wydział elektrotechniki elektroniki informatyki i automatyki

METODY TESTOWANIA OPROGRAMOWANIA

LABORATORIUM 2

Testy jednostkowe

wersja 1.3

przygotował:
dr inż. Radosław Adamus

Efekty:

Po ukończeniu laboratorium będziesz:
1. Potrafił projektować przypadki testowe i implementować je z wykorzystaniem frameworka JUnit.
2..Rozumiał potrzebę izolacji testu jednostkowego oraz różnice pomiędzy testami stanu a testami zachowania.
3. Potrafił samodzielnie opracować prosty mechanizm izolacji.

Wymagania wstępne:

1. Posiadanie konta na platformie Github.

Narzędzia:

1. Eclipse IDE
2. Git

Reguły wykonywania ćwiczeń laboratoryjnych:

1. Zmiany należy zatwierdzać często. Zatwierdzenie zbiorcze zmian na koniec laboratorium równoważne jest z jego niezaliczeniem.
2. Zmiany zatwierdzone w repozytorium kontroli wersji muszą posiadać znaczące komentarze.
3. Git powinien być tak skonfigurowany, aby zatwierdzane zmiany były identyfikowane danymi studenta (adres email oraz nazwisko).
4. W przypadku nieukończenia zadania w trakcie zajęć rezultat pośredni powinien być, po zatwiedzeniu w repozytorium lokalnym wypchnięte do macierzystego repozytorium na GitHub.

Opis laboratorium:

1. Implementacja testów na podstawie przypadków testowych:

1. Sklonuj (operacja fork) repozytorium https://github.com/mto-lab/lab2_1 na swoje konto GitHub (a następnie na lokalny komputer). Zaimportuj projekt do IDE.
2. Projekt zawiera metodę implementującą algorytm wyszukiwania. Początkowo będzemy projektować testy na podstawie założeń działania bez posiłkownia się wiedzą nt implementacji samego algorytmu wyszukującego. Założenia dotyczące działania są następujące:
   Sygnatura: SearchResult search(int key, int[] seq)
   Warunki wejściowe (preconditions):
   Długość sekwencji wejściowej > 0, sekwencja posortowana rosnąco bez duplikatów.
   Warunki wyjściowe (postconditions):
   Długość sekwencji wejściowej == 0 -> IllegalArgumentException
   LUB

element znaleziony -> searchResult.isFound() == true &&
searchResult.getPosition() == i takie, że seq[i] == key

LUB

element nieznaleziony -> searchResult.isFound() == false &&
searchResult.getPosition == -1

3. Zadanie polega na zaprojektowaniu oraz zimplementowaniu przypadków testowych opisanych w tabeli 1 (implementacja powinna wykorzystywać framework JUnit). Zmiany powinny być zatwierdzane w gałęzi o nazwie *testy_v1*. Jeżeli testy wykazały błąd niezgodności kodu ze specyfikacją - napraw go[1].

Tabela 1 - założenia przypadków testowych

Długość sekwencji wejściowej	**Element wyszukiwany**
1	Jest w sekwencji
1	Nie ma w sekwencji
>1	Jest pierwszym elementem
> 1	Jest ostatnim elementem
> 1	Jest środkowym elementem
> 1	NIe ma w sekwencji

4. Zapoznaj się ze wskazówkami dotyczącymi projektowania przypadków testowych (klasy równoważności i heurysytyki, dobre praktyki implementacji przypadków testowych w JUnit). Czy przypadki testowe z tabeli 1 są kompletne? Czy Twoje testy są zgodne z dobrymi praktykami? Dokonaj refaktoryzacji i ew. uzupełnienia przypadków testowych. Refaktoryzacja powinna również uwzględniać modyfikacje powodującą, że weryfikacja poprawności testów korzysta z metody Assert.assertThat (importowanej statycznie do przestrzeni nazw testu). Zatwierdź zmiany w gałęzi *test_v2*.

[1] Zakładamy, że mamy do czynienia z testem jednostkowym i rezultat testu nie jest raportowany tylko wykorzystywany bezpośrednio do poprawy kodu.

5. W jaki sposób na zestaw przypadków testowych wpłynie wiedza nt. implementacji algorytmu wyszukiwania (wyszukiwanie binarne). Uzupełnij przypadki testowe zatwierdzając zmiany w nowej gałęzi o nazwie test_v3.
6. Ostatecznie wszystkie utworzone gałęzie kodu powinny zostać wypchnięte (ang. push) do repozytorium zdalnego.

2. Projektowanie przypadków testowych

1. Sklonuj (operacja fork) repozytorium https://github.com/mto-lab/lab2_2 na swoje konto GitHub (a następnie na lokalny komputer). Zaimportuj projekt do IDE.
2. Projekt zawiera kod który poznaliśmy w ramach zadania 2 w laboratorium 1 (lab_2_1). Dokonaj analizy tego kodu pod kątem implementacji testów. Które klasy (metody klas) warto byłoby pokryć testami. Zaprojektuj i zaimplementuj przypadki testowe dla tych klas/metod.
3. Zmiany w repozytorium kontroli wersji zatwierdzaj po każdym teście. Ostatecznie rezultaty wypchnij do repozytorium na GitHub.

3. Testy jednostkowe a zależności zewnętrzne - testy stanu i testy zachowania

1. Sklonuj bezpośrednio na lokalny komputer projekt https://github.com/mto-lab/lab2_3_dep. Projekt musi być dostępny w ramach repozytorium zależności systemu maven. W tym celu należy z poziomu linii poleceń przejść do folderu projektu i wykonać komendę *mvn install*. Można tę operację wykonać również z poziomu IDE po zaimportowaniu projektu. Np. w Eclipse IDE należy dla projektu wykonać komendę Run As -> Maven install.
2. Sklonuj (operacja fork) repozytorium https://github.com/mto-lab/lab2_3 na swoje konto GitHub (a następnie na lokalny komputer). Zaimportuj projekt do IDE.
3. Projekt zawiera kod klasy SimilarityFinder oraz implementację metody implementującej indeks Jaccarda. Implementacja ta zależy od interfejsu dostarczającego operację wyszukiwania wartości w sekwencji. Jest to zewnętrzna zalezność, której implementacja nie jest obecnie dostepna.
4. Zaprojektuj przypadki testowe dla klasy SimiliarityFinder. Uwzględnij testy, które:
    a. sprawdzają poprawność rezultatu działania metody testowanej (np. czy rezultat jest zgodny z oczekiwanym).
    b. sprawdzają poprawnośćinterakcji z zależnością (np. czy metoda w interfejsie została wywołana z prawidłowym parametrem, określoną liczbę razy, itp).
5. Zaimplementuj przypadki testowe wraz z mechanizmami izolującymi (dublerami).
6. Zmiany w repozytorium kontroli wersji zatwierdzaj po każdym teście w dedykowanej gałęzi nazwanej *isolation_v1*. Ostatecznie rezultaty wypchnij do repozytorium na GitHub.

Przypisy

1. http://stackoverflow.com/questions/6197370/should-unit-tests-be-written-for-getter-and-setters
2. http://martinfowler.com/articles/mocksArentStubs.html