

УДК 004.8

ПРИМЕНЕНИЕ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LARGE LANGUAGE MODELS) В МЕТОДОЛОГИИ DEVOPS

Лелейкин С.С.

*Образовательная автономная некоммерческая организация высшего образования
«Московский технологический институт», Москва, Российская Федерация,
e-mail: ssoft@mail.ru*

Настоящая статья посвящена изучению роли больших языковых моделей (Large Language Models) в современном подходе DevOps, направленном на обеспечение эффективной организации совместной работы команд разработчиков и администраторов. За последние годы крупные достижения в области искусственного интеллекта привели к созданию мощных моделей, способных обрабатывать естественный язык и эффективно решать разнообразные задачи. Такие модели представляют собой инструмент, способствующий увеличению продуктивности труда и уменьшению временных затрат на выполнение рутинных действий. Цель настоящего исследования заключается в определении наиболее перспективных областей применения больших языковых моделей в процессах DevOps и оценке влияния их внедрения на повышение производительности команды. Автор исследует перспективы внедрения больших языковых моделей в различных аспектах методологии DevOps, рассматривая конкретные примеры использования моделей в таких важных операциях, как помощь в написании и комментировании кода, дебаге и анализе логов приложения, конфигурации, освоении новых технологий, рефакторинге проекта, выборе архитектурного решения, подсчете технических ресурсов и документировании. Основное внимание уделено возможностям увеличения производительности труда разработчиков и оптимизации рабочих процессов за счет использования интеллектуального ассистента на базе больших языковых моделей. Вместе с тем автор подчеркивает наличие определенных рисков и ограничений, связанных с качеством результатов, безопасностью обрабатываемых данных и необходимостью дальнейшей адаптации моделей к специфическим требованиям каждого отдельного проекта. Приводятся выводы о том, что грамотное использование больших языковых моделей может оказать положительное влияние на организацию процессов DevOps, помогая компаниям достигать целей по повышению гибкости и устойчивости бизнес-процессов. Таким образом, исследование посвящено актуальным проблемам современного DevOps и предлагает практические рекомендации по расширению арсенала инструментов инженера посредством внедрения больших языковых моделей.

Ключевые слова: DevOps, большие языковые модели, автоматизация, интеграция, поставка

APPLICATION OF LARGE LANGUAGE MODELS IN DEVOPS METHODOLOGY

Leleykin S.S.

*Educational Autonomous Non-Commercial Organization of Higher Education
“Moscow Technological Institute”, Moscow, Russian Federation, e-mail: ssoft@mail.ru*

This article is devoted to the study of the role of Large Language Models in the modern DevOps approach aimed at ensuring effective collaboration between development teams and administrators. In recent years, major advances in artificial intelligence have led to the creation of powerful models capable of processing natural language and effectively solving a variety of tasks. Such models are a tool that helps to increase labor productivity and reduce the time required to perform routine actions. The purpose of this study is to identify the most promising areas of application of large language models in DevOps processes and to assess the impact of their implementation on increasing team productivity. The author explores the prospects for implementing large language models in various aspects of the DevOps methodology, considering specific examples of using models in such important operations as assistance in writing and commenting code, debugging and analyzing application logs, configuration, mastering new technologies, refactoring a project, choosing an architectural solution, calculating technical resources and documenting. The main focus is on the possibilities of increasing developer productivity and optimizing workflows through the use of an intelligent assistant based on large language models. At the same time, the author emphasizes the presence of certain risks and limitations related to the quality of the results, the security of the processed data and the need for further adaptation of models to the specific requirements of each individual project. It is concluded that the competent use of large language models can have a positive impact on the organization of DevOps processes, helping companies achieve goals to increase the flexibility and sustainability of business processes. Thus, the study is devoted to the current problems of modern DevOps and offers practical recommendations for expanding the arsenal of engineer's tools through the introduction of large language models.

Keywords: DevOps, large language models, automation, integration, delivery

Введение

Методология DevOps способствует ускорению доставки изменений в производственные среды через непрерывную интеграцию (continuous integration, далее по тексту CI), непрерывное развертывание (continuous de-

livery, далее по тексту CD) и автоматизацию инфраструктурных процессов.

Проблематика, решаемая данным исследованием: низкий уровень описания потенциальных точек внедрения больших языковых моделей (large language models,

далее по тексту будет использовано сокращение LLM) в методологию DevOps, блокирующий дополнительные возможности для оптимизации процедур разработки и эксплуатации.

Цель настоящего исследования заключается в определении наиболее перспективных областей применения больших языковых моделей (LLM) в процессах DevOps и оценке влияния их внедрения на повышение продуктивности команды разработчиков и администраторов.

Основные задачи исследования:

- определение потенциала LLM в поддержке основных этапов жизненного цикла разработки, тестирования и поставки ПО;
- оценка преимуществ и недостатков использования LLM в конкретных подходах DevOps;
- выявление факторов, влияющих на успешность внедрения LLM в процессы DevOps;
- формулирование практических рекомендаций по эффективному применению LLM в реальных проектах.

Материалы и методы исследования

Исследование проводилось в компактной DevOps [1] команде (1 Senior и 2 Middle инженера). В качестве основной модели использовалась бесплатная версия GigaChat, а также другие свободно доступные бесплатные версии LLM. Оценка эффективности велась по agile-метрикам [2], с фокусом на трудозатраты, метрики качества выполнения и скорость решения задач.

В результате исследования были получены и сопоставлены метрики выполнения

схожих по сложности задач сотрудниками, применяющими LLM и не применяющими. Замер контроля качества производился лидером компетенции. Трудозатраты оценивались всей командой по методологии Scrum [2] во избежание различия в сложности выполнения сравниваемых задач. Время выполнения задач формировалось на основе выгрузки статистики из Jira.

Рассматривались следующие направления работы DevOps инженера: генерация и рефакторинг кода, комментирование и документирование, анализ логов и отладка, конфигурация инфраструктуры, оценка ресурсов и проектирование архитектуры, освоение новых технологий. Для исследования использовались следующие методы: наблюдение, эксперимент, измерение.

Результаты исследования и их обсуждение

Помощь в написании кода. Автоматическая генерация фрагментов кода – одна из важнейших функций LLM. Она ускоряет разработку прототипов, реализацию функций и модернизацию существующих компонентов, сокращая время написания кода на 20–80 %. Основная проблема – сложность оценки качества генерируемого кода инженерами уровня Junior, что требует дополнительной проверки опытными коллегами (уровня Middle, Senior).

Анализ данных, представленных в табл. 1, показывает, что за более короткий период времени LLM выдала код, аналогичный написанному Senior DevOps инженером. Оба результата приемлемы и выполняют поставленную задачу.

Таблица 1

Примеры написанного кода по задаче «Необходимо открыть файл test.txt и распечатать все строки, содержащие FLAG» на языке программирования Python

Исполнитель	Senior DevOps инженер	LLM
Задача/промпт	Необходимо открыть файл test.txt и распечатать все строки, содержащие FLAG	Ты – профессиональный Senior DevOps. Необходимо открыть файл test.txt и распечатать все строки, содержащие FLAG. Используй Python
Результат	file = open('test.txt') for line in file: if 'FLAG' in line: print(line.strip())	# Открываем файл test.txt в режиме чтения with open('test.txt', 'r') as file: # Читаем все строки файла lines = file.readlines() # Проходим по каждой строке и проверяем наличие слова 'FLAG' for line in lines: if 'FLAG' in line: print(line.strip()) # Печать строки без символа новой строки
Время	2 минуты	20 секунд

Примечание: составлена автором на основе ответа LLM и программного кода собственной разработки; промпт [3] составлен автором.

Помощь в комментировании. Комментирование кода улучшает его прозрачность и читаемость. Исследование показало, что LLM повышают покрытие кода комментариями на 20–40 %, детально описывая логику и учитывая контекст. Однако требуется контроль инженера, хорошо погруженного в контекст проекта, так как комментарии модели могут требовать коррекции. Качество генерации улучшается при доступе к широкому контексту

всего проекта и грамотно сформулированных промптах.

С примером комментирования собственного кода LLM можно ознакомиться в табл. 1.

Пример комментирования кода по промпту «Ты – Senior Java Developer. Добавь комментарии в код» (как контекст был передан код интеграционного теста, написанный автором на Java с использованием фреймворка SpringBoot) [4]:

```
// Интеграционный тест для HelloController с полной загрузкой контекста Spring Boot
@SpringBootTest
// Включает автоматическое создание экземпляра MockMvc для тестирования веб-эндпоинтов
@AutoConfigureMockMvc
public class HelloControllerTest {
    // Автоматически внедряется экземпляр MockMvc для выполнения тестов
    @Autowired
    private MockMvc mvc;
    // Точка входа в тестирование метода getHello контроллера
    @Test
    public void getHello() throws Exception {
        // Отправляем GET-запрос на корень «/» с ожиданием JSON-ответа
        mvc.perform(MockMvcRequestBuilders.get("/"))
            .accept(MediaType.APPLICATION_JSON)) // Устанавливаем Accept-заголовок для JSON
            .andExpect(status().isOk()) // Проверяем, что статус ответа равен 200 OK
            .andExpect(content().string(equalTo("Greetings from Spring Boot!"))); // Проверяем,
        что тело ответа соответствует ожидаемому значению
    }
}
```

Представленный фрагмент демонстрирует, как LLM добавила комментарии без отрыва от контекста всего отрывка кода интеграционного теста.

Помощь в дебаге / Анализ логов приложения. Процесс обнаружения и исправления ошибок в коде приложений и скриптов занимает значительную долю рабочего времени разработчиков и инженеров DevOps. Использование LLM помогает упростить этап локализации проблемы и оперативней найти возможные пути ее решения.

Помощь со стороны LLM в анализе причин неисправности сокращает время устранения в несколько раз. Ответ модели, даже в случае отсутствия в нем явной причины возникновения проблемы, позволяет акцентировать внимание инженера на важной информации, которая, в свою очередь, может сыграть решающую роль в локализации и устранении неисправности.

В табл. 2 представлен пример результата анализа лога ошибки с помощью LLM.

Практика показывает, что в случае возникновения подобной ошибки время решения инженером уровня Junior без помощи более квалифицированного коллеги может составить до 1 ч. Столь длительный поиск

причин ошибки связан с неоднозначностью трактования сообщения в логе «partially initialized module 'requests' has no attribute 'get'», что, в свою очередь, наводит инженера на неправильный путь решения (поиск потерянного атрибута get, проверка кода на правильность, повышение уровня логирования и т.д.).

LLM в данном случае выявила зависимость между названием запускаемого файла и названием стандартной библиотеки Python, что и являлось причиной возникновения ошибки, а также дала рекомендации, как избежать повторения подобных проблем.

Помощь в конфигурации. Конфигурация серверов и сервисов представляет собой трудоемкий и подверженный ошибкам процесс [5]. Использование LLM способно сократить число ошибок, возникающих вследствие применения некорректных настроек. В ходе исследования было проведено сравнение методов конфигурации человеком без и с использованием модели GigaChat, проведена оценка корректности конфигураций. Доля правильных конфигураций на стендах разработки увеличилась на 40 % (> 20 % – только за счет отсутствия орографических и синтаксических ошибок).

Таблица 2

Пример анализа лога ошибки, с которой часто встречаются новички при запуске приложения, написанного на Python

Исходный код	Файл requests.py import requests url = 'https://httpbin.org/get' response = requests.get(url)
Ошибка	Traceback (most recent call last): File “/home/in2thevoid/Документы/Научная деятельность/Применение LLM в методологии DevOps/requests.py”, line 1, in <module> import requests File “/home/in2thevoid/Документы/Научная деятельность/Применение LLM в методологии DevOps/requests.py», line 4, in <module> response = requests.get(url) ^^^^^^^^^ AttributeError: partially initialized module ‘requests’ has no attribute ‘get’ (most likely due to a circular import)
Промпт	Ты – профессиональный Python разработчик. Найди причину ошибки
Ответ LLM	Проблема вызвана конфликтом имен модулей. У вас имеется файл с именем requests.py, который совпадает с названием популярной сторонней библиотеки requests. Из-за этого возникает циклический импорт. Решение: Переименовать файл requests.py в новый уникальный файл (например, my_requests.py) Рекомендация: Лучше избегать совпадения собственных файлов с названиями популярных пакетов Python, чтобы избежать подобных конфликтов.
Время	10 секунд

Примечание: составлена автором на основе ответа LLM и программного кода собственной разработки; промпт составлен автором

Таблица 3

Пример конфигурации ingress и service для маршрутизации трафика внутри кластера k8s

Промпт	Ты – Senior DevOps Engineer. Напиши конфигурацию k8s ingress и service, распределяющие трафик на 8 разных сервисов и покрывающие соединение ssl
Результат	Файл конфигурации ingress.yaml apiVersion: networking.k8s.io/v1 kind: Ingress ... rules: - host: example.com http: paths: - path: /svc1(/ \$)(.*) pathType: Prefix backend: service: name: svc1 port: number: 8080 ... Пример Service ресурса для сервиса svc1 (остальные сервисы аналогичны) apiVersion: v1 kind: Service ... ports: - protocol: TCP port: 8080 targetPort: 8080 type: ClusterIP
Время	< 1 минуты

Примечание: составлена автором на основе ответа LLM; промпт составлен автором

Таблица 4

Пример очень глубокого рефакторинга кода, написанного на языке программирования Python, выполняющего сортировку «пузырьком»

Промпт	Ты – профессиональный Python разработчик. Проведи рефакторинг и расставь комментарии в коде. Сделай код масштабируемым, вынеси нужные блоки в классы. Классы кратко прокомментируй
Код	<pre>from random import randint N = 10 a = [] for i in range(N): a.append(randint(1, 99)) print(a) for i in range(N-1): for j in range(N-1-i): if a[j] > a[j+1]: a[j], a[j+1] = a[j+1], a[j] print(a)</pre>
Код после рефакторинга	<pre>import random class RandomArrayGenerator: """Класс для генерации случайного массива""" def __init__(self, size=10, min_value=1, max_value=99): """ :param size: размер массива :param min_value: минимальное значение элемента :param max_value: максимальное значение элемента """ self.size = size self.min_value = min_value self.max_value = max_value def generate(self): """Генерация случайного массива""" return [random.randint(self.min_value, self.max_value) for _ in range(self.size)]</pre>
	<pre>class BubbleSorter: """Класс для сортировки массива методом пузырьковой сортировки""" def sort(self, array): """ Сортируем массив методом пузырьковой сортировки :param array: массив для сортировки :return: отсортированный массив """ n = len(array) for i in range(n - 1): for j in range(n - 1 - i): if array[j] > array[j + 1]: array[j], array[j + 1] = array[j + 1], array[j] return array</pre>
Время	< 1 минуты

Примечание: составлена автором на основе ответа LLM и программного кода собственной разработки; промпт составлен автором

Полная автоматизация процесса пока невозможна ввиду сложности требований и необходимости контроля со стороны персонала. Поскольку от скорости конфигурации стендов разработки и тестирования напрямую зависит скорость конфигурации промышленных стендов, можно выявить прямую зависимость прироста скорости вывода новой функциональности и предоставления ее конечным пользователям (что, по факту, является уменьшением бизнес-метрики *time-to-market* [6]). В табл. 3 представлен пример конфигурации сети кластера k8s [7], составленной LLM.

Помощь в освоении технологий. Частое обновление техстека и версий ПО увеличивает нагрузку на команду DevOps, стимулируя обучение. LLM могут помочь, давая советы и подсказки. Однако неправильное использование LLM может усложнить процесс обучения: отсутствие нужного контекста ведет к выдаче некорректных рекомендаций. Например, запросы о нетипичных или новых сочетаниях технологий, таких как Gradle 8.8 с Kotlin [8], часто приводят к выдаче некорректных ответов. Для митигации последствий рисков некачественного обучения рекомендуется использование микротестирования [9], изучение официальных курсов и документации в сочетании с возможностями LLM (объяснения кода, его доработка или краткий пересказ статей и параграфов документации).

Помощь в рефакторинге. Рефакторинг [10] существующего кода является важной частью поддержания высокого качества программного продукта (в контексте DevOps – пайплайна и настроек инфра-

структур). Использование LLM позволяет автоматизировать создание небольших исправлений и оптимизаций, снижая нагрузку на команду. Также LLM позволяет полностью переписать большие объемы кода, но требует контроля и тестирования со стороны команды. Правильно составленный промпт позволяет существенно улучшить уровень рефакторинга LLM, пример показан в табл. 4.

В контексте данной статьи автор не будет углубляться в качество кода и его масштабируемость. Пример выше показывает, что во время рефакторинга [10] код не обязательно уменьшится в объеме, но, при правильно составленном промпте, может стать более масштабируемым/быстрым/экономным к ресурсам и т.д. В конкретном случае LLM была поставлена задача перевести код в парадигму объектно-ориентированного программирования, с чем она успешно справилась.

Выбор архитектурного решения. При выборе архитектурных решений должны быть учтены ключевые факторы: бизнес-требования, технические характеристики, бюджет и сроки реализации [11]. LLM помогают принять оптимальное решение, предлагая рекомендации на основе большого количества данных. Передача на вход LLM контекста инфраструктуры, языков программирования, протоколов и бизнес-требований позволяет ей строить схемы концептуальной и детальной архитектуры и предлагать несколько альтернативных вариантов. В DevOps данная функциональность может быть применена для выбора инфраструктурных решений, проектирования пайплайнов [12] и каналов передачи данных [13].

Таблица 5

Подсчет ресурсов приложения и кластера k8s

Промпт	Ты – Senior Solution Architect. Посчитай ресурсы (сри, ram, реплики), необходимые сервису, написанному на rust, использующему базу данных postgres и отвечающему на 100000 запросов в секунду. Сервис расположен в k8s. Также посчитай ресурсы кластера k8s и базы данных postgres. Запас по нагрузке – 30 %
Результат	Окончательные оценки ресурсов: Rust-сервис CPU130 RAM1.3 GB, Реплика Rust-сервиса CPU43 RAM 0.43 GB, База данных Postgres CPU4, RAM 16 GB Исходя из этих оценок, минимальная конфигурация узлов k8s-кластера может выглядеть следующим образом: Количество реплик Rust-сервиса: 3 Ресурсы узла k8s-кластера: CPU: 150 (для обслуживания всех реплик и самой базы данных) RAM: 20 GB (суммарно для трех реплик и одной базы данных). Это лишь ориентировочные расчеты, реальная потребность может варьироваться в зависимости от архитектуры сервиса, характеристик оборудования и инфраструктуры. Рекомендуется провести стресс-тестирование и профилирование перед внедрением.
Время	30 секунд

Примечание: составлена автором на основе ответа LLM; промпт составлен автором

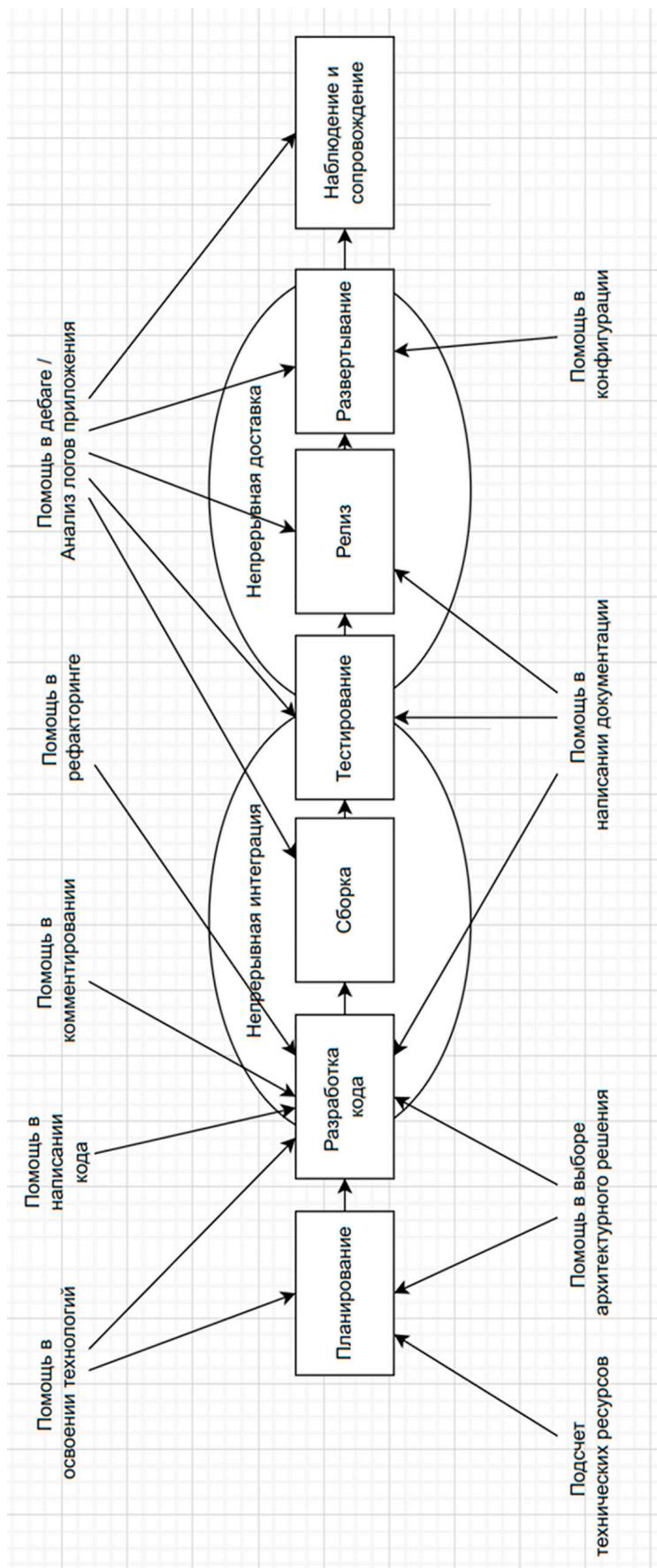


Схема точек потенциально целесообразного внедрения ИТМ в этапы процесса методологии DevOps
 Примечание: составлен автором на основе проведенного исследования, собранных данных и сделанных выводов

Подсчет технических ресурсов. Оценка вычислительных ресурсов – важная задача DevOps на начальном этапе разработки. LLM эффективно помогает в расчете потребности, используя исторические статистические данные и прогнозируя потенциальные нагрузки. Поскольку нагрузочное тестирование [14] на ранних этапах разработки приложения, как правило, отсутствует, расчет носит предварительный характер. Точность расчета LLM на данном этапе не критична, так как дальнейшая коррекция осуществляется по результатам реального НТ. Пример расчета ресурсов, сделанного LLM, представлен в табл. 5. Помощь в написании документации. Документация является неотъемлемой частью любого проекта, обеспечивающей передачу знаний внутри команды. LLM способны быстро формировать документацию на основании кода и архитектуры. Проведенное исследование подтверждает целесообразность написания документации с помощью LLM, но только при контроле результата командой. Языковые модели хорошо описывают не только свой код, но и код, переданный в контексте, отлично могут описать часть «легаси» кода [15], сокращая время на его исследование, а также хорошо обогащают уже существующую документацию примерами и уточнениями.

Как итог, автором составлена схема точек целесообразного внедрения языковых моделей в этапы процесса методологии DevOps. Схема представлена на рисунке.

Заключение

Проведенное исследование выявило значительный потенциал больших языковых моделей (LLM) в повышении эффективности практик DevOps. Получены положительные результаты во всех выбранных направлениях. Результатом стали снижение среднего времени выполнения задач, повышение качества артефактов и автоматизация рутинных задач. Также выявлены и риски, связанные с точностью рекомендаций и корректностью выдачи ответов. Таким образом, можно сделать вывод о том, что внедрение LLM в методологию DevOps является целесообразным и эффективным и позволяет добиться заметной оптимизации работы команды, а следовательно, и сокращения time-to-market и количества инцидентов, что на дистанции использования положительно повлияет на финансовый эффект как DevOps подразделения, так и программного продукта в целом.

Конфликт интересов: Авторы заявляют об отсутствии конфликта интересов.

Conflict of interest: The authors declare that there is no conflict of interest.

Список литературы

1. Пеленицына П.А. DEVOPS как взаимодействие между разработкой и эксплуатацией // Издательство. 2024. № 21–4 (338). С. 14–16. URL: <https://internauka.org/journal/science/internauka/338> (дата обращения: 12.10.2025).
2. Яковлева М.В., Лысенко М.В., Овсянникова У.С. Концепция Agile: возможность применения гибких методологий в производственных отраслях промышленности // Вестник качества. 2023. № 3. С. 207–217. URL: <http://www.agequal.ru/pdf/2023/323009.pdf> (дата обращения: 12.10.2025).
3. Aadi Juthani Optimizing ILM strategies for playing mend-ikot using prompt engineering // International Journal For Multidisciplinary Research. 2024. Т. 6. № 6. URL: <https://www.ijfmr.com/papers/2024/6/30130.pdf> (дата обращения: 22.10.2025).
4. Ибраимов Р.И., Джемалетдинов А.Б., Шевченко А.А. Spring boot: создание тестов для spring mvc контроллеров // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2017. № 4 (18). С. 104–111. EDN: VQLFBC.
5. Патрикеев А.В. Разработка отказоустойчивой конфигурации серверной инфраструктуры // Научный электронный журнал Меридиан. 2019. № 12 (30). С. 312–314. URL: <https://meridian-journal.ru/site/article1fl-2/> (дата обращения: 22.10.2025).
6. Mishra A., Otaiwi Z. Devops and software quality: a systematic mapping // Computer Science Review. 2020. Т. 38. С. 100308. URL: <https://www.sciencedirect.com/science/article/pii/S1574013720304081> (дата обращения: 14.10.2025).
7. Ковтун Д.П., Лапонина О.Р. Использование управляемого доступом на основе атрибутов и mTLS в микросервисной архитектуре // International Journal of Open Information Technologies. 2025. Т. 13. № 6. URL: <https://istina.msu.ru/publications/article/791271027/> (дата обращения: 14.10.2025).
8. Луценко Д.Ю. Сборка (CI / CD) проектов, не использующих JVM с помощью GRADLE / KOTLIN // Информационные технологии в управлении и экономике. 2021. № 2 (23). С. 61–67. URL: <https://itue.ru/Issue/Article/155> (дата обращения: 14.10.2025).
9. Бугаева А.А., Денисенко В.В. Процесс тестирования, методы и типы тестирования программного обеспечения // Синергия Наук. 2022. № 72. С. 92–102. EDN: WPKFPG.
10. Дерюгин Д.С. Рефакторинг, как метод обеспечения качества продукта // Синергия Наук. 2020. № 43. С. 409–415. EDN: MASOU.
11. Magomedova Z. Enterprise architecture framework (TOGAF) // Экономика и социум. 2018. № 10 (53). С. 715–717. URL: https://www.iupr.ru/files/ugd/b06fdc_90ab2c89da0840b4b82cacb0b2cc7739.pdf?index=true (дата обращения: 14.10.2025).
12. Priyanka M., Sindhuja K., Madhuvani V., Prasoona Sowpithika K., Kranthi Kumar K. DevOps optimized navigation: building a DevOps CI/CD pipeline // EPRA International Journal of Research & Development. 2024. С. 376–382. URL: <https://eprajournals.com/IJSR/article/12653> (дата обращения: 22.10.2025).
13. Абдраимова А.С. Оценка и контроль качества канала передачи данных с помощью его характеристик // Проблемы науки. 2017. Т. 2. № 5 (18). С. 19–24. URL: <https://scienceproblems.ru/images/PDF/2017/18/pn-5-18--2.pdf> (дата обращения: 22.10.2025).
14. Лебедев В.С., Михаэлис В.В. Обзор программных инструментов для проведения нагрузочного тестирования информационных систем // Информационные технологии и математическое моделирование в управлении сложными системами. 2021. № 2 (10). С. 40–45. DOI: 10.26731/2658-3704.2021.2(10).40-45. EDN: KKZECX.
15. Duvvuri V. Overcoming the challenges of legacy code in a modern IT landscape // International Journal of Science and Research. 2021. Т. 10. № 12. С. 1500–1502. URL: <https://www.ijsr.net/archive/v10i12/SR24517162432.pdf> (дата обращения: 27.10.2025).