

Министерство науки и высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №4
«Исследование способов интеграционного тестирования программного
обеспечения»
по дисциплине
«ТЕСТИРОВАНИЕ ПО»

Выполнил студент группы ИС/б-17-2-о

Горбенко К. Н.

Проверил

Тлуховская Н.П.

Севастополь
2019

1 ЦЕЛЬ РАБОТЫ

Исследовать основные принципы интеграционного тестирования программного обеспечения. Приобрести практические навыки организации интеграционных тестов для объектно-ориентированных программ.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать в качестве тестируемого взаимодействие двух или более классов, спроектированных в лабораторных работах № 1 – 4.
2. Составить спецификацию тестового случая.
3. Реализовать тестируемые классы и необходимое тестовое окружение на языке C#.
4. Выполнить тестирование с выводом результатов на экран и сохранением в log-файл.
5. Проанализировать результаты тестирования, сделать выводы.

3 ХОД РАБОТЫ

В качестве тестируемого взаимодействия выберем взаимодействие классов `GetColumnsUnitTestEngine`, `FileLogger` и `GetColumnsUnitTestDto`, где `GetColumnsUnitTestEngine` - тестовый двигатель для выполнения модульных тестов над методом `GetColumns` (Л.Р.№ 2).

```

1 public class GetColumnsUnitTestEngine
2 {
3     private ILogger Logger { get; set; }
4     private IEnumerable<GetColumnsUnitTestDto> TestSuite { get; set; }
5
6     public GetColumnsUnitTestEngine(ILogger logger, IEnumerable<
7         GetColumnsUnitTestDto> testSuite)
8     {
9         Logger = logger ?? throw new ArgumentNullException(nameof(logger));
10        TestSuite = testSuite ?? throw new ArgumentNullException(nameof(logger)
11            );
12    }
13
14    public void Run()
15    {
16        foreach (var test in TestSuite)
17        {
18            // ...
19        }
20    }
21 }

```

```

16         try
17         {
18             var actual = test.Input.GetColumns();
19             var result = actual.SequenceEqual(test.Expected, new
                Int32ArrayEqualityComparer());
20
21             Logger.Log($"Test {test.Name}," +
22                 $"result:{result}" +
23                 $"{{(result ? "" : $"},expected: {
                    PrintArrayCollection(test.Expected)} actual:{
                    PrintArrayCollection(actual)}}");
24         }
25         catch (Exception e)
26         {
27             Logger.Log($"Test {test.Name}," +
28                 $"result:{false}," +
29                 $"Exception: {e.Message}");
30         }
31     }
32 }
33
34 private string PrintArrayCollection(IEnumerable<int[]> collection)
35 {
36     var stringBuilder = new StringBuilder("{ ");
37
38     foreach (var array in collection)
39     {
40         stringBuilder.Append("[ ");
41         foreach (var item in array)
42         {
43             stringBuilder.Append($"{item} ");
44         }
45         stringBuilder.Append("] ");
46     }
47     stringBuilder.Append("}");
48     return stringBuilder.ToString();
49 }
50 }

```

Класс `GetColumnsUnitTestEngine` зависит от экземпляров `FileLogger` и `GetColumnsUnitTestDto`. Это взаимодействие 3-го типа (классы `FileLogger` и `GetColumnsUnitTestDto` - часть его реализации).

3.1 Тестовые случаи

Опишем два тестовых случая:

1. Тест экземпляра FileLogger

a. Взаимодействующие классы: GetColumnsUnitTestEngine и FileLogger.

b. Имя теста: GetColumnsUnitTestEngine вызывает метод Log класса FileLogger.

c. Описание теста: при прогоне тестовой последовательности, класс GetColumnsUnitTestEngine для записи результатов должен вызывать метод Log класса FileLogger.

2. Тест экземпляров тестов

a. Взаимодействующие классы: GetColumnsUnitTestEngine и GetColumnsUnitTestDto.

b. Имя теста: GetColumnsUnitTestEngine записывает результат выполнения каждого теста последовательности в файл.

c. Описание теста: после прогона тестовой последовательности, для каждого теста в тестовой последовательности должна остаться запись в файле.

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для тестирования случая № 1 создадим замену классу FileLogger:

```

1 public class FakeLogger : ILogger
2 {
3     public bool LogCalled { get; private set; } = false;
4
5     public void Log(string message)
6     {
7         LogCalled = true;
8     }
9
10    public void Dispose() { }
11 }
```

Свойство LogCalled позволяет определить, был ли метод Log вызван. Использование замены:

```

1 private bool TestThatLogMethodGetsCalled()
2 {
3     try
4     {
5         this.logger.Log($"Test {nameof(TestThatLogMethodGetsCalled)}.");
6         var logger = new FakeLogger();
```

```

7      var unitTestEngine = new GetColumnsUnitTestEngine(logger,
          UnitTestConstants.TestSuite);
8      this.logger.Log("Passing logger.");
9      this.logger.Log("Running unit tests.");
10     unitTestEngine.Run();
11     bool result = logger.LogCalled;
12     if (result == true)
13     {
14         this.logger.Log("Test passed.");
15     }
16     else
17     {
18         this.logger.Log("Test failed.");
19     }
20     return result;
21 }
22 catch
23 {
24     return false;
25 }
26 }

```

Для тестирования случая № 2 выполним последовательно проверку каждой строки файла. Она должна содержать имя прогоняемого теста. Реализация:

```

1 private bool TestThatAllTestSuiteTestsAreRunning()
2 {
3     try
4     {
5         this.logger.Log($"Test {nameof(TestThatAllTestSuiteTestsAreRunning)}.");
6         ;
7         var logger = new FileLogger(new StreamWriter(UnitTestConstants.Path));
8         var unitTestEngine = new GetColumnsUnitTestEngine(logger,
9             UnitTestConstants.TestSuite);
10        this.logger.Log("Passing logger.");
11        this.logger.Log("Running unit tests.");
12        unitTestEngine.Run();
13        logger.Dispose();
14        this.logger.Log("Checking file content.");
15        bool result = true;
16        using (var streamReader = new StreamReader(UnitTestConstants.Path))
17        {
18            foreach (var test in UnitTestConstants.TestSuite)
19            {
20                var fileLine = streamReader.ReadLine();
21                if (!fileLine.StartsWith($"Test {test.Name}"))
22                {
23                    result = false;
24                }
25            }
26        }
27    }
28    catch
29    {
30        result = false;
31    }
32 }

```

```

22         }
23     }
24 }
25 if (result == true)
26 {
27     this.logger.Log("Test passed.");
28 }
29 else
30 {
31     this.logger.Log("Test failed.");
32 }
33 return result;
34 }
35 catch
36 {
37     return false;
38 }
39 }

```

Тестовый драйвер полученных интеграционных тестов:

```

1 public class GetColumnsUnitTestEngineIntegrationTestEngine
2 {
3     private readonly ILogger logger;
4
5     public GetColumnsUnitTestEngineIntegrationTestEngine(ILogger logger)
6     {
7         this.logger = logger ?? throw new ArgumentNullException(nameof(logger))
8         ;
9     }
10
11     public void Run()
12     {
13         TestThatLogMethodGetsCalled();
14         TestThatAllTestSuiteTestsAreRunning();
15     }
16
17     private bool TestThatLogMethodGetsCalled()
18     {
19         ...
20     }
21
22     private bool TestThatAllTestSuiteTestsAreRunning()
23     {
24         ...
25     }
26 }

```

ВЫВОДЫ

В ходе лабораторной работы был протестирован тестовый драйвер, созданный для реализации модульных тестов к лабораторной работе № 2. Тестирование проводилось с точки зрения интеграции в него экземпляров объектов логгера и тестовой последовательности. Было выявлено, что тестовый драйвер правильно взаимодействует с используемыми объектами (в соответствии с их спецификацией).

Модульное и интеграционное тестирование представляет из себя трудоемкий процесс из-за необходимости создавать тестовое окружение для каждого тестируемого метода. Чтобы упростить этот процесс необходимо использовать тестовые и mock-фреймворки.