



# ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ


ЛЕКЦИЯ № 10

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.



# СТАТИЧЕСКИЙ АНАЛИЗ КОДА


- Статический анализ кода это процесс выявления ошибок и недочетов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс обзора кода. Остановимся на обзоре кода чуть подробнее.
- Обзор кода (code review) – один из самых старых и надежных методов выявления дефектов. Он заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые могут стать ошибочными в будущем. Также считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев. Если это условие не выполняется, то код должен быть доработан.
- Как правило, обзор кода хорошо работает, так как программисты намного легче замечают ошибки в чужом коде.

- 
- Единственный существенный недостаток методологии совместного обзора кода, это крайне высокая цена. Необходимо регулярно собирать нескольких программистов для обзора нового кода или повторного обзора кода после внесения рекомендаций. При этом программисты должны регулярно делать перерывы для отдыха. Если пытаться просматривать сразу большие фрагменты кода, то внимание быстро притупляется и польза от обзора кода быстро сходит на нет.
  - Получается, что с одной стороны хочется регулярно осуществлять обзор кода. С другой - это слишком дорого. Компромиссным решением являются инструменты статического анализа кода. Они без устали обрабатывают исходные тексты программ и выдают программисту рекомендации обратить повышенное внимание на определенные участки кода. Конечно, программа не заменит полноценного обзора кода, выполняемого коллективом программистов. Однако соотношение польза/цена делает использование статического анализа весьма полезной практикой, применяемой многими компаниями.



Задачи, решаемые программами статического анализа кода можно разделить на 3 категории:

- Выявление ошибок в программах. Подробнее про это будет рассказано ниже.
- Рекомендации по оформлению кода. Некоторые статические анализаторы позволяют проверять, соответствует ли исходный код, принятому в компании стандарту оформления кода. Имеется в виду контроль количества отступов в различных конструкциях, использование пробелов/символов табуляции и так далее.
- Подсчет метрик. Метрика программного обеспечения - это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций. Существует большое количество разнообразных метрик, которые можно подсчитать, используя те ли иные инструменты.
- Есть и другие способы использования инструментов статического анализа кода. Например, статический анализ можно использовать как метод контроля и обучения новых сотрудников, еще недостаточно знакомых с правилами программирования в компании.



Главное преимущество статического анализа состоит в возможности существенной снижении стоимости устранения дефектов в программе. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления.

Другие преимущества статического анализа кода:

- Полное покрытие кода. Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удастся протестировать другими методами. Это позволяет находить дефекты в обработчиках редких ситуаций, в обработчиках ошибок или в системе логирования.
- Статический анализ не зависит от используемого компилятора и среды, в которой будет выполняться скомпилированная программа. Это позволяет находить скрытые ошибки, которые могут проявить себя только через несколько лет. Например, это ошибки неопределенного поведения. Такие ошибки могут проявить себя при смене версии компилятора или при использовании других ключей для оптимизации кода. Другой интересный пример скрытых ошибок приводится в статье "Перезаписывать память - зачем?".
- Можно легко и быстро обнаруживать опечатки и последствия использования Copy-Paste. Как правило, нахождение этих ошибок другими способами является крайне неэффективной тратой времени и усилий. Обидно после часа отладки обнаружить, что ошибка заключается в выражении вида `strcmp(A, A)`. Обсуждая типовые ошибки, про такие ляпы, как правило, не вспоминают. Но на практике на их выявление тратится существенное время.

## Недостатки статического анализа кода

- Статический анализ, как правило, слаб в диагностике утечек памяти и параллельных ошибок. Чтобы выявлять подобные ошибки, фактически необходимо виртуально выполнить часть программы. Это крайне сложно реализовать. Также подобные алгоритмы требуют очень много памяти и процессорного времени. Как правило, статические анализаторы ограничиваются диагностикой простых случаев. Более эффективным способом выявления утечек памяти и параллельных ошибок является использование инструментов динамического анализа.
- Программа статического анализа предупреждает о подозрительных местах. Это значит, что на самом деле код, может быть совершенно корректен. Это называется ложно-позитивными срабатываниями. Понять, указывает анализатор на ошибку или выдал ложное срабатывание, может только программист. Необходимость просматривать ложные срабатывания отнимает рабочее время и ослабляет внимание к тем участкам кода, где в действительности содержатся ошибки.

# PVS-STUDIO

- PVS-Studio – это инструмент для выявления ошибок и потенциальных уязвимостей в исходном коде программ, написанных на языках C, C++, C# и Java.
- Скачать можно отсюда: <https://pvs-studio.ru/ru/pvs-studio/download/>

# УСТАНОВКА И АКТИВАЦИЯ

- Установка на нужную ОС с помощью UI-инсталлятора
- Активация бесплатной версии:

## Данные для активации

Имя	PVS-Studio Free
Серийный номер	FREE-FREE-FREE-FREE

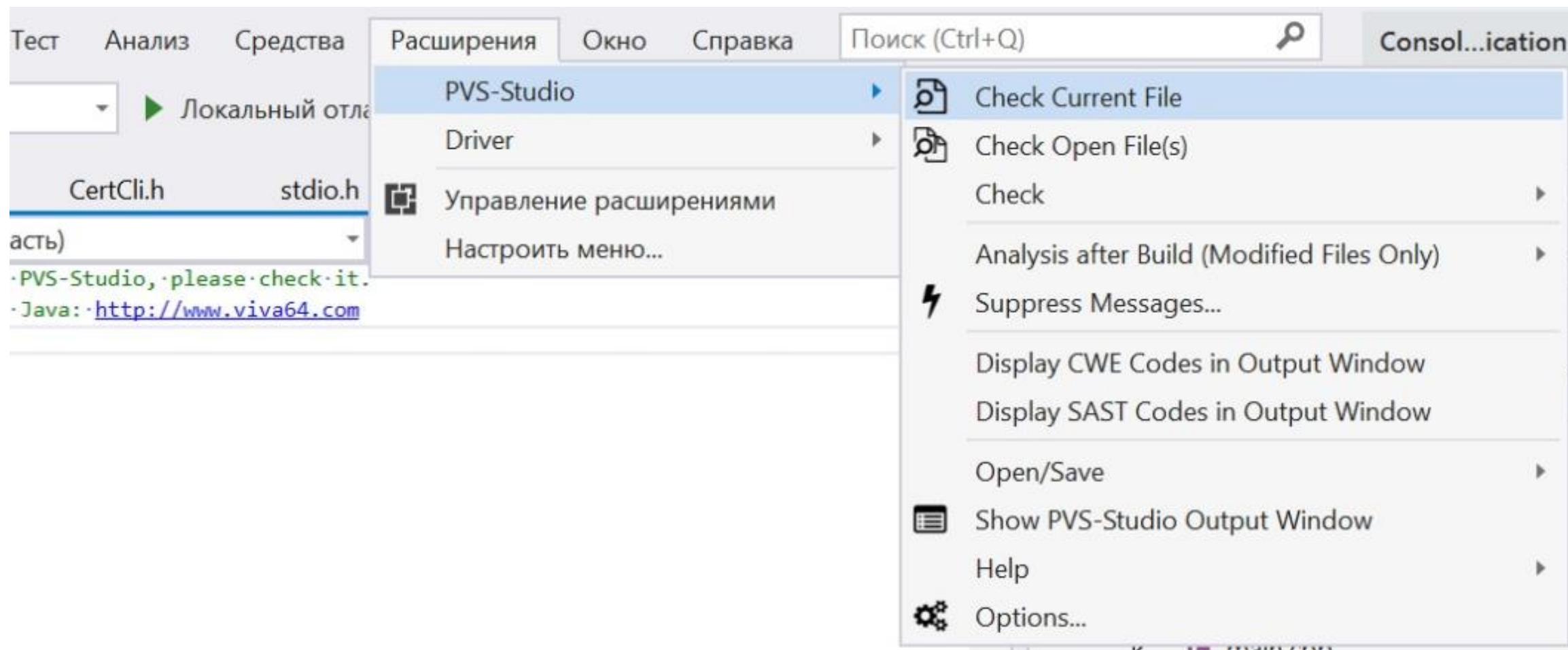
- Активировать PVS-Studio можно:
  - В плагине VisualStudio в верхней панели выбрать "PVS-Studio→Options..." и в открывшемся окне в разделе "PVS-Studio→Registration" ввести данные активации.
  - В утилите Standalone.exe (находится в папке, в которую Вы всё установили) в верхней панели выбрать "Tools→Options..." и в открывшемся окне в разделе "Registration" ввести данные активации.
- Процесс активации с официального сайта: <https://pvs-studio.ru/ru/blog/posts/0457/>



# ПРОВЕРКА ФАЙЛА

- Комментарий для бесплатной лицензии (должен присутствовать для всех проверяемых файлов):

```
// This is an independent project of an individual developer. Dear PVS-Studio, please check it.  
// PVS-Studio Static Code Analyzer for C, C++, C#, and Java: http://www.viva64.com
```



# ПРИМЕР АНАЛИЗА

- Предположим, у нас есть функция

```
#include <stdlib.h>
#include <stdio.h>

int func()
{
    int a;
    char* b;

    if (a == 5) {
        a /= 0;
    } else {
        a *= 2;
    }

    b = (char*) malloc(a * sizeof(char));
    for (int i = 0; i <= a; i++) {
        printf("%c", b[i]);
    }
}
```

- PVS-Studio нашла в ней следующие ошибки:

PVS-Studio		
☰	Fails: 0	▲ ▼   🎯 Best   High: 2   Medium: 4   Low: 0   General Optimization 64-bit ▼
★	Code	Message
☆	<a href="#">V609</a>	Divide by zero.
☆	<a href="#">V547</a>	Expression 'a = 5' is always true.
☆	<a href="#">V591</a>	Non-void function should return a value.
☆	<a href="#">V522</a>	There might be dereferencing of a potential null pointer 'b'. Check lines: 19, 17.
☆	<a href="#">V773</a>	Visibility scope of the 'b' pointer was exited without releasing the memory. A memory leak is possible.

```
5
6 int func()
7 {
8     int a;
9     char* b;
10
11     if (a == 5) {
12         a /= 0;
13     } else {
14         a *= 2;
15     }
16
17     b = (char*) malloc(a * sizeof(char));
18     for (int i = 0; i <= a; i++) {
19         printf("%c", b[i]);
20     }
21 }
```

80 %  Проблемы не найдены.

PVS-Studio



Fails: 0



Best

High: 2

Medium: 4

Low: 0



Code

Message



[V609](#)

Divide by zero.



[V547](#)

Expression 'a = 5' is always true.

V609. Possible division or mod by zero. ➔ ✕

pvs\_test.cpp

main.cpp

SCCOIDExt\_Functions.cpp

CertCli.h

URL-адрес: <https://pvs-studio.com/en/docs/warnings/v609/print/>

## V609. Possible division or mod by zero.

[PVS-Studio messa](#)

The analyzer has detected a situation when division by zero may occur.

Consider this sample:

```
for (int i = -10; i != 10; ++i)
{
    Foo(X / i);
}
```

While executing the loop, the 'i' variable will acquire a value equal to 0. At this moment, an operation of division by zero will occur. To fix it we need to specifically handle the case when the 'i' iterator equals zero.

This is the correct code:

```
for (int i = -10; i != 10; ++i)
{
    if (i != 0)
        Foo(X / i);
}
```

This diagnostic is classified as:

PVS-Studio



Fails: 0



Best

High: 2

Medium: 4

Low: 0

General

Optimization

64-bit



Code

Message



V609

Divide by zero.



```
6 int func()
7 {
8     int a;
9     char* b;
10
11     if (a == 5) {
12         a /= 0;
13     } else {
14         a *= 2;
15     }
16
17     b = (char*)malloc(a * sizeof(char));
18     for (int i = 0; i <= a; i++) {
19         printf("%c", b[i]);
20     }
21 }
```

80 % Проблемы не найдены.

PVS-Studio

≡ Fails: 0 ▲ ▼ Best High: 2 Medium

★	Code	Message
★	<a href="#">V609</a>	Divide by zero.
★	<a href="#">V547</a>	Expression 'a = 5' is always true.
★	<a href="#">V591</a>	Non-void function should return a value.

```
16
17     b = (char*)malloc(a * sizeof(char));
18     for (int i = 0; i <= a; i++) {
19         printf("%c", b[i]);
20     }
21 }
```

80 % Проблемы не найдены.

PVS-Studio

≡ Fails: 0 ▲ ▼ Best High: 2 Medium: 4 Low: 0

★	Code	Message
★	<a href="#">V609</a>	Divide by zero.
★	<a href="#">V547</a>	Expression 'a = 5' is always true.
★	<a href="#">V591</a>	Non-void function should return a value.

```
16
17     ... b = (char*)malloc(a * sizeof(char));
18     ... for (int i = 0; i <= a; i++) {
19         ... printf("%c", b[i]);
20     ... }
21 }
```

80 %

✓ Проблемы не найдены.

PVS-Studio



Fails: 0



Best

High: 2

Medium: 4

Low: 0

General

Optim

★	Code	Message
★	<a href="#">V609</a>	Divide by zero.
★	<a href="#">V547</a>	Expression 'a = 5' is always true.
★	<a href="#">V591</a>	Non-void function should return a value.
★	<a href="#">V522</a>	There might be dereferencing of a potential null pointer 'b'. Check lines: 19, 17.
★	<a href="#">V772</a>	Visibility scope of the 'b' pointer was exited without releasing the memory. A me



URL-адрес: <https://pvs-studio.com/en/docs/warnings/v522/print/>

## V522. Possible null pointer dereference.

[PVS-Studio r](#)

- [malloc, realloc](#)
- [Additional Settings](#)

The analyzer detected a fragment of code that might cause using a null pointer.

Let's study several examples the analyzer generates the V522 diagnostic message for:


```
if (pointer != 0 || pointer->m_a) { ... }
if (pointer == 0 && pointer->x()) { ... }
if (array == 0 && array[3]) { ... }
if (!pointer && pointer->x()) { ... }
```

In all the conditions, there is a logical error that leads to dereferencing of the null pointer. The error may be introduced into the code during code refactoring or through a misprint.

Correct versions:

```
if (pointer == 0 || pointer->m_a) { ... }
if (pointer != 0 && pointer->x()) { ... }
if (array != 0 && array[3]) { ... }
```

PVS-Studio






☰ Fails: 0 ▲ ▼ |  Best | High: 2 | Medium: 4 | Low: 0 | General Optimization 64-bit ▼

★ Code	Message
☆ <a href="#">V609</a>	Divide by zero.
☆ <a href="#">V547</a>	Expression 'a = 5' is always true.
☆ <a href="#">V591</a>	Non-void function should return a value.
☆ <a href="#">V522</a>	There might be dereferencing of a potential null pointer 'b'. Check lines: 19, 17.

```
6 int func()
7 {
8     int a;
9     char* b;
10
11     if (a = 5) {
12         a /= 0;
13     } else {
14         a *= 2;
15     }
16
17     b = (char*) malloc(a * sizeof(char));
18     for (int i = 0; i <= a; i++) {
19         printf("%c", b[i]);
20     }
21 }
```

80 %  Проблемы не найдены.

PVS-Studio

 Fails: 0    Best High: 2 Medium: 4 Low: 0 General Optimization 64-bit 

★ Code	Message
★ <a href="#">V609</a>	Divide by zero.
★ <a href="#">V547</a>	Expression 'a = 5' is always true.
★ <a href="#">V591</a>	Non-void function should return a value.
★ <a href="#">V522</a>	There might be dereferencing of a potential null pointer 'b'. Check lines: 19, 17.
★ <a href="#">V773</a>	Visibility scope of the 'b' pointer was exited without releasing the memory. A memory leak is possible.

URL-адрес: <https://pvs-studio.com/en/docs/warnings/v773/print/>

## V773. Function exited without releasing the pointer/handle. A memory/resource leak is possible.

The analyzer detected a potential memory leak. This situation occurs when memory allocated by using 'malloc' or 'new' remains unreleased after use.

Consider the following example:

```
int *NewInt()
{
    int *p = new int;
    ....
    return p;
}

int Test()
{
    int *p = NewInt();
    int res = *p;
    return res;
}
```

In this code, memory allocation is put into a call to another function. Therefore, the allocated storage needs to be released accordingly after the call.

This is the fixed code, without the memory leak:

```
int *NewInt()
{
    int *p = new int;
    ....
    return p;
}

int Test()
{
    int *p = NewInt();
    int res = *p;
    delete p;
    return res;
}
```

# ФОРМИРОВАНИЕ ОТЧЁТА ОБ ОШИБКАХ

Получить отчёт PVS-Studio можно:

- С помощью плагина VisualStudio: на верхней панели выбрать "PVS-Studio→Check" и проверять нужные файлы/проекты.
- С помощью утилит, которые находятся в папке, куда устанавливалась PVS-Studio

# КАК ПОЛЬЗОВАТЬСЯ УТИЛИТАМИ ИЗ КОНСОЛИ ДЛЯ СОСТАВЛЕНИЯ ОТЧЁТА?

- Для составления файла .plog минимальный набор аргументов:

`PVS-Studio_Cmd.exe -t "/path/to/project"`

- Где /path/to/project имеет расширение .sln/.csproj/.vcxproj

- Файл .plog будет создан рядом с /path/to/project (для изменения места сохранения используется -o "/path/to/plog") Для человеко-читаемого отчёта нужно сконвертировать .plog в нужный формат (мне нравится FullHtml):

`PlogConverter.exe "/path/to/plog" -t FullHtml`

- Ссылка на описание html-отчётов: <https://pvs-studio.ru/ru/blog/posts/0539/>

## ДОП.ССЫЛКИ

- Официальный сайт PVS: <https://pvs-studio.ru/ru/>
- Блог PVS на habr: <https://habr.com/ru/company/pvs-studio/profile/>
- Написан скрипт на python: `PrepareFilesForPVS.py`, для добавления/удаления строки с комментарием в файлы исследуемого проекта.
- У скрипта есть help, но если вкратце:
  - `--mode=0` добавляет комментарий
  - `--mode=1` удаляет комментарий
  - `--path="."` - строка с директорией (полный путь), внутри которой нужно внести изменения в файлы
  - `--isFile=0` работает со всеми файлами директории
  - `--isFile=1` работает с конкретным файлом, полный путь до которого нужно передать в `--path`
  - По умолчанию скрипт будет добавлять комментарии к файлам в "текущей директории".