РОЗРОБКА ЧЕРЕЗ ТЕСТУВАННЯ (TDD) ЯК ПРАКТИКА РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Васюта Василь Васильович
к.т.н., доцент
Барсуков Станіслав Георгійович
студент
Національний університет «Полтавська політехніка
імені Юрія Кондратюка»
м. Полтава, Україна

Вступ. /Introductions. Розробка через тестування – це процес, у якому ви пишете тест перед написанням коду. І коли всі тести пройдені, ви покращуєте свій код. Тим не менш, розробка через тестування – це не тестування [1].

Ідея розробки, що базується на тестуванні, полягає в тому, що весь код слід постійно тестувати та реорганізовувати.

Мета роботи. /Alim. Розглянути сутність розробки через тестування (TDD) як практики розробки програмного забезпечення.

Матеріали та методи. /Materials and methods. Для досягнення поставленої мети у роботі використані методи теоретичного узагальнення та аналізу.

Результати та обговорення. /Results and discussion. Тести, які пишуться в TDD, не суть, а засіб. Справа в тому, що, написавши спочатку тести і прагнучи спростити їх написання, робиться три важливі речі:

- створюється документація, що жива, ніколи не застаріває специфікації;
- розробляється свій код, щоб зробити його тестованим. І це робить його чистим, нескладним, легким для розуміння та зміни;
- створюється сітка страхування, щоб впевнено вносити зміни.

На додаток до переваг, згаданих вище, TDD також надає:

- раннє повідомлення про помилку;
- просту діагностики помилок, оскільки тести точно визначають, що

не так [2].

Все це означає для бізнесу:

- покращення «фактору автобуса», оскільки знання знаходяться не тільки в головах, і це полегшує прийом нових співробітників;
- зниження вартості покращень. Збереження коду у чистоті – це ще й спосіб мінімізувати ризик випадкових ускладнень. А це означає, що можна підтримувати постійний темп надання цінності;
- завдяки мережі безпеки розробники з більшою готовністю поєднують свої зміни та втягують зміни інших розробників. І вони робитимуть це частіше. І тоді справді може початися розробка на основі магістралі та безперервна інтеграція, доставка та розгортання;
- знижує кількість помилок, які «йдуть» у виробниче середовище, та знижує витрати на підтримку.

Розробка через тестування – це основна практика Agile. Вона безпосередньо підтверджує цінність Agile «Робоче програмне забезпечення, порівняно з вичерпною документацією». І робить це, захищаючи працююче програмне забезпечення за допомогою тестів та створюючи документацію як природний побічний продукт [3].

Розробка через тестування передбачає проходження трьох етапів (рис.1). Знову і знову та знову:

- Червона фаза: напишіть тест;
- Зелена фаза: завершіть тест, написавши код, який він охороняє;
- Синя фаза: рефакторинг.

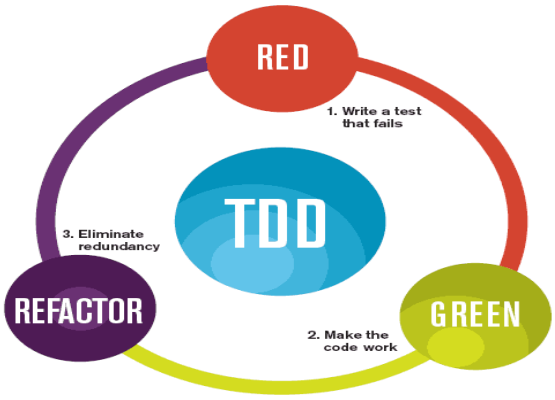

Рис.1. Етапи TDD

Роберт С. Мартін виклав правила TDD у розділі 5 "Розробка через тестування" своєї книги " The Clean Coder ":

- Вам не дозволяється писати виробничий код, якщо він не призначений для проходження невдалого проходження модульного тесту;
- Вам не дозволяється писати більше модульних тестів, ніж достатньо для відмови;
- Вам не дозволяється писати більше коду виробництва, ніж достатньо для проходження одного невдалого модульного тесту [4].

Ціль правил – зосередити увагу на кожній фазі. На практиці, це дуже допомагає тримати речі прямо в голові. Так, під час червоної фази (написання тесту) працюйте лише з тестовим кодом. Невдалий тест – це добре. Принаймні якщо це той, над яким ви працюєте. Всі інші мають бути зеленими. Під час зеленої фази (проходження тесту) працюйте лише з виробничим кодом, який пройде тест і не проводитиме жодних рефакторингів. Якщо тест, який ви щойно написали, зазнає невдачі, це означає, що ваша реалізація потребує доопрацювання. Якщо інші випробування зазнають невдачі, ви порушили існуючу функціональність і повинні повернутися. Під час синьої фази (рефакторингу) рефакторинг лише виробничого коду та не вносіть жодних функціональних змін. Будь-який невдалий тест означає, що ви порушили функціональність. Ви або не завершили рефакторинг, або вам потрібно відмовитись від нього.

Коли проводиться рефакторинг, це не роблять безсистемно. Але дотримуються структурованого процесу очищення коду. Визначається «запах коду» та застосовується певний рефакторинг для його усунення [5].

Мета рефакторингу – покращити розширюваність коду за рахунок покращення читаності, спрощуючи внесення змін, зниження складності, покращення внутрішньої архітектури (об'єктну модель) та зробити її більш виразною. Що дійсно важливо, так це те, що рефакторинг – це те, що дає чистий, нескладний код, який легко зрозуміти, змінити та протестувати.

Практика TDD – це не легко. Принаймні спочатку. Це тому що потрібно

подумати про те, чого ви хочете, щоб ваш код виконував і як захистити його від злому; бо має круту криву навчання. Необхідно вивчити принципи та шаблони проектування для створення чистого коду та дізнатися, як провести рефакторинг, щоб він залишався таким. Код чинить опір. Існуючий код, який не тестується, ловить вас між молотом і ковадлом. Вам потрібно його реорганізувати, щоб протестувати, і вам потрібні тести для його рефакторингу. Ви можете зазнати невдачі в практиці TDD з багатьох причин, а саме:

- Не дотримуючись принципу «спочатку тест».
- Не займатись рефакторингом постійно.
- Написання більш ніж одного тесту за раз.
- Не проводьте тести часто, втрачаючи ранні відгуки від них.
- Написання повільних тестів. Весь набір повинен бути завершений за хвилини або секунди.
- Використання модульних тестів для інтеграційного тестування. Немає нічого поганого у використанні середовища модульного тестування для запуску інтеграційних тестів. Але інтеграційні тести за своєю природою повільні, тому вам потрібно помістити їх в окремий набір тестів.
- Написання тестів без тверджень.
- Написання тестів для тривіального коду, такого як засоби доступу та подання без логіки.

Висновки. /Conclusions. Розробка керована поведінкою, і розробка керована тестуванням, схожі й водночас різняться. Обидва використовують підходи, орієнтовані спочатку тестування, але не стосуються тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Що таке TDD [Електронний ресурс]. URL: https://freehost.com.ua/faq/articles/chto–takoe–tdd–test–driven–development/.

2. Shore J. The Art of Agile Development / J. Shore, S. Warden., 2021. 540c. (2nd Edition).

3. Bender J. Professional Test Driven Development / James Bender.,

2011. 536 с.

4. Martin R. C. The Clean Coder / Robert C. Martin., 2011. 256 с. (1st edition).

5. Martin F. Refactoring: Improving the Design of Existing Code / Fowler Martin., 2018. 448 с. (2nd edition).

6. Лагунова І. А. Сутність та принципи концепції ризик-менеджменту. Актуальні

7. Проблеми держав- ного управління. 2018. № 1 (53). С. 44–52.

8. Лагунова І. А. Сутність та принципи концепції ризик-менеджменту. Актуальні

9. проблеми держав- ного управління. 2018. № 1 (53). С. 44–52.