3 Лабораторная работа №3
Исследование способов модульного тестирования программного обеспечения

3.1. Цель работы

Исследовать основные подходы к модульному тестированию программного обеспечения. Приобрести практические навыки составления модульных тестов для объектно-ориентированных программ.

3.2. Постановка задачи

Был выдан вариант 14.

Выбрать в качестве тестируемого один из классов, спроектированных в лабораторной работе №1. Составить спецификацию тестового случая для одного из методов выбранного класса. Реализовать тестируемый класс и необходимое тестовое окружение. Выполнить тестирование с выводом результатов на экран и сохранением в log-файл. Проанализировать результаты тестирования, сделать выводы.

3.3. Ход выполнения работы

3.3.1. Спецификация тестового случая

Было принято решение протестировать класс MyLine.

Для тестируемого класса был определён тестовый случай: проверяется правильность работы метода containsThreeDigitNumber() - определить, содержит ли переданная строка трёхзначные числа. В тесте подаются значения, определённые в лабораторной работе №1.

3.3.2.Текст программы

На основе спецификации был создан тестовый драйвер LineModuleTest, реализованный в виде отдельного класса, вызывающего public-методы тестируемого класса. Текст драйвера представлен в листинге 1.

Листинг 1 — Тестовый драйвер LineModuleTest

```
package org.cory7666.softwaretestingexample;

import java.io.PrintStream;
import java.util.Collection;

import org.cory7666.softwaretestingexample.task3.MyLine;

public class LineModuleTest implements AutoCloseable
{
  private int testNumCounter;
  private final Collection<PrintStream> outputStreams;
  public LineModuleTest (Collection<PrintStream> outputStreams)
  {
    this.testNumCounter = 0;
    this.outputStreams = outputStreams;
  }
  public LineModuleTest test (String testCase, boolean expected)
  {
    ++testNumCounter;
    boolean result = new MyLine(testCase).containsThreeDigitNumber();
    outputStreams.forEach(writer -> {
      try
      {
        writer.println(String.format("-===        Тест %2d        ===-", testNumCounter));
        writer
          .println(
            String
              .format(
                "Строка: \"%s\". Ожидаемый результат: %b. Получено: %b.",
                testCase,
                expected,
                result));
        writer.println(String.format("-=== Завершение теста %2d ===-\n", testNumCounter));
      }
      catch (Exception ex) {}
    });
    return this;
  }
  @Override
  public void close () throws Exception
  {
    for (var stream : outputStreams)
    {
      try
      {
        stream.close();
      }
      catch (Exception ex)
      {}
    }
  }
}
```

3.3.3. Тестовый запуск драйвера и анализ полученных данных

После написания кода программа была скомпилирована и запущена. Рисунок 3.1 демонстрирует вывод тестового драйвера.

```
❯ java -jar out.jar
-===        Тест  1        ===-
Строка: "34". Ожидаемый результат: false. Получено: false.
-=== Завершение теста  1 ===-

-===        Тест  2        ===-
Строка: "Было подобрано число 33.". Ожидаемый результат: false. Получено: false.
-=== Завершение теста  2 ===-

-===        Тест  3        ===-
Строка: "101 далматинец". Ожидаемый результат: true. Получено: true.
-=== Завершение теста  3 ===-

-===        Тест  4        ===-
Строка: "2022 год". Ожидаемый результат: false. Получено: false.
-=== Завершение теста  4 ===-

-===        Тест  5        ===-
Строка: "13 литров 00509 миллилитров". Ожидаемый результат: true. Получено: true.
-=== Завершение теста  5 ===-

-===        Тест  6        ===-
Строка: "11 точка 0000900". Ожидаемый результат: true. Получено: true.
-=== Завершение теста  6 ===-
```

Рисунок 3.1 — Результат запуска тестового драйвера

Полученные данных были проанализированы и сравнены со спецификацией. Данные полностью соответствуют спецификации.

Выводы

При выполнении данной лабораторной работы были получены навыки составления модульных тестов. Было выделено, что модульные тесты необходимо применять при тестировании модуля программы, или, в случае этой работы, при тестировании класса.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего применяется тестирование программного обеспечения?

Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.

Цель тестирования — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

2. Какие существуют разновидности тестирования?

Вот основные стратегии в тестировании программного обеспечения:

- Модульное тестирование

Этот подход к тестированию программного обеспечения используется программистом для тестирования отдельно взятого модуля программы. Это помогает разработчикам узнать, правильно ли работает каждый блок кода в изоляции от остальных.

- Интеграционное тестирование

Основное внимание уделяется созданию и проектированию программного обеспечения. Вы должны видеть, что при взаимодействии интегрированные блоки работают без ошибок.

- Системное тестирование

В этом методе ваше программное обеспечение компилируется как единое целое, а затем как единое целое тестируется. Эта стратегия проверяет, среди прочего, функциональность, безопасность и переносимость.

- Валидационное тестирование

Процесс оценки программного обеспечения с целью определить – удовлетворяет ли оно определенным бизнес-требованиям. Валидационное тестирование гарантирует, что продукт соответствует потребностям клиента. Его также можно расценивать как демонстрацию того, что продукт будет выполнять свое предназначение при развертывании в соответствующей среде.

3. Что такое модульное тестирование?

Модульное тестирование ПО – это процесс тестирования, при котором отдельные компоненты или модули программного обеспечения (функции, классы, методы и т.д.) тестируются в изоляции от других компонентов программы.

Цель модульного тестирования - проверить корректность работы каждого модуля отдельно, а также убедиться, что каждый модуль соответствует заявленным требованиям и выполняет свою функцию правильно.

Модульное тестирование позволяет быстро выявить и исправить ошибки и дефекты в коде, еще до того, как модуль будет включен в более крупные программные системы. Это снижает риск возникновения ошибок и дефектов на более высоких уровнях абстракции, где их устранение может оказаться более сложным и затратным процессом.

Модульное тестирование может быть автоматизировано с использованием специальных инструментов, таких как фреймворки для модульного тестирования или библиотеки тестирования. Эти инструменты позволяют быстро и эффективно создавать и запускать тесты на различных платформах и операционных системах.

4. Что понимается под модулем при модельном тестировании?

Цель тестирования программных модулей состоит в том, чтобы удостовериться, что каждый модуль соответствует своей спецификации. В процедурно-ориентированном программировании модулем называется процедура или функция, иногда группа процедур. Тестирование модулей обычно представляет собой неко-

торое сочетание проверок и прогонов тестовых случаев. Можно составить план тестирования модуля, в котором учесть тестовые случаи и построение тестового драйвера.

5. Каков порядок модульного тестирования классов?

Таким образом, для тестирования любого метода класса необходимо:

- Определить, какая часть функциональности метода должна быть протестирована, то есть при каких условиях он должен вызываться. Под условиями здесь понимаются параметры вызова методов, значения полей и свойств объектов, наличие и содержимое используемых файлов и т. д.

- Создать тестовое окружение, обеспечивающее требуемые условия.

- Запустить тестовое окружение на выполнение.

- Обеспечить сохранение результатов в файл для их последующей проверки.

- После завершения выполнения сравнить полученные результаты со спецификацией.

6. Какие разделы включает спецификация тестового случая?

Спецификация тестового случая включает следующие разделы:

- Идентификация - уникальный идентификатор тестового случая, который используется для управления тестами и отслеживания результатов.
- Описание - описание тестового случая, включая цель тестирования, действия пользователя и ожидаемый результат.
- Предусловия - начальные условия, которые необходимы для запуска теста, например, состояние программы или данные, на которых тестирование проводится.
- Шаги выполнения - последовательность шагов, которые необходимо выполнить для проведения тестирования, включая ввод данных, выполнение действий и проверку результатов.

- Ожидаемый результат - описание ожидаемых результатов после выполнения шагов тестирования.
- Фактический результат - результаты, полученные в результате выполнения теста, включая ошибки, предупреждения и любые другие наблюдения.
- Статус - результат выполнения теста, например, пройден, не пройден или отменен.
- Исполнитель - информация об исполнителе теста, кто проводил тестирование и когда оно было проведено.
- Комментарии - любые дополнительные комментарии, которые могут помочь понять результаты тестирования или описать особенности теста.

Спецификация тестового случая является важным документом для тестирования программного обеспечения, который позволяет определить, что должно быть протестировано и как. Она также помогает документировать результаты тестирования, что может быть полезно при поиске и исправлении ошибок и дефектов.