УДК 004.41

Т. Ланевич
(Тернопільський національний технічний університет імені Івана Пулюя)

ЗАСТОСУВАННЯ ДОМЕННО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ У ГНУЧКІЙ АРХІТЕКТУРІ

U. Lanevych

THE USE OF DOMAIN-DRIVEN DESIGN IN A FLEXIBLE ARCHITECTURE

У сучасному складному світі розробки програмного забезпечення, створення додатків, які вирішують реальні бізнес-проблеми, виходить далеко за межі написання хорошого коду. Це вимагає глибокого розуміння предметної області – конкретної галузі знань або діяльності, для підтримки якої розробляється програмне забезпечення. Це основна передумова доменно-орієнтованого проєктування (Domain-Driven Design, DDD) – методології, яка сформувала сучасну архітектуру програмного забезпечення, зокрема, у зв'язку з розвитком мікросервісів. DDD є життєво важливою основою для створення масштабованих, підтримуваних та бізнес-орієнтованих програмних систем.

Domain-Driven Design є підходом, спрямованим на проєктування програмного забезпечення через глибоке розуміння предметної області, зосереджуючись на бізнес-логіці. В умовах Agile цей підхід стає особливо цінним завдяки можливості швидко адаптуватися до змін, підтримуючи модульність та контроль залежностей.

Основні принципи DDD:

1. Єдина мова (Ubiquitous Language). Під час розробки програмного забезпечення всі члени команди користуються спільним словником або мовою. Вона має на меті усунути непорозуміння між розробниками та експертами предметної області, дозволяючи їм спілкуватися більш ефективно і точно.

2. Обмежені контексти (Bounded Contexts). Дозволяють розділяти систему на самостійні компоненти, щоб зменшити міжмодульні залежності.

3. Сутності (Entities). Це об'єкти, які мають чітку ідентичність, що зберігається з часом. Вони представляють основні концепції предметної області, де ідентичність має вирішальне значення, навіть якщо інші атрибути змінюються.

4. Об'єкти-значення (Value Objects) – На відміну від сутностей, об'єкти-значення не мають чіткої ідентичності. Вони визначаються своїми атрибутами і є незмінними, тобто не можуть бути змінені після створення.

5. Агрегати та агрегатні корені (Aggregates and Aggregate Roots). Це група пов'язаних сутностей і об'єктів, які розглядаються як єдине ціле. Корінь агрегату є первинною сутністю, яка контролює доступ до інших об'єктів у межах агрегату.

Переваги застосування доменно-орієнтованого проєктування:

1. Єдина мова і чіткі межі контексту покращують комунікацію між членами команди і зацікавленими сторонами. Це створює дуже важливий зв'язок між технічними і бізнес-командами, який може змінити правила гри для нового дизайну продукту, який потрібно підтримувати в часі.

2. Підвищена гнучкість. Використання обмежених контекстів та агрегатів у DDD створює більш адаптивний і гнучкий підхід до проєктування. Обмежені контексти встановлюють чіткі межі, що дозволяє різним частинам системи розвиватися незалежно,

зменшуючи вплив змін в одній частині на інші. Це особливо важливо для великих і складних систем, де різні модулі можуть мати специфічні вимоги. Агрегати, інкапсулюючи пов'язані сутності та об'єкти-значення, спрощують модель, роблячи систему більш керованою. Модульна структура підтримує чистоту коду та дає змогу використовувати різноманітні технології, відповідаючи на зміни бізнес-вимог і технологічних ландшафтів. DDD добре поєднується з мікросервісами та модульними архітектурами, сприяючи гнучкості у розробці.

3. Висока згуртованість і слабкі зв'язки. У доменно-орієнтованому дизайні агрегати забезпечують високу згуртованість, групуючи тісно пов'язані сутності та об'єкти-значення. Таке тісне внутрішнє групування підвищує чіткість і цілісність системи. Водночас, обмежені контексти підтримують вільний зв'язок, дозволяючи різним частинам системи взаємодіяти з мінімальними залежностями, тим самим підвищуючи гнучкість і простоту обслуговування.

Недоліки застосування доменно-орієнтованого проєктування:

1. Складність. Детальне моделювання, притаманне DDD, може зробити процес розробки більш складним. Це особливо помітно в тонкому моделюванні бізнес-логіки та розмежуванні обмежених контекстів і агрегатів. Хоча така складність допомагає в роботі зі складними бізнес-доменами, вона може бути непосильною для простіших проєктів.

2. Забирає багато часу. Впровадження DDD, зокрема створення єдиної мови та постійне вдосконалення моделі, є трудомістким процесом. Це робить DDD менш придатним для проєктів з обмеженими термінами або з прямолінійною логікою, де глибокий аналіз не потрібен. У ситуаціях, де важлива швидкість, тривалий процес DDD може стати перешкодою.

3. Вимоги до навичок. DDD вимагає високого рівня експертизи в принципах проєктування та глибоких знань бізнес-домену, що може бути викликом для недосвідчених команд. Брак досвіду призводить до труднощів в ефективному впровадженні DDD, що може збільшити час розробки і призвести до неоптимальних результатів.

Висновок. Доменно-орієнтоване проєктування пропонує структурований підхід до розробки програмного забезпечення для складних бізнес-сфер, орієнтуючись на глибоке розуміння домену за допомогою єдиної мови, обмежених контекстів та агрегатів. Це підвищує гнучкість систем і полегшує масштабованість архітектури. Водночас DDD потребує значних витрат часу та експертизи, що може ускладнити його застосування в простих проєктах або в умовах обмежених ресурсів і термінів.

Література

1. Architecture Approach : Domain-Driven Design (DDD) [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/architecture-approach-domain-driven-design-ddd-momen-negm-lprof.

2. Domain-Driven Design (DDD): The Foundation of Modern Software Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://roshancloudarchitect.me/domain-driven-design-ddd-the-foundation-of-modern-software-architecture-f2b07c1d1801.

3. Microservices Done Right: Using Domain Driven Design and Hexagonal Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@abdessamad.abidar/microservices-done-right-using-domain-driven-design-and-hexagonal-architecture-739f6c50ea3b.

4. Power of DDD (Domain-Driven Design) in Building Scalable Applications [Електронний ресурс] – Режим доступу до ресурсу: https://blog.emb.global/ddd-domain-driven-design/.