Tytuł kursu: Test Driven Development

Kod kursu: J-TDD

Dokument jest częścią oferty szkoleń firmy Javatech.

Pełna oferta znajduje się pod adresem: **http://www.javatech.com.pl/szkolenia.html**

Wstęp

Nie ma wątpliwości, że testowanie jest nieodłączną częścią procesu tworzenia oprogramowania. Jest to najbardziej popularna i najefektywniejsza w praktyce metoda wersyfikacji poprawności oprogramowania (choć zawsze należy pamiętać, że testowanie nie zapewnia 100% poprawności). Jednak tradycyjne „kaskadowe" podejście do tworzenia oprogramowania, w którym faza testów następuje po zakończeniu fazy implementacji, okazuje się nieefektywne i jest powszechnie krytykowane. Współczesne „iteracyjne" metodologie wytwarzania oprogramowania zalecają wykonywanie testów wraz z tworzeniem implementacji i przyrostowe budowanie poprawnego oprogramowania.

Idea programowania w oparciu o testy (Test-Driven Development) idzie jeszcze dalej i postuluje tworzenie w wielu mikro-iteracjach najpierw testu, a następnie implementacji realizującej ten test. Aby zapobiegać tworzeniu sztucznych rozwiązań spełniających tylko wybrany test, powstały kod należy jeszcze za każdym razem refaktoryzować.

Idea TDD jest wykorzystywana w metodyce tworzenia oprogramowania Extreme Programmingi innych metodykach z rodziny „zwinnych" (*agile*). Stosowanie TDD wpływa na sposób pracy w projekcie, a także na kształt wynikowego oprogramowania. O tym, czy ostatecznie wpływ ten jest pozytywny, trwają zażarte dyskusje. Idea TDD i metodologie „zwinne" uzyskały mimo wszystko dużą popularność w latach 2000-nych i wciąż są bardzo popularne w przemyśle, dlatego na pewno warto je poznać i wypróbować w praktyce.

Aby efektywnie pracować zgodnie z TDD, potrzebne są odpowiednie narzędzia. Wymagana jest przede wszystkim biblioteka do testowania jednostkowego (*unit testing*), ale przydatne mogą być także narzędzia do łatwego tworzenia obiektów zastępczych (*stub* lub *mock*) i biblioteki do testowania jednostkowego w specyficznych zastosowaniach (np. bazy danych, interfejs użytkownika).

Adresaci szkolenia

Szkolenie przeznaczone jest dla programistów Javy, którzy chcą podnieść komfort i jakość swojej pracy poprzez tworzenie testów dających informację zwrotną programiście, ułatwiających projektowanie klas i wspierających dobre praktyki programowania obiektowego. Wymagana jest praktyczna znajomość języka Java.

Cel szkolenia

Szkolenie koncentruje się przede wszystkim na praktycznym stosowaniu idei Test-Driven Development w Javie. Uczestnicy poznają szczegółowo bibliotekę JUnit, a dodatkowo także biblioteki Mockito, DBUnit, JFCUnit i HTTPUnit. Na szkoleniu przedstawione są także ogólne, niezależne od języka programowania, idee TDD i metodyk „zwinnego" tworzenia oprogramowania.

Podczas szkolenia, zgodnie z ideą TDD i jej różnymi wariantami, od podstaw tworzona jest przykładowa aplikacja. Wykorzystywana jest Java SE, baza danych (domyślnie SQLite), wspomniane narzędzia i biblioteki oraz środowisko programistyczne Eclipse.

Po zakończeniu szkolenia aktywny uczestnik:

- zna ideę tworzenia oprogramowania w oparciu o testy,
- potrafi tworzyć oprogramowanie w Javie w oparciu o testy,
- potrafi używać testu jako narzędzia do projektowania aplikacji,
- potrafi refaktoryzować kod aplikacji i kod testowy,
- potrafi korzystać z biblioteki JUnit,
- potrafi rozszerzać funkcjonalność dostępną w JUnit poprzez własne reguły, wzorce i zaawansowane sposoby uruchamiania testów,
- potrafi tworzyć testy zachowania, korzystając z obiektów zastępczych (*stub* i *mock*) tworzonych ze wsparciem biblioteki Mockito,
- potrafi pisać testy warstwy dostępu do bazy danych z wykorzystaniem biblioteki DBUnit,
- potrafi pisać testy GUI (swing) z wykorzystaniem biblioteki JFCUnit,
- potrafi pisać testy serwera HTTP (np. interfejsu WWW) z wykorzystaniem biblioteki HTTPUnit.

Czas i forma szkolenia

- 21 godzin (3 dni x 7 godzin) w tym wykłady i warsztaty praktyczne

Program szkolenia

1. Szybki start
   a) podstawowe idee TDD,
   b) JUnit – tworzenie i uruchamianie prostych testów.
2. JUnit 4
   a) klasa `Assert` i asercje,
   b) klasa `Assume` i warunki wstępne,
   c) inicjalizacja i porządki, pojęcie *fixture*,
   d) obsługa wyjątków i timeoutów,
   e) reguły (*Rule*) i wzorce (*Matcher*): przegląd wbudowanych, tworzenie własnych,
   f) grupowanie testów w zestawy,
   g) różne sposoby uruchamiania testów, m.in. `Runner`, `Request`,
   h) porównanie z JUnit 3.
3. Idee TDD
   a) proces budowy oprogramowania – różne metodyki,
   b) „zwinne” (*agile*) tworzenie oprogramowania,
   c) zasady pracy w w TDD, mikroiteracja,
   d) wpływ TDD na architekturę systemu i estetykę kodu,
   e) praktyczne odstępstwa od ortodoksyjnego TDD.
4. Wzorce i dobre praktyki TDD, refaktoryzacja.
5. Testowanie stanu a testowanie zachowania
   a) klasyczne podejście do testowania w oparciu o stan,
   b) idea programowania opartego o zachowanie (Behaviour-Driven Development),
   c) testowanie zachowania za pomocą obiektów zastępczych (*stub* i *mock*),
   d) biblioteka Mockito.
6. Organizacja, utrzymanie i refaktoryzacja kodu testowego.
7. Szczególne przypadki w testowaniu i TDD oraz związane z nimi narzędzia dla Javy:
   a) testowanie bazy danych – SQLUnit,
   b) testowanie warstwy dostępu do bazy danych – DBUnit,
   c) testowanie interfejsu użytkownika (GUI) – JFCUnit,
   d) testowanie przez sieć (tu HTTP) – HTTPUnit.