Code Style

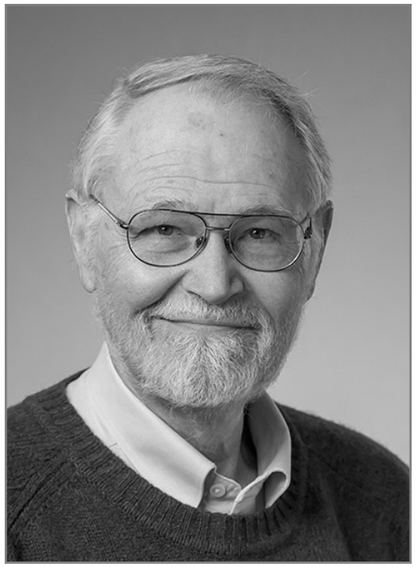
Yegor Bugayenko

Lecture #22 out of 24
80 minutes

Which One Is Better?

```
int f(int n)
{
  if (n == 1 || n < 2)
    return 1;
  int r = f (n-1);
  int r2 = f(n - 2);
  return r +r2;
  }
```

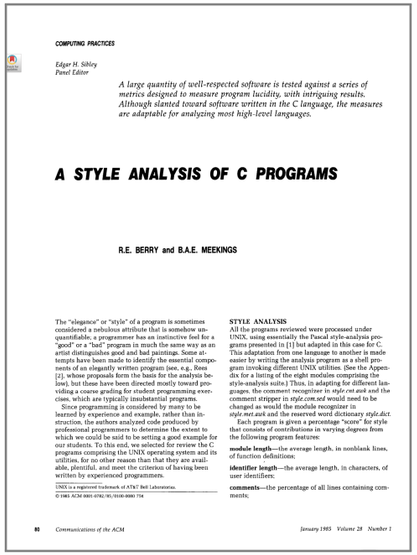
```
int fibonacci(int n) {
  if (n <= 2) {
    return 1;
  }
  return fibonacci(n - 1)
    + fibonacci(n - 2);
}
```



Brian Kernighan

“The harder it is for people to grasp the intent of any given section, the longer it will be before the program becomes operational. Trying to outsmart a compiler defeats much of the purpose of using one. Write clearly — don’t sacrifice clarity for ‘efficiency.”’

— Brian W. Kernighan and Phillip James Plauger. *The Elements of Programming Style.* McGraw-Hill, Inc., 1974



COMPUTING PRACTICES

Edgar H. Sibley
Panel Editor

A large quantity of well-respected software is tested against a series of metrics designed to measure program lucidity, with intriguing results. Although slanted toward software written in the C language, the measures are adaptable for analyzing most high-level languages.

A STYLE ANALYSIS OF C PROGRAMS

R.E. BERRY and B.A.E. MEEKINGS

The "elegance" or "style" of a program is sometimes considered a nebulous attribute that is somehow unquantifiable; a programmer has an instinctive feel for a "good" or a "bad" program in much the same way as an artist distinguishes good and bad paintings. Some attempts have been made to identify the essential components of an elegantly written program (see, e.g., Rees [2], whose proposals form the basis for the analysis below), but these have been directed mostly toward providing a coarse grading for student programming exercises, which are typically insubstantial programs.

Since programming is considered by many to be learned by experience and example, rather than instruction, the authors analyzed code produced by professional programmers to determine the extent to which we could be said to be setting a good example for our students. To this end, we selected for review the C programs comprising the UNIX operating system and its utilities, for no other reason than that they are available, plentiful, and meet the criterion of having been written by experienced programmers.

UNIX is a registered trademark of AT&T Bell Laboratories.
© 1985 ACM 0001-0782/85/0100-0080 75¢

STYLE ANALYSIS

All the programs reviewed were processed under UNIX, using essentially the Pascal style-analysis programs presented in [1] but adapted in this case for C. This adaptation from one language to another is made easier by writing the analysis program as a shell program invoking different UNIX utilities. (See the Appendix for a listing of the eight modules comprising the style-analysis suite.) Thus, in adapting for different languages, the comment recognizer in *style.ret.awk* and the comment stripper in *style.com.sed* would need to be changed as would the module recognizer in *style.met.awk* and the reserved word dictionary *style.dict*.

Each program is given a percentage "score" for style that consists of contributions in varying degrees from the following program features:

module length—the average length, in nonblank lines, of function definitions;

identifier length—the average length, in characters, of user identifiers;

comments—the percentage of all lines containing comments;

“The ‘elegance’ or ‘style’ of a program is sometimes considered a nebulous attribute that is somehow unquantifiable; a programmer has an instinctive feel for a ‘good’ or a ‘bad’ program in much the same way as an artist distinguishes good and bad paintings.”

— R E Berry and B AE Meekings. A Style Analysis of C Programs. *Communications of the ACM*, 28(1):80–88, 1985

Berry-Meekings Score

TABLE I. Metric Boundary Values

Metric	%	L	S	F	H
Module length	15	4	10	25	35
Identifier length	14	4	5	10	14
Comment lines (%)	12	8	15	25	35
Indentation (%)	12	8	24	48	60
Blank lines (%)	11	8	15	30	35
Characters per line	9	8	12	25	30
Spaces per line	8	1	4	10	12
Defines (%)	8	10	15	25	30
Reserved words	6	4	16	30	36
Include files	5	0	3	3	4
Gotos	−20	1	3	99	99

An individual score for each metric is determined by reference to the value in this table for

1. the point L, below which no score is obtained;
2. the point S, the start of the "ideal" range for the metric;
3. the point F, the end of the ideal range;
4. the point H, above which no

score is obtained.

Source: R E Berry and B AE Meekings. A Style Analysis of C Programs. *Communications of the ACM*, 28(1):80–88, 1985

Henry Ledgard

"An individual's body language helps clarify the spoken word. In a similar sense, the programmer relies on white space—what is not said directly—in the code to communicate logic, intent, and understanding."

— Robert Green and Henry Ledgard. Coding Guidelines: Finding the Art in the Science. *Communications of the ACM*, 54(12):57–63, 2011

Figure 9. Examples of K&R, ANSI, and Whitesmiths coding styles.

```
if (expression) {
    statements
}
```

```
if (expression)
{
    statements
}
```

```
if (expression)
    {
    statements
    }
```

Source: Robert Green and Henry Ledgard. Coding Guidelines: Finding the Art in the Science. *Communications of the ACM*, 54(12):57–63, 2011

Peter C. Rigby

"We list the reasons why our interviewees rejected a patch or required further modification before accepting it: Poor quality, Violation of style, Gratuitous changes mixed with 'true' changes, Code does not do or fix what it claims to or introduces new bugs, Fix conflicts with existing code, Use of incorrect API or library."

— Peter C. Rigby and Margaret-Anne Storey. Understanding Broadcast Based Peer Review on Open Source Software Projects. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 541–550, 2011

Jennifer Bauer

"While the perceptual processing of code is required to understand it, higher level processing, such as understanding its semantics and reasoning about its functionality, affect program comprehensibility more strongly. The influence of indentation could have been masked by these side effects, so it might well be that the effect of indentation comes more into play when the code is longer and more complex."

— Jennifer Bauer, Janet Siegmund, Norman Peitek, Johannes C. Hofmeister, and Sven Apel. Indentation: Simply a Matter of Style or Support for Program Comprehension? In *Proceedings of the 27th International Conference on Program Comprehension (ICPC)*, pages 154–164. IEEE, 2019

Weiqin Zou

"A pull request that is consistent with the current code style tends to be merged into the codebase more easily (faster)."

— Jennifer Bauer, Janet Siegmund, Norman Peitek, Johannes C. Hofmeister, and Sven Apel. Indentation: Simply a Matter of Style or Support for Program Comprehension? In *Proceedings of the 27th International Conference on Program Comprehension (ICPC)*, pages 154–164. IEEE, 2019

My Favorite Style Checkers

- ESLint for JavaScript
- Clang-Tidy for C++
- Pylint for Python
- Rubocop for Ruby
- PHP_CodeSniffer for PHP
- rustfmt for Rust
- Qulice by [Bugayenko, 2014] for Java (Checkstyle + PMD)

How Many Rules in Style Checkers?

- 690+ in Clang-Tidy (C++)
- 550+ in Rubocop (Ruby)
- 400+ in PMD (Java)
- 130+ in Checkstyle (Java)
- 120+ in Pylint (Python)

Some/most of the rules no only check style, but also find bugs

Some Exotic Style Checkers

- Shellcheck for Bash
- markdownlint for Markdown
- Checkmake for Makefile
- xcop for XML

References

Jennifer Bauer, Janet Siegmund, Norman Peitek, Johannes C. Hofmeister, and Sven Apel. Indentation: Simply a Matter of Style or Support for Program Comprehension? In *Proceedings of the 27th International Conference on Program Comprehension (ICPC)*, pages 154–164. IEEE, 2019.

R E Berry and B AE Meekings. A Style Analysis of C Programs. *Communications of the ACM*, 28(1): 80–88, 1985.

Yegor Bugayenko. Strict Control of Java Code Quality. `https://www.yegor256.com/140813.html`, August 2014. [Online; accessed 26-02-2024].

Robert Green and Henry Ledgard. Coding Guidelines: Finding the Art in the Science. *Communications of the ACM*, 54(12):57–63, 2011.

Brian W. Kernighan and Phillip James Plauger. *The Elements of Programming Style*. McGraw-Hill, Inc., 1974.

Peter C. Rigby and Margaret-Anne Storey. Understanding Broadcast Based Peer Review on Open Source Software Projects. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 541–550, 2011.