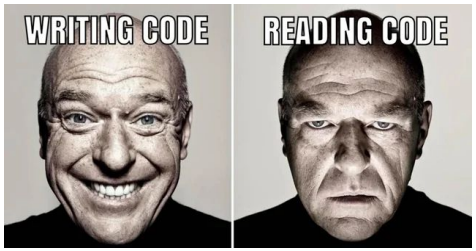
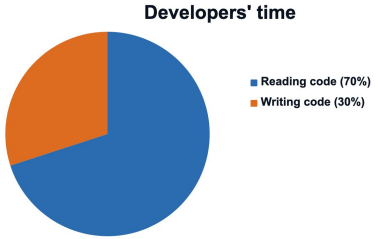
Understanding Code Understandability Improvements in Code Reviews

Zhiheng Lyu 21143093

Developers Spend 60–70% of Time Reading Code

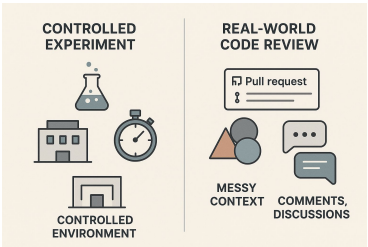
- Reading is the dominant activity, not typing code
- 58–70% of development time spent understanding existing code
- Maintaining unreadable code wastes time & money
- Understandability directly impacts productivity & maintainability





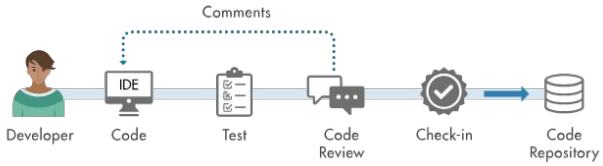
Prior Research: Controlled Experiments ≠ Real World

- **Lab studies** test readability via isolated snippets；They can't capture context, conventions, or developer background
- **Real-world** understandability depends on project culture & team norms; What looks "clear" varies across teams (e.g., ternary vs. if-statement)



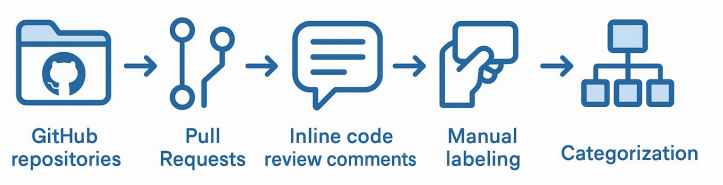
RQ: How Do Developers Improve Understandability in Practice?

- How do developers improve understandability during real code reviews?
- What kinds of issues are flagged by reviewers?
- Which kinds of changes get accepted?



Method: Mining Code Review Comments from GitHub

- 363 active OSS Java projects on GitHub
- 349,000 code review comments collected
- Stratified sampling → 2,401 comments manually analyzed
- Intention + patch change examined to confirm understandability



Inline Code Review: Understandability Suggestions Are Explicit

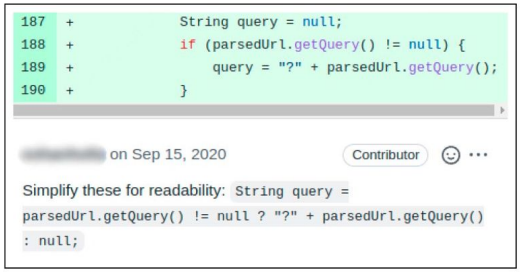


Fig. 1. Inline code review suggestion to replace an `if` statement by a ternary expression [14].

Inline Code Review: Same Issue, but Opposite Suggestions

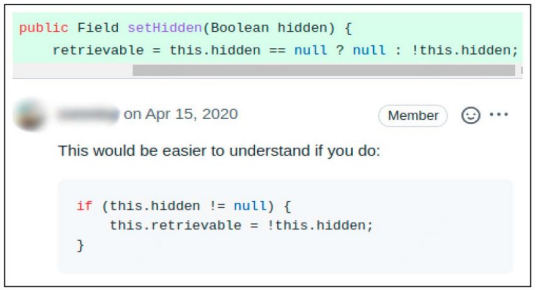


Fig. 2. Inline code review suggestion to replace a ternary expression by an `if` statement [22].

Code Review: Explicit vs Implicit suggestions

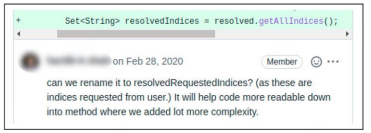


Fig. 5. The reviewer explicitly asks for understandability improvement [63].

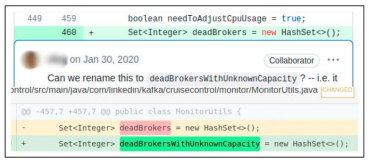
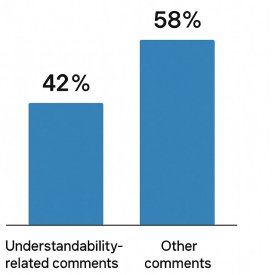


Fig. 6. The reviewer implicitly asks for understandability improvement [64].

Result #1: How often?

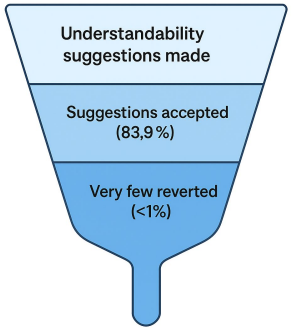
- 42.15% of comments suggest understandability improvements
- Understandability > correctness/performance (in comment frequency)
- Not just code — also applies to docs, logs, string literals



Result #2: What is the main issue?

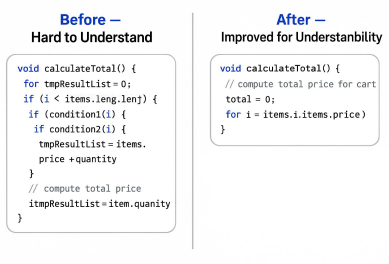
Result #3: How likely accepted?

- 83.9% readability suggestions are applied
- Less than 1% reverted later
- Indicates these changes are **stable, reliable, low-risk**



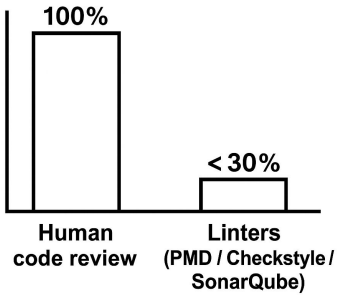
Result #5: How actually improved?

- Developers improve readability using **behavior-preserving refactors**
- Most frequent actions:
  - Rename identifiers (variables / methods / classes)
  - Extract method or simplify logic
  - Remove unnecessary / dead code
  - Improve documentation, comments, or logs
- Mostly Changes clarify intent, not functionality



Result #5: Do Linters Work?

- Popular linters: PMD, Checkstyle, SonarQube, SpotBugs
- <30% of real understandability issues are detected
- Human reviewers catch nuance static rules miss



Implications for LLM-Based Code Tools

- Data proves developers frequently improve readability
- Actions are consistent → great training material
- Use accepted patches as preference data
- AI agents can suggest readability edits **proactively**

Conclusion

Key findings:

- 42% of Review Comments Aim to Improve Understandability
- Eight Categories of Understandability Smells
- 83.9% of Suggestions Are Accepted — and Rarely Reverted (<1%)
- Linters Cover Less Than 30% of Real Understandability Issues

Takeaways:

- **Understandability** is one of the most frequent concerns in code reviews.
- These changes are highly **stable** and **reliable** — they are almost never reverted.
- There is significant untapped potential for AI/LLMs to automatically detect and refactor understandability issues

Review: Positive

- The study bridges the gap between controlled readability experiments and real-world developer behavior.
- It captures human reasoning from actual review conversations — signals richer than synthetic metrics (e.g., naming length, cyclomatic complexity).
- Findings are strong and actionable: 83.9% of understandability suggestions are accepted and rarely reverted, indicating high-quality and reliable change patterns.
- The dataset enables future LLM/agent systems to learn what human developers perceive as “clear” or “maintainable” code.

Review: Negative

- The approach relies heavily on **manual labeling** and subjective judgment, limiting scalability.
- The analysis is retrospective — it **explains** improvements after review comments appear, rather than **predicting** unclear code proactively.
- The taxonomy may be language- and project-specific (based on **Java** OSS projects), and generalizability to Python or industrial codebases remains uncertain.

Rating & Future Work

Rating: 4 / 5

Future Work:

- Build semi-automatic or LLM-based detectors that flag understandability smells proactively.
- Extend the dataset across multiple languages and industrial repositories to test cultural variation.
- Use accepted understandability edits as positive training examples for preference modeling or RL to teach LLMs what “good code” looks like.

Discussion Points & QA

- Can LLMs infer understandability issues from review conversations without explicit labels?
- Could proactive “review copilots” detect unclear code before submission, transforming code review from reactive to preventive?
- How much of understandability depends on project culture, and how could tooling adapt to different teams’ norms?

The End.

Thanks for your listening.