ГОСТ Р 71207–2024 — Статический анализ программного обеспечения. Общее описание и актуальность

Вебинар от команды PVS-Studio

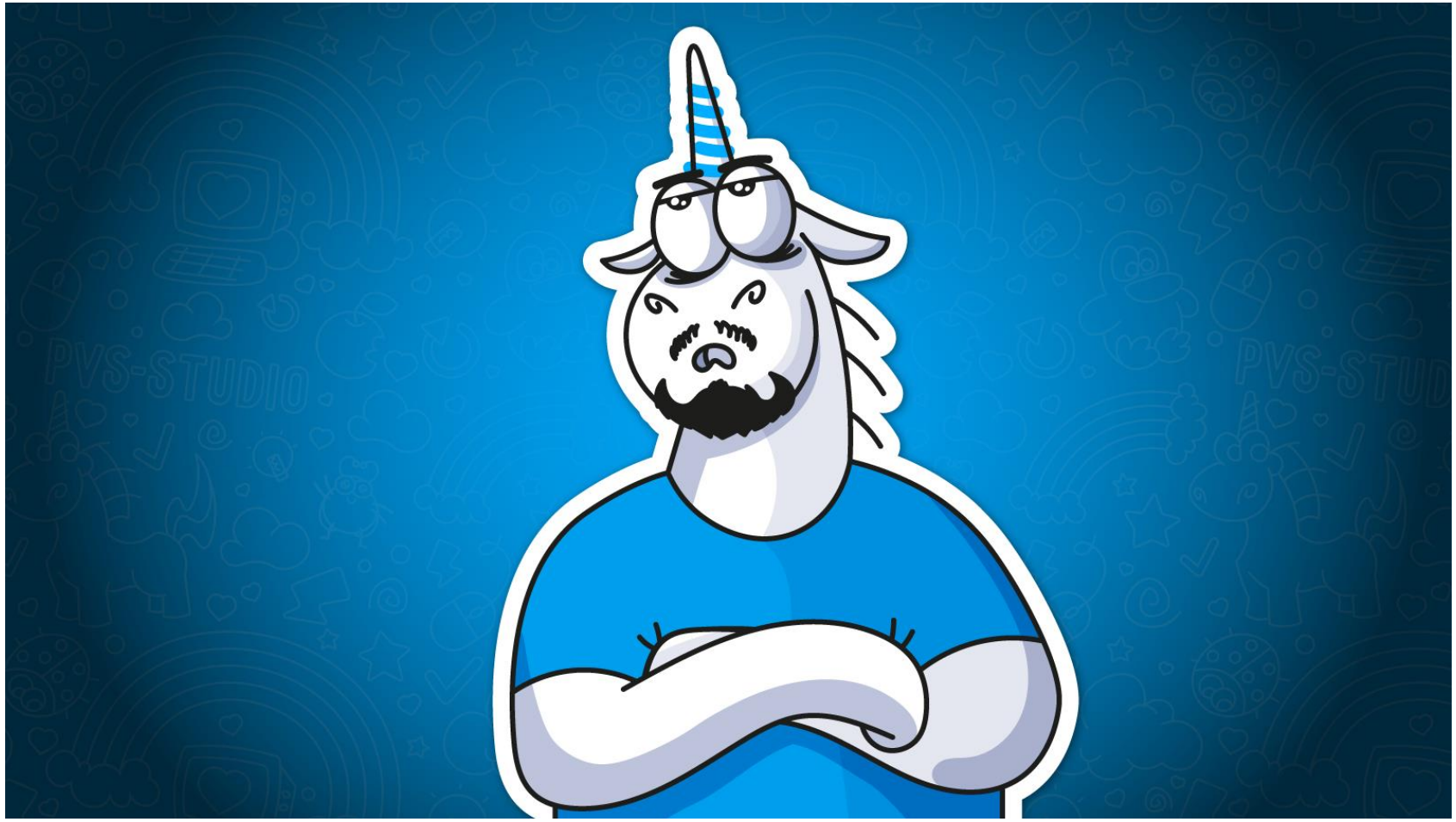
Андрей Карпов

DevRel

Андрей Карпов

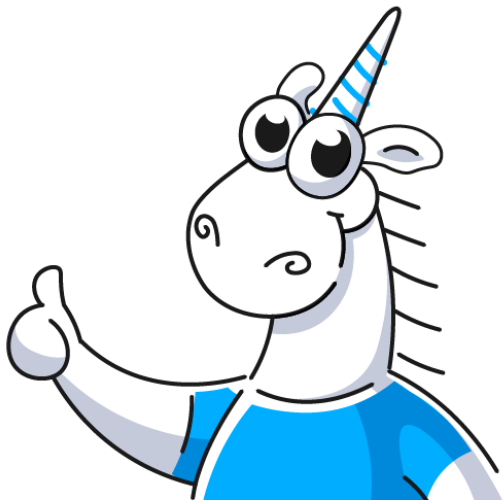
- Один из основателей проекта PVS-Studio — pvs-studio.ru
- Пишу и рассказываю про статический анализ и качество кода.

Первая реакция, когда слышим «ГОСТ»



Всё лучше и интересней, чем ожидал

- Возможно, у меня профессиональная деформация, но мне понравилось.
- ГОСТ написан понятно.
- Полезная информация и хорошие рекомендации.
- Не оторван от реальности! Всё по делу.
- Спасибо всем, кто работал над его составлением.
  - (ФСТЭК России, ИСП РАН)



ГОСТ Р 71207–2024

ГОСТ Р 71207–2024

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статический анализ программного обеспечения

Общие требования

Введён в действие 01.04.2024

Введён впервые

Применяется совместно с ГОСТ Р 56939

- РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ Общие требования
- Направлен на предотвращение и устранение уязвимостей программ

- Меры по разработке безопасного ПО:
  - …
  - Динамический анализ кода
  - **Статический анализ кода**
  - …

В соответствии с требованиями ГОСТ Р 56939

- Разработчик ПО должен проводить регулярный поиск уязвимостей на этапе эксплуатации жизненного цикла ПО.
- В состав применяемых инструментальных средств разработчик ПО должен включать **статический анализатор**.
- Рекомендуется настраивать конфигурацию статического анализатора и применять специализированные методы статического анализа для более глубокого поиска ошибок.

ГОСТ Р 71207–2024 описывает

- термины;
- порядок внедрения и выполнения статического анализа;
- требования к его выполнению;
- классификацию ошибок, обнаруживаемых статическими анализаторами;
- требования к методам анализа;
- требования к инструментам статического анализа;
- требования к специалистам, участвующим в выполнении анализа;
- требования к методике проверки статических анализаторов на соответствие требованиям стандарта.

Актуальность
ГОСТ Р 71207–2024

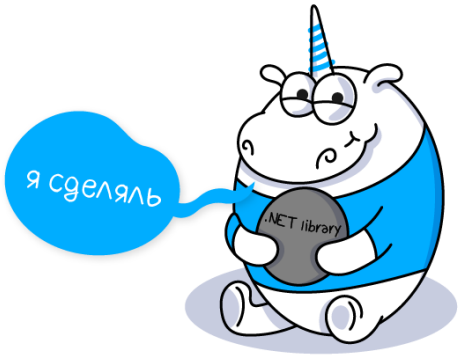
Проблематика: процессы

- Анализатор регулярно запускается на сервере.
  Значит, внедрён?

- У нас статический анализ внедрён. DevOps отдел настроил регулярный запуск какого-то анализатора. Правда, отчёты никто не смотрит.
- Про KPI и правку имён переменных.

Проблематика: свои инструменты

- Самописные инструменты.
- В этом нет плохого, если эти инструменты используются как дополнение.
- Но считать, что 2 программиста сделали «свой анализатор кода», – самообман.
- По завершению цикла вебинаров станет понятно, насколько это объёмные (сложные) решения.

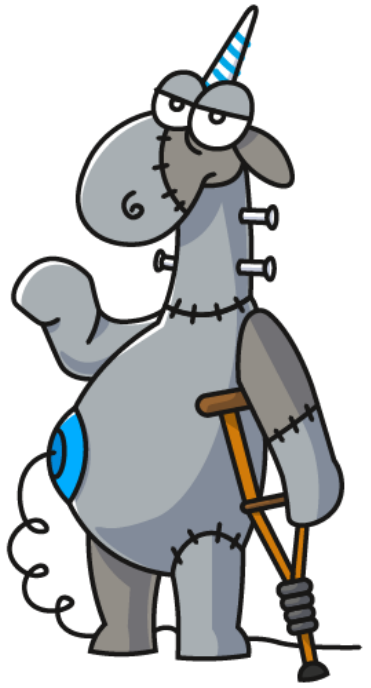


Проблематика: не те инструменты

- Путаница в понятиях.
- Да-да, у нас есть инструмент для автоматического форматирования кода.

Проблематика: устаревшие/слабые инструменты

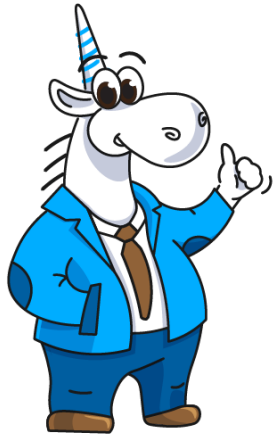
- Достаточно взять FindBugs для Java? Последний релиз был 9 лет назад.
- Ну, хорошо, а SpotBugs? Последний стабильный релиз был 2 года назад.
- Cppcheck для C++?



Актуальность для вашей команды

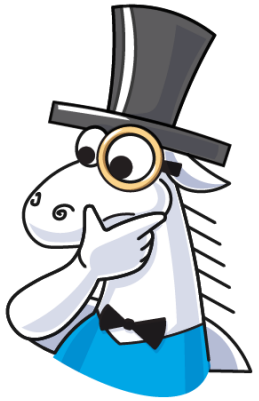
Если вы занимаетесь разработкой безопасного ПО (ГОСТ Р 56939)

- Уже правильные процессы или что-то следует улучшить?
- Используются подходящие инструменты анализа кода?
- На что обращать наибольшее внимание (про критические ошибки)?



Если хотите улучшить процесс разработки

- Вы сможете понять, следует ли что-то менять в инструментарии и процессах.
- ГОСТ можно использовать для вдохновения в улучшении практик, применяемых у вас в разработке.



Актуальность для разработчиков анализаторов

Хорошо описано с точки зрения сертификации

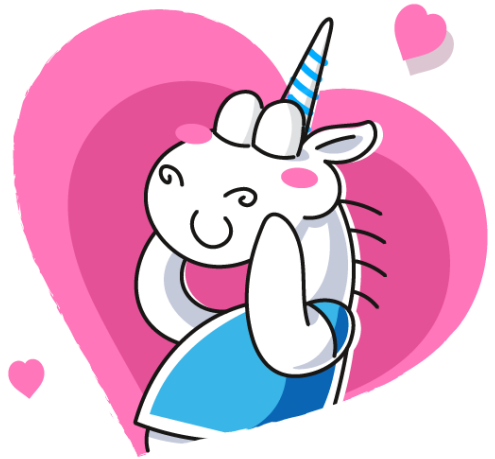
- В состав набора квалификационных тестов должны входить ….
- Для ошибок на наборе тестов статический анализатор должен обеспечивать достижение следующих показателей:
  - ложноположительных срабатываний — не более 50 %;
  - ложноотрицательных срабатываний — не более 50 %.
- Состав набора квалификационных тестов следует регулярно пересматривать.
- Примером открытого набора тестов, частично соответствующего требованиям, является Juliet Test Suite.

Но разработчикам анализатора непонятно, от чего отталкиваться

- На разных тестах можно получить разные характеристики.
- **Получается: “Хорошо делай — хорошо будет”.**
- Честно.
- Но всё-таки чем руководствоваться и в какую сторону развивать анализатор?

В целом

- Появился хороший и полезный ориентир, помогающий создавать мощные инструменты анализа кода.

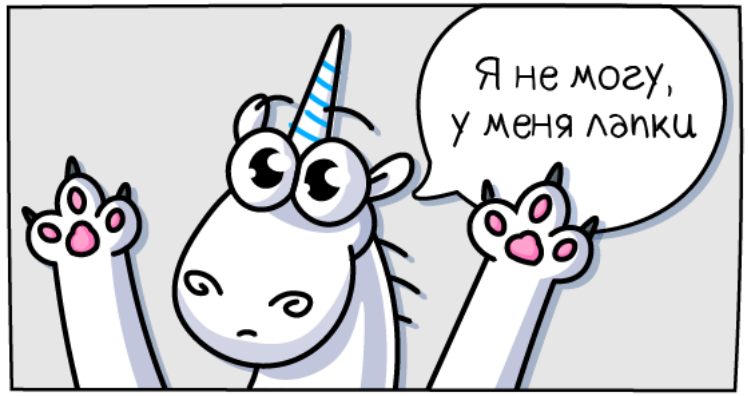


- PVS-Studio:
  - после чтения ГОСТ открыта большая эпик-задача на различные усовершенствования анализатора;
  - изменилась дорожная карта развития PVS-Studio.

Пример улучшений

- Если статический анализатор для поиска ошибок применяет анализ помеченных данных, должна быть предоставлена возможность конфигурации анализа: **должны задаваться процедуры-источники и процедуры-стоки чувствительных данных**.

- В PVS-Studio есть механизм аннотирования функций.
- Но нет возможности задавать источники/стоки.
- Реализуем в одном из ближайших релизов.



Немного конкретных требований для разминки

Статический анализатор должен содержать в документации описание всех типов ошибок

- Для каждого типа ошибки должны быть приведены:
  - описания;
  - возможная причина возникновения;
  - примеры ошибочного кода, для которого выдаётся предупреждение о данном типе ошибки;
  - примеры или рекомендации исправления данного типа ошибки.
- Очевидное требования, но на практике далеко не все анализаторы справляются с документацией.

Cppcheck (описания просто нет)

Auto Variables

A pointer to a variable is only valid as long as the variable is in scope.
Check:
- returning a pointer to auto or temporary variable
- assigning address of an variable to an effective parameter of a function
- returning reference to local/temporary variable
- returning address of function parameter
- suspicious assignment of pointer argument
- useless assignment of function argument

Boolean

Boolean type checks
- using increment on boolean
- comparison of a boolean expression with an integer other than 0 or 1
- comparison of a function returning boolean value using relational operator

SpotBugs (чуть лучше)

FI: Finalizer only nulls fields (FI_FINALIZER_ONLY_NULLS_FIELDS)

This finalizer does nothing except null out fields. This is completely pointless, and requires that the object be garbage collected, finalized, and then garbage collected again. You should just remove the finalize method.

FI: Finalizer nulls fields (FI_FINALIZER_NULLS_FIELDS)

This finalizer nulls out fields. This is usually an error, as it does not aid garbage collection, and the object is going to be garbage collected anyway.

UI: Usage of GetResource may be unsafe if class is extended (UI_INHERITANCE_UNSAFE_GETRESOURCE)

Calling `this.getClass().getResource(...)` could give results other than expected if this class is extended by a class in another package.

Ещё рекомендация по документированию диагностик

- Применяемая в статических анализаторах типизация разнится, что затрудняет соотнесение предупреждений, полученных от разных анализаторов.
- Для облегчения работы с предупреждениями в диагностическую информацию добавляют соотнесение ошибки с одной из популярных систем классификации дефектов безопасности, например с MITRE CWE.
- В описании ошибок также следует указывать их соответствие идентификаторам в системе классификации дефектов безопасности MITRE CWE.

Документация здорового человека (PVS-Studio)

Описание

Пример ошибочного кода

Примеры исправления

Сопоставление с CWE, SEI CERT, OWASP

V772. Calling a 'delete' operator for a void pointer will cause undefined behavior.

Анализатор обнаружил потенциально возможную ошибку в коде, связанную с тем, что оператор 'delete' или 'delete[]' применяется для нетипизированного указателя (void*). Согласно стандарту C++20 (п. п. §7.6.2.8/3) такое применение ведет к неопределенному поведению.

Рассмотрим пример такого кода:

```
class Example
{
  int *buf;

public:
  Example(size_t n = 1024) { buf = new int[n]; }
  ~Example() { delete[] buf; }
};

....
void *ptr = new Example();
....
delete ptr;
....
```

Подобный пример опасен тем, что компилятор в реальности не знает, к каком типу относится указатель 'ptr'. Поэтому, при удалении такого нетипизированного указателя могут произойти различные неприятности, например, может возникнуть утечка памяти: оператор 'delete' не вызовет деструктор объекта типа 'Example', на который ссылается указатель 'ptr'.

Если подразумевалась именно работа с нетипизированным указателем, то перед применением оператора 'delete' ('delete[]') его необходимо привести к изначальному типу, например так:

```
....
void *ptr = new Example();
....
delete (Example*)ptr;
....
```

Иначе, во избежание ошибок, рекомендуется использовать только типизированные указатели совместно с оператором 'delete' ('delete[]'):

```
....
Example *ptr = new Example();
....
delete ptr;
....
```

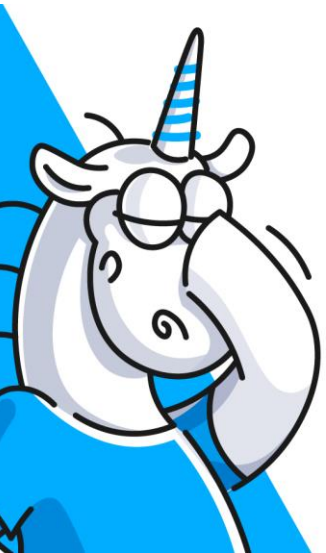
Данная диагностика классифицируется как:

- CWE-758
- CERT-MSC15-C

Требования к обновлениям

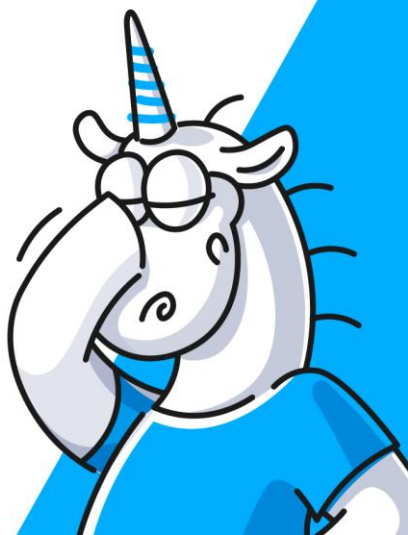
- Следует использовать **регулярно обновляющиеся статические анализаторы**, поддерживающие современные стандарты языков программирования и соответствующие системы сборки.

- Вспоминаем FindBugs, SpotBugs.



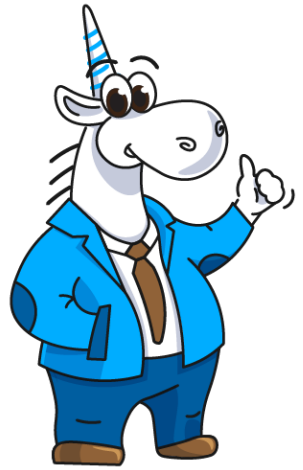
Требования к регулярности

- **Для своевременного выявления и исправления ошибок статический анализ должен выполняться регулярно.**
- Накопление непроанализированных изменений ухудшает качество проводимой экспертизы:
  - постоянно про это рассказываем;
  - боремся с подходом «запустим анализатор перед релизом».
- Рассказ про «релиз два раза в год,
скачиваем на неделю PVS-Studio перед этим».



Требования к специалистам

- В рамках доклада разбирать смысла нет.
- Суть: специалисты должны быть специалистами :)
- Описание поможет в составлении должностных инструкций, вакансий, матриц компетенций и т.д.
- Возможность сопротивляться подходу: «DevOps-ники анализатор прикрутили и гоняют, вот пусть и смотрят логи заодно».



Про выбор анализаторов

- Для соответствия требованиям настоящего стандарта разработчик ПО **должен** использовать в ходе разработки статический анализатор или **набор статических анализаторов** для поиска ошибок.
- Обратите внимание на «**набор анализаторов**». Не требуется выбирать один универсальный. Можно использовать несколько, выбирая наиболее подходящие и мощные решения.

Про выбор анализаторов

- При отсутствии применимых инструментов статического анализа, отвечающих всем требованиям, следует использовать в жизненном цикле ПО инструменты, наиболее полно выполняющие данные требования.
- **Приоритет следует отдавать инструментам, демонстрирующим лучшие ключевые показатели**.

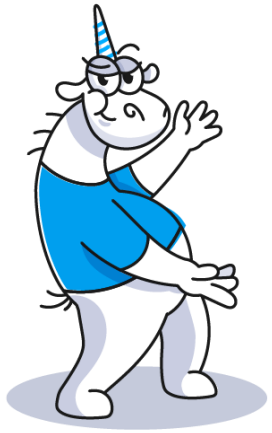
Требования к инструментам статического анализа

- Следует использовать такой статический анализатор (набор анализаторов), чтобы **полный анализ ПО** с используемыми заимствованными компонентами выполнялся не более **двух суток**.

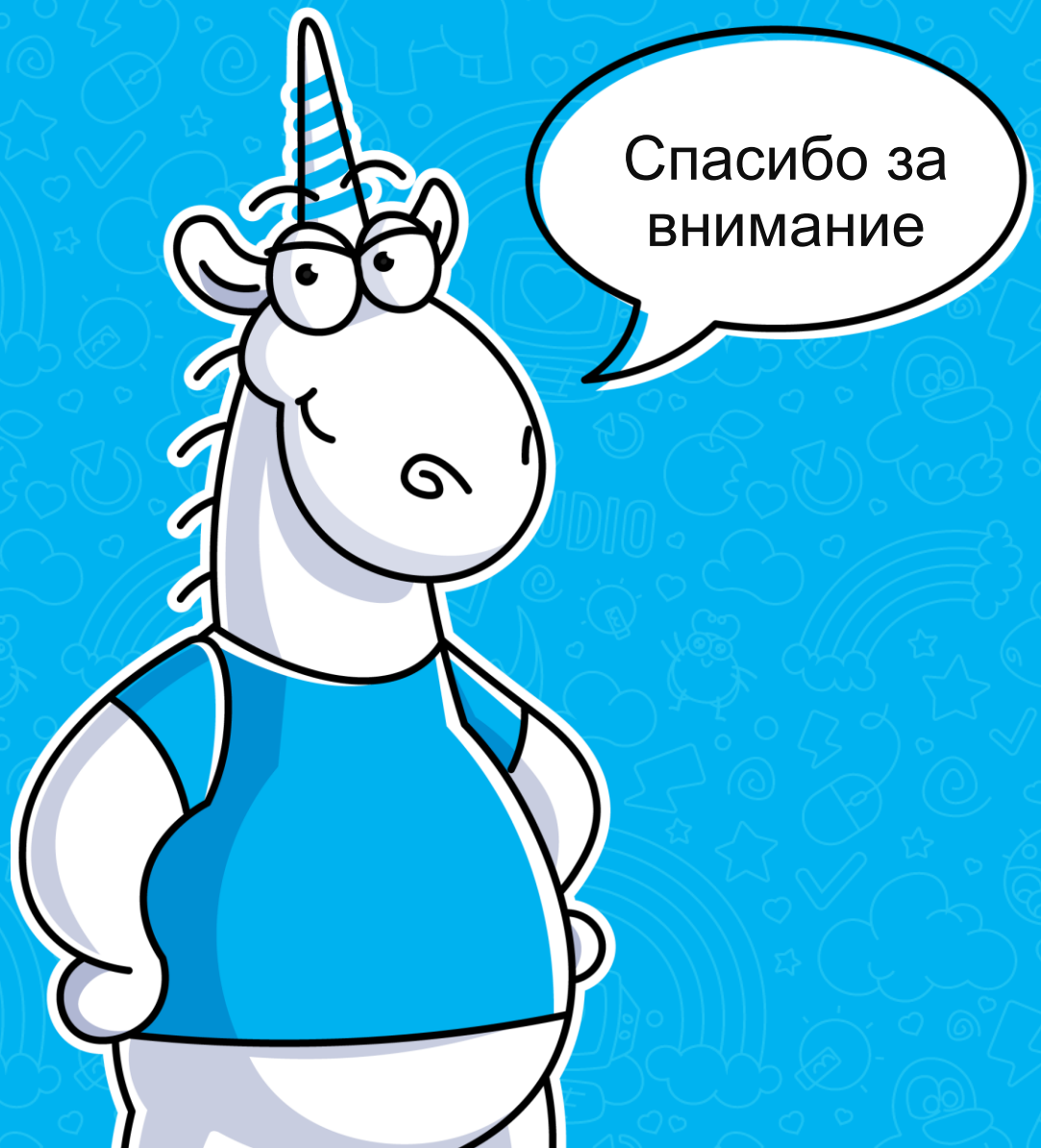
- PVS-Studio:
  - Xeon Gold 5220R, 24 ядра, 2.20 GHz,128 Gb;
  - 122 C и C++ проекта (Visual C++);
  - 1 час 34 минуты.

В следующих вебинарах

В следующих вебинарах



- терминология;
- критические ошибки;
- технологии анализа кода;
- процессы.



Q&A

Андрей Карпов

DevRel