УДК 001.817

# СТАТИЧЕСКИЕ АНАЛИЗАТОРЫ КОДА И ИХ РОЛЬ В ОБЕСПЕЧЕНИИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**М. А. Устинович, Е. И. Вакарь, Е. И. Баяк**

*Белорусский государственный университет информатики и радиоэлектроники, Минск, Беларусь, maksimustsinovich@gmail.com*

В статье рассмотрена роль статического анализа кода в обеспечении качества программного обеспечения. Описаны основные методы статического анализа, их применение для поиска ошибок, повышения безопасности и читаемости. Рассмотрены подходы к интеграциям анализаторов в процесс разработки ПО. Сравнены эффективности современных инструментов и будущие направления развития.

***Ключевые слова*:** статический анализ кода; качество программного обеспечения; анализ ошибок; безопасность кода; инструменты разработки; проверка кода; надёжность программного обеспечения.

# STATIC CODE ANALYZERS AND THEIR ROLE IN SOFTWARE QUALITY ASSURANCE

**M. A. Ustsinovich, E. I. Vakar, E. I. Bayak**

*Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus, maksimustsinovich@gmail.com*

The article considers the role of static code analysis in software quality assurance. The main methods of static analysis and their application for searching errors, increasing security and readability are described. Approaches to integrating analyzers into the process of software development are considered. The efficiency of modern tools and future development directions are compared.

***Keywords:*** static code analysis; software quality; error analysis; safety-free code; development tools; code verification; software reliability.

## 1. Введение

Современное создание программного обеспечения требует всё больше внимания к надёжности, безопасности и читаемости кода. С увеличением сложности программных систем растёт риск ошибок, которые могут вызывать сбои или уязвимости, которые могут использовать злоумышленники. Один из лучших способов обнаружить проблемы заранее

– использовать статические анализаторы кода, которые помогают находить ошибки без запуска программы. Эти инструменты полезны на разных этапах: от написания и тестирования кода до поддержки и улучшения больших проектов.

Статический анализ помогает автоматически проверять, соответствует ли код стандартам, выявлять логические ошибки, утечки памяти и потенциальные уязвимости ещё до тестирования. Это может существенно сократить затраты на исправление и улучшить качество конечного продукта, умело соблюдая правила безопасного программирования. Сегодня существует множество статических анализаторов с разными методами и уровнями автоматизации.

Статический анализ – это метод изучения программного кода без его выполнения. В отличие от динамического анализа, который требует запуска программы, статический анализ может выявить проблемы на ранних стадиях разработки, когда исправить их проще и дешевле. Это особенно важно для критически важных систем, где ошибки могут иметь серьёзные последствия, как в авиации или медицинских устройствах.

Появилось множество инструментов, таких как SonarQube, Pylint, ESLint и другие. Эти инструменты используются как в маленьких компаниях, так и у крупных игроков в ИТ. Они не только ищут синтаксические ошибки, но и анализируют управление потоком, потоки данных и обнаруживают уязвимости. Также многие из них поддерживают правила кодирования, используемые в промышленности.

Несмотря на очевидные достоинства, статические анализаторы сталкиваются с некоторыми трудностями, такими как большое количество ложных срабатываний и сложности интеграции в существующие процессы. Тем не менее, прогресс в разработке алгоритмов и увеличение мощности компьютеров значительно улучшили точность и полезность этих инструментов.

Целью данной статьи является рассмотрение современных подходов к статическому анализу кода и оценка их роли в обеспечении качества программного обеспечения. Для достижения поставленной цели решаются задачи классификации известных методов статического анализа, анализа популярных инструментов и их возможностей, а также рассмотрения практик внедрения таких средств в реальные проекты.

## 2. Методы и подходы в статическом анализе кода

Статический анализ кода – это процесс проверки программного обеспечения без его непосредственного выполнения. Он позволяет выявлять ошибки, уязвимости и отклонения от стандартов кодирования на ранних

этапах разработки. В зависимости от целей и сложности анализа, применяются различные методы и подходы.

Один из основных подходов – лексический и синтаксический анализ, при котором изучается структура кода на соответствие грамматическим правилам языка программирования. Такой анализ позволяет находить простые ошибки, например, опечатки, неверное использование операторов или несоответствие стандартам оформления кода (code style).

Более глубоким является семантический анализ, который исследует логику программы. Он направлен на обнаружение потенциальных проблем, таких как утечки памяти, деление на ноль, некорректная работа с указателями, выход за границы массива и другие дефекты. Для реализации семантического анализа часто используются такие методы, как анализ потока данных, анализ зависимостей и построение графов вызовов.

Ещё один важный метод – аннотационный подход, при котором в исходный код добавляются специальные комментарии или аннотации, задающие правила для статического анализатора. Это помогает более точно интерпретировать поведение программы и учитывать контекст использования переменных и функций.

В процессе статического анализа кода важную роль играют абстрактные синтаксические деревья. Это древовидная структура, представляющая синтаксическую структуру исходного кода на более высоком уровне абстракции, без деталей, связанных с конкретными символами или лексемами – такими как скобки или точка с запятой. Каждый узел дерева соответствует определённой конструкции языка программирования.

Построение AST обычно происходит на этапе синтаксического анализа, когда парсер преобразует поток токенов, полученных в результате лексического анализа, в иерархическую структуру, удобную для дальнейшей обработки. Именно с помощью AST реализуются многие виды статического анализа, включая проверку типов, анализ потока управления и выявление потенциальных ошибок.

Например, при семантическом анализе инструменты статического анализа обходят узлы AST, чтобы отследить, как значения передаются между переменными, где происходят вызовы функций, и какие условия могут привести к неопределённому поведению. Также AST позволяет применять паттерны поиска потенциально опасных конструкций – например, использование непроверенных входных данных в SQL-запросах, что может указывать на уязвимость типа SQL injection.

Существует два основных типа статического анализа: условный и формальный. Упрощённый анализ быстро проверяет код на основе эвристик и шаблонов, что позволяет находить типовые ошибки. Формальный анализ использует строгие математические методы для доказательства корректности программ, но требует больших ресурсов и времени.

Выбор конкретного метода зависит от задач проекта, уровня критичности системы, требований к безопасности и качества кода. Комбинация различных подходов позволяет достичь максимальной эффективности при использовании статического анализа в реальных условиях разработки программного обеспечения.

## 3. Интеграция статического анализа в процесс разработки программного обеспечения

Эффективное использование статического анализа кода возможно только при его тесной интеграции в процесс разработки программного обеспечения. Современные методологии разработки, такие как Agile, DevOps и CI/CD, подразумевают автоматизацию проверок на ранних этапах жизненного цикла продукта, что делает внедрение статического анализа особенно важным.

На этапе написания кода статические анализаторы могут быть интегрированы в среды разработки (IDE). Это позволяет разработчикам получать немедленную обратную связь о потенциальных ошибках, проблемах стиля или уязвимостях прямо в процессе работы с исходным кодом. Такой подход способствует повышению качества кода ещё до его добавления в репозиторий проекта.

На этапе проверки кода перед коммитом или при создании pull request, статический анализ может выполняться автоматически в системах контроля версий. Инструменты анализа запускаются в рамках пред настроенных проверок и не позволяют отправить в репозиторий код, содержащий критические ошибки или нарушающий принятые стандарты кодирования. Это помогает сохранять целостность базовой ветки проекта и минимизировать количество дефектов на более поздних этапах.

В системах CI/CD статический анализ обычно встраивается как отдельный этап сборки. Это позволяет регулярно выполнять полные проверки всего кода проекта, формировать отчёты по найденным проблемам и контролировать их устранение.

Для крупных проектов и систем с высокими требованиями к надёжности и безопасности (например, в области финансов, авиации, медицины) статический анализ может использоваться в рамках стандартов обеспечения качества кода. В этом случае инструменты статического анализа настраиваются на проверку соответствия конкретным нормам, а результаты анализа становятся частью документации для сертификации программного обеспечения.

Кроме того, современные системы статического анализа поддерживают интеграцию с трекерами задач, позволяя автоматически создавать задачи по найденным проблемам и отслеживать их исправление. Также доступны механизмы анализа изменений, которые позволяют фокусироваться только на тех участках кода, которые были изменены в последнем коммите, что значительно ускоряет процесс проверки.

Особое внимание стоит уделить тому, как статические анализаторы взаимодействуют с CI/CD-конвейерами, превращая проверку качества кода из периодической процедуры в неотъемлемую часть автоматизированного процесса доставки программного обеспечения. В рамках систем непрерывной интеграции и доставки, таких как, статический анализ становится не просто этапом проверки, а полноценным участником процесса принятия решений о готовности кода к выпуску. Настройка прохождения анализа как обязательного шага позволяет блокировать сборку или деплой при обнаружении критических ошибок, что особенно важно для проектов с высокими требованиями к надёжности.

Анализаторы умеют не только находить проблемы, но и классифицировать их по уровням критичности, категориям и типам, а также предоставлять рекомендации по исправлению. Интеграция с CI/CD даёт возможность сохранять историю результатов анализа, отслеживать динамику изменений и оценивать эффективность принимаемых мер. Это позволяет команде не только реагировать на текущие ошибки, но и выявлять системные слабые места в подходах к разработке, что способствует постепенному повышению общего уровня качества кода.

## 4. Заключение

Сложившаяся практика использования статического анализа в разработке программного обеспечения подтверждает его эффективность как средства повышения качества кода, предотвращения ошибок и обеспечения безопасности на ранних этапах разработки. Определение ключевых принципов интеграции, выбор рациональных методов анализа и использование современных технологий позволяют создать устойчивую основу для непрерывного совершенствования процессов разработки и обеспечения высокого уровня надёжности программных продуктов.

Однако эффективность статического анализа напрямую зависит от того, насколько последовательно и комплексно он интегрирован в существующие процессы. Его применение не должно ограничиваться единичными проверками или использоваться исключительно по требованию – только системный подход, охватывающий все этапы жизненного цикла

разработки, позволяет достичь максимального эффекта. Начиная с моментального анализа в IDE при написании кода и заканчивая автоматизированными проверками в рамках CI/CD-конвейеров, статический анализ становится неотъемлемой частью культуры качества внутри команды.

Современные системы статического анализа предоставляют широкие возможности для настройки правил, фильтрации результатов, интеграции с системами управления задачами и даже обучения под стиль кодирования проекта. Это позволяет адаптировать их под нужды как небольших стартапов, так и крупных корпоративных решений с высокими требованиями к надёжности и защищённости.

## Библиографические ссылки


1. *Рыжков Е. А., Середин О. С.* Применение технологии статического анализа кода при разработке параллельных программ // Известия Тульского государственного университета. Технические науки. 2008. № 3. С. 191–197. URL: https://cyberleninka.ru/article/n/primenenie-tehnologii-staticheskogo-analiza-koda-pri-razrabotke-parallelnyh-programm (дата обращения: 05.06.2025).

2. Применение гибридного анализа для поиска уязвимостей в исходном программном коде / А. Н. Баранов [и др.] // Известия Тульского государственного университета. Технические науки. 2022. № 12. С. 408–413. URL: https://cyberleninka.ru/article/n/primenenie-gibridnogo-analiza-dlya-poiska-uyazvimostey-v-ishodnom-programmnom-kode (дата обращения: 05.06.2025).

3. *Попов Г. А., Жунёва М. М.* Сбор метрик программного кода для анализа его уязвимостей // NBI-technologies. 2024. Т. 18, № 4. С. 21–24. URL: https://cyberleninka.ru/article/n/sbor-metrik-programmnogo-koda-dlya-analiza-ego-uyazvimostey (дата обращения: 05.06.2025).

4. *Зубов М. В., Пустыгин А. Н., Старцев Е. В.* Применение универсальных промежуточных представлений для статического анализа исходного программного кода // Доклады Томского государственного университета систем управления и радиоэлектроники. 2013. № 1. С. 64–68. URL: https://cyberleninka.ru/article/n/primenenie-universalnyh-promezhutochnyh-predstavleniy-dlya-staticheskogo-analiza-ishodnogo-programmnogo-koda (дата обращения: 05.06.2025).

5. *Моисеев М. Ю.* Итеративный алгоритм статического анализа для обнаружения дефектов в исходном коде программ // Информационно-управляющие системы. 2009. № 3. С. 34–39.