УДК 004.891

к.т.н., доц. **Муляр І.В.** (ХмНУ)
к.т.н., доц. **Браун В.О.** (ВІКНУ)
к.т.н., доц. **Шваб В.К.** (ВІКНУ)
Проценко Я.М. (ВІКНУ)
Глушко Р.М. (ХмНУ)

ПРЕДМЕТНО-ОРІЄНТОВАНИЙ ПОГЛЯД НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті розглядається теоретичні основи та переваги використання предметно-орієнтованого підходу до проектування та реалізації програмного забезпечення.

Зазвичай розробка программного забезпечення покликана вирішити проблеми певної предметної області або автоматизувати процеси, які там мають місце і важливі для кінцевого користувача. Проте сам процес розробки – це задача не тривіальна і його складність напряму залежить від того з якою предметною областю доведеться працювати. Як правило, складність предметної області – важко уникнути, так як вона порождена самою сутью цієї області. Тому доводиться опановувати складність проблеми або процесу.

На теперішній час існують багато підходів до проектування програмного забезпечення (наприклад, екстремальне програмування), які звертають увагу і беруть за основу різні аспекти життєвого циклу програмного продукту або проекту. Усі вони так чи інакше взаємодіють с предметною областю, проте жоден із них не орієнтується на неї. В них, процес проектування предметної моделі відділений від її реалізації засобами програмування, звертається особлива увага на проектування архітектури програмного забезпечення і предметна модель – є лише придатком до документації. Предметно-орієнтоване проектування показує, що такі підходи є неефективними, особливо це помітно на великих проектах із дуже складною предметною сферою. У ньому, головною частиною розробки програмного забезпечення є створення і удосконалення предметної моделі, на основі якої всі складності предметної області намагаються пояснити і викласти.

Побудова предметної моделі та її ітераційне удосконалення, в процесі розвитку та еволюції програмного продукту дозволяють побудувати зрозумілу та чітку архітектуру програмної системи, а також організувати ефективну комунікацію в середині команди, яка займається розробкою та спеціалістами в предметній області, які не маю потрібних навиків, щоб зрозуміти розроблений програмний код. Такий підхід дозволяє розробляти програмні продукти будь-якої складності та стане можливим додавання нового функціоналу або підтримку існуючого ефективно.

Ключові слова: розробка програмного забезпечення, моделювання, предмента сфера, багаторівнева архітектура, предметно-орієнтоване проектування, комунікація, документування коду

Вступ. Мета розробки будь-якого програмного забезпечення зводиться до оптимізації або автоматизації процесів та явищ, які мають місце в предметній області. Як правило, складність процесу розробки залежить від складності предметної області, яку практично неможливо уникнути, через те що вона породжена самою суттю предмета.

Розуміти достатньо глибоко предметну сферу аби побудувати с першого разу підходящу програмну архітектуру – важно або навіть не реально, якщо розглядати великі програмні продукти і дуже складні предметні сфери. Також часто не вистачає знань про предметну сферу у розробників програмної архітектури, щоб швидко побудувати ефективну та зрозумілу для всіх структуру проекту. Також це зумовлено широтою, глибиною, нечіткістю можливих предметних сфер, які людина не спроможна освоїти до початку роботи із проектом.

Часто зміни, які потрібно вносити розробникам в систему тісно зв’язані з предметною сферою. Знайти місце для цих змін часто буває важко, а результат таких змін – робить програмний код системи важким для розуміння того, для чого він призначений. Таким чином накопичується код, котрий важко розуміти, змінювати та відлагоджувати. Такий процес лімітує можливості програмістів розширяти систему, тому що постійно росте складність

системи, що неминуче веде до виходу проекту із під контролю та зниженню можливості здійснити корисні удосконалення для системи.

Не рідко розробники програмних систем звертають багато уваги на стек технологій та намагаються вирішити проблеми предметної області тільки за допомогою цього стеку. Вони мало цікавляться предметною стороною свого проекту та намагаються вирішувати його проблеми через свої технічні навики. В цьому випадку, ми отримаємо продукт, в якому не буде відображено особливості предметної моделі, що означає складність у внесненнях подальших змін та підтримці продукту.

Складні системи зазвичай розробляє група програмістів. Розробники спілкуються між собою, передають досвід та знання в процесі роботи. Таким чином в розробці програмних систем з'являється соціальний фактор, який також має свої проблеми. Основна і найбільш спільна проблема – це ефективність комунікації, яку знижують неоднозначність трактування термінів та використання незрозумілих трактовок, особливо у спілкуванні з спеціалістами у предметній області. Так і виникає певний мовний бар'єр.

Постановка задачі. Розробка програмних систем – це нетривіальна і складна справа. Вона потребує творчого мислення і стадії проектування та обдумування роботи. Також у ній є достатньо проблем, з якими може зустрітись і зустрічався кожен практикуючий розробник. Тому важливо які підходи використовуються при розробці програмної системи та які проблеми вони вирішують та породжують.

Зрештою ціллю практично будь-якого програмного проекту є створення системи, яка буде вирішувати певні бізнес задачі та задовольняти потрібні регламенти. Хоча це не завжди явно задано, але завжди клієнт буде очікувати від розробників систему, яку буде спроможна еволюціонувати, обростати новим функціоналом, відповідно до існуючих потреб в той чи інший момент.

Еволюція програмного продукту – це дуже важливий процес, в умовах наявності конкуренції та зміни трендів в галузі знань. Програмні системи, в архітектурі яких не закладений потенціал для розвитку і змін, рано чи пізно будуть витіснені такими, які це можуть робити, а від систем, які важко розвивати, як правило, відмовляються самі власники, тому що підтримка і розвиток такої системи буде потребувати довгого часу та великих грошових інвестицій.

Саме тому важливо розробляти системи, які можна буде за короткий час та адекватну ціну розвивати, розширяти та удосконалювати. Для цього нам потрібно оптимізувати комунікацію в середині команди розробників, будувати зрозумілу та очікувану архітектуру програмного продукту, яка буде відповідати галузі знань, створювати просту та зрозумілу документацію для важливих бізнес правил або процесів. В реалізації поставлених цілей може допомогти предметно-орієнтоване проектування.

Основна частина. Предметно-орієнтоване проектування розвиває дві основоположні ідеї [3]:

1. Логічна структура предметної області і взаємозв'язок в ній – це головне, на що потрібно звертати увагу, під час проектування програмних систем

2. Архітектура програмного забезпечення повинна базуватись на предметній моделі.

Предметно-орієнтоване проектування – це образ мислення и система пріоритетів, яка створена для прискорення розробки програмних продуктів, яка використовується в складних галузях знань. Для досягнення цих цілей, буде розглянуто набір прийомів, підходів та принципів проектування програмних забезпечень. Предметно-орієнтоване проектрування не заперечує і не перевизначає принципів SOLID [8], DRY або шаблонів проектування [1], проте формує новий підхід до аналізу і побудови програмних систем.

Для того щоб мати змогу використовувати предметно-орієнтоване проектування, процес розробки програмної системі повинен мати такі властивості:

- ітераційний характер розробки (таку властивість мають всі гнучкі методики розробки програмного забезпечення)

- тісний зв'язок розробників системи із спеціалістами в предметній сфері (особливість предметно-орієнтованого проектування в тому, що для його реалізації потрібно багато знань, які відображають глибоке розуміння предметної області та концентрацію на її ключових концепціях, тому необхідно мати діалог між фахівцями в предметній сфері та розробниками протягом всього часу реалізації проекту).

Основою предметно-орієнтованого проектування – є створення предметної моделі. В широкому сенсі, модель – спеціально відібраний набір знань, які представленні в структурованому вигляді, вона надає інформації сенсу і дозволяє сконцентруватись на проблемі. Предметна модель – це така вибірка інформації із галузі знань, яка описує аспекти потрібні для вирішення проставлених задач. Вона може буди представлена у вигляді схеми, яка відображає ідеї моделі або у вигляді опису людською мовою. Предметна модель повинна мати такий рівень деталізації, який дозволить будувати програмний механізм, який буде давати потрібний результат.

Вибір предметної моделі визначається такими фундаментальними способами її використання при розробці:

1. Предметна модель і архітектура взаємо визначають один одного
2. Предметна модель лежить в основі єдиної мови, котру використовують в середині команди розробників
3. Предметна модель – це головні значення, які потрібно взяти із предметної сфери.

Предметна модель повинна бути невід'ємно зв'язана із архітектурою програми. Саме цей зв'язок і робить предметну модель цінною і потрібною для розробки програмної системи. Також він гарантує, що аналіз, який був пророблений при її аналізі відобразиться в кінцевому результаті. Зв'язок між моделлю та реалізацією забезпечую краще розуміння програмного коду та в подальших розширеннях програми, оскільки код можна інтерпретувати базуючись на знаннях предметної моделі. Виходячи із цього правила, кожен розробник повинен мати досвід у виражені моделі в програмному коді, а всі, хто відповідають за створення моделей – повинні деякий час працювати з кодом, яку би передову роль в процесі розробки вони не займали.

Рис. 1. Двосторонній характер зв'язку предметної моделі із програмною архітектурою

Важливою частиною розробки програмного забезпечення, роль якою часто недооцінюють, є провадження єдиної мови для девелоперів та спеціалістів предметної сфери. Фахівці в предметній сфері використовують свою лексику для спілкування, у програмістів – своя мова, адаптована для опису предметної галузі в термінах програмної архітектури. Проте ні один із цих діалектів не може бити використаний, в якості спільної мови. Причиною того є те, що ні один із них не призначений для того, щоб описати поставлені задачі. Це створює свого роду мовний бар'єр, неоднозначність повідомлень та непорозуміння.

Рис. 2. Суть проблем комунікації між розробниками та фахівцями

Єдина мова покликана вирішити цю проблему в комунікації [10]. В її основі лежить предметна модель. Терміни та взаємозв'язки предметної моделі утворюють семантику єдиної мови. Це ще один зв'язок предметної моделі із програмною реалізацією. Єдина мова – це не тільки спосіб подолати непорозуміння, але і спосіб перевірити зручність і логічність термінів із предметної моделі. Часто озвучення елементів та звя'зків, які дозволяє предметна модель, дозволяє знайти і виявити неоднозначності та знайти простіші варіанти для опису тих же речей і відношень. В таких випадках, предметна модель повинна бути удосконалена. З іншого боку, будь-які зміни в предметній моделі повинні буди відображені єдиній мові. Таким чином, зв'язок між предметною моделлю та єдиною мовою – також двосторонній.

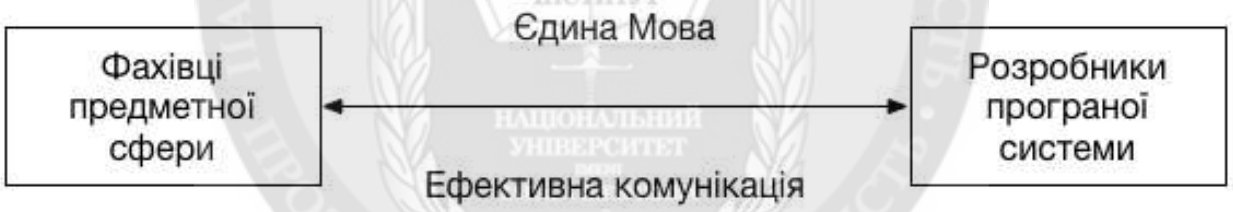

Рис. 3. Єдина мова вирішує проблему комунікації та руйнує мовний бар'єр

Єдина мова знаходить своє місце, при створені докуменації. Письмова документація – один із найважливіших способів децентралізувати знання про програмну систему та поділитися своїми знаннями про продукт і особливості його реалізації. В основі документації повинна лежати єдина мова, яка зробить її чіткою та однозначною. До неї ставлять наступні вимоги:

- документація повинна доповнювати код та усні дискусії,
- документація повинна активно обновлятись та не відставати від процесу розробки.

У документації потрібно відображати сенс, допомогти в розумінні крупномасштабних структур та сконцентруватись на ключевих моментах. Якщо дотримуватись цього правила, то документація буде містити ті частини, які не будуть мінятись швидко та динамічно, тому її оновлення не буде займати багато часу. Докумантація може прояснити суть програмної архітектури там, де в мові програмування не вистачає засобів для достатньо ясного її вираження. Часто документації доповняють схемами та UML-діагмами. Проте вони повинні бути простими та допомогати краще зрозуміти складні частини архітектури [2].

Для того, щоб реалізація програмної системи велась в чіткій відповідності до предметної моделі, потрібно використовувати найбільш надійні методи проектування та моделювання. У цій статі буде обговорено використання об’єктно-орієнтованої парадигми програмування. У свою чергу предметно-орієнтоване проектування лише зміщує деякі загально прийняті акценти у цій сфері.

Щоб предметна модель та її реалізація завжди відповідали та посилювали одне одну, необхідно приймати певні види рішень. Синхронізація моделі та програми вимагає уваги до деталі окремих структурних елементів програми. Тому потрібно детально проектувати архітектуру в малому масштабі, щоб отримати тверду основу для подальшої розробки.

Багато в чому предметно-орієнтоване проектування слідує принципам “проектуванню згідно з відповідальністю”, яке представляє об’єкти, як певну абстракціє для поведінки, яка характеризується через їхню поведінку.

Рис. 4. Основні структурні елементи предметно-орієнтованого проектування та зв'язок, котрий існує між ними

Важливою частиною предметно-орієнтованого проектування є ізоляція предметної області від іншого коду, котрий існує в системі. Це дозволить уникнути складності, яка виникає коли змішуються поняття із предметної області та програмними технологіями або взагалі втратити бачення предмету в загальній системі. Для такого розділення можна використовувати багаторівневу архітектуру.

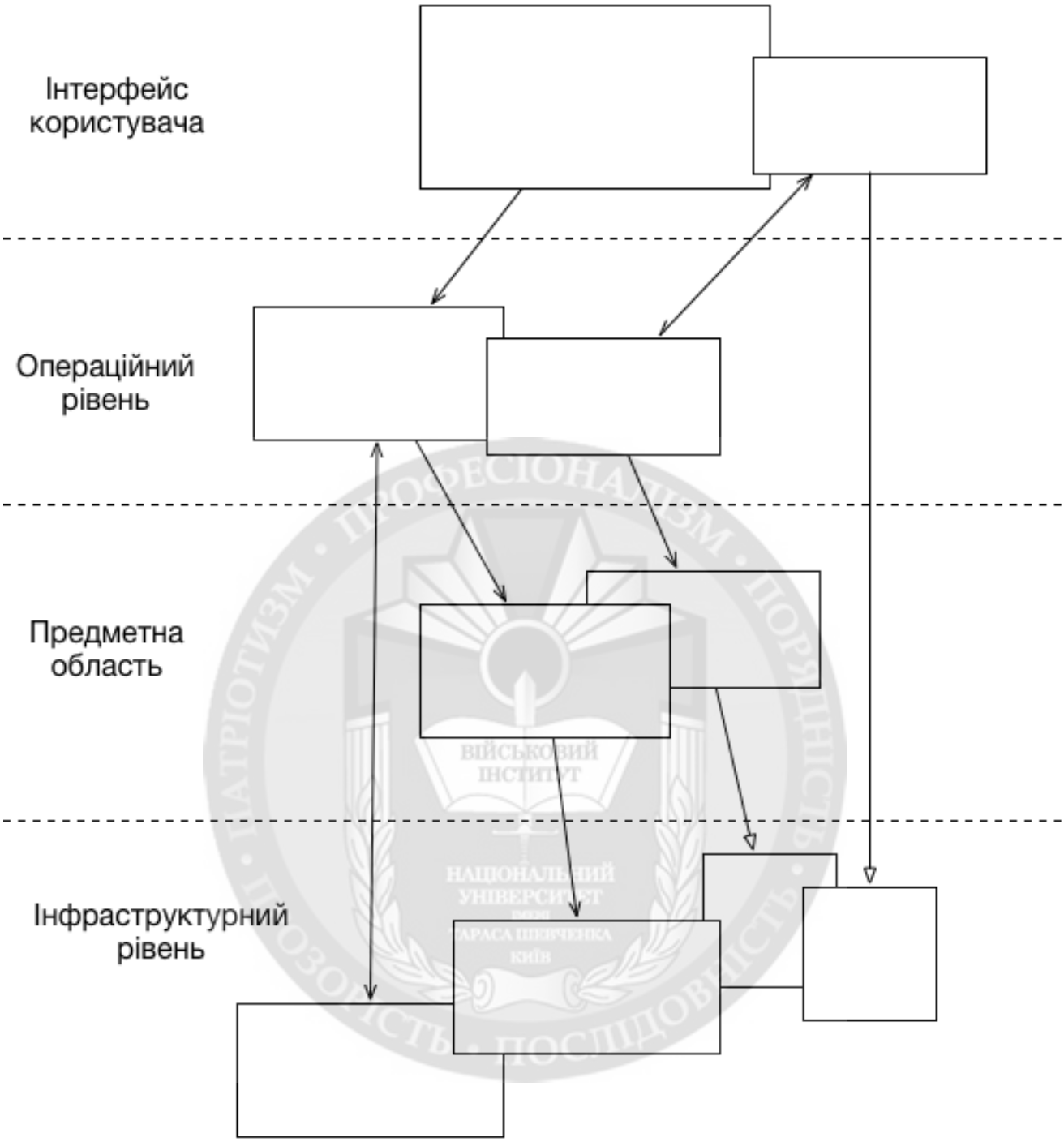

Рис. 5. Структура багаторівневої архітектури

Для зменшення складності розробки програмного забезпечення потрібно умовно розділити архітектуру на рівні. Всередині кожного рівня потрібно розробити зв'язну структуру, яка буде опиратися тільки на нижні рівні і залежну тільки від них. Для забезпечення зв'язку з верхніми рівнями, потрібно використовувати стандартні архітектурні шаблони [2].

Рівень інтерфейсу користувача відповідає за виведення інформації для користувача і інтерпретування його команд. Окрім людини, зовнішнім діючим суб'єктом може виступати інша комп'ютерна система.

Операційний рівень або рівень прикладних операцій – відповідає за координування дій програми, делегує виконання команд користувача іншим рівням системи.

Рівень предметної області відповідає за представлення понять прикладної предметної галузі, робочі стани та ділові регламенти. Там контролюється і виконується поточний стан прикладної моделі. Проте технічні деталі маніпуляції даними делегується інфраструктурі. Ця частина є головною, алгоритмічною частиною програми.

Інфраструктурний рівень забезпечує безпосередню технічну підтримку для верхніх рівнів системи: передачу повідомлень на операційному рівні, вивід інформації для клієнта, відправку повідомлень, а також брати на себе підтримку існування і взаємодію всіх чотирьох рівнів через архітектуру програми.

Ще однією задачею багаторівневої архітектури є відділення інтерфейсу користувача від прикладних операції та предметного рівня. Один із найпоширеніших підходів є MVC підхід [11]. У цій схемі розділення логіки, головними є три компоненти:

- контролер (інтерпретує дії користувача, оповіщає модель про зміну стану),
- модель (представляє данні, реагує на команди контролера, змінює свій стан),
- представлення (відображає данні для користувача, реагує на зміни у моделі).

Рис. 6. Схема взаємодіє компонентів у MVC, як приклад реалізації багаторівневого принципу побудови програмної системи

Зазвичай не доводиться мати діло із побудовує багаторівневої системи, замість цього використовують архітектурні середовища або фреймворки. Фрейморки зазвичай представляють свою архітектуру, уже реалізоване розбиття системи на рівні, підходи до реалізації і відображення інтерфесу користувача. Також вони надають способи для розширення цієї системи і значно прискорюють розробку програмного забезпечення, для якого вони зроблені.

Для пошуку компромісів в програмній реалізації і при цьому не втратити переваги предметно-орієнтованого проектування потрібно зробити детальну декомпозицію предметної моделі. Звязок між моделлю та реалізацією потрібно проробляти на рівні деталів. Для цього потрібно сконцетруватися на окремих елементах моделі та надати їм форми. Існує три шаблонних елементи за допомогою, яких можна виразити предметну модель [4]:

- сутності,
- дата об'єкти,
- сервіси.

Між сутностями та дата-об'єктами можуть існувати асоціації. Для будь-якої явної асоціації в предметній моделі, повинен існувати механізм в системі з тими же властивостями. Асоціації легко уявити і відображаються графічно, проте їх реалізація ускладнює програмну систему. Більшість асоціацій мають двонапраленний характер. Проте існують декілька правил, які дозволять оптимізувати наявні асоціації в моделі:

- звести асоціації до одноправленого виду,
- додати класифікаторів для зниження кратності зв'язку,
- прибрати непотрібні асоціації.

Важливо обмежити взаємозв'язки до мінімуму [6]. Двонаправлені асоціації означають, що два об'єкта можна розуміти тільки як єдине ціле. Якщо технічне завдання не потребує підтримки симетричних асоціації, то її краще звести до асиметричної. До такого ж висновку можна дійти вивчаючи предметну модель та поглиблюючи її.

Для деяких об'єктів предметної моделі значення полів – є не визначальними. Вони представляють собою індивідуально існуючі логічні одиниці. Такі об'єкти прийнято називати сутностями. Для них важливо визначити спосіб порівняння. Зазвичай він зводиться до присвоєння кожному об'єкту унікального ідентифікатора в рамках сутності, потім такий ідентифікатор допомагає зрозуміти чи два об'єкта сутності є однаковими. Часто принцип визначення однакових сутностей можна знайти в предметній області.

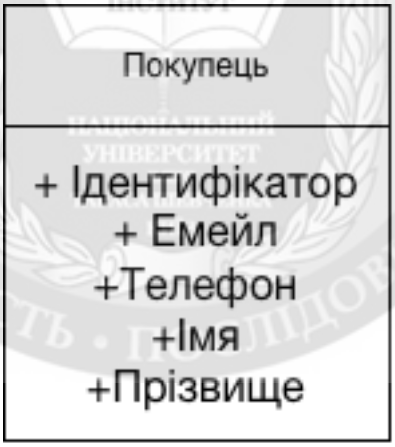

Рис. 7. Приклад зображення сутності

У наведенному прикладі, в залежності від предметної області, ідентифікаторами для сутності "Покупець» могли стати поля Емейл або Телефон, проте універсальним рішенням буде вибір поля Ідентифікатор в якості атрибуту для порівняння.

Окрім сутностей, предметну область можна описати дата об'єктами. Дата об'єктами називаються об'єкти, який представляє описовий аспект предметної області і не має індивідуального існування [7]. Такі об'єкти використовуються коли достатньо знати тільки, що вони собою представляють, а не ким вони являються. Часто дата об'єкти виступають інтерфейсами, через які система взаємодіє з даними.

Сутності та дата об'єкти можуть бути задіяні в важливих процесах чи перетвореннях в предметній галузі. Якщо ці дії не є природньою частиною попередніх елементів, то його можна

виразити через власний інтерфейс, створивши клас, який буде визначений через інші елементи предметної моделі. Такі класи називають сервісами [5]. Сервіси не мають власного стану. Вони можуть зустрічатись у всім трьох рівнях системи:

- Операційні служби (приймає вхідні дані, виконує сервіси із предметної області),
- Служби предметної області (проводить якісь операції із іншими елементами моделі),
- Інфраструктурні служби (відправка емейла).

Сервіси інкасують всю ту логіку, додавання якої безпосередньо в методи сутностей або дата об'єктів призведе до зайвих залежностей у цих класах.

Сервіси, сутності та дата об'єкти – це основні елементи предметної моделі. Проте існую інші допоміжні класи, які відповідають за типові задачі. Наприклад, створення екземпляра класу – це не завжди простий процес. Часто він потребує додаткової логіки та дотримання інваріантів. Зазвичай, таку логіку і відповідальність делегують окремому класу, котрий називають фабрикою [1][2].

Важливою і типовою задачею для програмних систем є збереження та відтворення даних із бази даних, якою може виступати будь-яке сховище. В таких випадках, роботу делегують окремому класу, який бере на себе відповідальність за маніпуляції з даними (відтворення, запис, видалення, пошук за заданими критеріями). Такі класи прийнято називати репозиторіями. Вони являють собою ті інтерфейси, за якими ховається інформація про джерело бази даних та способи роботи із нею.

Далі за допомогою розглянутих елементів предметної моделі та шаблонів проектування реалізовують програмну архітектуру. Сам процес розробки програмної системи згідно із предметно-орієнтованим проектуванням має ітераційний характер. Нові зміни в систему починають вносити через удосконалення та поглиблення її предметної моделі. Через двосторонній зв'язок її із єдиною мовою та програмною архітектурою, ці зміни знайдуть своє місце у них також. Такий процес називається поглибленням моделі або поглибленим рефакторінгом [9].

Основні причини використовувати предметно-орієнтований дизайн в проекті:

- Зменшити складність проектного рішення,
- Зберегти прив'язку до предметної галузі,
- Отримати систему, для якої буде просто писати тести,
- Зменшити поріг входу в систему для нових людей в команді,
- Підвищити підтримку коду

Висновки. Складність предметної галузі напряму визначає складність та швидкість розробки проекту. Проте якою би складною не була сфера знань, клієнти завжди будуть очікувати систему, яка виконує поставлену задачу та готова до розширення за адекватне співвідношення ціни та часу. Розробка таких систем вимагає від розробників використання певних практик в плані побудови архітектури програми. Одним із таких методів є предметно-орієнтовне проектування, в якому предметна модель – це головний спосіб опанувати складність предметної галузі та представити її в зрозумілому вигляді. Для цього використовуються деякі практики та постійне удосконалення попередніх моделей.

REFERENCES:

1. Erich Gamma Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides – 1st Edition – Addison-Wesley Profession, 1994, 395 p.

2. Martin Fowler, Patterns of Enterprise Application Architecture / Martin Fowler – 1st Edition - Addison-Wesley Profession, 2012, 558 p.

3. Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans – 1st Edition - Addison-Wesley Professional, 2003, 563 p.

4. Vaughn Vernon, Implementing Domain-Driven Design / Vaughn Vernon – 1st Edition – Addison-Wesley Profession, 2013, 656 p.

5. Vaughn Vernon, Domain-Driven Design Distilled / Vaughn Vernon – 1st Edition – Addison-Wesley Profession, 2016, 176 p.

6. Scott Millett, Patterns, Principles, and Practices of Domain-Driven Design / Scott Millett – 1st Edition – Wrox, 2015, 792 p.

7. Jimmy Nilsson, Applying Domain-Driven Design and Patterns: With Examples in C# and .NET / Jimmy Nilsson - 1st Edition – Addison-Wesley Profession, 2006, 576 p.

8. Robert C. Martin, “Getting a SOLID start.” / Robert C. Martin - objectmentor.com, 2013.

9. Abel Avram, Doman-Driven Design Quickly / Abel Avram, Floyd Marinescu – 1st Edition - InfoQ.com, 2006, 100 p.

10. Eric Evans, Domain-Driven Design Reference: Definitions and Pattern Summaries / Eric Evans – 1 st Edition - Dog Ear Publishing, 2014, 88 p.

11. Jon Galloway, Professional ASP.NET MVC 5 / Jon Galloway – 1 st Edition – Wrox, 2006, 654p.

Рецензент: д.т.н., проф. Лєнков С.В., головний науковий співробітник науково-дослідного центру Військового інституту Київського національного університету імені Тараса Шевченка

к.т.н. Муляр И.В. , к.т.н., доц. Браун В.О., к.т.н., доц. Шваб В.К.,
Проценко Я.Н., Глушко Р.М.

ПРЕДМЕТНО-ОРИЕНТИРОВАНЫЙ ВЗГЛЯД НА РАЗРАБОТКУ ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ

В этой статье рассматриваться теоретические основы та преимущества предметно-ориентированного подхода к проектированию и реализации программного обеспечения.

Обычно разработка программного обеспечения осуществляться для решения проблеми какой-то предметной области или автоматизации процессов, которые там происходят и важные для конечного пользователя. Не смотря на это, сам процесс разработки - это задача не тривиальная и его сложность напрямую зависит от предметной области, с которой нужно будет иметь дело. В большинстве случаев, сложность предметной области - трудно избежать, по тому что она рождена самой сущностью предмета. Поэтому, приходиться осваивать эту проблему или процесс.

Сегодня существует много подходов к проектированию программного обеспечения (например, экстремальное программирование), которые обращаться внимание и базируются на разных аспектах жизненного цикла программного продукта или проекта. Все они так или иначе имеют дело с предметной областью, но не один из них не ориентируется на неё. Процесс проектирования предметной модели у них отделен от её реализации, обращаться особое внимание на построения программной архитектуры, а предметная модель - существует в качестве дополнения к документации. Предметно-ориентированное проектирование показывает, что такие подходы - неэффективные, это особенно заметно на больших проектах из сложно предметной областью. В нем главной частью разработки программного обеспечения есть создание и усовершенствование предметной модели, на основании которой все сложности предметной сферы пытаются объяснить и изложить.

Построение предметной модели и её итерационное усовершенствование в процессе развития и усовершенствования программного продукта дает возможность построить понятную и ясную архитектуру программой системы, а также организовать эффективное общение внутри команды, которая занимается разработкой т специалистами в предметной области, которые не имеют достаточного понимания программного кода. Такой подход позволяет создавать программные продукты практически любой сложности, а также станет возможным добавление нового функционала и поддержка уже существующего.

Ключевые слова: разработка програмного обеспечения, моделирование, предментая область, многоуровневая архитектура, предметно-ориентированое проектирование, коммуникация, документирование кода.

Ph.D. Mulyar I.V., Ph.D. Braun V.O., Ph.D. Shvab V.K., Protsenko Ya.N., Hlushko R.M.

DOMAIN-DRIVEN VIEW ON SOFTWARE DEVELOPMENT

In the article there is shown the theoretical basis and advantages of using domain-driven design in developing the software program.

Developing the software program is usually used to fix issues in some domain or automate some processes which take place in it and are important for final users. Though the developing process is not a trivial task itself and its complexity is depended from the domain which is required to learn. As a rule, complexity of domain is hard to avoid because it comes from nature of the domain. That's why it has to be reduced somehow.

Nowadays, there are a lot approaches of software program design (e.g. extreme programming) which take into account a different points of lifecycle of software product or project. All of them interacts with the domain model in different ways, but none of them is focused on it. The process of designing the domain model is separated from actual implementing in them. The domain-driven design shows that such approaches are no efficient, especially it is significantly on a huge projects with a very complex domain. In the domain-driven design, the main part of developing software program is creating and improving the domain model, based on it all complex stuff of the domain is tried to explain and decouple.

Creating of the domain model and its improving are allowed to create clear and straight architecture of software system on a way of evolution and developing the software project. Also, it helps to configure effective communication inside the developer team which develops and communicates with people are into the domain and they don't have a required skills to understand the source code. Such approach is allowing to develop projects of any complexity and extending new features or supporting of existing ones.

Keywords: developing of software, modeling, domain, multilayer architecture, domain-driven architecture, writing documentation