Приложение

Учреждение образования

"Брестский государственный технический университет"

# Кафедра «ЭВМ и системы»

**ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ПРОВЕДЕНИЯ ЛАБОРАТОРНОГО ПРАКТИКУМА ПО СИСТЕМАМ КОНТРОЛЯ ВЕРСИЙ**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

**БрГТУ.100529-07 95 00**

## Листов 36

|  |  |
| --- | --- |
| Заведующий кафедрой | С. С. Дереченник |
| Руководитель | Четвёркина Г. А. |
|  |  |
| Выполнил | Самсонов Д. С. |
| Нормоконтроль | В. С. Разумейчик |

2015

БрГТУ.100529-07 95 00

**Аннотация**

Данное приложение служит основой для методических указаний к проведению лабораторного практикума по тестированию программного кода. В лабораторный практикум вошли модульное тестирование, тестирование графического пользовательского интерфейса и нагрузочное тестирование. Приложение содержит пять лабораторных работ, в которых приведен теоретический материал для выполнения предложенных заданий.

БрГТУ.100529-07 95 00

**Содержание**

1 Лабораторная работа №1. Основы модульного тестирования. Знакомство

со средой NUnit 4

1.1 Модульное тестирование 4

1.2 Среда NUnit 5

1.3 Создание тест-метода 6

1.4 Порядок выполнения работы 12

2 Лабораторная работа №2 «NUnit. Использование атрибутов» 13

2.1 Использование атрибутов 13

2.3 Порядок выполнения работы 16

3 Лабораторная работа №3 «Основы тестирования GUI. Знакомство с

библиотекой UI Automation» 17

3.1 Тестирование графического пользовательского интерфейса 17

3.2 Библиотека UIAutomation 18

3.3 Создание скрипта 23

3.4 Порядок выполнения работы 23

4 Лабораторная работа №4 «Разработка скрипта для формы с

использованием библиотеки UIAutomation» 24

4.1 Изучение паттернов 24

4.2 Порядок выполнения работы 26

5 Лабораторная работа №5 «Нагрузочное и функциональное тестирование в

JMeter» 27

5.1 Надежность информационной системы как элемент обеспечения безопасности 27

5.2 Метод нагрузочного тестирования для проверки надежности и безопасности

систем 28

5.3 Практическое применение инструментов для нагрузочного

тестирования телекоммуникационных информационных систем 28

5.4 Обзор JMeter 30

5.5 Создание тест-плана 33

5.6 Порядок выполнения работы 35

БрГТУ.91402-07 95 00 4

**1 ЛАБОРАТОРНАЯ РАБОТА №1 «МОДУЛЬНОЕ ТЕСТИРОВАНИЕ. ЗНАКОМСТВО СО СРЕДОЙ NUNIT»**

Цель работы: изучить принципы модульного тестирования, ознакомиться со средой NUnit, изучить создание тест-методов в среде NUnit.

**1.1 Модульное тестирование**

Модульное тестирование (или unit-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.

Именно эффективность обнаружения тех или иных типов дефектов должна определять стратегию модульного тестирования, то есть расстановку акцентов при определении набора входных значений. У организации, занимающейся разработкой программного обеспечения, как правило, имеется историческая база данных разработок, хранящая конкретные сведения о разработке предыдущих проектов: о версиях и сборках кода, зафиксированных в процессе разработки продукта, о принятых решениях, допущенных просчетах, ошибках, успехах и т.п. Проведя анализ характеристик прежних проектов можно предохранить новую разработку от старых ошибок, например, определив типы дефектов, поиск которых наиболее эффективен на различных этапах тестирования.

В данном случае анализируется этап модульного тестирования. Если анализ не дал нужной информации, например, в случае проектов, в которых соответствующие данные не собирались, то основным правилом становится поиск локальных дефектов, у которых код, ресурсы и информация, вовлеченные в дефект, характерны именно для //разрыв

БрГТУ.91402-07 95 00 5

данного модуля. В этом случае на модульном уровне ошибки, связанные, например, с неверным порядком или форматом параметров модуля, могут быть пропущены, поскольку они вовлекают информацию, затрагивающую другие модули (а именно, спецификацию интерфейса), в то время как ошибки в алгоритме обработки параметров довольно легко обнаруживаются.

Модульные тесты, или юнит-тесты – это просто фрагмент кода, который проверяет другой фрагмент кода и выдает ответ на вопрос: ведет ли себя проверяемый фрагмент как задумано или нет.

Один из наиболее эффективных подходов к модульному тестированию - это подготовка автоматизированных тестов до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешно пройдены.

В разработке через тестирование используется цикл из следующих шагов:

1) написание теста, который будет выдавать ошибку. Это подтвердит, что тест правильный. Для написания теста разработчик должен полностью разобраться в спецификации и требованиях;

2) написание кода. Создается только минимальный код, достаточный для выполнения теста;

3) прогон автоматизированных тестов. Все тесты должны работать;

4) рефакторинг;

5) повторение.

Цикл TDD процесса изображен на рисунке 1.1.

Важный вопрос при разработке модульных тестов: когда остановиться? В какой момент можно сказать, что код протестирован полностью? Однозначного ответа на этот вопрос нет, но есть некоторое количество методик, которые позволяют определить, что комплект тестов является неполным.

**1.2 Среда NUnit**

NUnit – это развивающееся среда, с открытым исходным кодом предназначенная для написания и запуска тестов на языках программирования Microsoft .NET. Среда является одним из аспектов test-driven development, TDD (с англ. «разработка через тестирование»).

Среда NUnit имеет графический пользовательский интерфейс (GUI). Тесты могут выполняться непрерывно, а результаты тестов предоставляются сразу после их завершения. Одновременно могут выполняться сразу несколько тестов. Результаты тестов не требуют иных субъективных человеческих суждений или интерпретаций. Простота среды дает возможность легко исправить найденные ошибки.

БрГТУ.91402-07 95 00 6

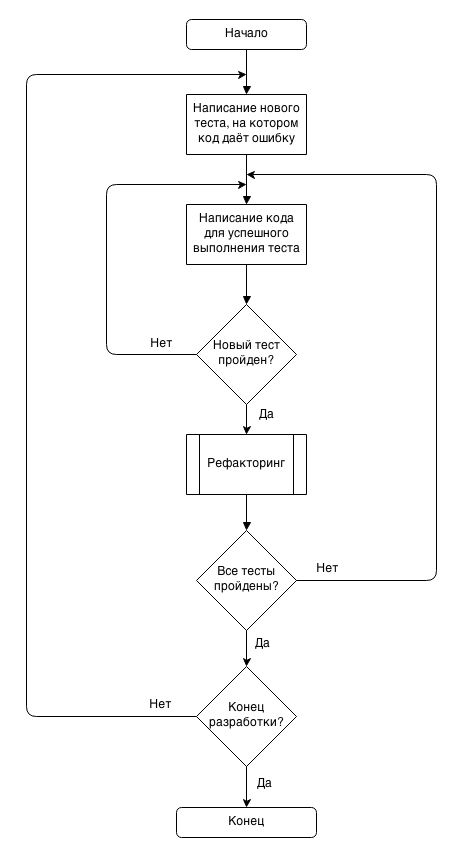


Рисунок 1.1 - Цикл TDD

Принцип работы среды NUnit заключается в использовании утверждений. Утверждение – основная операция unit-тестирования.

Утверждения (англ. assertions) – это  предикат, размещённый в программе и указывающий на то, что разработчик имеет в виду этот предикат в этом месте программы всегда истинным.

Иначе говоря, если утверждение не выполняется, тест будет проваленным. Так, например, есть функция, которая возвращает значение true. Если функция возвратит значение false, то тест будет проваленным.

Среда NUnit предоставляет набор утверждений как статических методов класса Assert. Если утверждение не выполняется, то вызов метода не возвращается и выдается сообщение об ошибки. Если тест-метод содержит несколько утверждения, то любое утверждение, следующее за проваленным утверждением, не запуститься. По этой причине лучше использовать одно утверждение на один тест.

Список используемых утверждений представлен в таблице 1.1.

**1.3 Создание тест-метода**

Для создания простейшего unit-теста необходимо к разрабатываемому тесту подключить библиотеку проекта nunit.framework и соответствующее пространство имён: NUnit.Framework.

БрГТУ.91402-07 95 00 7

Таблица 1.1 – Методы классов Asserts, StringAssert, FileAsserts, CollectionAssert

|  |  |  |
| --- | --- | --- |
| Утвержде-ние | Методы | Описание |
| Equality Asserts | AreEqual  AreNotEqual | Методы сравнения двух аргументов на равенство или соответственно на неравенство. |
| Identity Asserts | AreSame  AreNotSame  Contains | Методы идентификации объекта по ссылке и метод идентификации объекта в приведенном списке объектов. |
| Comparison Assert | Greater  GreaterOrEqual  Less  LessOrEqual | Методы сравнения двух объектов на значение больше, больше или равно, меньше, меньше или равно. Утверждения читаются в «математическом» порядке, например, Greater(x, y) как (x > y); |
| Type Asserts | IsInstanceOf<T>  IsNotInstanceOf<T>  IsAssignableFrom<T>  IsNotAssignableFrom<T> | Методы, утверждающие тип объекта. |
| Condition Tests | True  False  Null  NotNull  IsEmpty  IsNotEmpty  IsNaN | Методы, подтверждающие выполнение условия. |
| Exception Asserts | Throws  Throws<T>  DoesNotThrow  Catch  Catch<T> | Метод Throws пытается вызвать фрагмент кода, представленный в качестве делегата, для того, чтобы убедиться, что тот бросает определенное исключение. Метод возвращает исключение, если утверждение является успешным. Метод DoesNotThrow просто проверяет, что делегат не исключение. Catch похож на Throws, но пойдёт для исключения, которое является производным от указанного. |

БрГТУ.91402-07 95 00 8

Продолжение таблицы 1.1

|  |  |  |
| --- | --- | --- |
| Утвержде-ние | Методы | Описание |
| Utility Methods | Pass  Fail  Ignore  Inconclusive | Методы, обеспечивающие прямой контроль процессом тестирования. Метод Pass позволяет немедленно прекратить тест, записав его как успешный. Поскольку метод позволяет бросить исключение, он эффективнее чем простое завершение тест-метода. Тем не менее, Pass позволяет записать сообщение в результат теста, а так же в некоторых случаях может сделать тест более читаемым. Метод может быть вызван из вложенного вызова метода с немедленным успешным прекращением выполнения теста. Метод Fail предоставляет возможность генерировать сбой на основании результатов теста, которые не инкапсулированы другими методами. Это может быть полезно в разработке собственных утверждений. Метод Ignore предоставляет возможность игнорировать тест или набор тестов во время выполнения. Он может быть вызван в тест-методе или в методе тестовой установки. Метод Inconclusive указывает, что тест не может быть завершен с имеющимися данными. |
| String Asserts | Contains  StartsWith  EndsWith  AreEqualIgnoringCase  IsMatch | Класс StringAssert предоставляет методы сравнения строк. |
| File Asserts | AreEqual  AreNotEqual | Класс FileAsserts предоставляет методы для сравнения двух файлов, которые могут быть предоставлены в виде потоков, как FileInfos или как путь к файлу в виде строки. |

БрГТУ.91402-07 95 00 9

Завершение таблицы 1.1

|  |  |  |
| --- | --- | --- |
| Утвержде-ние | Методы | Описание |
| Collection Asserts | AllItemsAreInstancesOfType  AllItemsAreNotNull  AllItemsAreUnique  AreEqual  AreEquivalent  AreNotEqual  AreNotEquivalent  Contains  IsSubsetOf  IsNotSubsetOf  IsEmpty  IsNotEmpty  IsOrdered | Класс CollectionAssert предоставляет ряд методов, которые полезны для проверки коллекции и их содержимого или для сравнения двух коллекций. Метод AreEqual выполнится успешно, если соответствующие элементы двух коллекциях равны. Метод AreEquivalent сравнивает, совпадает ли содержимое коллекций, но без учета порядка следования элементов. |

Класс, содержащий тестовый код, должен иметь директиву public и иметь атрибут TestFixture. Пример класса изображен на рисунке 1.2.

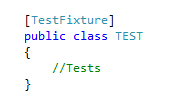


Рисунок 1.2 – Класс, содержащий тестовый код

Приведем пример тест-метода для функции умножения двух целых неотрицательных чисел.

Определим требования для реализации заданной функции умножения:

− так как умножение будет проводиться над двумя неотрицательными числами, то результат должен быть неотрицательный;

− так как умножение будет проводиться над целыми числами, то результат должен представлять целое число;

− результатом заданной функции должно быть умножение.

Объявим функция умножения двух чисел. Пример объявления представлен на рисунке 1.3.

Определим тестирующую функцию, которая будет непосредственно вызывать функцию Multiplication(int A, int B). Пример такого тест-метода изображен на рисунке 1.4.

БрГТУ.91402-07 95 00 10

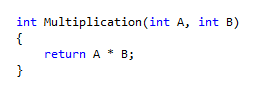


Рисунок 1.3 – Пример объявления функции умножения двух чисел

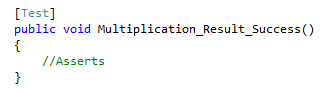


Рисунок 1.4 – Пример тестирующего тест-метода

Как видно из примера на рисунке 1.4 перед объявлением функции определен атрибут Test, который указывает на то, что это метод теста. Тест-метод не должен ничего возвращать и принимать какие-либо параметры. Тест-метод нужен для инициализации необходимых объектов, выполнения теста и проверки результата.

Дополним тест-метод тестовыми данными и утверждениями для проверки корректности результата. Для этого:

− определим тестовые данные, которые буду представлять собой два целочисленных неотрицательных числа;

− определим ожидаемый результат, который будет равен произведению заданных выше чисел;

− проверим, удовлетворяют ли предложенные тестовые данных нашим требованиям.

Пример обновленной функции изображен на рисунке 1.5.

Как показано на рисунке 4 тест-метод содержит следующие утверждения:

− IsTrue(bool condition) для проверки выполнения условия на не отрицательность результата;

− IsInstanceOf<type>(object actual) для проверки типа полученного результата;

− AreEqual(int expected, int actual) для проверки полученного результата.

Для запуска тест-метода необходимо скомпилировать текст программы, а затем открыть полученный exe-файл в среде NUnit. После успешного открытия exe-файла NUnit представит все тест-методы тестирующего класса в виде дерева. Пример открытого файла приведен на рисунке 1.6. Как видно дерево включает в себя:

− путь к exe-файлу;

− наименование пространства имен;

− наименование класса, в котором содержится тест метод;

− наименование тест-метода.

БрГТУ.91402-07 95 00 11

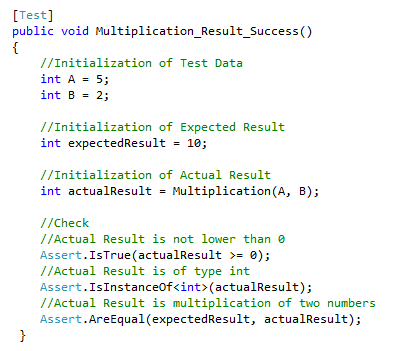


Рисунок 1.5 – Тестирующая функция

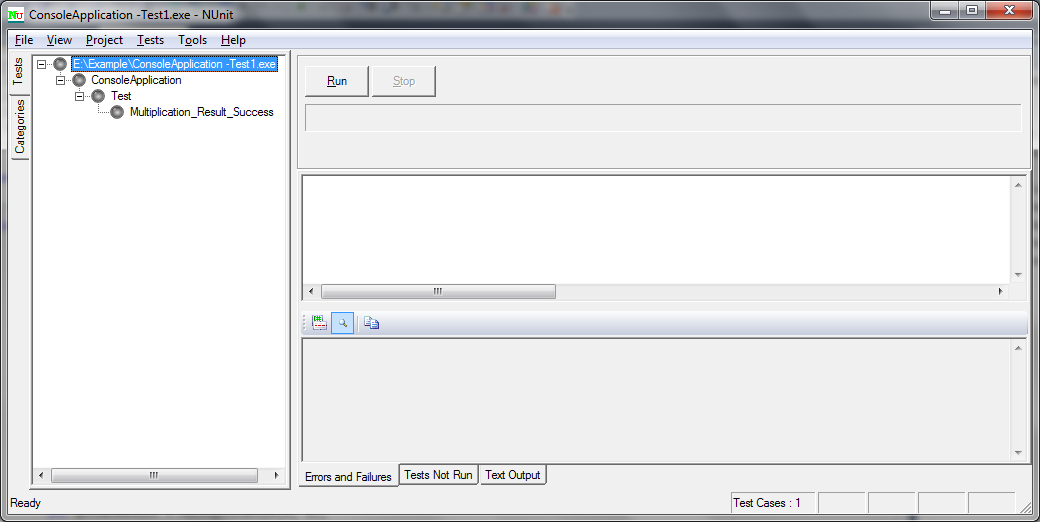


Рисунок 1.6 – Открытый exe-файл с тест-методом в среде NUnit

Для запуска тест-метода необходимо нажать на кнопку Run. Все определившиеся в дереве тест методы будут запущены последовательно. После завершения каждого тест-метода он будет помечен соответственно:

− при успешном прохождении тест-метод будет помечен зелёным цветом;

− при провальном прохождении тест-метод будет помечен в красный цвет.

Запустим тест. Пример результата отображен на рисунке 1.7. Как видно, тест успешный.

Приведем пример «неуспешного» тест-метода. Для этого модернизируем функцию Multiplication(int A, int B), изменив знак умножения «\*» на знак сложения «+».

Пример обновленной функции изображен на рисунке 1.8. Скомпилируем измененный текст программы и запустим тест-метод в среде NUnit.

БрГТУ.91402-07 95 00 12

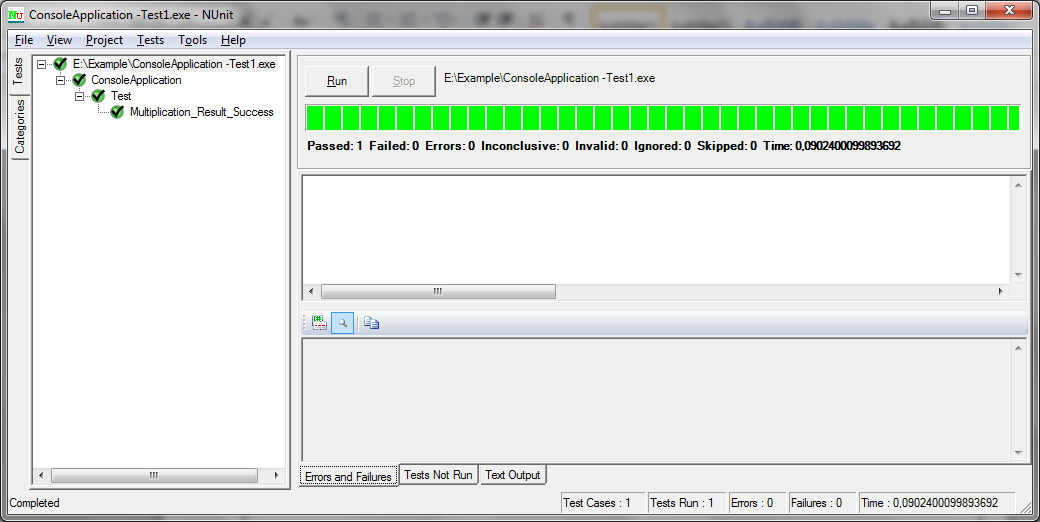


Рисунок 1.7 – Пример успешного выполнения теста

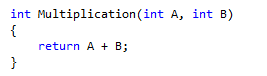


Рисунок 1.8 – Пример модернизированной функции Multiplication(int A, int B)

Результат выполнения отображён на рисунке 1.9. Как видно из примера тест не пройден. Проваленные тесты содержат информацию об ожидаемом и действительном результатах. В данном случае ожидаемым результатом было число 10, а действительным – число 7.

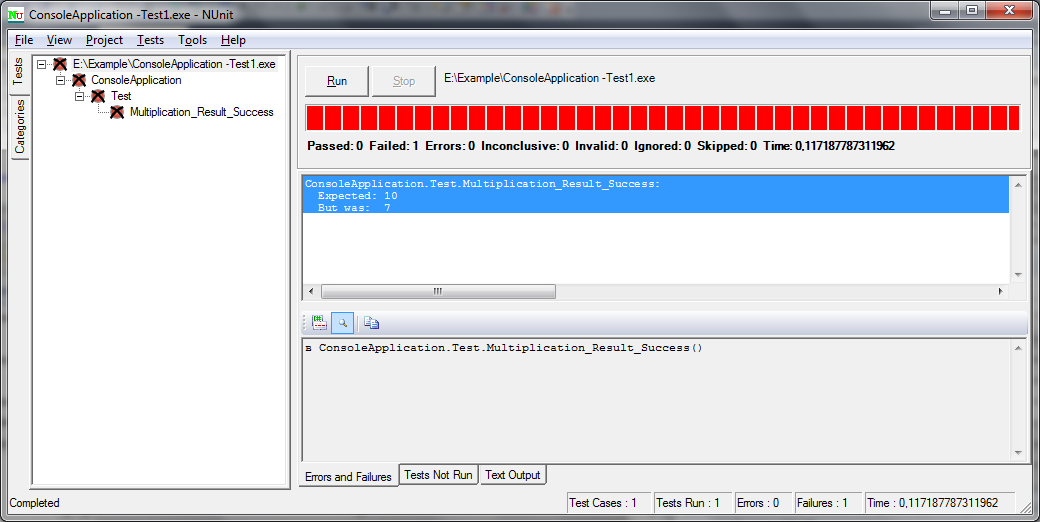


Рисунок 1.9 – Пример проваленного теста

**1.4 Порядок выполнения работы**

1) Определите функцию согласно варианту.

2) Разработайте полный набор тест-методов для функции, реализованной в пункте 1, используя библиотеку nunit.framework.

3) Убедитесь в выполнении всех разработанных тестов. В противном случае повторите выполнения шагов.

Для уточнения варианта задания обратитесь к вашему преподавателю

БрГТУ.91402-07 95 00 13

**2 ЛАБОРАТОРНАЯ РАБОТА №2 «NUNIT. ИСПОЛЬЗОВАНИЕ АТРИБУТОВ»**

Цель работы: углубить полученные знания использования unit-тестирования.

**2.1 Использование атрибутов**

Определения утверждений в тест-методах не достаточно. Тесты должны быть отделены от основного кода программы и находиться в отдельных сборках. В этих сборках должны быть классы, помеченные атрибутом TestFixture (тестовый набор). Сами тесты – это методы таких классов. Они помечаются атрибутом Test. Пример структуры тестового набора приведен на рисунке 2.1. Сборка с тестами открывается через основную программу NUnit и все тесты из нее запускаются. Программа NUnit выдает результаты о прогоне тестов.

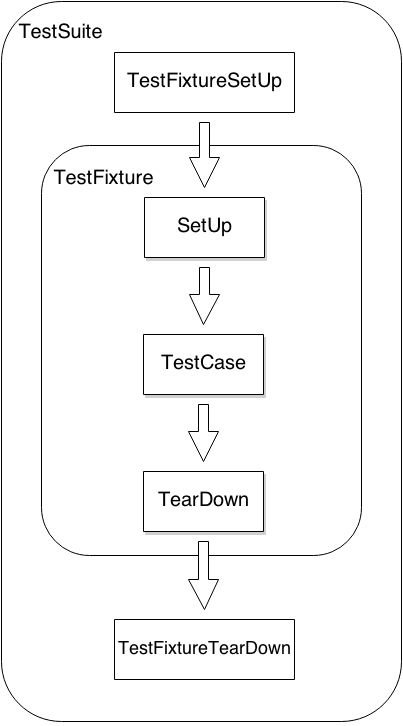


Рисунок 2.1 – Структура тестового набора

Тестовый набор, изображенный на рисунке 2.1, включает блоки SetUp и TearDown. Как правило, их используют для создания и уничтожения экземпляров объектов, подлежащих тестированию. Блок SetUp будет выполняться перед каждым запуском тестового метода, а блок TearDown – соответственно после.

Так же применяются такие атрибуты как:

− атрибут Category предоставляет альтернативу test suite для работы с группами тестов. Отдельные тестовые случаи или их множество могут быть закреплены к определенной категории. При прогоне тестов можно определить категорию испытаний. Будет запущены только те тесты, которые включены в выбранные категории. Остальные тест-методы, не включенные в категорию не будут запущено вообще;

БрГТУ.91402-07 95 00 14

− атрибут Description применяется для описания тест-методов и TestFixture. Текст описания появится в выходном файле XML и будет показан в диалоговом окне Test Propertie;

− атрибут ExpectedException указывает, что при выполнении теста будет сгенерировано исключение. Этот атрибут имеет ряд позиционных и именованных параметров, например, в качестве параметра можно передать тип ожидаемого исключения. В качестве второго передаваемого параметра ExpectedMessage можно определить текст ожидаемого текста при исключении. Третьим параметром MatchType можно определить тип соответствия второго параметра (Exact – полное совпадение, Contains – содержание второго параметра в ожидаемом сообщении, Regex – регулярное выражение предоставлено в качестве параметра, StartsWith – ожидаемое сообщение начинается со второго параметра). На рисунке 2.2 представлен пример определения атрибута ExpectedException;

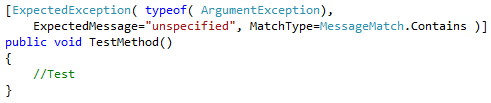


Рисунок 2.2 – Определение атрибута ExpectedException

− атрибут Ignore позволяет не выполнять тест или тестовый набор при его определении. Тогда при запуске тест-методов не будут запущены тестовые сценарии, отмеченные атрибутом Ignore. При этом индикатор выполнения будет желтым. Этот атрибут должен использоваться, чтобы временно не запускать тесты или их наборы;

− атрибут Random используется для указания набора случайных значений, которые будут предоставлены в качестве индивидуального параметра параметризованному тест-методу. При этом тест метод будет повторен с каждым набором параметров. Так, например, тест-метод, изображенный на рисунке 2.4, будет выполнен 5 раз со случайным вещественным числом в промежутке [-1;1];

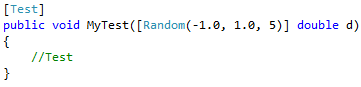


Рисунок 2.3 – Определение атрибута Random

− атрибут Range используется для указания диапазона значений, который будет представлен ​​для отдельного параметра параметризованному тест-методу. По умолчанию

БрГТУ.91402-07 95 00 15

NUnit создает тесты из всех возможных комбинаций данных, предоставленных в качестве параметра. Примеры поддерживаемых конструкторов для атрибута Range приведены на рисунке 2.4;

− атрибут TestCase служит двойной целью маркировки параметризованного тест-метода, параметры которого будут использоваться при его вызове. На рисунке 2.5 приведен пример тест-метода, который будет запущен три раза с различными наборами данных.

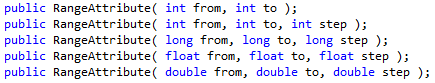


Рисунок 2.4 – Конструкторы атрибута Range

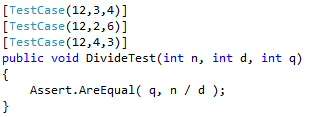


Рисунок 2.5 – Определение атрибута TestCase

Используя параметр Result можно упростить приведенный пример на рисунке 2.5. Упрощенный вызов тест-метода показан на рисунке 2.6.

Атрибут TestCase поддерживается рядом дополнительных параметров, таких как: Description, ExpectedException, ExpectedMessage, Ignore, Result и других;

− атрибут Values используется для указания набор значений, которые будут предоставлены в качестве индивидуального параметра параметризованному тест-методу. В примере, приведенном на рисунке 2.7, тест-метод будет выполнен 6 раз, принимая следующие параметры: (1, «A»), (1, «В»), (2, «A»), (2, «В»), (3, «A»), (3, «В»).

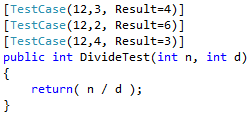


Рисунок 2.6 – Возможности определения атрибута TestCase

БрГТУ.91402-07 95 00 16

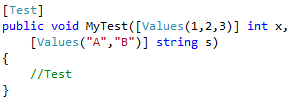


Рисунок 2.8 – Определение атрибута Values

**2.2 Порядок выполнения работы**

1) Определите класс согласно варианту.

2) Разработайте полный набор тест-методов для класса, определенного в пункте 1, используя библиотеку nunit.framework.

3) Модифицируйте тест-методы, используя изученные атрибуты.

4) Убедитесь в выполнении всех разработанных тестов. В противном случае повторите выполнения шагов.

Для уточнения варианта задания обратитесь к вашему преподавателю

БрГТУ.91402-07 95 00 17

**3 ЛАБОРАТОРНАЯ РАБОТА №3 «ОСНОВЫ ТЕСТИРОВАНИЯ GUI. ЗНАКОМСТВО С БИБЛИОТЕКОЙ UI AUTOMATION»**

Цель работы: ознакомиться с возможностями тестирования графического пользовательского интерфейса.

**3.1 Тестирование графического пользовательского интерфейса**

Графический интерфейс пользователя (Graphical User Interface, GUI) –разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

Задачей тестирования GUI является обнаружение ошибок следующего характера:

− ошибки в функциональности посредством интерфейса;

− необработанные исключения при взаимодействии с интерфейсом;

− потеря или искажение данных, передаваемых через элементы интерфейса;

− ошибки в интерфейсе (несоответствие проектной документации, отсутствие элементов интерфейса).

Особенности тестирования пользовательского интерфейса:

− тест-планы для проверки пользовательского интерфейса, как правило, представляют собой сценарии, описывающие действия пользователя при работе с системой;

− сценарии могут быть записаны либо на естественном языке, либо на формальном языке какой-либо системы автоматизации пользовательского интерфейса;

− выполнение тестов при этом производится либо оператором в ручном режиме, либо системой, которая эмулирует поведение оператора;

− при сборе информации о выполнении тестовых примеров обычно применяются технологии анализа выводимых на экран форм и их элементов (в случае графического интерфейса) или выводимого на экран текста (в случае текстового), а не проверка значений тех или иных переменных, устанавливаемых программной системой;

− под полнотой покрытия пользовательского интерфейса понимается то, что в результате выполнения всех тестовых примеров каждый интерфейсный элемент был использован хотя бы один раз во всех доступных режимах;

− отчеты о проблемах в пользовательском интерфейсе могут включать в себя как описания несоответствий требований и реального поведения системы, так и описания проблем в требованиях к пользовательскому интерфейсу.

Функциональное тестирование GUI состоит из пяти фаз:

− анализ требований к пользовательскому интерфейсу;

− разработка тест-требований и тест-планов для проверки пользовательского интерфейса;

− выполнение тестовых примеров и сбор информации о выполнении тестов;

БрГТУ.91402-07 95 00 18

− определение полноты покрытия пользовательского интерфейса требованиями;

− составление отчетов о проблемах в случае несовпадения поведения системы и требований либо в случае отсутствия требований на отдельные интерфейсные элементы.

Функциональное тестирование пользовательского интерфейса может проводиться различными методами:

− ручное тестирование (контроль проводится человеком);

− автоматическое тестирование (используются программные инструменты, эмулирующие поведение пользователя).

Плюсы ручного тестирования:

− контроль корректности проводится человеком;

− поиск «косметических» дефектов;

− анализ успешности прохождения теста будет выполняться не по формальным признакам, а согласно человеческому восприятию.

Минусы ручного тестирования:

− требуются значительные человеческие и временные ресурсы;

− при проведении регрессионного тестирования и вообще любого повторного тестирования – на каждой итерации повторного тестирования пользовательского интерфейса требуется участие специалиста по тестированию.

Плюсы автоматического тестирования:

− снижение стоимости тестирования;

− высокая скорость выполнения;

− больший объем покрытия;

− не требуется участие специалиста по тестированию при проведении регрессионного тестирования или любого другого повторного тестирования продукта.

Минусы автоматического тестирования:

− анализ успешности прохождения теста будет выполняться по формальным признакам;

− невозможность поиска «косметических» дефектов;

− высокая стоимость поддержки по сравнению с «обычными» функциональными тестами.

**3.2 Библиотека UIAutomation**

Для тестирования приложений Win32, приложений .NET Windows Forms, а также приложений WPF на компьютерах, использующих операционные системы с поддержкой .NET Framework 3.0, можно использовать библиотеку UIAutomation. Microsoft UIAutomation является доступной средой для Microsoft Windows и доступна на всех операционных системах, которые поддерживают Windows Presentation Foundation (WPF). UI Automation обеспечивает программный доступ к большинству элементам пользовательского интерфейса (UI) на рабочем столе.

БрГТУ.91402-07 95 00 19

Основная идея UI Automation заключается в обеспечении доступа к иерархии элементов пользовательского интерфейса, которые видны в Win приложениях. Если представить себе рабочий стол как корневой элемент иерархии, то все запущенные приложения будут являться узлами первого уровня корня, и то же самое для каждой видимой формы, которая может быть детализирована до её элементов управления и так далее. Каждый узел в этом дереве называется элементом пользовательского интерфейса. Пример такой структуры изображен на рисунке 3.1.

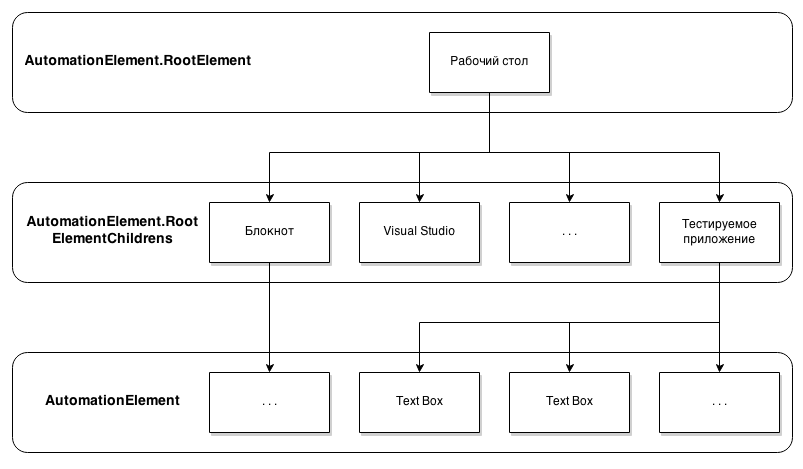


Рисунок 3.1 – Иерархия элементов пользовательского интерфейса

UI Automation даёт доступ к элементам пользовательского интерфейса и их свойствам. Основные поддерживаемые функции:

− поиск поддержки, которая позволяет найти нужный компонент пользовательского интерфейса;

− фильтрация элементов древовидной структуры, например, при запросе иерархии элементов, можно получить доступ только к разрешенным элементам управления;

− взаимодействие с элементами пользовательского интерфейса, например, можно программно нажать кнопку (которая содержится в другом приложении), из разрабатываемого приложения;

− подписка на события, что помогает отслеживать элементы пользовательского интерфейса и обработки внешних событий.

**3.3 Создание скрипта**

Для использования UI Automation в тестирующее приложение необходимо добавить ссылки проекта на библиотеки UIAutomationClient.dll и UIAutomationTypes.dll.

БрГТУ.91402-07 95 00 20

Эти библиотеки являются частью .NET Framework 3.0 и обычно находятся в каталоге %PROGRAMFILES%\Reference Assemblies\Microsoft\Framework\v3.0.

Как правило, при написании автоматизированных тестов с использованием библиотеки UI Automation доступ к исходному коды тестируемой системы не требуется. Доступ к элементам управления в большинстве случаев осуществляется через их подписи, а не по их внутреннему свойству Name.

Создание автоматического теста с использованием библиотеки UI Automation включает в себя:

1) вызов тестируемого приложения;

2) получение ссылки на форму;

3) получение ссылок на все пользовательские элементы управления;

4) ввод тестовых данных;

5) проверка результата;

6) завершение работы с тестируемой формой.

Чтобы вызвать тестируемую форму из консольного приложения можно использовать метод Process.Start и передать путь к форме (файлу формата exe) в качестве параметра.

При автоматизации тестирования пользовательского интерфейса, необходимо думать о каждой визуальной сущности (элементах управления, окнах и так далее) на рабочем компьютере как о части иерархического дерева с окном рабочего стола в качестве корневого элемента. Это общий принцип для большинства способов автоматизации тестирования пользовательского, включая использующие библиотеку UI Automation. Поэтому, чтобы получить ссылку на тестируемое приложение, нужно узнать ссылку на окно рабочего стола. А затем по ссылке на окно рабочего стола – ссылку на окно тестируемого приложения. Большинство объектов при автоматизации тестов с использованием библиотеки UI Automation принадлежат к типу AutomationElement. Таким образом, можно получить ссылку на визуальный элемент верхнего уровня («Рабочий стол») используя RootElement – статичное свойство AutomationElement.

Далее необходимо получить ссылку на тестируемое приложение. Для этого после открытия тестируемой формы можно вызвать метод FindFirst объекта ссылки на окно рабочего стола.

Пример получения ссылки на форму приведен на рисунке 3.2.

Как видно из примера метод FindFirst ищет первую доступную ссылку на AutomationElement, соответствующую указанному условию. Первый переданный аргумент TreeScope.Children указывает, что просматривать необходимо только дочерние элементы управления контекста (в данном случае aeDesktop). Второй аргумент описывает условие, которое можно интерпретировать как «Элемент со свойством Name, равным UI Test».

Так же, для получения ссылки на элемент можно использовать метод FindAll. В этом случае метод возвратит коллекцию элементов управления на текущем уровне, удовлетворяющую заданному условию.

БрГТУ.91402-07 95 00 21

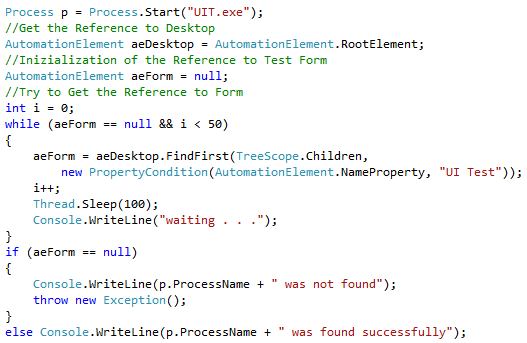


Рисунок 3.2 – Получение ссылки на форму

Метод получения ссылок на пользовательские элементы управления идентичен методу получения ссылки на форму. Пример получения ссылок на пользовательские элементы управления, такие как текстовое поле и кнопку представлены на рисунках 3.3 и 3.4 соответственно.



Рисунок 3.3 – Получение ссылки текстового поле



Рисунок 3.4 – Получение ссылки на кнопку

Как показано на рисунках 3.3 и 3.4 вместо NameProperty используется ControlTypeProperty со значением ControlType.Edit и ControlType.Button. Используемое условие ищет первый встречающийся элемент управления заданного типа (ControlType.Edit для текстового поля и ControlType.Button для кнопки).

Как можно будет убедиться, при написании средства автоматизации тестов с использованием библиотеки UI Automation, важна глубина дерева элемента управления. При наличии доступа к исходному коду тестируемого приложения, подразумеваемый порядок индексов для элементов управления установить несложно.

Однако если доступа с исходному коду нет, то порядок следования элементов управления можно узнать с помощью средства Spy++, поставляемого с Visual Studio. Пример использования Spy++ показан на рисунке 3.5.

БрГТУ.91402-07 95 00 22

После получения ссылок на все пользовательские элементы управления можно перейти непосредственно к тестированию – вводу тестовых данных. Однако изменения состояний элемента управления (ввод данных, выбор/снятие чекбокса, выбор значения из списка и др.) происходит с помощью использования, так называемых, паттернов – шаблона или образца. Объекты шаблонов можно представить себе как абстрактный способ предоставления функций элемента управления, независимых от его типа, или облика. Другими словами, конкретные экземпляры AutomationPattern, такие как ValuePattern, можно использовать для предоставления функций конкретного элемента управления.

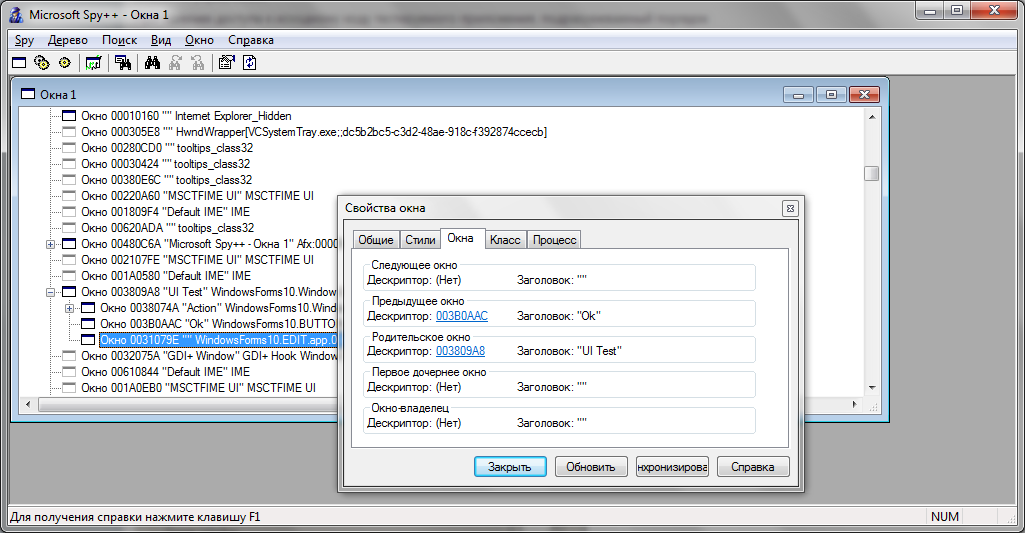


Рисунок 3.5 – Использование Spy++ в поиске подразумеваемого порядка индексов

Иначе говоря, ControlType элемента управления раскрывает, к какому роду элементов управления принадлежит данный элемент, а его Pattern раскрывает, что он может сделать. Важно отметить, что элемент управления может поддерживать несколько шаблонов.

Так, например, нажатие не кнопку можно выполнить используя паттерн InvokePattern. Пример использования представлен на рисунке 3.5.

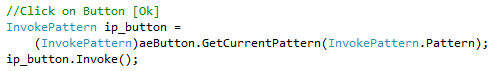


Рисунок 3.5 – Нажатие на кнопку

БрГТУ.91402-07 95 00 23

**3.4 Порядок выполнения работы**

Разработать скрипт, который:

− запускает калькулятор;

− вводит заданное выражение;

− получает результат выражения;

− сравнивает полученный результат с эталонным значением;

− выводит результат проверки.

Варианты к выполнению:

1)

2)

3) ;

4) ;

5) .

БрГТУ.91402-07 95 00 24

**4 ЛАБОРАТОРНАЯ РАБОТА №4 «РАЗРАБОТКА СКРИПТА ДЛЯ ФОРМЫ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ UIAUTOMATION»**

Цель работы: углубить знание использования библиотеки UIAutomation, изучить паттерны элементов управления.

**4.1 Изучение паттернов**

В лабораторной работе №3 были получены навыки автоматизированного тестирования приложения «Калькулятор», в котором ввод данных производился по нажатию кнопок. В данной лабораторной работе будет использована форма, содержащая не только кнопки, но и такие элементы управления как текстовые поля, выпадающие списки, чекбоксы и другие.

Как можно было убедиться в предыдущей лабораторной работе, что изменение состояний элемента управления (ввод данных, выбор/снятие чекбокса, выбор значения из списка и др.) происходит с помощью использования паттернов – шаблона или образца. Понимание классов AutomationPattern является ключевой частью умения проверять тестируемое приложение, используя библиотеку Microsoft UI Automation. Существуют 18 типов AutomationPattern, которые перечислены в таблице 4.1.

Таблица 4.1 – Типы паттернов AutomationPattern

|  |  |
| --- | --- |
| Класс | Описание |
| DockPattern | Используется для элементов управления, которые могут быть состыкованы в контейнере стыковки. Например, панели инструментов или средства палитры. |
| ExpandCollapsePattern | Используется для элементов управления, которые могут быть развернуты или свернуты. Например, элементы меню в приложении, такие как меню Файл. |
| GridPattern | Используется для элементов управления, которые поддерживают функциональные возможности сетки, такие как изменение размера и перемещение в указанную ячейку. Например, режим крупных значков в проводнике Windows или простые таблицы без заголовков в Microsoft Word. |
| GridItemPattern | Используется для элементов управления, имеющих ячейки в сетке. Отдельные ячейки должны поддерживать шаблон GridItem. Например, каждая ячейка в подробном представлении проводника Microsoft Windows. |

БрГТУ.91402-07 95 00 25

Продолжение таблицы 4.1

|  |  |
| --- | --- |
| Класс | Описание |
| InvokePattern | Используется для элементов управления, которые могут быть вызваны, таких как кнопки. |
| MultipleViewPattern | Используется для элементов управления, в которых можно переключаться между несколькими представлениями одинакового набора сведений, данных или потомков. Например, элемент управления представления списка, где данные доступны в виде эскизов, плитки, значков, списка или подробного представления. |
| RangeValuePattern | Используются для элементов управления, имеющих диапазон значений, применимых к элементу управления. Например, элемент управления «счётчик», содержащий годы, может иметь диапазон от 1900 до 2010, в то время как другой элемент управления «счётчик», представляющий месяцы, имеет диапазон от одного до двенадцати. |
| ScrollPattern | Используется для элементов управления, которые поддерживают прокрутку. Например, элемент управления с полосами прокрутки, активными, если в нем расположено больше информации, чем может быть отображено в видимой области элемента управления. |
| ScrollItemPattern | Используется для элементов управления, имеющих отдельные элементы в прокручиваемом списке. Например, элемент управления «список», имеющий отдельные элементы в прокручиваемом списке, такой как элемент управления «поле с раскрывающимся списком». |
| SelectionPattern | Используется для элементов управления контейнеров выделения. Например, списки и поля со списком. |
| SelectionItemPattern | Используется для отдельных элементов в элементе управления контейнера выделения, таких как списки и поля со списком. |
| TablePattern | Используется для элементов управления, имеющих сетку, а также данные заголовка. Например, листы Microsoft Excel. |
| TableItemPattern | Используется для элементов в таблице. |
| TextPattern | Используется для элементов управления редактирования и документов, которые предоставляют текстовую информацию. |

БрГТУ.91402-07 95 00 26

Завершение таблицы 4.1

|  |  |
| --- | --- |
| Класс | Описание |
| TogglePattern | Используется для элементов управления, в которых поддерживается переключение состояния. Например, флажки и элементы меню с флажками. |
| TransformPattern | Используется для элементов управления, которые можно изменять в размере, перемещать и поворачивать. Обычно используется для шаблонов элементов управления Transform, находящихся в конструкторах, формах, графических редакторах и графических приложениях. |
| ValuePattern | Позволяет клиентам получить или установить значение для элементов управления, которые не поддерживают диапазон значений. Например, элемент выбора даты и времени. |
| WindowPattern | Предоставляет сведения об окне, основные концепции операционной системы Microsoft Windows. Примеры элементов управления, являющихся окнами, включают окна приложений верхнего уровня (Microsoft Word, Проводник Microsoft Windows и т.д.), дочерние окна многодокументный интерфейс MDI и диалоговые окна. |

**4.2 Порядок выполнения работы**

Разработать скрипт, который:

− запускает форму «Form for Test : UIAutomation»;

− считывает тестовые данные их xml файла;

− заполняет форму «Form for Test : UIAutomation» тестовыми данными;

− считывает результат обработки данных;

− сравнивает полученный результат и эталонным значением;

− записывает полученный результат в xml файл.

БрГТУ.91402-07 95 00 27

**5 ЛАБОРАТОРНАЯ РАБОТА №5 «НАГРУЗОЧНОЕ И ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ В JMETER»**

Цель работы: ознакомится с понятием «нагрузочное тестирование», получить практический опыт создания функционального и нагрузочного теста для веб-приложения.

**5.1 Надежность информационной системы как элемент обеспечения безопасности**

Одним из важных составных аспектов безопасности информации в крупных информационных системах является надежность системы, в частности – отказоустойчивость в критических режимах. Угроза безопасности информации может возникать не только в результате преднамеренных атак на информационную систему или неправильных действий пользователя, но и из-за превышения критической нагрузки на систему.

Для надежного и предсказуемого функционирования информационной системы необходимо иметь данные о предельной нагрузке (выраженной, например, в количестве одновременно поступающих в систему пользовательских запросов), о характере поведения системы в условиях перегрузки и возможных последствиях в случае отказа системы из-за превышения допустимой нагрузки. К сожалению, расчетные значения таких характеристик, определенные теоретическими методами, не всегда совпадают с фактическими и не адекватно отражают поведение системы в реальной критической ситуации.

Кроме того, информационная система сама по себе неоднородна с позиции предельной производительности: всегда имеются критичные элементы программно-аппаратной платформы, накладывающие ограничение на производительность системы и снижающие ее надежность. Выявление таких элементов не всегда является тривиальной задачей, и экспериментальные данные дают порой весьма неожиданные результаты.

С позиции поведения информационной системы в условиях критической нагрузки их можно условно разделить на три группы:

− системы, у которых при повышении нагрузки производительность достигает максимального значения и затем остается практически постоянной при дальнейшем росте нагрузки («идеальные» системы);

− системы, у которых наблюдается медленный плавный спад производительности при превышении максимального уровня нагрузки;

− системы, у которых производительность резко падает при незначительном превышении максимальной нагрузки.

Системы третьего типа наиболее критичны с позиции надежности и безопасности информации и требуют соблюдения постоянного наличия резерва по производительности.

БрГТУ.91402-07 95 00 28

**5.2 Метод нагрузочного тестирования для проверки надежности и безопасности систем**

Оценка показателей функционирования информационной системы является сложным процессом, требующим организации работы и четкого взаимодействия большого числа пользователей для создания различных уровней нагрузки на систему. При этом возможно возникновение ситуаций, создающих реальную угрозу безопасности информации, особенно в режимах с пиковой нагрузкой. Поэтому решение задачи измерения показателей функционирования должно базироваться на применении специальных методов тестирования и, соответственно, специальных инструментальных программных средств, позволяющих значительно снизить стоимость испытаний и обеспечить необходимый уровень безопасности информации.

Для получения данных о производительности используется тестирование под нагрузкой с последующим анализом временных характеристик выполнения тестовых заданий. Тестирование под нагрузкой является методом, при котором тестовые задания имитируют нагрузку на информационную систему, подобную той, которая создается реальными пользователями системы.

**5.3 Практическое применение инструментов для нагрузочного тестирования телекоммуникационных информационных систем**

Существующая практика применения инструментов нагрузочного тестирования для телекоммуникационных информационных систем предусматривает, как правило, реализацию сеансов тестирования с имитацией параллельной работы необходимого числа пользователей, выполняющих определенные запросы к испытуемой системе в сетевой среде. Общая цель таких сеансов заключается в обеспечении в течение заданного времени определенного воздействия на тестируемую систему в диапазоне от расчетной нагрузки до пиковой, моделируя при этом как допустимые (штатные), так и дестабилизирующие действия пользователей.

Особенный практический интерес представляет моделирование с уровнем нагрузки, максимально приближенным к предельному, что позволяет выявить возможные проблемы надежности систем до того, как они станут реальной угрозой информационной безопасности.

Практически любые действия пользователей в пиковых режимах нагрузки оказывают значительное дестабилизирующее воздействие, поэтому устойчивость системы в этих условиях является чрезвычайно важным компонентом общей надежности. Существует три основных практических сценария применения инструментов для нагрузочного тестирования, ориентированные именно на анализ надежности и безопасности систем:

− моделирование DOS (Denial-Of-Service) атак на различном уровне работы информационных систем – моделирование преднамеренных дестабилизирующих

воздействий;

БрГТУ.91402-07 95 00 29

− моделирование нормальной деятельности пользователей пиковых режимах нагрузки;

− определение предельно допустимого уровня нагрузки на систему в конкретном окружении.

Моделирование DOS-атак ставит своей целью проверку устойчивости системы к преднамеренным дестабилизирующим воздействиям. Распределенные инструментальные средства нагрузочного тестирования как нельзя лучше подходят для организации моделирования таких атак. DOS-атака может быть организована на любом уровне системы, от уровня протокола TCP до прикладного уровня сервисов. Благодаря гибкости инструментальных средств тестирования возможна реализация алгоритмов любых атак и проведение комплексных испытаний с одновременным воздействием по всем уязвимым местам системы.

Целью DOS-атак является исчерпание ресурсов серверного приложения или системы в целом, что вызовет потерю работоспособности. Устойчивое приложение должно теми или иными способами блокировать нежелательные запросы и не допускать опасного уровня исчерпания ресурсов. Достаточно уязвимыми для таких атак являются приложения распределенных вычислений, которые ассоциируют с каждым открывающимся соединением определенные ресурсы системы, такие как области памяти, потоки, порты и т.д. Кроме того, выполнение даже относительно небольшого количества ресурсоемких запросов может существенно дестабилизировать работу системы.

Благодаря свойству масштабируемости инструментальных средств достаточно легко организовать массированную тестовую атаку с одновременным участием тысяч пользователей, используя всего 10–15 станций моделирования нагрузки. Целесообразно выполнять проведение атаки параллельно с моделированием нормальной работы пользователей, чтобы оценить степень влияния дестабилизирующих воздействий на нормальный ход процесса обработки.

Моделирование нормальной деятельности пользователей в режимах пиковой нагрузки предусматривает высокоинтенсивное выполнение реальных транзакций от имени имитируемых пользователей. Тестирование преследует две основные цели – определение нагрузочной способности системы и обнаружение ошибок.

Режим работы системы, при котором загружено более 80% системных ресурсов принципиальным образом отличается от обычного режима работы. В этих условиях вероятность обнаружения ошибок, которые в обычных условиях не проявляют себя, существенно повышается. К ошибкам, наиболее часто проявляющимся в условиях высокоинтенсивной нагрузки, относятся:

− взаимные блокировки (deadlocks) на таблицах баз данных;

− ошибки синхронизации и управления состоянием;

− ошибки управления ресурсами (утечки памяти, утечки ресурсов);

− переполнение выделенных областей памяти, переполнение буферов и очередей;

− ошибки обработки тайм-аутов, отказы в выделении ресурсов (памяти, сетевых соединений, портов);

БрГТУ.91402-07 95 00 30

− несбалансированность процессов обработки между собой (перегрузки одних процессов на фоне недогруженности других);

− потеря управляемости (невозможность выполнения управляющих команд);

− прогрессирующая деградация производительности вследствие неэффективности алгоритмов диспетчеризации, а также сегментации памяти или иных необратимых процессов.

Каждая из этих ошибок способна привести к внезапной остановке системы, сопровождающейся потерей обрабатываемых данных. Для обнаружения признаков возможных ошибок наряду с моделированием нагрузки используется расширенный мониторинг системы, включая наблюдение за характеристиками функционирования ядра операционной системы, сетевыми протоколами, параметрами СУБД, этапами прикладных процессов обработки.

Применение инструментов нагрузочного тестирования, обладающих способностью одновременного запуска различных по составу последовательных тестов, позволяют гибко моделировать характерные ситуации, выделяя проблемные «узкие места».

Определение предельно допустимого уровня нагрузки на систему в конкретном окружении – достаточно распространенная практика получения объективных сведений о характеристиках системы. С помощью тестового инструментария выполняется серия реализации одного и того же сеанса тестирования с определенным шагом увеличения интенсивности нагрузки от реализации к реализации. Увеличение интенсивности обычно заключается в добавлении на каждом шаге определенного числа новых пользователей. При фиксированных значениях показателей времени отклика становится возможным определить предельно допустимое число пользователей, которые при заданной частоте обращений могут получить гарантированный уровень качества обслуживания.

**5.4 Обзор JMeter**

Apache JMeter – Java-приложение открытым исходным кодом. Разработано для функционального и нагрузочного тестирования. Изначально создано для тестирования Web-приложений.

Apache JMeter может быть использован для нагрузочного тестирования как статических, так и динамических ресурсов. С помощью приложения можно производить тяжёлые нагрузочные тесты для серверов, сетей или объектов для тестирования устойчивости, общей производительности под различными видами нагрузок. Есть возможность создавать различные графические отчёты, отражающие текущее состояние тестируемой системы.

Apache JMeter имеет следующие возможности:

а) проведение нагрузочного тестирования, а также тестирования производительности для большого количества различных типов серверов:

1) Web - HTTP, HTTPS;

БрГТУ.91402-07 95 00 31

2) SOAP;

3) Database via JDBC;

4) LDAP;

5) JMS;

6) Mail - SMTP(S), POP3(S) and IMAP(S);

7) Native commands or shell scripts;

б) многопоточный режим и одновременный отбор различных функций в каждом из потоков;

в) воспроизведение и обработка данных после выполнения теста.

JMeter имеет встроенный прокси-сервер, который предназначен для записи сессий, но можно использовать и внешний. Перед началом тестирования необходимо составить тестовый план, описывающий серию заданий, которые необходимо выполнить JMeter. Он должен содержать одну или несколько групп потоков (Thread Groups) и другие элементы:

− логические контроллеры (Logic controllers);

− типовые контроллеры (Sample generating controllers);

− слушатели (Listeners);

− таймеры (Timers);

− утверждения (Assertions);

− конфигурационные элементы (Configuration elements).

Группа потоков (Thread Group). Элемент Thread group – это начальная точка любого тест плана. Все контроллеры и образцы должны быть данным элементом. Другие элементы, например слушатели, могут быть помещены непосредственно под тест планом, в этом случае они будут применяться ко всей группе потоков. Как следует из названия, элемент группы потока контролирует количество потоков, которые JMeter будет использовать для выполнения теста. Элементы управления для групп потоков позволяют:

− установить число потоков;

− установить период нарастания;

− установить количество циклов выполнения теста.

Каждый поток будет выполнять план тестирования в полном объеме и совершенно независимо от других тестовых потоков. Несколько потоков используется для имитации параллельных подключений к серверному приложению.

JMeter для групп потоков использует период нарастания, который показывает, как долго необходимо увеличивать объем, чтобы добиться полного количества выбранных потоков. Если используется 10 потоков, а период нарастания равен 100 секундам, то JMeter займет 100 секунд, чтобы получить и выполнить все 10 потоков. Каждый поток будет начинаться через 10 (100/10) секунд после запуска предыдущего потока. Если есть 30 потоков и период нарастания 120 секунд, то каждый последующий поток будет отложен на четыре секунды.

БрГТУ.91402-07 95 00 32

Время нарастания должно быть достаточно длинным, чтобы избежать слишком большой рабочей нагрузки в начале теста, и достаточно коротким, чтобы последние потоки начинались перед завершением первых потоков.

Контроллеры (Controllers). JMeter имеет два типа контроллеров: Samplers (далее сэмплер) и логические контроллеры (Logic Controllers). Они контролирует процесс тестирования.

Сэмплеры сообщают JMeter об отправки запросов на сервер. Например, можно добавить запроса Sampler HTTP, если нужно, чтобы JMeter отправил HTTP запрос.

Логические контроллеры позволяют настраивать логику, которую использвует JMeter, чтобы решить, когда отправить запросы. Например, можно добавить Interleave Logic Controller для переключения между двумя HTTP Request сэмплерами.

Слушатели (Listeners). Слушатели обеспечивают доступ к информации, которую JMeter собирает в тестовых случаях во время выполнения тест плана. Graph Results отображает время отклика на графике. View Result Tree показывает детали запросов и ответов сэмплера, и может отображать основные HTML и XML представления ответа.

Слушатели могут быть добавлены в любой точке теста, в том числе непосредственно под тест планом. Они будут собирать данные только из элементов на уровне или ниже их уровня.

Таймеры (Timers). По умолчанию, поток JMeter отправляет запросы без пауз между каждым запросом. Однако, рекомендуется задавать задержку, добавив один из доступных таймеров в группу потока.

Таймер вызовет JMeter, и он задержится определенное количество времени перед каждым сэмплером, который находится в области таймера. Если добавить больше чем один таймер в Thread Group, JMeter возьмёт сумму таймеров и будет задерживаться на это количества времени прежде, чем выполнить сэмплер, к которому применяются таймеры. Таймеры могут быть добавлены под сэмплеры или контроллеры, в качестве дочернего элемента, чтобы ограничить сэмплеры, к которым они применяются.

Утверждения (Assertions). Утверждения позволяют утверждать факты ответов, полученных от испытываемого сервера. Используя утверждение можно по существу тестировать результаты, возвращаемые приложением, которые можно ожидать.

Например, можно утверждать, что ответ на запрос будет содержать тот или иной текст. Можно добавить утверждение любому сэмплеру. Например, можно добавить утверждение к HTTP запросу, который проверяет наличие текста. Если JMeter не сможет найти текст, то этот запрос будет обозначен как неуспешный.

Порядок выполнения элементов: таймеры, утверждения, пред- и пост-процессоры обрабатываются только, если есть сэмплер к которым они применяются. Логические контроллеры и сэмплеры обрабатываются в том порядке, в котором они появляются в дереве. Другие элементы испытаний обрабатываются в соответствии с областью, в которой они находятся, и тип элемента теста.

БрГТУ.91402-07 95 00 33

**5.5 Создание тест-плана**

Рассмотрим создания тест плана для тестирования веб-приложения. Будет создано пять пользователей, которые посылают запросы на две страницы веб сайта. Каждый пользователь выполнить свой тест два раза. Поэтому, всего будет сгенерировано 20 HTTP запросам.

Для создания тест плана будут использованы следующие элементы:

− Thread Group;

− HTTP Request;

− HTTP Request Defaults;

− Graph Results.

Первый шаг, который нужно сделать для создания тест плана, - это добавить элемент Thread Group. В этом элементе будет задаваться количество пользователей, и имитироваться, как часто пользователи должны отправлять запросы и сколько запрос должен отправить каждый пользователь.

Для добавления Tread Group необходимо выполнить следующую последовательность действий: Test Plan > Add > Thread Group. Элемент Thread Group будет добавлен в под Test Plan.

Выбрав добавлен элемент Thread Group в правой части фреймворка отобразятся свойства элемента. Изменим их согласно нашему условию. Элемент Thread Group должен иметь вид, как показано на рисунке 5.1.

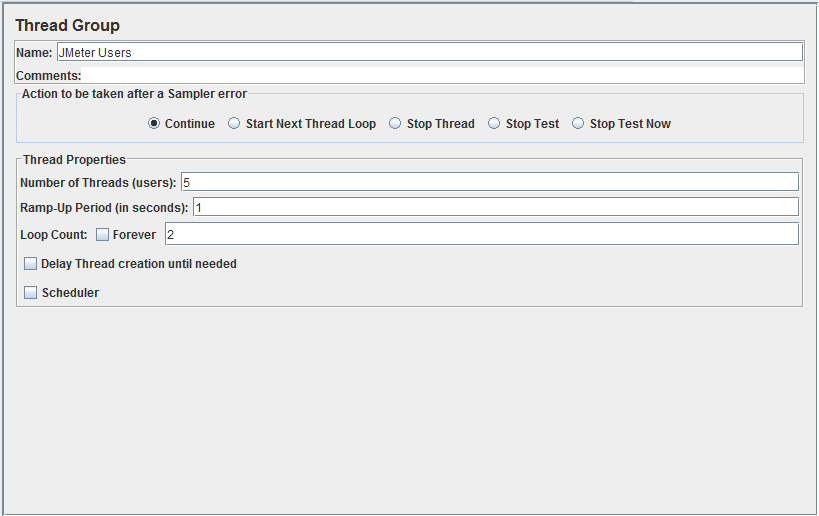


Рисунок 5.1 – Элемент Thread Group

Создадим два HTTP-запроса. Первый для страницы http://nunit.org, а второй для страницы http://nunit.org/index.php?p=download. JMeter будет посылать запросы в порядке их появления в дереве. Итак, для добавления HTTP-запроса необходимо добавить элемент HTTP Request (JMeter Users > Add > Sampler > HTTP Request).

БрГТУ.91402-07 95 00 34

Примеры запросов изображены на рисунках 5.2 и 5.3 для первой и второй страниц. Как видно из приведенных рисунков, в качестве имени сервера указали nunit.org.

Добавим в тест-план элемент слушателя View Results Tree (Add > Listeners > View Results Tree), который будет отвечать за предоставление всех результатов HTTP запросов.

После запуска теста, элемент View Results Tree будет отображать все выполнившиеся запросы. Пример такого результата изображен на рисунке 5.4. Как видно из рисунка все HTTP запросы были выполнены успешно.

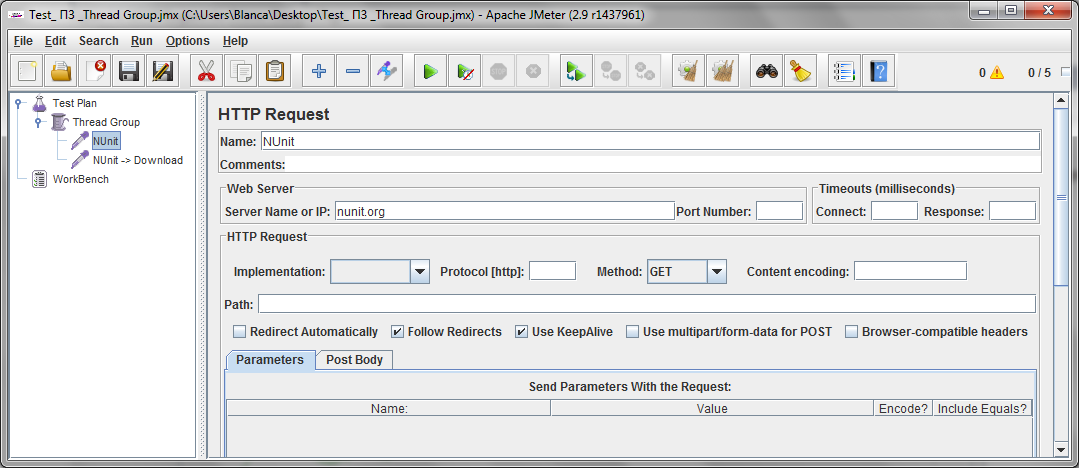


Рисунок 5.2 – HTTP запрос к странице http://nunit.org

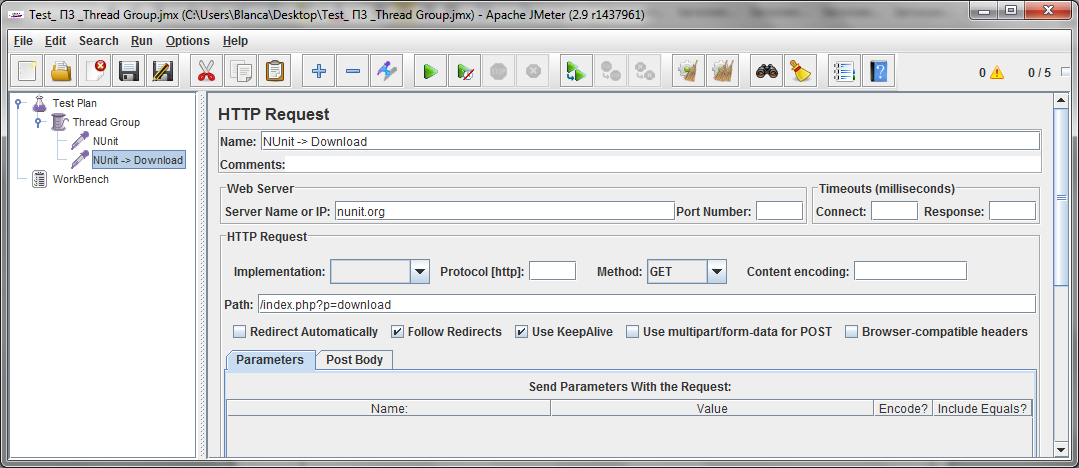


Рисунок 5.3 – HTTP запрос к странице http://nunit.org/index.php?p=download

Теперь добавим элемент слушателя Graph Results (Add > Listeners > View Results Tree), который так же будет отвечать за предоставление результата запросов, но в графическом представлении. Увеличим количество циклов повторения до 25 раз и запустим тест-план. Результат такого тест-плана рассмотрим на рисунке 5.5, представленным элементом Graph Results.

БрГТУ.91402-07 95 00 35

Все значения предоставляются в миллисекундах. Graph Result предоставляет следующую информации:

− data – время отклика каждой отдельной единицы данных т.е. каждого проверенного адреса;

− average – усредненное время отклика, объективный график изменения нагрузки;

− median – значение медианы (используется в статистике);

− deviation – погрешность, стандартное отклонение;

− throughput – пропускная способность выполняемых запросов.

В данном пример достаточно значений Average и Throughput. Они отобразят нагрузку на веб-сервер и пропускную способность запросов. По графику выше видно, что время отклика равно 1567 мс, и дальше оно не растет. А это значит, что сервер нормально выдерживает нагрузку в 5 виртуальных пользователей.

Тест-план может быть модифицирован дополнительными элементами.

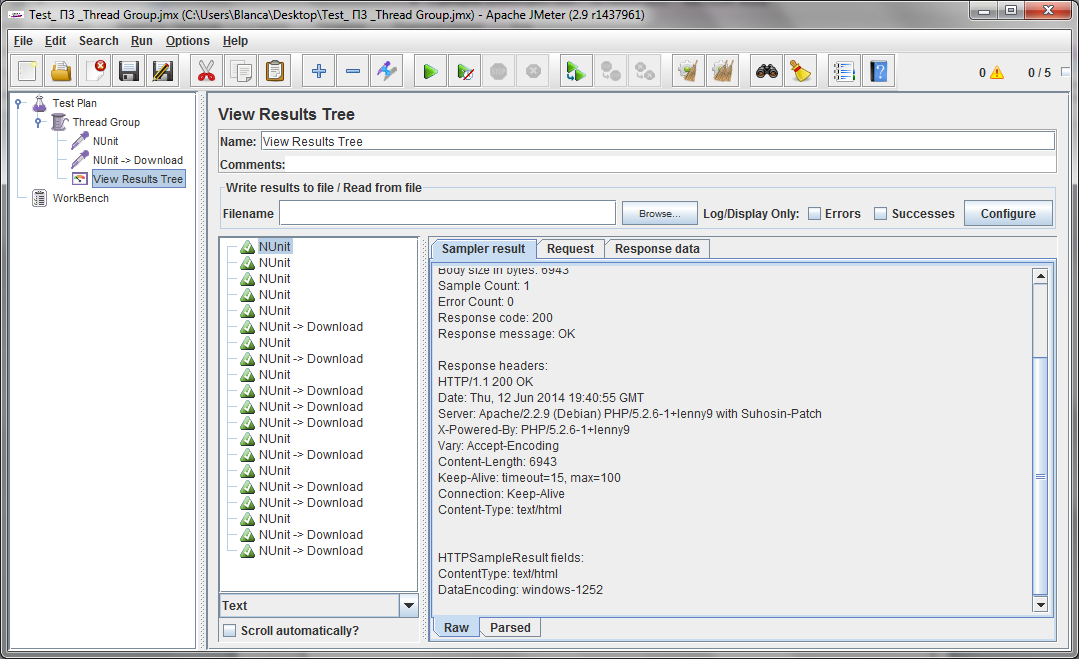


Рисунок 5.4 – Результат использования элемента View Result Tree

**5.6 Порядок выполнения работы**

1) Добавьте элемент Thread Group в тест план.

2) Добавьте элемент HTTP Request и определите GET-запрос к странице «Home.aspx». С помощью элемента Response Assertion убедитесь, что запрос был выполнен успешно.

3) Отправьте запрос HTTP Get с параметром «code=2».

4) Повторите пункт 3 с параметром «code = 3».

5) Добавьте элемент HTTP Request и определите POST-запрос к странице «Feedback.aspx», передавая валидные данные.

БрГТУ.91402-07 95 00 36

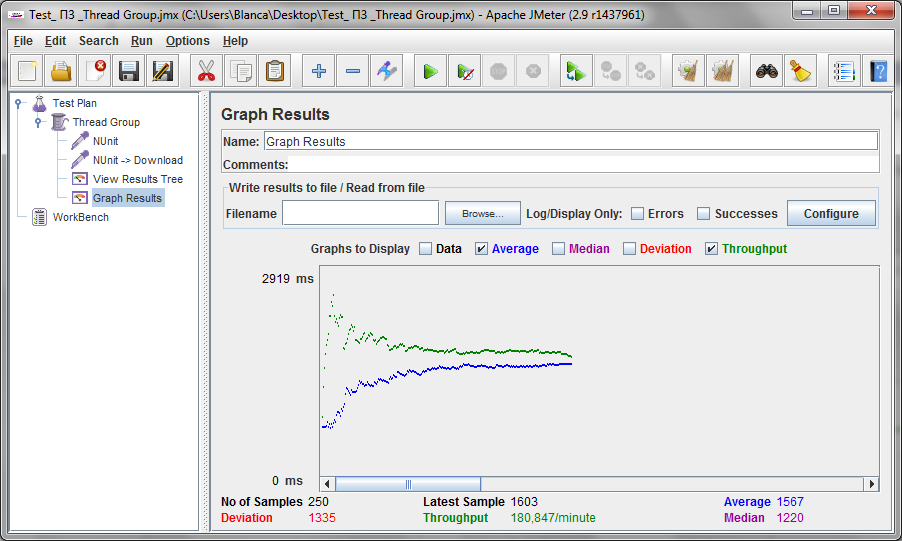


Рисунок 5.5 – Результат использования элемента Graph Results

6) Повторите пункт 5, передавая невалидные данные. Сравните полученный результат с результатом пункта 5.

7) Добавьте в тест-план элемент Counter и передавайте текущее значение счётчика в качестве параметра GET-запроса к странице «Home.aspx».

8) Добавьте элемент CSV Data Set Config и настройте его на считывание данных с указанного файла. С помощью элемента Debug Sampler убедитесь о корректности считывания данных.

9) Модифицируйте POST-запрос к странице «Feedback.aspx», передавая в качестве параметров считанные данные с csv файла.

10) Добавьте в тест-план элементы If Controller для обеспечения успешного выполнения каждого запроса.

11) Определите количество виртуальных пользователей, время их генерации и количество циклов повторения.

12) Повторяя пункт 11 и падавая различную нагрузку на сервер, проанализируйте изменение значений пропускной способности и его времени отклика.