НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем автоматизованого проектування

Звіт

З дисципліни

«Екстремальне програмування»

Лабораторної робота №5

«Unit-тести»

Варіант №14

Виконав:

Студент гр. КН-311

Куриляк Ю.А.

Прийняв:

Щербак С.С.

Львів – 2020

Теоретичні відомості:

Юніт-тести дозволяють швидко і автоматично протестувати окремі ділянки коду незалежно від іншої частини програми. При належному складанні юніт-тести цілком можуть покрити більшу частину коду програми.

Більшість юніт-тестів так чи інакше мають ряд таких ознак:

Тестування невеликих ділянок коду ("юнітів")

При створенні юніт-тестів вибираються невеликі ділянки коду, які треба протестувати. Як правило, тестований ділянка повинна бути менше класу, а в більшості випадків тестується окремий метод класу. Упор на невеликі ділянки дозволяє досить швидко писати простенькі тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в юніт-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

Тестування в ізоляції від решти коду

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

Тестування тільки загальнодоступних кінцевих точок

Всього лише невеликі зміни в класі можуть привести до невдачі багатьох юніт-тестів, оскільки реалізація використовуваного класу змінилася. Тому при написанні юніт-тестів обмежуються тільки загальнодоступними кінцевими точками, що дозволяє ізолювати юніт-тести від багатьох деталей внутрішньої реалізації компонента. В результаті зменшується ймовірність, що зміни в класах можуть привести до провалу юніт-тестів.

Автоматизація тестів

Написання юніт-тестів для невеликих ділянок коду веде до того, що кількість цих юніт-тестів стає дуже велике. І якщо процес отримання результатів і проведення тестів не автоматизовано, то це може привести до непродуктивної витраті робочого часу і зниження продуктивності. Тому важливо, щоб результати юніт-тестів представляли собою просте рішення,

яке означає, пройдений тест чи ні. Для автоматизації процесу розробники зазвичай звертаються до фреймворками юніт-тестування

Фреймворки тестування

MS Test: фреймворк юніт-тестування від компанії Microsoft, який за замовчуванням включений в Visual Studio (починаючи з VS 2012 у всі версії). Який використаємо у лабораторній роботі.

NUnit: портований фреймворк з JUnit для платформи .NET

xUnit.net: ще один фреймворк тестування від творців NUnit для платформи .NET

Виконання роботи:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using wcf_chat;
using Moq;
using MySql.Data.MySqlClient;
using System.Data;

namespace Backend.Tests
{
   [TestClass]
   public class ServiceChatTests
   {

      [TestMethod]
      public void CheckAuthorizationWithNotNullDataFromDatabase()
      {
         string login = "login";
         string password = "password12341231";
         string name = "name";
         int expID = 100;
```

```
            var mock = new Mock<I_InteractionDB>();
            mock.Setup(a => a.CheckAutorization(login, password)).Returns(new
ServerUser() { ID = expID, Name = name });
            ServiceChat serviceChat = new ServiceChat(mock.Object);

            int rez = serviceChat.UserAuthorization(login, password);

            Assert.AreEqual(rez, expID, "Function of avthorization returned value
{0} but not {1}", rez, expID);
        }

        [TestMethod]
        public void IntegrationCheckAuthorization()
        {
            string login = "testLogin";
            string password = "testPass";
            int expID = 1;
            ServiceChat serviceChat = new ServiceChat();



            int rez = serviceChat.UserAuthorization(login, password);

            Assert.AreEqual(rez, expID, "Function of avthorization returned value
{0} but not {1}", rez, expID);
        }

        /* [TestMethod]
         public void CheckAddToUserList()
         {
            string login = "testLogin";
            string password = "testPass";
            string name = "name";
            int expID = 1;

            var mock = new Mock<ServiceChat>();
```

```
            mock.Setup(a => a.CheckAuto).Returns(new ServerUser() { ID =
expID, Name = name });

            ServiceChat serviceChat = new ServiceChat(mock.Object);
            serviceChat.UserAuthorization(login, password);


            int rez = serviceChat.UserAuthorization(login, password);


            int rez = serviceChat.UserAuthorization(login, password);

            Assert.AreEqual(rez, expID, "Function of avthorization returned value
{0} but not {1}", rez, expID);
        }
        */

       /*[TestMethod]
       public void TestUsingMsgCallback()
       {
          string login = "login";
          string password = "password12341231";
          string name = "name";
          int expID = 100;

          var mock = new Mock<IServerChatCallback>();
          ServiceChat serviceChat = new ServiceChat(mock.Object);


          int rez = serviceChat.UserAuthorization(login, password);

          Assert.AreEqual(rez, expID, "Function of avthorization returned value
{0} but not {1}", rez, expID);
       }
       */
```

```
        [TestMethod]
        public void IntegrationCheckAddToDatabase()
        {
            string table = "users";
            string login = "login122311";
            string password = "password123412311211";
            string name = "name1231112";
            int expID = 100;

            //var mock = new Mock<InteractionDB>();
            string connectionStr = "server = localhost;user = root; database =
test_pm_database; password = root";
            //string connectionStr =
"server=localhost;user=root;database=PM_DB;password=root";
            MySqlConnection connection = new MySqlConnection(connectionStr);
            connection.Close();
            connection.Open();

            //string commandStr = "create table users ( id INT, login   varchar(50),
username varchar(30), pass    varchar(100)); ";
            //MySqlCommand command = new MySqlCommand("drop table users",
connection);
            //MySqlCommand command = new MySqlCommand(commandStr,
connection);

            InteractionDB interactionDB = new InteractionDB(connection);

            bool rez = interactionDB.AddToDB(table, """ + login + "","" + name +
"","" + password + """);

            connection.Open();
            MySqlCommand command = new MySqlCommand("Select * from " +
table + " where login = '" + login + "' and pass = '" + password + """,
connection);
```

```
            MySqlDataReader reader = command.ExecuteReader();

            Assert.IsTrue(rez, "Function AddToDB returned value 'false'");
            Assert.IsTrue(reader.Read(), "User exist in table ");
            connection.Close();
        }

        [TestMethod]
        public void IntegrationCheckGetUserByID()
        {
            string table = "users";
            string login = "login1241";
            string password = "password111";
            string name = "name112";
            int expID = 100;

            string connectionStr = "server = localhost;user = root; database =
test_pm_database; password = root";
            MySqlConnection connection = new MySqlConnection(connectionStr);
            connection.Close();
            connection.Open();

            string commandStr = "insert into users(id, login, pass, username) values
(" + expID.ToString() + ",'"  + login + "','" + password + "','" + name + "')";

            InteractionDB interactionDB = new InteractionDB(connection);

            int rez = interactionDB.GetIDByUsername(name);

            Assert.AreEqual(rez, expID, "expID is {0} but no {1}", expID, rez);
            connection.Close();
        }

        [TestMethod]
        public void CheckAuthorizationWithNotNullDataFromDatabase()
```

```
        {
            string login = "123";
            string password = "123";

            var mock = new Mock<ServiceChatClient>();
            var mockWindow = new Mock<MainWindow>();
            mock.Setup(a => a.UserAuthorization(login, password)).Returns(1);

            pAuthorization authorization = new
pAuthorization(mockWindow.Object, mock.Object);
            authorization.checkAuthorization(login, password);

            mockWindow.Verify(a => a.setUserId(2));
        }

    }
}
```