Building Cppcheck

What We Learned From 17 Years of Development

DANIEL MARJAMÄKI

Daniel Marjamäki

- Creator of Cppcheck
  - Cppcheck looks for bugs in your C/C++ code
- Live in Sweden
- Have worked professionally with C/C++ for almost 20 years

What will I say

- Chronological walk through
- Learnings

Before I started Cppcheck

- Feeling that quality was not very good
- Our compilers would not find obvious bugs

```
int buf[10];
....
buf[20] = 0;
```

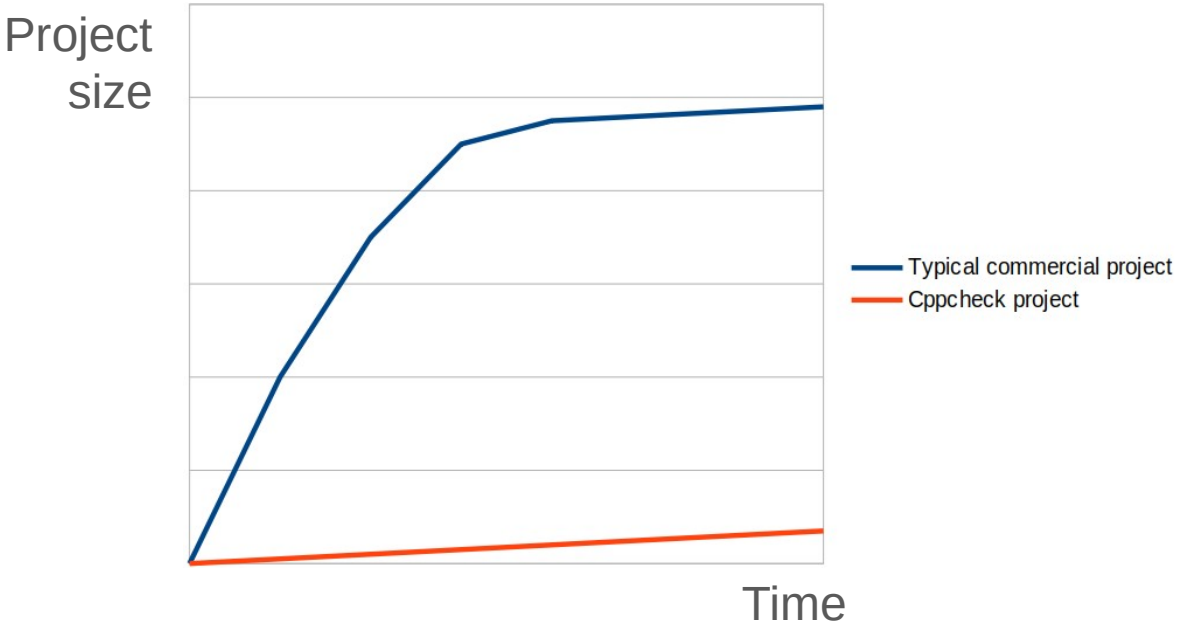
I was looking for off the shelf static analysis solutions.

- Major problem: very little support for our toolchains.
- False positives

Hobby project preconditions

- Time
- Knowledge
- Perseverance

Planning


Project size
Time
Typical commercial project
Cppcheck project

First experiment

- Started experimenting with perl and regular expressions

Sourceforge

I registered the Cppcheck project on Sourceforge on may 7th, 2007.

First commit:

- Source files, only 1
  - Main.cpp
    - “Infrastructure” => 440 lines
    - Checkers => 665 lines

Revision 1: infrastructure

- token list
  - No preprocessing except: `#include “somefile.h”`
- function to match tokens

Revision 1: data

```
std::vector<std::string> Files;

struct TOKEN
{
    unsigned int FileIndex;
    char *str;
    unsigned int linenr;
    struct TOKEN *next;
};
struct TOKEN *tokens, *tokens_back;
```

Revision 1: checkers

- Look for X >= ‘0’ && X <= ‘9’ and recommend to use ‘isdigit’ instead.
- Warn if memset is used on class
- Warn about includes that are not needed
- Redundant condition: if (ptr) delete ptr;
- Member variable that is not initialized in constructor

Using token list

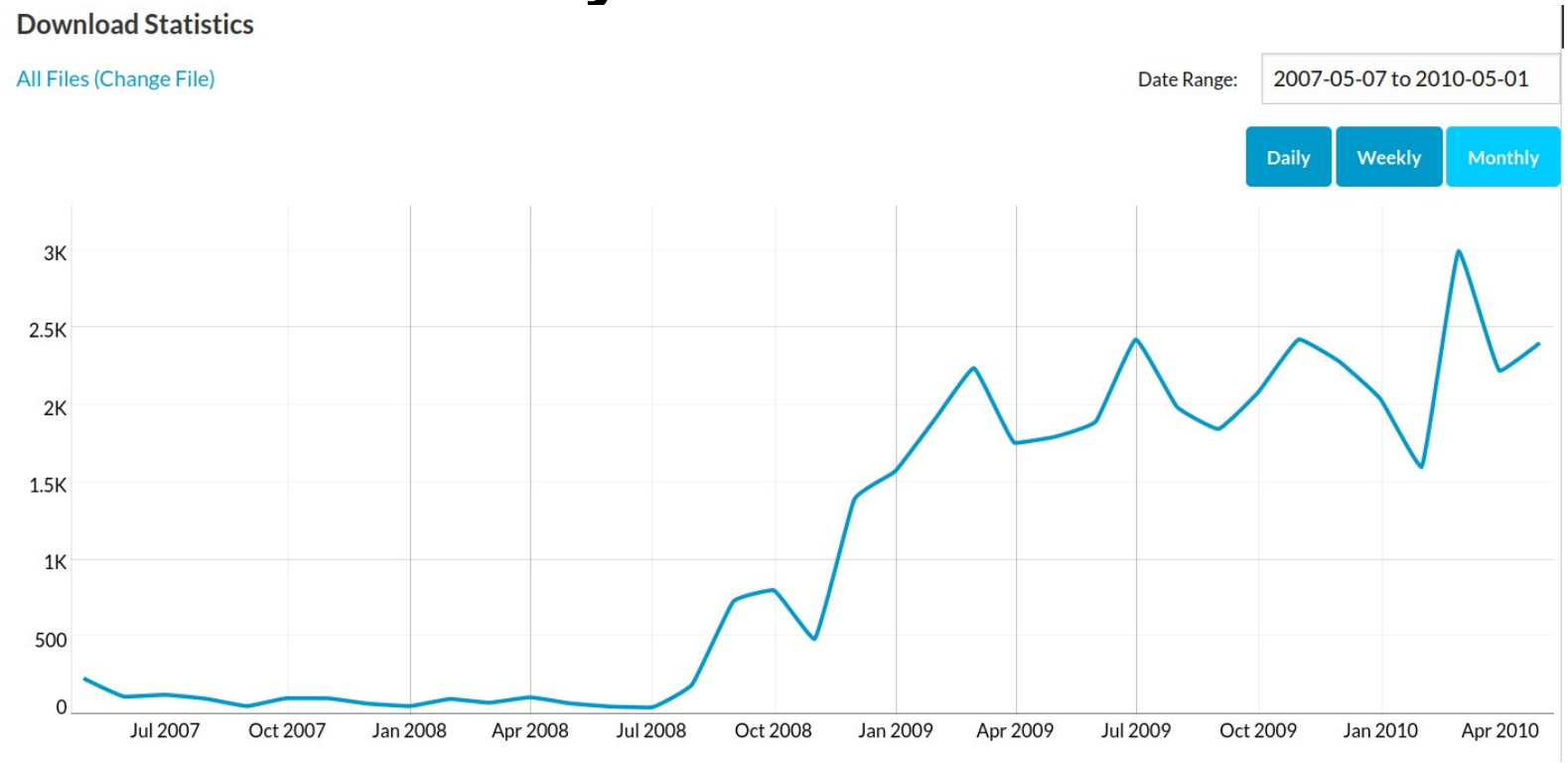
Why did I do this.

- Pros and cons:
  - Compiler agnostic
  - For well written code I got few false positives
  - But: False negatives

Commits in first year

May	97
June	33
July	16
August	1
September	0
October	4
November	0
December	0
January	4
February	32
March	37
April	29

Downloads first 3 years



Reports/bugfixes

Open source projects started to receive reports/bugfixes

Easy-to-use can lead to problems

Preprocessor

2009

- String based preprocessing
- Extract configurations

```
#ifdef X
a = b / 0;
#else
a = c / 0;
#endif
```

Symbol database

2010

- Scopes
- Variables
- Types
- Functions

Instantiating templates

2011

Match compiler

2012

Original code

```
if (Token::Match(tok, “%var% = 0”))
```

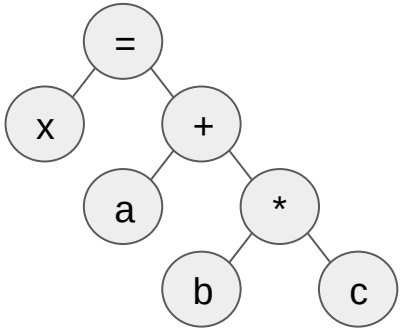
Transformed code

```
static inline bool match1234(const Token* tok) {
   if (!tok || !(tok->varId() != 0))
       return false;
   tok = tok->next();
   if (!tok || !((tok->tokType() == Token::eAssignmentOp) && tok->str() == "="))
       return false;
   tok = tok->next();
   if (!tok || !(tok->str() == "0"))
       return false;
   return true;
```

Syntax tree

In 2012 we started implementing a syntax tree

X = a+b*c;



Dataflow

In 2014 we implemented a generic dataflow

Rewritten preprocessor

2015

Simplecpp, it’s open source and BSD licensed

Simple library: 1 header and 1 source file.

https://github.com/danmar/simplecpp

ValueType

2015

```
void foo(int *ptr)
{
    x = *(14 + ptr);
}
```

```
* 'signed int'
`-+ 'signed int *'
  |-14 'signed int'
  `-ptr 'signed int *'
```

An interesting bug that we have found

- X.org announcement in January 2014:

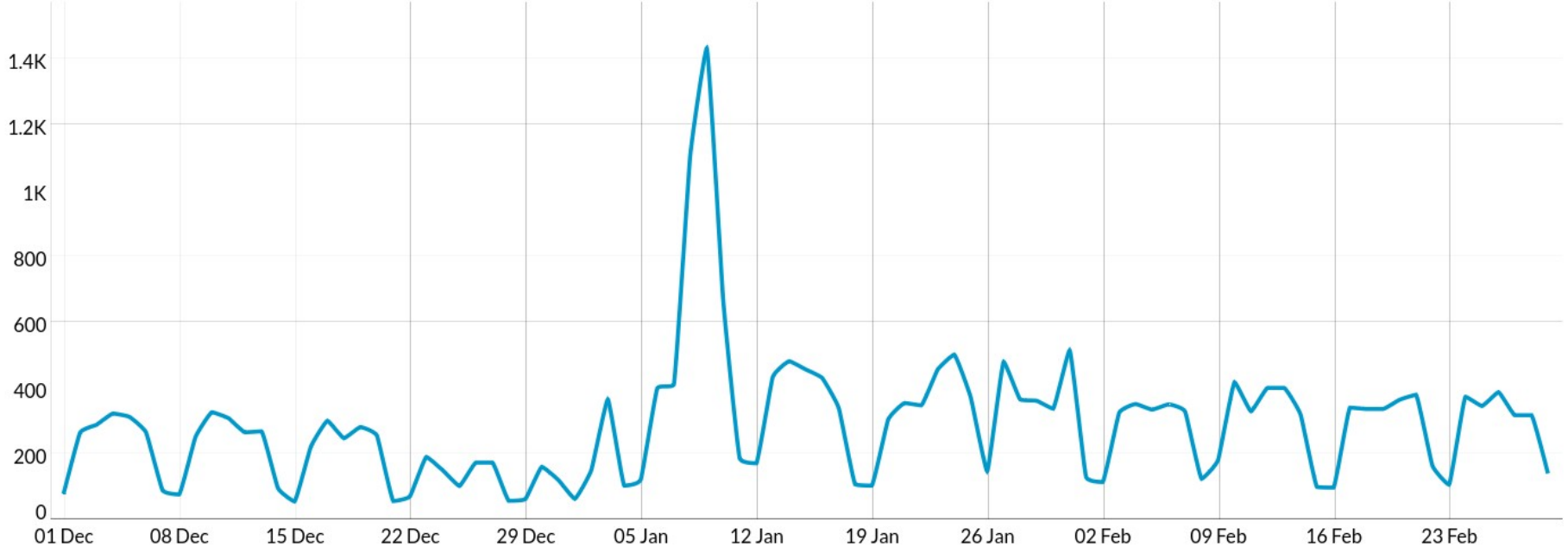
  As libXfont is used to read user-specified font files in all X servers distributed by X.Org, including the Xorg server which is often run with root privileges or as setuid-root in order to access hardware, this bug may lead to an unprivileged user acquiring root privileges in some systems.

- Affected all X Windows releases in latest 22 years

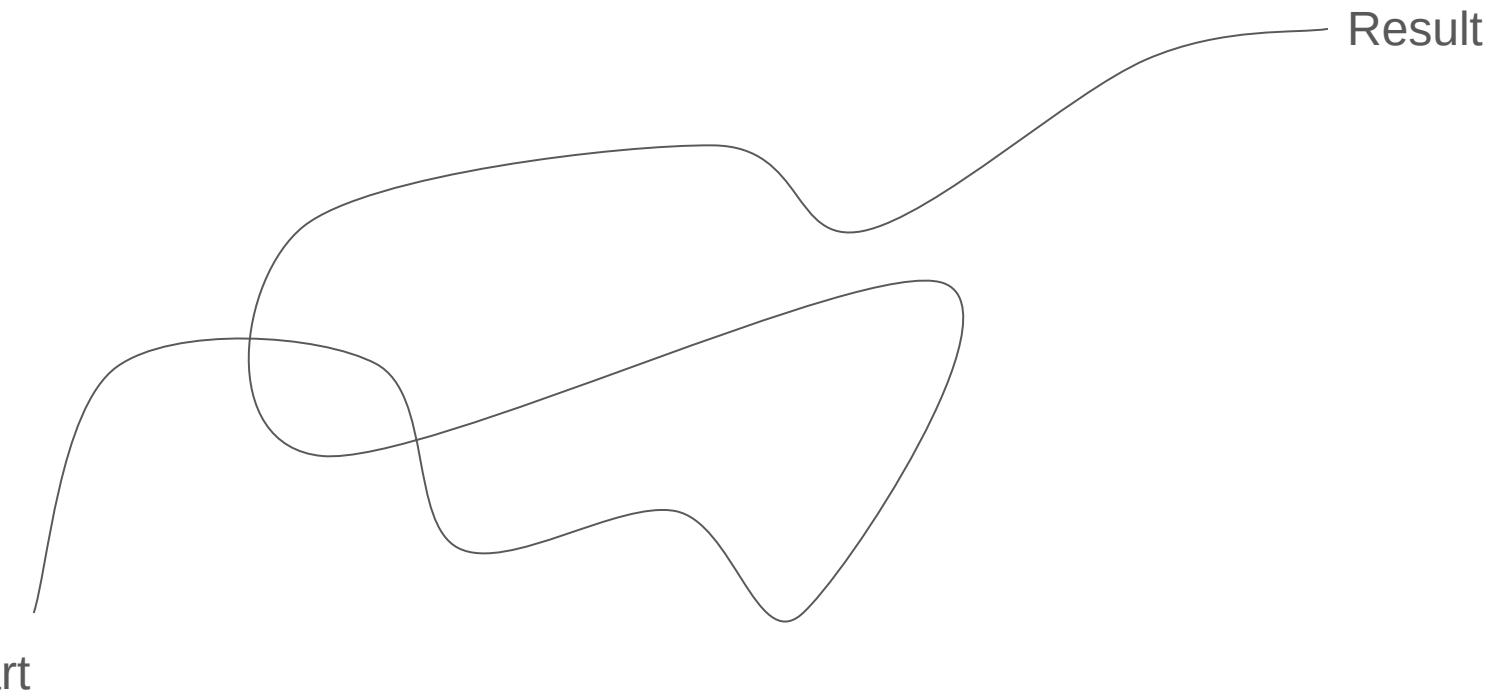
```
char charName[100];

if (sscanf((char *) line, "STARTCHAR %s", charName) != 1) {
   bdfError("bad character name in BDF file\n");
   goto BAILOUT; /* bottom of function, free and return error */
}
```

22-year old bug in X Windows



Cppcheck Development



Learning: Canonicalizations

Split up declarations:

```
int a, b=3;  =>  int a; int b; b=3;
```

Simplify mathematical expressions

```
a = 3; buf[a] = 0;  =>  buf[3] = 0;
```

Etc..

- Checker:
  - Only token list
  - Has parsing

Learning: make warnings as clear as possible

- Avoid misunderstandings
  - Misunderstand motivation
  - Different valueflow analysis

```
void foo(int i) {
  int buf[10];
  buf[i] = 0;
  ....
  if (i == 20) {}
}
```

- Our philosophy is to rely on configuration rather than hard coding

Learning: no false positives

Changed philosophy

- I wanted it to be simple to use => No configuration => unknown types, functions, variables
  - No false positives, avoid false negatives

Help us improve Cppcheck by reporting false positives instead of suppressing

Learning: technically correct errors are sometimes considered false positives

- We report some undefined behavior as “portability” problems
- Syntax error:

```
#ifdef SOME_MACRO
void foo()
#endif
{
    /**/
}
```

What checkers did I want to implement

KISS, keep it simple stupid. As far as I remember:

- I wanted to find common bugs
- simple to detect
- No false positives

Learning:

Not many checkers meet all those criteria =>

- real mistakes that have been made in real code
- Simple analysis is good
- No false positives

Synthetic test cases

I do not like synthetic test suites. Typically no heuristics.

We write checkers based on real mistakes seen in real code using heuristics.

Testing: Check Debian

- To test Cppcheck
- We check the Debian code continuously using a “BOINC” like system.
  - We have a server provides URLs to source packages and maintains a database of results
  - Clients perform analysis and uploads results

Reports

- Crash report
- Diff report
- HEAD report
- Time report (regressed)
- Time report (slowest)

What does it find in Debian source code

- Buffer overflows => ~1900
- Uninitialized variables => ~16000
- Null pointer dereference => ~8000
- Number of “error” => 94275

- If everybody used Cppcheck

Wanted to work full time on Cppcheck

I have wanted to work full time on Cppcheck

- I like working on this project
- No resources ; just “good enough”

Solution: started company

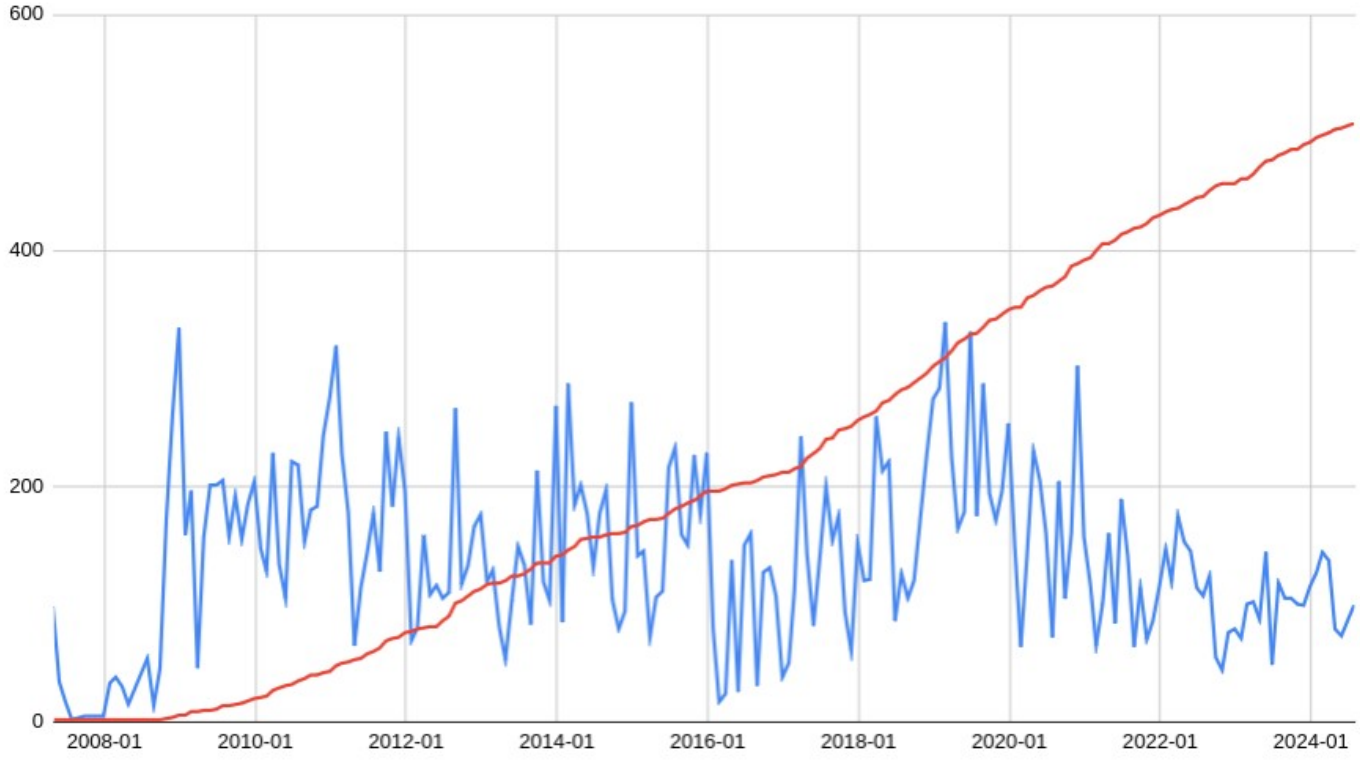
Open source

- Free to use
- Active community
- Compiler agnostic
- Fast
- Accurate
- Easy to use
- Widely used

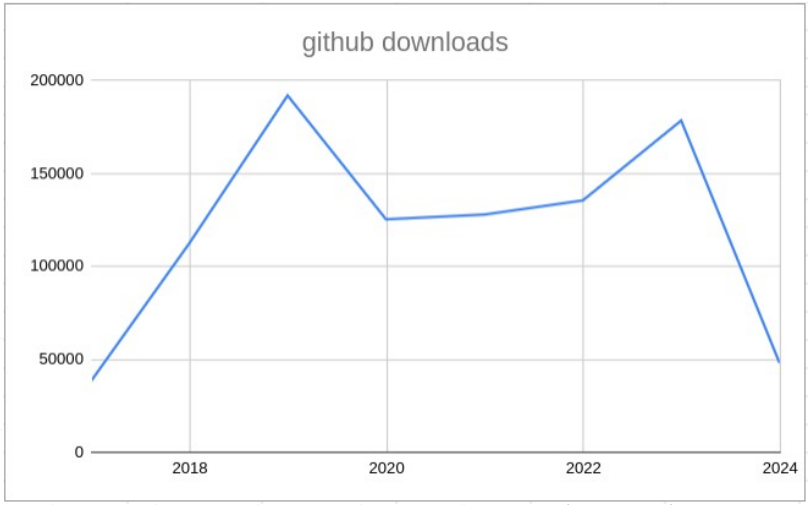
Commercial

- Coding standards:
  - Autosar, Misra C/C++, Cert C/C++
- Extra checkers
- TÜV certified
- Support

contributors commits per month



Downloads


github downloads
200000
150000
100000
50000
0
2018
2020
2022
2024

Sourceforge / month
12K
10K
8K
6K
4K
2K
0
Jan 2008
Jan 2009
Jan 2010
Jan 2011
Jan 2012
Jan 2013
Jan 2014
Jan 2015
Jan 2016
Jan 2017
Jan 2018
Jan 2019
Jan 2020
Jan 2021
Jan 2022
Jan 2023
Jan 2024

Main contributors


danmar
8 533 commits


orbitcowboy
2 407 commits


PKEuS
1 760 commits


chrchr-github
1 276 commits


IOBYTE
1 229 commits


firewave
1 009 commits


pfultz2
950 commits


kimmov
915 commits


amai2012
860 commits


aggro80
781 commits


versat
545 commits


Dmitry-Me
454 commits


matthiaskrgr
327 commits


php-coder
230 commits


XhmikosR
224 commits


simartin
193 commits

Summary

- Started as a hobby project. No big ambitions I couldn’t dream for this.