Научная статья
УДК 005.334:004.056.5
https://doi.org/10.17586/2713-1874-2026-1-89-100

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ УПРАВЛЕНИЯ ОРГАНИЗАЦИОННЫМИ СИСТЕМАМИ НЕПРЕРЫВНОЙ РАЗРАБОТКИ НА ОСНОВЕ ПРИОРИТИЗАЦИИ СОБЫТИЙ СТАТИЧЕСКОГО АНАЛИЗА

Антон Юрьевич Харитонов[1]✉, Даниил Вадимович Дорошенко[2]

[1,2]Университет ИТМО, Санкт-Петербург, Россия
[1]akharitonov@itmo.ru✉, https://orcid.org/0000-0002-8826-8583
[2]dandoroshenko@mail.ru, https://orcid.org/0009-0001-7673-4284
Язык статьи – русский

Аннотация: Статья посвящена задаче управления потоком событий, формируемых средствами статического анализа исходного кода, в CI/CD-процессах организации. Показано, что высокая доля нерелевантных срабатываний приводит к росту операционной нагрузки, снижению эффективности распределения ресурсов и ухудшению управляемости процессов разработки.

На основе анализа эмпирических данных и современных подходов к управлению организационными системами предложена классификация причин нерелевантных срабатываний и разработана концептуальная модель многоуровневой приоритизации событий, интегрируемая в CI/CD-контур.

Предлагаемая модель рассматривает результаты статического анализа как входящий информационный поток, подлежащий фильтрации и скорингу с учётом контекста выполнения, достижимости кода, характеристик среды эксплуатации и критичности компонентов. Такой подход позволяет перейти от инструментальной обработки результатов анализа к управлению процессом принятия решений на уровне организационной системы, обеспечивая более рациональное использование ресурсов команд разработки.

Сценарная оценка демонстрирует, что внедрение дополнительного слоя интеллектуальной обработки событий позволяет существенно сократить объём нерелевантных сообщений и трудозатраты на их анализ без ухудшения качества принимаемых решений. Работа рассматривает CI/CD-процесс как управляемую социотехническую систему, в которой приоритизация событий интерпретируется как задача организационной оптимизации.

Ключевые слова: организационные системы, поддержка принятия решений, приоритизация событий, статический анализ кода, управление информационными потоками, CI/CD, DevSecOps

Ссылка для цитирования: Харитонов А. Ю., Дорошенко Д. В. Повышение эффективности управления организационными системами непрерывной разработки на основе приоритизации событий статического анализа // Экономика. Право. Инновации. – 2026. – Т. 14. – № 1. – С. 89–100. – https://doi.org/10.17586/2713-1874-2026-1-89-100

IMPROVING THE EFFICIENCY OF MANAGING ORGANIZATIONAL CONTINUOUS DEVELOPMENT SYSTEMS BASED ON PRIORITIZATION OF STATIC ANALYSIS EVENTS

Anton Yu. Kharitonov[1]✉, Daniil V. Doroshenko[2]

[1,2]ITMO University, Saint Petersburg, Russia
[1]akharitonov@itmo.ru✉, https://orcid.org/0000-0002-8826-8583
[2]dandoroshenko@mail.ru, https://orcid.org/0009-0001-7673-4284
Article in Russian

Abstract: This paper addresses the problem of managing the flow of events generated by static source code analysis tools within organizational CI/CD processes. It is shown that a high proportion of irrelevant alerts leads to increased operational workload, reduced efficiency of resource allocation, and decreased controllability of software development processes. Based on the analysis of empirical data and contemporary approaches to organizational systems management, a classification of the causes of irrelevant alerts is proposed, and a conceptual multi-level prioritization model designed for integration into the CI/CD pipeline is developed.

The proposed model treats static analysis results as an incoming information stream subject to filtering and scoring, taking into account execution context, code reachability, characteristics of the runtime environment, and component

criticality. This approach enables a transition from purely instrumental processing of analysis results to decision-making management at the level of the organizational system, ensuring more efficient use of development team resources.

A scenario-based evaluation demonstrates that introducing an additional intelligent event processing layer can significantly reduce the volume of irrelevant alerts and the associated manual effort without degrading decision quality. The CI/CD process is considered as a managed socio-technical system in which event prioritization is framed as a problem of organizational optimization.

Keywords: organizational systems, decision support, event prioritization, static code analysis, information flow management, CI/CD, DevSecOps

For citation: Kharitonov A. Yu., Doroshenko D. V. Improving the Efficiency of Managing Organizational Continuous Development Systems Based on Prioritization of Static Analysis Events. *Ekonomika. Pravo. Innovacii.* 2026. Vol. 14. No. 1. pp. 89–100. (In Russ.). https://doi.org/10.17586/2713-1874-2026-1-89-100

Введение. В последние годы DevOps стал стандартом для быстрой и надёжной поставки программного обеспечения. DevOps как подход представляет собой трансформацию процессов разработки, ускоряющий выход продукта на рынок, что позволяет быстрее реагировать на потребности пользователя и внедрять новые функции.

Однако высокий темп внедрения не всегда позволяет уделять должное внимание качеству программных компонентов, что приводит к накоплению потенциально проблемных участков кода. Так, в сентябре 2025 года был зафиксирован инцидент Shai-Hulud, представлявший собой самораспространяющийся npm-пакет с вредоносным postinstall-скриптом, осуществлявший компрометацию токенов и автоматически использовавший их для публикации новых версий. Критическую роль в быстром распространении сыграли CI/CD-процессы, автоматические публикации позволили оперативно выпускать новые инфицированные версии пакетов [1].

Также современное ПО всё активнее используют open-source-компоненты, что несёт за собой серьёзные риски. Согласно отчёту Black Duck Open Source Security and Risk Analysis (OSSRA) за 2025 год, 86 % коммерческих кодовых баз содержат уязвимые open-source-библиотеки [2].

Все эти факторы – высокая скорость разработки и поставки, активное использование open-source-компонентов, сложность контроля качества программных компонентов в условиях CI/CD указывают на необходимость интеграции механизмов контроля качества непосредственно в процесс разработки.

Часто в компаниях выделяют отдельный этап проверки, однако опыт показывает, что контроль качества – это не отдельный этап, а комплексный процесс. Из рисунка 1 видно, что чем раньше выявляется дефект в программном обеспечении, тем проще и дешевле его устранить. Поскольку рассматриваемые проблемные участки кода являются частным случаем дефекта, их раннее обнаружение также снижает стоимость их исправления. Одним из ключевых паттернов DevSecOps является Shift Left, предполагающий анализ на ранних этапах CI/CD-пайплайна. В ответ на эти вызовы всё больше организаций внедряют расширенные CI/CD-пайплайны, где элементы анализа встроены в каждый этап разработки и поставки ПО, что позволяет выявлять и устранять проблемные участки ещё до того, как код попадёт в продакшн.

Описание проблемы. Интеграция статического анализа (SAST) в CI/CD-пайплайны часто приводит к большому количеству ложных срабатываний. Это снижает доверие разработчиков к инструментам безопасности и замедляет разработку. Исследование компании Ghost Security наглядно это демонстрирует, при анализе почти 3000 open-source проектов на Go, Python и PHP из 2116 обнаруженных уязвимостей, 1936 (91%) оказались ложными. Настоящими были лишь 180. Ситуация с Python/Flask оказалась наиболее критичной – 99,5% предупреждений об инъекциях команд операционной системы были ложными предупреждениями [3].

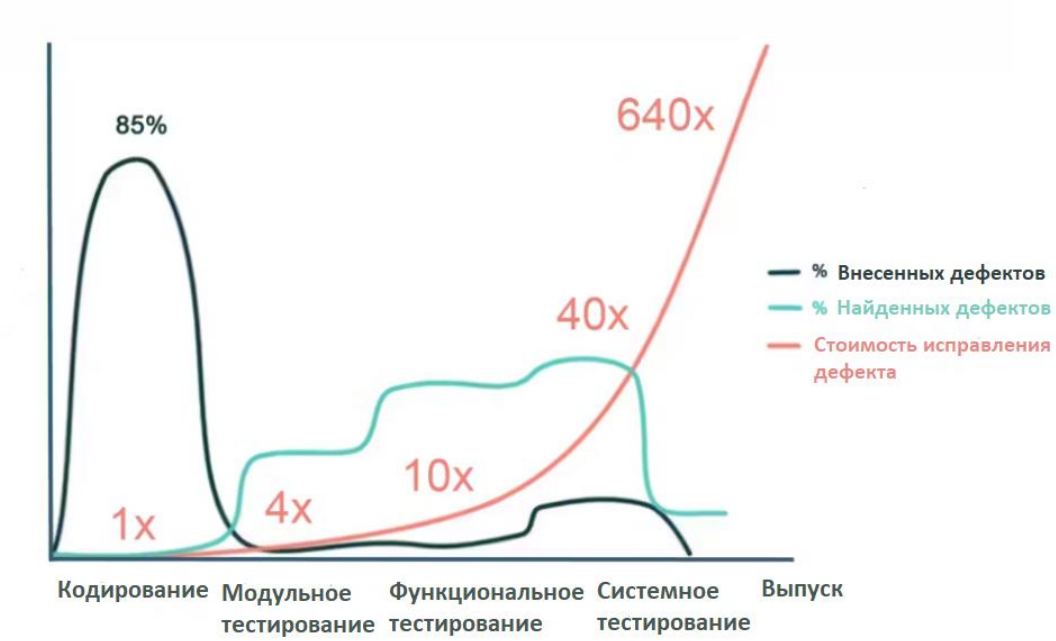

Рисунок 1 – Стоимость исправления дефектов на разных стадиях SDLC
Источник: составлен авторами

Даже если абстрагироваться от объема ложных срабатываний, перед командой встает проблема приоритизации. К примеру, у команды есть возможность проверить все предупреждения. Традиционный подход опирается на систему оценки CVSS (Common Vulnerability Scoring System). Однако данные CISA и проекта EPSS (Exploit Prediction Scoring System) свидетельствуют, что оценка CVSS сама по себе не является надежным индикатором реального риска, уязвимость с баллом 7.0 может представлять как высокий, так и низкий риск в зависимости от контекста [4].

В серии отчётов Kenna Security показано, что между CVSS-уровнем и реальной угрозой системе существует огромный разрыв. Исследование подтверждает, что эксплуатируются менее 20% из high/critical уязвимостей, а большинство опасных эксплойтов связаны с меньшей частью CVEs, часто имеющей не самый высокий CVSS [5]. Стандарт CVSS не учитывает контекст конкретного продукта, доступность уязвимости из интернета, наличие работающих эксплойтов и критичность защищаемых данных. В результате команды тратят ресурсы на исправление формально опасных уязвимостей, которые не могут быть использованы злоумышленником в данной системе, тогда как реальные угрозы остаются без внимания.

Типичный сценарий для npm-проекта с 50 зависимостями иллюстрирует эту проблему, сканирование выявляет 80 уязвимостей, из которых 35 имеют CVSS ≥ 7.0 и требуют «срочного исправления». Однако согласно каталогу эксплуатируемых уязвимостей CISA KEV (Known Exploited Vulnerabilities), реально эксплуатируемых из них всего 2–3 [6]. В результате команда тратит от одной до двух недель на обновление пакетов, не представляющих реальной угрозы, в то время как действительно опасные уязвимости могут остаться незамеченными на фоне информационного шума.

Таким образом, инструменты помечают сотни критических уязвимостей, тогда как реально опасными являются лишь единицы. Это создает парадокс, когда всё помечено как срочное, ни одна проблема не воспринимается таковой.

Инструменты статического анализа ставят разработчиков перед дилеммой, либо игнорировать все предупреждения, рискуя пропустить реальную угрозу, либо заниматься трудоемкой ручной обработкой тысяч ложных срабатываний.

Постоянный поток избыточных предупреждений от систем безопасности формирует у специалистов состояние «усталости от алертов», при котором наступает профессиональное выгорание и критически снижается

способность адекватно реагировать на реальные инциденты [7].

Литературный обзор. В современных исследованиях DevSecOps рассматривается как подход, предполагающий интеграцию механизмов безопасности в процессы разработки и эксплуатации программного обеспечения на всех этапах жизненного цикла, включая CI/CD-контур, что позволяет переносить выявление уязвимостей на более ранние стадии разработки и сопровождения программных систем [8]. Значительное место в литературе занимает анализ средств статического тестирования безопасности приложений как одного из базовых инструментов раннего обнаружения потенциальных дефектов безопасности. Современные систематические обзоры показывают, что SAST-инструменты играют важную роль в процессах безопасной разработки, однако их практическое применение сопровождается проблемами оценки качества, ограниченностью существующих бенчмарков и высокой зависимостью результатов от особенностей контекста использования [9].

Эта проблематика получает дальнейшее развитие в исследованиях, ориентированных на индустриальную практику применения SAST. Показано, что существующие подходы к оценке таких инструментов не всегда отражают реальные потребности команд разработки, поскольку акцент часто делается на формальных метриках качества, а не на пригодности результатов для принятия управленческих решений в условиях производственной эксплуатации [10].

Эмпирические исследования современных инструментов статического анализа также подтверждают, что качество предупреждений существенно различается в зависимости от используемого инструмента, класса проверяемых дефектов и особенностей сценария анализа, а сами результаты требуют дополнительной интерпретации перед практическим применением [11].

Отдельное направление современных работ связано со снижением доли ложноположительных срабатываний статического анализа. В частности, предлагаются методы дополнительной динамической верификации предупреждений, позволяющие отсекать часть нерелевантных результатов и тем самым повышать практическую полезность статического анализа для команд разработки и безопасности [12].

Наряду с этим в литературе выделяются организационные и методические барьеры внедрения SAST-инструментов, включая сложность интерпретации результатов, недостаток доверия к предупреждениям, рост операционной нагрузки на специалистов и трудности интеграции таких решений в реальные процессы разработки [13].

Переход от анализа отдельных предупреждений к задаче их приоритизации связан с ограничениями традиционных систем формализованного скоринга уязвимостей. Современные исследования показывают, что различные системы оценки нередко дают противоречивые сигналы и не всегда адекватно отражают фактическую значимость уязвимости в конкретном контексте эксплуатации, что ограничивает возможности их прямого использования в качестве единственного основания для управленческого решения [14].

Следует отметить, что проблема высокой доли нерелевантных предупреждений для SAST не является новой. Более ранние эмпирические исследования также показывали, что значительная часть сообщений таких инструментов не подтверждается как реальные уязвимости при углубленном анализе. Это снижает доверие разработчиков к результатам сканирования и затрудняет их использование в повседневной практике. Вместе с тем такие работы целесообразно рассматривать прежде всего как основу для постановки проблемы, а не как отражение текущего состояния области [15].

В более новых публикациях развивается подход к приоритизации уязвимостей как к многокритериальной задаче, в рамках которой учитываются не только формальные характеристики severity, но и дополнительные признаки, связанные с эксплуатационной реализуемостью, контекстом применения и совокупностью факторов риска. Появляются модели тонкой приоритизации, использующие комбинированные признаки и методы интеллектуального анализа данных для более точного ранжирования уязвимостей по степени практической опасности [16].

Обзорные исследования последних лет подтверждают, что современная приоритизация все чаще строится не только на стандартизованных метриках тяжести, но и на сочетании признаков эксплуатируемости, контекста развертывания, критичности актива и потенциального организационного ущерба. Это позволяет рассматривать приоритизацию уязвимостей как контекстно-зависимую и риск-ориентированную задачу, требующую объединения технических, эксплуатационных и организационных факторов в едином механизме поддержки решений [17].

Таким образом, современное состояние исследований показывает, что в литературе уже достаточно подробно раскрыты отдельные аспекты DevSecOps, применения SAST-инструментов, причин ложноположительных срабатываний и ограничений традиционного скоринга уязвимостей, однако сохраняется недостаточная разработанность моделей, предназначенных специально для многоуровневой приоритизации SAST-результатов в DevSecOps-пайплайнах с учетом достижимости кода, эксплуатационного контекста, сетевой доступности и критичности актива.

Теоретическая значимость работы состоит в том, что предложенная модель развивает риск-ориентированный подход к приоритизации уязвимостей применительно к SAST-предупреждениям в DevSecOps пайплайнах. В отличие от существующих подходов, ориентированных преимущественно на ранжирование уязвимостей по совокупности признаков, включая модели на основе методов машинного обучения, предложенная в статье схема акцентирует связь между причинами ложноположительных срабатываний, недостающими контекстными сигналами и организационной логикой принятия решений в CI/CD-контуре. Тем самым модель может рассматриваться не как альтернатива ML-подходам, а как концептуальная рамка их интерпретации и практического применения в задачах контекстной фильтрации и приоритизации результатов SAST.

Следует отметить, что отраслевые аналитические отчеты и прикладные материалы могут использоваться как источник сведений о практической значимости проблемы, однако теоретическое обоснование исследования должно опираться прежде всего на современные рецензируемые публикации. В этом контексте предлагаемая в статье модель ориентирована на восполнение выявленного исследовательского пробела и направлена на переход от статической оценки тяжести к более обоснованной контекстно-зависимой приоритизации уязвимостей в DevSecOps-процессах.

Описание материалов и методов исследования. Материалы исследования включают опубликованные агрегированные результаты эксперимента компании Ghost Security по применению традиционных SAST-инструментов к open-source-проектам на стеках Go/Gin, Python/Flask и PHP/Laravel, а также современные рецензируемые публикации по DevSecOps, статическому анализу и приоритизации уязвимостей [8].

Внешние данные Ghost Security используются в работе не как самостоятельное доказательство выводов, а как эмпирический кейс, позволяющий проиллюстрировать типовые проблемы интерпретации результатов статического анализа, прежде всего высокий уровень ложноположительных срабатываний и возникающую вследствие этого перегрузку команд разработки и безопасности. Агрегированные показатели, характеризующие рассматриваемые технологические стеки и классы уязвимостей, приведены в таблице 1 [3].

Исследовательский дизайн работы представляет собой аналитико-концептуальное исследование с элементами вторичного анализа опубликованных эмпирических данных. Цель данного этапа исследования состоит в выявлении повторяющихся причин нерелевантных SAST-срабатываний, их систематизации и последующем построении многоуровневой модели приоритизации, применимой в DevSecOps-пайплайнах [16].

Таблица 1

Итог статического анализа open-source репозиториев

Источник: составлена авторами

Язык программирования / фреймворк	Класс уязвимостей	Количество репозиториев просканировано	Потенциальные уязвимости	Ложно-положительные срабатывания
Go/Gin	SQL-инъекция	856	805	646 (80,25%)
Python/Flask	Инъекция команд ОС	1000	1166	1160 (99,49%)
PHP/Laravel	Произвольная загрузка файлов	1000	145	130 (89,66%)

Исследование выполнялось в четыре этапа. На первом этапе были отобраны и систематизированы опубликованные агрегированные данные и описания кейсов, относящиеся к трем технологическим стекам и трем классам уязвимостей, представленным в отчете Ghost Security. На втором этапе был выполнен качественный анализ описанных сценариев срабатывания SAST-инструментов с целью выявления типовых причин, по которым предупреждение формально фиксируется анализатором, но не соответствует реальной эксплуатируемой уязвимости в конкретном контексте. На третьем этапе выявленные случаи были сгруппированы в укрупненные категории ложноположительных срабатываний. На четвертом этапе полученные категории были сопоставлены с подходами риск-ориентированного управления уязвимостями и использованы как основа для построения концептуальной многоуровневой модели приоритизации.

В качестве основных методов исследования применялись вторичный анализ опубликованных эмпирических данных, качественный сравнительный анализ кейсов, классификация причин ложноположительных срабатываний и концептуальное моделирование. Вторичный анализ использовался для интерпретации уже опубликованных результатов без проведения собственного инструментального эксперимента. Сравнительный анализ позволил сопоставить особенности ложноположительных срабатываний в разных стеках и для разных классов уязвимостей. Метод классификации применялся для выделения повторяющихся типов нерелевантных алертов, а концептуальное моделирование для формализации связей между типом срабатывания, недостающими контекстными сигналами и механизмами риск-ориентированной приоритизации.

В рамках исследования все рассматриваемые ложноположительные срабатывания интерпретируются не как случайные ошибки конкретного инструмента, а как следствие ограниченного учета контекста выполнения кода, условий развертывания и эксплуатационной достижимости потенциально уязвимых участков. Такой подход позволяет перейти от описания частных технических дефектов к анализу управленческой задачи фильтрации и ранжирования алертов в организационной DevSecOps-среде.

Результатом проведенного анализа стала классификация причин ложноположительных срабатываний SAST, включающая 5 типов алертов:

1) совпадение с сигнатурой правила при отсутствии реально контролируемого пользовательского ввода;

2) наличие пользовательского ввода, прошедшего достаточную валидацию или санитизацию;

3) формально уязвимый, но практически недостижимый код;

4) тестовый, демонстрационный или вспомогательный код, не входящий в продуктивный контур;

5) случаи, в которых фактический риск существенно снижается за счет компенсирующих инфраструктурных мер.

Данная классификация далее используется как аналитическая основа для построе-

ния многоуровневой модели приоритизации уязвимостей в DevSecOps-пайплайнах.

Для оценки применимости предложенной модели использовалась сценарная аналитическая оценка, основанная на сопоставлении выделенных категорий ложноположительных срабатываний с возможными механизмами их контекстного обогащения и понижения приоритета. Такая оценка не рассматривается как окончательная эмпирическая валидация модели, а служит средством предварительной проверки ее логической согласованности и практической реализуемости в условиях ограниченных ресурсов команд разработки и безопасности.

Ограничения исследования связаны с тем, что работа опирается на вторичный анализ опубликованных данных внешнего исследования и не включает собственный контролируемый эксперимент на независимой выборке проектов. Кроме того, рассматриваемый эмпирический кейс охватывает ограниченное число технологических стеков и классов уязвимостей, что не позволяет автоматически переносить количественные оценки на все типы SAST-сценариев. По этой причине полученные результаты следует интерпретировать прежде всего как концептуально-аналитические, а предложенную модель — как инструмент, требующий последующей пилотной апробации в реальных DevSecOps-процессах

Результаты исследования. В работе рассматривается задача снижения объёма нерелевантных предупреждений SAST в DevSecOps-пайплайнах за счёт их контекстной интерпретации и риск-ориентированной приоритизации. В качестве эмпирической иллюстрации использованы опубликованные агрегированные результаты эксперимента Ghost Security по open-source-проектам на стеках Go/Gin, Python/Flask и PHP/Laravel, которые в настоящем исследовании рассматриваются как внешний эмпирический кейс для аналитической интерпретации и концептуального обобщения [3].

Исходные данные показывают, что базовый уровень ложноположительных срабатываний традиционных SAST-инструментов в рассматриваемых стеках остается высоким: для Python/Flask он составляет 99,49%, для Go/Gin – 80,25% (646 ложноположительных срабатываний из 805), для PHP/Laravel – 89,66%. Эти показатели подтверждают практическую значимость задачи фильтрации и приоритизации предупреждений, поскольку при сохранении традиционного подхода значительная часть ресурсов команды расходуется на ручную проверку алертов, не соответствующих реально эксплуатируемым уязвимостям.

Основным результатом исследования является авторская классификация причин ложноположительных срабатываний SAST, используемая как аналитическая основа для дальнейшего построения многоуровневой модели приоритизации. В рамках проведённого анализа выделены следующие категории:

1) отсутствие реально контролируемого пользовательского ввода;

2) наличие пользовательского ввода, уже прошедшего достаточную валидацию;

3) формально уязвимый, но практически недостижимый код;

4) тестовый, демонстрационный или вспомогательный код, не входящий в продуктивный контур;

5) случаи, в которых фактический риск существенно снижается за счёт компенсирующих инфраструктурных мер.

Научная новизна данного результата состоит в том, что перечисленные категории рассматриваются не как изолированные частные случаи шумовых алертов, а как основание для систематического контекстного обогащения результатов SAST и их включения в единый механизм риск-ориентированной приоритизации в DevSecOps-среде. Тем самым предложенная классификация связывает причины ложноположительных срабатываний с конкретными типами недостающих сигналов, которые должны учитываться при принятии решения о приоритете обработки предупреждения.

Представленные в таблице 2 значения по пяти категориям следует интерпретировать как аналитическую экспертную оценку, сформированную на основе интерпретации опубликованных кейсов Ghost Security и их сопоставления с современными исследованиями в области SAST и приоритизации уязвимостей. Эти значения не претендуют на статус статистически обоснованного распределения для

генеральной совокупности SAST-срабатываний и используются прежде всего для структурирования проблемы и демонстрации применимости предложенной классификации в рамках рассматриваемого эмпирического кейса. В анализируемом материале наибольшая доля ложноположительных срабатываний приходится на категории, связанные с отсутствием реального пользовательского ввода и с уже провалидированным вводом. В совокупности они формируют ориентировочно 75% рассмотренных случаев. Данный вывод следует трактовать как результат экспертной интерпретации выбранного кейса, а не как универсальную количественную закономерность для всех DevSecOps пайплайнов и всех классов SAST-предупреждений.

Проведённый анализ показывает, что для каждой из выделенных категорий существует набор контекстных сигналов, отсутствие которых и приводит к завышению приоритета предупреждения традиционными SAST-инструментами. Для первой категории критичен более точный учёт источников данных и различение пользовательского ввода и внутренних констант. Для второй категории определяющее значение имеет корректное распознавание санитайзеров и безопасных паттернов валидации. Для третьей категории ключевым становится анализ достижимости потенциально уязвимого участка от внешних точек входа. Для четвёртой категории значимым сигналом выступает тип файла и принадлежность к непроизводственному окружению, а для пятой сведения о сетевой экспозиции сервиса и компенсирующих инфраструктурных решениях.

На основе этой классификации разработана концептуальная многоуровневая модель приоритизации, в которой базовая техническая серьёзность предупреждения дополняется признаками достижимости, типом окружения, сетевой экспозицией, наличием компенсирующих мер, а также данными об эксплуатируемости и критичности актива. В отличие от традиционного ранжирования по severity, такая модель ориентирована на поддержку управленческого решения в DevSecOps-пайплайне и позволяет интерпретировать поток SAST-алертов как объект риск-ориентированной фильтрации и скоринга.

Представленная в таблице 2 оценка потенциального эффекта снижения ложноположительных срабатываний носит сценарный характер и используется для предварительной проверки практической применимости предложенной модели. Сценарное сопоставление показывает, что совокупный объём ложноположительных срабатываний потенциально может быть сокращён примерно на 50–55%, а время ручной верификации алертов на 60–70% по сравнению с базовой конфигурацией обработки результатов. Сценарное сопоставление показывает, что предложенная модель потенциально может сократить объём ложноположительных срабатываний и снизить нагрузку на ручную верификацию алертов по сравнению с базовой конфигурацией обработки результатов.

Таким образом, полученные результаты носят преимущественно концептуально-аналитический характер. Основной вклад работы заключается в систематизации причин ложноположительных срабатываний SAST и разработке многоуровневой модели приоритизации, обеспечивающей переход от формальной обработки предупреждений к их контекстно-зависимой интерпретации в DevSecOps-среде. Потенциальный эффект ложноположительных срабатываний по категориям представлен в таблице 3. Сценарное сопоставление, представленное в таблице 3, позволяет предположить, что наибольший эффект достигается для категорий, где недостающий сигнал может быть получен посредством относительно простых конфигурационных изменений, прежде всего для тестового или демонстрационного кода и недостижимых участков. Наименьший эффект, вероятно, связан с кейсами провалидированного ввода, поскольку они требуют более глубокого семантического анализа и точного описания. Такая картина в целом согласуется с практикой DevSecOps, где в первую очередь реализуются наиболее простые и малозатратные меры с заметным влиянием на снижение шума, а более сложные улучшения модели анализа кода внедряются постепенно.

Таблица 2

Аналитическая оценка распределения ложноположительных срабатываний SAST по категориям.

Источник: составлена авторами

Категория ложных срабатываний	Доля	Характерные примеры
1 – Отсутствие реального пользовательского ввода	35%	Служебные скрипты с жёстко заданными параметрами
2 – Провалидированный пользовательский ввод	40%	Параметры после strconv.Atoi в Go-приложениях
3 – Недостижимый код	12%	Закрытые эндпоинты, внутренние обработчики
4 – Тестовый/демонстрационный код	8%	Файлы скриптов не используемые в конечном артефакте
5 – Компенсирующие инфраструктурные контроли	5%	Сервисы с доступом только через localhost или ограниченные ACL

Таблица 3

Сценарная оценка потенциала снижения ложноположительных срабатываний по категориям

Источник: составлена авторами

Категория	Механизм снижения	Потенциальный эффект
1 – Отсутствие пользовательского ввода	Улучшенный анализ потока данных с учётом доверенных источников	65%
2 – Провалидированный ввод	Использование моделей санитизации и распознавание безопасных паттернов валидации	45%
3 – Недостижимый код	Интеграция с DAST и анализ достижимости путей от внешних точек входа	78%
4 – Тестовый код	Исключение из области сканирования	92%
5 – Компенсирующие контроли	Учёт контекста развертывания и сетевой экспозиции сервиса	60%

Сценарное сопоставление, представленное в таблице 3, позволяет предположить, что наибольший эффект достигается для категорий, где недостающий сигнал может быть получен посредством относительно простых конфигурационных изменений, прежде всего для тестового или демонстрационного кода и недостижимых участков. Наименьший эффект, вероятно, связан с кейсами провалидированного ввода, поскольку они требуют более глубокого семантического анализа и точного описания. Такая картина в целом согласуется с практикой DevSecOps, где в первую очередь реализуются наиболее простые и малозатратные меры с заметным влиянием на снижение шума, а более сложные улучшения модели анализа кода внедряются постепенно.

Предлагаемый DevSecOps-пайплайн, показанный на рисунке 2, строится вокруг использования SAST как источника первично найденных уязвимостей и последующего риск-ориентированного обогащения этих данных. На этапе сборки в CI/CD выполняется сканирование только продуктового кода и связанных конфигураций с явным исключением тестовых, демонстрационных и вспомогательных директорий из области анализа. Результаты SAST выгружаются в централи-

зованную систему управления уязвимостями, где для каждого срабатывания автоматически учитываются тип файла и окружения, достижимость подозрительного участка из внешних точек входа, а также сетевой контекст. На следующем шаге алерты классифицируются по предложенной классификации причин ложноположительных срабатываний и получают приоритет с учётом этих контекстных сигналов, после чего политики реакции, блокировка сборки, постановка задач, перевод в бэклог, определяются уже не только по технической серьёзности, но и по реальному профилю риска.

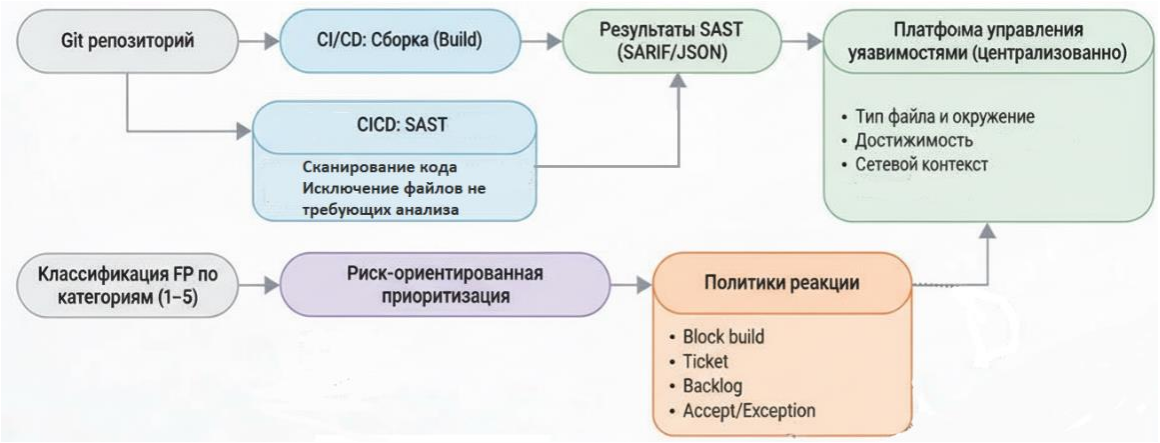

Рисунок 2 – Предлагаемый DevSecOps-пайплайн
с использованием описанной классификации
Источник: составлен авторами

Такой подход соответствует логике риск-ориентированного управления уязвимостями RBVM, в рамках которой приоритизация строится не только на формальных показателях вроде CVSS, но и на признаках эксплуатируемости. В предлагаемой модели это отражается в многоуровневом скоринге, сначала учитывается техническая серьёзность и класс уязвимости, затем – наличие известных эксплойтов или аналогов KEV, далее – контекст (достижимость, тип окружения, принадлежность к категориям 3–5), и только после этого – роль компонента в бизнес-процессах. Такой риск-ориентированный слой над результатами SAST позволяет систематически понижать приоритет срабатываний, отнесённых к категориям 3, 4 и 5, и наоборот, усиливать внимание к тем случаям, где техническая уязвимость сочетается с высокой вероятностью эксплуатации уязвимости.

Ожидается, что пилотная апробация на некотором количестве open-source проектов среднего размера позволит количественно оценить фактическое снижение доли ложноположительных срабатываний, подтвердить и скорректировать моделируемые диапазоны ложноположительных срабатываний. При этом планируется ограничиться изменениями на уровне настройки существующих инструментов и обработки их результатов, области сканирования, исключения, пост-обработка алертов.

С учётом текущих результатов предложенную модель целесообразно рассматривать как концептуальный инструмент снижения количества ложноположительных срабатываний SAST при сохранении контроля над реальными рисками. Универсальность классификации и отсутствие жёсткой привязки к конкретному поставщику позволяют рекомендовать её к пилотному внедрению в разнородных технологических средах, где статический анализ уже интегрирован в DevSecOps-пайплайны, но команды сталкиваются с высокой долей ложноположительных срабатываний. Эмпирическая проверка этих эффектов остаётся задачей последующих этапов работы.

Выводы. Проведённое исследование показало, что интеграция SAST-инструментов в DevSecOps-процессы при отсутствии контекстной фильтрации приводит к экстремально высокой доле ложноположительных срабатываний и перегрузке команд разра-

ботки и безопасности. На примере данных Ghost Security продемонстрировано, что даже при кастомных правилах подавляющее большинство алертов не соответствует реально эксплуатируемым уязвимостям, а традиционная приоритизация, опирающаяся преимущественно на CVSS, не позволяет надёжно выделять действительно критичные проблемы.

В ответ на обозначенную проблему в работе предложена классификация причин ложноположительных срабатываний SAST, включающая отсутствие пользовательского ввода, провалидированный ввод, недостижимый код, тестовый или демонстрационный код, а также компенсирующие инфраструктурные контроли. В рамках рассмотренного эмпирического материала полученные аналитические оценки позволяют предположить, что значительная часть ложноположительных срабатываний связана не столько с некорректностью самих правил, сколько с недостаточным учетом потока данных и контекста исполнения. На основе данной классификации разработана концептуальная многоуровневая модель риск-ориентированной приоритизации, дополняющая базовую техническую серьёзность уязвимости сигналами эксплуатируемости, достижимости и экспозиции компонента; кроме того, описан целевой DevSecOps пайплайн, в котором результаты SAST последовательно обогащаются указанными контекстными признаками.

Сценарная оценка позволяет предположить, что внедрение предложенной схемы фильтрации и скоринга потенциально может сократить общий объём ложноположительных срабатываний SAST примерно на половину, снизить трудозатраты на ручные проверки и ускорить обработку действительно критичных уязвимостей за счёт перераспределения внимания аналитиков.

Дальнейшие работы целесообразно направить на пилотное внедрение классификации и риск-ориентированного скоринга в реальных DevSecOps-пайплайнах для нескольких разнородных технологических стеков, количественную проверку фактического снижения доли ложноположительных срабатываний, а также на разработку инструментальной поддержки, в том числе с использованием методов ИИ для автоматической классификации и обогащения результатов SAST контекстной информацией.

Список источников (Reference)

1. ReversingLabs. Shai-hulud npm Attack: What You Need to Know. *ReversingLabs, 2024* [Electronic resource]. URL: https://www.reversinglabs.com/blog/shai-hulud-worm-npm (Accessed: 15.11.2025).

2. Synopsys. 2025 Open Source Security and Risk Analysis (OSSRA) Report. *Synopsys, 2025* [Electronic resource]. URL: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html (Accessed: 15.11.2025).

3. Ghost Security. Exorcising the SAST Demons: Why Traditional SAST Falls Short in Modern AppSec. *Ghost Security, 2025* [Electronic resource]. URL: https://ghostsecurity.com/report (Accessed: 03.12.2025).

4. FIRST. Exploit Prediction Scoring System (EPSS). *EPSS User Guide* [Electronic resource]. FIRST, 2024. URL: https://www.first.org/epss (Accessed: 15.11.2025).

5.Cisco Systems, Inc. How Kenna Works: A Peek Under the Hood of Modern Vulnerability Management. *Cisco Systems, Inc*., 2022. 16 p. URL: https://www.cisco.com/c/en/us/products/security/kennasecurity/how-kenna-works.html (Accessed: 15.11.2025).

6. Cybersecurity and Infrastructure Security Agency (CISA). *Known Exploited Vulnerabilities Catalog (KEV). 2025* [Electronic resource]. URL: https://www.cisa.gov/known-exploited-vulnerabilities-catalog (Accessed: 15.11.2025).

7. Tariq S. et al. Alert Fatigue in Security Operations Centres: Research Challenges and Opportunities. *ACM Computing Surveys*. 2025. No. 57 (9). DOI: 10.1145/3723158.

8. Myrbakken H., Colomo-Palacios R. Cybersecurity in DevOps Environments: A Systematic Literature Review. *Future Internet*. 2023. Vol. 15(2). Article 57. DOI: 10.3390/fi15020057.

9. Dalaq D., Daya K. F., Dalaq A., Arefin M. N., Niazi M. K. A Systematic Literature Review on Static Application Security Testing (SAST) Tools: Evaluation, Benchmarks, Challenges, and Future Directions. *EASE Companion 2025*. 2025. P. 162–168. DOI: 10.1145/3727967.3756838.

10. Li Y., Yao P., Yu K., Wang C., Ye Y., Li S. et al. Understanding Industry Perspectives of Static Application Security Testing (SAST) Evaluation. *Proceedings of the ACM on Software Engineering*. 2025. DOI: 10.1145/3729404.

11. Charoenwet W., Thongtanunam P., Pham V.-T., Treude C. An Empirical Study of Static Analysis Tools for Secure Code Review. *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024*). 2024. pp. 691–703. DOI: 10.1145/3650212.3680313.
12. Murali A., Mathews N. S., Alfadel M., Nagappan M., Xu M. FuzzSlice: Pruning False Positives in Static Analysis Warnings through Function-Level Fuzzing. *ICSE 2024 Research Track*. 2024. DOI: 10.1145/3597503.3623321.
13. Wadhams Z., Izurieta C., Reinhold A. M. Barriers to Using Static Application Security Testing (SAST) Tools: A Literature Review. *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24*). 2024. DOI: 10.1145/3691621.3694947.
14. Manzi-Muneza A. R. et al. Software Bill of Materials in Software Supply Chain Security: A Systematic Literature Review. *arXiv preprint*. 2025. DOI: 10.48550/arXiv.2506.03507.
15. Aloraini B., Nagappan M., German D. M., Hayashi S., Higo Y. An Empirical Study of Security Warnings from Static Application Security Testing Tools. *Journal of Systems and Software*. 2019. Vol. 158. Article 110427. DOI: 10.1016/j.jss.2019.110427.
16. Wang S., Yu D., Liang X., Huang C. FVulPri: Fine-grained Vulnerability Prioritization Based on BERT-BGRU and Multiple Indicators. *Information and Software Technology*. 2025. Vol. 187. Article 107853. DOI: 10.1016/j.infsof.2025.107853.
17. Jiang Y., Oo N., Meng Q., Lim H. W. et al. A Survey on Vulnerability Prioritization: Taxonomy, Metrics, and Research Challenges. *arXiv*. 2025. DOI: 10.48550/arXiv.2502.11070.

Статья поступила в редакцию 26.02.2026; одобрена после рецензирования 23.03.2026; принята к публикации 24.03.2026.
The article was submitted 26.02.2026; approved after reviewing 23.03.2026; accepted for publication 24.03.2026.