

**Wydział Fizyki i Informatyki Stosowanej**

Praca inżynierska

**Zastosowanie metodyki Test Driven Development do tworzenia aplikacji internetowych opartych na Java Spring**

Test Driven Development of web applications

based on Java Spring

Autor: Hleb Shypula

Kierunek studiów: Informatyka Stosowana

Opiekun pracy: dr hab. inż. Maciej Wołoszyn

Kraków, 2021/2022

1. **Wprowadzenie**

Moją ambicją jest elegancki, elastyczny i zrozumiały kod, który można łatwo modyfikować, który działa poprawnie i nie sprawia przykrych niespodzianek podczas tworzenia oprogramowania. Aby zrealizować ten zamysł, spróbuję przetestować program, zanim go napiszę. To właśnie ta paradoksalna idea stanowi podstawę metodyki Test-Driven Development (programowanie sterowane testami). Czy to jest nielogiczne? Ja nie spieszyłem się z wyciąganiem pochopnych wniosków.

Głównym celem niniejszej pracy było napisanie aplikacji internetowej opartej na Java Spring przy zastosowaniu metodyki Test-Driven Development. A mianowicie:

* Przedstawienie zasadniczych cech metodyki Test Driven Development;
* Zdobywanie osobistego doświadczenia w programowaniu sterowanym testami;
* Sprawdzenie, czy poziom wejścia do tej metodyki jest akceptowalny dla programisty niezbyt doświadczonego w jakimkolwiek testowaniu;
* Przejście po większości wzorców TDD, ich porównanie między sobą oraz zdefiniowanie dla każdego najbardziej pasujących zadań w tworzeniu oprogramowania.

Temat „Zastosowanie metodyki Test Driven Development do tworzenia aplikacji internetowych opartych na Java Spring” ma na celu wskazać czytelnikowi, że w odróżnieniu od XP (Extreme Programming) metodyka TDD nie jest absolutna. XP mówi: „Mamy rzeczy, które powinniśmy opanować, zanim przejdziemy do kolejnego etapu”. Natomiast TDD – to nie tak konkretna metodyka. TDD zakłada, że podczas programowania w pełni wyobrażamy sobie odległość od momentu podjęcia decyzji o designie do etapu kontroli jakości otrzymanych wyników. Także TDD udostępnia techniki umożliwiające kontrolowanie tej odległości[[1]](#footnote-1).

Główną tezą niniejszego opracowania jest stwierdzenie, że dzięki TDD kod będzie zawierał znacznie mniej uszkodzeń i niedoskonałości. Jeżeli tak, to napisanie kodu, którego wymaga test by przejść, produkuje tezę podrzędną – codziennie będziemy w stanie dostarczać w pełni działający produkt wraz z dodaną nową funkcjonalnością. Dzięki temu współpraca z wirtualnym klientem osiągnie nowy poziom.

1. **Wstęp teoretyczny**

**2.1 Definicja TDD**

Programowanie sterowane testami (ang. *test-driven development, TDD*) – technika tworzenia oprogramowania, polegająca na powtórzeniu bardzo krótkich cykli projektowania: najpierw pisze się test, pokrywający żądane zmiany, następnie pisze się kod, którego wymaga test by przejść, i w końcu przeprowadza się refaktoryzacja nowego kodu, żeby spełniał odpowiednie standardy. Kent Beck, uznawany za wynalazcę tej techniki, argumentował w 2003 r., że programowanie oparte na testach zachęca do prostego designu i wzbudza wiarę w siebie (ang. *inspires confidence*)[[2]](#footnote-2).

W 1999 roku, kiedy TDD pojawiło się po raz pierwszy, technologia była ściśle związana z koncepcją test-first, stosowaną w programowaniu ekstremalnym, ale później pokazała się jako niezależna metodyka. TDD różni się od innych metod tym, że łączy programowanie z pisaniem testów przez tego samego programistę. Ta koncepcja odnawia powszechny szacunek dla testów tworzonych przez programistę[[3]](#footnote-3).

Test to procedura, która pozwala potwierdzić lub zdementować, że kod działa. Testy funkcjonalne pozwalają zobaczyć funkcjonalność aplikacji z punktu widzenia użytkownika. Oznacza to, że testy funkcjonalne mogą być rodzajem specyfikacji aplikacji. Można wydzielić: testy czarnej skrzynki, testy w przeglądarce, testy behawioralne, testy wydajnościowe. Natomiast testy jednostkowe testują aplikację od środka, z punktu widzenia programisty. Testowanie wtedy odbywa się w celu upewnienia się, że obiekty (moduły, podsystemy) działają poprawnie.

**2.2 Wymagania TDD**

Test Driven Development wymaga od dewelopera tworzenia automatycznych testów jednostkowych, które definiują wymagania dla kodu jeszcze przed napisaniem rzeczywistego kodu. Test zawiera procedury sprawdzające warunki, które mogą przejść lub nie przejść. Powodzenie testu weryfikuje zachowanie zgodne z zamierzeniami programisty. Deweloperzy często używają frameworków testowych do tworzenia i automatyzacji wykonywania zestawów testów. W praktyce testy jednostkowe obejmują krytyczne i nietrywialne sekcje kodu. Może to być kod podlegający częstym zmianom lub kod z dużą liczbą zależności.

TDD obejmuje nie tylko walidację, ale także wpływa na design programu. Opierając się na testy, programiści mogą szybciej wyobrazić sobie, jakich funkcjonalności potrzebuje użytkownik. Wtedy szczegóły interfejsu pojawiają się długo przed ostatecznym wdrożeniem rozwiązania.

**2.3 Cykl TDD**

**1. Napisanie testu**

W programowaniu opartym na testach wcielenie każdej nowej funkcjonalności do aplikacji zaczyna się od napisania testu. Aby to zrobić, programista powinien z pewnością rozumieć wymagania do nowej funkcjonalności. W tym celu brane są pod uwagę możliwe przypadki użycia i historyjki użytkownika. Nowe wymagania mogą również wymuszać zmiany w istniejących testach. I właśnie ten pierwszy punkt cyklu odróżnia programowanie sterowane testami od technik, w których testy są tworzone po napisaniu kodu.

**2. Uruchomienie wszystkich testów: nowe testy powinny się zakończyć niepowodzeniem**

Właśnie napisany test nieuchronnie skończy się niepowodzeniem, ponieważ odpowiedni kod nie jest jeszcze napisany. Natomiast jeżeli test przeszedł, oznacza to, że rozważana funkcjonalność już istnieje lub napisany test ma wady. Wiedząc to, na tym etapie sprawdza się, czy napisane właśnie testy nie przechodzą. One nie powinny przejść z oczywistych dla dewelopera powodów. Zwiększy to pewność (chociaż nie gwarantuje całkowicie), że test rzeczywiście sprawdza to, do czego został zaprojektowany.

**3. Napisanie kodu**

W tym momencie programista powinien zaprojektować nowy kod, aby wcześniej napisany test przeszedł pomyślnie. Ten kod nie musi być doskonały. Dozwolone jest, aby zdał „egzamin” w jakiś nieelegancki sposób. Jest to akceptowalne, ponieważ kolejne kroki go poprawią i wypolerują. Ważne jest, aby napisać kod specjalnie do przejścia konkretnego testu. Nie należy dodawać zbędnych, a za tym, niepokrytych testami funkcjonalności.

**4. Uruchomienie wszystkich testów: wszystkie testy powinny się zakończyć powodzeniem**

Jeśli wszystkie testy zakończą się powodzeniem, programista może mieć pewność, że kod spełnia wszystkie testowane wymagania. Następnie można przejść do ostatniego etapu cyklu.

**5. Refaktoryzacja**

Po osiągnięciu wymaganej funkcjonalności kod można „posprzątać”. Refaktoryzacja to proces zmiany wewnętrznej struktury programu, który nie wpływa na jego zewnętrzne zachowanie i ma na celu ułatwienie zrozumienia jego działania, wyeliminowanie duplikacji kodu oraz ułatwienie wprowadzania zmian w najbliższej przyszłości[[4]](#footnote-4).

Opisany cykl powtarza się, wdrażając coraz więcej nowych funkcjonalności. Jeśli nowy kod nie spełnia odpowiednich testów lub stare testy przestają przechodzić, programista powinien zwrócić się do debugowania lub po prostu cofnąć się do poprzedniej wersji aplikacji i spróbować podejść do problemu z innej strony. Kroki powinny być małe, pozwoli to na łatwiejsze zidentyfikowanie błędów w przypadku ich wystąpienia, a w przypadku nierozwiązalnych trudności cofnięcie do poprzedniej wersji nie będzie aż tak bolesne.

**2.4 Dlaczego TDD**

Przed rozpoczęciem tworzenia aplikacji, zastanowiłem się, dlaczego chcę użyć TDD, jakie problemy, które spotkałem w swoich poprzednich projektach ta metodyka by pokonała. Podczas przygotowywania się do napisania pracy i czytania tematycznej literatury wymieniłem dla siebie kilka bolączek, które cały czas nie dawały mi spokoju oraz które TDD ma na celu wyleczyć:

* Większość moich testów były manualne;
* Jeżeli miałem testy automatyczne, to nie wykrywały one rzeczywiste problemy;
* Testy automatyczne były pisane i przeprowadzane za późno, żeby wnieść użyteczny wkład w rozwój aplikacji;
* Zawsze miałem coś bardziej ważnego do zrobienia, niż napisanie testów;
* Niemożliwość refaktoryzacji kodu ze względu na strach zepsucia się jakichś funkcjonalności;
* Сiągły lęk przed wdrożeniem aplikacji, ponieważ na pewno coś opuściłem podczas przeprowadzenia swoich biednych testów[[5]](#footnote-5).

Oczywiście, rozumiałem, że TDD nie pokona magicznie wszystkie te problemy, ale miałem nadzieję, że wyciągnę dla siebie niezmierzony postęp na tych poziomach tworzenia oprogramowania.

Także na początku wydzieliłem kilka zasadniczych zalet TDD, i nie mogłem się doczekać, żeby sprawdzić je na własnym przykładzie. Wyszczególnię zwłaszcza kilka z nich:

* Zmniejsza zależność od debugowania – ponieważ najpierw nacisk jest na pisanie testów, a następnie generowanie kodu, aby przejść te testy, wiele programistów uważa, że znacznie zmniejsza się potrzeba debugowania. Ponieważ podczas pisania i kodowania testu wymagane jest głębsze zrozumienie wymagań logicznych i funkcjonalnych, często można szybko zidentyfikować i rozwiązać przyczynę niepowodzenia testu;
* Programowanie sterowane testami oferuje więcej niż tylko walidację kodu, ma również wpływ na projektowanie aplikacji. Skupiając się początkowo na testach, łatwiej wyobrazić sobie, jakiej funkcjonalności potrzebuje użytkownik. W ten sposób programista zastanawia się nad szczegółami interfejsu przed napisaniem kodu. Testy zmuszają nas do pisania kodu bardziej testowalnego. Na przykład, porzucenie zmiennych globalnych, singletonów, sprawia, że klasy są mniej spójne i łatwiejsze w użyciu. Silnie powiązany kod będzie znacznie trudniejszy do przetestowania. Testy jednostkowe przyczyniają się do tworzenia przejrzystych i małych interfejsów. Każda klasa będzie miała określoną (zwykle małą) rolę;
* Testy pozwalają na refaktoryzację kodu bez ryzyka zepsucia go. Wprowadzając zmiany w dobrze przetestowanym kodzie, ryzyko nowych błędów jest znacznie mniejsze. Jeśli nowa funkcjonalność prowadzi do błędów, testy natychmiast to wykryją. Dobrze pokryty testami kod jest łatwy do refaktoryzacji. Programiści będą mieli dużo większą swobodę, jeśli chodzi o wprowadzanie zmian w architekturze, które mają na celu ulepszenie projektu;
* Testy mogą służyć jako dokumentacja. Dobry kod powie, jak działa lepiej niż jakakolwiek dokumentacja. Dokumentacja i komentarze do kodu mogą być nieaktualne – może to być mylące dla programistów czytających kod. A ponieważ dokumentacja, w przeciwieństwie do testów, nie może powiedzieć, że jest przestarzała, nierzadko zdarzają się sytuacje, w których dokumentacja nie odpowiada rzeczywistości;
* Praktyka pokazuje, że ogólny czas pracy nad projektem jest skrócony w porównaniu z tradycyjnymi metodami tworzenia oprogramowania. Podczas gdy całkowita liczba linijek kodu wzrasta (dzięki testom), częstsze testowanie eliminuje błędy w procesie i znacznie wcześniej identyfikuje istniejące, zapobiegając problemom później[[6]](#footnote-6).

1. **Opis aplikacji**

**3.1 Charakterystyka problemu**

Zadaniem aplikacji jest kontrola oraz obsługa procedury i warunków prowadzenia certyfikacji zawodowej pracowników medycznych, farmaceutycznych i innych pracowników opieki zdrowotnej.

Niżej przedstawiłem tłumaczenie najbardziej znaczących zagadnień z aktu normatywnego dotyczącego tego tematu[[7]](#footnote-7).

Certyfikacja jest przeprowadzana w celu przypisania lub potwierdzenia kategorii kwalifikacyjnej pracownika. Przypisanie lub potwierdzenie kategorii kwalifikacyjnej odbywa się sekwencyjnie – druga, pierwsza, wyższa.

Dekret z dnia 28 maja 2021 r. określa, że pracownicy sfery medycznej, mające w dniu wejścia dekretu kategorię kwalifikacyjną, powinni przejść certyfikację zawodową w celu potwierdzenia własnej kategorii w ciągu 5 lat od ww daty.

Certyfikacji podlegają:

* kadra kierownicza i specjaliści medyczni oraz farmaceutyczni, pracownicy opieki zdrowotnej;
* przedstawiciele służby zdrowia;
* kadra dydaktyczna państwowych placówek edukacyjnych, realizujących szkolenia i przekwalifikowanie specjalistów o profilu kształcenia „Medycyna” (oraz specjalność „Farmacja”);
* inni pracownicy medyczni o wykształceniu wyższym lub średnim, pracujący w jednostkach służby zdrowia, placówkach edukacyjnych, szkoleniowych, specjaliści o profilu kształcenia „Opieka zdrowotna”.

Pracownicy mogą ubiegać się o wyższą kategorię kwalifikacyjną nie wcześniej niż trzy lata od daty wydania poprzedniej kategorii kwalifikacyjnej. Certyfikacja na potwierdzenie kategorii kwalifikacyjnej przeprowadzana jest raz na pięć lat. Dokumentację na potwierdzenie kategorii kwalifikacyjnej należy złożyć do komisji atestacyjnej nie później niż trzy miesiące przed wygaśnięciem kategorii kwalifikacyjnej. Certyfikacja do przypisania drugiej kategorii kwalifikacyjnej a potwierdzenie kategorii kwalifikacji jest obowiązkowe.

Z obowiązku certyfikacji zwolnieni są:

* pracownicy, którzy pracowali na odpowiednim stanowisku krócej niż rok;
* kobiety w ciąży;
* pracownicy, przebywające na długotrwałym (powyżej czterech miesięcy) leczeniu – na 6 miesięcy po wyjściu do pracy;
* osoby odbywające staż w szkole wyższej, studia doktoranckie – w okresie studiów, szkolenia;
* osoby przebywające na urlopie macierzyńskim, na urlopie wychowawczym dla dziecka do ukończenia trzeciego roku życia – na rok po zwolnieniu.

Kategorie kwalifikacyjne są przydzielane na podstawie kwalifikacji, odpowiednie z stanowiskiem pracownika (prowizor, prowizor-organizator, farmaceuta, lekarz organizujący opiekę zdrowotną, lekarz-ekspert).

Aby przypisać pracownikowi kategorię kwalifikacyjną, konieczne jest:

* druga kategoria kwalifikacyjna – mieć wiedzę teoretyczną oraz praktyczne umiejętności z zakresu działalności zawodowej, doświadczenie w pracy w odpowiedniej specjalności – co najmniej 3 lata, odbyć min. 100 godzin szkoleń zawodowych (dla pracowników o wykształceniu średnim medycznym lub farmaceutycznym – co najmniej 50 godzin);
* pierwsza kategoria kwalifikacyjna – mieć wiedzę teoretyczną oraz praktyczne umiejętności z zakresu działalności zawodowej, doświadczenie w pracy w odpowiedniej specjalności – co najmniej 6 lat, staż pracy w drugiej kategorii kwalifikacyjnej – co najmniej 3 lata, odbyć min. 160 godzin szkoleń zawodowych (dla pracowników o wykształceniu średnim medycznym lub farmaceutycznym – co najmniej 80 godzin);
* wyższa kategoria kwalifikacyjna – mieć wiedzę teoretyczną oraz praktyczne umiejętności z zakresu działalności zawodowej, doświadczenie w pracy w odpowiedniej specjalności – co najmniej 9 lat, staż pracy w pierwszej kategorii kwalifikacyjnej – co najmniej 3 lata, odbyć min. 160 godzin szkoleń zawodowych (dla pracowników o wykształceniu średnim medycznym lub farmaceutycznym – co najmniej 80 godzin).

Aby potwierdzić kategorię kwalifikacyjną pracownik powinien odbyć min. 160 godzin szkoleń zawodowych (dla pracowników o wykształceniu średnim medycznym lub farmaceutycznym – co najmniej 80 godzin).

Akumulacja wymaganego czasu szkoleń zawodowych, niezbędnych do przypisania (potwierdzenia) kategorii kwalifikacyjnej, odbywa się w ciągu 5 lat. Warunkiem koniecznym do przypisania (potwierdzenia) kategorii kwalifikacyjnej jest samodzielne opanowanie materiałów szkoleniowych, dla pracowników o wykształceniu wyższym medycznym w wymiarze co najmniej 80 godzin (dla pracowników z wykształceniem średnim medycznym lub farmaceutycznym - co najmniej 40 godzin). Pracownicy, którym przypisano kategorię kwalifikacyjną, którzy przeszli na nowe stanowisko służbowe, są dopuszczone do obrony na równorzędną kategorię kwalifikacyjną na nowym stanowisku pracy za dwa lata od daty rozpoczęcia pracy na nowym stanowisku. W tym czasie ważna jest kategoria kwalifikacyjna przypisana na poprzednim stanowisku pracy

1. K. Beck, *TDD. Sztuka tworzenia dobrego kodu*, Helion 2014, s. 10. [↑](#footnote-ref-1)
2. <https://bit.ly/3xRUGCE>, [dostęp: 14.08.2021]. [↑](#footnote-ref-2)
3. Tamże. [↑](#footnote-ref-3)
4. <https://bit.ly/3m94iqK>, [dostęp: 15.08.2021]. [↑](#footnote-ref-4)
5. V. Farcic, A. Garcia, *Test-Driven Java Development*, Packt Publishing Ltd., Birmingham 2015, s. 2. [↑](#footnote-ref-5)
6. <https://bit.ly/3z4S7hW>, [dostęp: 15.08.2021]. [↑](#footnote-ref-6)
7. <https://bit.ly/2ZeP7Ty>, [dostęp: 20.10.2021]. [↑](#footnote-ref-7)