Clean Code!
Horrible Performance?

SANDOR DARGO

Cppcon
The C++ Conference

2025

September 13 - 19

Who Am I?

Sándor DARGÓ

- Senior Engineer at **Spotify**
- Enthusiastic blogger: sandordargo.com
- Curious oenophile
- Fortunate father of two

Have you ever seen engineers prematurely optimizing their code?

Have you ever prematurely optimized your code?

Did it feel great?

Did it bring more value to the business?

Why this talk?

My earlier talk - *Why clean code is not the norm* - lead to discussions about performance

I don’t like absolutes but too many people talk in absolutes

I think engineering is more nuanced

Agenda

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

What is clean code?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Clean code is *NOT* a closed set of rules

The book Clean Code != The concept of clean code

Depends on the language

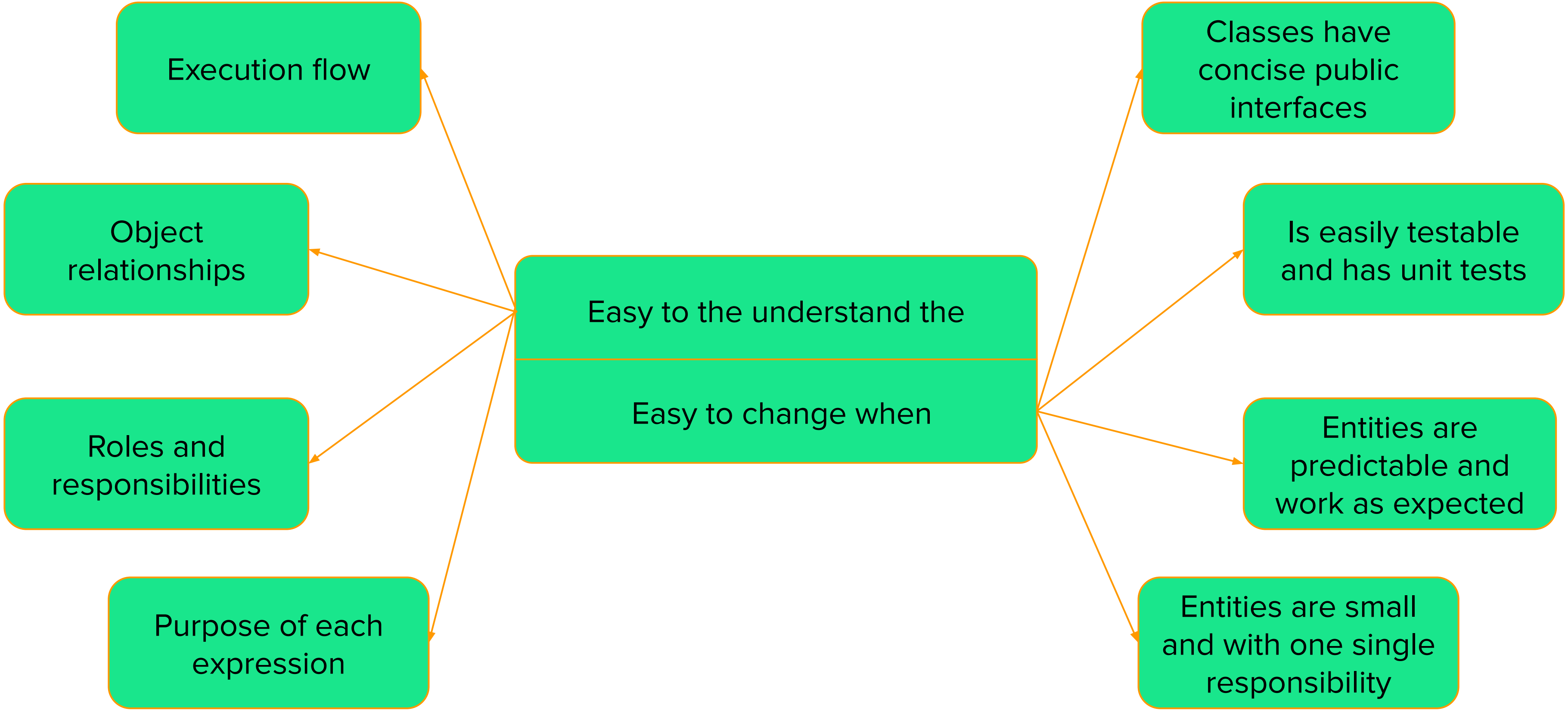
Depends on the context

Changes over time

Not a silver bullet

Then what is clean code?

Code that is easy to understand and easy to change



Easy to the understand the

- Execution flow
- Object relationships
- Roles and responsibilities
- Purpose of each expression

Easy to change when

- Classes have concise public interfaces
- Is easily testable and has unit tests
- Entities are predictable and work as expected
- Entities are small and with one single responsibility

Clean code is part of software quality!

But is high quality software important?

Quality isn’t so important when you

Build a proof of concept

Search a product market fit

Must deliver (fast) to survive

But what if you plan for a longer term?

What is software quality?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

What is quality?

Quality is undefinable

You recognize it when you see it

Software quality has several definitions

It might even be a meaningless term

According to Derek Jones (The aura of software quality)

> *"People in industry are very interested in software quality, and sometimes they have the confusing experience of talking to me about it. My first response, on being asked about software quality, is to* ***ask what the questioner means by software quality****. After letting them fumble around for 10 seconds or so, trying to articulate an answer,* ***I offer several possibilities*** *(****which they are often not happy with****).* ***Then I explain how "software quality" is a meaningless marketing term.*** *This leaves them confused and unhappy. People have a yearning for software quality which makes them easy prey for the snake-oil salesmen."*

CISQ defines the 4 pillars of structural quality

Consortium for Information & Software Quality ™

Security

Reliability

Maintainability

Performance efficiency

Clean code is part of *software quality*!

Helps reducing the number of bugs (**Security**, **Reliability**, **Maintainability**)

Helps reducing the time to fix bugs (**Reliability**, **Maintainability**)

Decreases maintenance costs (**Maintainability**)

CISQ defines the 4 pillars of structural quality

Consortium for Information & Software Quality ™

Security

Reliability

Maintainability

Performance efficiency

What about performance?

Where do we use C++?

What is clean code?

What is software quality?

Where and why do we use C++?

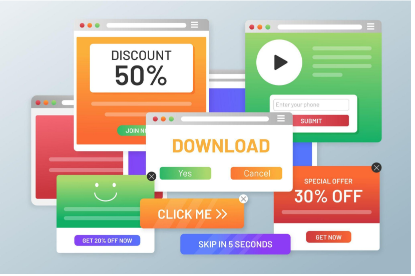
What is software performance?

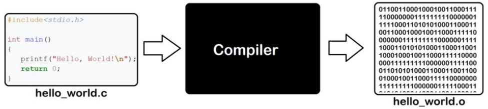
Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

C++ is just everywhere





```c
#include<stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

hello_world.c

Compiler

```
0110011000100010011000111
1100000001111111110000001
1111000110101010001100011
0011000100010011000111110
0000001111111110000001111
1000110101010001100011001
1000100010011000111110000
0001111111110000001111100
0110101010001100011001100
0100010011000111110000000
1111111110000001111100011
```

hello_world.o

Why do we use C++?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Because it’s legacy!

It’s been with us since the 80s

Tons of (often complex and critical) existing code

Rewrites rarely make sense even if C++ is not needed

- Loss of time

- Problem of existing ecosystem

- Problem of knowledge

C++ is also evolving!

Since C++11, stable and steady evolution

A new standard every 3 years

Safer and more expressive code

Not owned by a single corporation!

C++ has an economic advantage!

It’s fast!

Good for performance critical apps!

It uses little space!

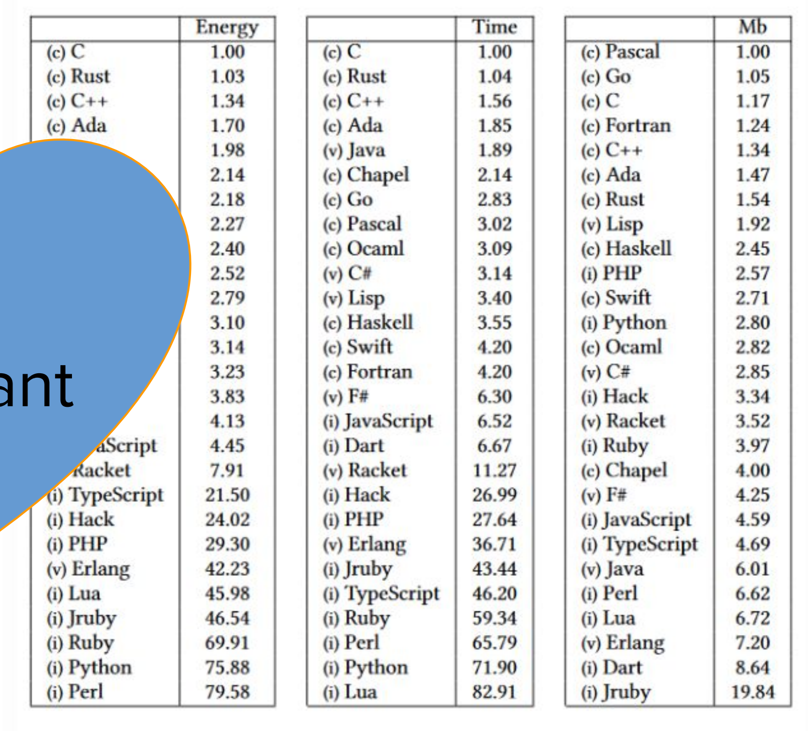
Good for embedded

For anything that has to be distr

Consumes less energy!

Good for everyone!

C++ is performant



	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
[illegible]	1.98
[illegible]	2.14
[illegible]	2.18
[illegible]	2.27
[illegible]	2.40
[illegible]	2.52
[illegible]	2.79
[illegible]	3.10
[illegible]	3.14
[illegible]	3.23
[illegible]	3.83
[illegible]	4.13
aScript	4.45
Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

What is software performance?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Performance is...

A non-functional property, like reliability or usability

Often listed as a requirement — but rarely defined

A measure of efficiency — like cost, time to deliver

Like quality, it's context-dependent and multi-dimensional

Sufficient performance ensures that the software can achieve its functional requirements in a **reasonable** timespan

Performance != Big-O notation

Language is important

Hardware is important

I/O can - and will - dominate

Moreover, it’s not only about execution speed

Dimensions That Actually Matter

Response time: how fast it responds

Throughput: how many requests it can handle

Resource utilization: how efficiently it uses system resources

Scalability: ability to maintain performance levels as the workload increases

Capacity: maximum load or number of users before severe performance degradation

Stability: ability to perform consistently over time without crashes, memory leaks, or other issues

The Real Enemies of Performance

Inefficient algorithms

Unmeasured assumptions

Abstraction overload

Bad data access patterns

Network & I/O bottlenecks

Is clean code actually slower?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Clean code is actually slower

Clean code is actually slower than what?

Clean code is not the fastest of all possible codes

Clean code is optimization for maintainability

It’s not the most efficient / fastest code

It’s not the smallest code either

But will it **harm** performance?

Optimization is always about compromises

Optimization is always about compromises

Optimize for readability

You will hurt runtime performance

You will hurt binary size

Optimize for runtime performance

You will hurt readability

You might hurt portability and binary size

Optimize for binary size

You might hurt readability and portability

You will hurt runtime performance

Should we drop clean code for faster code?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Sometimes every bit of performance matters

Every 100ms increase in page load time costs Amazon 1% in sales

For every additional second of page load time, 10% of BBC users leave

A broker could lose $4 million/ms in revenue, if a competitor is 5 ms ahead

An extra 500ms in search page generation time, dropped Google’s traffic by 20%

Akamai found that every 100ms delay in a website load time drops the conversion rate by 7%

Performance is crucial for Spotify too when you

Open the app

Start playing a track

Tap on any button

Otherwise, there are more important aspects

Small binary

- Thinking about low bandwidth
- And embedded devices

Maintainable code

- We’re almost 20 years old!
- With >7k employees!

But even when performance matters, the first answer is not low-level code optimization

Optimize for maintainability first, then for performance if needed

Start by writing clean code, clean architecture and later deal with performance - if needed

Algorithmic complexity isn’t always so important

My very first project back in 2013 suffered from

Slow speed

Immense memory leak

Speed or operability was the bigger problem?

Some said speed, culprit was an O(n^4) “algo”

I wasn’t really bothered by that piece of nice code…

I measured and measured

It took a fraction of the execution time

"The overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used" - Amdahl's law

You can often ignore slow code if it’s not on the hot path

The key was to use a 3rd party library properly

Some startup / cleanup functions were called too often

Calling them at the right time solved all the imminent issues

Reading documentation and using libraries as they are intended to **is** more **important** than writing fast code...

How to support 15x traffic?

Instead of making the code “faster”:

- Make your system more scalable
- Improve operability
- Improve resource utilization
- Finetune network settings

Improve operability

Remove memory leaks

Fix core dumps

Remove as much UB as possible

Consolidate external requests

Optimize orders of magnitude slower network/db calls first

If the API offers some batching option, use it!

If not, try to batch it on call site

Think about (local) caching

Think about anything that reduces the number of calls you have to make

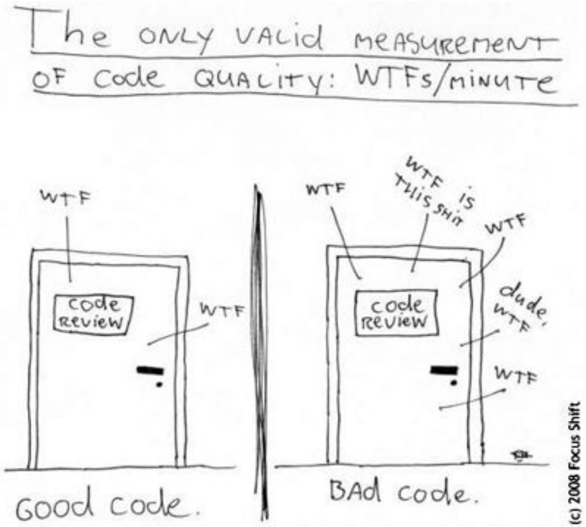
But what about all those costly language features?

Will the common enemies ruin your performance?

Unnecessary heap allocations?

Copying large objects unnecessarily?

Unnecessary virtual functions?



You won't have many of those if your code is clean

"Some" usage will not ruin your code

Optimization efforts must be concentrated

Overall performance improvement gain in a single part is limited!

Follow best practices, keep your code clean you’ll have a reasonable performance

Start optimizing the hot path

Leave the non-hot path clean as long as you prove else

Performance is not l’art pour l’art

Many devs in a code review

Software engineering is not about performance

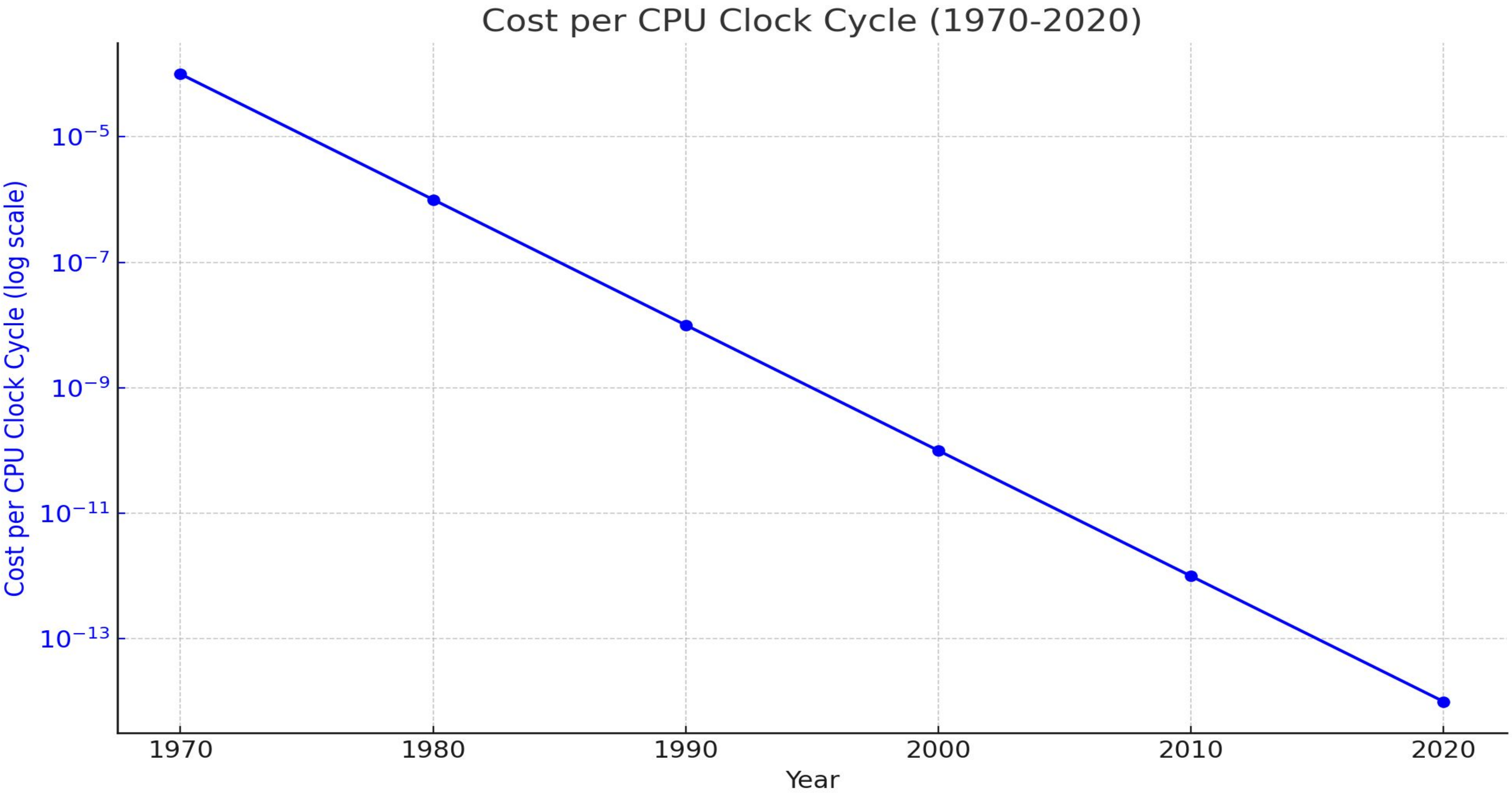
Engineering is in the crossroads of defining and recognizing what is good enough

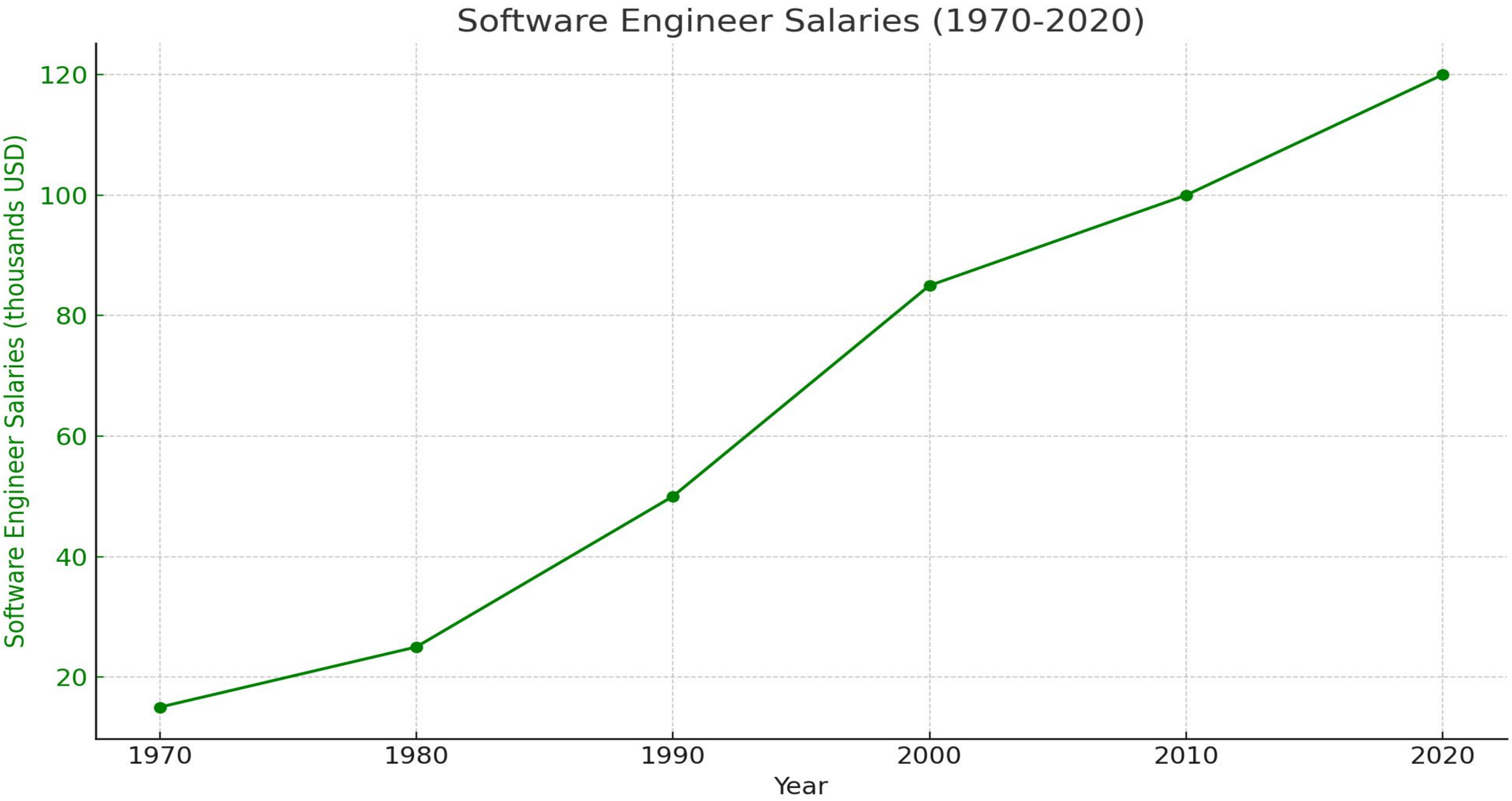
Just because we **can** do something, we don’t have to

Software engineering is not competitive programming

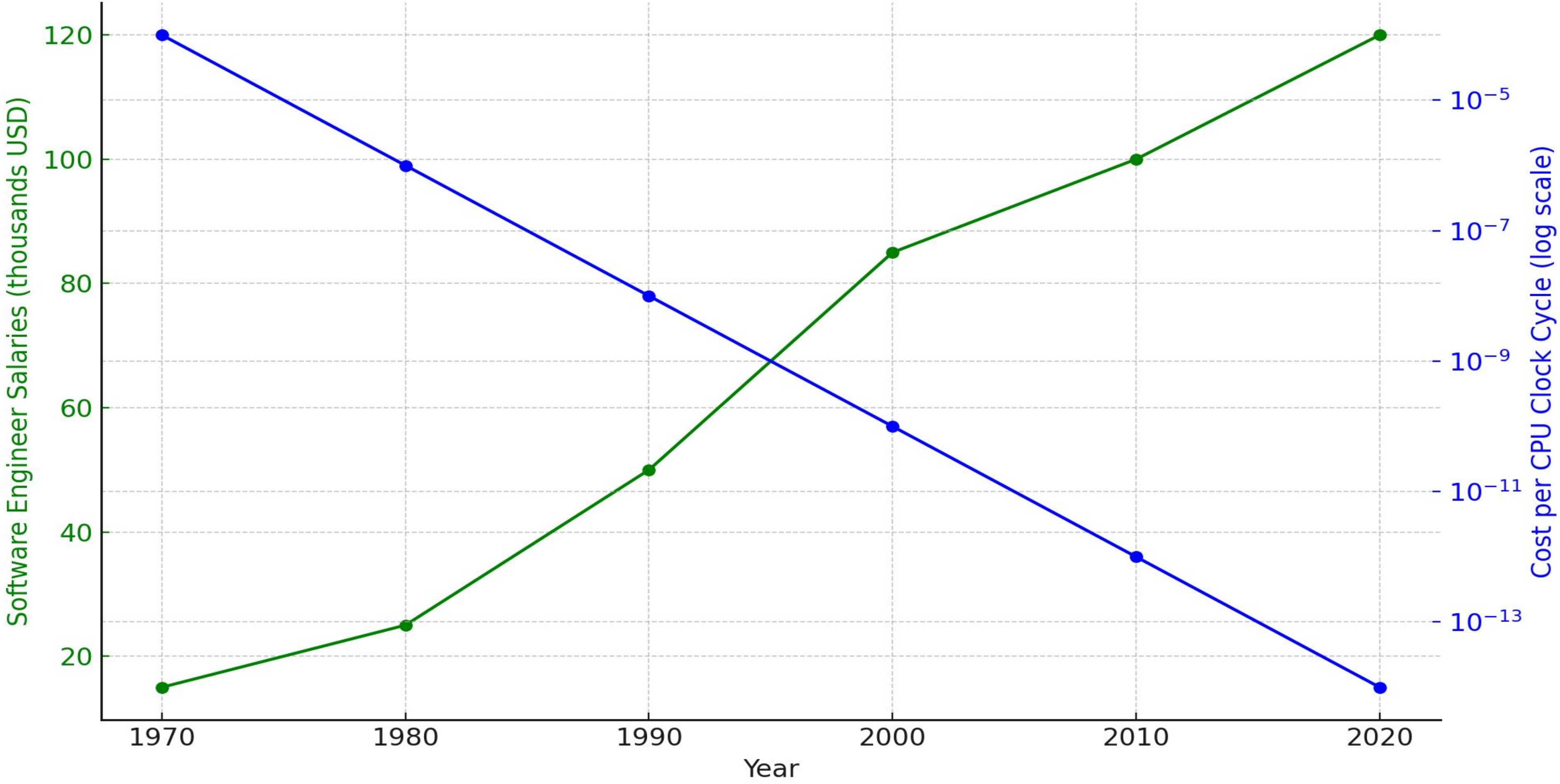
Our goal is the help achieve business goals - for the long term

But why maintainability is so important?


Cost per CPU Clock Cycle (1970-2020)
Cost per CPU Clock Cycle (log scale)
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-13}
1970
1980
1990
2000
2010
2020
Year


Software Engineer Salaries (1970-2020)
Software Engineer Salaries (thousands USD)
120
100
80
60
40
20
1970
1980
1990
2000
2010
2020
Year

Software Engineer Salaries vs CPU Cost per Clock Cycle (1970-2020)



Performance is not l’art pour l’art

We have “cheap” and strong hardware

We also have expensive developers

Unless you have other strict requirements, optimize for maintainability

Does clean code imply horrible performance?

What is clean code?

What is software quality?

Where and why do we use C++?

What is software performance?

Is clean code actually slower?

Should we drop clean code for faster code?

Does clean code imply horrible performance?

Sometimes

Sometimes

If requirements are extreme...

Sometimes

If requirements are extreme…

In most cases, it won’t

Sometimes

If requirements are extreme…

In most cases, it won’t

Not even in the realm of C++

Optimize for maintainability for better ROI

Keep your code **clean, understandable** and **extensible**

Aim for **sufficient** performance

Optimize your hot path **if needed**

Focus on **deliver**ing **business value** for the long run

Clean Code!
Horrible Performance?

SANDOR DARGO

Cppcon
The C++ Conference

2025 | September 13 - 19