TDD (Test-Driven Development)

Michał Korwel

Czym jest TDD?

TDD (Test-Driven Development) jest techniką tworzenia oprogramowania (nie jest to technika pisania testów), w której główną ideą jest w pierwszej kolejności pisanie testów do nieistniejącej funkcjonalności, a dopiero potem napisanie kodu implementującego tę funkcjonalność. TDD nie jest jedyną słuszną i dobrą techniką tworzenia programowania. Można pisać testy równolegle z pisaniem kodu logiki biznesowej, można je pisać po implementacji, lecz wtedy nie jest to już TDD. W Test-Driven Development testy piszemy zawsze przed implementacją.

Cykle życia TDD

01

Napisanie przypadku testowego

Na tym etapie musimy wiedzieć, co chcemy przetestować i czego oczekujemy od testowanej funkcjonalności. Dzięki temu również możemy sprawdzić, czy dokładnie rozumiemy logikę biznesową, a zapewne zdarzyło Wam się programować „w ciemno”.

02

Uruchomienie napisanego testu

Próbujemy uruchomić nasz napisany test i oczekujemy błędu kompilacji, ponieważ nie ma kodu, który chcemy testować.

03

Napisanie minimalnego kodu do przejścia testu

04

Refaktoryzacja kodu

05

Powtarzamy cykl

Jak długie powinny być cykle TDD?

Jak długie powinny być testy? Jak duży fragment kodu powinny pokrywać?
Jednym z założeń TDD jest pisanie jak najkrótszych cykli, dużej ilości testów, w której każdy test pokrywa mały wycinek kodu. Zauważyłem w tym podejściu wadę – pisanie bardzo dużej ilości testów jednostkowych było dla mnie uciążliwe. Duża ilość małych testów nie jest zła, wręcz przeciwnie, jednak nie można popadać w skrajność.
Moim zdaniem cykle powinny być na tyle duże, żeby można było łatwo „ogarnąć” to, co się dzieje w przypadku testowym i co ważniejsze – żeby inny programista w łatwy sposób mógł je ze zrozumieniem przeczytać.

Kiedy nie należy stosować TDD?

TDD na pewno nie sprawdzi się w przypadku małych, nieskomplikowanych projektów. Wtedy czas poświęcony na praktykowanie tej techniki mógłby się nigdy nie zwrócić.
Często podczas programowania rozwiązanie problemu, nad którym pracujemy, nie jest oczywiste. W takiej sytuacji najpierw staramy się jak najszybciej napisać kod, dzięki któremu będziemy mieli pewność, że potrafimy poprawnie rozwiązać problem. W takim przypadku na początku nie interesuje nas obsługa błędów tego kodu, przypadki graniczne itd. **TDD** podczas takiego procesu będzie tylko nas spowalniał. Oczywiście należy pamiętać, żeby po znalezieniu rozwiązania zadbać o refaktor i późniejsze napisanie testów jednostkowych.
TDD nie należy również stosować, w przypadku, kiedy testy są czasochłonne i nie przynoszą dużych korzyści. Nie testujmy kodu trywialnego, który nie wnosi żadnej logiki biznesowej, a jest jedynie „łącznikiem” pomiędzy modułami czy po prostu jest to „boilerplate code”.

Zalety TDD

Podczas tworzenia oprogramowania w sposób klasyczny (najpierw piszemy większy blok kodu, później go testujemy) często w nasz kod wkrada się „bug”, który zostanie odkryty dopiero po jakimś czasie. Wykrycie błędu może trwać godzinę, kilka dni lub nawet kilka miesięcy, kiedy nasz kod już od dawna funkcjonuje na produkcji. Prawdopodobne jest, że jeśli od czasu wprowadzenia błędu do czasu jego znalezienia powstanie dużo dodatkowego kodu, to czas na naprawienie błędu się wydłuży. Jeśli programujemy zgodnie z TDD, to większość błędów zostanie wykrytych podczas testów przed wprowadzeniem kodu w środowisko testowe lub produkcyjne. **Testy** je wykryją podczas programowania modułu. W takim wypadku możemy sami naprawić błąd i angażujemy mniej osób do jego naprawy (inni developerzy, testerzy, klient). Dzięki temu oszczędzamy czas. Oczywiście nawet stosując TDD, nie unikniemy pisania błędów, ale zminimalizujemy zakres ich występowania.

Podsumowanie

TDD jest ciekawą techniką tworzenia oprogramowania. Warto się jej nauczyć i spróbować ją wykorzystać w naszych projektach po to, żeby przekonać się, czy przynosi nam ona korzyści. Nie należy jej traktować jako „wyrocznię”, jedyną słuszną technikę tworzenia oprogramowania. Ma ona swoje zalety i wady. Musimy pamiętać o tym, żeby dobierać narzędzia oraz techniki do indywidualnych potrzeb projektu.

Zadanie

Zacznijmy od wymagań, które nasz program ma spełniać. Ma to być program, który przechowuje oceny jednego ucznia. Poniższe punkty opisują wymagania:

- Jako nauczyciel chcę dodawać przedmioty do dzienniczka,
- jako nauczyciel chcę dodać ocenę dla jednego z przedmiotów,
- jako nauczyciel chcę policzyć średnią ocen dla danego przedmiotu,
- jako nauczyciel chcę policzyć średnią ocen z wszystkich przedmiotów.

Pisząc ten program użyj dwóch klas **GradeBook** reprezentującej dzienniczek oraz klasy **Subject**, która będzie opisywała przedmiot. Te wymagania podziel na kilka etapów. Staraj się postępować zgodnie z nimi. W każdym z tych etapów możesz mieć kilka cykli, w których dodawał będziesz kolejne testy jednostkowe:

1. Utworzenie dzienniczka,
2. Utworzenie przedmiotu,
3. Dodanie przedmiotu do dzienniczka,
4. Pobranie przedmiotu z dzienniczka,
5. Dodanie oceny do przedmiotu,
6. Obliczenie średniej dla przedmiotu,
7. Obliczenie średniej dla dzienniczka.


THEY DIDN'T USE
TDD