Найти в открытых источниках:

**1. Цели анализа исходного кода**

**Анализ кода** – это процесс изучения программного кода с целью выявления ошибок, уязвимостей и улучшения его качества.

**Статический анализ кода** – анализ исходного кода [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), производимый без реального выполнения исследуемых программ. Он позволяет выявить ошибки, уязвимости и потенциальные проблемы на ранних стадиях разработки. Основные преимущества статического анализа включают:

* Обнаружение ошибок на ранних этапах разработки;
* Повышение качества кода;
* Снижение затрат на исправление ошибок;
* Улучшение читаемости и поддерживаемости кода;
* Снижение вероятности возникновения критических ошибок в продакшене.

Статический анализ кода осуществляется путем проверки исходного кода на соответствие определенным правилам и стандартам. Это может включать проверку на наличие синтаксических ошибок, соответствие стилю кодирования, а также выявление потенциальных уязвимостей и проблем с производительностью

**Динамический анализ кода** – анализ [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), производящийся при помощи выполнения программ на реальном или виртуальном [процессоре](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80). Он позволяет выявить ошибки, которые невозможно обнаружить при статическом анализе. Основные преимущества динамического анализа включают:

* Обнаружение ошибок в реальном времени;
* Проверка производительности и поведения программы;
* Тестирование взаимодействия компонентов;
* Выявление проблем с памятью и ресурсами;
* Анализ поведения программы в различных условиях эксплуатации.

**2. Какие сайты входят в ТОП-5 по публикации известных уязвимостей кроме owasp?**

* [cgisecurity.com](https://www.similarweb.com/ru/website/cgisecurity.com/)
* [webappsec.org](https://www.similarweb.com/ru/website/webappsec.org/)
* [securityfocus.com](https://www.similarweb.com/ru/website/securityfocus.com/)
* [cert.org](https://www.similarweb.com/ru/website/cert.org/)
* [acunetix.com](https://www.similarweb.com/ru/website/acunetix.com/)

**3. Кто и в рамках чего впервые описал пирамиду тестирования?**

Мартин Фаулер (Martin Fowler)

Тестовая пирамида (Test pyramid) - способ использования тестов разного вида для создания сбалансированного портфолио. Ключевая особенность в том, что у вас должно быть больше низкоуровневых [модульных (unit-) тестов](https://technical-excellence.ru/wiki/UnitTest.html), чем высокоуровневых ([BroadStackTest](https://technical-excellence.ru/wiki/BroadStackTest.html)), которые вызывают приложение через GUI.

Под автотестами понимались тесты, которые обращались к приложению через пользовательский интерфейс. Такие инструменты часто предоставляли возможность записывать взаимодействие пользователя и приложения в виде сценариев, запускать их проверяя, что приложение ведёт себя корректно. Изначально такой подход работал хорошо. Потому что записывать сценарии довольно легко, и заниматься этим могут люди без навыков программирования.

Но этот подход быстро стал источником проблем, став “рожком с мороженым”. Тестирование через пользовательский интерфейс довольно медленное, и в целом увеличивает время разработки. Часто оно требует лицензии на специальное ПО для автоматизации тестирования, что означает, что такое ПО может быть использовано только на индивидуальном рабочем месте. Обычно это ПО не может запущенно в консольном режиме, и не может быть корректно помещено в [deployment pipeline](https://technical-excellence.ru/wiki/DeploymentPipeline.html).

Самое главное, что такие тесты очень хрупкие. Дальнейшая разработка системы может легко привести к тому, что многие такие сценарии сломаются, и их придется заново записывать. Вы можете обойти эту проблему, отказавшись от инструментов записи и воспроизведения, но это усложняет написание тестов.

Даже при использовании хороших практик при написании таких тестов они более подвержены [недетерминированным](https://martinfowler.com/articles/nonDeterminism.html) проблемам, что может подорвать доверие к ним.

Короче говоря, тесты, которые проходят через пользовательский интерфейс, являются хрупкими, дорогостоящими в написании и длительными по времени для выполнения. Таким образом, пирамида утверждает, что вам следует проводить гораздо больше автоматизированного тестирования с помощью модульных тестов, чем с помощью традиционного тестирования на основе графического интерфейса.

Источники:

<https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/test-asp-net-core-mvc-apps>

<https://technical-excellence.ru/wiki/TestPyramid.html>