ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2020 Управление, вычислительная техника и информатика № 53

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 004.434
DOI: 10.17223/19988605/53/11

И.А. Жуков, Ю.Л. Костюк

МОДЕЛЬ ПРЕДСТАВЛЕНИЯ МНОГОВАРИАНТНЫХ ЗАДАНИЙ ДЛЯ АВТОМАТИЗИРОВАННОГО КОНТРОЛЯ ЗНАНИЙ ПО ПРОГРАММИРОВАНИЮ

Рассмотрена проблема автоматизированного контроля знаний при обучении программированию. Существующий подход к проверке программ, основанный на тестировании методом черного ящика, исключает автоматизированный семантический анализ программного кода. Описана модель представления многовариантных заданий, основанная на конструктивно-выборочном методе. Предложен формальный язык описания заданий при помощи множества шаблонов. Разработаны два алгоритма автоматизированного контроля знаний.
Ключевые слова: автоматизированный контроль знаний; конструктивно-выборочный метод; обучение программированию.

Вопросы автоматизации контроля знаний актуальны на протяжении многих лет. В частности, технологии автоматизированного контроля применяются для дисциплин укрупненных групп специальностей и направлений (УГСН) 02.00.00 «Компьютерные и информационные науки» и 09.00.00 «Информатика и вычислительная техника». Большинство профессиональных дисциплин данных УГСН связанно с программированием. Многие авторы ищут новые подходы для совершенствования традиционного метода обучения программированию. Автор [1] предлагает применять методики, использующие интерактивные технологии и стимулирующие методы. Проведен анализ успеваемости двух групп: экспериментальной, для которой лекционные занятия строились на основе специально созданного программного комплекса, и контрольной, в которой применялась традиционная методика преподавания. Автор отмечает, что уровень знаний в экспериментальной группе оказался выше, чем в контрольной. Сделан вывод, что при обучении программированию более эффективными будут отход от традиционных методов преподавания и внедрение в обучение новых методик, использующих интерактивные технологии и стимулирующие методы.

Важную роль в учебном процессе играет контроль знаний. Основным методом автоматизированного контроля знаний является альтернативно-выборочный метод (тесты). Для изучения основ программирования такой метод применим. Например, в [2] описан разработанный авторами алгоритм генерации тестов для проверки теоретических знаний и навыков владения студентами языком программирования в рамках дисциплины «Основы программирования на языке Delphi 7.0». Отметим, что для проверки навыков применяются тесты в формате: «Что будет являтся результатом работы приведенного фрагмента программного кода?»

Помимо тестов требуется более глубокий контроль, при котором обучающийся мог бы не просто выбирать один или несколько ответов из предложенных. Многие стараются решить этот вопрос, разрабатывая новые методики и техники контроля знаний или поддержки процесса обучения. Например, авторы [3] разрабатывают систему поддержки обучения тестированию программного обеспечения. Для этого они создали язык описания диаграмм, который основан на высказываниях, причем

каждое высказывание относится к одному из двух типов: «функция» или «ограничение». Язык может транслироваться в более «низкоуровневые» языки описания графов. Авторы [4] предлагают систему обучения программированию, в которой учащиеся составляют интеллект-карту для ответа на задание. Обсуждается идея игры по теме «Программирование», основанной на принципе интеллект-карты. Обучающемуся предлагается ряд задач, и ему следует выбрать несколько операторов, необходимых для решения задачи, их порядок, а затем выбрать типы данных для переменных.

Особенности обучения будущих программистов заключаются в том, что необходимо огромное количество практики, а именно написание тысяч строк программного кода. В образовательных программах предусмотрены лабораторные работы, в рамках которых обучающиеся разрабатывают программы для решения поставленных учебных задач. Преподаватель оценивает знания и полученные навыки обучающихся в ходе личной беседы или чтения отчетов. Одним из способов автоматизации этого процесса является автоматическое тестирование программ. В таком случае применяется поведенческое тестирование, где в качестве спецификации, в которой описаны входные данные и ожидаемый результат, используются условия задачи. Поведенческое тестирование также называют тестированием черного ящика [5], или тестированием методом черного ящика. Такой подход, например, используют авторы [6], которые разработали систему поддержки процесса обучения программированию, предлагающую задания в форме тестов для проверки теоретических знаний и тестирование программ для проверки умений и навыков. Для тестирования программ авторы используют набор тестов, включающий в себя входные и выходные данные, а также максимально допустимое время выполнения программы.

Является ли тестирование черного ящика решением проблемы автоматизации контроля практических навыков программирования? В [7, 8] рассматриваются различные подходы к автоматизации контроля знаний при обучении программированию. В работе [7] для контроля знаний по программированию, помимо тестирования теоретических знаний студентов, применяется автоматическое тестирование программ, написанных в рамках лабораторных работ. В [8] отмечена двойственность систем автоматизированного тестирования по программированию: для преподавателей выполняется формализованная проверка программы с отчетом о функциональности решений, а студентам необходимо применять творческие умения. Сделан вывод о недостаточности тестирования методом черного ящика, которое не позволяет определить реальные причины некорректной работы программы, так как полностью исключается семантический анализ программного кода. Заметим, что при обучении основам программирования студенты изучают определенный набор базовых алгоритмов. На лабораторных занятиях они пишут программный код, реализуя эти алгоритмы. Тестирование таких программ методом черного ящика не позволяет оценить уровень владения студентом тем или иным базовым алгоритмом. Например, проверка отсортированного массива свидетельствует только о том, произошла ли сортировка исходных данных, но не дает информации о степени владения студентом конкретными алгоритмами. Косвенным признаком может служить время исполнения программы, которое зависит от трудоемкости алгоритма. Однако единственным надежным способом является семантический анализ исходного кода программы. Таким образом, контроль ответа без контроля хода решения нельзя признать исчерпывающим. Тестирование методом черного ящика дает лишь представление о функционировании программы, но не о том, как она написана.

Оценка преподавателем программного кода, написанного каждым обучающимся, при массовом обучении требует больших затрат времени. Поэтому актуален вопрос создания инструментов, осуществляющих первичный (предварительный) контроль знаний базовых алгоритмов. При обучении основам программирования полезно давать обучающимся задания на составление решения (программного кода) из предложенных фрагментов (компонентов). Следует учесть, что в программировании многие фрагменты даже простейших программ могут быть написаны разными способами, но выполнять одно и тоже действие. Такие задания будем в дальнейшем называть многовариантными.

В данной статье рассматривается задача автоматизации контроля решения многовариантных заданий, при котором исключается семантический анализ программного кода преподавателем вручную. Для этого разработана модель представления многовариантных заданий, а также алгоритм контроля.

1. Модель представления многовариантных заданий

В [9] изложена идея конструктивно-выборочного метода, при котором обучающийся составляет свой ответ из элементов, являющихся допустимыми смысловыми единицами. В дальнейшем такие единицы будем называть компонентами. Авторы [10] отмечают, что конструктивно-выборочный метод является более удачным, чем альтернативно-выборочный, так как требует творческого подхода от обучающегося и уменьшает вероятность угадывания правильного ответа. В программировании в качестве компонента логично взять одну строку программного кода. Даже для простейшей программы существуют десятки компонент, образующих в различных комбинациях правильные решения. Для преподавателя возможно три способа подбора компонентов:

1) из предложенных компонентов можно составить единственное верное решение, причем каждый компонент входит в верное решение;

2) из предложенных компонентов можно составить единственное верное решение, причем часть компонентов не входит в верное решение (т.е. являются избыточными);

3) из предложенных компонентов можно составить несколько верных решений, причем в каждом случае некоторые компоненты не входят в верное решение, а также допустимы компоненты, не входящие ни в одно решение.

Отметим, что каждый следующий способ подбора компонентов включает в себя предыдущие, которые для него являются частными случаями.

В разработанной модели за основу взят третий способ подбора компонентов, так как в программировании правильным решением конкретной задачи может быть множество программ. Обучающийся предлагает свой вариант решения в виде упорядоченной последовательности, составленной из множества предложенных компонентов. С формальной точки зрения вариант решения – конкретная упорядоченная последовательность компонентов. Вариант решения может быть правильным (верным) или неправильным (неверным). Все правильные варианты решения должны быть описаны конечным множеством шаблонов. Каждый шаблон состоит из элементов. С точки зрения программирования элемент – какое-либо действие, состоящее из одной или нескольких строк программного кода. То есть элемент – последовательность компонентов. В модели выделено три типа элементов: один компонент, один из компонентов, перестановка компонентов. Элемент «один из компонентов» представляет собой список компонентов, каждый из которых удовлетворяет верному решению, но любое верное решение содержит только один компонент из этого списка. Элемент «перестановка компонентов» представляет собой список компонентов, любая перестановка которых удовлетворяет верному решению.

Длиной шаблона будем называть количество элементов в шаблоне. Каждый шаблон для конкретного задания может иметь разную длину.

Предложенный обучающимся вариант решения оценивается в баллах. Максимальная оценка принята равной единице. Итоговый балл складывается из оценок за каждый элемент. Если итоговый балл попадает в интервал (0, 1), то решение считается частично правильным. По умолчанию все элементы равнозначны, вес каждого элемента равен единице, деленной на длину шаблона. По усмотрению преподавателя элементам могут быть назначены разные веса.

Каждый элемент может быть отмечен как рубежный или как допускающий отсутствие (опциональный). Отсутствие непомеченного элемента добавляет в итоговую оценку 25% его веса. Если в ответе отсутствует элемент, отмеченный как «допускается отсутствие», при расчете итогового балла добавляется 50% веса пропущенного элемента. Если отсутствует рубежный элемент, то в итоговую оценку не добавляется ничего. Если в приведенном ответе отсутствуют все рубежные элементы, ответ оценивается в ноль баллов. Элемент «перестановка компонентов» не может быть помечен как рубежный. Любой элемент, стоящий после элемента «перестановка компонентов», всегда считается рубежным. Следовательно, любой шаблон не может содержать два элемента «перестановка компонентов» подряд. Один компонент может встречаться в нескольких элементах внутри одного шаблона, но два соседних элемента не могу содержать одинаковые компоненты.

2. Язык описания шаблонов

Для записи шаблонов разработан специализированный формальный язык. Описание предыдущей версии языка и выбранного алгоритма синтаксического анализа приведено в [11]. В данной статье обсуждается расширенная версия языка. Основным нововведением является возможность пометки элементов. Приведем описание языка в расширенной форме Бэкуса–Наура (РБНФ), жирным шрифтом выделены метасимволы, относящиеся к РБНФ:

Задание = Шаблон **{**"|" Шаблон**}**

Шаблон = "{" Элемент **{**";" Элемент**}** "}"

Элемент = "["**? ((**ОдинКомпонент | ОдинИзКомпонентов**)** "*" | ПерестановкаКомпонентов**))** "]"**?**

ОдинКомпонент = ИдентификаторКомпонента

ЦифрыБезНуля = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Цифры = "0" | ЦифрыБезНуля

ИдентификаторКомпонента = ЦифрыБезНуля **{**Цифры**}**

ОдинИзКомпонентов = ОдинКомпонент "|" ОдинКомпонент **{**"|" ОдинКомпонент**}**

ПерестановкаКомпонентов = "(" ОдинКомпонент ";" ОдинКомпонент **{**";" ОдинКомпонент**}** ")".

В языке используются терминальные символы: круглые скобки, квадратные скобки, фигурные скобки, вертикальная черта («|»), точка с запятой и звездочка. Шаблоны разделяются вертикальной чертой. Компонент решения нумеруется целым числом, начиная с единицы. Элементы внутри шаблона разделяются точкой с запятой. В элементе «один из компонентов» компоненты разделяются вертикальной чертой. В элементе «перестановка компонентов» в круглых скобках записываются компоненты, разделенные точкой с запятой. Элемент, допускающий отсутствие, находится в квадратных скобках. Рубежный элемент помечается звездочкой.

Задание, записанное на языке шаблонов, транслируется в массив массивов словарей (формат JSON). Процесс трансляции состоит из трех этапов: лексический анализ, синтаксический анализ, трансформация дерева разбора. На входе лексический анализатор получает запись на языке шаблонов, выходом будет список лексем, после этого список лексем передается в синтаксический анализ, который строит дерево разбора. Затем при помощи специального алгоритма каждой вершине дерева ставится в соответствие запись в массиве массивов словарей. Шаблон представляется массивом словарей, задание – массивом шаблонов. Каждый элемент описывается словарем с тремя ключами: type, components и flag. Ключ type соответствует типу элемента; значением является целое число в интервале [1, 3], где 1 – один компонент, 2 – один из компонентов, 3 – перестановка. Ключ flag соответствует метке элемента; значением является целое число в интервале [0, 2], где 0 – метка отсутствует, 1 – рубежный элемент, 2 – элемент, допускающий отсутствие. Ключ components соответствует массиву компонентов, из которых состоит элемент.

Приведем пример задания по дисциплине «Основы программирования» на языке программирования Pascal и его записи на языке шаблонов.

Пример. Из заданных ниже компонентов составьте программу, результатом работы которой будет вычисление суммы всех элементов массива *m*. Длина массива *n*. Результат суммирования должен быть записан в переменную *S*.

1. S:=0;	4. i:=1;	7. i:=i + 1;	10. repeat	13. i++;
2. for i:=1 to n do	5. while i <= n do	8. inc(i);	11. until i > n;	14. repeat do
3. S:=S + m[i];	6. begin	9. end;	12. i += 1;	15. until i < n;

Обучающемуся для составление варианта решения предложено 15 компонентов, среди них есть избыточные компоненты: компоненты 7 и 8 идентичны по сути, компоненты 12–14 являются неверными с точки зрения синтаксиса языка Pascal, а компонент 15 не входит в какое-либо правильное решение.

Описания задания на языке шаблонов:

{[1];2;3*}|{(1;4);5;6;3*;7|8;9}|{(1;4);10;3*;7|8;11}

Первый шаблон – [1];2;3*. Он состоит из трех элементов. Каждый элемент относится к типу «один компонент». Первый элемент помечен как допускающий отсутствие. Последний элемент помечен как рубежный. Для этого шаблона правильным вариантом решения будет упорядоченная последовательность компонентов 1, 2 и 3. Частично верным будет вариант 2;3.

Второй шаблон – (1;4);5;6;3*;7|8;9. Он состоит из шести элементов. Первый элемент относится к типу «перестановка компонентов». Элементы со второго по четвертый – «один компонент», четвертый элемент при этом помечен как рубежный. Пятый элемент – «один из компонентов». Шестой элемент – «один компонент». Для этого шаблона правильными вариантами решения будут четыре последовательности: 1;4;5;6;3;7;9, 1;4;5;6;3;8;9, 4;1;5;6;3;7;9, 4;1;5;6;3;8;9. Эти последовательности содержат компоненты 1, 4 в любом порядке, компоненты 5, 6, 3 в указанном порядке, один из компонентов 7 или 8 (но не оба сразу) и компонент 9.

Третий шаблон – (1;4);10;3*;7|8;11. Он состоит из пяти элементов. Первый элемент относится к типу «перестановка компонентов». Второй и третий элементы – «один компонент», при этом третий элемент помечен как рубежный. Четвертый элемент – «один из компонентов». Пятый элемент – «один компонент». Для этого шаблона правильными вариантами решения будут четыре последовательности: 1;4;10;3;7;11, 1;4;10;3;8;11, 4;1;10;3;7;11, 4;1;10;3;8;11. Эти последовательности содержат компоненты 1, 4 в любом порядке, компоненты 10 и 3 в указанном порядке, один из компонентов 7 или 8 (но не оба сразу) и компонент 11.

Для задания из примера 1 получим следующий массив массивов JSON-объектом:

[[{"type": 1, "components": [1], "flag": 2}, {"type": 1, "components": [2], "flag": 0}, {"type": 1, "components": [3], "flag": 1}], [{"type": 3, "components": [1, 4], "flag": 0}, {"type": 1, "components": [5], "flag": 1}, {"type": 1, "components": [6], "flag": 0}, {"type": 1, "components": [3], "flag": 1}, {"type": 2, "components": [7, 8], "flag": 0}, {"type": 1, "components": [9], "flag": 0}], [{"type": 3, "components": [1, 4], "flag": 0}, {"type": 1, "components": [10], "flag": 1}, {"type": 1, "components": [3], "flag": 1}, {"type": 2, "components": [7, 8], "flag": 0}, {"type": 1, "components": [11], "flag": 0}]].

3. Алгоритм контроля

Для автоматизированной оценки варианта решения авторами статьи предложен следующий алгоритм. Введем понятие эксперта. Экспертом в дальнейшем будем называть автомат, оценивающий соответствие варианта решения (входной последовательности) одному шаблону. Эксперт считывает по два значения из входной последовательности, если текущий элемент не является «перестановкой компонентов» или не помечен как рубежный. Для рубежного элемента эксперт считывает только одно значение. Для элемента «перестановка компонентов» эксперт просматривает все значения до следующего рубежного элемента. Эксперт оценивает соответствие считанной последовательности и текущего элемента, а затем выставляет оценку. Возможно 5 случаев:

1) считанная последовательность полностью соответствует элементу, т.е. компоненты, входящие в элемент, расположены в допустимом порядке, начиная с первого считанного значения;

2) считанная последовательность частично соответствует элементу, т.е. компоненты, входящие в элемент, расположены в допустимом порядке, начиная с элемента n считанной последовательности, тогда количество лишних компонентов увеличивается на $n - 1$;

3) считанная последовательность не соответствует элементу, тогда количество отсутствующих элементов увеличивается на 1;

4) если текущий элемент является рубежным и считанное значение не соответствует текущему элементу, то количество пропущенных рубежных элементов увеличивается на 1;

5) если текущий элемент помечен как допускающий отсутствие и считанная последовательность не соответствует текущему элементу, то количество пропущенных опциональных элементов увеличивается на 1.

В зависимости от результата сравнения эксперт перемещает метку текущего элемента на входной последовательности и независимо от результатов оценки переходит к следующему элементу. Эксперт выставляет представленному ответу оценку от 0 до 1.

Поскольку задание может быть описано более чем одним шаблоном, то один вариант решения может оцениваться более чем одним экспертом. Например, задание, описанное тремя шаблонами, оценивают три эксперта. Каждый из экспертов выставляет свою оценку, затем из них выбирается наибольшая. Альтернативой такому способу оценивания может быть алгоритм выбора эксперта. Алгоритм оценивает компетентность каждого эксперта для любого представленного варианта решения. Критерии оценки компетентности:

1. Насколько длина представленного варианта решения соответствует длине оцениваемой экспертом последовательности?
2. Насколько множество компонентов представленного ответа соответствует множеству компонентов оцениваемой экспертом последовательности?
3. Насколько множество компонентов оцениваемой экспертом последовательности покрывает множество компонентов, из которых может быть составлен ответ на задание?
4. Сколько рубежных для эксперта элементов присутствует в представленном варианте решения?

Эксперту, получившему наибольшую оценку, алгоритм передает оцениваемую последовательность. Оценка, выставленная выбранным экспертом, будет оценкой представленного обучающимся варианта решения.

Рассмотрим работу алгоритма оценивания на примерах. Будем использовать задание из примера, представленного выше. Пусть обучающимся предложен вариант решения “1;4;5;6;3;8;9”. Этот вариант решения является верным и соответствует шаблону 2, который оценивает эксперт 2. Результаты оценки экспертов приведены в табл. 1.

Таблица 1

Результаты оценки предложенного варианта решения

Эксперт	Оценка	Лишних компонентов	Отсутствует элементов	Отсутствует опциональных элементов	Отсутствует рубежных элементов
1	0	3	1	0	1
2	**1**	0	0	0	0
3	0	1	4	0	2

Алгоритм выбора эксперта подтверждает, что эксперт 2 имеет наибольшую оценку компетентности для предложенного варианта решения. Результаты оценки компетентности приведены в табл. 2.

Таблица 2

Выбор эксперта для оценки предложенного варианта решения

Эксперт	Оценка компетентности	Критерий 1	Критерий 2	Критерий 3	Критерий 4
1	3,106	2,333	0,273	0,667	1
2	**3,186**	1,167	0,273	0,875	1
3	2,408	1,4	0,636	0,571	0,5

Пусть обучающимся дан ответ “2;4;10;3;8;11”. Этот вариант решения является частично верным и частично соответствует шаблону 3, который оценивает эксперт 3. Результаты оценки экспертов приведены в табл. 3.

Таблица 3

Результаты оценки предложенного варианта решения

Эксперт	Оценка	Лишних компонентов	Отсутствует элементов	Отсутствует опциональных элементов	Отсутствует рубежных элементов
1	0	0	3	0	0
2	0,167	0	4	0	1
3	**0,75**	1	1	0	1

Алгоритм выбора эксперта подтверждает, что эксперт 3 имеет наибольшую оценку компетентности для предложенного варианта решения. Результаты оценки компетентности приведены в табл. 4.

Таблица 4

Выбор эксперта для оценки предложенного варианта решения

Эксперт	Оценка компетентности	Критерий 1	Критерий 2	Критерий 3	Критерий 4
1	2,939	2	0,273	0,667	1
2	2,102	1	0,273	0,375	0,5
3	**2,951**	1,2	0,636	0,714	1

Из табл. 1–4 можно сделать вывод, что алгоритм выбора эксперта достаточно эффективен.

Заключение

В результате решения задачи автоматизации контроля знаний в области основ программирования разработаны модель представления многовариантных заданий, язык описания шаблонов и алгоритм контроля знаний. Их применение к реальным задачам дало положительные результаты. Разработанный алгоритм решает проблему автоматизированной проверки программ методом белого ящика и позволяет осуществлять полноценную интеллектуальную автоматизированную проверку знаний в области основ программирования.

ЛИТЕРАТУРА

1. Гимранова Ф.Э. Экспериментальная апробация программного комплекса на основе интернет-сервисов для обучения программированию студентов учреждений среднего профессионального образования // Вестник Чувашского государственного педагогического университета им. И.Я. Яковлева. 2018. № 3 (99). С. 163–169.
2. Магомедов А.М., Лавренченко С.А., Ибрагимова З.И. Алгоритм автоматизации создания тестов // Вестник Дагестанского государственного университета. Сер. 1. Естественные науки. 2019. Т. 34, № 2. С. 63–71.
3. Полевщиков И.С., Баяндин К.Н. Разработка и применение средств автоматизации для обучения навыкам тестирования программного обеспечения (на примере методов «черного ящика») // Научно-технический вестник Поволжья. 2019. № 12. С. 88–91.
4. Кукарских К.А., Баранова К.С. Интеллект-карта в организации контроля знаний по программированию // Математическое и информационное моделирование : сб. науч. тр. / Тюмен. гос. ун-т, Ин-т математики и компьютерных наук. Тюмень : Изд-во Тюмен. гос. ун-та, 2019. С. 303–311.
5. Бейзер Б. Тестирование черного ящика технологии функционального тестирования программного обеспечения и систем // СПб. Питер. 2004: [пер. с англ.] 317 с.
6. Плеско М.С., Антипин А.Ф. Автоматизированная система поддержки процесса обучения программированию // Colloquium-journal. 2018. № 6-1. С. 48–54.
7. Антипин А.Ф. О разработке сетевой автоматизированной системы для контроля знаний по программированию // Современные наукоемкие технологии. 2016. № 10-1. С. 19–23.
8. Гладких И.Ю., Якушин А.В. Системы автоматизированного тестирования по программированию в образовательном пространстве // Современные проблемы науки и образования. 2016. № 3. С. 326.
9. Довгялло А.М., Ющенко Е.Л. Программированный учебник // Энциклопедия кибернетики. Киев : Гл. ред. УСЭ, 1975. Т. 2. С. 243–244.
10. Шевелев М.Ю., Вишнякова Л.А., Шевелев Ю.П. Контроль знаний в компьютерном обучении. Неантропоморфный подход. Saarbrücken: LAP LAMBERT Academic Publishing, 2014. 395 с.
11. Жуков И.А., Костюк Ю.Л. Модель представления заданий с многовариантными ответами для автоматизированного тестирования // Информационные технологии и математическое моделирование (ИТММ-2019) : материалы XVIII Междунар. конф. им. А.Ф.Терпугова, 26−30 июня 2019 г. Томск : Изд-во НТЛ, 2019. Ч. 1. С. 31–36.

Поступила в редакцию 28 мая 2020 г.

Zhukov I.A., Kostyuk Yu.L. (2020) MODEL OF REPRESENTATION OF MULTIVARIATE TASKS FOR AUTOMATED CONTROL OF PROGRAMMING KNOWLEDGE. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitelnaya tehnika i informatika*. [Tomsk State University Journal of Control and Computer Science]. 53. pp. 110–117

DOI: 10.17223/19988605/53/11

One way to automate the process of monitoring the practical skills of students in programming is to automatically test programs. Testing programs by the black box method does not allow assessing the student's level of proficiency in one or another basic algorithm. Evaluation by a teacher of the program code written by each student during mass learning is time-consuming. Therefore, the issue of

creating tools that carry out primary (preliminary) control of knowledge of basic algorithms is relevant. The authors of this article set the task of automating assessment of solving multivariate tasks, excluding semantic analysis of the program code manually by a teacher.

A model for the representation of multivariate tasks based on the constructive-selective method is developed. The student composes his answer from elements that are valid semantic units, which in the model are called components. The student offers his own variant of solution in the form of an ordered sequence composed of the proposed components. The solution may be right or wrong. All the correct solutions should be described by a finite set of patterns. Each pattern consists of elements. From a programming point of view, an element is an action that consist of one or more lines of program code. Three types of elements are distinguished in the model: one component, one of the components, permutation of components. The solution proposed by the student is evaluated in points. The maximum score is taken equal to one. The total score is made up of points for each element. The solution is considered partially correct if the final score falls within the interval (0, 1). A specialized formal language for describing patterns has been developed.

Algorithm for automated assessment of a variant of solution is proposed by the authors of the article. The automaton, called the expert, evaluates the correspondence of the solution variant (input sequence) to one pattern. The expert evaluates the correspondence of the read sequence and the current element, and then puts a mark. Since the task can be described by more than one pattern, one solution can be evaluated by more than one expert. For example, a task described by three patterns is evaluated by three experts. Each expert gives his own score, then the highest one is selected from them. An alternative to this method of assessment may be an expert selection algorithm. The algorithm evaluates the competence of each expert for any presented solution. To the expert who received the highest rating will be chosen to assess given variant of solution. Score given by the selected expert will be the score of the solution presented by the student.

As a result of solving the task of automating knowledge control in the field of programming basics, a model for representing multivariate tasks, a template description language, and a knowledge control algorithm are developed. Their application to real tasks yielded positive results. Developed algorithm solves the problem of automated assessment of programs using the white box method and allows full-fledged intelligent automated knowledge assessment in the field of programming basics.

Keywords: automated knowledge assessment; constructive-selective method; learning programming.

ZHUKOV Igor Andreevich (Post-graduate Student, Institute of Applied Mathematics and Computer Science, National Research Tomsk State University, Tomsk, Russian Federation).
E-mail: Ig.Zhukov963@yandex.ru

KOSTYUK Yuriy Leonidovich (Doctor of Technical Sciences, Professor, Institute of Applied Mathematics and Computer Science, National Research Tomsk State University, Tomsk, Russian Federation).
E-mail: kostyuk_y_l@sibmail.com

REFERENCES

1. Gimranova, F.E (2018) Experimental testing of software package based on internet services for teaching programming in the system of secondary vocational education. *Vestnik Chuvashskogo gosudarstvennogo pedagogicheskogo universiteta im. I.Ya. Yakovleva – I. Yakovlev Chuvash State Pedagogical University Bulletin*. 3(99). pp. 163–169.
2. Magomedov, A.M., Lavrenchenko, S.A. & Ibragimova, Z.I. (2019) Algorithm for automating test creation. *Vestnik Dagestanskogo gosudarstvennogo universiteta. Ser. 1. Estestvennye nauki – Herald of Dagestan State University. Series 1. Natural Sciences*. 34(2). pp. 63–71.
3. Polevshchikov, I.S. & Bayandin, K.N. (2019) Development and application of automation tools for teaching software testing skills (on the example of “black box” methods)**.** *Nauchno-tekhnicheskiy vestnik Povolzh'ya – Scientific and Technical Volga region Bulletin*. 12. pp. 88–91.
4. Kukarskikh, K.A. & Baranova, K.S. (2019) Intellekt-karta v organizatsii kontrolya znaniy po programmirovaniyu [Intellect map in the organization of knowledge control in programming]. In: Ivashko, A.G. (ed.) *Matematicheskoe i informatsionnoe modelirovanie* [Mathematical and information modeling]. Tyumen: Tyumen State University. pp. 303–311.
5. Beizer, B. (2004) *Testirovanie chernogo yashchika tekhnologii funktsional'nogo testirovaniya programmnogo obespecheniya i system* [Testing the black box of functional testing technology for software and systems]. St. Petersburg: Piter.
6. Plesko, M.S. & Antipin, A.F. (2018) Automated system for supporting the learning process of programming. *Colloquium-Journal*. 6-1. pp. 48–54.
7. Antipin, A.F. (2016) About the development of the network automated system for control of knowledge in programming. *Sovremennye Naukoemkie Tekhnologii – Modern High Technologies*. 10-1. pp. 19–23.
8. Gladkikh, I.Yu. & Yakushin, A.V. (2016) System of automated testing programs in learning environment. *Sovremennye problemy nauki i obrazovaniya – Modern Problems of Science and Education*. 3. pp. 326.
9. Dovgiallo, A.M. & Yushchenko, E.L. (1975) Programmirovannyy uchebnik [Programmed textbook]. *Entsiklopediya kibernetiki*. 2. pp. 243–244.
10. Shevelev, M.Yu., Vishnyakova, L.A. & Shevelev, Yu.P. (2014) *Kontrol' znaniy v komp'yuternom obuchenii. Neantropomorfnyy podkhod* [Knowledge control in computer learning. Non-anthropomorphic approach]. Saarbrücken: LAP LAMBERT Academic Publishing.
11. Zhukov, I.A. & Kostyuk, Yu.L. (2019) [Model for the presentation of tasks with multivariate answers for automated testing]. *Informatsionnye tekhnologii i matematicheskoe modelirovanie (ITMM-2019)* [Information technologies and mathematical modeling (ITMM-2019)]. Proc. of the 18th International Conference. Tomsk, June 26−30, 2019. Tomks: NTL. pp. 31–36.