УДК 004.4'4
DOI: 10.26102/2310-6018/2025.49.2.004

Разработка транслятора кода из C в Promela, как компонента интеллектуальной обучающей системы для обучения программированию

К.С. Кулюкин✉, Г.А. Якимов, Д.А. Смутин

Волгоградский государственный технический университет, Волгоград, Российская Федерация

Резюме. Представленная работа посвящена разработке транслятора с языка Си в язык Promela для автоматической верификации программ студентов, обучающихся программированию. Целью является создание инструмента, позволяющего проверять корректность промежуточных шагов выполнения программы с использованием Model Checking. Предложенный транслятор ориентирован на последовательные программы с единственной функцией main, работающие с целыми числами и массивами. Он анализирует С-код студента, требования к диапазонам входных данных и иерархическую LTL-спецификацию (дерево целей), описывающую ожидаемое поведение программы. В процессе трансляции используется clang для построения синтаксического дерева, а также создаются дополнительные переменные для отслеживания обращений к массивам. Сгенерированный Promela-код содержит объявления переменных, главный процесс, включающий недетерминированный ввод переменных и транслированный С-код, а также блок LTL-свойств. Полученная Promela-модель верифицируется с помощью Spin на LTL-свойствах, заданных преподавателем. В случае обнаружения нарушения генерируется контрпример, демонстрирующий трассу выполнения программы с ошибкой. Результатом работы является консольная утилита на Python, которая генерирует .pml файл с Promela-кодом и LTL-свойствами, а также .json файл с размеченным деревом целей и контрпримером. В дальнейшем планируется автоматизировать генерацию LTL-свойств из описания требований на естественном языке и генерировать подсказки для студентов на основе контрпримеров.

Ключевые слова: интеллектуальные обучающие системы, обучение программированию, формальная верификация, проверка моделей, трансляция кода.

Для цитирования: Кулюкин К.С., Якимов Г.А., Смутин Д.А. Разработка транслятора кода из С в Promela, как компонента интеллектуальной обучающей системы для обучения программированию. *Моделирование, оптимизация и информационные технологии.* 2025;13(2). URL: https://moitvivt.ru/ru/journal/pdf?id=1860 DOI: 10.26102/2310-6018/2025.49.2.004

Development of a code translator from C to Promela as a component of an intelligent tutoring system for programming learning

K.S. Kulyukin✉, G.A. Yakimov, D.A. Smutin

Volgograd State Technical University, Volgograd, the Russian Federation

Abstract. This work focuses on the development of a C to Promela translator for the automated verification of programs written by programming students. The goal is to create a tool that allows checking the correctness of intermediate program execution steps using Model Checking. The proposed translator is geared towards sequential programs with a single main function, operating on integers and arrays. It analyzes the student's C code, input data range requirements, and a hierarchical LTL specification (goal tree) that describes the expected program behavior. The translation process utilizes clang to construct the syntax tree and creates additional variables to track array accesses. The generated Promela code contains variable declarations, a main process that includes non-deterministic variable

input and the translated C code, and an LTL properties block. The resulting Promela model is verified using Spin against the LTL properties specified by the instructor. If a violation is detected, a counterexample is generated, demonstrating the program execution trace containing the error. The result of this work is a command-line utility written in Python that generates a .pml file with Promela code and LTL properties, as well as a .json file containing the annotated goal tree and the counterexample. Future plans include automating the generation of LTL properties from natural language requirement descriptions and generating hints for students based on the counterexamples.

Keywords: intelligent tutoring systems, programming learning, formal verification, model checking, code translation.

For citation: Kulyukin K.S., Yakimov G.A., Smutin D.A. Development of a code translator from C to Promela as a component of an intelligent tutoring system for programming learning. *Modeling, Optimization and Information Technology*. 2025;13(2). (In Russ.). URL: https://moitvivt.ru/ru/journal/pdf?id=1860 DOI: 10.26102/2310-6018/2025.49.2.004

Введение

В эпоху цифровой трансформации [1] возрастает потребность в обучении IT-специалистов. При обучении программированию новички часто совершают ошибки [2] в кодировании, для устранения которых требуется взаимодействие преподавателем. Для частичной автоматизации работы преподавателя применяют интеллектуальные обучающие системы (в англоязычной литературе известные как «Intelligent Tutoring Systems») [3], т. е. программы, имитирующие взаимодействие студента и преподавателя при выполнении учебного задания.

В интеллектуальной обучающей системе Ask-Elle [4], предназначенной для обучения программированию, спецификации задач представлены в виде набора моделей некорректных решений и соответствующих им диагностических сообщений. Такой подход, хотя и позволяет предоставить студенту релевантную обратную связь, создает значительные трудности для преподавателя при расширении системы на новые задачи. Solvelets [5] использует в качестве спецификации эталонный код. Однако, в случае возникновения ошибок в решении, студент получает обратную связь, указывающую на несоответствие его кода эталонному на уровне отдельных лексем, что зачастую не позволяет понять причины отклонения в поведении программы. CodeQ [6] использует спецификацию в виде эталонного кода, при этом используя также правила для нормализации кода студента с целью его дальнейшего сравнения. В качестве обратной связи CodeQ сообщает студенту фрагменты эталонного кода, тем самым мешая ему осознавать причину ошибки.

Плагин Preg [7] для Moodle предоставляет возможность создавать вопросы с открытым ответом, где множество допустимых решений определяется регулярным выражением. В контексте заданий по программированию это позволяет задавать спецификацию корректного кода в виде регулярного выражения, описывающего все возможные правильные решения. Однако, в случае ошибки, обратная связь ограничивается синтаксическими исправлениями (например, рекомендациями по замене символов), упуская из виду необходимость логического объяснения некорректности кода. Более того, создание всеобъемлющего регулярного выражения, охватывающего все допустимые варианты решения, представляет собой сложную задачу.

Использование спецификации в виде тестов «черного ящика» [8] позволяет обнаруживать ошибки на основе несоответствия выходных данных программы студента эталонным значениям. Однако этот метод не позволяет проверить корректность промежуточных шагов вычислений. Более того, тесты, разработанные без учета специфики программы, могут пропустить определенные ошибки из-за недостаточного покрытия тестовыми сценариями.

В отличие от тестов «черного ящика», спецификация в виде property-based тестов [9] представляет собой математическое описание ожидаемого результата работы функции. Тем не менее, данный тип спецификации предназначен для проверки функций целиком, а не для анализа корректности отдельных строк кода.

Метод проверки моделей (в англоязычной литературе – «Model Checking») [10, 11] подразумевает построение модели поведения системы с целью проверки соответствия ее спецификации, заданной в виде утверждений темпоральной логики [12]. Спецификация, являющаяся независимым от кода объектом [10], позволяет использовать ее для проверки разных решений одной и той же задачи. Однако требуется перевод кода в язык описания моделей (например, Promela [13]), а существующие инструменты, такие как Modex [14], не учитывают диапазоны входных данных и требуют ручного аннотирования кода [10] преподавателем, из-за чего проверка каждого нового решения требует создания новой спецификации.

Целью данной работы является разработка транслятора программ на Си в язык Promela для Model Checking. Транслятор ориентирован на последовательные трансформирующие программы с единственной функцией main, работающие с целыми числами (в ограниченном диапазоне) и целочисленными массивами ограниченной длины.

Материалы и методы

Разрабатываемый инструмент для проверки промежуточных шагов принимает на вход код студента на Си C_code, требования к диапазонам входных переменных Input_req и иерархическую LTL-спецификацию промежуточных шагов программы Spec_ltl, также называемую «деревом целей» [15], листьями которого являются цели программы, представленные в виде LTL-свойств.

Для построения синтаксического дерева используется clang. Метод преобразования кода аналогичен предложенному Ке Янгом [16], но дополнен созданием переменных для отслеживания обращений к переменным и элементам массива. Во избежание конфликтов, переменные студента получают префикс stud_, а переменные для отслеживания индексов массивов – префикс indref _<arr>_<ind>. Функция преобразования С-кода в Promela обозначена как Tr(C) -> VarDecls, AddStmts, Stmts, где VarDecls – объявления переменных, Stmts – операторы для узла, AddStmts – операторы, добавляемые в вышестоящие узлы (например, запись индекса перед обращением к массиву). Имитация ввода значений переменных из Input_req реализуется недетерминированным выбором значений переменных в заданном диапазоне (для массивов – и длины, от 0 до максимума). Функция генерации недетерминированного ввода обозначена: NondetInit(Input_req) -> Promela. Правила преобразования кода из Си в Promela приведены в Таблице 1.

Таблица 1 – Правила преобразования из языка Си в Promela
Table 1 – Rules for translating from C language to Promela

<table>
<tr><td></td><td>C</td><td colspan="2">Tr(C,[Input_req])->VarDecls,AddStmts,Promela
Условные обозначения: V – VarDecls, A – AddStmts, P – Promela</td></tr>
<tr><td rowspan="2">Функция main</td><td rowspan="2">int main() {
<function body>
}</td><td>V</td><td>Tr(<function body>).VarDecls</td></tr>
<tr><td>P</td><td>init {
NondetInit(Input_req) +
Tr(<function body>).Promela
}</td></tr>
</table>

Таблица 1 (продолжение)
Table 1 (continued)

Условный оператор	if (<cond>){ <ifbody> } [else{ <elsebody> }]	V	Tr(<cond>).VarDecls + Tr(<ifbody>).VarDecls + Tr(<elsebody>).VarDecls
		P	Tr(<cond>).AddStmts + if :: (Tr(<cond>).Promela) -> Tr(<ifbody>).Promela ::else -> [Tr(<elsebody>).Promela] fi;
Цикл while	While (<cond>) { <whilebody> }	V	Tr(<cond>).VarDecls + Tr(<whilebody>).VarDecls
		P	Tr(<cond>).AddStmts + do ::true -> if :: (!(Tr(<cond>))) -> break ::else -> Tr(<whilebody>) od;
Цикл do-while	do { <whilebody> } while (<cond>)	V	Tr(<cond>).VarDecls + Tr(<whilebody>).VarDecls
		P	Tr(<whilebody>) + Tr(<cond>).AddStmts + do ::true -> if :: (!(Tr(<cond>))) -> break ::else -> Tr(<whilebody>) od;
Цикл for	for (<s1>;<s2>;<s3>) { <forbody> }	V	Tr(<s1>).VarDecls + Tr(<s2>).VarDecls + Tr(<forbody>).VarDecls
		P	<s1>; do ::true -> if :: ((!Tr(<s2>))) -> break ::else -> Tr(<forbody>);Tr(<s3>) od
Оператор-выражение	<expr>;	V	Tr(expr).VarDecls
		P	Tr(expr).AddStmts + Tr(expr).Promela
Обращение к элементу массива	<arr>[<ind>]	V	Tr(<ind>).VarDecls + <arr>_indref_<numindref>
		A	Tr(<ind>).AddStmts + indref_<arr>_<numindref>=<ind>; + refered_<arr>[indref_<arr>_<numindref>] = 1;
		P	<arr>[indref_<arr>_<numindref>]
Бинарная операция	<leftop><operator> <rightop>	V	Tr(<leftop>).VarDecls + Tr(<rightop>).VarDecls
		A	Tr(<leftop>).AddStmts + Tr(<rightop>).AddStmts
		P	Tr(<leftop>)<operator>Tr(<rightop>)
Переменная	<var_name>	P	stud_<var_name>
Числовая константа	<num>	P	<num>
Объявление переменной	<type> <var_name>	P	TypeMap(<type>) stud_<var_name>;

Итоговый Promela-код состоит из глобальных объявлений переменных (необходимость для проверки LTL-свойств) и процесса init, включающего в себя недетерминированный ввод переменных и код, полученный трансляцией С-кода (Tr(C).Promela). К LTL-свойствам преподавателя автоматически добавляется LTL-свойство для проверки выхода за границы массивов. На Рисунке 1 приведена общая структура генерируемого файла .pml. На Рисунке 2 приведены код студента на языке Си и сгенерированный из него код на Promela. На Рисунке 3 приведены следующие компоненты файла .pml: блок объявлений переменных, недетерминированный выбор значений входных переменных и блок LTL-свойств.

```
//...Блок объявлений переменных

init
{
     //... Недетерминированный выбор значений
входных переменных
     //... Код, сгенерированный из кода студента
}

// Блок LTL-свойств
```

Рисунок 1 – Общая структура файла .pml
Figure 1 – The common structure of the .pml file

```
// Задание – определить, является
ли массив      arr неубывающим
// Длина массива – от 0 до 5
// Значения элементов массива от -
3 до 3
int main() {
    int i,len,arr[5],is_inc;
    is_inc = 1;
    // Ошибка: из-за того, что i
увеличивается          не на 1, а
на 2, не все пары элементов
       рассматриваются
    for (i=0; i<len-1; i=i+2) {
        if (arr[i]<arr[i+1]) {
            is_inc = 0;
        }
    }
}
```

a) Код студента на языке Си

```
//...Код студента, переведенный на Promela
stud_is_inc=1; stud_i=0;
do
   ::(true) ->
      if
         ::(stud_i<stud_len-1) ->
indref_arr_1=stud_i;
          refered_arr[indref_arr_1]=1;
          indref_arr_2=stud_i+1;
          refered_arr[indref_arr_2]=1;
           if
                ::(stud_arr[indref_arr_1] <
stud_arr[indref_arr_2]) -> stud_is_inc=0
::else ->
            fi;
            stud_i=stud_i+2
            ::else -> break
         fi
   ::else ->
od
```

b) Код на языке Promela, сгенерированный из кода студента на языке Си

Рисунок 2 – Пример генерации кода Promela из кода студента на языке Си
Figure 2 – An example of generating of Promela code from student's C code

```
//...Объявления переменных
из программы студента
int stud_i; int stud_len;
int stud_arr[5]; int
stud_is_inc;

//...Объявления
вспомогательных переменных
bool refered_arr[5]; int
indref_arr_1;
int indref_arr_2;
```

a) Объявления переменных в коде на языке Promela

```
//..Недетерминированный выбор
значений входных переменных
int num;
// Недетерминированный выбор
длины массива
if  :: stud_len = 0 : stud_len =
1 ... :: stud_len = 4
::else ->  fi;
//Заполнение элементов массива
do ::(num < stud_len) ->
 //Недетерминированный выбор
значения элемента массива
 if :: stud_arr[num] = -3  ...
:: stud_arr[num] = 2  ::else ->
fi; num=num + 1::else -> break
od;
```

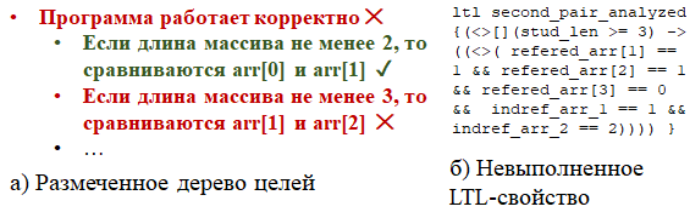
b) Недетерминированный выбор значений входных переменных

```
//... Блок LTL-свойств

//...Автоматически
сгенерированное свойство: никогда
не происходит выхода за пределы
массива
ltl indexes_in_bound {[](
(indref_arr_1>=0 &&
indref_arr_1<=5) &&
(indref_arr_2>=0 &&
indref_arr_2<=5) )}


//...Свойства, созданные
преподавателем

// Если длина массива не меньше
2, то первая пара элементов
(arr[0] и arr[1]) всегда
рассматривается

ltl first_pair_analyzed
{(<>[](stud_len >= 2) -> ((<>(
refered_arr[0] == 1 &&
refered_arr[1] == 1 &&
refered_arr[2] == 0 &&
indref_arr_1 == 0 && indref_arr_2
== 1)))) }
```

c) Блок LTL-свойств

Рисунок 3 – Примеры компонентов файла .pml
Figure 3 – An example of .pml file's components

Далее модель Promela верифицируется с помощью Spin. Сначала проверяется автоматически сгенерированное свойство, контролирующее выход за границы массивов. Затем, в порядке их указания, проверяются LTL-свойства, заданные преподавателем. В случае нарушения любого свойства проверка останавливается, соответствующее свойство отмечается в дереве целей, и генерируется контрпример, представляющий собой трассу выполнения программу, демонстрирующую нарушение LTL-свойства. Пример верификации программы приведен на Рисунке 4.



а) Размеченное дерево целей

б) Невыполненное LTL-свойство

stud_arr	stud_i	refered_Arr[0]	referef_arr[1]	refered_arr[2]	refered_arr[3]
[0,0,0]	0	0	0	0	0
[0,0,0]	0	1	1	0	0
[0,0,0]	2	1	1	0	0
[0,0,0]	2	1	1	1	1

в) Контрпример в виде трассы выполнения программы

Рисунок 4 – Пример верификации программы
Figure 4 – An example of program verification

Результаты и обсуждение

Реализована консольная утилита на Python, принимающая на вход код программы студента на языке Си, требования к входным переменным и спецификацию программы в виде LTL-свойств, заданных преподавателем. Выходами программы являются: а) .pml файл с Promela-кодом и LTL-свойствами б) .json файл с размеченным деревом целей и контрпримером (при наличии).

Данная утилита может быть использована как компонент интеллектуальной обучающей системы для обучения программированию. В будущем планируется генерация LTL-свойств из требований к алгоритму программы, описанных на естественном языке [17]. Контрпример, генерируемый утилитой, сложен для понимания [18], особенно новичкам. Однако, коды студента и контрпример, полученный с помощью разработанной утилиты, могут быть использованы для автоматизированной генерации подсказок об ошибках в программе студента [18] в виде текстовых сообщений о причинах ошибок [19] и визуализации [20] некорректной работы программы. На Рисунке 5 приведен желаемый пример обратной связи со студентом.

```
int main() {
    int i,len,arr[5],is_inc;
    is_inc = 1;
for (i=0; i<len-1; i=i+2) {
        if (arr[i]<arr[i+1]) {
            is_inc = 0;
        }
    }
}
```

В связи с тем, что на каждой итерации цикла счетчик увеличивается на 2, а не на 1, программа рассматривает не все пары подряд идущих элементов массива

Рисунок 5 – Пример желаемой обратной связи
Figure 5 – An example of the desired feedback

Заключение

Разработанная утилита, включающая в себя транслятор кода из языка C в Promela, позволяет автоматически проверить соответствие поведения программного кода студента спецификации его поведения, заданной в виде LTL-свойств, означающих требования к промежуточным шагам программы, и генерировать контрпримеры в случае ее нарушения. Для использования разработанной утилиты, как компонента интеллектуальной обучающей системы для обучения программированию, планируется автоматизировать построение спецификации поведения программы и генерацию обратной связи со студентом на основе контрпримеров.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Рудник П.Б., Зинина Т.С., Акиндинова Н.В. и др. *Цифровая трансформация: эффекты и риски в новых условиях*. Москва: ИСИЭЗ ВШЭ; 2024. 156 с.
   Rudnik P., Zinina T., Akindinova N., et al. *Digital Transformation: Effects and Risks in New Conditions*. Moscow: ISSEK HSE; 2024. 156 p. (In Russ.).
2. Rathore A.S., Arjaria S.K. Intelligent Tutoring System. In: *Utilizing Educational Data Mining Techniques for Improved Learning: Emerging Research and Opportunities*. Hershey PA: IGI Global; 2020. P. 121–144. https://doi.org/10.4018/978-1-7998-0010-1.ch006
3. Lewandowski G., Gutschow A., McCartney R., Sanders K., Shinners-Kennedy D. What Novice Programmers Don't Know. In: *ICER '05: Proceedings of the International Computing Education Research Workshop, 01–02 October 2005, Seattle, WA, USA*. New York: Association for Computing Machinery; 2005. P. 1–12. https://doi.org/10.1145/1089786.1089787
4. Gerdes A., Heeren B., Jeuring J., Van Binsbergen Th. Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback. *International Journal of Artificial Intelligence in Education*. 2017;27(1):65–100. https://doi.org/10.1007/s40593-015-0080-x
5. Kumar A.N. Solvelets: Tutors to Practice the Process of Programming. In: *ITiCSE '22: Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2022), 08–13 July 2022, Dublin, Ireland*. New York: Association for Computing Machinery; 2022. P. 151–157. https://doi.org/10.1145/502718.3524811
6. Lazar T., Sadikov A., Bratko I. Rewrite Rules for Debugging Student Programs in Programming Tutors. *IEEE Transactions on Learning Technologies*. 2018;11(4):429–440. https://doi.org/10.1109/TLT.2017.2743701
7. Сычев О.А., Терехов Г.В. Инструменты помощи автору регулярных выражений для тестовых вопросов в СДО Moodle. *Открытое образование*. 2016;(3):43–50. https://doi.org/10.21686/1818-4243-2016-3-43-50
   Sychev O.A., Terehov G.V. Helping Tools for the Regular Expression Author for Test Questions in LMS Moodle. *Open Education*. 2016;(3):43–50. (In Russ.). https://doi.org/10.21686/1818-4243-2016-3-43-50
8. Verma A., Khatana A., Chaudhary S. A Comparative Study of Black Box Testing and White Box Testing. *International Journal of Computer Sciences and Engineering*. 2017;5(12):301–304. https://doi.org/10.26438/ijcse/v5i12.301304
9. MacIver D.R., Hatfield-Dodds Z., Contributors M. Hypothesis: A New Approach to Property-Based Testing. *Journal of Open Source Software*. 2019;4(43). https://doi.org/10.21105/joss.01891

10. Brain M., Polgreen E. A Pyramid Of (Formal) Software Verification. In: *Formal Methods: 26th International Symposium, FM 2024: Proceedings, Part II, 09–13 September 2024, Milan, Italy.* Cham: Springer; 2024. P. 393–419. https://doi.org/10.1007/978-3-031-71177-0_24
11. Clarke E.M., Grumberg O., Peled D.A. *Model Checking*. Cambridge: MIT Press; 2001. 314 p.
12. Clarke E.M., Emerson E.A., Sistla A.P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*. 1986;8(2):244–263. https://doi.org/10.1145/5397.5399
13. Neumann R. Using Promela in a Fully Verified Executable LTL Model Checker. In: *Verified Software: Theories, Tools and Experiments: 6th International Conference, VSTTE 2014: Revised Selected Papers, 17–18 July 2014, Vienna, Austria.* Cham: Springer; 2014. P. 105–114. https://doi.org/10.1007/978-3-319-12154-3_7
14. Holzmann G.J., Ruys Th.C. Effective Bug Hunting with Spin and Modex. In: *Model Checking Software: 12th International SPIN Workshop: Proceedings, 22–24 August 2005, San Francisco, CA, USA.* Berlin; Heidelberg: Springer; 2005. P. 24. https://doi.org/10.1007/11537328_3
15. Kulyukin K., Litovkin D. An Approach to Formal Verification of Programs in Learning C Programming. In: *Novel and Intelligent Digital Systems: Proceedings of the 4th International Conference (NiDS 2024), 25–27 September 2024, Athens, Greece.* Cham: Springer; 2024. P. 443–447. https://doi.org/10.1007/978-3-031-73344-4_37
16. Jiang K., Jonsson B. Using SPIN to Model Check Concurrent Algorithms, using a translation from C to Promela. In: *MCC '2009: Proceeding of the Second Swedish Workshop on Multi-Core Computing, 26–27 November 2009, Uppsala, Sweden*. Uppsala: Department of Information Technology, Uppsala University; 2009. P. 67–69.
17. Wang X., Li G., Li Ch., Zhao L., Shu X. Automatic Generation of Specification from Natural Language Based on Temporal Logic. In: *Structured Object-Oriented Formal Language and Method: 10th International Workshop, SOFL+MSVL 2020: Revised Selected Papers, 01 March 2021, Singapore.* Cham: Springer; 2021. P. 154–171. https://doi.org/10.1007/978-3-030-77474-5_11
18. Kaleeswaran A.P., Nordmann A., Vogel Th., Grunske L. A Systematic Literature Review on Counterexample Explanation. *Information and Software Technology*. 2022;145. https://doi.org/10.1016/j.infsof.2021.106800
19. Feng L., Ghasemi M., Chang K.-W., Topcu U. Counterexamples for Robotic Planning Explained in Structured Language. In: *2018 IEEE International Conference on Robotics and Automation (ICRA), 21–25 May 2018, Brisbane, QLD, Australia*. IEEE; 2018. P. 7292–7297. https://doi.org/10.1109/ICRA.2018.8460945
20. Nguyen T.Th.Th., Ogata K. A Way to Comprehend Counterexamples Generated by the Maude LTL Model Checker. In: *2017 International Conference on Software Analysis, Testing and Evolution (SATE), 03–04 November 2017, Harbin, China*. IEEE; 2017. P. 53–62. https://doi.org/10.1109/SATE.2017.15

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Кулюкин Кирилл Сергеевич, аспирант, преподаватель кафедры «Программное обеспечение автоматизированных систем», Волгоградский государственный технический университет, Волгоград, Российская Федерация.
e-mail: kirill.kuliuckin2016@ya.ru

Kirill S. Kulyukin, Postgraduate, Lecturer at the Software Engineering Department, Volgograd State Technical University, Volgograd, the Russian Federation.

ORCID: 0000-0003-2676-2110

Якимов Григорий Алексеевич, аспирант кафедры «Программное обеспечение автоматизированных систем», Волгоградский государственный технический университет, Волгоград, Российская Федерация.
e-mail: greg.yakimov@gmail.com
ORCID: 0009-0003-6216-9805

Gregory A. Yakimov, Postgraduate at the Software Engineering Department, Volgograd State Technical University, Volgograd, the Russian Federation.

Смутин Даниил Александрович, магистрант кафедры «Программное обеспечение автоматизированных систем», Волгоградский государственный технический университет, Волгоград, Российская Федерация.
e-mail: danilsmutin@gmail.com
ORCID: 0009-0004-0775-6705

Daniil A. Smutin, Master's Degree student at the Software Engineering Department, Volgograd State Technical University, Volgograd, the Russian Federation.

Статья поступила в редакцию 23.03.2025; одобрена после рецензирования 08.04.2025; принята к публикации 11.04.2025.

The article was submitted 23.03.2025; approved after reviewing 08.04.2025; accepted for publication 11.04.2025.