Библиотека инструментов для разработки и автоматического тестирования консольных приложений любой сложности на языках программирования C# и C++

В.Д. Кротков[1], А.Н. Даниленко[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. В настоящей работе приведено описание функционала библиотеки, предназначение которой – упрощение создания консольных приложений на языках C# и C++ путем предоставления программисту возможностей использовать встроенное настраиваемое меню произвольной степени вложенности, специальные средства для осуществления контроля над переменными программы и отслеживания состояния программных сущностей. Библиотека содержит функции ручной и автоматической проверки и многоуровневого анализа работы консольного приложения. Также библиотека включает в себя вспомогательный функционал случайной или шаблонной генерации тестовых данных пользовательского или стандартного типа и дает возможности для точечного выявления ошибок, допущенных при разработке приложения. Она создана с использованием современных технологий объектно-ориентированного программирования в соответствии с передовыми архитектурными и алгоритмическими решениями.

1. Введение

Современные тенденции объектно-ориентированного программирования направлены на универсализацию, алгоритмизацию, повышение уровня абстракции и упрощение работы со сложными программными системами. Для тестирования таких систем, а также для проверки их архитектуры на корректность используются отдельные инструменты, надстройки и подпрограммы (инструменты отладки кода, Clang Power Tools, JUnit). Однако не всегда можно легко и быстро проверить, правильно ли работает алгоритм, протестировать реакцию приложения или программного компонента на действия пользователя. Библиотека, описание которой представлено в настоящей работе, избавляет программиста от необходимости вручную тестировать каждый функциональный элемент, позволяя ему сконцентрироваться непосредственно на написании дальнейшего взаимодействия программных сущностей и на реализации алгоритмов.

2. Материалы и методы

Предлагаемая авторами библиотека дает программисту следующие возможности:

- использовать встроенное меню;
- создавать пул рабочих переменных;
- автоматически генерировать программные данные;

— проводить контролируемые тесты нужного блока кода;
— получать результаты тестирования;
— получать информацию о runtime-ошибках.

2.1 Встроенное меню

Меню – главный элемент библиотеки. При запуске оно позволяет выбрать на выполнение один из своих пунктов, представленных в виде сигнатур функций или лямбда-функций. Меню снабжено защитой от некорректного выбора пункта (актуально при ручном тестировании) и возможностью запустить несколько пунктов на последовательное выполнение (актуально при автоматическом и полуавтоматическом тестировании). Меню также способно запускать на выполнение функции с аргументами, полученными из пула аргументов, либо введенными вручную. У меню присутствуют три режима работы:

— ручной – программист сам выбирает, какой пункт меню запустить и какие действия произвести в пределах этого пункта;
— полуавтоматический – программист вводит последовательность команд, которые нужно выполнить, а меню перехватывает управление консолью и вводит в нее эти команды;
— автоматический – меню само генерирует последовательность команд и необходимые данные и затем запускает нужные пункты.

В данной библиотеке меню представлено классом Menu, а пункты меню – базовым классом MenuItem, причем Menu содержит список объектов MenuItem и само наследуется от MenuItem. Иначе говоря, запускаемые пункты меню сами могут быть как простыми пунктами (представленными классом CommonMenuItem), так и другими меню. Список пунктов меню имеет древовидную структуру, ветками которой служат объекты Menu, а листьями – объекты CommonMenuItem. Таким образом, был применен структурный шаблон проектирования Composite, и была достигнута цель – сделать обращение с пунктами меню универсальным, независимо от конкретного класса-наследника MenuItem. В такой архитектуре оставлена возможность расширения иерархии наследования, строящейся вокруг MenuItem, без изменения кода, взаимодействующего с пунктами меню.

Так как конструирование объекта класса Menu может оказаться громоздким, было принято архитектурное решение – реализовать порождающий шаблон Builder, написав класс MenuBuilder. Его задача – упростить создание сложного объекта класса Menu, разбив это создание на цепочку простых вызовов. Строитель MenuBuilder тем отличается от других строителей, что он позволяет одновременно создавать объекты класса Menu непосредственно на месте пунктов текущего меню. Такая возможность реализована за счет перевода текущего строящегося меню в стек и постановки на строительство нового меню. По окончанию конструирования вершина стека удалится из него и станет текущим конструирующимся меню [1].

Чтобы взаимодействие меню и его пула рабочих переменных протекало проще, и была возможность в дальнейшем добавить новые взаимодействия, был реализован структурный шаблон проектирования Facade – он контролирует доступ к меню и его пулу переменных, предоставляя пользователю удобный интерфейс для взаимодействия со всей связкой компонентов [2].

2.2 Пул рабочих переменных

Пул рабочих переменных инкапсулирует программные сущности. Он обеспечивает контролируемый доступ к своему содержимому, информирует программиста при попытке изменить состояние данных или предотвращает некорректные их изменения (дополнительный функционал). Пул рабочих переменных является частью системы инкапсуляции запроса на вызов функции с принимаемыми аргументами, сигнатура которой содержится в соответствующем объекте CommonMenuItem. Пул также способен давать доступ к содержимому по значению или по ссылке, открывая возможности для более вариативной передачи параметров в функцию.

В библиотеке пул рабочих переменных представлен шаблонным классом от неограниченного числа переменных DataPool. Все данные, поступающие в DataPool, оборачиваются в шаблонный класс FlexibleArgument. Именно он инкапсулирует обращение к подконтрольным ему данным или переменной и поддерживает ссылочное и значимое обращение к содержимому. Ссылочное обращение организованно в виде двух лямбда-функций – геттера и сеттера. Они позволяют захватывать внешний контекст оборачиваемой переменной и более гибко настраивать это обращение [3].

2.3 Генерация случайных данных

Библиотека включает в себя функции-генераторы псевдослучайных данных. Они могут генерировать как данные стандартных типов (int, double, string), так и данные пользовательского типа. Получение случайных величин происходит путем выбора случайного значения, включенного в границы, которые также можно сгенерировать случайно или ввести вручную. Случайные строки получаются путем генерации случайных значений и перевода их в символы ASCII кода. Также предусмотрена возможность генерации строк на основе regex-паттерна. Чтобы сгенерировать данные пользовательского типа, программисту необходимо написать конкретную реализацию шаблонного класса RandomFactory, в котором есть метод получения данных шаблонного типа. Такое программное решение было принято в виду невозможности корректной реализации интерфейса, имеющего шаблонный нестатический метод GetRandomData, поскольку реализация такого интерфейса противоречит некоторым принципам современного объектно-ориентированного программирования [4].

2.4 Контролируемые тесты кода

На текущем этапе разработки библиотеки значительное место занимает концепт контролируемых тестов. Он предполагает то, что автоматика должна уберегать программиста от ошибки при вводе тестовых данных, чтобы он мог быстро, надежно и удобно тестировать необходимые блоки кода. Поэтому все меню и другие элементы библиотеки находится под надзором системы контроля ввода данных. В случае некорректного ввода программа продолжит работать, а пользователь получит предупреждение о некорректном вводе и просьбу выполнить повторный ввод. Программист может использовать встроенные проверки на корректность или сам определять, какой именно ввод данных считать правильным. Для этого в библиотеке есть функционал ввода строки по заранее заданному шаблону. Далее программист может использовать полученную строку для построения собственного объекта.

Однако такого функционала не всегда достаточно. С целью максимально полной проверки данных на корректность разрабатывается система соответствия дополнительным условиям. Программисту нужно будет воспользоваться соответствующей функцией библиотеки и передать в нее проверяющее лямбда-выражение или сигнатуру функции дополнительной проверки.

Чтобы завершить концепт контролируемых тестов и целиком переложить получение корректных данных на автоматику, был написан шаблонный класс Parser, задача которого – непосредственно получать объекты нужных типов из введенной строки. В силу невозможности корректной реализации интерфейса IParseable, соответствующей современным принцам написания качественного кода, было принято решение написать шаблонный класс Parser и дать программисту возможность самому специфицировать шаблонную реализацию. Реализация данного класса используется в самой библиотеке для конвертации строк в числа. В числе преимуществ такого подхода – более гибкая настройка конвертации строк в объекты и возможности дальнейшей настройки классов-конвертеров.

2.5 Получение результатов тестирования

Программист может настроить меню на выполнение серии тестов блока кода. Результаты тестирования он может получать в виде информации в консоли. Если меню настроено на выполнение последовательности команд, оно получает каждую команду из внутреннего буфера команд и перед выполнением печатает ее в консоль, создавая видимость того, что команда

была введена самим пользователем. Так в консоли выстраивается цепочка из всех пунктов, которые запустились на выполнение в меню, и результатов их работы; отчета в текстовом файле. При выполнении цепочки команд меню печатает содержимое консоли в отдельный файл; объекта класса Results. Он инкапсулирует данные, которые в ходе работы должны были напечататься в консоль и некоторую другую информацию.

Все способы получения результатов включают в себя получение информации о количестве успешных тестов, которые прошел блок кода, о том, какие тесты и на каких наборах данных оказались неудачными, об исключениях, сгенерировавшихся в процессе, и о времени выполнения программы.

2.6 Получение информации о runtime-ошибках

При возникновении ошибки в пределах блока меню пользователю будет доступен отчет о месте, времени, характере и заложенном описании ошибки. Ему также придет запрос, прекратить или продолжить выполнение программы.

3. Результаты и обсуждение

На рисунках 1-2 представлены диаграмма основных взаимосвязей элементов библиотеки и их состава, а также процесс конструирования меню на клиентской стороне.

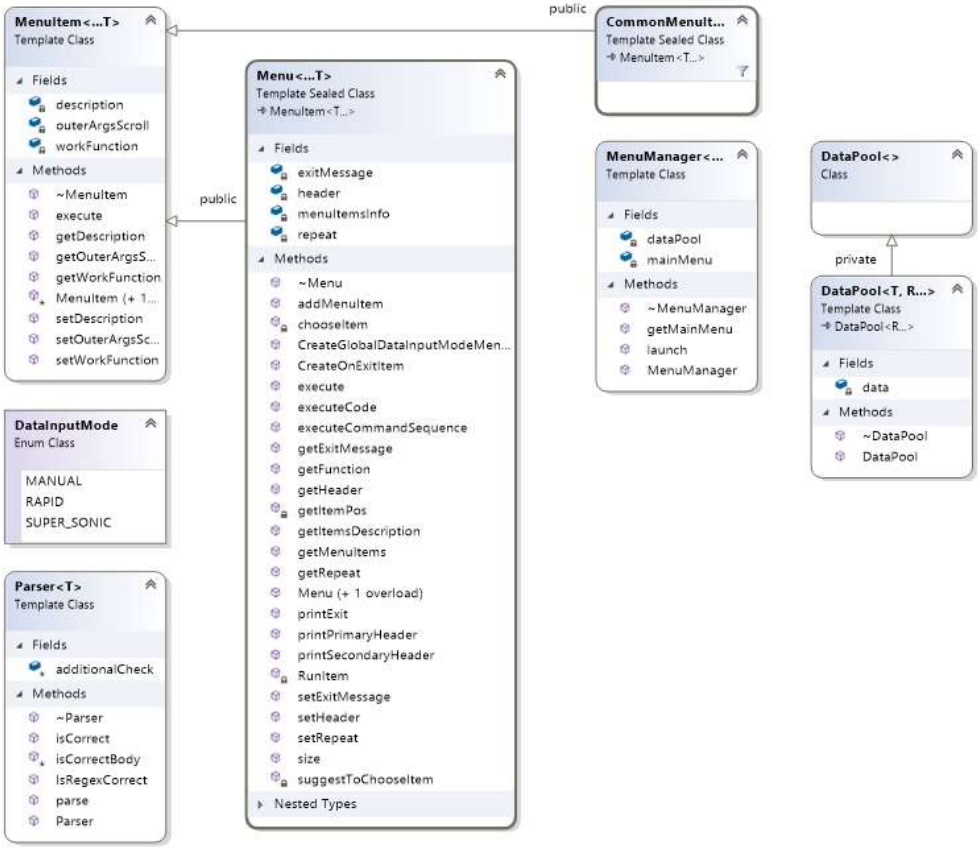

Рисунок 1. Диаграмма основных взаимосвязей элементов библиотеки и их состава.

4. Заключение

Разработанная авторами библиотека позволяет существенно снизить временные затраты программиста на необходимость написания однообразного кода, а также позволяет облегчить задачу тестирования приложений. Основными преимуществами данной библиотеки является возможность автоматической генерации программных данных и возможность получения по ним результатов тестирования с указанием информации о runtime-ошибках. Библиотека была интегрирована и апробирована на реальных задачах.

В настоящий момент происходит расширение функционала библиотеки.

```
void testMenu()
{
    const MenuManager<> menuManager(
        Menu<>::MenuBuilder().
        buildHeader(new string("Добро пожаловать в Главное меню")).
        buildDescription(new std::string("С возвращением в Главное меню")).
        buildExitMessage(new string("До новых встреч в Главном меню")).
        buildRepeat(true).
        addMenuItem(new CommonMenuItem<>(
            new string("Первый тестовый пункт - обычный пункт"),
            firstTest)).
        addMenuItem(new CommonMenuItem<>(
            new string("Второй тестовый пункт - обычный пункт"),
            []() { ... })).
        switchBuildTarget().
        buildDescription(new string("Третий тестовый пункт - одинарное меню из 2 подпунктов")).
        buildExitMessage(new string("Выход из тестового подменю")).
        buildRepeat(false).
        addMenuItem(new CommonMenuItem<>(
            new string("Первый тестовый подпункт - обычный пункт"),
            new std::function<void()>([]() { ... }))).
        addMenuItem(Menu<>::CreateGlobalDataInputModeMenuItemChanger()).
        travelHigher().
        addMenuItem(Menu<>::CreateOnExitItem()).
        build());
    menuManager.launch();
}
```

Рисунок 2. Процесс конструирования меню на клиентской стороне.

5. Литература

[1] Gamma, E. Design Patterns Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides – Addison-Wesley, 1994. – 395 p.

[2] Freeman, E. Head First Design Patterns: a Brain-Friendly Guide / E. Freeman, B. Bates, K. Sierra, E. Robson – СПб.: Питер, 2018. – 656 с.

[3] Variadic templates. Tuples, unpacking and more [Electronic resource]. – Access mode: https:// habr.com/ru/post/228031/ (10.12.2019).

[4] How to write Regular Expressions? [Electronic resource]. – Access mode: https://www. geeksforgeeks.org/ write-regular-expressions/ (дата обращения: 10.12.2019).

Library of tools, aimed at simplifying the development and performing automatic tests of console applications of any difficulty on C# and C++ programming languages

V.D. Krotkov[1], A.N. Danilenko[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. In the present paper there is represented the description of the library, which purpose is to simplify the process of creating console applications on C# and C++ programming languages by giving programmer an opportunity to use built-in custom menu of any nesting level, as well as special facilities to control the state of program variables and keep track of condition of program entities. The library contains functions of manual and automatic application work check and includes tools of multilevel performance analysis. The library also includes auxiliary functional of random, pre-determined or template test data generation, working with both: standard data-types and user-defined data types. This library provides possibilities to directly detect pieces of error code, which may had been written during the development of the program. The library has been developed in compliance with contemporary object-oriented approach on application design using modern algorithmic solutions.