ЗМІСТ

ВСТУП 3

1 СУЧАСНИЙ СТАН ПРОБЛЕМИ ТА ОСНОВНІ ЗАДАЧІ РОБОТИ . . 5

1.1 Огляд існуючих практик розробки програмного забезпечення . . 5

1.1.1 Керована тестами розробка 5

1.1.2 Керована поведінкою розробка 6

1.1.3 Неперервна інтеграція 7

1.2 Концепція предметно-орієнтованого проектування 9

1.3 Висновки 11

ПЕРЕЛІК ДЖЕРЕЛ 12

ВСТУП

Актуальність. При розробці програмного забезпечення зустрічаєшся з різними труднощами. Основна перешкода - це предметна область, в якій існує дана проблема. Будь-яке програмне забезпечення має застосування в тій чи іншій сфері діяльності або області інтересів. Наведемо приклад, найбільш пов'язаний з реальним світом: онлайн покупка квитків на літак. Тобто предметна область - це така галузь знань, в якій користувач використовує ПЗ.

Все частіше зустрічаються команди розробників, що займаються проектуванням ПЗ по моделі предметної області. Предметно орієнтоване проектування це підхід до розробки програмного забезпечення, який зосереджує розробку на програмуванні моделі предметної області, що відображає бізнес-логіку [3].

Для реалізації ПЗ з предметної області, яка невідома розробнику, йому необхідно заповнити недолік, але не читаючи багато сторінкові і малозрозумілі книги, наукові статті, оскільки це дасть розпливчасте уявлення. Інструментом для подолання цих труднощів є модель, яка будується з навмисно спрощених і строго відібраних знань. Якщо модель дозволяє зосередитися на проблемі, то вона побудована правильно [2].

Актуальним є створення веб-додатку за використанням практик предметно орієнтованого проектування, що дозволить розробникам краще розуміти бізнес модель проекту через використання єдиної мови та термінів з експертами предметної області.

Метою роботи є дослідження методик предметно орієнтованого проектування на прикладі розробки веб-додатків.

Для досягнення мети необхідно розв'язати наступні задачі:

а) Провести аналіз існуючих практик розробки програмного забезпечення.
б) Розробити багаторівневу архітектуру, для підвищення надійності, полегшення розробки нових модулів, та кращої тестувальності системи.
в) Розробити модель домену вибраної предметної області

г) Розробити бізнес-правила додатку, що реалізують усі випадки використання системи.

д) Розробити адаптери, які перетворюють дані з формату, найбільш зручного для випадків використання та сутностей, у формат, найбільш зручний для якоїсь зовнішньої служби.

е) Реалізувати сервіси для взаємодії з зовнішніми службами.

Об'єктом дослідження є процес розробки програмного забезпечення з використанням практик предметно орієнтованого проектування.

Предметом дослідження є методи та засоби створення програмного забезпечення з використанням предметно орієнтованого проектування.

Методи дослідження. У роботі використовуються методи дослідження, а саме аналіз, моделювання, класифікація, узагальнення, спостереження, прогнозування та експерименту; методи передачі даних та методі представлення результату.

У роботі застосовуються поняття багаторівневої архітектури та предметно орієнтованого проектування.

Апробація результатів та публікації. За результатами даної роботи опубліковано доповідь [1] на науково-технічній конференції Вінницького національного технічного університету, факультету комп'ютерних систем і автоматики (м.Вінниця 2021).

1 СУЧАСНИЙ СТАН ПРОБЛЕМИ ТА ОСНОВНІ ЗАДАЧІ РОБОТИ

1.1 Огляд існуючих практик розробки програмного забезпечення

Коли команда розробників програмного забезпечення залучається до проекту, його основною метою є постачання якісного продукту. Якість означає повне дотримання вимог, відсутність помилок, високий рівень безпеки та здатність витримувати великі навантаження [4].

Додаток або веб-сайт також повинні додавати цінності клієнтам, працювати за призначенням та забезпечувати інтуїтивно зрозумілий інтерфейс, щоб ним можна було користуватися навіть не думаючи. Не дивлячись на те, що все це досить складно, для спрощення та вдосконалення розробки веб-додатків з'явилися найкращі практики розробки програмного забезпечення.

1.1.1 Керована тестами розробка

Керована тестами розробка (Test-driven development, TDD) - це підхід до розробки програмного забезпечення, в якому розробляються тестові кейси, які визначають необхідні покращення або нові функції. Якщо говорити простими словами, спочатку створюються і перевіряються тестові кейси для кожної функціональності, а якщо пройти тест не вдається, то для проходження тесту пишеться новий код [5].

Основними перевагами керованою тестами розробки є:

- Поліпшення якості шляхом виправлення помилок якнайшвидше під час розробки.
- Значне підвищення якості коду.
- Покращення розуміння коду оскільки рефакторинг вимагає регулярного вдосконалення.
- Покращення швидкості розробки, оскільки розробникам не потрібно витрачати час на відлагодження програми.

Принцип роботи керованою тестами розробки зображений на рис. 1.1.

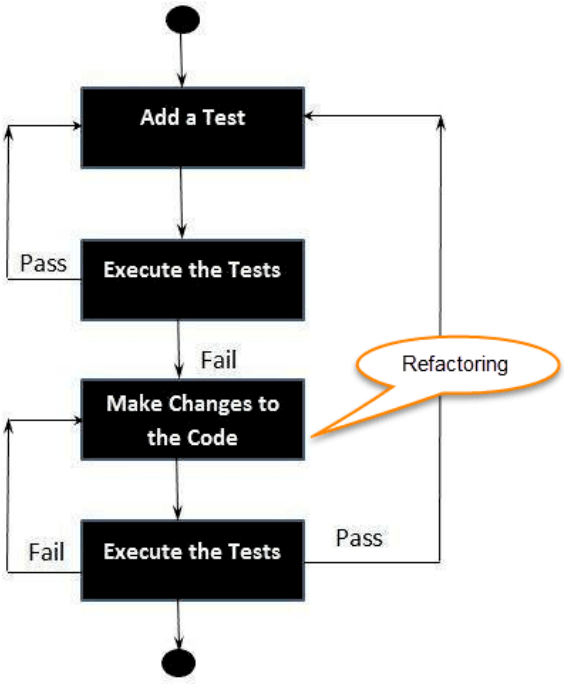

Рис. 1.1 – Цикл керованою тестами розробки

Відповідно до досліджень недоліками використання підходу є:

– Неможливість гарантувати відсутність помилок у програмі, навіть за наявності широкого спектру тестових кейсів.
– Велика витрата часу на розробку тестових кейсів та підтримку належних наборів тестів [6].

1.1.2 Керована поведінкою розробка

Керована поведінкою розробка (Behavior-driven development, BDD) - це синтез та вдосконалення практик, що випливають з керованої тестами розробки (TDD) та керованою тестами розробки прийняття (Acceptance test–driven development, ATDD) [7]. BDD доповнює TDD та ATDD за допомогою наступних технік:

– Мислення «ззовні всередину», іншими словами, застосовувати лише ті способи поведінки, які найбільше сприяють цим результатам бізнесу, щоб мінімізувати витрати.

– Описування поведінки в одній нотації, яка є безпосередньо доступною для експертів області, тестувальників та розробників, з метою покращення комунікації.
– Застосування цих методів аж до найнижчих рівнів абстрагування програмного забезпечення, приділяючи особливу увагу розподілу поведінки, щоб прогресування залишалося дешевим.

Команди, які вже використовують TDD або ATDD, можуть захотіти розглянути BDD саме з таких причин:

– BDD пропонує більш точні вказівки щодо організації бесіди між розробниками, тестувальниками та експертами предметної області.
– Інструменти, орієнтовані на підхід BDD, як правило, дозволяють автоматично створювати технічну документацію та документацію для кінцевих користувачів із “специфікацій” BDD.

Недоліком даного підходу є необхідність представити команду розробників для роботи з клієнтом. Короткий час реакції, необхідний для процесу, означає високий рівень доступності. Однак, якщо клієнт добре розуміє, що задіяно у проекті розробки, заснованому на принципах Agile, експерт-клієнт буде доступний у разі потреби. І якщо команди розробників працюють максимально ефективно, їх вимоги до експерта-клієнта будуть мінімізовані [8].

1.1.3 Неперервна інтеграція

Неперервна інтеграція (Continuous Integration, CI) - це практика розробки програмного забезпечення, при якій зміни кодової бази є інтегрованими в сховища потоків після побудови та перевірки за допомогою автоматизованого робочого процесу [9]. Принцип роботи неперервної інтеграції зображений на рис. 1.2.

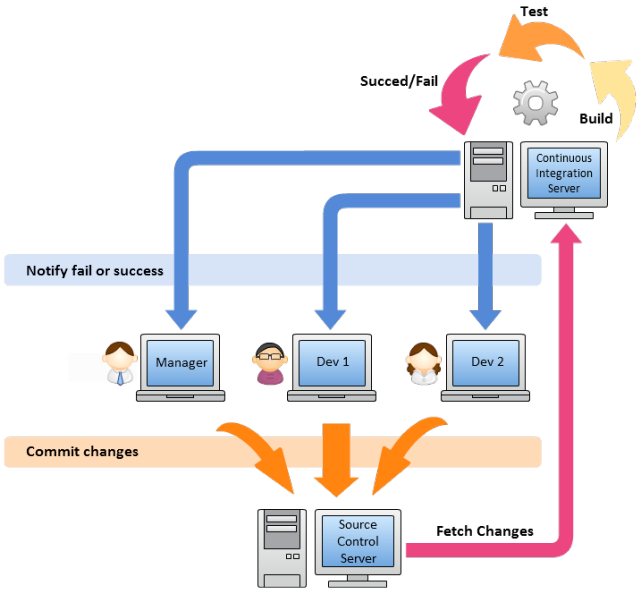

Рис. 1.2 – Цикл роботи неперервної інтеграції

Основні переваги використання неперервної інтеграції:

– Середній час до роздільної здатності (Mean time to resolution, MTTR) швидший і коротший.
– Ізоляція несправностей менша і швидша.
– Підвищений рівень випуску допомагає швидше виявляти та виправляти несправності.
– Автоматизація в CI зменшує кількість помилок, які можуть виникнути на багатьох етапах.

Недоліки використання неперервної інтеграції:

– Кодова база повинна бути готова і негайно впроваджена у виробництво, як тільки поточний результат буде успішним.
– Підхід вимагає суворої дисципліни з боку учасників. Невдачі у дотриманні процесів незмінно породжуватимуть помилки, витрачаючи час і гроші.
– Деякі галузеві середовища не підходять для неперервної інтеграції. Медична сфера та авіація вимагають багато випробувань, щоб включити код у загальну систему [10].

1.2 Концепція предметно-орієнтованого проектування

Предметно-орієнтоване проектування (Domain-driven design, DDD) - це підхід до розробки програмного забезпечення, який зосереджує розробку на програмуванні моделі предметної області, що має глибоке розуміння процесів та правил домену. Назва походить від книги Еріка Еванса 2003 року, яка описує підхід через каталог шаблонів [2]. З тих пір спільнота практиків розвивала ідеї, породжуючи різні інші книги та навчальні курси. Підхід особливо підходить для складних доменів, де потрібно організувати багато часто безладної логіки.

У книзі Domain-Driven Design [2], сформульований ряд концепцій і практик. Так, наприклад, особлива увага приділяється значенню загальної мови (Ubiquitous language). При проектуванні моделі предметної області необхідно сформувати спільну мову предметної області для опису вимог до системи, яка працює однаково добре як для бізнес-користувачів або спонсорів, так і для розробників програмного забезпечення. Ця мова означується експертами в обраній галузі. Книга зосереджена на описі доменного рівня, як одного із загальних рівнів в об'єктно-орієнтованій системі з багаторівневою архітектурою. У DDD є засоби для висловлення, створення та вилучення моделей предметної області:

- *Сутність:* Категорія індивідуальних об'єктів, які залишаються незмінними на різних етапах програми, для яких атрибути не грають великого значення, а послідовність та ідентичність, які поширюється в житті усієї системи називаються сутностями.
- *Об'єкт значення:* Об'єкт, який містить атрибути, але не має концептуальної ідентичності. Він повинен розглядатися як незмінний об'єкт.
- *Сукупність:* Колекція об'єктів, які пов'язані між собою завдяки головній сутності (Root Entity), інакше відомій як Aggregate root. Коренева сутність колекції об'єктів гарантує узгодженість змін, що вносяться до сукупності, забороняючи зовнішнім об'єктам посилатися на членів колекції.

– *Доменна подія:* Подія, яка сталася в певному домені, і фіксує пам'ять про нього. Це дає можливість повідомляти інші частини того самого домену про те, що сталося якась зміна, і ці інші частини потенційно можуть реагувати [11].
– *Сервіс:* Коли будь-яка операція концептуально не відноситься до будь-якого об'єкту, вона може бути реалізована в сервісі.
– *Сховище:* Отримання об'єктів предметної області повинно делегуватися в спеціалізовані сховища об'єктів. Це дає можливість підміняти місце збереження об'єктів.
– *Фабрика:* Створення об'єктів предметної області повинно бути делеговане до спеціалізованих фабрик. Це дає можливість підміняти реалізацію створення об'єктів.

Для того, щоб добре використовувати предметно-орієнтоване проектування, нам потрібно прийняти до уваги принципи SOLID [13], організувати рівень бізнес логіки в основі нашої багаторівневої архітектури, використовувати багато інверсії та впровадження залежностей, щоб підключити адаптери до рівня персистентності, веб та зовнішніх технологій.

Переваги використання предметно-орієнтованого проектування:

– Загальнодоступна мова, полегшує спілкування між розробниками та представниками бізнесу, а також між самими розробниками.
– Оскільки система була побудована для моделювання ділового домену, її, як правило, буде гнучкіше змінювати, оскільки нові функціональні вимоги цілком вписуються.
– А також завдяки способу побудови моделей доменів з використанням інкапсуляції зрозумілості та хорошого розподілу моделей, доменні системи, як правило, є більш підтримуваною за своєю природою [12].

Для того, щоб допомогти зберегти модель як чисту та корисну конструкцію мови, команда, як правило, повинна здійснити велику кількість ізоляції та інкапсуляції в рамках доменної моделі. Отже, система, що базується на предметно-орієнтованому дизайні, може мати відносно високу вартість. Хоча предметно-орієнтоване проектування забезпечує багато технічних переваг,

Microsoft рекомендує застосовувати його лише до складних доменів, де модель та лінгвістичні процеси дають чіткі переваги при передачі складної інформації та формулюванні загального розуміння домену.

1.3 Висновки

В першому розділі розглянуто основні аспекти роботи: проведено огляд та аналіз існуючих практик розробки програмного забезпечення, представлено основні поняття по темі, наведено переваги використання предметно-орієнтованого проектування.

ПЕРЕЛІК ДЖЕРЕЛ

1. Московко С.Г. Аналіз архітектурних шаблонів програмного забезпечення / С.Г. Московко, І.В. Богач. // Науково-технічна конференція факультету комп'ютерних систем і автоматики ВНТУ, 2021. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2021/paper/view/11903/10500

2. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans., 2003. - 320 с. - ISBN 9780321125217.

3. Domain Driven Design [Електронний ресурс]. Електронні дані. URL: https://martinfowler.com/bliki/DomainDrivenDesign.html

4. How to build quality software solutions using TDD and BDD? [Електронний ресурс]. Електронні дані. URL: https://y-sbm.com/blog/difference-between-tdd-and-bdd

5. What is Test Driven Development (TDD)? [Електронний ресурс]. Електронні дані. URL: https://www.guru99.com/test-driven-development.html

6. Khanam Z. Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls / Z. Khanam, M. Ahsan. // International Journal of Applied Engineering Research., 2017. – с. 7705–7716.

7. Behavior Driven Development (BDD) [Електронний ресурс]. Електронні дані. URL: https://www.agilealliance.org/glossary/bdd/

8. Behavior Driven Development: Alternative to the Waterfall Approach [Електронний ресурс]. Електронні дані. URL: https://www.agilest.org/devops/behavior-driven-development/

9. Duvall P.M. Continuous Integration: Improving Software Quality and Reducing Risk / P. M. Duvall, A. Glover, S. Matyas., 2007. – 336 с. - ISBN 9780321336385.

10. Continuous Integration [Електронний ресурс]. Електронні дані. URL: https://www.agilest.org/devops/continuous-integration/

11. How to publish and handle Domain Events [Електронний ресурс]. Електронні дані. URL: http://www.kamilgrzybek.com/design/how-to-publish-and-handle-domain-events/

12. What is Domain-Driven Design (DDD) | Pros & Cons [Електронний ресурс]. Електронні дані. URL: https://codezup.com/what-is-domain-driven-design-ddd-pros-cons/

13. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. Martin., 2017. - 432 с. - ISBN 9780134494166.