ВЕСТНИК ПЕРМСКОГО УНИВЕРСИТЕТА
2010 Математика. Механика. Информатика Вып. 2(2)

УДК 004.4'23

Автоматизация контроля обучения студентов тестированию учебных программ методами "чёрного ящика"

А. С. Петров, М. А. Плаксин, Д. И. Сергеев
Пермский государственный университет, Россия, 614990, Пермь, ул. Букирева, 15
Государственный университет «Высшая школа экономики» (Пермский филиал),
Россия, 614070, Пермь, ул. Студенческая, 38
mapl@list.ru; (342) 2-396-594

При обучении программированию важную роль играет освоение методов тестирования программ. В статье описывается система для автоматизации контроля обучения студентов тестированию учебных программ методами "черного ящика".

Ключевые слова: *программирование для ЭВМ; тестирование; "черный ящик"; автоматизация обучения.*

1. Введение

Описанная в данной статье программа BlackBoxChecker является частью комплекса средств по автоматизации труда преподавателя, ведущего начальное обучение программированию, предназначена для автоматизации контроля обучения студентов тестированию учебных программ методами "черного ящика".

Обучение программированию включает ряд аспектов. К ним относятся постановка задачи, содержательное решение задачи, запись программы на выбранном языке программирования, правильное оформление, тестирование и отладка и др.

На сегодня самым проблемным является обучение тестированию. Неумение тестировать программы мы ощущаем постоянно. Любой пользователь регулярно сталкивается с "зависанием" тех или иных компьютерных приложений, причем как разработанных программистами-одиночками, так и созданных крупными софтверными фирмами.

В школе тестированию не обучают вообще. Поэтому даже те первокурсники, которые пришли в университет, не просто успешно сдав ЕГЭ по информатике, а успешно сдав именно программистскую его часть (часть С), в данном вопросе оказываются совершенно невежественны.

В вузе задача осложняется тем, что учить всем перечисленным методам программирования приходится одновременно. Внимание необходимо рассредоточивать сразу на несколько направлений. Это требует значительных усилий как от обучаемого, так и от преподавателя.

Еще более осложняет ситуацию требование Болонского процесса сократить число аудиторных часов и увеличить долю самостоятельной работы студентов.

Для того чтобы уменьшить нагрузку на преподавателя, предлагается автоматизировать его работу. Создание единой программы "автоматического преподавания информатики" представляется слишком сложным. Более перспективным кажется разделение обучения программированию на несколько аспектов и попробовать автоматизировать преподавание каждого из них отдельно.

Данная работа посвящена автоматизации контроля обучения студентов тестированию учебных программ методами "черного ящика".

2. Постановка задачи автоматизации контроля обучения тестированию учебных программ методами "черного ящика"

Под тестированием методом "черного ящика" подразумевается тестирование программы с точки зрения постановки задачи без учета структуры программы[1].

Теория тестирования требует, чтобы первые тесты разрабатывались сразу же после постановки задачи до написания текста программы. Цель этих тестов – уточнить понимание задачи до того, как будут потрачены усилия на ее решение. Естественно, что в тот момент, когда текст программы еще просто не написан, программист может пользоваться только методами "черного ящика".

Теория тестирования предлагает некоторый канонический набор (схему) критериев, которыми следует руководствоваться при построении тестов "черного ящика". На основе этой схемы для каждой конкретной задачи строится конкретный набор критериев, отражающий специфику именно этой задачи. После этого студент должен разработать набор тестов, который соответствовал бы предложенному набору критериев. Набор должен быть полон (т.е. должен быть хотя бы один тест, который проверял бы каждый из критериев) и неизбыточен (т.е. каждый из критериев в идеале должен проверяться ровно одним тестом).

Таким образом, контроль тестирования методами "черного ящика" разбивается на две подзадачи. Во-первых, требуется оценить, насколько правильно сформулирован набор критериев "черного ящика" для конкретной задачи. Во-вторых, надо оценить, насколько предложенный набор тестов соответствует предложенному набору критериев.

Достаточно очевидно, что сформулированная задача в общем виде неразрешима. Вообще говоря, задача может быть сформулирована на естественном языке и формулировка может быть любой степени сложности. Соответственно на естественном языке будут сформулированы критерии, а тесты могут представлять собой весьма сложные структуры данных.

В таком виде поставленная задача автоматически решена быть не может. Но нам и не надо решать ее в таком общем виде. Речь идет о гораздо более узкой задаче. В самом начале курса программирования студенты получают задание, предусматривающее решение большого числа (20) очень простых задач[2]. Цель этого задания двояка. Для студентов, которые в школе программированию не обучались, это задание является заданием на разработку ряда простых программ. Для студентов, которые программировать уже немножко умеют, составление соответствующих программ трудности не представляет. Они могут полностью сосредоточиться на таких аспектах, как тестирование программ методами "черного и белого ящиков", отработка стиля программирования.

В "первом приближении" от BlackBox-Checker'а требуется проконтролировать правильность тестирования студентами набора именно этих, очень простых программ. В крайнем случае можно пойти на то, чтобы просто зафиксировать набор задач, выдаваемых студентам. Именно эти 20 и ни какие другие! Это позволит вручную составить для каждой задачи эталонный набор критериев и эталонный набор тестов.

В таком, сильно упрощенном виде задачу уже можно попробовать решить.

В заключение данного параграфа одно замечание по сравнению предлагаемого подхода с существующим положением. Идея автоматического тестирования разработанных студентами программ не нова. Однако в настоящее время наиболее распространенным методом автоматической проверки студенческих программ является применение проверяющих систем по типу олимпиадных задач. В таких системах задается строгий формат ввода/вывода данных, каждая студенческая программа прогоняется на заданном наборе

[1] В отличие от тестирования методами "белого ящика", которые требуют учета структуры программы.

[2] Примеры задач из начального набора:
Ввести число, если оно больше нуля, напечатать его квадрат, если меньше – куб.
Ввести три числа (a, в, с), если они удовлетворяют неравенству a >= в >= с, удвоить их, иначе – напечатать их абсолютные величины.
Ввести последовательность чисел, посчитать их среднее арифметическое, среднее арифметическое положительной и отрицательной подпоследовательностей.
Ввести последовательность чисел. Проверить, образуют ли числа в четных позициях монотонную подпоследовательность.

тестов и результаты каждого прогона сравниваются с эталонными.

Такой подход, несомненно, экономит усилия преподавателя. Однако к поставленной нами задаче – контролю освоения студентами умений и навыков тестирования программ – он никакого отношения не имеет. Выдача студенту сообщения о том, что его программа прошла или не прошла проверку на таком-то тесте, ничего не говорит обучаемому о том, откуда эти тесты взялись, почему их именно столько и почему они именно таковы.

3. Реализация системы BlackBox-Checker

Для решения поставленной задачи была разработана система, которая на основе текста задания на естественном языке строит критерии для данного задания с точки зрения тестирования методом "черного ящика". Система позволяет редактировать (удалять, изменять, добавлять) автоматически созданные критерии и оценивать набор тестов и критериев, предложенных студентом для выбранного задания. Схема системы приведена на рис. 1.

Далее дается пояснение этой схемы и приводится пример работы системы.

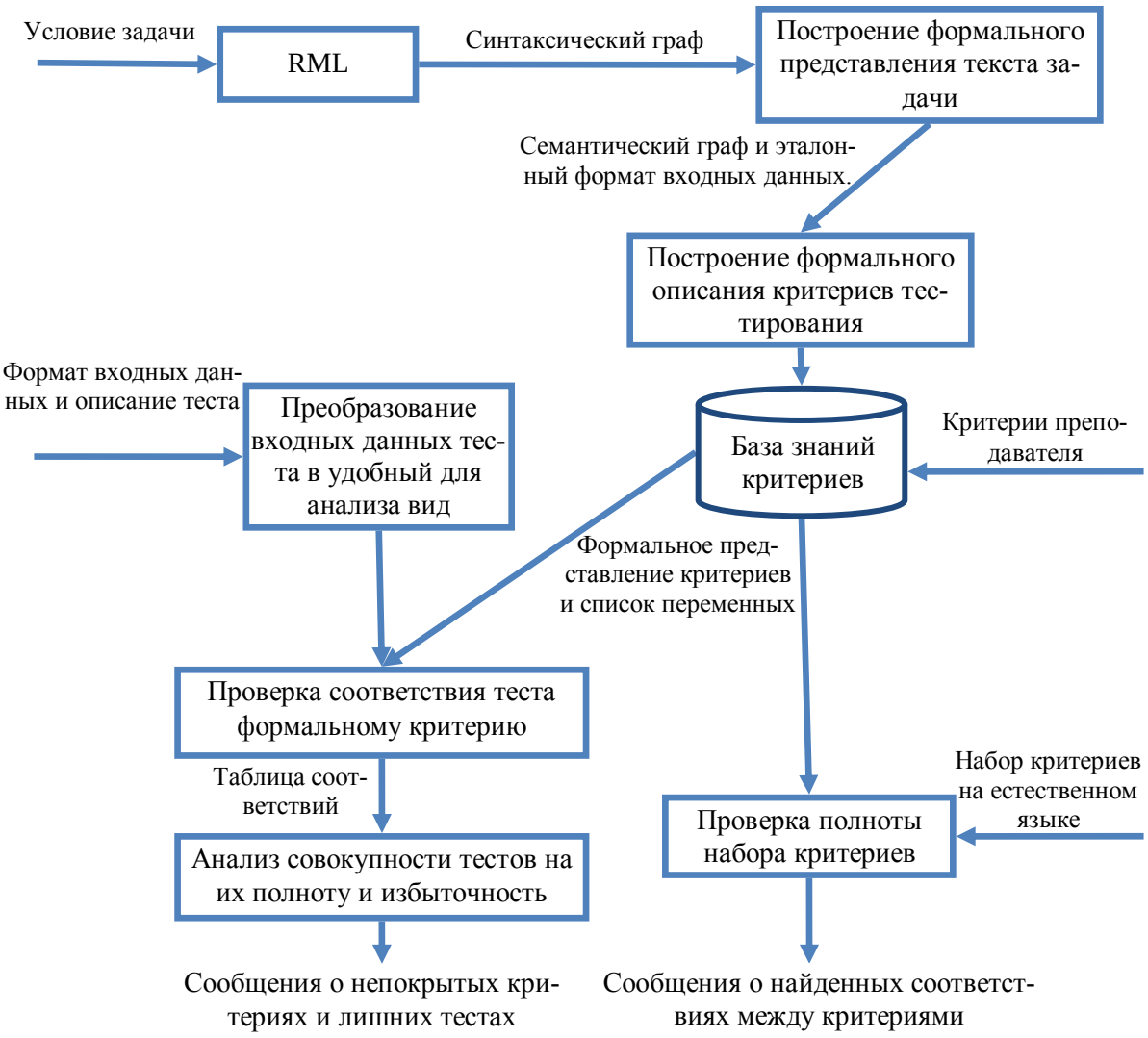

Рис. 1. *Схема системы BlackBoxChecker*

Построение формального представления текста задачи

Текст задачи представляет собой записанное на русском языке ее условие с указанием входных и выходных данных.

Текст задачи должен быть преобразован в некоторый формальный вид, по которому можно будет строить критерии.

Для этого текст задачи подается на вход стороннему синтаксическому анализатору (RML), который строит синтаксический граф. Далее, на основе онтологии (рис. 2) в этом графе выделяются переменные, операции над переменными и связи между ними. Таким образом, строится семантический граф.

Кроме того, для каждой задачи на основе выделенных переменных строится эталонный формат входных данных.

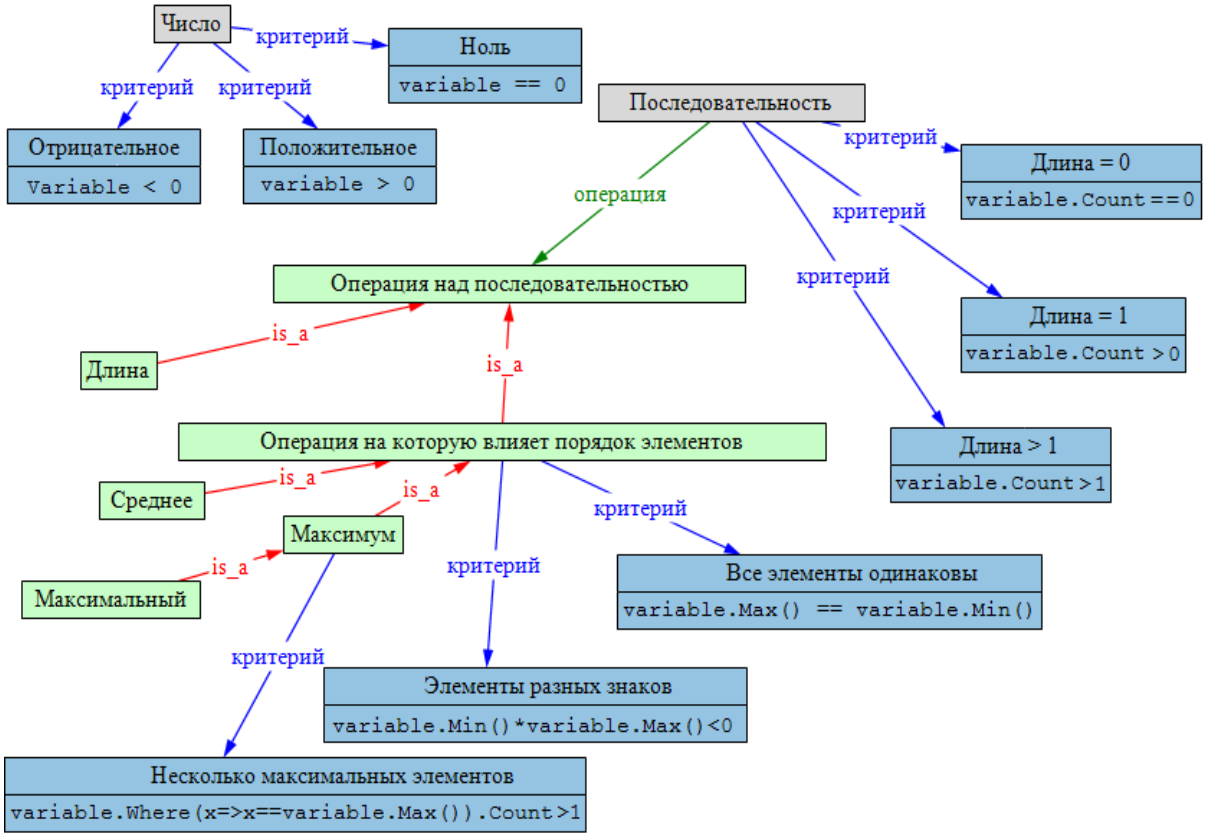

Рис. 2. *Фрагмент онтологии предметной области*

Построение формального описания критериев тестирования

На основе полученного формального описания задачи строится формальное описание критериев тестирования.

Формальное описание одного критерия формулируется как логическая функция от входных данных задачи, записанная на каком-либо языке. Значение функции показывает, исполняется ли данный критерий при прогоне программы с этими входными данными.

В качестве языка представления критериев был выбран Microsoft .NET C# 3.0, в котором, по сравнению с предыдущей версией, появилось существенное нововведение – встроенный механизм запросов LINQ, который добавляет к массивам так называемые методы расширения, реализующие основные агрегирующие и преобразующие функции.

BlackBoxChecker автоматически генерирует некоторый набор критериев. Преподаватель имеет возможность откорректировать этот набор: добавить новые критерии, удалить или заменить существующие.

Сформированный набор критериев служит эталоном для сравнения с набором критериев, предложенных студентом.

Автоматически набор критериев строится по следующим правилам.

Для каждой переменной в список критериев добавляются критерии из онтологии, связанные с типом переменной.

Для каждой пары переменная - операция, в список критериев добавляются крите-

рии из онтологии, находящиеся на пути движения от вершины, соответствующей типу данных переменной, к вершине, соответствующей операции.

Анализ набора критериев, предложенных студентом, на полноту и избыточность

Для проверки предложенного студентом набора критериев на полноту и избыточность проводится сопоставление студенческого набора критериев, с эталонным, построенным системой и/или преподавателем. К сожалению, простое сравнение в данном случае не годится. Причиной является неоднозначность описания критерия и невозможность точно распознать составленный студентом набор критериев.

Сопоставление студенческого набора тестов с эталонным выполняется по следующим правилам.

Для простейших случаев (для критериев, в которых используется только одна переменная) делается попытка сопоставления студенческого критерия с эталонным с помощью операции сопоставления с образцом. Это возможно, если студент при описании критерия точно следовал стандартному формату (имя переменной – двоеточие – словесное описание критерия). В случае удачи соответствие критериев определяется однозначно. Но такой случай является, скорее, исключением, чем правилом.

В случае неудачи сопоставление производится по ключевым словам. Описание эталонного критерия содержит комплект ключевых слов. Причем для каждого ключевого слова может быть задано множество синонимов. Этот комплект сопоставляется поочередно с описанием каждого из студенческих критериев. Сопоставление учитывает синонимы и словоформы русского языка. (Отметим, что словоформы учитываются системой автоматически).

При неудачной попытке поиска по ключевым словам система считает, что предложенный студентом набор критериев неполон.

При нескольких удачных сопоставлениях, набор студенческих критериев считается избыточным.

Преобразование входных данных теста в удобный для анализа вид

Описание теста включает описания формата входных/выходных данных тестируемой программы и непосредственно сами входные/выходные данные программы.

Как уже было сказано, в отличие от систем для проверки олимпиадных задач BlackBoxChecker дозволяет студенту самому определять формат входных и выходных данных. Но в чистом виде такая свобода делает автоматический анализ тестов, практически, невозможным. Поэтому от студента требуется предоставить BlackBoxChecker'у некоторое описание выбранного им формата ввода/вывода. В общем виде эта задача кажется неразрешимой. Но нас интересует описание форматов для очень простых задач. Самое сложное описание, которое требуется, – это описание последовательности чисел. Для подобных описаний достаточно весьма простого языка.

Проверка соответствия теста формальному критерию

Чтобы узнать, покрывает ли тест, написанный студентом, выбранный критерий, нужно вычислить логическую функцию, соответствующую критерию, применительно к выбранному тесту.

Для функций-критериев используется метод динамической компиляции кода в библиотеку. В результате проверка соответствия критерия тесту сводится к обращению к функциям скомпилированной библиотеки. Это обеспечивает преподавателю возможность подмены критериев в наборе критериев, построенных системой автоматически.

Анализ набора тестов, предложенных студентом, на полноту и избыточность

Анализ состоит в проверке: все ли критерии покрыты набором тестов и нет ли таких тестов, изъятие которых сохраняет полноту покрытия.

В результате проверки каждого теста на каждом критерии получается матрица, строки которой соответствуют тестам, столбцы – критериям, элемент матрицы равен "+" если тест покрывает критерий, и равен "–" в противном случае.

Набор тестов полон, если в каждом столбце есть хотя бы один плюс.

Задача об избыточности ставится так. В матрице, состоящей из плюсов и минусов, выбрать минимальный по включению набор столбцов такой, что в каждой строке среди этих столбцов есть хотя бы один плюс. Если представить каждый тест как подмножество множества всех критериев, состоящее из тех критериев, которые этот тест покрывает, то задача легко идентифицируется как задача о наименьшем покрытии множества.

4. Пример применения системы BlackBoxChecker

Рассмотрим пример работы системы.

Задача: Ввести последовательность A из N чисел, найти номер максимального нечетного числа.

Работа идет в следующем порядке.

1) Текст задачи подается на вход синтаксическому анализатору. Результатом его работы является синтаксический граф (рис. 3).

2) По синтаксическому графу на основе онтологии (рис. 2) строится семантический граф (рис. 4) и эталонный формат входных данных (записывается на языке описания формата входных данных).

Для этого выделяем типы переменных: *A – последовательность (рис. 5), N – число (рис. 6).*

Эталонный формат входных данных: N: Число; A: Последовательность[N];

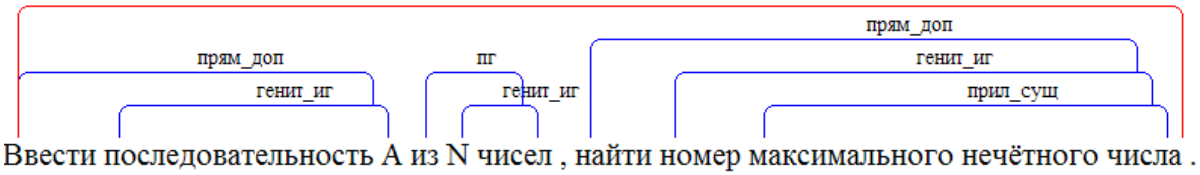

Рис. 3. *Синтаксический граф для задачи*

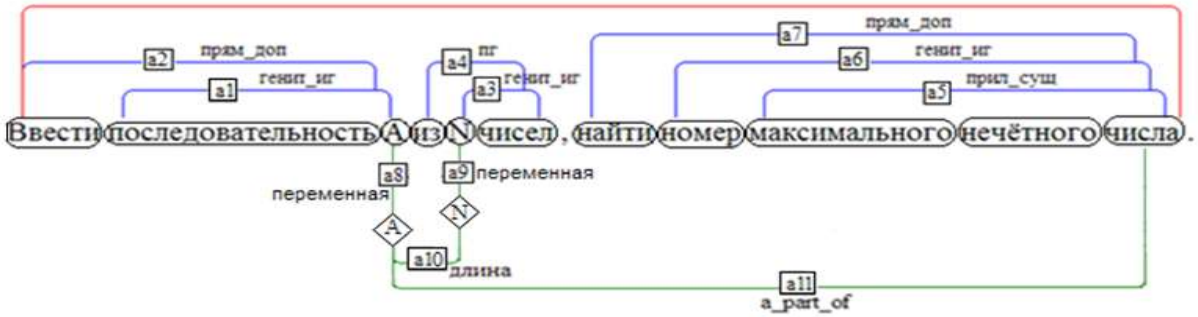

Рис. 4. *Семантический граф для задачи*

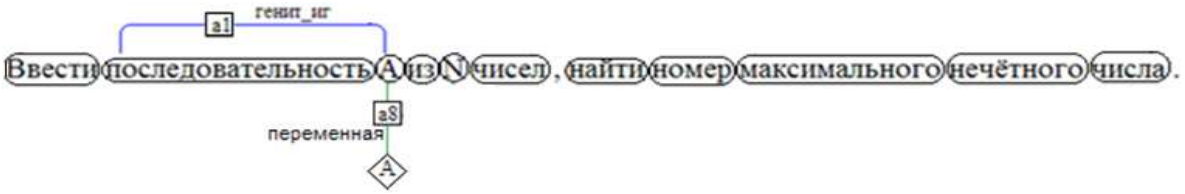

Рис. 5. *Поиск типа данных для переменной A*

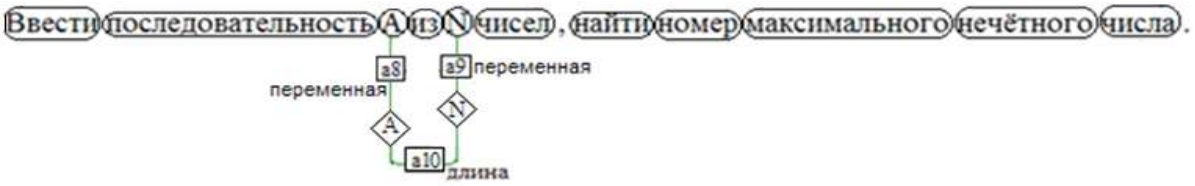

Рис. 6. *Поиск типа данных для переменной N*

3) На основе синтаксического графа, формата входных данных и онтологии (рис. 2) строится формальное описание критериев.

Для переменной A типа Последовательность набор критериев будет иметь вид:

Неформальное описание	Формальное описание
A: Длина=0	A.Count() == 0
A: Длина=1	A.Count() == 1
A: Длина>1	A.Count() > 1

Для переменной N типа Число набор критериев будет иметь вид:

Неформальное описание	Формальное описание
N: положительное	N>0
N: отрицательное	N<0
N: ноль	N==0

Неформальное описание	Формальное описание
A: Все элементы одинаковы	A.Max() == A.Min()
A: Несколько максимальных элементов	A.Where(x=>x == A.Max()).Count()>1
A: Элементы разных знаков	A.Min()*A.Max()<0

Рис. 7. *Поиск отношения между переменной A и операцией поиска максимального элемента*

Критерии для операции поиска максимального элемента приведены на рис. 7.

4) Проверка набора тестов, предложенного студентом для задачи (рис. 8).

Каждый тест проверяется на всем наборе критериев, построенном в предыдущем пункте. Результат тестирования приведен на рис. 9.

формат	N: Число; A: Последовательность[N];
№	входные данные
1	-5
2	0
3	3 8 3 8

Рис. 8. *Пример описания тестового набора студентом*

Неформальное описание	Формальное описание	Тест1	Тест2	Тест3
N: положительное	N>0	–	–	+
N: отрицательное	N<0	+	–	–
N: ноль	N==0	–	+	–
A: Длина=0	A.Count() == 0	+	+	–
A: Длина=1	A.Count() == 1	–	–	–
A: Длина>1	A.Count() > 1	–	–	+
A: Все элементы одинаковы	A.Max() == A. Min()	–	–	–
A: Несколько максимальных элементов	A.Where(x=>x==A.Max()).Count()>1	–	–	–
A: Элементы разных знаков	A.Min()*A.Max()<0	–	–	–

Рис. 9. *Результат проверки набора тестов*

5) Анализ результатов тестирования.

На основе полученной таблицы соответствий делается вывод о полноте тестирования. В данном случае будет выведено сообщение о недостаточности набора тестов, так как он не покрывает 4 критерия из 9.

6) Проверка полноты набора критериев. Кроме формального описания эталонные критерии содержат дополнительную информацию (рис. 10) для их сопоставления с критериями, предложенными студентом (рис. 11). Результат сопоставления показан на рис. 12.

Описание критерия	N: отрицательное
Формальное описание	N<0
Ключевые слова	N<0, длина, отрицательная

Описание критерия	A: Все элементы одинаковы
Формальное описание	A.Max() == A.Min()
Ключевые слова	последовательность, из, одинаковые, все, элементы

Рис. 10. *Фрагмент полного описания критериев*

№	Описание критерия
1	n<0
2	n=0
3	n>1
4	последовательность из одинаковых элементов
5	вывод максимального числа
6	недостаточно элементов для сравнения

Рис. 11. *Критерии, предложенные студентом*

Элемент	Статус
✓ n<0	точное соответствие
✓ n=0	точное соответствие
✓ n>1	точное соответствие
✓ последовательность из одинаковых элементов	по ключевым словам
✗ вывод максимального числа	нет соответствия
✗ не достаточно элементов для сравнения	нет соответствия
✗ Эталонный Длина последовательности = 0	не обнаружен
✗ Эталонный Длина последовательности = 1	не обнаружен
✗ Эталонный Длина последовательности > 1	не обнаружен
✗ Эталонный A: Несколько максимальных элементов	не обнаружен
✗ Эталонный A: Элементы разных знаков	не обнаружен

Рис. 12. *Анализ критериев, составленных студентом*

5. Заключение

Описанный в статье вариант программы BlackBoxChecker является по сути демоверсией. Достигнутые результаты оказались значительно выше тех, что изначально можно было ожидать от решения задачи такой степени сложности, но значительно ниже тех, что можно было бы на практике применять в учебном процессе.

Задача проверки набора студенческих тестов на полноту и избыточность решена удовлетворительно. Сложности связаны с анализом предложенных студентом критериев. Достигнутый уровень надежности сопоставления студенческих критериев с эталонными недостаточен для внедрения системы в учебный процесс. Причина этого в недостатках используемого метода сопоставления по ключевым словам.

В качестве направления дальнейшего развития имеет смысл попробовать совершенствование описания эталонных критериев за счет выбора оптимального множества ключевых слов (словосочетаний) и совершенствование алгоритма сопоставления студенческих критериев с эталонными.

Открытым остается вопрос о методике применения системы BlackBoxChecker. Должна ли она быть инструментом преподавателя или ее имеет смысл передать студентам?

Список литературы

1. *Бейзер Б.* Тестирование "черного ящика". Технологии функционального тестирования программного обеспечения и систем. СПб.: Питер, 2004. 320 с.
2. *Гирш Э.А.* Курс лекций по эффективным алгоритмам. URL: http://www.csin.ru/ courses/effektivnye-algoritmy (дата обращения 20.06.2009).
3. *Зализняк А.А*. Грамматический словарь русского языка: словоизменение. М.: Русский язык, 1987.
4. *Калбертсон Р., Браун К., Кобб Г.* Быстрое тестирование. М.: Вильямс, 2002. 384 с.
5. *Котляров В.П*. Критерии выбора тестов. URL: http://www.excode.ru/art6095p1.html (дата обращения 20.06.2009).
6. *Котляров В.П., Коликова Т.В.* Основы тестирования программного обеспечения. М.: Бином, 2006. 285 с.
7. *Леонтьева Н.Н.* Строение семантического компонента в информационной модели автоматического понимания текста: автореф. и дис. д.т.н. М., 1990.
8. *Плаксин М.А.* Тестирование и отладка программ – для профессионалов будущих и настоящих. М.: Бином. Лаборатория знаний, 2007. 167 с.
9. *Эллиотт Х.* Тестирование методом "черного ящика" URL:http://www.ibm.com/ developerworks/ru/library/j-fuzztest/ index.html (дата обращения 20.11.2008).
10. *Myers G.J.* The Art of Software Testing, 2nd Edition. 2004. 234 p.
11. *Pollice G.* Test before you code. URL: http://www.ibm.com/developerworks/rational/library/4929.html?S_TACT=105AGX99&S_CMP=CP (дата обращения 20.06.2009).

Automation of the checking of training of students to testing of programs by methods of "a Black box"

A. S. Petrov, M. A. Plaksin, D. E. Sergeev
Perm State University, Russia, 614990, Perm, Bukireva st., 15
State University Economy School, Russia, 614070, Perm, Student's st., 38
mapl@list.ru; (342) 2-396-594

At training to programming the important role is played by development of methods of testing of programs. In article the system for automation of the cheking of training of students to testing of programs by methods of "a black box" is described.

Key words: *computer programming; testing of computer programs; "black box" methods; automation learning.*