Математические структуры и моделирование
2011, вып. 24, с. 69–74

УДК 004.05

ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ МОДУЛЬНЫХ ТЕСТОВ ДЛЯ ОРГАНИЗАЦИИ НАГРУЗОЧНОГО, СТРЕСС-ТЕСТИРОВАНИЯ, ТЕСТИРОВАНИЯ СТАБИЛЬНОСТИ

Я.Г. Лило, Е.А. Тюменцев

В статье приведён один подход к повторному использованию модульных тестов для организации нагрузочного, стресс-тестирования, тестирования стабильности.

В настоящее время получили широкое распространение различные практики автоматического тестирования программного обеспечения. Автоматическое тестирование подразумевает написание программного кода, который занимается проверкой работоспособности разрабатываемого программного обеспечения. Сама идея принадлежит Кенту Беку [1], который является автором практики «Разработка через тестирование» (Test-Driven Development), а также разработчиком первых библиотек модульного тестирования SUnit [2] для языка SmallTalk и JUnit [3] для языка Java. Подход модульного тестирования заключается в разбиении программы на небольшие фрагменты и проверки корректности каждого фрагмента в отдельности. Библиотека JUnit послужила прототипом для разработки библиотек модульного тестирования на других языках программирования. Совокупность этих библиотек принято называть xUnit библиотеками.

Большинство проектов, применяющих автоматическое тестирование, используют xUnit библиотеки для выполнения функционального тестирования. Однако, наряду с функциональным тестированием, часто возникает потребность в следующих видах тестирования:

- приёмочное,
- регрессионное,
- нагрузочное,
- стабильности,
- стресс,

ООО «ИСС Арт», Омский государственный университет им. Ф.М. Достоевского
E-mail: Lilo-Jasha@yandex.ru, etyumentcev@gmail.com

- конфигурационное.

Повторное использование существующего кода является одним из ключевых моментов повышения производительности труда в разработке программного обеспечения. Естественным образом возникает идея о возможности повторного использования модульных тестов для выполнения перечисленных выше видов тестирований. Такие идеи в разном виде высказываются на различных интернет-ресурсах, например, [4].

Нами был проанализирован ряд специализированных инструментов автоматического тестирования: Selenium [5], SilkTest [6], IBM Rational Robot [7] на предмет повторного использования модульных тестов. Выбор был сделан на этих инструментах, так как разные источники, например, [8], указывают на них как на наиболее распространённые и популярные. Анализ показал, что использование готовых модульных тестов в данных приложениях возможно, но потребует большой объем дополнительной работы по их интеграции.

В настоящей статье мы опишем реализацию библиотеки автоматического тестирования, которая позволяет осуществлять нагрузочное, стресс-тестирование и тестирование стабильности на основе модульных тестов. Сама библиотека распространяется по лицензии GPL v.3 и находится в свободном доступе в Интернет по адресу: `http://code.google.com/p/cw-proj/`.

1. Определения

Прежде чем приступать к описанию библиотеки, уточним понятия нагрузочного, стресс-тестирования и тестирования стабильности.

Определение 1 (Нагрузочное тестирование [9])**.** Предназначено для сбора показателей производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос.

В частности, нагрузочное тестирование применяется для вычисления максимального количества транзакций, которое способно обработать приложение за фиксированный промежуток времени.

Определение 2 (Стресс-тестирование)**.** Проверка работоспособности системы в условиях использования ресурсов, близких к предельным.

Стресс-тестированием является, например,

- выполнение очень большого количества транзакций;
- функционирование системы при минимально возможном объёме памяти;
- одновременное обращение большого числа пользователей системы.

Определение 3 (Тестирование стабильности)**.** Проводится с целью оценки работоспособности системы при проектной нагрузке в течение очень длительного времени.

К тестированию стабильности, можно отнести, например, вычисление среднего времени работы сервера без перезагрузки.

2. Библиотека

Библиотека реализована на платформе .Net и интегрирована с библиотекой модульного тестирования NUnit [10], но может быть легко портирована на любой другой язык и/или интегрирована с любой другой библиотекой модульного тестирования.

2.1. Интерфейс ITest

Базовым элементом библиотеки является интерфейс ComplicatedTests.ITest, который инкапсулирует в себе особенности вызова одиночного теста. Ниже представлен листинг с определением данного класса:

```
namespace ComplicatedTests
{
    public interface ITest
    {
        void setUp();
        void tearDown();
        void run();
        String GetXML();
    }
}
```

где метод setUP() используется для инициализации окружения теста, tearDown() — для восстановления окружения к исходному состоянию после выполнения теста, run() — для выполнения самого теста, GetXML() — для сериализации теста в XML.

Класс TogetherTests.Test реализует интерфейс ComplicatedTests.ITest для вызова теста, написанного с помощью библиотеки NUnit. Таким образом, данный класс является Фасадом [11] для модульных тестов, написанных с помощью библиотеки NUnit. Точно так же можно реализовать фасады к любой другой библиотеке модульных тестов. Тем самым мы получили, что описываемая нами библиотека является расширяемой относительно библиотек модульного тестирования.

Далее, если это не оговорено особо, будем считать, что все классы и интерфейсы принадлежат пространству имён ComplicatedTests.

2.2. Последовательность тестов

Класс GivenSequenceTests реализует интерфейс ITest как последовательное выполнение заданного набора тестов указанное число раз. Этот класс реализован с помощью паттерна Компоновщик [11], что позволяет ему выполнять любой набор тестов, реализующих интерфейс ITest.

2.3. Параллельное множество тестов

Класс ConcordantTests реализует интерфейс ITest как параллельное выполнение заданного множества тестов. Этот класс, так же как и

TogetherSequenceTests, реализован с помощью паттерна Компоновщик, что позволяет легко организовывать параллельные последовательности тестов или, наоборот, серию параллельных тестов.

2.4. Последовательность тестов с заданным распределением

Последовательное выполнение тестов, реализованное классом TogetherTests, можно обобщить: предположим, что у нас есть множество T из n тестов. Поставим во взаимнооднозначное соответствие каждому тесту из множества T число от 1 до n. Предположим, что номер теста, который надо выполнить следующим, определяется значением случайной величины $\psi : N \to (1, \ldots, n)$. Можно управлять частотой и порядком выполнения тестов, выбирая случайные величины с различным распределением. Для вычисления значений случайных величин используется библиотека Агнера [12]. Поддерживаются случайные величины со следующими распределениями:

- равномерное распределение,
- нормальное распределение,
- распределение Бернули,
- распределение Пуассона,
- биномиальное распределение,
- гипергеометрическое распределение.

2.5. XML-конфигурирование тестов

Библиотека предоставляет возможность описания сценариев тестирования в виде XML-документов. Рассмотрим следующий пример:

```
<?xml version="1.0"?>
<TogetherTests>
   <Test assembly="Tests1.dll" class="Tests" method="True"/>
    <ConcordantTests >
      <Test assembly="Tests1.dll" class="Tests" method="True"/>
      <GivenSequenceTests  sequence="Normal" m="1" s="1"  quantity="10">
         <Test assembly="Tests1.dll" class="Tests" method="True"/>
         <Test assembly="Tests1.dll" class="Tests" method="False"/>
     </GivenSequenceTests>
  </ConcordantTests>
</TogetherTests>
```

Для описания тестов применяются следующие XML-тэги.

- Test — одиночный модульный тест. Имеет следующие атрибуты: assembly — имя сборки, содержащий класс с тестом, class — имя класса, содержащего тест, method — имя теста,

- TogeterTests — последовательность тестов,
- GivenSequenceTests — последовательность тестов с заданным распределением. Имеет следующие атрибуты: sequence — название распределения, quantity — количество повторов, также могут быть дополнительные параметры, зависящие от распределения,
- ConcordantTests — множество тестов, выполняемое параллельно.

В рассмотренном примере определена последовательность из двух тестов: одиночного и параллельного. В параллельном тесте, в свою очередь, одновременно выполняются одиночный тест и последовательность из двух тестов с нормальным распределением.

Описание сценариев в XML-документе позволяет создавать новые тесты без написания программного кода.

3. Реализация нагрузочного, стресс-тестирования и тестирования стабильности

Рассмотрим возможности организации указанных видов тестирования с помощью данной библиотеки.

Начнём с тестирования стабильности. Разработанная нами библиотека позволяет выполнять любую последовательность тестов сколько угодно раз, что легко позволяет организовать подобный вид тестирования.

Выполнение тестовых наборов в нескольких потоках одновременно позволяет моделировать изменение нагрузки системы, а представление сценария тестирования в виде XML позволяет быстро организовать проверку любой последовательности тестов при предопределённом наборе ресурсов. Эти возможности позволяют ускорить разработку тестов производительности и стресс-тестов.

Литература

1. Бек К. Экстремальное программирование: разработка через тестирование. Библиотека программиста. СПб. : Питер, 2003. 224 с.
2. Camp SmallTalk SUnit. URL: `http://sunit.sourceforge.net/` (дата обращения: 29.05.11)
3. JUnit. URL: `http://www.JUnit.org` (дата обращения: 08.02.11)
4. Повторное использование Selenium и UnitTest (Python). URL: `http://automated-testing.info/knowledgebase/article/povtornoe-ispolzovanie-selenium-i-unittest-python` (дата обращения: 30.08.2011)
5. SeleniumHQ. Web application testing system. URL: `http://seleniumhq.org/` (дата обращения: 08.02.11)
6. SilkTest 2010. URL: `http://www.borland.com/us/products/silk/silktest/index.html` (дата обращения: 16.02.11)

7. IBM – Rational Robot. URL: http://www-142.ibm.com/software/products/ru/ru/robot/ (дата обращения: 16.02.11)
8. Винниченко И.В. Автоматизация процессов тестирования. СПб. : Питер, 2005. 203 с.
9. Нагрузочное тестирование Web-приложений при помощи IBM Rational Performance Tester: Часть 3. Удаленное выполнение тестов с возрастающей пользовательской нагрузкой. URL: http://www.ibm.com/developerworks/ru/library/r-1211_lee-tham3/index.html (дата обращения: 09.05.11)
10. NUnit. URL: http://www.nunit.org/ (дата обращения: 08.02.11)
11. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб. : Питер, 2001. 368 с.
12. Pseudo random number generators. URL: http://www.agner.org/random/ (дата обращения: 09.05.11)