ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ОБРАЗОВАТЕЛЬНОЙ ДЕЯТЕЛЬНОСТИ

DOI: 10.24143/2072-9502-2020-1-73-83
УДК 378.147:004

МЕТОДЫ И СРЕДСТВА ПРОДУКЦИОННОЙ ГЕНЕРАЦИИ ТЕСТОВ ПО КОМПЬЮТЕРНЫМ ДИСЦИПЛИНАМ

Н. А. Зацепин, Л. Н. Чернышов

Московский авиационный институт (национальный исследовательский университет), Москва, Российская Федерация

Проблема автоматической генерации тестов имеет большое значение в современной системе высшего образования, в особенности при дистанционном обучении. Рассмотрены различные разработки данного направления. К рассмотрению предложена разработка новых методов и средств генерации именно тестовых заданий. С целью снижения трудоемкости и временных затрат при формировании тестов по компьютерным дисциплинам разработана новая форма шаблона тестового задания, использующая заданные преподавателем продукционные правила. Они позволяют генерировать из группы таких шаблонов (шаблона теста) тесты, пригодные для импорта в систему дистанционного обучения Moodle. Разработка модуля для системы Moodle обеспечивает генерацию тестов из шаблонов непосредственно в момент проведения тестирования и позволяет сформировать шаблон теста вместо написания всех вариантов данного теста, что должно значительно облегчить труд и сэкономить время преподавателя. Предложены алгоритм генерации тестового задания и примеры готовых шаблонов по темам «Операционные системы» и «Python». Предлагаемый подход может быть использован в работе преподавателя как инструмент для подготовки тестирования в зависимости от его цели и характера изучаемой дисциплины.

Ключевые слова: автоматизация, тест, шаблон, тестовое задание, генерация варианта задания, контекстно-свободная грамматика, параметр, алгоритм, скрипт обратной связи.

Для цитирования: *Зацепин Н. А., Чернышов Л. Н.* Методы и средства продукционной генерации тестов по компьютерным дисциплинам // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. 2020. № 1. С. 73–83. DOI: 10.24143/2072-9502-2020-1-73-83.

Введение

В современной системе получения высшего образования тестирование студентов играет важную роль. Составление тестов и их вариантов стало отнимать много времени у преподавателей, т. к. написание тестов представляет собой трудоемкий творческий процесс, который в данное время автоматизирован весьма слабо. А ведь автоматизация этого процесса хотя бы частично позволила бы преподавателю уделять больше времени другим важным обязанностям. Поэтому работа в данном направлении очень актуальна.

К сожалению, в доступных источниках проблема освещена недостаточно широко. Однако работы в данном направлении есть. Авторы статьи [1] использовали параметризуемые тесты в собственной системе QuizPACK. По результатам исследования влияния работы студентов с данной системой на оценку, полученную на экзамене, установлено, что успешность работы студентов с QuizPACK коррелирует с полученной оценкой.

В работах [2–4] были представлены два интересных метода. В первом методе преподаватель задавал контекстно-свободную грамматику (КС-грамматику) и с ее помощью генерировались вопросы и задачи. Во втором методе преподаватель использовал реализованную систему, позволяющую из текстов учебных пособий с помощью метода, основанного на применении лингвистических процессов, сгенерировать тестовые задания. Они подвергались отбору и редактированию со стороны преподавателя, затем экспортировались в систему дистанционного обучения Moodle.

Авторы статьи [5] предложили использовать КС-грамматику, заданную в XML-формате, для генерации на ее основе задач по программированию. Следует отметить, что предлагаемый формат шаблона позволял выбирать условия генерации значений указанных параметров. Данная идея получила развитие в работе [6], где был предложен язык шаблонов, разработанный на базе xml, – taml. В нем пользователь задавал метаинформацию о шаблоне, тело задания с параметрами и параметры. Следует отметить, что параметры в taml могут зависеть от других параметров и имеют тип.

В работах [7, 8] авторы используют собственный вид шаблонов тестовых заданий. При написании шаблона каждому параметру задаются условия его генерации (программа) и пишется программа для получения решения данного задания в зависимости от сгенерированных параметров, и если решение существует, то формируется задача. Таким образом, количество тестов существенно увеличивается, в результате чего повышается объективность оценивания навыков студентов.

Автор работы [9] предложил генерацию вариантов заданий с помощью морфологического синтеза. Описанный шаблон содержит параметры трех видов: независимые, зависимые и вложенные. Система генерирует независимые и вложенные параметры, и из них выбираются значения зависимых параметров. Описанная обучающая система обеспечивает генерацию вариантов заданий по шаблонам «на лету», что позволяет ей не терять актуальность со временем.

В работе [10] реализован генератор вариантов заданий по дисциплине «Дискретная математика», в котором применяются специфические методы для генерации формул.

Следует отметить, что описанные выше методы и средства генерации заданий требуют изучения собственных синтаксисов шаблонов, дополнительного редактирования и отбора сгенерированных тестовых заданий, а также не всегда предусматривают генерацию именно тестовых заданий, отдавая предпочтение обычным заданиям или вопросам. Поэтому актуальна разработка новых методов и средств генерации именно тестовых заданий.

Тестовые задания по дисциплинам программирования имеют особенности, не характерные для других дисциплин. Например, тестовое задание может включать фрагмент кода на некотором языке программирования, а в ответе необходимо указать результат выполнения этого кода. При составлении такого задания для полной надежности желательно действительно выполнить этот фрагмент и получить результат. Но если подобные фрагменты параметризовать так, что число возможных вариантов задания станет очень большим, их проверка станет очень трудоемкой. Возможное решение этой проблемы – исполнение сгенерированного фрагмента непосредственно после его создания в генераторе заданий. Именно это решение предлагается в описываемом нами методе.

Тестовое задание, предназначенное для генерации, предоставляется в виде специального шаблона, в котором присутствует текст на языке JavaScript. Этот текст, названный *скриптом обратной связи*, может содержать программу, аналогичную программе (фрагменту программы) на языке, который предлагается в формулировке задания. Исполнение программы обеспечивается встроенной функцией eval, которую выполняет интерпретатор JavaScript.

Описание метода продукционной генерации тестов

Текст тестового задания состоит из двух частей: формулировки задания и возможных ответов на это задание. Генерация задания производится из некоторого шаблона, который также состоит из двух частей: из первой части генерируется формулировка задания, из второй – ответы на задание.

Предлагаемый метод в первом варианте ограничивается тестовыми заданиями открытого типа и закрытого типа с множественным выбором, т. е. либо ответ дается в виде единственного слова или нескольких слов, либо указанием одного или нескольких из предложенных вариантов ответов.

Структура шаблона тестового задания описывается JSON-файлом, представленным записью с пятью парами «ключ-значение»:

{
"title": "тема тестового задания",
"type": T,
"rules": {
"p_1": ["альтернатива$_{1,1}$", … , "альтернатива$_{1,k1}$"],
"p_2": ["альтернатива$_{2,1}$", … , "альтернатива$_{2,k2}$"],
…

"p_n": ["альтернатива$_{n,1}$", … , "альтернатива$_{n,kn}$"]
},
"testText": "текст задания",
"feedbackScript": "скрипт задания"
}

За ключом title следует заголовок задания. Значение type соответствует типу задания: 0 – открытый ответ, 1 – закрытый ответ с одним правильным вариантом, 2 – закрытый ответ с несколькими правильными вариантами. Пара с ключом rules задает правила КС-грамматики. Каждое правило определяется парой с названием нетерминального символа и массивом альтернатив. Альтернатива – это произвольный текст, в который могут включаться названия нетерминалов в виде ${$p$_i$} и вызовы функций (табл. 1).

Таблица 1

Синтаксис	Описание
rInteger (min, max)	Генерирует случайное целое число в указанном диапазоне
rFloat (min, max, length)	Генерирует случайное вещественное число в указанном диапазоне с заданным количеством чисел после точки
rElement (...)	Берет случайный элемент из списка аргументов

Для вызова функций используется символ «$». Чтобы использовать терминальный символ «$», необходимо написать «$$». Набор функций в перспективе будет увеличен.

Текст задания, которое фактически является правой частью правила для начального нетерминала, также может содержать вхождения нетерминалов и вызовы функций. Для алгоритма генерации заданий должны соблюдаться следующие ограничения: 1) альтернативы 1-го правила не содержат вхождений нетерминалов; 2) альтернативы *i*-го правила могут содержать вхождения нетерминалов только с меньшими индексами. В нотации РБНФ правила грамматики можно записать так (p_0 – начальный символ грамматики):

$p_0 =$ текст задания.
$p_1 =$ альтернатива$_{1,1}$| … |альтернатива$_{1,k1}$.
$p_2 =$ альтернатива$_{2,1}$| … |альтернатива$_{2,k2}$.
…
$p_n =$ альтернатива$_{n,1}$| … |альтернатива$_{n,kn}$.

Для формирования шаблона тестового задания помимо описанных выше ключей требуется написание скрипта на языке JavaScript (скрипта обратной связи). Исполнение этого скрипта позволяет сформировать правильный ответ на поставленный в задании вопрос в зависимости от сгенерированного текста задания. Для этого в скрипте есть возможность, используя встроенные переменные и функции (табл. 2 и 3), получить информацию о том, какие конкретно правые части были выбраны и каковы их значения (конечные, после всех подстановок).

Таблица 2

Имя константы	Значение
$ «левая часть»_«индекс правой части»	Натуральное число, соответствующее номеру правой части указанной левой части. Правые части нумеруются, начиная с нуля
$ «левая часть»_value	Строка, соответствующая сгенерированной правой части указанной левой части
$ «левая часть»	Натуральное число, соответствующее номеру выбранной транслятором правой части указанной левой части

Таблица 3

Синтаксис	Описание
$$rSubArray(array, length)	Генерирует подмассив длины *length*, состоящий из случайных элементов исходного массива *array*
$$AnswerIs(array) $$AnswerIs(string)	Задает множество верных вариантов ответа
$$FalseOptionsIs(array)	Задает неправильные варианты ответов

Следует отметить, что пользователь пишет только часть этого скрипта, из которой добавлением в начало программы описанных выше переменных и функций генерируется исполняемый скрипт обратной связи. Результатом его выполнения является информация для проверки данного задания тестирующей системой, например указание верных и неверных вариантов ответов. Данный скрипт выполняется при генерировании тестового задания из его шаблона.

Для простых тестовых заданий необходимость написания данного скрипта может показаться избыточной. Однако для них скрипты обратной связи будут типовыми и семантически слабо различаться. Набор встроенных функций со временем будет расширяться, что в перспективе дальнейшей разработки позволит максимально упростить написание вышеописанных скриптов.

Приведем алгоритм генерации тестового задания.

1. Выбрать случайным образом одну из альтернатив 1-го правила, присвоить ее значение переменной p_1.

2. $i := 2$.

3. Пока $i <= n$:

3.1. Выбрать случайным образом одну из альтернатив i-го правила, присвоить ее значение переменной p_i.

3.2. Произвести замену в значении p_i всех вхождений $\{p_j\}$ для всех $j < i$.

3.3. Заменить все входящие в значении p_i вызовы специальных функций на их вычисленные значения.

3.4. $i := i + 1$.

4. Конец цикла.

5. Произвести замену в тексте задания всех вхождений $\{p_j\}$ для всех j.

6. Заменить в тексте задания все входящие вызовы специальных функций на их вычисленные значения.

7. Сгенерировать и выполнить скрипт обратной связи.

Примеры шаблонов

1. Тестовое задание содержит описание команды UNIX, а тестируемый должен будет вводить имя команды, которая соответствует этому описанию. Шаблон тестового задания содержит только одно правило с тремя альтернативами, и по нему будут генерироваться три разных задания. В скрипте содержится список из трех слов, соответствующих тексту задания. Функция $$AnswerIs позволяет сгенерировать правильный ответ в зависимости от выбранной альтернативы правила. Текст шаблона в JSON-формате:

```
{
"title": "Команды Shell ОС UNIX",
"type": 0,
"rules": {
"p": ["получает справки о команде",
"выполняет переход из каталога в каталог",
"определяет имя текущего каталога"
]
},
"testText": "Команда ОС Unix, которая ${p}:",
"feedbackScript": "let answer = ['man','cd','pwd']; $$AnswerIs(answer[$p]);"
}
```

2. Во втором примере по шаблону генерируются три задания, в которых будет предъявлено три варианта ответа. Функция $$AnswerIs определяет правильный вариант ответа, а функция $$FalseOptionsIs – два неправильных варианта. Текст шаблона в JSON-формате:

```
{
"title": "Типы ядер ОС",
"type": 1,
"rules": {
"p": [«Предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием.»,
```

«Предоставляет только функции для взаимодействия между процессами безопасного выделения и освобождения ресурсов.»,

«Ядро выполняет лишь одну задачу – обработку аппаратных прерываний, генерируемых устройствами компьютера."]

},

"testText": "${p}\nЭто описание ядра операционной системы:",

"feedbackScript": "let options = ['Микроядро', 'Экзоядро', Наноядро'];

$$AnswerIs([options[$p]]);

options.splice($p, 1);

$$FalseOptionsIs($$rSubArray(options,3));"

}

Скрипт обратной связи, представленный значением ключа feedbackScript, – лишь небольшая часть того кода, который будет выполнен. Скрипт для выполнения формируется исходя из выбранных альтернатив. Для приведенного примера полный скрипт будет таким (полагается, что была выбрана первая альтернатива):

```
let $p = 0;
let $p_value = 'Предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием.';
let $p_0 = 0;
let $p_1 = 1;
let $p_2 = 2;
let __FALSE_OPTIONS = [];
let __ANSWER = –1;
function $$rSubArray(array, length) {
  if (isNaN(+length)) {
    throw Error('$$rSubArray: length is not a number');
  }
  if (length < 0) {
    throw Error('$$rSubArray: length is bad');
  }
  function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min)) + min;
  }
  let result = [];
  let copy = array.slice(0);
  for (let _ = 0; _ < length; ++_) {
    if (copy.length === 0) {
      break;
    }
    let i = getRandomInt(0, copy.length);
    result.push(copy[i]);
    copy.splice(i, 1);
  }
  return result;
}
function $$FalseOptionsIs(falseOptions) {
__FALSE_OPTIONS = falseOptions;
}
function $$AnswerIs(answer) {
  if (typeof answer !== 'object') {
    __ANSWER = [answer];
  } else {
    __ANSWER = answer;
  }
```

```
}
function _END_SCRIPT() {
  return [__ANSWER, __FALSE_OPTIONS];
}
let options = ['Микроядро', Экзоядро', 'Наноядро'];
$$AnswerIs([options[$p]]);
options.splice($p, 1);
$$FalseOptionsIs($$rSubArray(options,3));
__END_SCRIPT();
```

Результатом выполняемого скрипта обратной связи является(ются) ответ(ы) и неверные варианты выбора, последние могут отсутствовать. После того как будет получен этот результат, он вместе со сгенерированным текстом тестового задания транслируется в тестовое задание формата GIFT (формат системы Moodle).

Для приведенных выше примеров наличие скрипта обратной связи может показаться избыточным. Но для более сложных заданий, где текст задания содержит фрагменты программ, скрипт просто необходим. Для преподавателя, составляющего задания, необходимы знания JavaScript, но это не является проблемой, т. к., во-первых, JavaScript является языком с низким порогом вхождения, во-вторых, программы для тестовых заданий, как правило, не слишком сложные.

Сгенерированные тестовые задания должны объединяться в наборы, которые будут представлены каждому студенту. При этом должен соблюдаться принцип равной сложности набора заданий. Этого можно достигнуть, если по одному и тому же шаблону тестового задания будет сгенерировано столько заданий, сколько в группе студентов, которые будут тестироваться.

Описанная техника генерации может использоваться для обычных заданий, выполнение которых требует длительного времени. Это могут быть, например, задания на написание программы на некотором языке программирования. Такие задания обычно подразумевают ручную проверку преподавателем. Автоматизация проверки правильности также возможна, что практикуется при проведении олимпиад по программированию, однако задания, предлагаемые в процессе обучения, обычно более простые, и для автоматизации проверки можно применять другой подход, а именно проверять правильность получения результата работы программы на некотором тестовом примере, заранее сгенерированном и предоставленном студенту.

Тестовые наборы данных (одномерные и многомерные массивы, списки, тексты и т. п.), обладающие определенными свойствами, можно создавать с помощью специально написанных программ. Эти наборы будут подходить для некоторого класса алгоритмов. Например, поиск позиции наибольшего (наименьшего) из положительных (отрицательных) значений в заданном столбце (строке) числовой матрицы. Такой шаблон дает 8 различных формулировок. Генератор тестовой матрицы может создавать данные, соответствующие конкретной постановке задачи. Также можно использовать заранее написанную универсальную программу, которая определит результат по заданным параметрам для любого случайно сгенерированного набора.

Разработка инструментов для работы с шаблонами

Реализация предложенного метода осуществлена в двух вариантах: веб-сервиса и не связанного с ним настольного приложения.

Веб-сервис обладает следующими функциональными возможностями:

- регистрация и авторизация пользователя;
- просмотр документации;
- хранение, загрузка на сервер, загрузка с сервера, создание, редактирование и удаление шаблонов теста и тестовых заданий с помощью редактора;
- загрузка архива со сгенерированными вариантами тестов в формате GIFT из выбранного шаблона теста.

Создаются и редактируются шаблоны тестовых заданий в редакторе. В нем шаблон представляется как форма, которую необходимо заполнить. Для задания описанной выше КС-грамматики в редакторе используется следующий синтаксис:

параметр = [*альтернатива_0* | ... | *альтернатива_n*],

где *параметр* – это название нетерминального символа; *альтернатива_i* – *i*-я альтернатива, конструкция «|» – выбор; конструкция «[]» – условное вхождение (данная конструкция отсутствует, если параметр не подразумевает наличие пустой альтернативы. Пустая альтернатива при наличии нумеруется *n* + 1-м номером); символ «.» – конец описания параметра. Замечание: для написания символа «.» как части текста альтернативы необходимо написать «..».

В редакторе шаблона необходимо написать заголовок (метаинформация о его семантике), выбрать тип тестового задания, составить грамматику по заданному выше синтаксису, задать текст тестового задания и написать скрипт обратной связи. Для удобства синтаксис JavaScript в поле скрипта обратной связи подсвечивается с помощью модуля CodeMirror. В перспективе будет добавлена подсветка синтаксиса в полях грамматики и текста тестового задания. Следует отметить, что редактор с каждым изменением шаблона проверяет его корректность, и пользователь информируется, если данное состояние шаблона некорректное.

Рассмотрим на простых примерах процесс создания шаблона тестового задания и теста в редакторе. Пусть необходимо разработать шаблон теста по тематике «Операционные системы». Разберем написание шаблона одного из тестовых заданий. Для тестируемого задание будет сформулировано в виде повествовательного предложения. Его ответ заключается в написании одного слова. Тестовое задание будет проверяться полным соответствием со словом-ответом. На рис. 1 мы видим форму, в которой уже задан шаблон данного тестового задания.

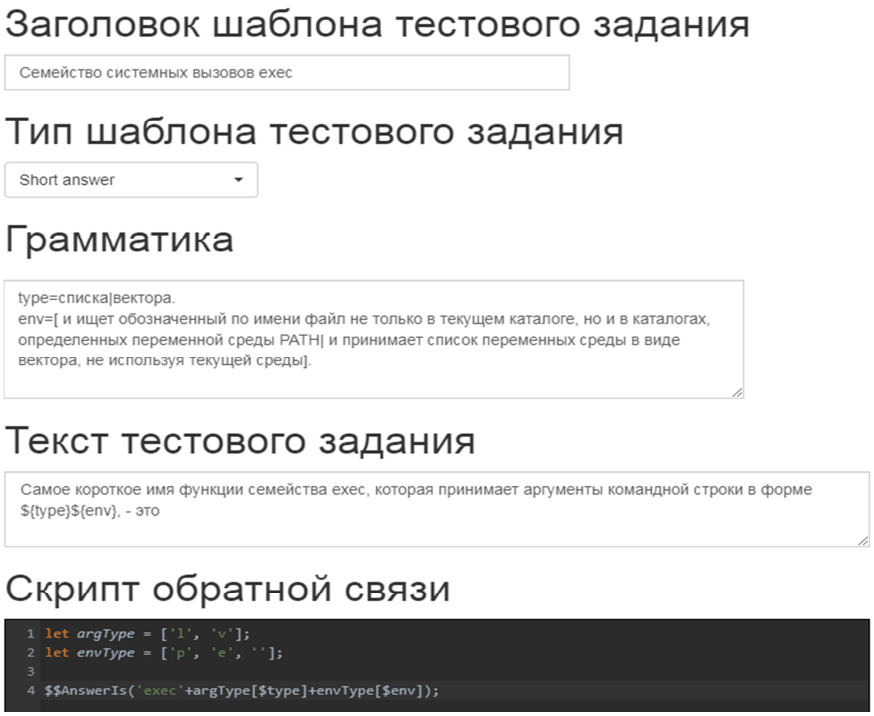

Рис. 1. Пример шаблона тестового задания по теме «Операционные системы»

Скрипт обратной связи заданного шаблона тривиален. С использованием выбранных индексов альтернатив, которые заданы во встроенных переменных $type и $env, для нетерминалов type и env задается продолжение строки «exec», в результате чего формируется правильный ответ. Результирующая строка задается как ответ на задание с помощью встроенной функции $$AnswerIs.

В качестве примера использования функций в альтернативах для нетерминалов рассмотрим шаблон тестового задания по теме «Python» на рис. 2.

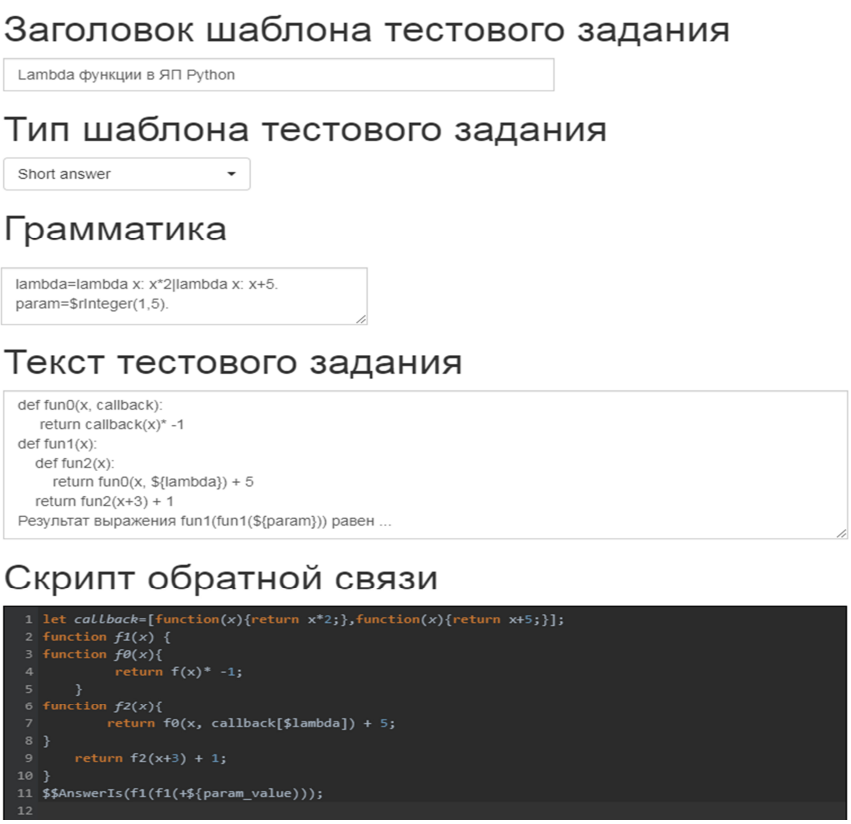

Рис. 2. Пример шаблона тестового задания по теме «Python»

Для тестируемого задание будет сформулировано так же в виде повествовательного предложения, и его ответ тоже заключается в написании одного слова.

Скрипт обратной связи по структуре соответствует программе на Python, но написан на JavaScript. Здесь была использована функция rInteger из табл. 1. Левая часть «param» используется в качестве случайного аргумента функции «fun1» в тексте тестового задания (рис. 3).

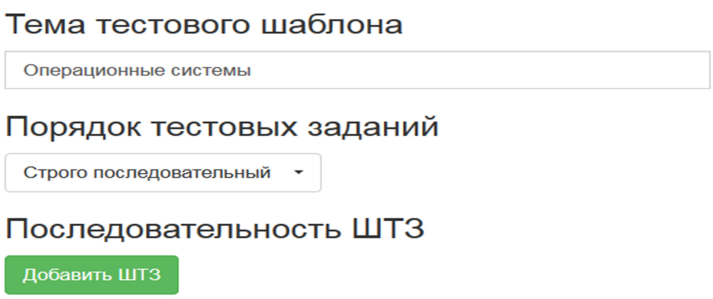

Рис. 3. Создание шаблона теста

При написании скрипта обратной связи для получения значения параметра была использована встроенная константа «$param_value».

Для создания и редактирования шаблонов теста тоже используется редактор. Шаблон теста – это совокупность некоторых шаблонов тестовых заданий и метаинформации о тематике теста. Чтобы создать шаблон теста, необходимо указать его тему и выбрать загруженные ранее шаблоны тестовых заданий (рис. 3). После создания шаблона теста появится возможность загрузить архив со сгенерированными вариантами тестов в формате GIFT. Этот формат пригоден для импорта в систему Moodle.

Тема шаблона теста задается в поле «Тема тестового шаблона». Опция «Порядок тестовых заданий» является избыточной, т. к. в системе Moodle есть возможность их перемешивать, но было решено ее оставить, т. к. GIFT поддерживается не только Moodle и, возможно, в каких-то тестирующих системах нет такой функции. Добавление в шаблон теста шаблонов тестовых заданий происходит с помощью нажатия на кнопку «Добавить ШТЗ». Из списка шаблонов тестовых заданий, имеющихся в базе данных текущего пользователя, нужно составить шаблон теста.

Шаблоны тестов хранятся в формате JSON. Сервер использует СУБД MySQL. Шаблоны хранятся в базе данных в сжатом бинарном представлении. При их загрузке пользователь получает JSON-файлы. Для использования уже готового шаблона надо его загрузить на сервер. Веб-сервис написан на Node.JS с использованием сторонних модулей. В качестве клиента было написано одностраничное веб-приложение с использованием jQuery, Bootstrap, CodeMirror и некоторых других библиотек.

Настольное приложение не связано с веб-сервисом. Оно имеет схожие функциональные возможности:

- просмотр документации;
- создание и редактирование шаблонов тестов и тестовых заданий;
- генерация вариантов тестов в формате GIFT из выбранного шаблона теста.

Настольное приложение написано на Node.JS с использованием фреймворка Electron. Веб-сервис и настольное приложение используют аналогичный редактор шаблонов.

Заключение

Предложен метод генерации вычисляемых тестов, подходящий преимущественно для компьютерных дисциплин.

В результате была разработана его реализация в виде веб-сервиса и настольного приложения. Тесты генерируются в формате GIFT и, следовательно, могут быть импортированы в систему дистанционного обучения Moodle. Были разработаны шаблоны тестов по темам «Операционные системы» и «Python».

В перспективе возможна разработка модуля для системы Moodle. Он поможет решить проблему потери актуальности базы тестов. Имеется в виду, что загруженные тесты для новых потоков студентов приходится вручную обновлять каждый учебный год. Модуль сможет обеспечить генерацию тестов из шаблонов тестов непосредственно в момент проведения тестирования. Таким образом, преподаватель будет загружать в систему не тесты в формате GIFT, обновляя их каждый год, а предложенные шаблоны тестов. Это уменьшит объем работы, необходимый для проведения тестирований.

СПИСОК ЛИТЕРАТУРЫ

1. *Sosnovsky S., Shcherbinina O., Brusilovsky P.* Web-based Parameterized Question as a Tool for Learning // Allison Rossett (ed.): Proceedings of E-Learn 2003, Phoenix, Arizona USA, November 7–11. 2003. P. 2151–2154.

2. *Сергушичева А. П., Швецов А. Н.* Синтез интеллектуальных тестов средствами формальной продукционной системы // Математика, компьютер, образование. 2003. Вып. 10. Ч. 1. С. 310–320.

3. *Сергушичева А. П., Швецов А. Н.* Гибридный подход к синтезу тестовых заданий в тестирующих системах // Математика, компьютер, образование. 2006. Вып. 13. Т. 1. С. 215–228.

4. *Крутасов А. М., Швецов А. Н.* Метод автоматизированной генерации заданий для тестов контроля знаний из текстов учебных пособий // Современные информационные технологии и ИТ-образование. 2013. № 9. С. 218–228.

5. *Лаптев В. В., Толасова В. В.* Генерация вариантов заданий для лабораторных работ по программированию // Вестн. Астрахан. гос. техн. ун-та. Сер.: Управление, вычислительная техника и информатика. 2010. № 1. С. 127–131.

6. *Лаптев В. В.* Морфологический синтез вариантов заданий в обучающей системе по программированию // Вестн. Астрахан. гос. техн. ун-та. Сер.: Управление, вычислительная техника и информатика. 2015. № 1. С. 140–152.

7. *Кручинин В. В., Морозова Ю. В.* Модели и алгоритмы генерации задач в компьютерном тестировании // Изв. Том. политехн. ун-та. Сер.: Технические науки. 2004. Т. 307. № 5. С. 127–131.

8. *Кручинин В. В., Магазинников Л. И., Морозова Ю. В.* Модели и алгоритмы компьютерных самостоятельных работ на основе генерации тестовых заданий // Изв. Том. политехн. ун-та. Сер.: Технологии инженерного образования. 2006. Т. 309. № 8. С. 258–262.

9. *Лаптев В. В.* Генерация заданий в обучающей среде для программистов // Материалы XXI Междунар. конф. по вычислит. механике и соврем. приклад. програм. системам (ВМСППС'2019) (Алушта, 24–31 мая 2019 г.). М.: Изд-во МАИ, 2019. С. 772–773.

10. *Алексеев Н. С.* Автоматизированное рабочее место преподавателя на основе технологий Google // Информационные и телекоммуникационные технологии. 2019. № 42. С. 32–37.

Статья поступила в редакцию 21.11.2019

ИНФОРМАЦИЯ ОБ АВТОРАХ

Зацепин Никита Андреевич – Россия, 125993, Москва; Московский авиационный институт (национальный исследовательский университет); магистрант кафедры вычислительной математики и программирования; new_mail_conf@protonmail.com.

Чернышов Лев Николаевич – Россия, 125993, Москва; Московский авиационный институт (национальный исследовательский университет); канд. физ.-мат. наук, доцент; доцент кафедры вычислительной математики и программирования; levchern@gmail.com.

METHODS AND MEANS OF PRODUCTION GENERATION OF TESTS ON COMPUTER DISCIPLINES

N. A. Zatsepin, L. N. Chernyshov

Moscow Aviation Institute (National Research University),
Moscow, Russian Federation

Abstract. The article describes the problem of automatic test generation, which is important in the modern education system, especially in distance learning. Various developments in this area have been considered. The development of new methods and means of generating precisely test tasks has been submitted. In order to reduce the complexity and time in the course of developing tests on computer disciplines, a new type of the test task template has been produces using the production rules specified by the teacher. They make it possible to generate tests from a group of such templates (test template) suitable to be imported into the Moodle distance learning system. The development of a module for the Moodle system provides generating tests from templates directly at the time of testing and allows to create a test template instead of writing all the options for this test in hand, which should greatly facilitate the work and save the teacher's time. The algorithm for generating a test task and examples of ready-made templates on the topics "Operating Systems" and "Python" have been presented. The proposed approach can be used in the teacher' work as a tool for test training depending on its purpose and the nature of the discipline studied.

Key words: automation, test, template, test task, generation of task variant, context-free grammar, parameter, algorithm, feedback script.

For citation: Zatsepin N. A., Chernyshov L. N. Methods and means of production generation of tests on computer disciplines. *Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics*. 2020;1:73-83. (In Russ.) DOI: 10.24143/2072-9502-2020-1-73-83.

REFERENCES

1. Sosnovsky S., Shcherbinina O., Brusilovsky P. Web-based Parameterized Question as a Tool for Learning. *Allison Rossett (ed.): Proceedings of E-Learn 2003.* Phoenix, Arizona USA, November 7–11. 2003. Pp. 2151-2154.

2. Sergushicheva A. P., Shvetsov A. N. Sintez intellektual'nykh testov sredstvami formal'noi produktsionnoi sistemy [Synthesis of intelligent tests by means of formal production system]. *Matematika, komp'iuter, obrazovanie*, 2003, iss. 10, part 1, pp. 310-320.

3. Sergushicheva A. P., Shvetsov A. N. Gibridnyi podkhod k sintezu testovykh zadanii v testiruiushchikh sistemakh [Hybrid approach to synthesis of test tasks in testing systems]. *Matematika, komp'iuter, obrazovanie*, 2006, iss. 13, part 1, pp. 215-228.

4. Krutasov A. M., Shvetsov A. N. Metod avtomatizirovannoi generatsii zadanii dlia testov kontrolia znanii iz tekstov uchebnykh posobii [Method of automated task generation for knowledge control tests from textbooks]. *Sovremennye informatsionnye tekhnologii i IT-obrazovanie*, 2013, no. 9, pp. 218-228.

5. Laptev V. V., Tolasova V. V. Generatsiia variantov zadanii dlia laboratornykh rabot po programmirovaniiu [Generation of task variants for programming labs]. *Vestnik Astrakhanskogo gosudarstvennogo tekhnicheskogo universitetata. Seriia: Upravlenie, vychislitel'naia tekhnika i informatika*, 2010, no. 1, pp. 127-131.

6. Laptev V. V. Morfologicheskii sintez variantov zadanii v obuchaiushchei sisteme po programmirovaniiu [Morphological synthesis of task options in programming training system]. *Vestnik Astrakhanskogo gosudarstvennogo tekhnicheskogo universitetata. Seriia: Upravlenie, vychislitel'naia tekhnika i informatika*, 2015, no. 1, pp. 140-152.

7. Kruchinin V. V., Morozova Iu. V. Modeli i algoritmy generatsii zadach v komp'iuternom testirovanii [Models and algorithms for problems generating in computer testing]. *Izvestiia Tomskogo politekhnicheskogo universiteta. Seriia: Tekhnicheskie nauki*, 2004, vol. 307, no. 5, pp. 127-131.

8. Kruchinin V. V., Magazinnikov L. I., Morozova Iu. V. Modeli i algoritmy komp'iuternykh samostoiatel'nykh rabot na osnove generatsii testovykh zadanii [Models and algorithms of self-study computer works using test task generation]. *Izvestiia Tomskogo politekhnicheskogo universiteta. Seriia: Tekhnologii inzhenernogo obrazovaniia*, 2006, vol. 309, no. 8, pp. 258-262.

9. Laptev V. V. Generatsiia zadanii v obuchaiushchei srede dlia programmistov [Generation of tasks in learning environment for programmers]. *Materialy XXI Mezhdunarodnoi konferentsii po vychislitel'noi mekhanike i sovremennym prikladnym programmnym sistemam (VMSPPS'2019) (Alushta, 24–31 maia 2019 g.).* Moscow, Izd-vo MAI, 2019. Pp. 772-773.

10. Alekseev N. S. Avtomatizirovannoe rabochee mesto prepodavatelia na osnove tekhnologii Google [Automated Teacher's Workstation based on Google technologies]. *Informatsionnye i telekommunikatsionnye tekhnologii*, 2019, no. 42, pp. 32-37.

The article submitted to the editors 21.11.2019

INFORMATION ABOUT THE AUTHORS

Zatsepin Nikita Andreevich – Russia, 125993, Moscow; Moscow Aviation Institute (National Research University); Master´s Course Student of the Department of Computational Mathematics and Programming; new_mail_conf@protonmail.com.

Chernyshov Lev Nikolaevich – Russia, 125993, Moscow; Moscow Aviation Institute (National Research University); Candidate of Physics and Mathematics Sciences, Assistant Professor; Assistant Professor of the Department of Computational Mathematics and Programming; levchern@gmail.com.