# АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПОКРЫТИЯ ТЕСТАМИ ПРОГРАММНОГО ПРОДУКТА

**Влацкая И.В., канд. техн. наук, доцент,**
**Побежимова Е. В., Секретева А. Д.**
**Оренбургский государственный университет**

В данной статье рассматривается один из подходов к автоматизированной оценки тестового покрытия программного продукта при функциональном тестировании основных алгоритмических конструкций.

Под тестированием программного обеспечения понимают проведение испытаний работоспособности программы с целью обнаружения потенциальных ошибок и недекларированного поведения системы на конечном наборе тестов. Следует отметить, что тестирование - обязательный этап разработки программного продукта. Оно тесно связано с проектированием и разработкой, поскольку может дать ответ на некоторые вопросы о качестве создаваемого продукта. Безусловно, программисты с высоким уровнем профессионализма владеют множеством приёмов тестирования, знают о различных его стратегиях. Но применение всех этих знаний – довольно трудновыполнимая задача. При попытках составить тесты вручную в дело может вступить такой аспект, как человеческий фактор, и возникает риск, что специалист-тестировщик упустит из внимания тот или иной участок кода. И поэтому особо остро встает вопрос об автоматизированной генерации тестовых наборов для программного продукта.

Существуют различные подходы к созданию тестов. Сгенерировать можно как случайные наборы, так и наборы для целенаправленного тестирования тех или иных структур программы. Случайный набор тестов является худшим из вариантов для тестирования кода, поскольку невозможна гарантия проверки работоспособности программы в разных состояниях. При осмысленном тестировании возможны два варианта стратегии проектирования тестов: интеграционное и модульное [1].

Интеграционное тестирование - это тестирование части системы, состоящей из более, чем одного модуля. Смысл такого тестирования заключается в поиске дефектов кода, которые могли возникнуть вследствие недочетов в реализации, и проверке программы на соответствие внешним спецификациям программ и модулей. Данный подход также называется стратегией «черного ящика».

Модульное тестирование - это тестирование отдельно взятых модулей, функций, и т.д. В этом случае составление тестов основывается на знании внутренней структуры программы, на необходимости проверки каждой ветви алгоритма, прохождения каждого возможного участка кода. При этом внешняя спецификация, как ясно из определения, во внимание не принимается. Такое

тестирование, также носящее название «белого ящика», как правило, применяют на ранних стадиях создания программного обеспечения для выявления и дальнейшего устранения преимущественно алгоритмических недочетов. Именно поэтому на этапе модульного тестирования проще всего найти ошибки в написании кода алгоритмов (например, неверная работа с условиями и счетчиками циклов).

В основу разрабатываемой автоматизированной системы оценки покрытия тестами программного продукта ляжет как раз именно тестирование по принципу «белого ящика». Это обосновано тем, что целью является исследование внутренней структуры программного средства – его кода. Иными словами, на выходе должны получиться тестовые наборы, гарантирующие проверку всех независимых маршрутов программы: прохождение ветвей true-false для всех логических решений, выполнение циклов и прохождение граничных значений.

В силу большой трудоемкости и многогранности тестирования программных продуктов в данной системе были введены следующие ограничения:

- язык программирования C#;
- *.cs-файлы должны содержать код консольного приложения;
- код не подразумевает наличие таких типов данных, как массивы, структуры, перечисления, классы, интерфейсы, делегаты, события, и т.д.;
- в функции не передаются никакие параметры - система учитывает только те значения переменных, которыми были инициализированы переменные внутри функций;
- система сильно ограничена в типах данных - она работает только с переменными типа string, int, double;
- структура кода состоит из объявлений переменных и условных конструкций if-else, switch и while;
- переменные, для которых определяются тестовые наборы, должны инициализироваться при помощи операции консольного ввода Console.ReadLine().

При открытии программы пользователь увидит окно ввода C#-кода или загрузки *.cs-файла (рисунок 1). Эти 2 пункта взаимоисключают друг друга, т.е. при выборе «Загрузить файл» вкладка «Ввести данные» автоматически закроется, а содержимое внутри очистится.

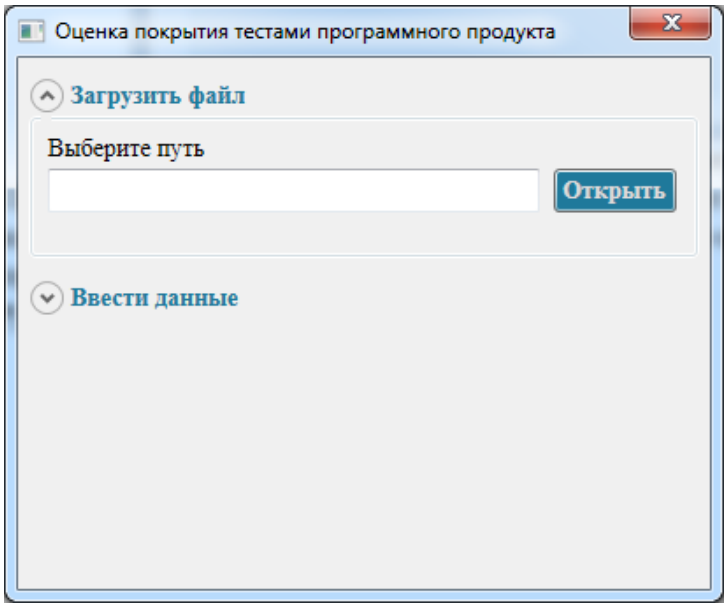


Рисунок 1 – Окно ввода C#-кода или загрузки *.cs-файла

Выбрав вкладку «Ввести данные», пользователь сможет в открывшееся текстовое поле вставить или написать свой код (рисунок 2).

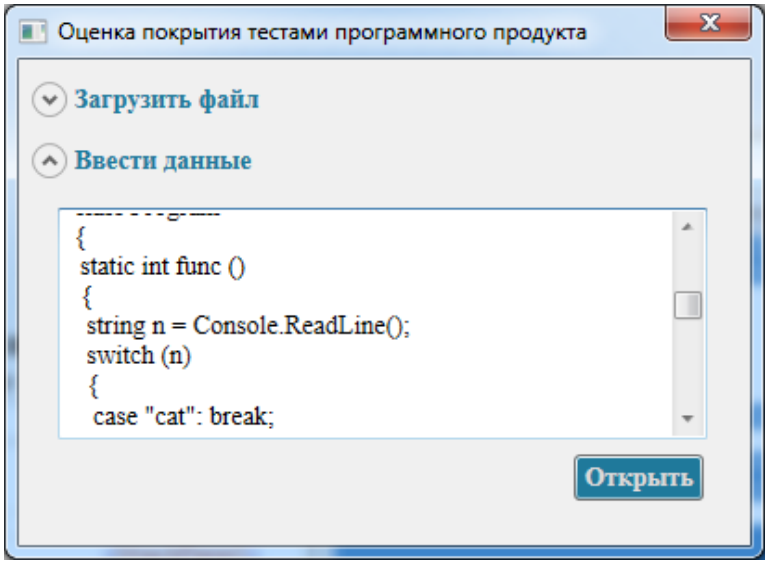


Рисунок 2 – Ввод данных вручную

Нажав на кнопку «Открыть», пользователь будет перенаправлен на окно результатов анализа кода (рисунок 3), а данные переданы системе. После того, как система получила текст C#-кода (введенный в текстовом поле, или же из подгруженного *.cs-файла), она начинает его декомпозицию, вычленяя объявленные методы. Система перебирает строки кода до тех пор, пока не находит строку с синтаксисом типа <модификатор_доступа>*

<объявление_статического_метода>* <тип_метода> <имя метода> (<передаваемые_аргументы>*), где * означает возможное отсутствие в строке. Затем все тем же перебором строк кода определяется тело метода, ограниченное фигурными скобками; при этом учитывается, что внутри метода есть конструкции, чьи границы также обозначаются скобками. Решается эта проблема подсчетом открывающих и закрывающих скобок. На следующем этапе нужно определить переменные, используемые в каждом методе, а также способ их инициализации (ввод с консоли, или же предопределенное кодом программы значение). Здесь уже идет перебор строк тела метода, где система обращает внимание на строки, которые выглядят как <тип_переменной> <имя_переменной_и_ее_инициализация>+;, где + - одно и больше повторений. Такие строки подвергаются детальному разбору, где выявляется, сколько переменных объявлено и какие значения им присваиваются.

В системе определен класс Variables, включающий в себя такие поля, как <имя_переменной>, <тип_переменной>, <значение_переменной>. В специальный список типа Variables заносятся все найденные на предыдущем этапе переменные и их характеристики; при этом, если значение задано в коде, то в список вносится именно оно, а если предполагается, что пользователь должен вводить значение с клавиатуры, ставится пометка «by_user». В свою очередь, такие списки переменных имеет каждый экземпляр класса Procedure, помимо этого содержащий имя метода и его тело. Все методы и их характеристики помещаются в список типа Procedure для упрощения дальнейшего анализа, который начинается после этапа декомпозиции кода.

Анализ представляет собой проход по каждому методу из созданного ранее списка, где, в свою очередь, в теле метода определяется наличие конструкций if-else, while и switch. Когда система находит такие конструкции, то вызывает соответствующую функцию, которая должна проанализировать уже их.

В случае с оператором управления switch система проверяет выражение, передающееся в конструкцию и сравнивает его со списком переменных, определенным в том же методе, что и оператор. Исходя из типа переменной, совпадающей со switch-выражением, ее значения и значений меток case, система генерирует возможные варианты для переменной – как совпадающие с case-метками, так и те, что не совпадают, для прохода всех возможных путей кода.

Когда в коде встречаются условные операторы if-else, функция анализа проверяет условие, записанное в скобках после ключевого слова if. Условие

разбивается на два операнда и стоящий между ними знак. Каждый операнд, как и в случае, описанном выше, проверяется на наличие в списке переменных метода и на принадлежность типу данных. Когда система понимает, что, с чем и как она сравнивает, она предлагает свои варианты значений операндов, исходя, опять же, из принципа, что пройти нужно все варианты путей кода.

После того, как для каждой переменной в коде, чье значение подразумевает ввод с консоли, были определены ее возможные значения, система комбинирует все варианты между собой, получая таким образом тестовые наборы для заданного программного C#-кода.

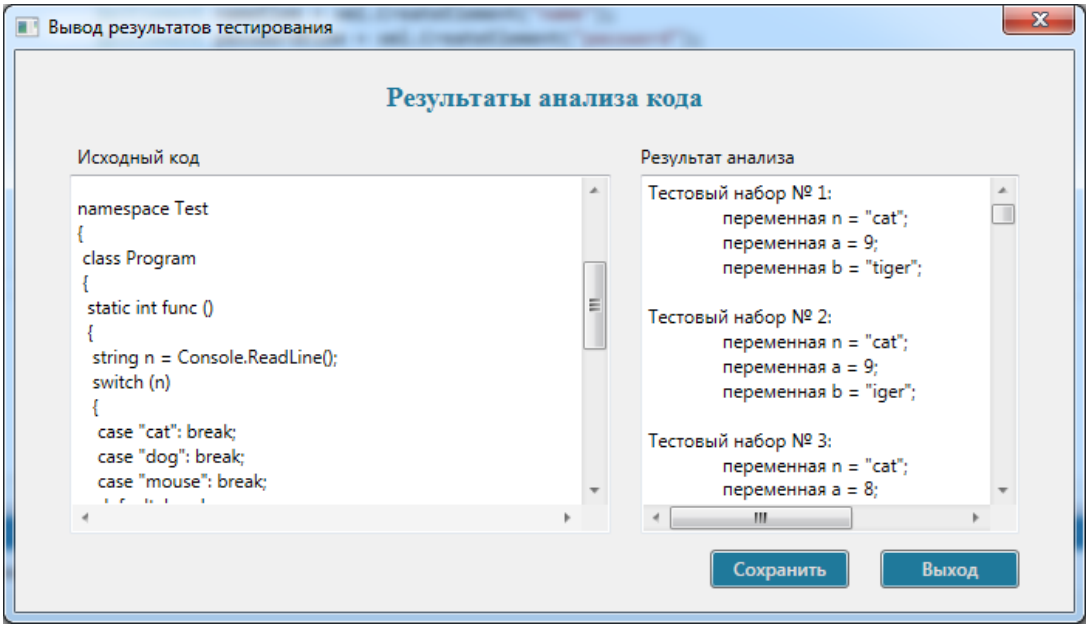


Рисунок 3 – Окно результата анализа кода

Для удобства пользователя полученный результат анализа кода можно сохранить в файле формата *.txt.

При выполнении сохранения пользователь получит окно подтверждения (рисунок 5).

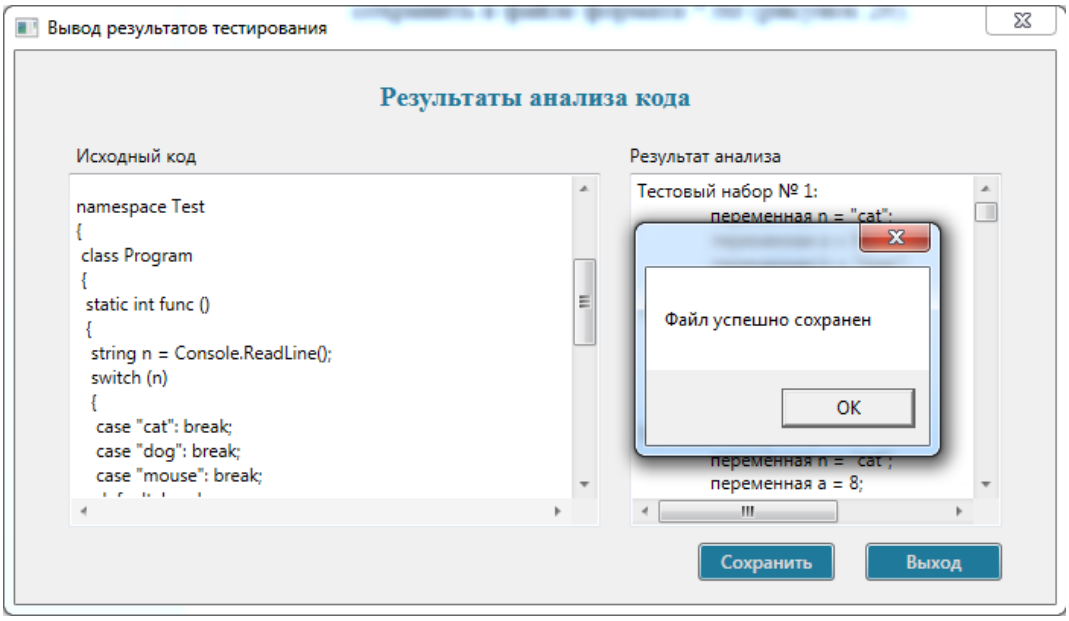


Рисунок 5 – Подтверждение сохранения файла

Аналогичные действия происходят и при загрузке *.cs-файла.

*Список литературы*

1. *Майерс Г. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – 176 с*

2. *An Introduction to Software Testing [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: /~zyl/articles/testing_intro.pdf, свободный.*