Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Таныгин Максим Олегович
Должность: и.о. декана факультета фундаментальной и прикладной информатики
Дата подписания: 21.09.2023 12:55:38
Уникальный программный ключ:
65ab2aa0d384efe8480e6a4c688eddbc475e411a

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» 12 2017 г.

ТЕСТИРОВАНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРОГРАММ

Методические указания
к лабораторным работам по дисциплине
«Языки объектно-ориентированного программирования»
для студентов направления подготовки
09.03.04 «Программная инженерия»

Курск 2017

УДК 681.3

Составитель Е.И.Аникина

Рецензент
Кандидат технических наук, доцент кафедры программной инженерии *Н.Н. Бочанова*

Тестирование объектно-ориентированных программ: методические указания к лабораторным работам по дисциплине «Языки объектно-ориентированного программирования» для студентов направления подготовки 09.03.04 «Программная инженерия»/Юго-Зап. гос. ун-т; сост. Е.И. Аникина. Курск, 2017. 13 с.

Содержит теоретические сведения, примеры разработки тестовых наборов и задания к лабораторной работе по разделам дисциплины, связанным с технологией тестирования приложений в парадигме объектно-ориентированного подхода.

Предназначено для студентов направления подготовки бакалавров 09.03.04 «Программная инженерия».

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл. печ. л. . Уч.-изд. л. . Тираж 100 экз. Заказ . Бесплатно.

Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Лабораторная работа

ТЕСТИРОВАНИЕ КЛАССОВ

Цель тестирования программных модулей состоит в том, чтобы удостовериться, что каждый *модуль* соответствует своей спецификации. Если это так, то причиной любых ошибок, которые возникают при их объединении, является неправильная стыковка модулей. В процедурно-ориентированном программировании модулем называется процедура или *функция*, иногда *группа*процедур, которая реализует абстрактный *тип данных*. Тестирование модулей обычно представляет собой некоторое сочетание проверок и **прогонов тестовых случаев**. Можно составить план тестирования модуля, в котором учесть тестовые случаи и построение **тестового драйвера.**

Тестирование классов аналогично тестированию модулей. Основным элементом объектно-ориентированной программы является *класс*. Рассмотрим методику тестирования отдельного класса. Тестирование классов охватывает виды деятельности, ассоциированные с проверкой реализации класса на точное соответствие спецификации класса. Если реализация корректна, то каждый экземпляр этого класса ведет себя подобающим образом.

Эффективного тестирования классов можно достичь при помощи **ревью** и **тестовых прогонов. Ревью** представляет собой просмотр исходного кода *ПО* с целью обнаружения ошибок и дефектов, возможно, до того, как это *ПО* заработает. **Ревьюирование** предназначено для выявления таких ошибок, как неспособность выполнять то или иное **требование спецификации**или ее неправильное понимание, а также алгоритмических ошибок в реализации. **Тестовый прогон** обеспечивает тестирование *ПО* в процессе выполнения программы. Осуществляя прогон программы, *тестировщик* стремится определить, способна ли *программа* вести себя в соответствии со спецификацией. *Тестировщик* должен выбрать наборы входных данных, определить соответствующие им правильные наборы выходных данных и сопоставить их с реально получаемыми выходными данными.

Рассмотрим тестирование классов в режиме **прогона тестовых случаев**. После идентификации **тестовых случаев** для класса нужно реализовать **тестовый драйвер**, обеспечивающий прогон каждого тестового случая, и запротоколировать результаты каждого прогона. При тестировании классов **тестовый драйвер** создает один или большее число экземпляров тестируемого класса и осуществляет прогон тестовых

случаев. **Тестовый драйвер** может быть реализован как автономный тестирующий *класс*.

Кто, что, когда, как и в каком объеме?

Рассмотрим эти вопросы в контексте тестирования классов.

Кто выполняет тестирование? Обычно тестирование классов выполняют их разработчики. В этом случае время на изучение спецификации и реализации сводится к минимуму. Недостатком подхода является то, что если разработчик неправильно понял спецификации, то он для своей неправильной реализации разработает и "ошибочные" тестовые наборы.

Что тестировать? Необходимо удостовериться, что программный код класса в точности отвечает требованиям, сформулированным в его спецификации, и что он не делает ничего более.

В какой момент следует выполнять тестирование? План тестирования или хотя бы тестовые случаи должны разрабатываться после составления полной спецификации класса. Разработка тестовых случаев *по* мере реализации класса помогает разработчику лучше понять спецификацию. Тестирование класса должно проводиться до того, как возникнет необходимость использовать этот *класс* в других компонентах *ПО*. *Регрессионное тестирование* класса должно выполняться всякий раз, когда меняется реализация класса. *Регрессионное тестирование* позволяет убедиться в том, что разработанные и оттестированные функции продолжают удовлетворять спецификации после выполнения модификации *ПО*.

Как будет выполняться тестирование? Тестирование классов обычно выполняется путем разработки тестового драйвера, который создает экземпляры классов и окружает эти экземпляры соответствующей средой (тестовым окружением), чтобы стал возможен прогон соответствующего тестового случая. *Драйвер* посылает сообщения экземпляру класса в соответствии со спецификацией тестового случая, а затем проверяет *исход* этих сообщений. Тестовый *драйвер* должен удалять созданные им экземпляры тестируемого класса. *Статические элементы* данных класса также необходимо тестировать.

Какие объемы тестирования следует считать адекватными? *Адекватность* может быть измерена полнотой охвата тестами спецификации или реализации. Будем использовать оба способа.

Что тестировать?

Можно выделить два типа классов с точки зрения их взаимодействия с другими классами:

- примитивные классы;
- непримитивные классы.

Примитивный класс может порождать экземпляры, и эти экземпляры можно использовать без необходимости создания экземпляров каких-либо других классов, в том числе и данного класса. Такие объекты представляют собой простейшие компоненты системы и, несомненно, играют важную роль при выполнении любой программы. Тем не менее, в объектно- ориентированной программе существует сравнительно небольшое количество примитивных классов, которые реалистично моделируют объекты задачи и все отношения между этими объектами. Обычным явлением для хорошо спроектированных объектно-ориентированных программ является использование непримитивных классов. Основываясь на этой информации, определим, к какому типу относится каждый класс в нашем приложении (табл. 2.1).

Таблица 2.1. Типы Классов

Класс	Тип
TBearingParam	Примитивный
TAxleParam	Примитивный
TCommand	Примитивный
TLog	Примитивный
TCommandQueue	Непримитивный
TStore	Непримитивный
TTerminalBearing	Непримитивный
TTerminalAxle	Непримитивный
TModel	Непримитивный
MainForm	Непримитивный

В большинстве объектно-ориентированных языков члены класса имеют один из трех уровней доступа:

Public. Члены с доступом **public** доступны из любых классов. Они образуют интерфейс класса, которым будет пользоваться любой разработчик, использующий данный класс в своем приложении.

Private. Члены с доступом **private** доступны только внутри самого класса, то есть из его методов. Они являются частью внутренней реализации класса и недоступны стороннему разработчику.

Protected. Члены с доступом **protected** доступны из самого класса и из классов, являющихся его потомками, но недоступны извне. Использование этих методов возможно только при создании класса-потомка, расширяющего функциональность базового класса.

Таким образом, необходимость тестирования функциональности класса зависит от того, предоставляется ли им возможность наследования. Если класс является законченным (**final**) и не предполагает наследования, необходимо тестирование его **public** части (впрочем, классы **final** не содержат **protected** членов). Если же класс рассчитан на расширение за счет наследования, необходимо тестирование также его **protected** части.

Кроме того, во многих языках класс может содержать статические (**static**) члены, которые принадлежат классу в целом, а не его конкретным экземплярам. При наличии **public static** или **protected static** членов, кроме тестирования объектов класса, должно отдельно выполняться тестирование статической части класса.

Как тестировать?

Как уже упоминалось, для тестирования классов применяются **тестовые драйверы**. Существует несколько способов реализации тестового драйвера:

Тестовый драйвер реализуется в виде отдельного класса. Методы этого класса создают объекты тестируемого класса и вызывают их методы, в том числе статические методы класса. Таким способом можно тестировать public часть класса.

Тестовый драйвер реализуется в виде класса, наследуемого от тестируемого. В отличие от предыдущего способа, такому тестовому драйверу доступна не только public, но и protected часть.

Тестовый драйвер реализуется непосредственно внутри тестируемого класса (в класс добавляются диагностические методы). Такой тестовый драйвер имеет доступ ко всей реализации класса, включая private члены. В этом случае в методы класса включаются вызовы отладочных функций и агенты, отслеживающие некоторые события при тестировании.

В дальнейшем мы будем использовать первый способ при реализации драйверов.

При разработке **спецификации** класса можно задействовать один из следующих подходов:

Контрактный подход. Интерфейс определяется в виде обязательств отправителя и получателя, вступивших во взаимодействие. Операция определяется как набор обязательств каждой стороны, причем ответственность по отношению друг к другу соблюдается как отправителем, так и получателем.

Подход защитного программирования. Интерфейс определяется главным образом в терминах получателя. Операция возвращает результат запроса - успешное или неудачное выполнение по конкретной причине (например, по недопустимому входному значению). Другими словами, соответствующий получатель следит за тем, чтобы на вход не попали некорректные данные, т.е. проверяет правильность и допустимость входных данных, и после получения запроса сообщает отправителю результат обработки запроса.

Различие между **контрактным** и **защитным методами проектирования** распространяется и на тестирование. **Контрактное проектирование** возлагает большую ответственность на проектировщика, чем на программы поиска ошибок. Основное внимание во время тестирования взаимодействий в условиях контактного подхода уделяется проверке того, выполнены ли объектом-отправителем предусловия методов получающего объекта. Не допускается построение тестовых случаев, нарушающих эти предусловия. Обычно практикуется перевод объекта- получателя в некоторое заданное состояние, после чего инициируется выполнение тестового драйвера, по условиям которого объект- отправитель требует, чтобы объект-получатель находился в другом состоянии. Смысл подобной проверки заключается в том, чтобы установить, выполняет ли объект-отправитель проверку предусловий объекта-получателя, прежде чем отправить заранее неприемлемое сообщение, и корректно ли он прекращает свою работу.

Подробное описание тестового случая

Рассматривается пример тестов на C# для класса TCommand (приложение 3 (HLD)). При выполнении заданий необходимо будет самостоятельно написать тесты для других классов приложения. Параллельно с изучением этого раздела полезно открыть проект ModuleTesting\ModuleTests.sln.

Рассмотрим тестирование класса TCommand. Этот *класс* реализует единственную операцию GetFullName(), которая возвращает полное название команды в виде строки. Разработаем **спецификацию тестового случая** для тестирования метода GetFullName на основе спецификации этого класса (*приложение* 3):

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, где -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"

На основе спецификации был создан тестовый *драйвер* - *класс* TCommandTester, наследующий функциональность абстрактного класса *Tester*.

```
public class Log
{
        static private StreamWriter log=new
            StreamWriter("log.log"); //Создание лог файла
        static public void Add(string msg)
            //Добавление сообщения в лог файл
        {
            log.WriteLine(msg);
        }
        static public void Close() //Закрыть лог файл
        {
            log.Close();
        }
}
abstract class Tester
{
protected void LogMessage(string s)
//Добавление сообщения в лог-файл
        {
            Log.Add(s);
        }
}
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());
```

```
                }
            }
            [STAThread]
            static void Main()
            {
                TCommandTester CommandTester = new TCommandTester();
                Log.Close();
            }
}
```

Листинг 2.1. Тестовый драйвер

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, где -1 - запрещенное *значение*, и получить соответствующие им полное название команды с помощью метода GetFullName(). Пары соответствующих значений заносятся в *log-файл* для последующей проверки на соответствие спецификации.

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.

Описание тестовых процедур

Как запустить тест?

Для того чтобы запустить тест, нужно:

- В методе Run тестового драйвера TCommandTester вызвать метод TCommandTest1, реализующий тест.
- Собрать и запустить приложение.

Проверка результатов выполнения тестов (сравнение с ожидаемым результатом)

После завершения теста следует просмотреть текстовый журнал теста (..\ModuleTesting\bin\Debug\log.log), чтобы сравнить полученные результаты с ожидаемыми результатами, заданными в спецификации тестового случая TCommandTest1. Журнал теста:

-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

Задание 1

Для остальных примитивных классов (табл. 2.1) в соответствии с приведенным примером необходимо самостоятельно разработать спецификации тестовых случаев, соответствующие тесты и провести тестирование. Отчет требуется составить в следующей форме (табл. 2.2):

Таблица 2.2. *Тестовый отчет*

Название тестового случая: Тестировщик: Тест пройден: Да/Нет (PASS/FAIL) Степень важности ошибки: Фатальная (3 уровень - *crash*) Серьезная (2 уровень - расхождение в спецификации) Незначительная (1 уровень - незначительная ошибка) Описание проблемы: Как воспроизвести ошибку: Предлагаемое исправление (необязательно): Комментарий тестировщика (необязательно):