

# Comments Density

YEGOR BUGAYENKO

Lecture #19 out of 24  
80 minutes

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.



BRIAN KERNIGHAN

“The best documentation for a computer program is a clean structure. It also helps if the code is well formatted, with good mnemonic identifiers, labels, and a smattering of enlightening comments. Flowcharts and program descriptions are of secondary importance; the only reliable documentation of a computer program is the code itself.”

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

domains. Consider this piece of code and comment in a FORTRAN program:

```
C THIS SHIFTS B LEFT 4 BITS  
  A = B*2**4
```

While the comment comes close to violating Kernighan and Plauger's prohibition against echoing the code, it does provide useful information to the programmer; it alerts him to the fact that an operation in the programming language domain - multiplication - is to be mapped into an operation - shifting - in the domain of program execution. In writing comments, then, the programmer should be aware of possible knowledge domains and should strive to make each comment provide information about the mapping from the programming language domain into another domain.

“Comments which precede a group of statements, and which describe them in terms of operations in another domain, will be particularly helpful. The role of comments is to bridge between knowledge domains.”

Source: Ruven Brooks. Using a Behavioral Theory of Program Comprehension in Software Engineering. In *Proceedings of the 3rd International Conference on Software Engineering*, pages 196–201, 1978



HUBERT E. DUNSMORE

“An experiment was conducted to investigate how comments are related to programmers’ ability to understand programs. Those programmers whose programs contained comments were able to answer more questions than those without comments.”

— Scott N. Woodfield, Hubert E. Dunsmore, and Vincent Y. Shen. The Effect of Modularization and Comments on Program Comprehension. In *Proceedings of the 5th International Conference on Software Engineering*, pages 215–223, 1981



“[The degree of] intramodule commenting is the number of lines with comments divided by the total number of lines in the module, averaged over all modules.”

— Paul Oman and Jack Hagemeister. Metrics for Assessing a Software System’s Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE Computer Society, 1992



DAVID PARNAS

“Documentation that seems clear and adequate to its authors is often about as clear as mud to the programmer who must maintain the code six months or six years later.”

— David Lorge Parnas. Software Aging. In *Proceedings of the 16th International Conference on Software Engineering*, pages 279–287. IEEE, 1994

## Comments Affect Maintainability

After every change, during maintenance stage, we have the next information for each modified module:

### Maintainability Metrics

**LC = Lines of comments**

**LEM = Lines of easy modifiability**

**LED = Lines of Errors Detection**

**IM = Investment on maintainability**

**( LC + LEM + LED )**

Source: Manuel J. Barranco Garcia and Juan Carlos Granja Alvarez. Maintainability as a Key Factor in Maintenance Productivity: A Case Study. In *Proceedings of the International Conference on Software Maintenance*, pages 87–93. IEEE, 1996



MARTIN FOWLER

“Don’t worry, we aren’t saying that people shouldn’t write comments. In our olfactory analogy, comments aren’t a bad smell; indeed they are a sweet smell. The reason we mention comments here is that comments often are used as a deodorant. It’s surprising how often you look at thickly commented code and notice that the comments are there because the code is bad.”

— Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts.  
*Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999





ANDY HUNT

“Programmers are taught: good code has lots of comments. Unfortunately, they are never taught why code needs comments: bad code requires lots of comments. The DRY principle tells us to keep the low-level knowledge in the code, where it belongs, and reserve the comments for other, high-level explanations. Otherwise, we’re duplicating knowledge, and every change means changing both the code and the comments. The comments will inevitably become out of date, and untrustworthy comments are worse than no comments at all.”

— Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999



ERIKO NURVITADHI

“The results indicated that method comments do increase low-level program understanding, while class comments did not increase high-level understanding. This raises questions about the role of class comments in Object-Oriented programs...”

— Eriko Nurvitadhi, Wing Wah Leung, and Curtis Cook. Do Class Comments Aid Java Program Understanding? In *Proceedings of the 33rd Annual Frontiers in Education*, volume 1, pages T3C–T3C. IEEE, 2003



STEVE McCONNELL

“The main contributor to code-level documentation isn’t comments, but good programming style... Comments are easier to write poorly than well, and commenting can be more damaging than helpful.”

— Steve McConnell. *Code Complete*. Pearson Education, 2004



BEAT FLURI

“Code and comments rarely co-evolve: despite its growth rate, newly added code barely is commented. Also, 97% of comment changes are done in the same revision as the associated source code change.”

— Beat Fluri, Michael Wursch, and Harald C. Gall. Do Code and Comments Co-evolve? On the Relation Between Source Code and Comment Changes. In *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 70–79. IEEE, 2007



ROBERT C. MARTIN

“Indeed, comments are, at best, a necessary evil. If our programming languages were expressive enough, or if we had the talent to subtly wield those languages to express our intent, we would not need comments very much—perhaps not at all.”

— Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008



OLIVER ARAFAT

“Comment density is the percentage of comment lines in a given source code base, that is, comment lines divided by total lines of code. Comment density is assumed to be a good predictor of maintainability and hence survival of a software project. In this study we focus on one particular code metric, the comment density, and assess it across 5,229 active open source projects, representing about 30% of all active open source projects.”

— Oliver Arafat and Dirk Riehle. The Comment Density of Open Source Software Code. In *Proceedings of the 31st International Conference on Software Engineering, Companion Volume*, pages 195–198. IEEE, 2009

## Project Size vs. Comments Density



Source: Oliver Arafat and Dirk Riehle. The Comment Density of Open Source Software Code. In *Proceedings of the 31st International Conference on Software Engineering, Companion Volume*, pages 195–198. IEEE, 2009



HOUARI SAHRAOUI

“We defined a taxonomy of comments to guide this analysis. Our study showed that programmers comment some constructs more often than others. In the majority of cases, comments are intended to explain the code that follows them. The second more widely used category of comments are dedicated to communication between programmers and personal notes (we call them working comments).”

— Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011



## Types of Comments



Source: Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011

## Frequency of Comments



Source: Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011



TOBIAS RÖHM

“Source code is more trusted than documentation: 21 participants reported that they get their main information from source code and inline comments whereas only four stated that documentation is their main source of information.”

— Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional Developers Comprehend Software? In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 255–265. IEEE, 2012



LUCA PASCARELLA

“Code comments contain valuable information to support software development, especially during code reading and code maintenance. Nevertheless, not all the comments are the same.”

— Luca Pascarella, Magiel Bruntink, and Alberto Bacchelli. Classifying Code Comments in Java Software Systems. *Empirical Software Engineering*, 24(3): 1499–1537, 2019

## Taxonomy of Comment Types



Source: Luca Pascarella, Magiel Bruntink, and Alberto Bacchelli. Classifying Code Comments in Java Software Systems. *Empirical Software Engineering*, 24(3):1499–1537, 2019



HAO HE

“We analyzed the comment density of 150 projects in 5 different programming languages. We have found that there are noticeable differences in comment density, which may be related to the programming language used in the project and the purpose of the project.”

— Hao He. Understanding Source Code Comments at Large-scale. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019

## Comments Density by Language



Source: Hao He. Understanding Source Code Comments at Large-scale. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019



SEAN STAPLETON

“Participants reviewed Java methods and summaries and answered established program comprehension questions. In addition, participants completed coding tasks given summaries as specifications. We found that participants performed significantly better using human-written summaries versus machine-generated summaries.”

— Sean Stapleton, Yashmeet Gambhir, Alexander LeClair, Zachary Eberhart, Westley Weimer, Kevin Leach, and Yu Huang. A Human Study of Comprehension and Code Summarization. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 2–13, 2020





XING HU

“Code comment generation is a popular area of research in recent years. In this work, we interviewed 16 professionals and surveyed 720 practitioners on commenting practices and issues they face and their expectations on code comment generation tools. Practitioners are enthusiastic about research in comment generation techniques and expect tools to generate comments for different granularity levels (especially class and method levels).”

— Xing Hu, Xin Xia, David Lo, Zhiyuan Wan, Qiuyuan Chen, and Thomas Zimmermann. Practitioners’ Expectations on Automated Code Comment Generation. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1693–1705, 2022

## Nine Rules of Good Code Comments

1. Comments should not duplicate the code.
2. Good comments do not excuse unclear code.
3. If you can't write a clear comment, there may be a problem with the code.
4. Comments should dispel confusion, not cause it.
5. Explain unidiomatic code in comments.
6. Provide links to the original source of copied code.
7. Include links to external references where they will be most helpful.
8. Add comments when fixing bugs.
9. Use comments to mark incomplete implementations.



Source: Ellen Spertus. Best practices for writing code comments.

<https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/>, 2021

Some Open Source Repositories (9 Feb 2024)

Github Repository	Stack	Files	Comments	LoC	Com/LoC
<u>nodejs</u>	JS & C++	32559	1003K	8381K	0.12
<u>pytorch</u>	Python	11562	414K	2527K	0.16
<u>moby</u> (a.k.a. Docker)	Go	8389	272K	1685K	0.16
<u>flutter</u>	Dart	5517	244K	1353K	0.18
<u>spring-framework</u>	Java	9883	400K	880K	0.45
<u>guava</u>	Java	1984	131K	479K	0.27
<u>curl</u>	C	2014	63K	314K	0.20

## My Own Statistics (9 Feb 2024)

Github Repository	Stack	Comments	LoC	Com/LoC
<a href="#">zerocracy/farm</a>	Java	34380	58330	0.59
<a href="#">objectionary/eo</a>	Java	23383	49151	0.48
<a href="#">yegor256/cactoos</a>	Java	25857	33826	0.76
<a href="#">yegor256/takes</a>	Java	21393	26769	0.80
<a href="#">zold-io/zold</a>	Ruby	4306	11807	0.36
<a href="#">yegor256/tacit</a>	CSS	259	1110	0.23

All repositories are open source.

# References

Oliver Arafat and Dirk Riehle. The Comment Density of Open Source Software Code. In *Proceedings of the 31st International Conference on Software Engineering, Companion Volume*, pages 195–198. IEEE, 2009.

Manuel J. Barranco Garcia and Juan Carlos Granja Alvarez. Maintainability as a Key Factor in Maintenance Productivity: A Case Study. In *Proceedings of the International Conference on Software Maintenance*, pages 87–93. IEEE, 1996.

Ruven Brooks. Using a Behavioral Theory of Program Comprehension in Software Engineering. In *Proceedings of the 3rd International Conference on Software Engineering*, pages 196–201, 1978.

Beat Fluri, Michael Wursch, and Harald C. Gall. Do Code and Comments Co-evolve? On the Relation Between Source Code and Comment Changes. In *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 70–79.

IEEE, 2007.

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011.

Hao He. Understanding Source Code Comments at Large-scale. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019.

Xing Hu, Xin Xia, David Lo, Zhiyuan Wan, Qiuyuan Chen, and Thomas Zimmermann. Practitioners' Expectations on Automated Code Comment Generation. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1693–1705, 2022.

Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999.

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

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008.

Steve McConnell. *Code Complete*. Pearson Education, 2004.

Eriko Nurvitadhi, Wing Wah Leung, and Curtis Cook. Do Class Comments Aid Java Program Understanding? In *Proceedings of the 33rd Annual Frontiers in Education*, volume 1, pages T3C–T3C. IEEE, 2003.

Paul Oman and Jack Hagemester. Metrics for Assessing a Software System’s Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE Computer Society, 1992.

David Lorge Parnas. Software Aging. In *Proceedings*

*of the 16th International Conference on Software Engineering*, pages 279–287. IEEE, 1994.

Luca Pascarella, Magiel Bruntink, and Alberto Bacchelli. Classifying Code Comments in Java Software Systems. *Empirical Software Engineering*, 24(3):1499–1537, 2019.

Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional Developers Comprehend Software? In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 255–265. IEEE, 2012.

Ellen Spertus. Best practices for writing code comments.  
<https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/>, 2021.

Sean Stapleton, Yashmeet Gambhir, Alexander LeClair, Zachary Eberhart, Westley Weimer, Kevin Leach, and Yu Huang. A Human Study of Comprehension and Code Summarization. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 2–13, 2020.

Scott N. Woodfield, Hubert E. Dunsmore, and  
Vincent Y. Shen. The Effect of Modularization and  
Comments on Program Comprehension. In

*Proceedings of the 5th International Conference on  
Software Engineering*, pages 215–223, 1981.