Vestnik of Astrakhan State Technical University.
Series: Management, computer science and informatics. 2023. N. 2
ISSN 2072-9502 (Print), ISSN 2224-9761 (Online)
Computer engineering and software

Научная статья
УДК 004.4'416
https://doi.org/10.24143/2072-9502-2023-2-93-100
EDN PEHBOU

Исследование WebAssembly и сравнение производительности с JavaScript

Виолетта Валерьевна Рокотянская[1], Вадим Сергеевич Абрамов[2]✉

[1]Российский государственный аграрный университет – МСХА имени К. А. Тимирязева, Москва, Россия

[2]Национальный исследовательский университет ИТМО, Санкт-Петербург, Россия, vadim-abramov-00@mail.ru✉

Аннотация. За долгую историю развития Интернета основным языком программирования в веб-разработке был JavaScript. Благодаря простому синтаксису и поддержке во всех популярных браузерах язык обрел большую популярность среди разработчиков. Однако с развитием компьютерных мощностей и требований пользователей простые сайты переросли в веб-приложения, являющиеся полноценными аналогами десктопных приложений. С ростом возможностей выросли и требования к производительности таких программ. В браузерных движках появились способы оптимизация запускаемого кода, а активная конкуренция между браузерами способствовала качественному приросту производительности. Несмотря на все способы увеличить скорость исполнения JavaScript кода главным стоп-фактором была динамическая типизация языка. Из-за динамической типизации движку браузера необходимо каждый раз при выполнении программы проверять, является ли переменная целым числом, float или любым другим допустимым типом. Таким образом, каждая инструкция в JavaScript должна пройти через несколько проверок типов и преобразований, что замедляет ее выполнение. В связи с этим появилась идея использовать языки со строгой типизацией, что могло бы компенсировать данный недостаток. Но браузерные движки не способны выполнять код других языков, поэтому появилась технология под названием WebAssembly. Она позволяет писать код на языках со статичной типизацией и после парсит его в более нативный и приближенный формат для машин, что значительно ускоряет выполнение программ по сравнению с JavaScript. Технология является кроссплатформенной и поддерживает основные языки программирования: C++, C, Java, C#. Также реализован вызов WebAssembly с помощью JavaScript, что позволяет использовать лаконичный синтаксис JavaScript и вычислительные мощности WebAssembly вместе.

Ключевые слова: веб-технологии, JavaScript, WebAssembly, производительность, браузер, скорость работы приложения

Для цитирования: *Рокотянская В. В., Абрамов В. С.* Исследование WebAssembly и сравнение производительности с JavaScript // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. 2023. № 2. С. 93–100. https://doi.org/10.24143/2072-9502-2023-2-93-100. EDN PEHBOU.

Original article

Studying WebAssembly and comparison of its performance with JavaScript

Violetta V. Rokotianskaya[1], Vadim S. Abramov[2]✉

[1]Russian Timiryazev State Agrarian University, Moscow, Russia

[2]ITMO University, Saint-Petersburg, Russia, vadim-abramov-00@mail.ru✉

Abstract. Over the long history of the Internet, JavaScript has been the primary programming language in web development. Because of its simple syntax and support in all popular browsers, the language has gained popularity among

the developers. However, as computer power and user demands evolved, simple sites turned into the web applications that are full-fledged analogues of the desktop applications. As capabilities grew, so did the performance requirements of such programs. Browser engines have developed ways to optimize the code they run, and intense competition between browsers has contributed to a qualitative increase in performance. Despite all the ways to increase the speed of execution of JavaScript code, the main stopping factor was the dynamic typing of the language. Because of dynamic typing, the browser engine needs to check each time the program is executed whether the variable is an integer, a float, or any other valid type. Thus, each JavaScript instruction has to go through several type checks and conversions, which slows down the execution. This led to the idea of using languages with strict typing, which could compensate for this drawback. But the browser engines cannot execute the code of other languages, that is why there appeared the technology WebAssembly. It allows writing code in languages with static typing, and then parses it into a more native and machine-readable format, which speeds up execution of programs compared to JavaScript. The technology is cross-platform and supports the main programming languages: C++, C, Java, C#. WebAssembly is also implemented using JavaScript, which allows to use the concise JavaScript syntax and computing power of WebAssembly together.

Keywords: web technologies, JavaScript, WebAssembly, performance, browser, the speed of the application

For citation: Rokotianskaya V. V., Abramov V. S. Studying WebAssembly and comparison of its performance with JavaScript. *Vestnik of Astrakhan State Technical University. Series: Management, computer science and informatics*. 2023;2:93-100. (In Russ.). https://doi.org/10.24143/2072-9502-2023-2-93-100. EDN PEHBOU.

Введение

Изначально Интернет был разработан как простая сеть обмена документами, но теперь это самая распространенная платформа для поиска информации и работы с приложениями, доступными для огромного спектра операционных систем и типов устройств. Исторически сложилось так, что Java Script являлся единственным языком программирования, который изначально поддерживается и широко используется на большинстве платформ. Из-за широкого распространения и значительного прироста производительности в современных виртуальных машинах JavaScript стал на долгое время практически единственным языком, работающим на стороне браузера. Также появилась возможность компиляции из других языков. При помощи Emscripten даже программы на C и C++ могут быть скомпилированы в стилизованное низкоуровневое подмножество Java Script, называемое asm.js [1]. В итоге появилась возможность разработки веб-приложений на других языках программирования, которые бы по итогу все равно компилировались в JavaScript код.

Из-за долгой истории и, следовательно, постоянного первенства на веб-платформе язык поддерживается и используется практически на всех устройствах и браузерах, но, несмотря на активное развитие и увеличение мощностей вычислительных устройств, производительность JavaScript значительно ниже, чем у других языков. Данный факт может стать проблемой при разработке современных веб-приложений. Веб-платформа перестала состоять исключительно из простых статичных сайтов, и уже существует большое количество веб-аналогов для десктопных приложений, имеющих динамический контент и выполняющих трудные вычислительные операции. Однако JavaScript изначально не разрабатывался под подобные решения, из-за чего язык имеет проблемы со скоростью выполнения кода.

JavaScript имеет непостоянную производительность, а скорость выполняемого кода сильно зависит от различных движков, которые используются в браузерах. Они парсят JavaScript код, образуя из него абстрактное синтаксическое дерево (AST), из которого позже генерируется байткод, компилируемый в машинный, с которым могут работать другие программы. Список операций во время выполнения JavaScript кода представлен на рис. 1, пример преобразования кода в AST-формат представлен на рис. 2.

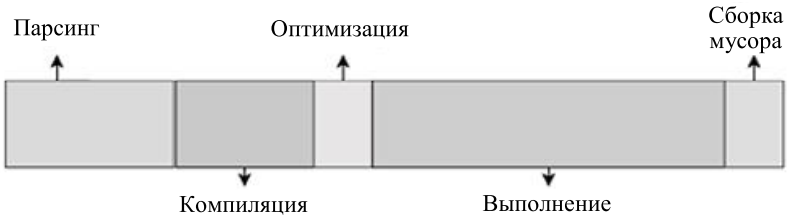

Рис. 1. Операции в процессе выполнения JavaScript кода

Fig. 1. Operations during JavaScript code execution

Рис. 2. Преобразование JavaScript кода в AST-формат

Fig. 2. Converting JavaScript code to AST-format

Слабой стороной языка является динамическая типизация. При выполнении кода браузерный движок собирает информацию о типах объектов, используемых в коде, и запускает компилятор, который формирует машинный код. В языках со статической типизацией нет необходимости в сборе информации, т. к. все типы известны заранее, вследствие чего этап сбора информации отбрасывается и код выполняется быстрее.

Парсинг и компиляция кода являются причинами невысокой производительности JavaScript по сравнению с другими языками. Браузерные движки постоянно обновляются и увеличивают скорость парсинга и выполнения кода. Но все равно многим разработчикам приходится идти на некоторые компромиссы для корректного и быстрого исполнения кода на большинстве устройств. Исходя из всего вышесказанного, технологии, используемые в браузерах, должны удовлетворять следующим требованиям:

– нулевая настройка – это должна быть технология, работающая сразу в браузере;

– безопасность – новая технология не должна создавать новых угроз;

– кросс-платформа – браузеры работают на всех основных процессорах, включая мобильные платформы;

– удобство разработки – если технология имеет плохую документацию или неудобное для работы API, это может сильно повлиять на будущий интерес к ней, а значит и ее развитие.

Под перечисленные выше критерии подходит технология под названием WebAssembly, которая имеет более высокую скорость благодаря низкоуровневому скомпилированному коду.

Актуальность темы определяется необходимостью использования более быстрых и оптимизированных технологий на стороне клиента для повышения производительности веб-приложений при выполнении ресурсоемких задач.

Целью исследования является анализ технологии WebAssembly, сравнение его производительности с языком JavaScript и определение ситуаций и веб-приложений, в которых предпочтительнее будет использование WebAssembly.

Для выполнения поставленной цели необходимо:

– провести анализ технологии WebAssembly, изучить принцип работы и отличие компиляции кода в сравнении с JavaScript кодом;

– сравнить скорость выполнения кода обеих технологий, провести замеры и сравнить их для определения производительности каждой технологии.

WebAssembly

WebAssembly – это технология, предоставляющая новый тип кода, который можно запускать в современных веб-браузерах, что обеспечивает новые функции и значительное повышение производительности [2]. Обычно код для WebAssembly не пишется вручную, он спроектирован для эффективной компиляции из низкоуровневых исходных языков, таких как C, Rust и т. д. Результат компиляции C кода в WebAssembly представлен на рис. 3.

Благодаря статичной типизации WebAssembly код можно сразу же компилировать в машинный, а т. к. в процессе выполнения программы типы измениться не могут, нет никакой деоптимизации. Сами же типы, которые поддерживает WebAssembly, легко можно перевести в машинный код.

```
1  int sum(int a, int b) {
2    int value = 42;
3
4    return a + b + value;
5  }
```

```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "sum" (func $sum))
 (func $sum (; 0 ;) (param $0 i32) (param $1 i32) (result i32)
  (i32.add
   (i32.add
    (get_local $0)
    (get_local $1)
   )
   (i32.const 42)
  )
 )
)
```

Рис. 3. Код на языке C, преобразованный в WebAssembly формат

Fig. 3. C code converted to WebAssembly format

WebAssembly позволяет запускать код, написанный на других языках в веб-приложениях практически с естественной скоростью. Это имеет большое значение для развития веб-приложений, т. к. раньше не было подобной возможности. Также не требуется понимать и знать, как создавать код WebAssembly для его использования. WebAssembly функции и модули можно легко импортировать в веб-приложение и использовать в JavaScript коде. JavaScript фреймворки могут использовать модули WebAssembly для получения огромных преимуществ в производительности и новых функций, в то же время делая их функциональность легко доступной для веб-разработчиков. Основные операции при выполнении WebAssembly кода представлены на рис. 4.

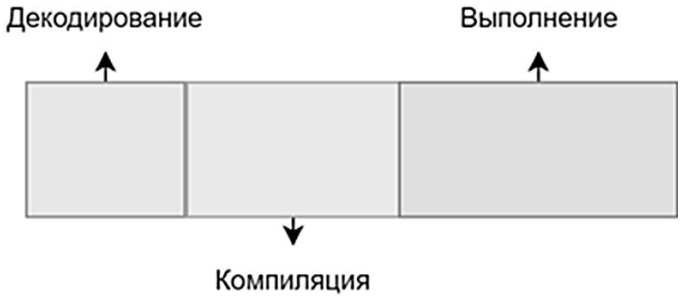

Рис. 4. Операции в процессе выполнения WebAssembly кода

Fig. 4. Operations during WebAssembly code execution

При создании WebAssembly разработчики руководствовались следующими целями [3]:

1. Скорость – код WebAssembly может выполняться практически на естественной скорости на разных платформах, используя преимущества аппаратных возможностей.

2. Читаемость и отладка – WebAssembly – это низкоуровневый язык, который имеет читаемый текстовый формат, позволяющий редактировать и тестировать сгенерированный код.

3. Безопасность – код WebAssembly предназначен для запуска в безопасной, изолированной среде выполнения. Как и другой веб-код, он будет соблюдать политики безопасности браузера.

4. Сохранение текущего веба – технология WebAssembly разработана так, что она сочетается с другими веб-технологиями и поддерживает обратную совместимость.

Так как каждый язык имеет свои стандартные библиотеки, WebAssembly необходимо подключать их для каждого скомпилированного файла. Но технология разработана с учетом формирования бинарных файлов малого размера, что позволяет быстро загружать файлы с исполнимым кодом,

а использование технологии gzip позволит еще сильнее уменьшить итоговый размер. Следовательно, использование технологии в проектах не сильно повлияет на загрузку приложения.

В 2017 г. технология была объявлена готовой к использованию. Была опубликована спецификация на все основные модули, вышла первая стабильная версия технологии [4]. На данный момент она поддерживается всеми популярными браузерами и 88 % среди всех установленных браузеров. Список версий популярных браузеров, которые поддерживают WebAssembly, представлен в табл. 1.

Таблица 1

Table 1

Поддерживающие технологию браузеры

Supporting browsers

Браузер	Chrome	Edge	Safari	Firefox	Opera
Поддерживаемые версии	57–109	16–106	11–16	52–108	44–91

Исключением является Internet Explorer, для него поддержки нет и уже не будет. Предполагалось, что для старых браузеров будет полифил – возможность преобразования WebAssembly в asm.js. Разработка была приостановлена, и продолжение не планируется. Однако данное исключение не является критичным, т. к. компания Microsoft полностью приостановила поддержку Internet Explorer и полностью переключилась на разработку браузера Edge.

Также технология способна компилировать в новый формат уже большую часть основных языков программирования [5]:

– C/C++ – через Emscripten;

– Rust – поддержка экосистемы WebAssembly строится во многом на основе Rust;

– Java – через TeaVM и JWebAssembly;

– Kotlin – есть поддержка в Kotlin/Native через LLVM;

– Go;

– C# – через Blazor и Uno Platform;

– TypeScript – через AssemblyScript.

Тестирование и замеры производительности

Тестирование будет происходить в браузере Google Chrome, т. к. он является наиболее популярным, активно разрабатывается и имеет весь необходимый функционал для работы с WebAssembly и тестирования веб-приложений. Выбранный браузер дает более точное время исполнения кода по сравнению с такими конкурентами, как Mozilla Firefox.

Оборудование для тестирования:

– процессор AMD Ryzen 7 4800 H с тактовой частотой 2,9 ГГц;

– оперативная память – 16 Гб поколения DDR4.

Также тесты будут проходить без дополнительной нагрузки, т. к. в такие моменты браузерные движки начинают подключать внутренние механизмы оптимизации и «прогрева» кода, что может исказить итоговые измерения.

Для замеров производительности будут использованы функции, написанные на языке C, скомпилированные в WebAssebmly модули, и функции на языке JavaScript.

Используемые функции для теста:

– функция перемножения: функция будет принимать 3 аргумента: 2 числа для перемножения и количество повторения операции, с помощью цикла for. По итогу выполнения функция будет возвращать результат перемножения;

– функция быстрой сортировки: функция принимает 3 аргумента: массив чисел для сортировки, начальная и конечная границы массива. Сначала выбирается опорный элемент из массива, далее происходит перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные – после. Затем предыдущие операции рекурсивно применяются к массивам слева и справа от опорного элемента;

– функция суммирования: функция принимает 1 аргумент: массив с числами. Затем происходит суммирование всех элементов массива и возвращение результата из функции;

– функция перемножения векторов: функция принимает в качестве аргументов два вектора – массивы с числами, количество повторений перемножения. Затем с помощью цикла for происходит поочередное перемножение элементов векторов соответствующего индекса и по итогу возвращает результирующий вектор;

– функция поиска числа Фибоначчи: функция принимает 1 аргумент – искомое число. Функция будет рекурсивно вызывать саму себя и передавать в качестве аргумента самый первый переданный аргумент, из которого вычитают 1 и 2. Результаты рекурсивного вызова вычитаются, и в итоге будет возвращено искомое число.

Замер производительности выполняемого кода будет произведен интерфейсом Performance, предоставляемым самим движком браузера. Метод performance.now() возвращает временную метку в миллисекундах [6]. Для измерения времени, затра-

ченного на выполнение кода, необходимо применить метод до и после начала тестируемого фрагмента и вычесть разницу из полученных временных меток (рис. 5).

```
1  const time0 = performance.now()
2
3  const array = []
4
5  for (let i = 0; i < 1000; i++ ) {
6      array.push()
7  }
8
9  const time1 = performance.now()
10 console.log(time1 - time0)
```

Рис. 5. Пример измерения производительности

Fig. 5. Example of measuring performance

Полученное число является затраченным временем в миллисекундах для выполнения тестируемой функции.

Тестирование будет реализовано с помощью специального скрипта, который будет поочередно выполнять тестируемые функции в JavaScript и WebAssembly формате, а затем сохранять время выполнения функции для последующего анализа. Так как в момент тестирования процессор и оперативная память могут быть загружены фоновыми процессами операционной системы, а сам браузер способен неявно оптимизировать исполняемый код, было решено выполнить 100 запусков каждой функции, и результатом тестирования каждой функции будет среднее арифметическое от всех прогонов функции.

Процесс тестирования будет разбит на следующие этапы:

1. Запуск и замер результатов функции на JavaScript.
2. Запуск и замер результатов выполнения функции на WebAssembly.
3. Сравнение результатов.
4. Переход к тестированию следующего набора функций из табл. 2.

Таблица 2

Table 2

Тестовые данные

Test data

Функция	Данные
Перемножение	123456789, 9087654321
Быстрая сортировка	131, 7, 1, 4 , 5, 71, 786, 9, 2, 213, 3, 56, 40, 0
Суммирование	7283412431, 1243521204
Перемножение векторов	Вектор 1: 6, 4, 11, 14, 99, 31 Вектор 2: 3, –2, 10, –10, 1, 4
Число Фибоначчи	79

Результаты тестирования

Итоги замеров производительности представлены в табл. 3 и на рис. 6.

Таблица 3

Table 3

Результаты тестирования

Test results

Функция	JavaScript, мс	WebAssembly, мс
Перемножение	719	188
Быстрая сортировка	429	343
Суммирование	98	105
Перемножение векторов	38	69
Число Фибоначчи	607	246

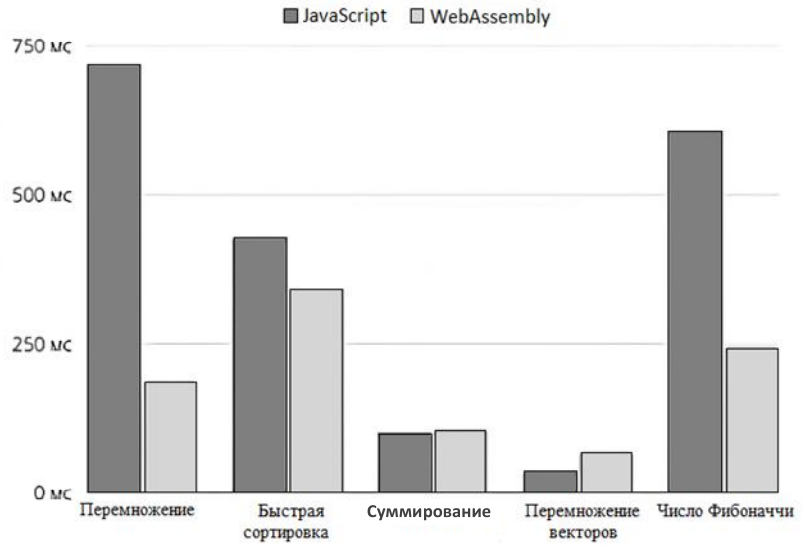

Рис. 6. Сравнение скорости выполнения функций

Fig. 6. Comparison of function executing speed

По итогам тестирования заметно, что WebAssembly быстрее JavaScript в большинстве операций. Однако нельзя делать вывод, что WebAssembly однозначно быстрее JavaScript. Нужно сравнивать каждый отдельный случай, потому что производительность каждой из технологий может получиться и во много раз лучше, и в несколько раз хуже. Также на скорость влияет браузер, в котором выполняется тестируемый код.

WebAssembly показал себя гораздо быстрее в большинстве случаев, чем JavaScript. Однако по результатам тестирования также видно, что технология в некоторых случаях выполнила код с практически такой же или даже худшей скоростью. Это доказывает, что каждый конкретный случай замены JavaScript кода на WebAssembly нужно рассматривать и тестировать отдельно. В операциях с объемными вычислениями технология показала себя гораздо лучше, но когда требуется работа с памятью, то скорость выполнения становится либо на уровне, либо даже хуже JavaScript.

По результатам исследования можно выявить основные области применения технологии:

– обработка видео;
– 3D-рендеринг;
– игры;
– криптографические вычисления;
– AR/VR приложения.

То есть использование технологии в приложениях с большими вычислениями на стороне клиента и с требованием высокой производительности. Использование WebAssembly для разработки статичных сайтов не повлияет значительно на скорость работы приложения, а наоборот усложнит и растянет разработку. Однако благодаря тому, что WebAssembly модули можно подключать к JavaScript коду, можно продолжать использовать привычный JavaScript для работы с динамическим контентом на веб-странице, а все большие вычисления отдавать подключаемым WebAssembly модулям с высокой производительностью.

Заключение

WebAssembly является уникальной технологией, позволяющей использовать максимально понятный и читаемый для машины код, что значительно позволяет увеличить скорость его выполнения. Однако технология не является полной заменой JavaScript, а лучше подходит как инструмент, которому следует отдавать в работу большие и трудные вычисления.

Список источников

1. Компиляция кода C/C++ в WebAssembly. URL: https://developer.mozilla.org/ru/docs/WebAssembly/C_to_wasm (дата обращения: 10.01.2023).

2. Haas A., Rossberg A., Schuff D. L., Titzer B. L., Holman M., Gohman D., Wagner L., Zakai A., Bastien JF. Bringing the Web up to Speed with WebAssembly // PLDI '23: ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017. P. 185–200. URL: https://css.csail.mit.edu/6.858/2022/readings/wasm.pdf (дата обращения: 10.01.2023).

3. Watt C., Rossberg A., Pichon-Pharabod J. Weakening WebAssembly // Proceedings of the ACM on Programming Languages, 2019. P. 1–28. DOI: 10.1145/3360559.

4. WebAssembly: что и как. URL: https://habr.com/ru/post/475778/ (дата обращения: 12.01.2023).

5. Sletten B. WebAssembly: The Definitive Guide: Safe, Fast, and Portable Code. O'Reilly Media, Inc., 2021. 334 p.

6. Selakovic M., Pradel M. Performance issues and optimizations in JavaScript: an empirical study // Proceedings of the 38th International Conference on Software Engineering, 2016. P. 61–72.

References

1. *Kompiliatsiia koda C/C++ v WebAssembly* [Compiling C/C++ Code to WebAssembly]. Available at: https://developer.mozilla.org/ru/docs/WebAssembly/C_to_wasm (accessed: 10.01.2023).

2. Haas A., Rossberg A., Schuff D. L., Titzer B. L., Holman M., Gohman D., Wagner L., Zakai A., Bastien JF. Bringing the Web up to Speed with WebAssembly. *PLDI '23: ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017. Pp. 185-200. Available at: https://css.csail.mit.edu/6.858/2022/readings/wasm.pdf (accessed: 10.01.2023).

3. Watt C., Rossberg A., Pichon-Pharabod J. *Weakening WebAssembly. Proceedings of the ACM on Programming Languages*, 2019. Pp. 1-28. DOI: 10.1145/3360559.

4. *WebAssembly: chto i kak* [WebAssembly: what and how]. Available at: https://habr.com/ru/post/475778/ (accessed: 12.01.2023).

5. Sletten B. *WebAssembly: The Definitive Guide: Safe, Fast, and Portable Code*. O'Reilly Media, Inc., 2021. 334 p.

6. Selakovic M., Pradel M. Performance issues and optimizations in JavaScript: an empirical study. *Proceedings of the 38th International Conference on Software Engineering*, 2016. Pp. 61-72.

Статья поступила в редакцию 07.03.2023; одобрена после рецензирования 28.03.2023; принята к публикации 19.04.2023
The article is submitted 07.03.2023; approved after reviewing 28.03.2023; accepted for publication 19.04.2023

Информация об авторах / Information about the authors

Виолетта Валерьевна Рокотянская – кандидат экономических наук, доцент; доцент кафедры управления; Российский государственный аграрный университет – МСХА имени К. А. Тимирязева; rokotyanskay_v_v@mail.ru

Violetta V. Rokotianskaya – Candidate of Economic Sciences, Assistant Professor; Assistant Professor of the Department of Management; Russian Timiryazev State Agrarian University; rokotyanskay_v_v@mail.ru

Вадим Сергеевич Абрамов – магистрант факультета программной инженерии и компьютерной техники; Национальный исследовательский университет ИТМО; vadim-abramov-00@mail.ru

Vadim S. Abramov – Master's Course Student of the Department of Software Engineering and Computer Engineering; ITMO University; vadim-abramov-00@mail.ru