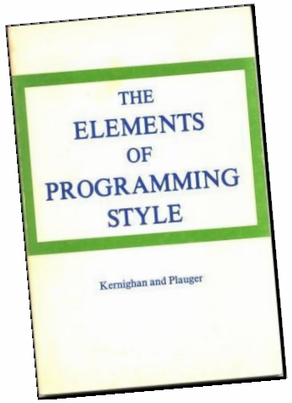
Comments Density

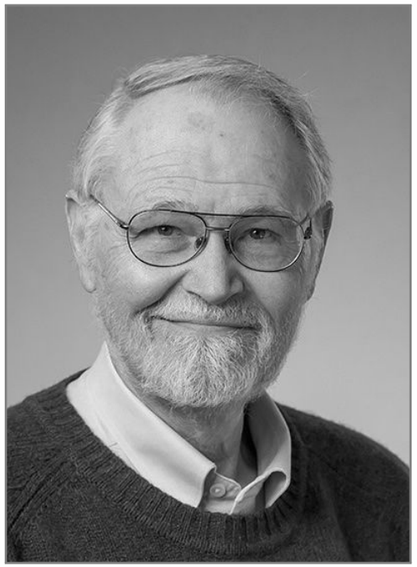
Yegor Bugayenko

Lecture #19 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

1. A properly written program doesn't need comments, or maybe it does?





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. doi:10.5555/601121

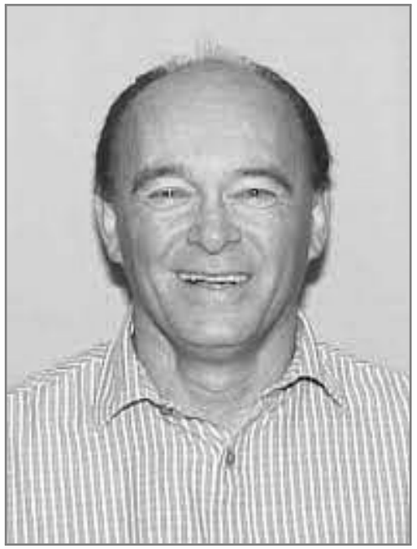
domains. Consider this piece of code and comment in a FORTRAN program:

```
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."

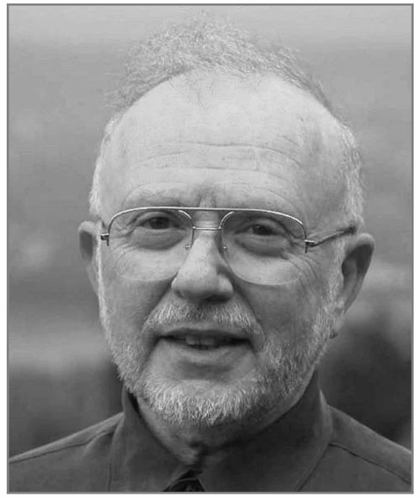
Source: Ruven Brooks. Using a Behavioral Theory of Program Comprehension in Software Engineering. In *Proceedings of the 3rd International Conference on Software Engineering*, pages 196–201, 1978. doi:10.5555/800099.803210



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. doi:10.5555/800078.802534



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."

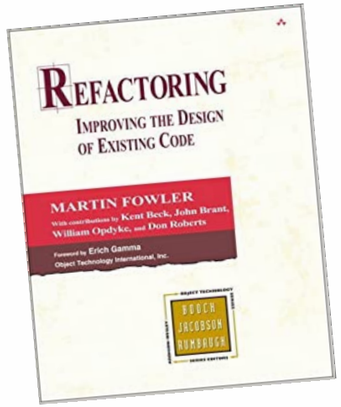
— David Lorge Parnas. Software Aging. In *Proceedings of the 16th International Conference on Software Engineering*, pages 279–287. IEEE, 1994. doi:10.1109/icse.1994.296790

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)

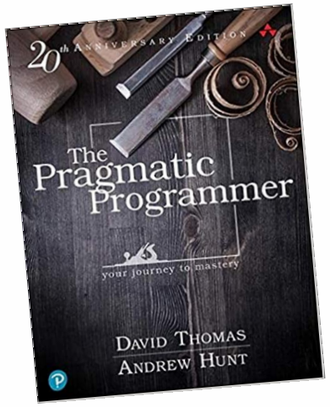
Source: Manuel J. Barranco Garcia and Juan Carlos Granja Alvarez. Maintainability as a Key Factor in Maintenance Productivity: A Case Study. In *Proceedings of the International Conference on Software Maintenance*, pages 87–93. IEEE, 1996. doi:10.1109/icsm.1996.564992



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, 1999. doi:10.5555/311424



Andy Hunt

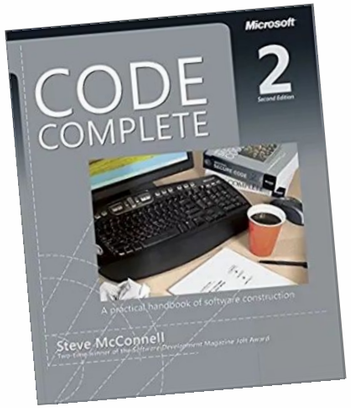
“Programmers are taught: good code has lots of comments. Unfortunately, they are never taught why code needs comments: bad code requires lots of comments. The DRY principle tells us to keep the low-level knowledge in the code, where it belongs, and reserve the comments for other, high-level explanations. 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. doi:10.5555/320326

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 130–131. IEEE, 2003. doi:10.1109/fie.2003.1263332



Steve McConnell

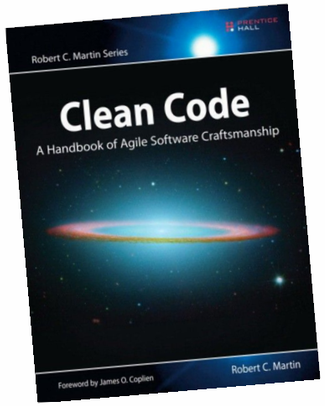
“The main contributor to code-level documentation isn’t comments, but good programming style… Comments are easier to write poorly than well, and commenting can be more damaging than helpful.”

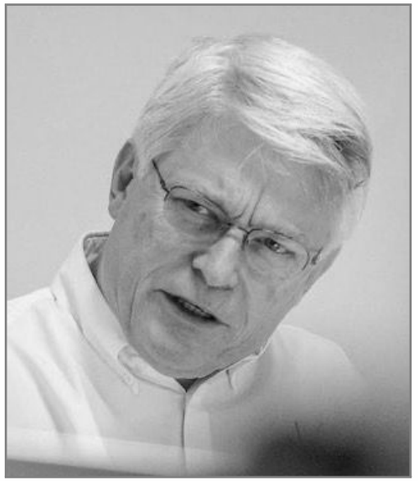
— Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143

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*, pages 70–79. IEEE, 2007. doi:10.1109/wcre.2007.21





Robert C. Martin

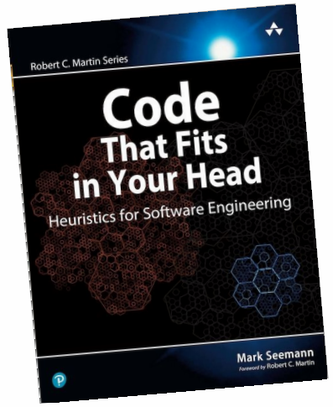
"Indeed, comments are, at best, a necessary evil. If our programming languages were expressive enough, or if we had the talent to subtly wield those languages to express our intent, we would not need comments very much—perhaps not at all."

— Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education, 2008. doi:10.5555/1388398

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. doi:10.1109/icse.2012.6227188

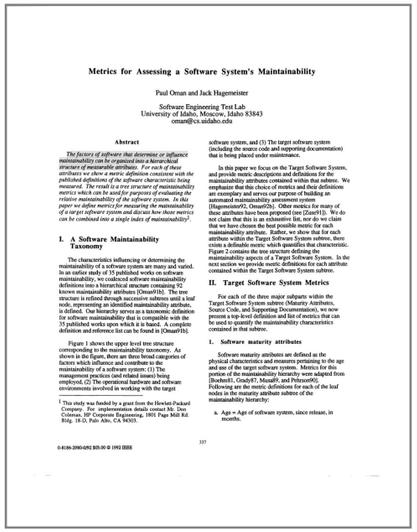


Mark Seemann

"Comments may deteriorate as the code evolves around them. What was once a correct comment becomes misleading as time passes. Ultimately, the only artefact you can trust is the code. Not the comments in the code, but the actual instructions and expressions that are compiled to working software."

— Mark Seemann. Code That Fits in Your Head: Heuristics for Software Engineering, 2021

2. We can measure the amount of comments in a code base.



Metrics for Assessing a Software System's Maintainability

Paul Oman and Jack Hagemeister

Software Engineering Test Lab
University of Idaho, Moscow, Idaho 83843
oman@cs.uidaho.edu

Abstract

The factors of software that determine or influence maintainability can be organized into a hierarchical structure of measurable attributes. For each of these attributes we show a metric definition consistent with the published definitions of the software characteristic being measured. The result is a tree structure of maintainability metrics which can be used for purposes of evaluating the relative maintainability of the software system. In this paper we define metrics for measuring the maintainability of a target software system and discuss how these metrics can be combined into a single index of maintainability[1].

I. A Software Maintainability Taxonomy

The characteristics influencing or determining the maintainability of a software systems are many and varied. In an earlier study of 35 published works on software maintainability, we coalesced software maintainability definitions into a hierarchical structure containing 92 known maintainability attributes [Oman91b]. The tree structure is refined through successive subtrees until a leaf node, representing an identified maintainability attribute, is defined. Our hierarchy serves as a taxonomic definition for software maintainability that is compatible with the 35 published works upon which it is based. A complete definition and reference list can be found in [Oman91b].

Figure 1 shows the upper level tree structure corresponding to the maintainability taxonomy. As shown in the figure, there are three broad categories of factors which influence and contribute to the maintainability of a software system: (1) The management practices (and related issues) being employed, (2) The operational hardware and software environments involved in working with the target software system, and (3) The target software system (including the source code and supporting documentation) that is being placed under maintenance.

In this paper we focus on the Target Software System, and provide metric descriptions and definitions for the maintainability attributes contained within that subtree. We emphasize that this choice of metrics and their definitions are exemplary and serves our purpose of building an automated maintainability assessment system [Hagemeister92, Oman92b]. Other metrics for many of these attributes have been proposed (see [Zuse91]). We do not claim that this is an exhaustive list, nor do we claim that we have chosen the best possible metric for each maintainability attribute. Rather, we show that for each attribute in the Target Software System subtree, there exists a definable metric which quantifies that characteristic. Figure 2 contains the tree structure defining the maintainability aspects of a Target Software System. In the next section we provide metric definitions for each attribute contained within the Target Software System subtree.

II. Target Software System Metrics

For each of the three major subparts within the Target Software System subtree (Maturity Attributes, Source Code, and Supporting Documentation), we now present a top-level definition and list of metrics that can be used to quantify the maintainability characteristics contained in that subtree.

1. Software maturity attributes

Software maturity attributes are defined as the physical characteristics and measures pertaining to the age and use of the target software system. Metrics for this portion of the maintainability hierarchy were adapted from [Boehm81, Grady87, Musa89, and Peterson95]. Following are the metric definitions for each of the leaf nodes in the maturity attribute subtree of the maintainability hierarchy:

a. Age = Age of software system, since release, in months.

[1] This study was funded by a grant from the Hewlett-Packard Company. For implementation details contact Mr. Don Coleman, HP Corporate Engineering, 1801 Page Mill Rd. Bldg. 1B-23, Palo Alto, CA 94303.

337

0-8186-2980-6/92 $03.00 © 1992 IEEE

"[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."

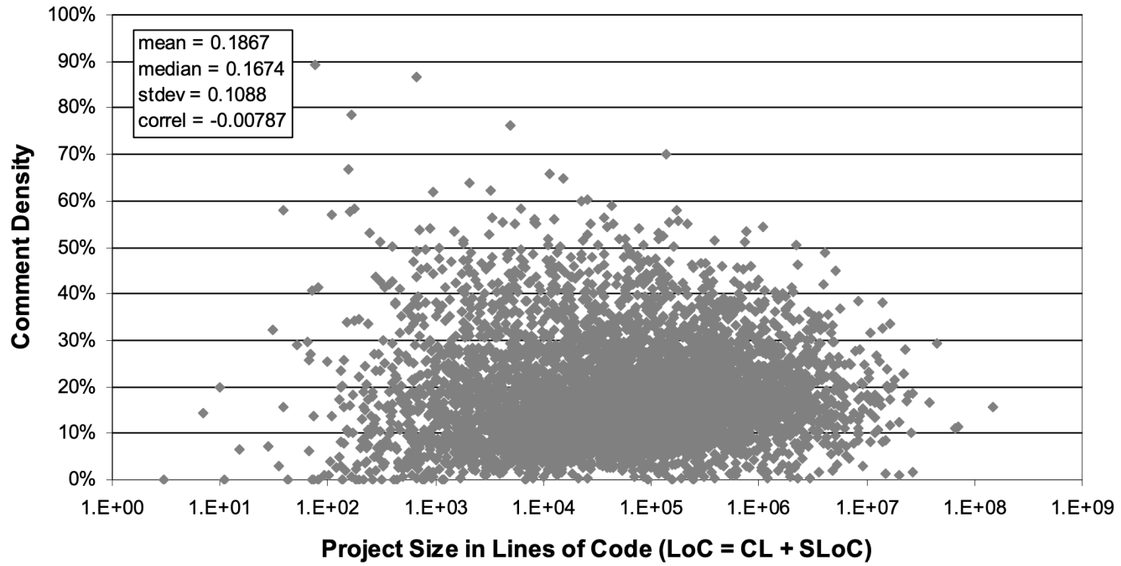
— Paul Oman and Jack Hagemeister. Metrics for Assessing a Software System's Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE, 1992. doi:10.1109/icsm.1992.242525

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. doi:10.1109/icse-companion.2009.5070980

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. doi:10.1109/icse-companion.2009.5070980

Some Open Source Repositories (9 Feb 2024)

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

My Own Statistics (9 Feb 2024)

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

All repositories are open source.

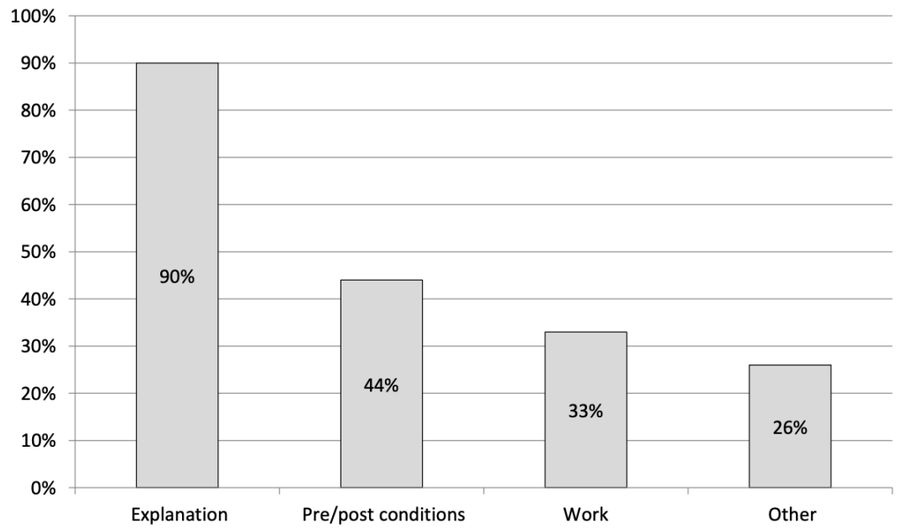
3. Not all comments are the same, some of them are helpful, while others aren't.

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)."

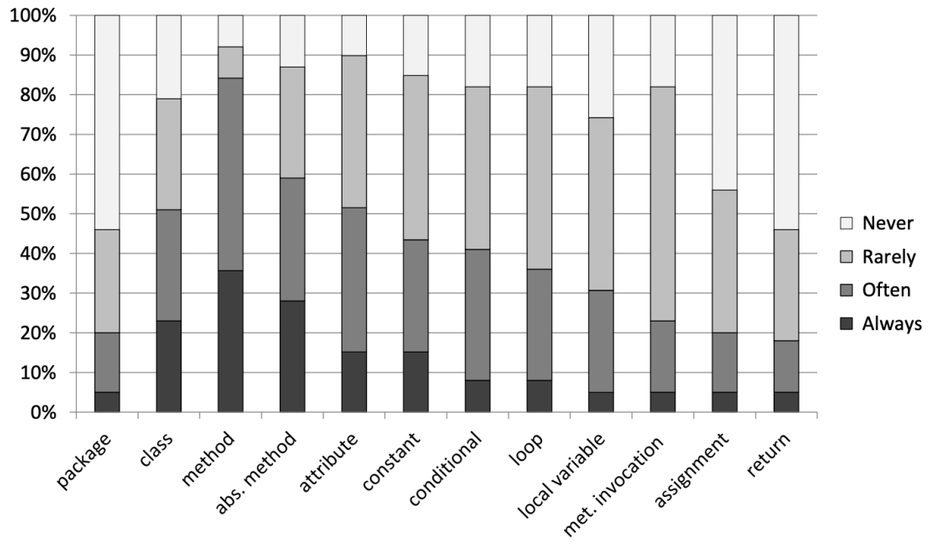
— Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011. doi:10.1109/esem.2011.22

Types of Comments

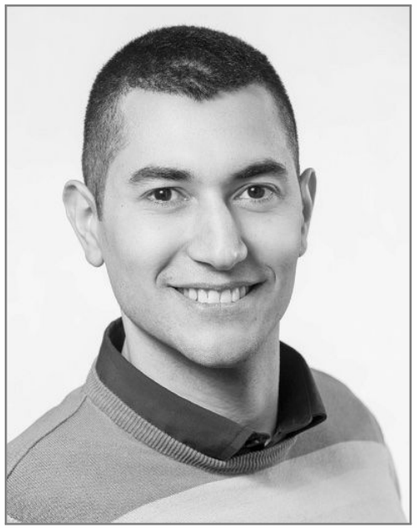


Source: Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011. doi:10.1109/esem.2011.22

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 International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011. doi:10.1109/esem.2011.22

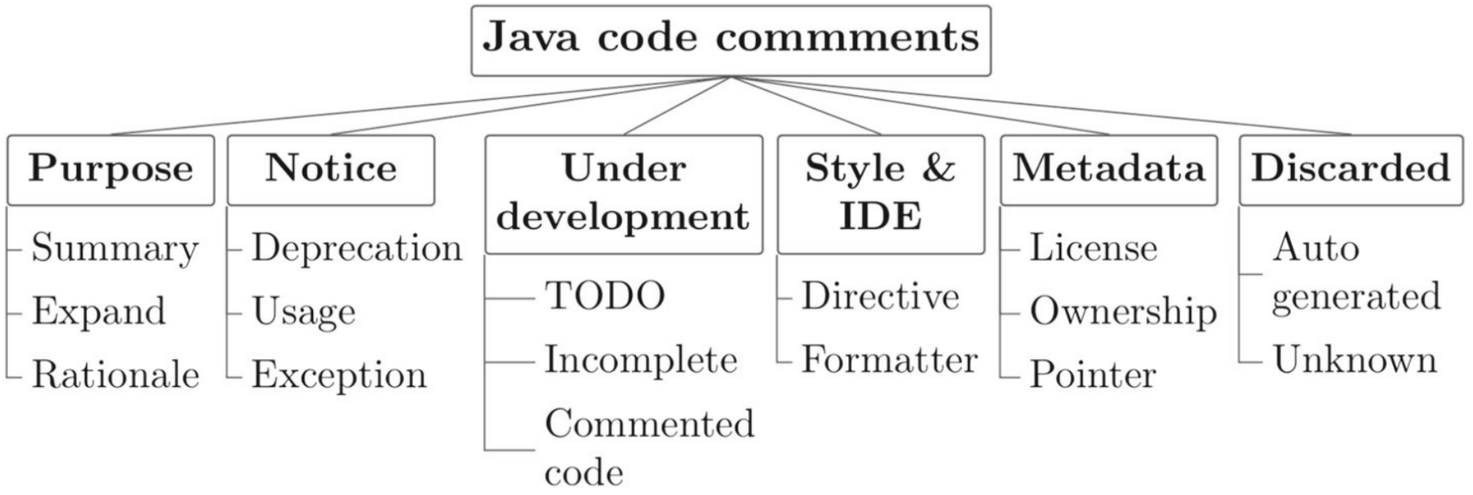


Luca Pascarella

"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. doi:10.1007/s10664-019-09694-w

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. doi:10.1007/s10664-019-09694-w

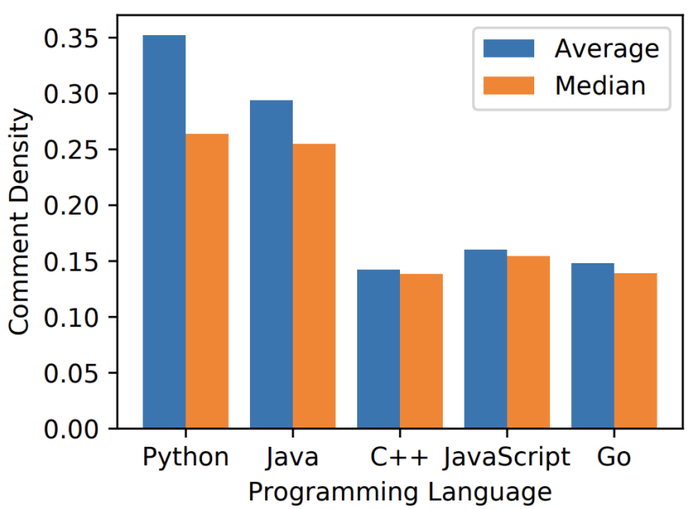
4. Comment density may differ in different programming languages.

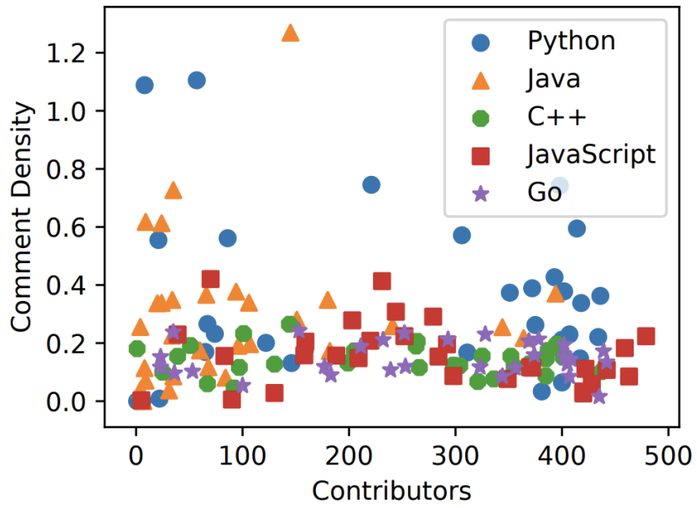
Hao He

"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 Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019. doi:10.1145/3338906.3342494

Comments Density by Language





Source: Hao He. Understanding Source Code Comments at Large-Scale. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019. doi:10.1145/3338906.3342494

5. Comments may be auto-generated by ML/LLM, but they may be less helpful.

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. doi:10.1145/3387904.3389258

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. doi:10.1145/3510003.3510152

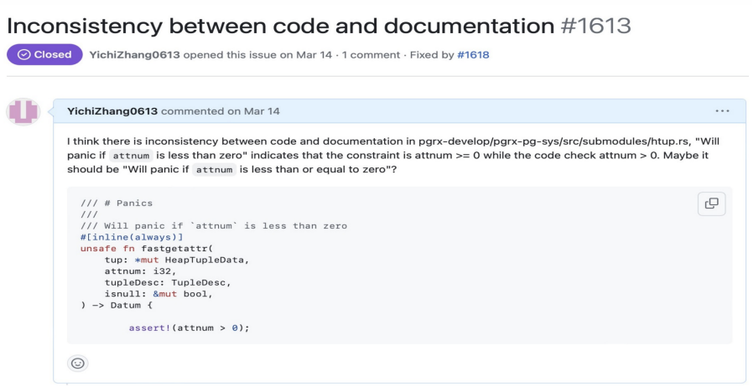


Figure 2: A reported inconsistency example [11].

"In the experiment on 13 open-source projects, 'LLM with program analysis' approach identified 160 inconsistencies between code comments and code design, and 23 of them have been confirmed and fixed by the developers."

Source: Yichi Zhang. Detecting Code Comment Inconsistencies Using LLM and Program Analysis. In *Proceedings of the 32nd International Conference on the Foundations of Software Engineering*, pages 683–685, 2024. doi:10.1145/3663529.3664458

***Answer after generating comments using* AUTOGENICS**

```python
with open("file_path.json", "r") as f:
    # Read the entire JSON string
    json_string = f.read()
    # Convert the JSON string to a Python dictionary
    json_as_dict = json.loads(json_string)
    # Extract the values (user objects) from the dictionary as a list
    list_of_dicts = list(json_as_dict.values())
    # Create a Spark DataFrame from the list of user objects
    df = spark.createDataFrame(list_of_dicts)
```

(b) Answer after adding comments using AUTOGENICS.

Fig. 2: A motivational example [36] where users requested a code explanation to validate its accuracy contrasted with the same answer improved by AUTOGENICS-generated comments.

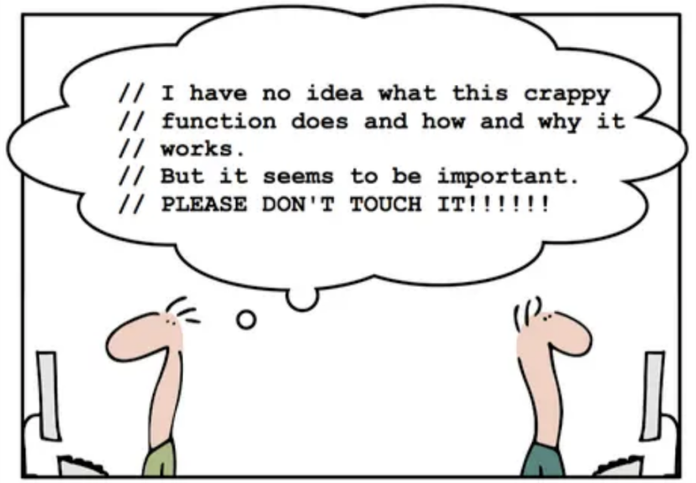
"The majority of participants found the LLM-generated inline comments effective and expressed a strong demand for tool support. In particular, they preferred an automated inline commenting tool as a browser plugin on the StackOverflow site."

Source: Suborno Deb Bappon, Saikat Mondal, and Banani Roy. AUTOGENICS: Automated Generation of Context-Aware Inline Comments for Code Snippets on Programming Q&A Sites Using LLM. In *Proceedings of the International Conference on Source Code Analysis and Manipulation (SCAM)*, pages 24–35. IEEE, 2024. doi:10.1109/SCAM63643.2024.00013

6. A few practical hints may help you write better comments.

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://jttu.net/spertus2021, 2021

Bibliography

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. doi:10.1109/icse-companion.2009.5070980.

Suborno Deb Bappon, Saikat Mondal, and Banani Roy. AUTOGENICS: Automated Generation of Context-Aware Inline Comments for Code Snippets on Programming Q&A Sites Using LLM. In *Proceedings of the International Conference on Source Code Analysis and Manipulation (SCAM)*, pages 24–35. IEEE, 2024. doi:10.1109/SCAM63643.2024.00013.

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. doi:10.1109/icsm.1996.564992.

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. doi:10.5555/800099.803210.

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*, pages 70–79. IEEE, 2007. doi:10.1109/wcre.2007.21.

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. doi:10.5555/311424.

Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. How Good Is Your Comment? A Study of Comments in Java Programs. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 137–146. IEEE, 2011. doi:10.1109/esem.2011.22.

Hao He. Understanding Source Code Comments at Large-Scale. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1217–1219, 2019. doi:10.1145/3338906.3342494.

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. doi:10.1145/3510003.3510152.

Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999. doi:10.5555/320326.

Brian W. Kernighan and Phillip James Plauger. *The Elements of Programming Style*. McGraw-Hill, Inc, 1974. doi:10.5555/601121.

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398.

Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143.

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 130–131. IEEE, 2003. doi:10.1109/fie.2003.1263332.

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, 1992. doi:10.1109/icsm.1992.242525.

David Lorge Parnas. Software Aging. In *Proceedings of the 16th International Conference on Software Engineering*, pages 279–287. IEEE, 1994. doi:10.1109/icse.1994.296790.

Luca Pascarella, Magiel Bruntink, and Alberto Bacchelli. Classifying Code Comments in Java Software Systems. *Empirical Software Engineering*, 24(3):1499–1537, 2019. doi:10.1007/s10664-019-09694-w.

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. doi:10.1109/icse.2012.6227188.

Mark Seemann. Code That Fits in Your Head: Heuristics for Software Engineering, 2021.

Ellen Spertus. Best Practices for Writing Code Comments. `https://jttu.net/spertus2021`, 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. doi:10.1145/3387904.3389258.

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. doi:10.5555/800078.802534.

Yichi Zhang. Detecting Code Comment Inconsistencies Using LLM and Program Analysis. In *Proceedings of the 32nd International Conference on the Foundations of Software Engineering*, pages 683–685, 2024. doi:10.1145/3663529.3664458.