

**МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ,
КОМПЛЕКСОВ И КОМПЬЮТЕРНЫХ СЕТЕЙ / MATHEMATICAL SOFTWARE FOR COMPUTERS,
COMPLEXES AND COMPUTER NETWORKS**

DOI: <https://doi.org/10.23670/IRJ.2023.135.11>

**ВОЗМОЖНОСТИ ИНТЕГРИРОВАННОГО ПОДХОДА СТАТИЧЕСКОГО И ДИНАМИЧЕСКОГО МЕТОДОВ
АНАЛИЗА КОДА В ПРОГРАММИРОВАНИИ**

Обзор

Макеева О.В.^{1,*}, Сартаков М.В.², Чернов Е.А.³, Рысин М.Л.⁴

¹ORCID : 0000-0001-9697-4827;

¹Московский региональный социально-экономический институт, Видное, Российская Федерация

²Российский химико-технологический университет имени Д.И. Менделеева, Москва, Российская Федерация

^{3,4}МИРЭА — Российский технологический университет, Москва, Российская Федерация

* Корреспондирующий автор (makeeva-oks[at]yandex.ru)

Аннотация

Безопасность программного обеспечения стала важным компонентом процесса разработки программного обеспечения. Организации-разработчику необходимо поддерживать безопасность программного обеспечения, чтобы обеспечить целостность, подлинность и доступность программного продукта. Для обеспечения безопасности программного обеспечения одной из основных задач является выявление уязвимостей, присутствующих в исходном коде, до развертывания программного обеспечения. Обнаружение уязвимостей на ранних стадиях цикла разработки программного обеспечения значительно упрощает процесс устранения этих уязвимостей для разработчиков программного обеспечения. Обнаружение уязвимостей может быть выполнено либо на этапе производства, то есть когда программное обеспечение все еще разрабатывается, путем статического аудита исходного кода, либо динамически во время выполнения. Проанализированы системы статического и динамического анализа, поскольку они являются автоматизированными инструментами анализа кода, которые проверяют весь исходный код на предмет его качества и выявляют любые потенциальные уязвимости в системе безопасности. Отмечена важность этих систем анализа в целом и для C++ в частности.

Ключевые слова: системный анализ, программирование на C++, система статического анализа, система динамического анализа.

**POSSIBILITIES OF INTEGRATED APPROACH OF STATIC AND DYNAMIC CODE ANALYSIS METHODS IN
PROGRAMMING**

Review article

Makeeva O.V.^{1,*}, Sartakov M.V.², Chernov E.A.³, Rysin M.L.⁴

¹ORCID : 0000-0001-9697-4827;

¹Moscow Regional Socio-Economic Institute, Vidnoe, Russian Federation

²D. Mendeleev University of Chemical Technology of Russia, Moscow, Russian Federation

^{3,4}MIREA — Russian Technological University, Moscow, Russian Federation

* Corresponding author (makeeva-oks[at]yandex.ru)

Abstract

Software security has become an important component of the software development process. The developer organization needs to maintain software security to ensure the integrity, authenticity and availability of the software product. To ensure software security, one of the major tasks is to identify vulnerabilities present in the source code before the software is implemented. Detecting vulnerabilities early in the software development cycle greatly simplifies the process of remediating these vulnerabilities for software developers. Vulnerability detection can be performed either at the production stage, i.e., when the software is still being developed, by static source code auditing, or dynamically at runtime. Static and dynamic analysis systems are analysed as they are automated code analysis tools that audit all source code for quality and identify any potential security vulnerabilities. The importance of these analysis systems in general and for C++ in particular is pointed out.

Keywords: system analysis, C++ programming, static analysis system, dynamic analysis system.

Введение

Высокое качество программного обеспечения является важным требованием в программировании. Современные программные продукты становятся все более сложными и объемными. Они включают в себя множество компонентов, модулей и взаимодействующих элементов. Использование методов системного анализа позволяет разбить сложные системы на более простые компоненты и изучать их взаимодействие, что способствует более эффективной разработке и управлению сложностью. Методы системного анализа позволяют оптимизировать использование ресурсов, выявлять узкие места в системе, потенциальные проблемы и ошибки на ранних этапах разработки, предоставить инструменты для тестирования, отладки и проверки программного обеспечения и предлагать решения для повышения производительности и эффективности программ.

В связи с этим необходимость исследования методов системного анализа в программировании обусловлена ростом сложности программных систем, требованием оптимизации ресурсов, обеспечением качества, адаптацией к изменениям и обеспечением интеграции и взаимодействия между системами.

Анализ научных исследований по статистическому и динамическому анализу кода в программировании позволяет оценить текущее состояние и направления развития статистического и динамического анализа кода. Это помогает исследователям и практикам ориентироваться в существующих методах и техниках и находить области для дальнейшего исследования и улучшения.

Выявление новых методов и подходов: Анализ исследований позволяет обнаружить новые исследовательские методы, инструменты и подходы к статистическому и динамическому анализу кода. Это способствует инновациям и развитию области.

Оценка эффективности и эффективности методов: Исследования могут предоставлять данные о том, насколько эффективны и точны различные методы анализа кода. Это позволяет выбрать наиболее подходящие методы в зависимости от задач и контекста.

Идентификация проблем и ограничений: Анализ исследований позволяет выявить проблемы и ограничения существующих методов и инструментов анализа кода. Это позволяет искать пути улучшения и оптимизации анализа.

Внедрение лучших практик: Исследования могут выявлять лучшие практики и подходы к статистическому и динамическому анализу кода, которые затем могут быть применены в реальных проектах для улучшения качества и безопасности программного обеспечения.

Поддержка принятия решений: Анализ научных исследований предоставляет информацию и аргументы, которые могут помочь принимать обоснованные решения в области анализа кода, особенно в отношении выбора методов и инструментов.

Совершенствование образования и обучения: Исследования позволяют определить лучшие практики в обучении статистическому и динамическому анализу кода и применять их для улучшения качества образования и подготовки специалистов в данной области.

Развитие новых инструментов и технологий: Результаты исследований могут служить основой для разработки новых инструментов и технологий статистического и динамического анализа кода, которые могут быть полезными для разработчиков и тестировщиков программного обеспечения.

Следует отметить, что анализ научных исследований по статистическому и динамическому анализу кода и их сравнительная характеристика является важным инструментом для продвижения и развития данной области, повышения качества программного обеспечения и обеспечения безопасности приложений. Тем не менее в научной литературе недостаточно работ, посвященных сравнению данных методов. В большинстве работ отмечаются возможности и ограничения каждого метода с точки зрения их применения друг от друга, однако не учитываются возможности и интеграции.

Цель исследования – дать сравнительную характеристику методов анализа кода в программировании и предложить интегрированный подход к анализу кода с их применением.

Системы статического анализа в программировании

Анализ программ на языке C++ представляет собой процесс изучения программного кода, имеющего ошибки, для выявления их причин и разработки стратегий для их устранения. Анализ программ на C++ является важной частью разработки программного обеспечения и позволяет программистам обнаруживать ошибки до того, как они станут критическими и будут вызывать сбои в работе программы на более поздних этапах [1]. В целом, это важный этап в разработке высококачественного программного обеспечения и оптимизации его производительности [2].

В процессе анализа программ C++, разработчики используют различные инструменты и техники, такие как статический и динамический анализ кода, тестирование кода, анализ производительности и профилирование. Данные методы позволяют изучить каждый аспект программы и выявить слабые места, ошибки и узкие места производительности, что позволяет устранить их и улучшить качество и эффективность программы. Такой подход позволяет снизить количество ошибок и повысить качество программного обеспечения на C++.

В своих исследованиях Kaur A., Nayyar R. [2], Mahmood R., Mahmoud Q.H. [6], Szabo R.M., Khoshgoftaar T.M. [10], Arusoai A. et al. [1] отмечают, что анализ программируемого кода на C++ также помогает сократить временные и финансовые затраты на разработку и тестирование программного обеспечения.

Fatima A., Bibi S., Hanif R. выделили основные преимущества системного анализа [3]:

1. Обнаружение ошибок на раннем этапе разработки помогает снизить стоимость исправления ошибок;
2. Точное определение возможной ошибки в исходном коде;
3. Полное покрытие кода. Независимо от того, как часто тот или иной блок кода получает управление при выполнении, статический анализ проверяет всю кодовую базу;
4. Простота использования, поскольку нет необходимости готовить какие-либо входные наборы данных для проверки;
5. Статические анализаторы быстро и легко обнаруживают ошибки опечаток и ошибки, связанные с копированием и вставкой кода.

В научных исследованиях выделяют преимущественно два метода анализа кода: статический и динамический.

Статический анализ кода имеет много преимуществ с точки зрения автоматизации, безопасности и скорости. Различные языки программирования имеют свой собственный набор определенных уязвимостей. Для обнаружения этих уязвимостей доступны различные инструменты статического анализа кода. В этом исследовании обсуждаются различные инструменты статического анализа с открытым исходным кодом для языка программирования C/C++ и языка программирования JAVA. В этом разделе описывается инструмент статического анализа, используемый для данного исследования [3].

- FLAWFINDER: Статический анализатор, используемый с целью обнаружения уязвимостей в исходном коде, написанном на C/C++ язык программирования. Программа на C/C++ подается в качестве входных данных в инструмент. Затем инструмент генерирует список обращений (уязвимостей), отсортированных по уровню риска. Каждой уязвимости, о которой сообщает Flawfinder, присваивается уровень риска в диапазоне от 0 до 5, причем 0 указывает на уязвимость с наименьшим риском использования.

- RATS (сокращенно от Rough Auditing Tool for Security) – это инструмент статического анализа кода для проверки исходного кода C/C++ на наличие уязвимостей в системе безопасности. Переполнение буфера, условия гонки TOCTOU (Time of Check – Время использования) – вот несколько примеров уязвимостей в системе безопасности, обнаруженных этим инструментом;

- CPPCHECK – статический анализатор для обнаружения уязвимостей в исходном коде C/C++. CPPCheck не удается обнаружить никаких синтаксических ошибок. Инструмент фокусируется на отсутствии ложноотрицательных результатов, это означает, что инструмент не сообщает ни о каких неправильных ошибках, однако не обо всех ошибках может не сообщаться. Это означает, что инструмент сообщает о меньшем количестве ошибок, чем на самом деле присутствует в исходном коде. Это, в свою очередь, приводит ко множеству ложноотрицательных результатов. CPPCheck сильно фокусируется на обнаружении неопределенного поведения, такого как мертвые указатели, деление на ноль, переполнение целых чисел и разыменованное нулевого указателя, и это лишь некоторые из них;

- SPOTBUGS – статический анализатор, который проверяет исходный код Java на наличие различных категорий ошибок. Точечные ошибки сообщают о различных категориях ошибок. SPOTBUGS поставляется с плагином для обнаружения уязвимостей безопасности в исходном коде Java под названием FIND_SEC_BUGS (Найти ошибки безопасности);

- PMD – инструмент для статического анализа исходного кода JAVA на наличие различных программных недостатков, таких как неиспользуемые переменные, пустые блоки catch, создание ненужных объектов и многих других. Он поддерживается эксклюзивный API для написания ваших собственных правил, которые могут быть выполнены либо на JAVA, либо в виде автономного запроса XPath. Одним из его главных преимуществ является то, что наряду с реальными дефектами выявляются плохие практики возможна интеграция с различными платформами, такими как JDeveloper, Eclipse, ide, JBuilder, BlueJ, Code Guide и т.д.

Анализ исследований Mahmood R., Mahmoud Q.H. [6], Stefanović D. et al. [9], Goseva-Popstojanova K., Perhinschi A. [5] позволил выделить следующие недостатки статического анализа [8]:

1. Неизбежные ложные срабатывания. Статический анализатор может разозлиться из-за фрагментов кода, в которых в действительности нет никаких ошибок. В данном случае необходимо пометить предупреждение как ложноположительное;

2. Статический анализ, как правило, плохо выявляет утечки памяти и ошибки, связанные с параллелизмом. Чтобы их, необходимо выполнить какую-то часть программы в виртуальном режиме, что является чрезвычайно сложной задачей. Кроме того, такие алгоритмы требуют значительный объем памяти и процессорного времени. Статические анализаторы обычно не выходят за рамки анализа некоторых простых случаев. Динамические анализаторы больше подходят для диагностики утечек памяти и ошибок, связанных с параллелизмом.

В целом статический системный анализ играет важную роль в программировании, помогая оптимизировать производительность, улучшать качество программного обеспечения, принимать обоснованные решения и оптимизировать использование ресурсов. Однако статические анализаторы не фокусируются исключительно на поиске ошибок. Например, они могут давать рекомендации по форматированию кода. Некоторые инструменты позволяют проверить код на соответствие стандартам кодирования, например, включают отступы различных конструкций, использование пробелов/табуляции и т.д. Кроме того, статический анализ может помочь при измерении метрик, количественной меры степени, в которой программа или ее спецификации обладают каким-либо свойством [10].

Системы динамического анализа в программировании

Динамический анализ кода выполняется на программе во время выполнения. Для этого необходимо сначала скомпилировать исходный код в исполняемый файл. Другими словами, код, содержащий ошибки компиляции или сборки, не может быть проверен этим типом анализа. Проверка выполняется с помощью набора входных данных, передаваемых программе на анализ. Именно эти данные определяют степень охвата кода по окончании теста. Таким образом, эффективность динамического анализа напрямую зависит от качества и количества тестовых данных. Эти данные определяют степень охвата кода в конце теста.

Согласно Gosain A., Sharma G.A., в качестве метрик, которые могут быть отмечены динамическими системами анализа, выступают [4]:

- ресурсы, используемые во время выполнения всей программы или ее отдельных частей, количество внешних запросов (например, к базе данных), объем оперативной памяти и другие ресурсы, используемые программой;

- степень охвата кода тестами и другие метрики;

- ошибки программного обеспечения: деление на ноль, обращение к нулевому указателю, утечки памяти, состояния;

- некоторые уязвимости безопасности;

- основные преимущества динамического анализа.

Для анализа программы не требуется доступ к ее исходному коду. Однако, следует отметить, что инструменты динамического анализа отличаются способом взаимодействия с программой на анализе, например, один из распространенных методов динамического анализа предполагает инструментирование кода перед проверкой, то есть добавление специальных фрагментов кода к исходному коду приложения для возможности диагностики ошибок анализатором. В таком случае необходимо иметь под рукой исходный код программы, поскольку могут быть обнаружены сложные ошибки обработки памяти, такие как индексация внешних массивов и утечки памяти. При этом

многопоточный код можно анализировать во время выполнения, обнаруживая потенциальные проблемы, связанные с доступом к общим ресурсам или возможными взаимными блокировками.

Следует отметить, что большинство динамических анализаторов не генерируют ложные срабатывания, поскольку ошибки попадают в момент их возникновения. В связи с этим предупреждение, выданное динамическим анализатором, не является прогнозом, выполненным на основе анализа модели программы, а является констатацией факта возникновения ошибки.

Gosain A., Sharma G.A. [4], Shijo P.V., Salim A. [8] выделили недостатки динамического анализа [9]:

- Полное покрытие всего кода не гарантировано, то есть, очень маловероятно получить 100% покрытие динамическим тестированием;

- Динамические анализаторы недостаточно обнаруживают ошибки логики. Например, условие, всегда истинное, не является багом с точки зрения динамического анализатора, так как такая неправильная проверка просто исчезает на более ранней стадии компиляции;

- Трудности в точной локализации ошибки в коде программы;

- Использование динамического анализа сложнее, чем статического, поскольку для получения лучших результатов и достижения максимального покрытия кода необходимо предоставить достаточное количество данных программе.

Приведем примеры инструментов динамического анализа кода и их использования на C++:

- SonarQube предоставляет комплексную среду анализа кода и способен анализировать C++, а также другие языки программирования, что позволяет разработчикам отслеживать качество кода и обеспечивать соблюдение стандартов кодирования, а также позволяет выявить уязвимости в системе безопасности, такие как атаки с использованием межсайтовых сценариев (XSS), ошибки внедрения и другие проблемы;

- DAST направлен на тестирование безопасности, который не зависит от языка программирования. DAST не просматривает исходный код, байт-код или ассемблерный код – он анализирует готовый продукт на наличие уязвимостей;

- OCLint фокусируется на выявлении проблем с кодированием и обеспечении соблюдения стандартов кодирования, например, неиспользуемые переменные, затенение переменных и другие проблемы, которые могут повлиять на качество кода и удобство сопровождения;

- Valgrind можно использовать для обнаружения утечек памяти, разыменования нулевых указателей и других проблем во время выполнения кода. Основным преимуществом данного инструмента является то, что он позволяет выявить проблемы, которые могут быть не сразу очевидны во время проверки кода или статического анализа;

- Frama-C – инструмент, который можно использовать для анализа кода на C и C++ с целью выявления проблем с качеством и безопасностью. Он включает в себя ряд аналитических плагинов, которые способны обнаруживать потенциальные переполнения буфера, разыменования нулевых указателей и другие проблемы.

На основе проведенного исследования целесообразно представить сравнительную характеристику статического и динамического анализа кода (рис. 1).

Статический анализ
<ul style="list-style-type: none"> •+ Используется уже на ранних этапах жизненного цикла программного обеспечения, прежде чем код готов для исполнения и до начала тестирования. •+ Могут анализироваться существующие базы кодов, которые уже прошли тестирование. •- Могут обнаруживаться программные ошибки и уязвимости, которые могут привести к отказу программы или воздействуют на поведение программы во время ее реального исполнения. •- Есть вероятность «ложного срабатывания»
Динамический анализ
<ul style="list-style-type: none"> •+ Редко возникают «ложные срабатывания», высокая продуктивность по нахождению ошибок •+ Для отслеживания причины ошибки может быть произведена полная трассировка стека и среды исполнения •+ Захватываются ошибки в контексте работающей системы, как в реальных условиях, так и в режиме имитационного моделирования •- Происходит вмешательство в поведение системы в реальном времени, важно учитывать это при работе с критическим ко времени кодом •- Полнота анализа ошибок зависит от степени покрытия кода

Рисунок 1 - Сравнительная характеристика статического и динамического анализа кода

DOI: <https://doi.org/10.23670/IRJ.2023.135.11.1>

Как видно, каждый метод имеет свои преимущества и недостатки. Тем не менее можно предположить, что разработчик можно использовать оба метода интегрированно. Представим общий алгоритм применения интегрированного подхода к анализу кода:

1. Определение задачи: Определить цели анализа. Например, выявление потенциальных ошибок, утечек памяти, улучшение производительности, повышение безопасности;

2. Статический анализ:

- Выбор инструмента: Выбрать подходящий статический анализатор кода. Примеры: SonarQube, ESLint, PyLint, Checkstyle;

- Настройка: Настроить анализатор под требования вашего проекта;
 - Запуск анализа: Запустить статический анализатор на коде проекта;
 - Анализ результатов: Проанализировать полученные отчеты, выявить потенциальные проблемы, такие как структурные ошибки, стиль кодирования, потенциально опасные участки;
3. Динамический анализ:
- Выбор инструмента: Выбрать инструменты для динамического анализа, например, профилировщики (Profiler) и отладчики;
 - Подготовка тестов: Разработать тестовые сценарии, которые будут воспроизводить различные ситуации выполнения кода;
 - Запуск тестов: Запустить тестовые сценарии с использованием выбранных инструментов;
 - Анализ результатов: Изучить полученные данные, определить места, где происходят утечки памяти, медленные операции или другие проблемы во время выполнения;
4. Сопоставление результатов: Сопоставить результаты статического и динамического анализа. Идентифицировать проблемы, которые могут быть обнаружены только одним из методов. Определить, какие из выявленных проблем требуют наиболее срочного решения и какие могут быть отложены;
5. Исправление и оптимизация:
- Коррекция ошибок: Исправить выявленные ошибки и недочеты на основе результатов статического анализа;
 - Оптимизация: Оптимизировать участки кода, выявленные в результате динамического анализа, для повышения производительности и снижения утечек памяти;
6. Повторный анализ: После внесения изменений провести повторный статический и динамический анализ для проверки эффективности внесенных изменений;
7. Тестирование: Провести тестирование, чтобы убедиться, что внесенные изменения не привели к новым проблемам;
8. Документирование: Зафиксировать результаты анализа, внесенные изменения и принятые решения;
9. Итерации: Повторять анализ и работу по исправлениям на протяжении жизненного цикла проекта, чтобы постоянно улучшать качество кода.

На наш взгляд, интегрированное применение статического и динамического методов анализа кода позволяет значительно улучшить процесс разработки программного обеспечения, выявлять и устранять ошибки, оптимизировать производительность и обеспечивать более надежное и эффективное программирование.

Заключение

Таким образом, в основе динамического анализа кода находится изучение функционирования программы в реальном времени, анализ поведения программы при различных сценариях использования, включая входные данные, взаимодействие с внешними системами и т.д. Тем самым в результате динамического анализа можно получить детальную информацию о выполнении программы, такую как время выполнения отдельных частей кода, использование памяти, обращение к ресурсам и другие аспекты работы программы. Основной задачей его использования является возможность выявить ошибки, уязвимости и проблемы производительности на ранних стадиях разработки или при тестировании программы.

В основе статического анализа лежит анализ статистических данных, собранных во время выполнения программы или на основе исторических данных. В его задачи входят исследование различных статистических показателей, таких как среднее, стандартное отклонение, корреляция и др., для получения информации о поведении программы и ее производительности. При этом статический анализ позволяет выявить общие закономерности и тенденции в данных, а также проводить статистические тесты для проверки гипотез и принятия статистически обоснованных решений.

Итак, статический анализ и динамический анализ представляют разные методы к анализу программного обеспечения. Статический анализ работает с общими закономерностями и большим объемом данных, в то время как динамический анализ фокусируется на конкретном выполнении программы и предоставляет детальную информацию о ее работе. Оба подхода имеют свою ценность и могут быть использованы в зависимости от конкретных задач и целей анализа. Тем не менее представленный алгоритм комбинирования статического и динамического анализа кода позволяет предотвращать возврат к более ранним этапам разработки по мере развития программного продукта. Представленный интегрированный подход с использованием сразу двух методов анализа кода помогает избежать проявления большинства проблем еще на ранних этапах разработки, когда их легче исправить.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. Arusoai A. A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in c/c++ Code / A. Arusoai, S. Ciobăca, V. Craciun [et al.] // 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). — 2017. — P. 161-168.

2. Kaur A. A Comparative Study of Static Code Analysis Tools for Vulnerability Detection in c/c++ and Java Source Code / A. Kaur, R. Nayyar // *Procedia Computer Science*. — 2020. — Vol. 171. — P. 2023-2029.
3. Fatima A. Comparative Study on Static Code Analysis Tools for c/c++ / A. Fatima, S. Bibi, R. Hanif // *15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. — 2018. — P. 465-469.
4. Gosain A. A Survey of Dynamic Program Analysis Techniques and Tools / A. Gosain, G.A. Sharma // *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA)*. — 2014. — Volume 1. — P. 113-122.
5. Goseva-Popstojanova K. On the Capability of Static Code Analysis to Detect Security Vulnerabilities / K. Goseva-Popstojanova, A. Perhinschi // *Information and Software Technology*. — 2015. — Vol. 68. — P. 18-33.
6. Mahmood R. Evaluation of Static Analysis Tools for Finding Vulnerabilities in Java and C/C++ Source Code / R. Mahmood, Q.H. Mahmoud. — 2018. — DOI: 10.48550/arXiv.1805.09040.
7. Russell R. Automated Vulnerability Detection in Source Code Using Deep Representation Learning / R. Russel, L. Kim, L. Hamilton [et al.] // *17th IEEE international Conference on Machine Learning and Applications (ICMLA)*. — 2018. — P. 757-762.
8. Shijo P.V. Integrated Static and Dynamic Analysis for Malware Detection / P.V. Shijo, A. Salim // *Procedia Computer Science*. — 2015. — Vol. 46. — P. 804-811.
9. Stefanović D. Static Code Analysis Tools: a systematic literature review / D. Stefanović, D. Nikolić, D. Dakic [et al.] // *Annals of DAAAM & Proceedings*. — 2020. — Vol. 7. — №1. — P. 565-573.
10. Szabo R.M. An Assessment of Software Quality in a C++ Environment / R.M. Szabo, T.M. Khoshgoftaar // *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*. — 1995. — P. 240-249.