Лабораторная работа № 3 Модульное тестирование: задачи и цели. Понятие модуля и его границ. Тестирование классов. Проектирование тестового окружения (8 часов)

Цель:разработать тестовое окружение, включающее в себя драйвер и заглушки

Порядок выполнения работы:

1. Изучение теоретических сведений по теме лабораторной работы
2. Для каждого модуля, подвергаемого тестированию, разработать тестовое окружение, включающее в себя драйвер и заглушки
3. Подготовить тест-требования и тест-планы, описывающие конкретные тестовые примеры.
4. Оформление отчета по лабораторной работе, который должен содержать:
   - Название и задание к лабораторной работе
   - Ответы на контрольные вопросы и задания
5. Защита лабораторной работы

Модульное тестирование - это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель *модульного тестирования* состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. *Модульное тестирование* проводится по принципу "белого ящика", то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Модульное тестирование обычно подразумевает создание вокруг каждого модуля определенной среды, включающей заглушки для всех интерфейсов тестируемого модуля. Некоторые из них могут использоваться для подачи входных значений, другие для анализа результатов, присутствие третьих может быть продиктовано требованиями, накладываемыми компилятором и сборщиком.

На уровне *модульного тестирования* проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов. Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне *модульного тестирования* и выявляются на более поздних стадиях тестирования.

Пример модульного тестирования

Предлагается протестировать класс TCommand, который реализует команду для склада. Этот класс содержит единственный метод TCommand.GetFullName(), спецификация которого описана следующим образом:

...

Операция `GetFullName()` возвращает полное имя команды, соответствующее ее допустимому коду, указанному в поле `NameCommand`. В противном случает возвращается сообщение "ОШИБКА : Неверный код команды". Операция может быть применена в любой момент.

...

Разработаем спецификацию тестового случая для тестирования метода GetFullName на основе приведенной спецификации класса (Табл. 1):

Таблица 1. Спецификация теста

Название класса: `TCommand`	**Название тестового случая:** `TCommandTest1`
Описание тестового случая: Тест проверяет правильность работы метода `GetFullName` - получения полного названия команды на основе кода команды. В тесте подаются следующие значения кодов команд (входные значения): -1, 1, 2, 4, 6, 20, (причем -1 - запрещенное значение).	
Начальные условия: Нет.	
Ожидаемый результат: Перечисленным входным значениям должны соответствовать следующие выходные: Коду команды -1 должно соответствовать сообщение "ОШИБКА: Неверный код команды" Коду команды 1 должно соответствовать полное название команды "ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ" Коду команды 2 должно соответствовать полное название команды "ОТПРАВИТЬ ИЗ ЯЧЕЙКИ В ВЫХОДНУЮ ЯЧЕЙКУ" Коду команды 4 должно соответствовать полное название команды "ПОЛОЖИТЬ В РЕЗЕРВ" Коду команды 6 должно соответствовать полное название команды "ПРОИЗВЕСТИ ЗАНУЛЕНИЕ" Коду команды 20 должно соответствовать полное название команды "ЗАВЕРШЕНИЕ КОМАНД ВЫДАЧИ"	

Для тестирования метода класса TCommand.GetFullName() был создан тестовый драйвер - класс TCommandTester. Класс TCommandTester содержит метод TCommandTest1(), в котором реализована вся функциональность теста. В данном случае для покрытия спецификации достаточно перебрать следующие значения кодов команд: -1, 1, 2, 4, 6, 20, (-1 - запрещенное значение) и получить соответствующее им полное название команды с помощью метода GetFullName() (Пример 5). Пары значений (X, Yв) при исполнении теста заносятся в log-файл для последующей проверки на соответствие спецификации.

После завершения теста следует просмотреть журнал теста, чтобы сравнить полученные результаты с ожидаемыми, заданными в спецификации тестового случая TCommandTest1 (Пример 6).

```
class TCommandTester:Tester // Тестовый драйвер
```

```
{
  ...
  TCommand OUT;
  public TCommandTester()
  {
    OUT=new TCommand();
    Run();
  }
  private void Run()
  {
    TCommandTest1();
  }
   private void TCommandTest1()
  {
    int[] commands = {-1, 1, 2, 4, 6, 20};

    for(int i=0;i<=5;i++)
    {
      OUT.NameCommand=commands[i];
      LogMessage(commands[i].ToString()+
        " : "+OUT.GetFullName());
    }
  }
  ...
}
```

Пример 5. Тестовый драйвер (html, txt)

```
-1 : ОШИБКА : Неверный код команды
1 : ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ
2 : ОТПРАВИТЬ ИЗ ЯЧЕЙКИ В ВЫХОДНУЮ ЯЧЕЙКУ
4 : ПОЛОЖИТЬ В РЕЗЕРВ
6 : ПРОИЗВЕСТИ ЗАНУЛЕНИЕ
20 : ЗАВЕРШЕНИЕ КОМАНД ВЫДАЧИ
```

Рекомендуемая литература

1. В.В. Кулямин. Методы верификации программного обеспечения. - М.: Институт системного программирования РАН, 2019. – 211 с.
2. В.В. Липаев. Программная инженерия. – М.: Гос. ун-т - Высшая школа экономики. – ТЕИС, 2016. – 608 с.
3. С.В. Синицын. Верификация программного обеспечения: учебное пособие. – Саратов: Профобразование, 2019. — 368 с.

Контрольные задания для СРС

1. Разработать схему иерархии создаваемых классов.

2. Чем различаются основные подходы к проектированию тестового окружения?