SPRAWOZDANIE

Zaawansowane aplikacje WWW

Testy aplikacji WWW

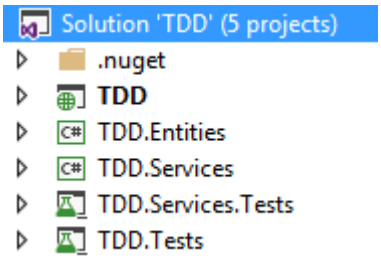
18.01.2015

Karol Suwalski 125NCI B

https://github.com/SuwalskiKarol/orw.git

TDD w aplikacji ASP.NET MVC na przykładzie operacji CRUD

Wygląd przykładowej aplikacji.



(Nie zdziwiłbym sięjakby Pan doktor jużwidział ten program. Stworzony na podstawie oficjalnego poradnika Microsoft. ☺. Lazinesslvlexpert.)

TDD - jest aplikacją MVC.

TDD.Entities – Tutaj przechowywane są podmioty jak np. reprezentacja naszych danych(ta wiem, mogłem to dać do modelu).

```
      publicint Id { get; set; }
publicStringFirstName { get; set; }
publicStringLastName { get; set; }
publicString Email { get; set; }
```

TDD.Services – Tutaj przechowywane są usługi

```
publicinterfaceIContactService
    {
IQueryable<Contact>GetAllContacts();
intAddContact(Contact contact);
ContactGetContact(int id);
voidEditContact(Contact contact);
voidDeleteContact(int id);
}
```

TDD.Services.Test – Tu piszemy testy usług

TDD.Test – testy aplikacji.

Kilka informacji zanim przejdziemy do przedstawienia samych testów.

Mamy różne metodologie pisania kodu za pomocą testów. Nie tylko TDD.

-BDD(Behaviordrivendevelopment)- Praktyka powstała na podstawie TDD, wykorzystywana w zwinnych metodykach. Celem BDD jest uzyskanie maksymalnego zrozumienia pożądanego zachowania oprogramowania w drodze dyskusji z zainteresowanymi stronami. Jest rozszerzeniem TDD wzbogacony o język naturalny służący do tworzenia przypadków testowych, siłą tego faktu jest to, że stanowi czytelną i przestępną formę dla osób nietechnicznych.

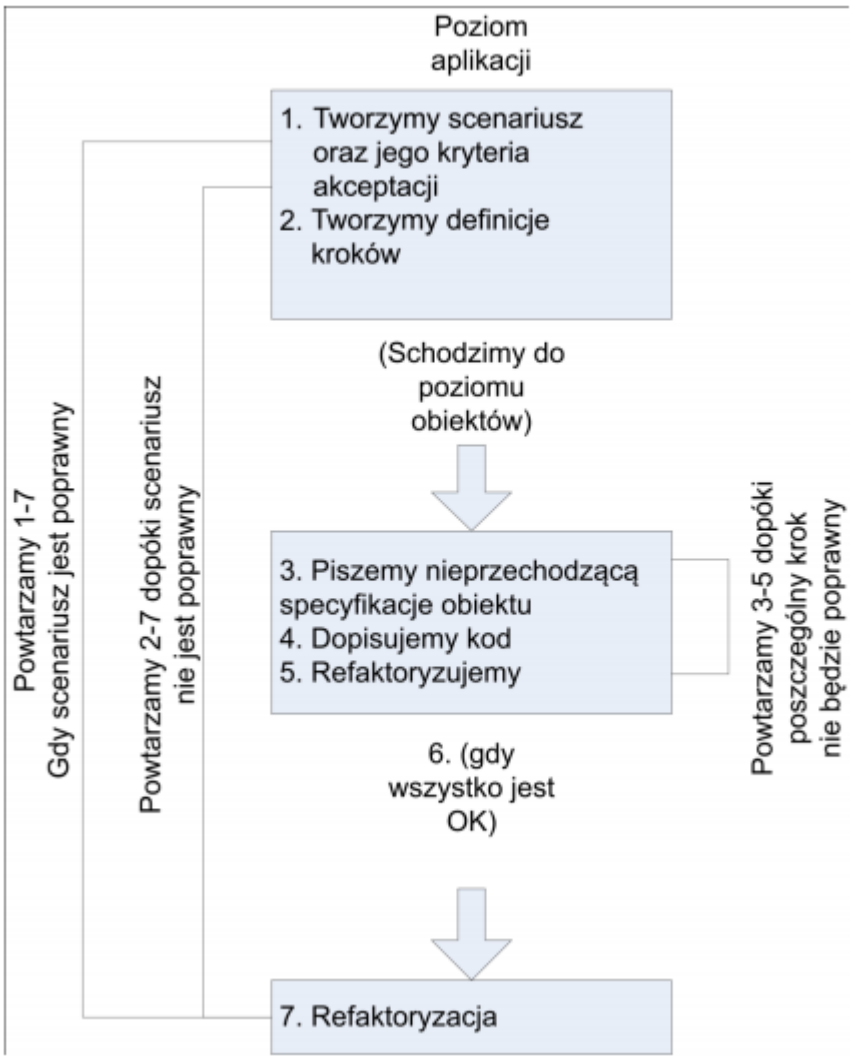
-SBE(Specification by Example) – niewiem co to jest :D

-ATDD(Acceptance Test-Driven Development) - ATDD jest metodą wytwarzania oprogramownia, w której kluczem do stworzenia linijki kodu jest przygotowany z wyprzedzeniem test napisany przez klienta.

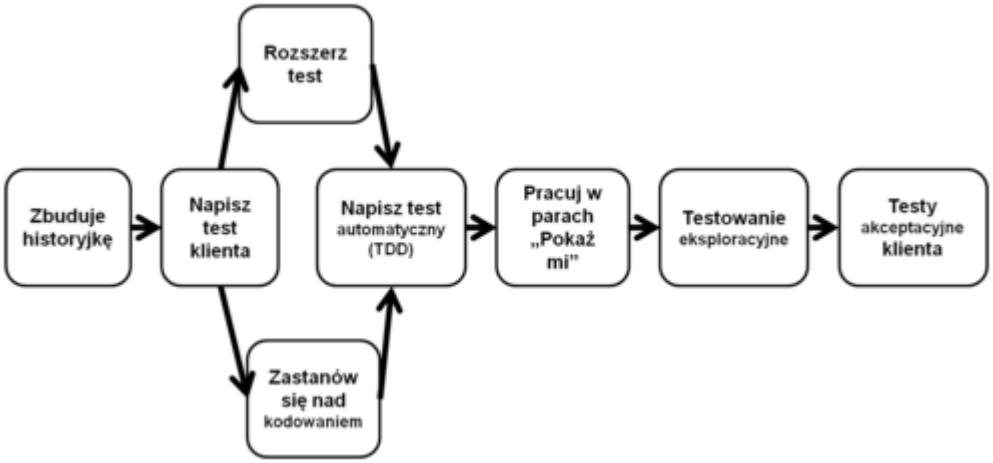
Zakładamy, że w tej praktyce (stosowanej głównie w Agile) cały zespół wspólnie definiuje kryteria akceptacji dla oprogramowania, a następnie tworzy z nich testy akceptacyjne (zanim wytworzony zostanie kod). Podczas wytwarzania kodu programiści automatyzują testy w oparciu o język naturalny zdefiniowany w testach klienta. Dzięki temu testy ATDD stają się wykonywalnymi wymaganiami.

-PBT(Property-BasedTesting) - Wymyślenie „dobrych" danych i przypadków testowych nie jest trywialnym zadaniem. Z pomocą przychodzi technika Property-BasedTesting, w której generalizuje się pojedyncze przypadki testowe do właściwości, a następnie sprawdza dla dużej liczby losowo wygenerowanych danych. Podejście to pozwala na odcięcie się od specyficznych danych testowych i skupienie się na działaniu programu.

Cykl BDD



Cykl ATDD



Samych testów też mamy kilka rodzajów:

- jednostkowe
- integracyjne
- akceptacyjne
- testy aplikacji webowych
- UI
- związane z pamięcią (memoryleaks)
- związane z programowaniem współbieżnym
- mutacyjne

Unit tests

Przede wszystkim napiszmy, że strukturę testu jednostkowego definiuje zasada Arrange–Act–Assert (AAA):

Arrange: wszystkie dane wejściowe i preconditions,

Act: działanie na metodzie/funkcji/klasie testowanej,

Assert: upewnienie się, że zwrócone wartości są zgodne z oczekiwanymi.

Jakie korzyści płyną ze stosowania tego wzorca? Przede wszystkim porządek; wzorzec zapewnia logiczny porządek w pojedynczym teście.

Czym jest moq?

Moq to najpopularniejszy framework do tworzenia atrap w .NET.Atrapa żyje tylko i wyłącznie w świecie testów, a jej celem jest symulacja zachowania prawdziwej zależności w oparciu o dane wyjściowe, które sami zdefiniujemy.Test jednostkowy z definicji testuje zachowanie w izolacji, a więc bez zależności zewnętrznych. Takimi zależnościami są najczęściej inne klasy lub interfejsy, które posiadają zachowanie.

Przykłady testów

Test odpowiadający za utworzenie metody Create.

```
[TestMethod]
publicvoidAddContact_Given_contact_ExpectContactAdded()
        {
var contact = newContact()
            {
FirstName = "Anna",
LastName = "kowalska"
            };
constintexpectedId = 1;
            _mockContactContext.Setup(x =>x.SaveChanges()).Callback(() =>contact.Id =
expectedId);

int id = _contactService.AddContact(contact);

            _mockContacts.Verify(x =>x.Add(contact), Times.Once);
            _mockContactContext.Verify(x =>x.SaveChanges(), Times.Once);
Assert.AreEqual(expectedId, id);
        }
```

UtworzeniemetodyEdit:

```
[TestMethod]
publicvoidEditContact_Given_contact_ExpectExistingContactUpdated()
        {
varstubData = (newList<Contact>
            {
newContact()
                {
                    Id = 1,
FirstName = "John",
LastName = "Doe"
                },
newContact()
                {
                    Id = 2,
FirstName = "Jane",
LastName = "Doe"
                }
            }).AsQueryable();
SetupTestData(stubData, _mockContacts);
var contact = newContact()
            {
                Id = 1,
FirstName = "Ted",
LastName = "Smith",
                Email = "test@gmail.com"
            };

            _contactService.EditContact(contact);
varactualContact = _mockContacts.Object.First();
```

```
Assert.AreEqual(contact.Id, actualContact.Id);
Assert.AreEqual(contact.FirstName, actualContact.FirstName);
Assert.AreEqual(contact.LastName, actualContact.LastName);
Assert.AreEqual(contact.Email, actualContact.Email);
            _mockContactContext.Verify(x =>x.SaveChanges(), Times.Once);
}
```

Utworzenie metody Delete:

```
[TestMethod]
publicvoidDeleteContact_Given_id_ExpectContactDeleted()
        {
varstubData = (newList<Contact>
            {
newContact()
                {
                    Id = 1,
FirstName = "John",
LastName = "Doe"
                },
newContact()
                {
                    Id = 2,
FirstName = "Jane",
LastName = "Doe"
                }
            }).AsQueryable();
SetupTestData(stubData, _mockContacts);
var contact = stubData.First();

            _contactService.DeleteContact(1);

            _mockContacts.Verify(x =>x.Remove(contact), Times.Once);
            _mockContactContext.Verify(x =>x.SaveChanges(), Times.Once);
}
```

Oczywiście testy piszemy do pustych klas. Kod w aplikacji MVC dopisujemy dopiero po nieudanym przejściu testu(RED). Potemrefaktoryzujemy kod i piszemy kolejne testy. Tak aż do powstania gotowej aplikacji.