Code Review

Learning goals

- Be able to explain why code review is beneficial.

- Be able to conduct a code review.

Testing has Limitations

- Costly to get 100% coverage (all code / behaviors)
  - 80/20 rule!

- Not all properties can be checked at runtime
  - Good design?
  - Simple implementation? Understandable code?
  - Follows coding conventions?
  - UI looks as intended? Follows UI guidelines?
  - Are the tests adequate (coverage, kind)?

Code Review

- *Systematic* reading or examination of the code

- Focused on what can't be tested (cost-benefit)

- Sometimes done in pairs or groups, often asynchronous
  - at least one is non-author
    (authors can't see flaws in their code)
  - find & work through more complex problems (e.g., design)
  - promote learning and knowledge transfer (not just QA!)
  - super valuable for "onboarding" new devs
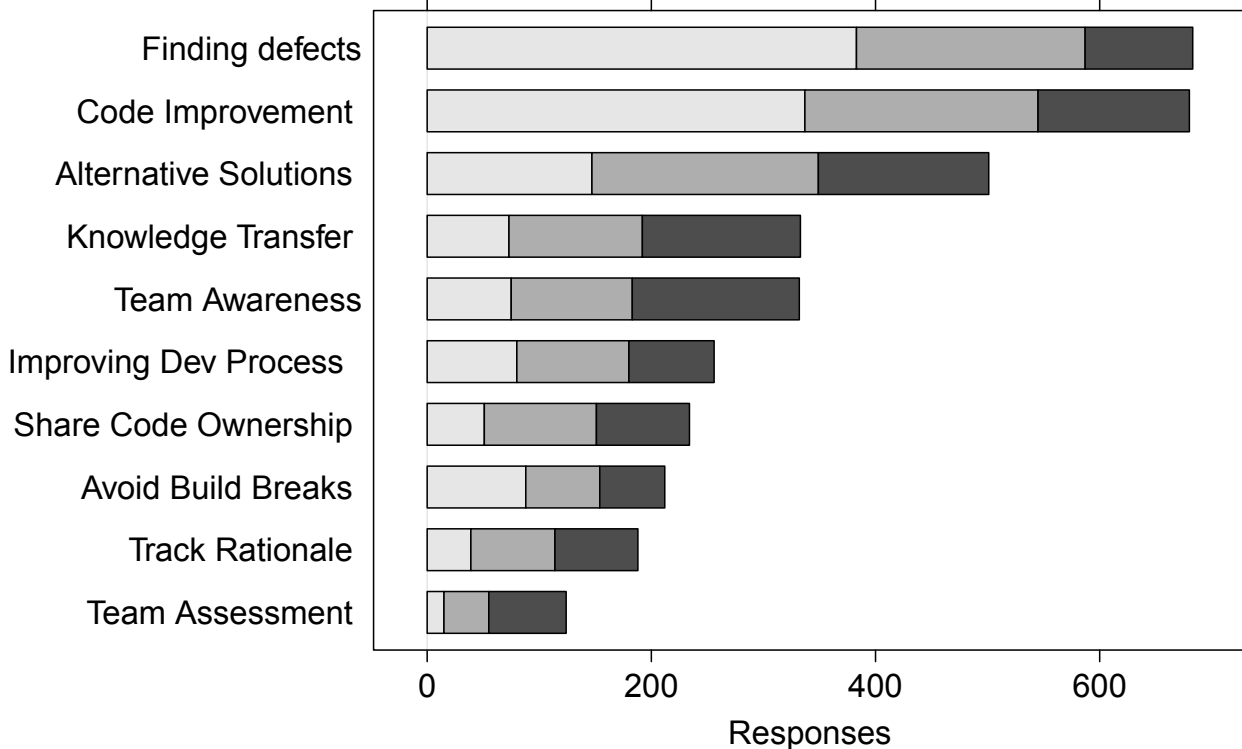 - ***pair programming is instantaneous code review***

History

- Previously: formal code review ("inspection")

  - Sit in a meeting, read all code

  - Have been found effective at finding bugs

  - Too slow for practical use (not done in most settings)

- Now: change-based code review

  - Every change gets reviewed by someone

  - Various policies: who can review?

Motivations and Benefits (Bacchelli et al.)

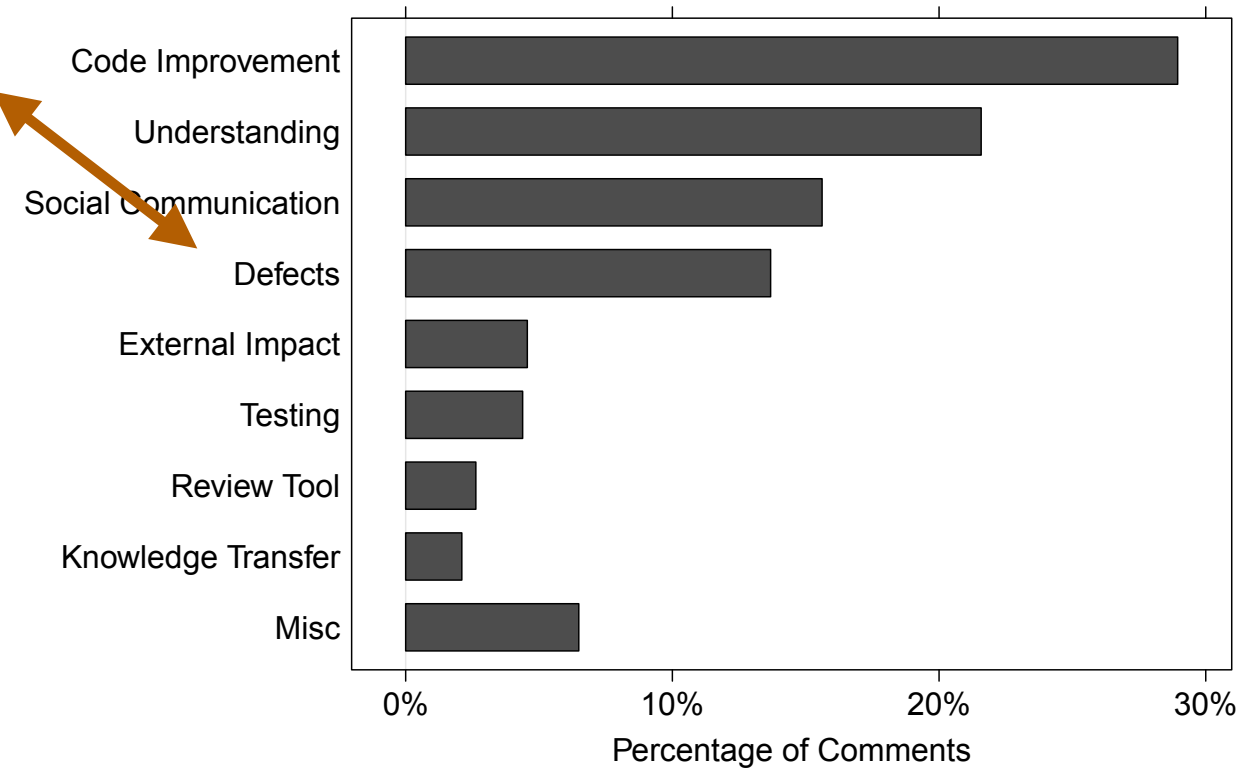**Ranked Motivations From Developers**

Top □ Second ▨ Third ■

Fig. 3. Developers' motivations for code review.

Comments in each Category

Benefits of code review (according to analysis of 200 code threads)

Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, 712–721.

Code Review at Google

- Each directory is owned by certain people

  - An owner must review and approve changes

- "Readability": ensure consistent style

  - Developers can be certified for individual languages

  - Every change must be written or reviewed by someone with "readability" certification in the appropriate language

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE–SEIP '18). https://doi.org/10.1145/3183519.3183525

Google Process

1. Create a change

2. Authors preview results of static analyzers

3. Reviewers write comments; *unresolved* comments must be addressed

4. Addressing feedback: author changes code or replies to comments

5. Approving: reviewers mark "LGTM"

Productivity?

- How many commits do you think the median Google developer makes each week?

A. 1

B. 2-3

C. 4-7

D. 8-10

E. > 10

Stats

- Median developer authors about 3 changes a week

- 80 percent of authors make fewer than 7 changes a week

- Median is 4 reviewers/developer

- 80 percent of reviewers review fewer than 10 changes a week.

- Median time: < 1 hour for small changes, about 5 hours for very large changes. All changes: 4 hours.

More Google stats

- > 35% of changes only modify one file

- 90% modify < 10 files

- 10% modify one line of code

- Median number of lines: 24

Code review productivity

- Recommendation: <= 200 LOC/hour

  - C. F. Kemerer and M. C. Paulk, "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data," in IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 534-550, July-Aug. 2009, doi: 10.1109/TSE.2009.27.

- You'll likely find you tend to go faster.

Tone

- The code is the team's code, not your code

- Use "we" language, not "you" language

- Avoid blame

  - "you have a bug here" -> "this code might be buggy"

- "What if…"

Review breakdowns (what *not* to do)

- Power (use reviews to induce unrelated behavior)

- Subject: is this the right place to do design?

- Context: why are we doing this?

Newbies write more comments

- Newbies ask more questions
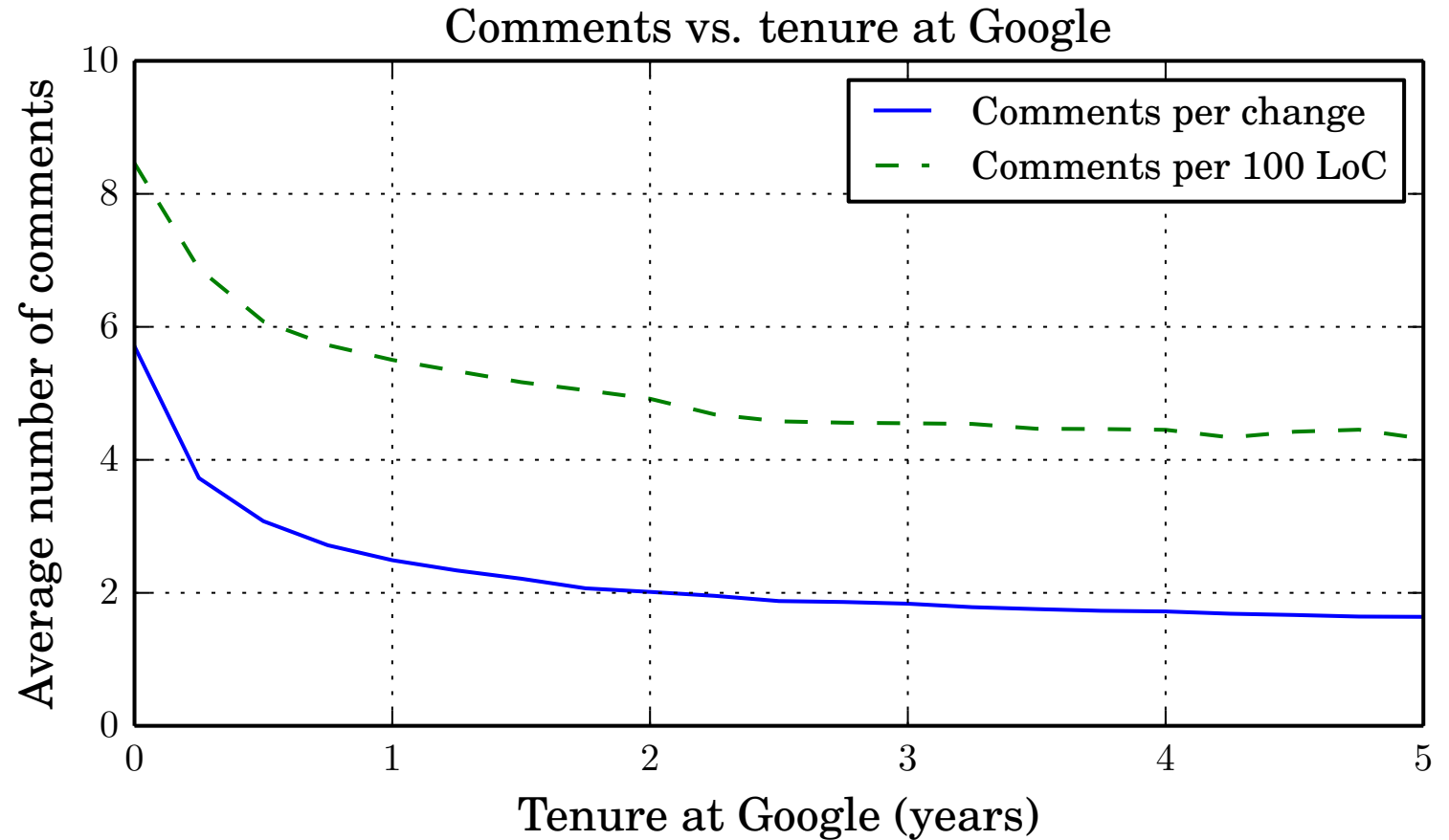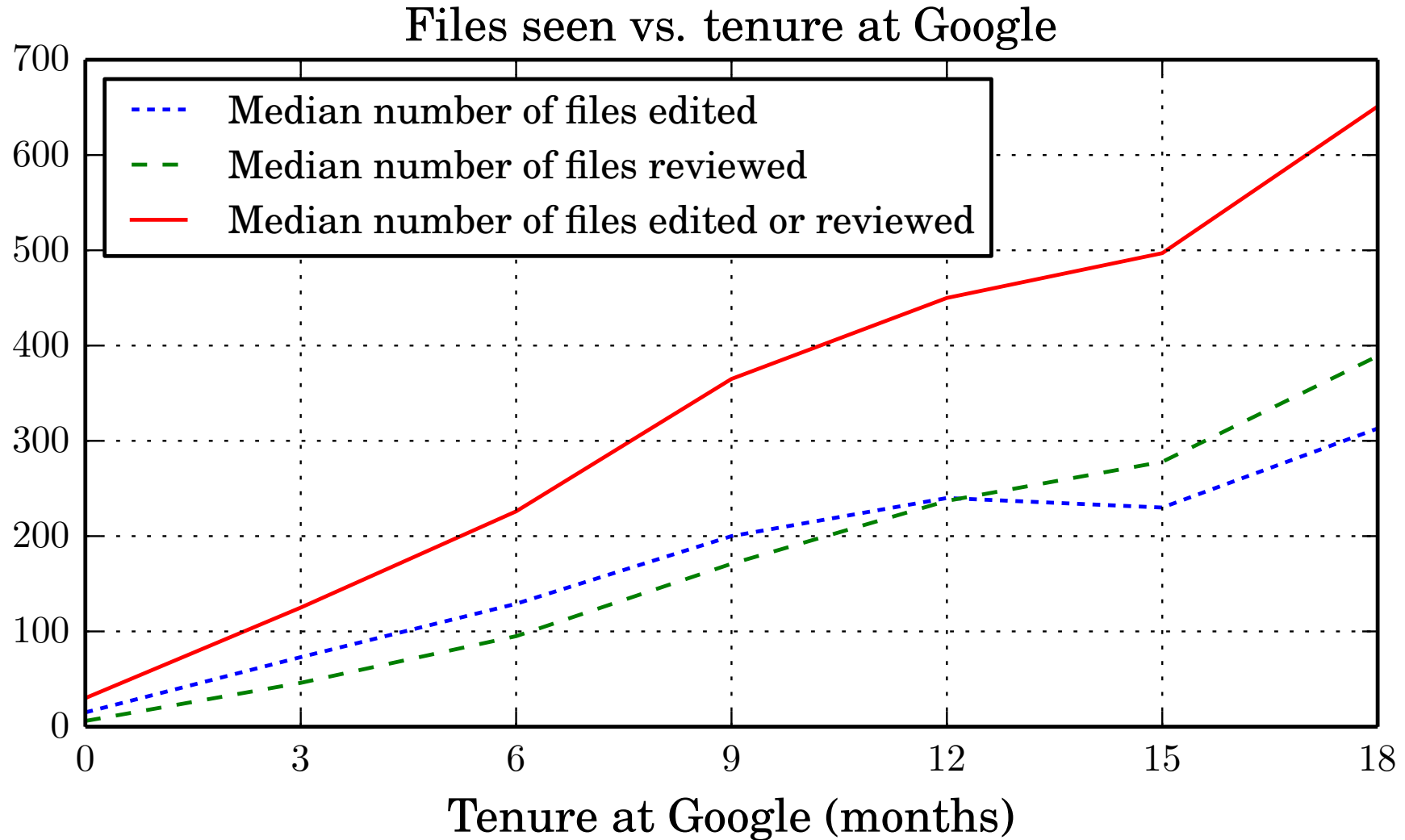
- But questions are considered unhelpful

Figure 2: Reviewer comments vs. author's tenure at Google

Files vs. time

Files seen vs. tenure at Google

Legend:
- Median number of files edited
- Median number of files reviewed
- Median number of files edited or reviewed

X-axis: Tenure at Google (months)

What comments are most useful?

- Identification of functional issues (though these are relatively rare)

- Validation issues, corner cases

- For new developers: API suggestions, design ideas, coding conventions

- Somewhat useful: nit-picking (identifier naming, comments); refactoring ideas

- Not useful: questions, future tasks

Systematic Review: How

- **Use checklists** to remind reviewers what to look for
  - E.g., expanded list of properties from slide #1

- Specific techniques for specific issues
  - Design is reviewed by working through likely change(s) (Is the code OCP for likely changes?)

- Use tools in GitHub or IDE
  - List of changed files
  - Textual diff between old and new files (linked to files)
  - Line-level code commenting support
  - work-flow support for choosing/assigning reviewers
  - protecting main branch

Fixed issue 147.

Browse file

```
Integral values (byte, short, integer, long, BigInteger) are now comparable to each other.
Floating point values (float, double, BigDecimal) are now comparable to each other.
```

⎇ master 🏷 gson-parent-2.8.2 ... gson-2.4

🟩 inder123 committed on Sep 23, 2009   1 parent 50eb582   commit fdcd3945c53c4a1c921ea8097cbecbbf154f

📄 Showing **2 changed files** with **134 additions** and **1 deletion**.

Unified S

🟠 gson/src/main/java/com/google/gson/JsonPrimitive.java   +43 −1 ⬛

🟠 gson/src/test/java/com/google/gson/JsonPrimitiveTest.java   +91 −0 ⬛

44 🟩🟩🟩🟩⬜ gson/src/main/java/com/google/gson/JsonPrimitive.java   View

```
     @@ -344,7 +344,19 @@ private static boolean isPrimitiveOrString(Object target) {
344                                                          344
345         @Override                                        345         @Override
346         public int hashCode() {                          346         public int hashCode() {
347  -       return (value == null) ? 31 : value.hashCode();  347  +       if (value == null) {
                                                              348  +          return 31;
                                                              349  +       }
                                                              350  +       // Using recommended hashing algorithm from Effective Java for longs and
                                                                          doubles
                                                              351  +       if (isIntegral(this)) {
                                                              352  +          long value = getAsNumber().longValue();
```

A Checklist for Your Project

1. Good design?
   - ☐ Isomorphic to requirements
   - ☐ Sound like the requirements
   - ☐ SRP
   - ☐ Open-closed principle (OCP) for likely changes

2. Straightforward implementation?
   - ☐ Understandable code
   - ☐ Good choice of data structures

3. Follows coding conventions?
   - ☐ formatting (indents, spacing, line breaks)
   - ☐ naming conventions (sound like behavior)

4. UI looks as intended, fits guidelines

5. Code look correct?
   - ☐ Omitted cases (e.g., boundary/edge cases)
   - ☐ Off-by-one errors (e.g., "<" instead of "<=")

6. Are the tests adequate (coverage)?
   - ☐ Unit, Story tests

- Not strictly ordered by importance
- If fail at a step, *can* skip less impt. steps (low cost/benefit to continue)
- E.g., Hard to debug complex code

```java
1   public static boolean leap(int y) {
2     String t = String.valueOf(y);
3     if (t.charAt(2) == '1' || t.charAt(2) == '3' || t.charAt(2)
4   == 5 || t.charAt(2) == '7' || t.charAt(2) == '9') {
5       if (t.charAt(3)=='2'||t.charAt(3)=='6') return true;
6       else
7         return false;
8     }else{
9       if (t.charAt(2) == '0' && t.charAt(3) == '0') {
10        return false;
11      }
12      if (t.charAt(3)=='0'||t.charAt(3)=='4'||
13  t.charAt(3)=='8')return true;
14    }
15    return false;
16  }
```

*http://web.mit.edu/6.005/www/fa15/classes/04-code-review/

Feedback for your teammate?

- variable naming – unclear

- hard to read – formatting/indentation

- call same functions multiple times with same numbers
  - name temp vars, extract functions (make code sound like what it's doing)

- uses strings; should use integer calculations
  - maybe could use shift…really modulus

- assumes 4 digit number…future dates, historical dates
  - we don't know the context of use

- use of "true" and "false" rather than returning boolean

- 5 is not a character

Worst problem? Unnecessarily complex.

```
// https://www.timeanddate.com/date/leapyear.html
public static boolean isLeapYear(int year) {
  return year % 4 == 0 &&
        (year % 100 !== 0 || year % 400 == 0);
}
```

- Found a simpler approach
- Method name and parameter sound like the requirements
- Comment citing approach
- Formatted for readability