УДК 004.588

Симуни М.Л.[1,3], Соловьев А.Ю.[2], Шайтан В.И.[2]

[1] Санкт-Петербургский государственный университет, г. Санкт-Петербург, Россия
[2] JobToday SA, г. Санкт-Петербург, Россия
[3] Санкт-Петербургский политехнический университет Петра Великого, г. Санкт-Петербург, Россия

АВТОМАТИЗИРОВАННАЯ ПРОВЕРКА ЗАДАЧ В КУРСЕ «ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ»

АННОТАЦИЯ

В статье рассматривается система тестирования и проверки задач для курса «Функциональное программирование», читаемого на математико-механическом факультете СПбГУ. Обсуждаются основные возможности системы тестирования, средства автоматизированной проверки задач и основные возможности проектируемой версии системы.

КЛЮЧЕВЫЕ СЛОВА

Тестирование; автоматизированная проверка программ; функциональное программирование.

Simuni M.L.[1,3], Soloviov A.Y.[2], Shaitan V.I.[2]

[1] St.Petersburg State University, St.Petersburg, Russia
[2] JobToday, St.Petersburg, Russia
[3] Peter the Great St.Petersburg Polytechnic University, St.Petersburg, Russia

AUTOMATED PROBLEM CHECKING FOR FUNCTIONAL PROGRAMMING COURSE

ABSTRACT

In the article the system for problem testing and checking is considered. The most important features of testing system, tools for automated problem checking and the features for the next system version are considered.

KEYWORDS

Testing; problem checking; functional programming.

В работе рассматриваются возможности автоматизации проверки задач для курса «Функциональное программирование» [1], читающемся на математико-механическом факультете СПбГУ. Последние четыре года при проведении курса используется система тестирования и учета решения задач [2]. В результате внедрения этой системы заметно увеличилась активность студентов при решении задач, и, в то же время, заметно упростилась работа преподавателя. В 2016 году система была дополнена средствами автоматизированной проверки задач. Это позволило преподавателю избавиться от части рутинной работы и сосредоточиться на содержательной работе со студентами. В то же время, первые результаты работы с новой версией системы показали необходимость ее усовершенствования. В настоящее время разрабатывается проект новой версии системы, позволяющей сделать работу преподавателя более комфортной.

Перечислим кратко особенности рассматриваемого курса. Курс «Функциональное программирование» проводится на математико-механическом факультете СПбГУ для студентов 4 или 3 курс, в зависимости от специальности. Курс читается на основе языка Haskell [3]. Для участников не предполагаются какие-либо знания в области функционального программирования, но предполагается достаточно хорошие навыки программирования вообще и общая математическая культура. Рассматриваются темы от основ функционального программирования (функции высшего порядка, алгебраические типы данных и т.д.) до достаточно сложных тем, таких, как продолжения (continuations), монады, «бесплатные теоремы» (theorems for free) и т.д.

Курс слушают примерно 70-80 человек. Подготовка и заинтересованность студентов варьируется, но большинство студентов, около 60%, достаточно хорошо подготовлены и активно участвуют в работе, решая большое количество задач. Остальные примерно пополам делятся на очень хорошо подготовленных студентов, для которых приходится придумывать задачи

повышенной сложности, и на студентов, которые не проявляют интерес к материалу. Для последних приходится формулировать четкий минимум знаний и умений, и, в том числе, задач, которые они должны уметь решать для получения зачета.

Методической основой курса является большое количество задач. Каждую неделю дается примерно 8-10 задач. Их можно разбить на следующие группы:

Задачи для закрепления материала. После каждого занятия на дом даются 4-5 задач, обычно достаточно простых. Эти задачи используются, для того, чтобы и преподаватель, и сами студенты могли проверить, как они освоили текущий материал. Иногда они используются, чтобы подготовить студентов к восприятию очередного занятия. Эти задачи почти всегда разбираются на следующем занятии

Дополнительные задачи. Каждую неделю дается 2-3 задачи посложнее, они предназначены для хорошо подготовленных студентов. Их решение – это достаточно большая и сложная программа и они, как правило, на занятиях не разбираются.

Дополнительные задачи 'на обычном языке'. Каждую неделю дается дополнительная задача 'на обычном языке'. В ней студентам предлагается реализовать рассмотренные на занятии темы на каком-нибудь распространённом языке (C#, C++, Java, Python и т.д.)

Таким образом, на преподавателя ложится довольно большая нагрузка. Каждую неделю приходится проверять около 150 задач. Хотя большинство из них простые, нагрузка, тем не менее, очень значительная, и справиться с ней без инструментальных средств было бы невозможно. В дальнейшем мы будем говорить о средствах, которые позволяют упросить проверку задач первых двух типов. Задачи на обычном языке настоящее время проверяются вручную.

Система тестирования задач

Для проверки задач используется тестирование. Оно было реализовано в системе в первой версии, в 2011 г., в дипломной работе, выполненной А.Ю.Соловьевым. К задачам прилагаются тесты, и решение, не прошедшее эти тесты, не считается засчитанным.

Проверка задач с помощью тестов используется в большинстве массовых онлайн курсов (например, в курсе Stepic, подготовленным Д.Н.Москвиным [4]). В отличие от таких курсов, в рассматриваемой системе автоматическое засчитывание задач, у которых прошли все тесты, не предусмотрено, они в любом случае просматриваются и засчитываются вручную. В следующей версии предполагается реализовать автоматическое засчитывание задач с пройденными тестами, но не для всех, а только для задач, повышенной сложности. Для такого автоматического засчитывания будет необходимо также обеспечить контроль списывания, он обсуждается ниже.

По сравнению с распространёнными системами тестирования, реализованная подсистема тестирования обладает рядом особенностей:

1. Студенту сообщается тестовый пример вызова, на котором его решение дало неправильные результаты. В большинстве систем тестирования [4,5] сообщается только номер теста. Однако практика показала, что для большинства студентов такой подход оказывается слишком сложным. Им сложно самостоятельно придумать пример, для которого программа могла бы не работать, и, в результате, отсутствие информации о тестах привело бы к тому, что эти тесты появились бы у студентов в сети и т.д.
2. Существует, однако, возможность не показывать некоторые тесты. В этом случае преподаватель может задать подсказку для студента. Такой подход применяется, когда показ тестового примера делает задачу слишком простой. Например, если студент забыл обработать случай, когда входное значение меньше 0, то интереснее будет ему на это намекнуть (выведя сообщение «Числа бывают отрицательными)»), а не просто показать пример вызова.
3. Еще одна особенность - это то, что система является достаточно гибкой, чтобы задавать тесты в тех случаях, когда условие предусматривает не единственное решение. Например, одна из решаемых на курсе задач такая: «Создать бесконечный список из всех пар положительных целых чисел». В этой задаче предлагается придумать порядок обхода всех пар самостоятельно, и он может быть различным. Однако система тестирования оказывается достаточно гибкой для того, чтобы в качестве тестов можно было задавать такие условия, как, например, «проверить, что первые 1000 элементов последовательности содержат пару (3,5) ровно один раз» и т.д. (рис 1). Такие условия, конечно, не являются достаточными, однако, на практике, успешно отсеивают неправильные решения.

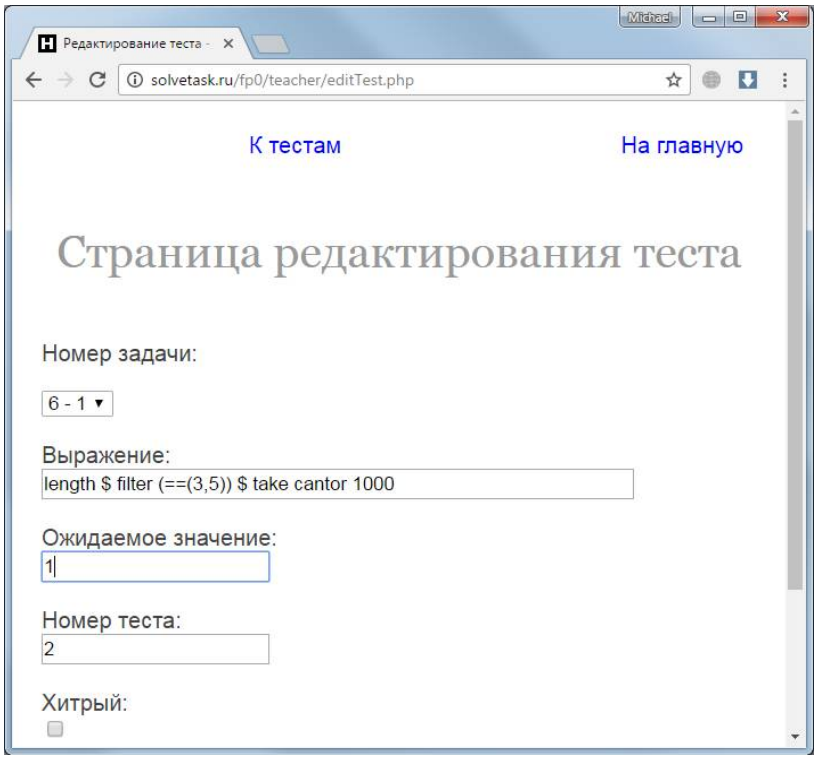

Рис.1. Пример теста со сложным условием

Автоматизация проверки задач

Для большинства задач засчитывание их только по тестам представляется, на наш взгляд, недостаточным. Особенно это относится к простым задачам. Для таких задач имеет значение, не только то, что задача прошла тесты, но и как она написана. Студент может выбрать неоптимальный алгоритм, или использовать не лучшее представление данных, или даже просто использовать неудачное имя переменной, и на такие ошибки очень желательно ему указать. На наш взгляд, именно замечания, которые студент получает от преподавателя, являются наиболее важным элементом процесса обучения (и это подтверждают многие отзывы студентов, окончивших курс).

Приведем некоторые другие причины, по которым засчитывание задач на основе тестов является нежелательным:

- Многие задачи включают в себя дополнительные условия. Например, «Решение не должно использовать рекурсию» или «Решение должно использовать функцию foldr» или «Решение должно использовать стиль передачи продолжений (continuation-passing style)». Для проверки таких условий тестов, естественно, недостаточно.
- Преподавателю важно получить представление о том, какие варианты решения и какие ошибки наиболее типичны для данных студентов, что у них вызывает трудности. Именно разбор типичных ошибок является, по нашему опыту, одной из наиболее интересных и полезных для студента частью занятия.
- К сожалению, еще одна важная проблема автоматического засчитывания программ по тестам - это списывание. В отличие от массовых онлайн курсов, мы точно не можем тут рассчитывать на сознательность участников.

Таким образом, одна из целей, которой мы пытаемся добиться на курсе - это чтобы у студентов возникало ощущение личного общения с преподавателем, и чтобы он получал личные замечания о его программе – комментарии об ошибках, советы и т.д. Такой подход неизбежно требудет довольно больших трудозатрат.

В то же время и при таком подходе есть возможность облегчить работу преподавателя. В.И.Шайтан в 2016 году при подготовке бакалаврской ВКР разработал систему автоматического зачитывания задач на основе типичных решений и автоматизированного проставления замечаний. Рассмотрим ее основные возможности.

Автоматическое засчитывание типичных правильных решений. Одна из особенностей простых задач, которые даются для закрепления материала в том, что у них, как правило, есть небольшое количество возможных правильных решений, и большинство студентов присылают одно из них.

Например, для закрепления темы «бесконечные списки» дается задача «Описать бесконечный список [7,77,777,7777,7777…]». Для этой задачи есть не так мало решений, которые приходят в голову студентам. Но есть примерно 3-4 наиболее типичных правильных решения, которые, в совокупности, присылают примерно 50% студентов. Из оставшихся студентов примерно 40% присылает тоже 4-5 типичных вариантов решения, но эти варианты не совсем правильные, для них требуется написать замечания.

Важной особенностью, характерной, видимо, именно для функционального программирования, является то, что эти типичные решения часто совпадают (с точностью до имен переменных). Это связано с тем, что что в языке Haskell студенту не надо реализовывать ввод и вывод данных, подключать библиотеки и писать другой вспомогательный код, который каждый студент напишет немного по-другому. Решение простой задачи состоит, во многих случаях, из 1-10 строк кода.

Система автоматического засчитывания задач на основе правильных решений позволяет преподавателю хранить такие правильные решения в базе и засчитывать их автоматически (рис.2). Таким образом, у преподавателя освобождается время для работы с 10-20% студентов, приславших нестандартные решения – или решения с ошибками, или, наоборот, решения, использующие нестандартные подходы.

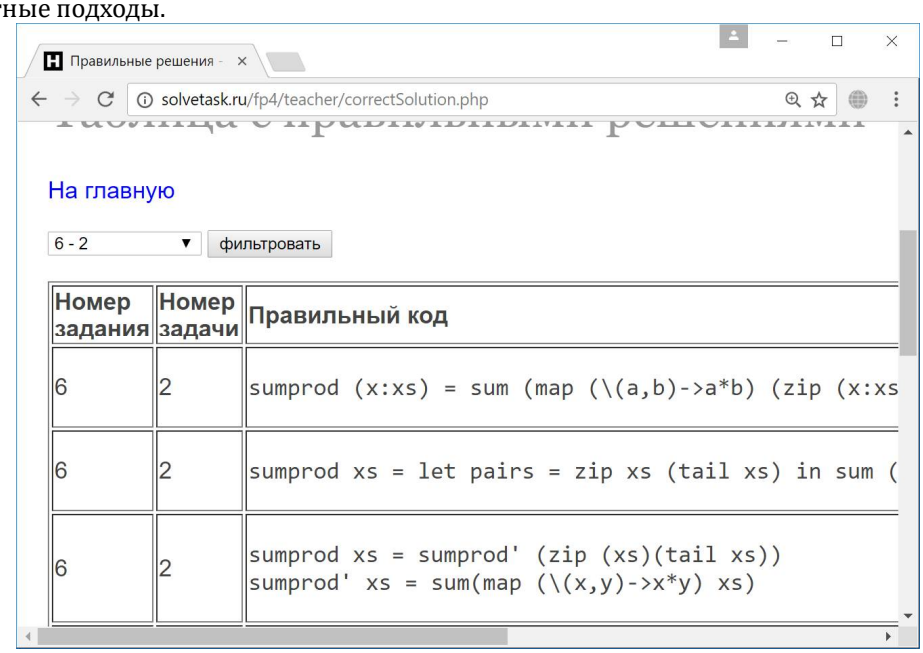

Рис.2. Пример типичных правильных решений

Отметим некоторые особенности засчитывания задач:

1. Даже если решение студента совпадает с правильным, ему не сообщается об этом сразу. Преподаватель должен нажать кнопку «Засчитать одобренные решения», и это делается, как правило, раз или два в день. Смысл в том, чтобы студент не знал, что некоторые из его решений засчитываются автоматически. Как нам кажется, если студент будет знать, что при некоторых обстоятельствах его решения зачтется сразу, у него может возникнуть ощущение, что такое решения является более правильным, он может попытаться подобрать его и т.д.
2. Перед сравнением присланного решения с правильными образцами оно, в любом случае, проверяется тестами. Это значительно упрощает сравнение задач. Например, мы можем, заменять переменные на их порядковые номера (чтобы сравнить программы с точностью до имен переменных) и не беспокоиться о том, что можем про неправильную программу решить, что она в точности равна правильной и засчитать. Сравниваемые программы точно по крайней мере дают правильные результаты, иначе они не прошли бы тесты.

Каталог типичных замечаний. Замечания, которые преподаватель пишет студентам, тоже часто повторяются. Как правило, для каждой задачи есть 3-5 типичных замечаний, которые охватывают 30-80% всех случаев. В текущей версии системы реализован простой набор средств, позволяющий облегчить процесс проставления замечаний. Для каждой задачи можно ввести и запомнить несколько типичных замечаний. При проверке задачи этот список показывается рядом с проверяемым решением, и преподаватель может одним нажатием вставить его в нужное место (рис. 3).

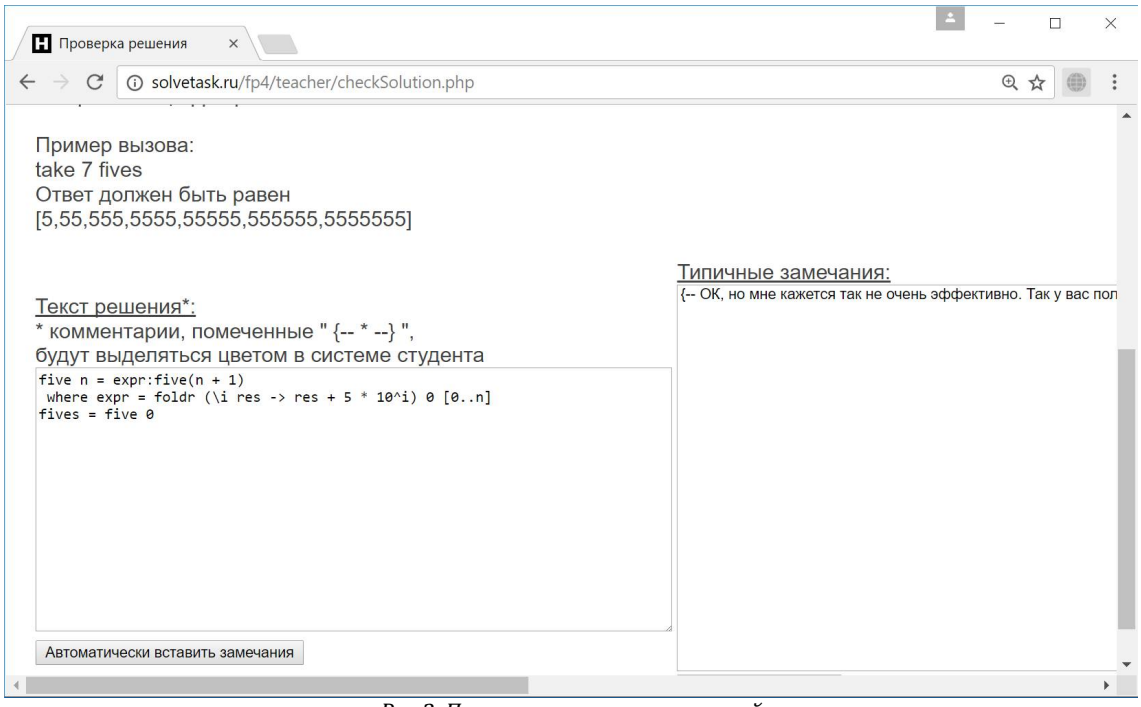

Рис.3. Пример типичных замечаний

Автоматизированное проставление замечаний. Еще одна возможность, реализованная в системе, но оказавшаяся не очень востребованной, - это автоматизированное проставление замечаний. Замечания могут снабжаться условием, при котором они возникают. В настоящее время условие – это просто фрагмент текста. Например, “if (“ - это условие для того, чтобы добавить типичное замечание «После if скобки можно не писать».

По результатам эксплуатации выяснилось, что эта возможность востребована мало, и, видимо, требует доработки. Во-первых, как оказалось, для большинства реальных задач трудно сформулировать надежное условие, которе позволяло бы совершенно точно добавить это замечание. При этом цена ошибки довольно высока. Не сообщить студенту об ошибке не так страшно, а вот написать ему об ошибке, которую он, на самом деле, не делал, это было бы неприятно. Поэтому в следующей версии предполагается реализовать более надежный подход, в котором замечания проставляются автоматически только в тех случаях, когда мы точно в этом уверены.

Недостатки текущей версии системы и планируемые дополнения

В настоящее время проектируется следующая версия системы. Опишем ее основные возможности:

1. Нормализация типичных решений

Даже в самых простых задачах все-таки образуется много типичных решений из-за того, что студенты используют разные варианты записи конструкций. Приведем простой пример. Допустим, в задаче требуется записать выражение x*x+1. Даже те студенты, которые прислали в остальном одинаковое решение, напишут это по-разному – x^2+1, 1+x*x, (x*x)+1, (x*x+1) и различные сочетания этих вариантов. Таким образом, приходится хранить в базе очень много похожих вариантов решения.

Для решения этой проблемы предполагается задавать набор преобразований, которые, если есть такая возможность, следует применить к решению, чтобы привести его к единому нормализованному виду. Например, в рассмотренном выше примере, таким преобразованиями могли бы быть правила «Заменить x*x на x^2” и т.д. Применение таких преобразований позволит значительно уменьшить набор правильных решений, хранящихся в системе. Схожая технология для автоматической генерации замечаний описана в [6].

2. Автоматическая обработка типичных неправильных или не совсем правильных решений.

Хотелось бы иметь возможность автоматически засчитывать (или не засчитывать) типичные не совсем правильные решения задач, добавляя, при необходимости, нужные замечания. Сейчас в системе это не предусмотрено, автоматически можно засчитывать только полностью правильные задачи. Это добавление просто реализовать, и оно существенно упростило бы работу преподавателя.

3. Генерация статистки по типичным решениям

Хотелось бы иметь представление о том, как распределились решения студентов – сколько

процентов выбрало оптимальное решение, сколько не очень. Такая статистка, была бы полезна для преподавателя, и, видимо, интересна для студентов. Предполагается обеспечить возможность узнать, какой процент участников выбрал тот или иной вариант решения или сделал ту или иную типичную ошибку.

4. Контроль списывания

При использовании автоматического засчитывания задач стала еще более актуальной проблема борьбы со списыванием. Списывание является сложной проблемой и вряд ли возможно решить ее полностью, особенно для такого курса, где одинаковые решения для многих задач являются скорее нормой. Но хотелось бы обнаруживать хотя бы самые очевидные случаи, например, полностью идентичные решения.

Следует отметить, однако, что анонимный опрос, проведенный среди участников курсов прошлых лет [7] показал, что большинство участников, если и писали задачи не самостоятельно, то делали это 1-2 раза за курс. Возможно это связано с подбором задач, они, именно с целью минимизировать списывание даются или достаточно простыми, чтобы даже слабо подготовленным студентам было интересно попробовать решить из самим, или достаточно сложными, чтобы студенты не рисковали копировать чужое решения, понимая, что это легко может быть обнаружено.

Тем не менее, предполагается включить в состав системы базовые возможности по контролю за списыванием:

Проверка задач на точное совпадение, с учетом имен переменных, пробелов и переносов.

Ведение статистики по неточному совпадению и поиск пар участников, у которых большое количество задач совпадают без учета имен переменных, пробелов и других простых преобразований.

Более далекие планы развития системы

Перечислим кратко направления, в которых предполагается развивать систему в дальнейшем:

Персонализация замечаний. Часто бывает, что преподаватель пишет разные замечания для одной и той же ошибки для разных студентов. Текст зависит от подготовки студента (сильным студентам требуется короткое замечание, а слабым – более подробное), от того, делал ли студент уже эту ошибку раньше, иногда и от других факторов. Предполагается обеспечить поддержку таких персонализированных сообщений. Для этого в системе должна храниться информация об уровне подготоки студента, об уже полученных им замечаниях, возможно и другая информация, помогающая обеспечить более понятное для данного участника сообщение.

Использование методов извлечения знаний для сопоставления. Возможным направлением развития системы является использование методов извлечения знаний для сопоставления программ, не совпадающих с типичными образцами, и типичных замечаний. Аналогичная работа описана в [8], там с этой целью используются методы кластеризации.

Литература

1. Функциональное программирование (мат-мех СПбГУ) URL: http://msimuni.wikidot.com/fp4
2. Функциональное программирование. Страница пользователя URL: http://solvetask.ru/fp4
3. Haskell Language URL: https://www.haskell.org/.
4. Функциональное программрование на языке Haskell – Stepic.org URL: https://stepik.org/course/Функциональное-программирование-на-языке-Haskell-75.
5. Sphere Online Judge (SPOJ) URL: http://www.spoj.com/.
6. Singh, Rishabh and Gulwani, Automated Feedback Generation for Introductory Programming Assignments. SIGPLAN Not. 48 (6), 2013, pp.15—26
7. Анкета про списывание и совместную работу над решениями в курсе ФП URL: http://webanketa.com/forms/64r38c1r6rqkjdhpcgtk0rk1/ru/statistic/
8. Gulwani, Sumit, Ivan Radicek, and Florian Zuleger. "Automated Clustering and Program Repair for Introductory Programming Assignments." arXiv preprint arXiv:1603.03165 (2016).

References

1. Functional Programming (mat-meh SPbGU) URL: http://msimuni.wikidot.com/fp4
2. Functional Programming. User home page URL: http://solvetask.ru/fp4
3. Haskell Language URL: https://www.haskell.org/.
4. Functional Programming in Haskell – Stepic.org URL: https://stepik.org/course/Функциональное-программирование-на-языке-Haskell-75.
5. Sphere Online Judge (SPOJ) URL: http://www.spoj.com/.
6. Singh, Rishabh and Gulwani, Automated Feedback Generation for Introductory Programming Assignments. SIGPLAN Not. 48 (6), 2013, pp.15—26
7. Poll on cheating in functional programming course. URL: http://webanketa.com/forms/64r38c1r6rqkjdhpcgtk0rk1/ru/statistic/

8. Gulwani, Sumit, Ivan Radicek, and Florian Zuleger. "Automated Clustering and Program Repair for Introductory Programming Assignments." arXiv preprint arXiv:1603.03165 (2016).

Поступила 14.10.2016

Об авторах:

Симуни Михаил Лазаревич, старший преподаватель кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета, simuni@mail.ru;

Соловьев Алексей Юрьевич, ведущий программист, JobToday SA;

Шайтан Василий Ильич, студент Санкт-Петербургского политехнического университета Петра Великого, институт компьютерных наук и технологий.