The Asian Conference on Education (ACE2025) | Kyoto, Japan | Stream: Primary & Secondary Education

Can Programming Languages Inspire Good Habits? — Abstraction, Memory Management, and More

D. Tony Sün (dsun2019@hotmail.com)

Yeshiva University/William Paterson Univ. (Fall '25-) New Jersey, U.S.A.

virtual presentation (changed to pre-recorded) for Nov. 28, 2025

(Initial Abstract) In most modern programming languages, abstraction goes hand in hand with scoping, the semantic expression of visibility of the code, and C++ has been one of the dominant languages which always decided to implement such abstraction within scopes as opposed to introducing "external" garbage collectors like the ways in Microsoft's C#, Java, and even the generally-purposed Python. This fascinated me while the author was a collegiate learner about over a decade ago, although any technical or philosophical advantages of such design choice were not observed nor reasoned through. This presentation here I try to give a retrospective sketch with some brief analysis on some pedagogical advantages and the meaningfulness of garbage memory management that are based off scoping, the idea of which sprang initially from a dialogue with my former advisor in regards of the concerning of, if not worrying for, students' career choices as well as well-being of each-other as human beings in any society. It is argued that to teach young programmers to write organized code in C++ potentially ignites dialogues on the development of healthy personality and rigorous character, sociopolitical awareness, and good habits, envisioning a fostering outlook for aspiring active and responsible thinkers for beginner-level computer programmers.

(submission #95461)

starter–helloworld/Py.—viewpoint2010—2darrays–checksum—
—cppcon2014—cppcon2017—clevercode—
—conclusion

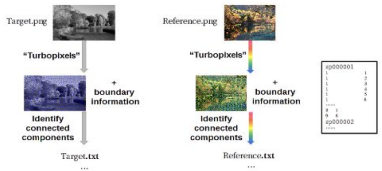
Starter: Dilemma of Efficiency

When running a (continuing) project in software development, especially after an established team has been orchestrated for a while.... one day your Team Leader decided to add more manpower there, then...

sometimes, it **gets worse**

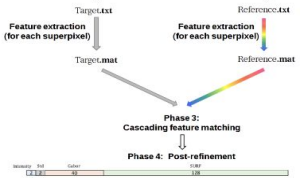
Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.

- knowing some good/right measure, of and when working as a team
- for software programming and development: "how often" is team-work?
  collaborative and human/emotion intelligence, "no hard feelings," of saying no instead of yes & cutting down **time of collaboration**
  (namely compared to individual work)
- ~100 email messages (there asynchronous) two-men team project with only a few times of in-person meeting discussions for a self-directed computational project in a graduate course in computer science



Phase I: Identifying Superpixels


'Turbopixel' at ~40 Pixels Per Cell



Phase II: Mixed Gabor-SURF 172-D Feature


Leveraging Multiple Examplers

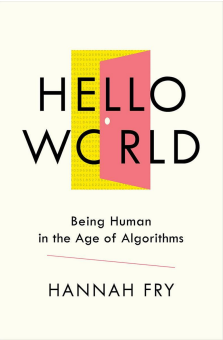
The "Hello, World" Phrase, to the Prevalent Python Nowadays..

Since half a century ago programming learners would try to print or instruct a "Hello world" output as "the first moment when chit-chat with your computer was a possibility" when this phrase was first suggested in Brian Kernighan and Dennis Ritchie's 1978 book, for which Kernighan shared later that this idea came from a cartoon he saw of "a newly hatched chick's chirping the words 'Hello world!' as it was born" (Fry, 2018).

Contemporary computer science education/classrooms more likely adopt Python and/or Java (AP Computer Science, etc.) as well as for learning some part of algorithms/data structures. My first/inaccurate impression on Python is:

- "experiment" code
- general-purpose computing
- not the best for writing classes

However, in fact Python is not bad for classes (in the O.O.P. sense), and actually is much "**overused**," e.g. store all the data for a task into (instead) a dictionary will have faster performance, compared to using a (separate) class.



Overall: the emphasis for Python and its primary advantage is about the reducing/saving "time spent writing code," and definitely about the **concise**ness which is linked to **readability**.. (what does this entail, both aspects; C++?)

Fry, H. (2018). *Hello World: Being Human in the Age of Algorithms*. W. W. Norton & Company.

A Little About Software Company and Working Environment as Programmers..


DOI:10.1145/1629175.1629192 Bjarne Stroustrup

Viewpoint

What Should We Teach New Software Developers? Why?

Fundamental changes to computer science education are required to better address the needs of industry.

Implementation "as black boxes"?

project cycles, addition of "resources" (management & developing), ..

- **disconnect** (job in industry versus academics): at the root of many problems and it complicates attempts to remedy them
- **maintenance** and **code quality**, as opposed to a "strange combination of unprincipled hacking and invoking other people's libraries) in programming in general

What does collaborating/coding in C++ entail?

Stroustrup, B. (2010). "What Should We Teach New Software Developers? Why?" *Communications of the ACM* (Viewpoints), Vol. 53(1).

Example: 2-D Arrays

```
const int WIDTH  = 10;
const int HEIGHT =  5;
//float A[5][3] = { ... };

// the 'common' way of making a matrix '2-dim' array
float A[HEIGHT][WIDTH] = {};  // started empty
A[0][9] = 1.75;
std::cout << "the entry just specified at indexes 0, 9 is: " <<
A[0][9] << "\n";

// float *A1 = new float[HEIGHT * WIDTH]; // can be an 1-D array also

// the 'dynamic' way to provide each 'row'
float **A1 = new float*[HEIGHT];  // each row is a float-pointer
for (int r=0; r<HEIGHT; r++)
   A1[r] = new float[WIDTH];
   // i.e. provide each row 'WIDTH' many elements/entries
   // so that each row is a regular array of size WIDTH
```

Example: Think "Antecedents," for Checksum/Error Prevention

```
def is_poweroftwo(n): // in Python
    return 0 == (n & (n-1))
```

E.g. when $n = 8 = 2^{**}3$
then n is 1000, and
$(n-1)=7$ is 111 (or 0111)
then performing such "bitwise and" (&) we'll get 0, 0, 0, and 0 for each digit positions respectively, which is (in binary)
0000 = 0 (also zero in decimal)
so here the "0 == bitwise and calculation" is satisfied as true hence the function returning true.

This is because/based on the fact that any power of two is eventually equivalent to
1000....
i.e. with a leading 1 followed by some digits of zeros (in binary) since 2**k = 1 * (2**k) which is aligned with the meaning/definition of the binary presentation. Although this function merely writes the functionality.

Dealing better with types, then

```
def is_poweroftwo(n: int) -> bool:
    # only consider natural numbers 2, 3, ..
    return n > 1 and 0 == (n & (n-1))
```

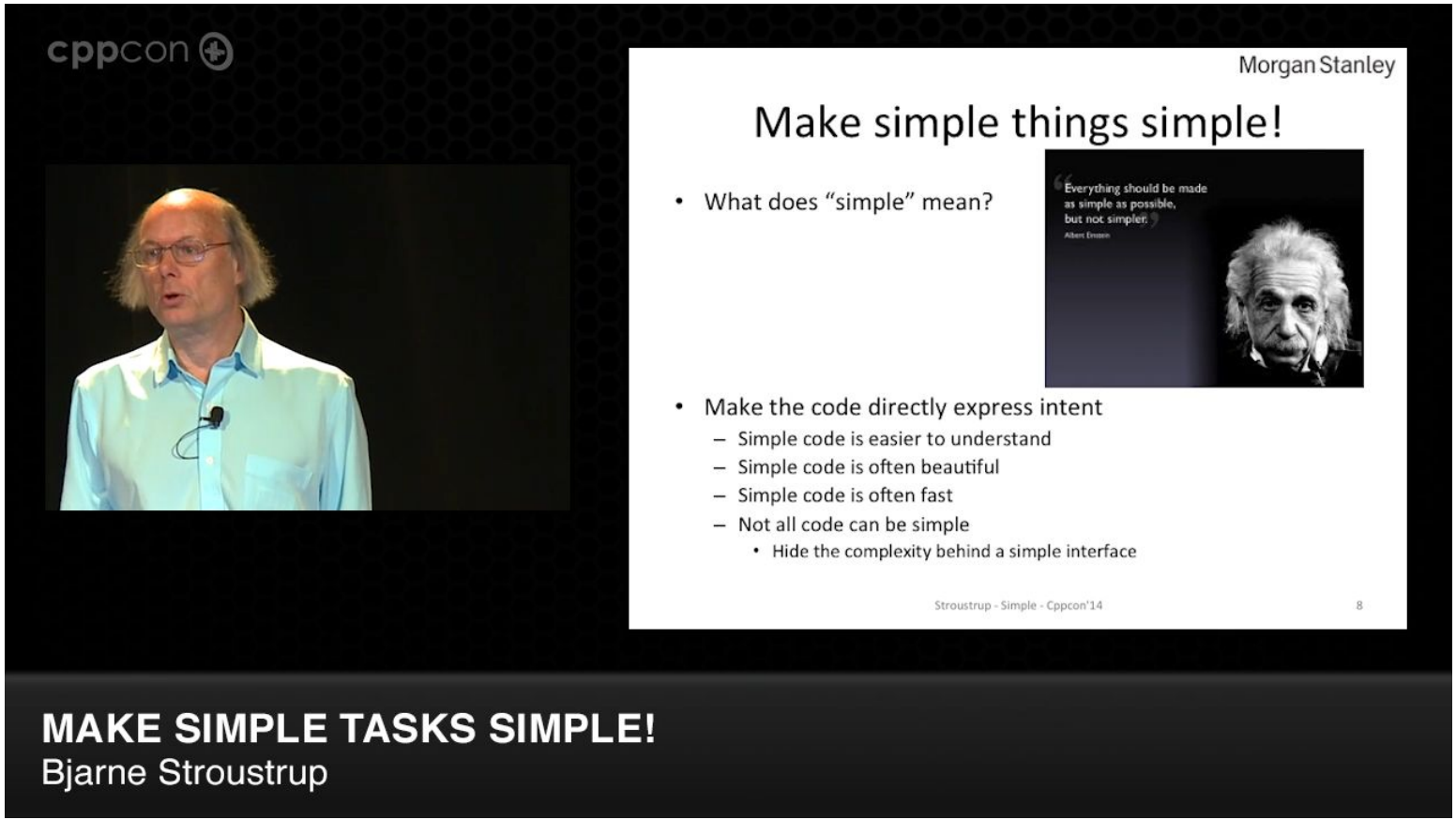
C++ version

```
#include <iostream>

bool is_poweroftwo(int n) {
    return n > 1 && 0 == ((n-1) & n);
}

int main()
{
    std::cout << is_poweroftwo(16) << "\n";
    std::cout << is_poweroftwo(17) << "\n";
}
```

Generally, Habits in Studying, for Work Preparation, etc.

"Improve Your Study Habits" (McDaniel, 1961)

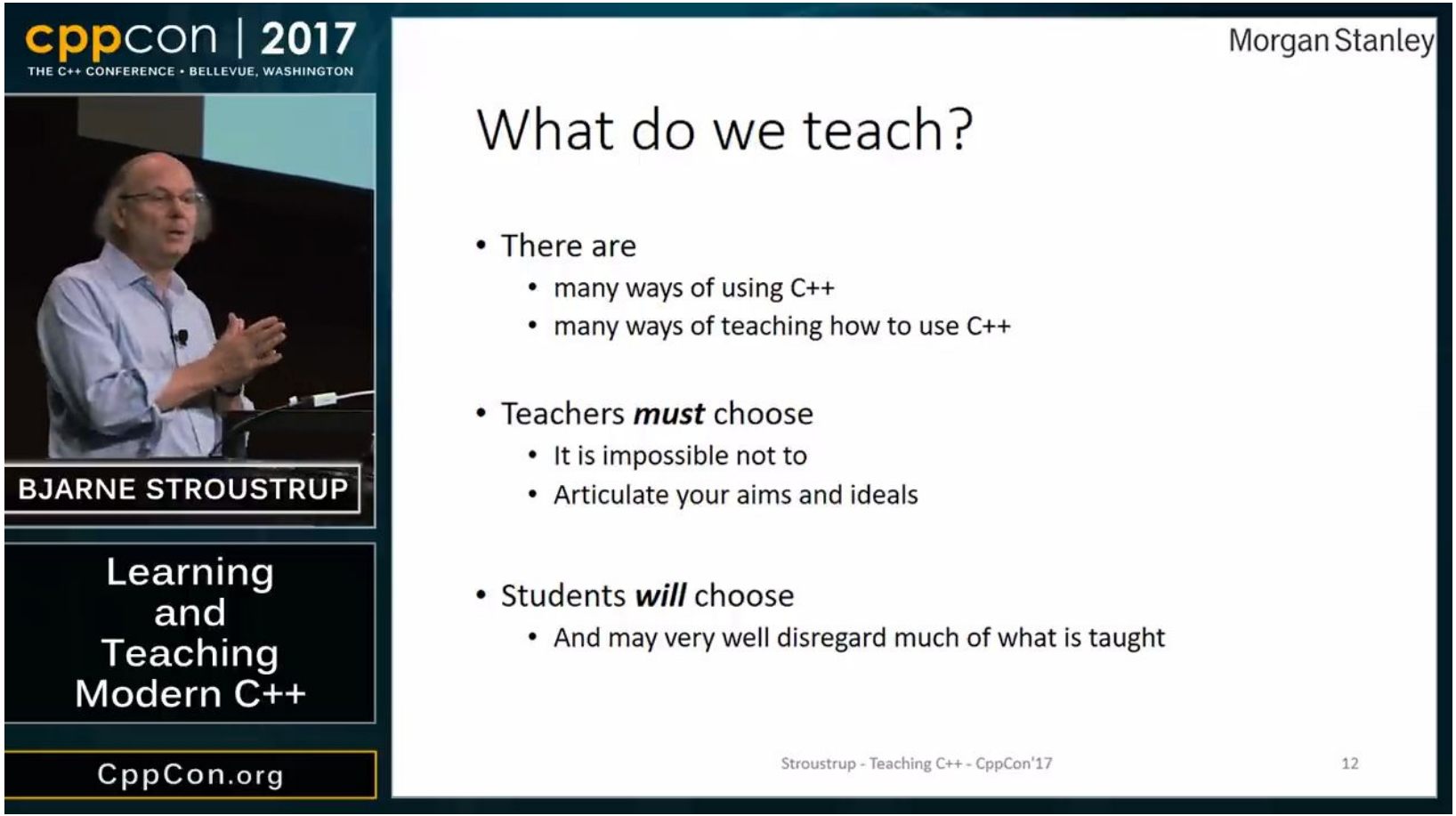
“Make Simple Tasks Simple!” (Stroustrup, at *CppCon 2014*)



a low-level (programming) language

word-level parallelism

“Learning and Teaching Modern C++” (Stroustrup, at *CppCon 2017*)



1:27 min.

an “existential lemma/irony” ?

"Clever Code"

(Most recently, another note from presentation by Stroustrup:)

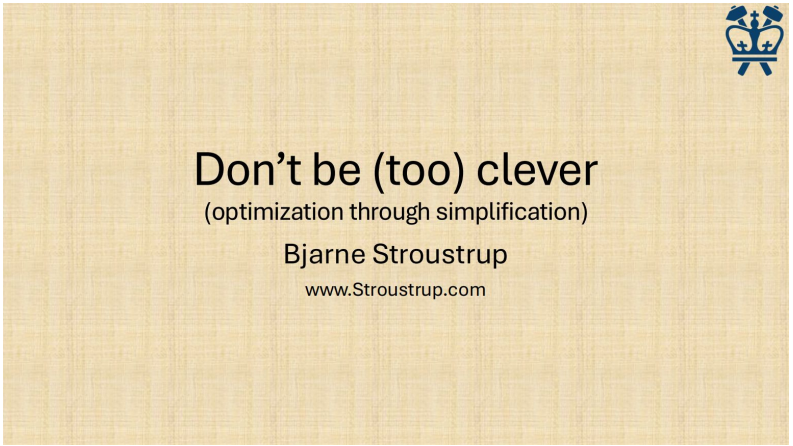
Even to engineers who are responsible for building the fastest trading systems on Wall Street (finance)–a few pieces of advice:

- measure everything that matters (i.e. be considerate and judicious comprehensively); "to start optimizing code, throw out the clever code," and "if in doubt, simplify"
- *optimization* is valuable for <u>about 3% of the code</u> (Knuth); not to optimize prematurely (*what's an optimizer?*)
- costly "distributed fat"
- throw out the clever code–perhaps, if we/the programmer can't understand what's going on, neither can the optimizer..

—according to LinkedIn post by Corcoran, J. (July 2025)

Don't be too clever: Optimization through simplification STAC

Bjarne Stroustrup, Creator of C++, presented this at the 13 May 2025 STAC Summit in New York.



In other words, implementation with C++ (or any other language) is a good structure of the **engineering process**, for the objectives to achieve, which shall <u>not be "equated"</u> <u>with just being fancy</u>, etc.

—this just as reminder, and in particular for code written "in responsibility" programs

Conclusion: Whenever We Code

Process and time spent on/during coding, writing things in any programming languages:

> **such time *counts***, both as serious industrial-engineering **work** and as **for joy** (algorithm invention, experimentation, scientific research, or programming as art), and shall be corresponded with our attention as to how it **affects our habits/behavioral pattern** personally.

Similar to *mathematics* as once the major cultural force in Western civilization (from the Renaissance, classical mechanics, the Age of Reason, non-Euclidean geometry..), *programming languages*—as a form or writing at first—shall aim to drive civilization in a similar spirit/manner as well;

a good language like C++ contributes positively in this aspect and this process.

Type more efficiently, type "with purpose," and type for "readability" especially where it needs the most.

Mathematics "carries the main burden of scientific reasoning." —Mathematics in Western Culture (Kline, 1953)

Differences (any) between software engineering and science? (or engineering in general)

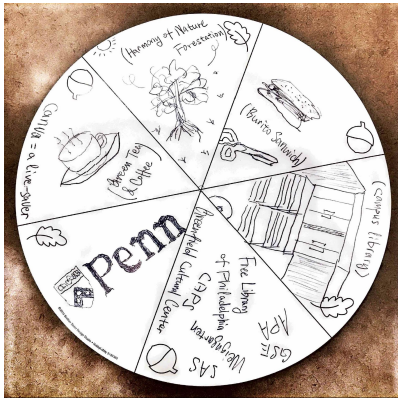
Most "scientists" might code less often in C++ or lower-level languages nowadays.

An average data-science classroom; what about computer science education

(e.g. Introduction to Computer Science '101')

About Me (Currently of M.A. in Mathematics Program at Yeshiva)

(me at my rental office in north of Princeton, New Jersey as of late 2024)



studying mathematical analysis (e.g., metric spaces), multivariate Calculus (vector and tensor algebra); besides, a few other research projects about political philosophy and jurisprudence

Fall 2023 to Now: Mathematics

Reading/Writing "Literacy Education" at the University of Pennsylvania (2021-22)

Computing Science: Computer Graphics, and Image Analysis and Computational Photography/Geometry Algorithms (2014-18)

Engineering Student in College (pre-2014)