Podstawy testowania jednostkowego

Testy jednostkowe

- Test jednostkowy
  - Fragment/jednostka kodu (zazwyczaj metoda lub funkcja), który:
    - wywołuje inny fragment/jednostkę kodu (nazwijmy go B) a następnie sprawdza poprawność założeń dotyczących działania fragmentu B.
    - jeżeli założenia nie są spełnione to uznajemy, że test jednostkowy zakończył się niepowodzeniem.

Własności dobrego testu jednostkowego

- Automatyczny i powtarzalny
- Łatwy do zaimplementowania
- Raz napisany – wielokrotnie wykorzystywany
- Możliwy do uruchomienia przez każdego
- Uruchamiany „jednym kliknięciem”
- Szybki

Fast
Independent
Repetable
Self validating
Timely

Test jednostkowy – bardziej precyzyjnie

- Test jednostkowy to **zautomatyzowany** fragment kodu:
  - wywołujący metodę/funkcję a następnie weryfikujący założenia dotyczące logiki zachowania metody/funkcji. Jest **prosty do zaimplementowania** oraz **szybki w działaniu**. Dodatkowo jest **wiarygodny**, **czytelny** i **możliwy do zarządzania**.

Kanoniczna struktura testu

1. **Inicjalizacja** (ang. setup) - inicjalizacja danych wejściowych, środowiska (SUT, DOC) oraz oczekiwań w stosunku do rezultatów.
2. **Wykonanie** (ang. call) – wykonanie przypadku testowego – wywołanie kodu podlegającego testowaniu.
3. **Weryfikacja** (ang. assertion) **założeń** – porównanie rezultatów wywołania z oczekiwaniami. Jeżeli założenia są spełnione (asercja ewaluuje do wartości prawda) – test zakończył się sukcesem. Jeżeli asercja ewaluuje do wartości fałsz – test zawiódł.
4. **Porządkowanie –** (ang. tear down) – sprzątanie po teście (np. usuwanie pozostawionego stanu, który mógłby mieć wpływ na działanie innych testów).
   - Opcjonalne

Kanoniczna struktura testu

1. **Given** setup) inicjalizacja danych wejściowych, środowiska (SUT, DOC) oraz oczekiwań w stosunku do rezultatów.
2. **When** ykonanie przypadku testowego – wywołanie metod podlegających testowaniu.
3. **Then** orównanie rezultatów wywołania z oczekiwaniami. Jeżeli założenia są spełnione (asercja ewaluuje do wartości prawda) – test zakończył się sukcesem. Jeżeli asercja ewaluuje do wartości fałsz – test zawiódł.
4. **Porządkowanie –** (ang. tear down) – sprzątanie po teście (np. usuwanie pozostawionego stanu, który mógłby mieć wpływ na działanie innych testów).
   - Opcjonalne

Testy jednostkowe - framework

- API (biblioteka) pozwalające na:
  - Łatwiejsze opracowywanie testów
  - Odpowiednią strukturalizację testu
  - Automatyczne wykonanie testów
  - Przeglądanie rezultatów testów

xUnit

- Nazwa grupy frameworków automatyzujących wykonywanie testów jednostkowych wywodzących się z SUnit (Smalltalk, Kent Beck)

- Architektura:
  - Test runner – środowisko wykonania testu
  - Test case – najmniejsza jednostka wykonania testu
  - Test fixture - kontekst wykonania testu
  - Test suite – zestaw testów
  - Test execution – wykonanie pojedynczego testu
  - Test result formatter – wyjściowy format rezultatów
  - Assertions – funkcja lub makro weryfikująca zachowanie lub stan testu

Meszaros, Gerard *xUnit Test Patterns*

JUnit

- Implementacja xUnit dla Javy

```
public class TestCaseStructure {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testMethod() {
        fail("Not yet implemented");
    }

}
```

Asercje

- Assertion – instrukcja typu predykat
  - Ewaluowana do wartości logicznej
  - Weryfikowanie poprawności założeń (np. przypadku testowego)
- Wbudowane w język
- Dostępne z frameworkiem
- Dostępne z jako odrębne narzędzia
  - Różne style
  - Różna czytelność

Junit a asercje

- Klasa Assert z zestawem statycznych metod
  - assert[Not]Equals(expected, actual);
  - assertFalse(expression);
  - assertTrue(expression);
  - assert[Not]Same, assert[Not]Null…
- Od JUnit 4.8.2 wykorzystanie Matchers (framework Harmcrest)
  - **assertThat**(T actual, Matcher<? super T> matcher)

Matchers

- assertEquals(expected, actual);
  - vs
- assertThat(actual, is(equalTo(expected)));

- assertThat("test", anyOf(is("test2"), containsString("te")));

Matchers - rozszerzanie

```
public class TestMatcher extends TypeSafeMatcher<String> {
   private String expected;

   private TestMatcher(String expected) {
      this.expected = expected;
   }

   @Override
   public void describeTo(Description description) {
      description.appendValue(expected);
   }

   @Override
   protected boolean matchesSafely(String item) {
      return item.equals(expected);
   }

   public static TestMatcher myEquals(String expected) {
      return new TestMatcher(expected);
   }
}
```

Bibliografia

- Gerard Meszaros: xUnit Test Patterns Refactoring Test Code
- https://github.com/junit-team/junit/wiki
- http://www.objectpartners.com/2013/09/18/the-benefits-of-using-assertthat-over-other-assert-methods-in-unit-tests/