"Jeśli coś nie zostało przetestowane, należy uznać, że nie działa"

Test jednostkowy - *unit test*

„Test jednostkowy (unit test) to fragment kodu, który sprawdza inny fragment kodu”

Test jednostkowy - *unit test*

Metoda testowania tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania pojedynczych elementów (jednostek) programu - np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Testowany fragment programu poddawany jest testowi, który wykonuje go i porównuje wynik (np. zwrócone wartości, stan obiektu, wyrzucone wyjątki) z oczekiwanymi wynikami - tak pozytywnymi, jak i negatywnymi.

Dlaczego używa się testów jednostkowych ?

Pozwalają efektywnie wyszukiwać błędy w istniejącym kodzie.
Umożliwiają szybką weryfikację nowo- powstałych elementów programu.
Ułatwiają pracę w grupie dzięki łatwiejszemu łączeniu różnych fragmentów kodu podanych wcześniej testom.
Automatyzacja procesu testowania.

Korzyści testów jednostkowych

Testowanie zmusza do lepszego przemyślenia rozwiązań. Musimy dokładnie określić jakie zadania dana metoda ma wykonywać.
Łatwo przeprowadzić testy regresyjne. Po prostu po wprowadzeniu zmian przeprowadzamy stworzone przedtem testy sprawdzając czy zmiana nie wpływa na działanie innych fragmentów kodu.
Oszczędność czasu - testy są w pełni powtarzalne i zautomatyzowane.

Wady testów jednostkowych

Zmiana stylu pracy

Nigdy nie jest możliwe w 100 % wykluczenie pojawienia się jakiegoś błędu

Czego potrzebujemy:

kodu testowanego - kodu, który zawiera pewną funkcjonalność, którą będziemy testować

działających, wykonujących się poprawnie testów jednostkowych dla kodu testowanego.

NUnit

*J*est jednym z testujących narzędzi pod platformę .NET. Za jego pomocą można łatwo i przyjemnie pisać testy jednostkowe kodu (przyszłego lub wcześniej zaprojektowanego).

NUnit c.d.

Pakiet ten składa się z dwóch części:

zestawu klas bazowych umieszczonych w bibliotekach dll, które dołączamy do projektu i wykorzystujemy w naszych klasach testujących, oraz drugiej części, aplikacji uruchamiającej testy (mamy dwie wersje okienkową oraz konsolową).

Przykłady testów

Czas na przykład !

Instalacja NUnit
http://www.nunit.org/

Część Praktyczna

Abyśmy mogli zacząć testować cokolwiek musimy najpierw dodać referencje do framework’a Nunit.

Część Praktyczna

Posiadając jakiś projekt możemy śmiało rozpocząć pisanie naszego pierwszego testu. Klasę zawierającą testy jednostkowe oznaczamy atrybutem [TextFixture], natomiast poszczególne metody testowe atrybutem [Test]. Pozwala to środowisku testowemu na znalezienie metod które zawierają nasze testy jednostkowe. Wszystkie te elementy (zarówno klasa, jak i metody) muszą być publiczne, przy czym metody nie powinny nic zwracać i nie powinny przyjmować żadnych argumentów.

Pozostałe atrybuty

[Ignore] – ignorowanie testu
[TestFixtureSetUp] – oznaczanie metody wywoływanej przed testami
[TestFixtureTearDown] – oznaczanie metody wywoływanej po zakończeniu testów
[Category] – oznaczenie przynależności klasy testowej do danej kategorii
[Explicite] – ignorowanie testu w przypadku uruchomienia wszystkich testów naraz

Asercja

Zazwyczaj przyjmuje formę wyrażenia logicznego, które zwraca albo prawdę albo fałsz. Stanowi więc doskonałe narzędzie, dzięki któremu możemy w prosty sposób, wychwycić błędy w pisanych aplikacjach.

Przykłady asercji

AreEqual – porównanie wartości zmiennej
AreSame – porównanie referencji do obiektów
IsFalse, IsTrue – porównanie wartości logicznych
IsNull, IsNotNull – porównanie referencji z nullem
Fail – wymuszenie niepowodzenia testu

I wiele innych ...

Integracja NUnit z VS

Podpięcie pod tryb debugowania VS

TestDriven.NET- 3.0 (darmowy) pozwala na uruchamianie Nunit bezpośrednio z menu kontekstowego http://weblogs.asp.net/nunitaddin

Podsumowanie

Jak się okazuje Nunit jest bardzo pomocnym narzędziem pozwalającym nam sprawdzić, czy dany fragment programu (metoda) zachowuje się zgodnie z oczekiwaniami. Jak wiadomo nie wszystko da się przetestować za pomocą gui, a pisanie małych programów do testowania metod mija się z celem. Dzięki takiemu narzędziu, jakim jest Nunit możemy tworzyć stabilniejsze programy, działające zgodnie z oczekiwaniami, lepiej przetestowane.

Dziękuje za uwagę.

Paweł Łęgowski

Kontakt: pawellegowski89@gmail.com