моделей, які відображують глибоке розуміння предметної області.

В інженерії програмного забезпечення модель дизайну - це загальне повторюване вирішення поширеної проблеми в розробці програмного забезпечення. Шаблон дизайну не є готовим дизайном,

який можна перетворити безпосередньо в код. Це саме опис або зразок того, як вирішити проблему, яку можна використовувати у багатьох різних ситуаціях.

Шаблони проектування можуть пришвидшити процес розробки на підґрунті перевірених, доведених парадигм розвитку. Для ефективної розробки програмного забезпечення необхідно враховувати проблеми, яких може бути не видно до тих пір, поки вони не будуть розглянуті в ході впровадження. Повторне використання шаблонів проектування допомагає запобігти незначним проблемам, які потім можуть призвести до серйозніших. Також покращує читаність коду для програмістів і архітекторів які знайомі із шаблонами.

Метою дослідження є підвищення ефективності розробки програмного забезпечення завдяки застосуванню оптимального предметно-орієнтованого проектування та створенню програмних моделей, що відображують глибоке розуміння предметної області.

Об'єктом дослідження є процес створення моделі дизайну ефективної розробки програмного забезпечення із застосуванням DDD.

Предметом дослідження є метод предметно-орієнтованого проектування та його використання при розробці програмного забезпечення моделі дизайну.

Головною задачею роботи є - розробка моделі дизайну на основі опису завдання; уточнення питань, що виникають під час розробки моделі з експертами в галузі знань; вдосконалення методу проектування програмного забезпечення, який би дозволив розробляти складні системи зберігаючи високу підтримку існуючого коду; розробка архітектури програми на основі моделі дизайну, сформування єдиної мови на основі моделі.

Методи дослідження: методи функціонального, інформаційного моделювання, апарат теорії алгоритмів, основи та принципи проектування програмних систем.

Результати дослідження

Кожний програмний додаток стосується певної діяльності чи інтересу свого користувача. Тематична модель, до якої користувач застосовує програму називають доменною моделлю програмного забезпечення. Домен програми бронювання авіаквитків, який включає реальних людей, що потрапляють у літаки представляє собою модель фізичного світу. Домен облікової програми «гроші та фінанси» представляє собою нематеріальну модель, а домен системи управління вихідним кодом і є розробкою програмного забезпечення.

Щоб створити цінне програмне забезпечення, повинні бути використанні знання, які відносяться до активності сфери використання додатку. При великому об'ємі інформації, які описують дану сферу і область діяльності, команди розробників можуть використовувати моделювання, щоб спростити і структурувати знання. В результаті правильного моделювання створюється бачення проблеми і методі її вирішення [2, ст. 12].

Що стосується самої моделі предметної області вона не є конкретною діаграмою – це ідея, яку діаграма призначила для передачі. Це не тільки знання в голові експерта певної області – це строго організована і розумна абстракція цього знання. Діаграма може представляти і пояснювати модель, як і ретельно написаний код, так і будь-яке речення.

Моделювання предметної області - це не питання створення якомога більш «реалістичної» моделі, та навіть не області матеріальних і реальних речей, а створення штучного творіння. Вона дає необхідні результати, хоча не є просто конструкцією програмного механізму.

Для прикладу, модель предметної області більше схоже на кіно. Так само, як кінорежисер вибирає аспекти та досвід, а потім представляє їх унікальним чином, щоб розповісти історію або сформувати ту точку зору, щоб це було як найбільш реально, модель також визначає те, що потрібно їй.

Тому визначення предметно-орієнтованого проектування (Domain-Driven Design, DDD) слід почати із означення слова домен, а саме точного розуміння та розглянути його в цілому.

Домен (domain) – це сфера знань та діяльності, навколо якої обертається логіка програми. Під час розробки програмного забезпечення, використовують поняття логіки домену або доменного рівня – бізнес-логіка. Бізнес-логіка програми описує взаємодію бізнес-об'єктів один з одним, для створення та зміни модельованих даних.

Як зазначав Ерік Еванс – основоположник терміну проектно-орієнтованого програмування, у своїй книзі «Предметно-орієнтоване програмування: структуризація складних програмних систем (Domain-Driven Design: Tackling Complexity in the Heart of Software)», що предметно-орієнтоване програмування – це розширення та застосування доменної концепції і її відношення до розробки програмного забезпечення [2, ст. 11]. Він спрямований на полегшення створення складних програм шляхом підключення відповідних фрагментів програмного забезпечення до моделі, що постійно розвивається.

Варто зазначити, що предметно-орієнтоване проектування не має конкретної технологією або методології. Це набір правил, які дозволяють приймати правильні проектні рішення. Даний підхід дозволяє значно прискорити процес проектування програмного забезпечення в незнайомій предметній області.

Визначаючи три основних принципи предметно-орієнтованого програмування, складається концепція його успішного застосування:

- Зосередьтеся на основній області та логіці домену.
- Засновувати складні конструкції на моделях домену.
- Постійно співпрацюйте з домен-експертами, щоб поліпшити модель програми та вирішити виникаючі проблеми, пов'язані з доменом.

Е. Еванс згадує кілька загальних термінів, які є важливими при описі методів предметно-орієнтованого програмування:

- Контекст (Context): параметр, у якому з'являється слово чи вислів, що визначає його значення. Значення моделі можна зрозуміти лише в контексті;
- Модель (Model): система абстракцій, яка описує вибрані аспекти домену та може бути використана для вирішення проблем, пов'язаних із цим доменом;
- Загальна мова (Ubiquitous Language): мова, структурована навколо доменної моделі та використовується всіма членами команди для з'єднання всіх видів діяльності команди з програмним забезпеченням;
- Обмежений контекст (Bounded Context): опис межі (як правило, підсистеми або роботи певної команди), в межах якої визначена та застосована конкретна модель;

Враховуючи терміни вище, підхід DDD особливо корисний в ситуаціях, коли розробник не є фахівцем в області розробки продукту. Наприклад: програміст не може знати всіх областей, в яких потрібно створити ПЗ, але за допомогою правильного уявлення і розуміння структури та методів, за допомогою предметно-орієнтованого підходу, може без складності спроектувати додаток, ґрунтуючись на ключових моментах і знаннях робочої області.

Використання предметно-орієнтованим підходу високого рівня концепцій, які можна використати спільно один з одним, є можливим створення та зміна таких доменних моделей:

- Сутність (Entity): Об'єкт, який ідентифікується за його послідовною ниткою безперервності, на відміну від традиційних об'єктів, які визначаються їх атрибутами;
- Об'єкт значення (Value Object): незмінний об'єкт, який має атрибути, але не має виразної ідентичності;
- Подія домену (Domain Event): Об'єкт, який використовується для запису дискретної події, пов'язаної з діяльністю моделі в системі. Хоча всі події в системі можна відслідковувати, подія домену створюється лише для типів подій, про які піклуються фахівці з домену;
- Сукупність (Aggregate) : Скупчення сутностей і об'єктів значень із визначеними межами навколо групи. Замість того, щоб дозволити кожному об'єкту чи об'єкту значення виконувати всі дії самостійно, колективному сукупності елементів присвоюється єдиний сукупний кореневий елемент. Тепер зовнішні об'єкти більше не мають прямого доступу до кожної окремої сутності або об'єкта значення в сукупності,

а натомість мають лише доступ до одного сукупного кореневого елемента та використовують його для передачі інструкцій до групи в цілому;

- Сервіс (Service) : Сервіс - це операція або форма ділової логіки, яка природно не вписується в сферу об'єктів. Іншими словами, якщо якась функціональність повинна існувати, але вона не може бути пов’язана з об'єктом чи об'єктом цінності, це, мабуть, послуга;
- Репозиторії (Repositories): Це особливий вид надійних колекцій, які будуть використовуватися, щоб зберігати і фільтрувати сутності;
- Фабрики (Factories): Відповідає за створення об'єктів предметної області, цим самим покриваючи логіку створення нових об'єктів.

Для вирішення реальних проблем розробники створюють та удосконалюють створений програмний продукт, який буде виконувати поставлену задачу та вирішувати проблему певної сфери, але все, від вибору інструменту до проблеми, що зупиняє (перебиває), обмежує і формує створене програмне забезпечення. Щоб зрозуміти, як зробити код і проблемний домен поєднаними разом, потрібно знатись у їхньому функціонуванні та зв’язках. На рис.1 зображено відношення та функціонування компонентів між собою:

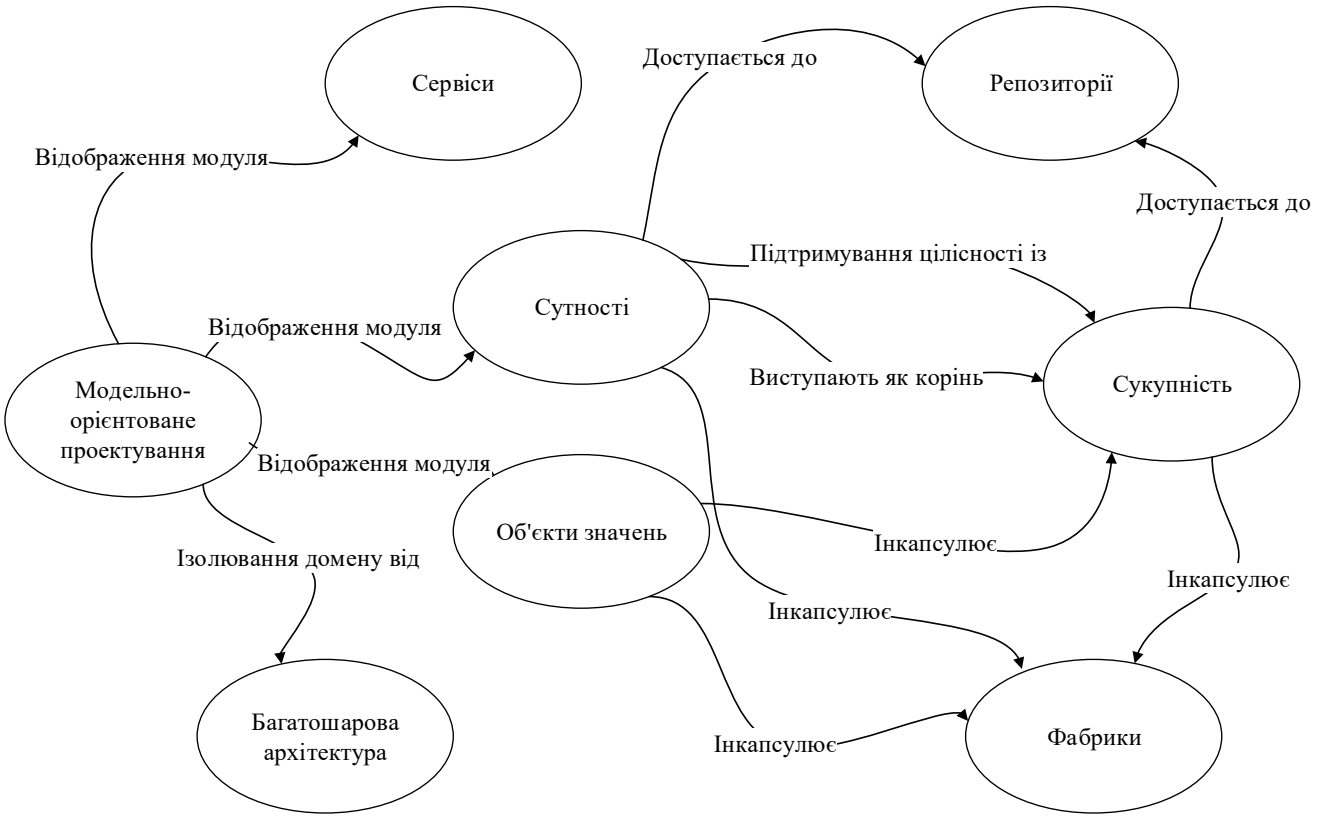

Рисунок 1 – Відношення компонентів предметно-орієнтованого проектування

Предметно-орієнтоване проектування сильно підкреслює все більш популярну практику безперервної інтеграції, яка вимагає від усієї команди розробників використовувати одне спільне сховище коду і щодня надсилати на нього зміни (якщо не кілька разів на день). В кінці робочого дня виконується автоматичний процес, який перевіряє цілісність всієї кодової бази, виконуючи автоматизовані тести блоку, регресійні тести тощо, для швидкого виявлення будь-яких потенційних проблем, які, можливо, були введені в останні коміти.

За допомогою інструментів і технологій, які вже є сьогодні, можна створити набагато цінніші системи, ніж більшість звичайних проектів. Можливо написати програмне забезпечення, яке приємно використовувати та з яким приємно працювати; програмне забезпечення, яке не є незрозумілим для клієнта, а навпаки створює нові можливості та продовжує додавати цінність для своїх власників [2, ст. 352].

Для створення потужної системи, слід враховувати як переваги, так і недоліки. До недоліків предметно-орієнтоване проектування можна віднести:

- Потрібна надійна експертиза домену: Навіть якщо є найбільш технічно налаштовані розуми, які працюють над розробкою – це все нанівець, якщо в команді немає хоча б одного експерта з домену, який знає точні входи та теми в тематичній області, на яку призначено додаток застосування. У деяких випадках дизайн, орієнтований на домен, може вимагати інтеграції одного або декількох сторонніх членів команди, які можуть діяти як доменні експерти протягом життєвого циклу розвитку;
- Заохочує ітеративні практики: Хоча багато хто вважає це перевагою, не можна заперечувати, що «DDD practices» сильно покладається на постійну ітерацію та постійну інтеграцію, щоб створити придатний проект, який може коригувати себе за необхідності. Деякі організації можуть мати проблеми з цими практиками, особливо якщо їхній минулий досвід значною мірою пов'язаний з менш гнучкими моделями розвитку, такими як модель водоспаду тощо;
- Непридатний для високо технічних проектів: Хоча DDD чудово підходить для програм, де існує велика складність домену (де бізнес-логіка є досить і досить складною), DDD не дуже підходить для додатків, які мають граничну складність домену, але навпаки, мають велику технічну складність. Оскільки DDD настільки сильно підкреслює необхідність (та важливість) доменних експертів для створення належної загальної мови, а потім доменної моделі, на якій базується проект, домен.

Перевагою даного виду проектування є ряд таких принципів:

- Полегшує спілкування: З раннім акцентом на встановленні загальної мови, пов'язаної з доменною моделлю проекту, команди часто знайдуть спілкування протягом усього життєвого циклу розвитку набагато простішим. Зазвичай DDD потребує менш технічного жаргону при обговоренні аспектів програми, оскільки загальна мова, створена на початку, швидше за все, визначить більш прості терміни для посилання на ці більш технічні аспекти;
- Поліпшує гнучкість: Оскільки DDD настільки сильно базується на концепціях об'єктно-орієнтованого аналізу та дизайну, майже все в доменній моделі буде базуватися на об'єкті і, отже, буде досить модульним та інкапсульованим. Це дозволяє регулярно і постійно змінювати та вдосконалювати різні компоненти або навіть всю систему в цілому;
- Підкреслює інтерфейс над доменом: Оскільки DDD є практикою побудови навколо концепцій домену та того, що радять фахівці з домену в рамках проекту, DDD часто виробляє додатки, які точно підходять та представляють домен, який знаходиться в руці, на відміну від цих програм які підкреслюють передусім UI/UX. Незважаючи на те, що потрібен очевидний баланс, фокус на домені означає, що підхід DDD може створити продукт, який добре резонує з аудиторією, асоційованою з цим доменом.

Висновки

Переваги DDD полягають в концентрації основної уваги на предметній області та створенні програмних моделей, що відображують глибоке розуміння предметної області. Його слід застосовувати лише в тому випадку, якщо впроваджуються складні мікро сервіси зі значними правилами ведення бізнесу.

Ключовим завданням при розробці та визначенні мікро сервісу є визначення меж. Шаблони DDD допомагають зрозуміти складність у домені. Для моделі домену та кожного обмеженого контексту розробник ідентифікує та визначає сутності, цінні об'єкти та агрегати, що моделюють домен.

Отже, коли створюється програмне забезпечення, воно будується для вирішення проблем певного аспекту області, бізнесу тощо. І тому, зрештою, не має сенсу створювати програмне забезпечення, яке не в змозі досягти очікуваного результату.

Предметно-орієнтоване проектування допомагає розробникам полегшити розуміння поставленої проблеми та побудувати чітке бачення домену та процесів, що виконуються всередині середовища, де вирішується проблема.

Впровадження предметно-орієнтованого проектування не є простим, тому його слід піддавати аналізу, якщо це є необхідним. Воно існує для застосування у складних бізнес-сценаріях. Якщо домен, який розробляється, достатньо складний, предметно-орієнтоване проектування – це найкращий вибір.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Design Patterns: https://sourcemaking.com/design_patterns
2. Evans E. Domain-Driven Design - Tackling Complexity in the Heart of Software. — Addison-Wesley, 2004. — 529 c. — ISBN 978-0-3211-2521-7
3. Domain-Driven Design – What is it and how do you use it?: https://airbrake.io/blog/software-design/domain-driven-design
4. What is DDD?: https://thedomaindrivendesign.io/what-is-ddd/
5. DDD 101 — The 5-Minute Tour: https://medium.com/the-coding-matrix/ddd-101-the-5-minute-tour-7a3037cf53b8
6. Implementing DDD with the Spring ecosystem: https://speakerdeck.com/mploed/implementing-ddd-with-the-spring-ecosystem?slide=4
7. Get your feet wet with domain-driven design: 3 guiding principles: https://techbeacon.com/app-dev-testing/get-your-feet-wet-domain-driven-design-3-guiding-principles
8. Domain Driven Design на практике: https://habr.com/ru/post/334126/
9. Учимся проектировать на основе предметной области (DDD: Domain Driven Design): https://habr.com/ru/post/61524/

Горбачова Евеліна Олександрівна — студентка групи 2АКІТ-17б, факультет комп'ютерних систем і автоматики, Вінницький національний технічний університет, м. Вінниця, e-mail: e.horbachova@gmail.com
Науковий керівник: ***Васюра Анатолій Степанович***—професор кафедри автоматики і інформаційно-вимірювальної техніки, Вінницький національний технічний університет, м. Вінниця

Horbachova Evelina O.—Department of Computer System and Automation, Vinnytsia National Technical University, Vinnytsia,, e-mail: e.horbachova@gmail.com
Supervisor: ***Vasyura Anatoliy S.***— Professor of Automation and Information and Measurement Engineering Department, Vinnytsia National Technical University, Vinnytsia