

Code Review

Martin Kellogg

Code Review

Today's agenda:

- **Finish slides on interviewing**
- Reading Quiz
- What is code review (and why we do it)
- How to do a code review (with empirical evidence)
- Good and bad examples of code review comments

Interview format

- “For about 45 minutes you meet with a single technical interviewer, who will present a programming problem and ask you to work out one or more solutions to it.”
 - some variations of this, such as “tell me about a technical problem you’ve solved” and “design (but don’t implement) a solution to this problem”
- Interviewer perspective: “you know in the first ten minutes”

Example interview problem

“The Two-Sum Problem”:

- You are given an array of n integers and a number k . Determine if there is a pair of elements in the array that sums to exactly k .
- For example, given the array $[1, 3, 7]$ and $k = 8$, the answer is “yes,” but given $k = 6$ the answer is “no.”

Example interview problem

“The Two-Sum Problem”:

- You are given an array of n integers and a number k . Determine if there is a pair of elements in the array that sums to exactly k .
- For example, given the array $[1, 3, 7]$ and $k = 8$, the answer is “yes,” but given $k = 6$ the answer is “no.”

What do you do first?
(Hint: it's not trying to solve the problem!)

Example interview problem: ask questions!

Example interview problem: ask questions!

- Can you modify the array? Yes.
- Do we know something about the range of the numbers in the array? No, they can be arbitrary integers.
- Are the array elements necessarily positive? No, they can be positive, negative, or zero.
- Do we know anything about the value of k relative to n or the numbers in the array? No, it can be arbitrary.
- Can we consider pairs of an element and itself? No, the pair should consist of two different array elements.
- Can the array contain duplicates? Sure, that's a possibility.
- What about integer overflow? Don't worry about it.

Example interview problem: brute force

- don't prematurely optimize your solution: write something that works

Example interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
boolean sumsToTarget (int[]arr, int k) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[i] + arr[j] == k) {  
                return true;  
            } } }  
    return false;  
}
```

Example interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
boolean sumsToTarget (int[]arr, int k) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[i] + arr[j] == k) {  
                return true;  
            } } }  
    return false;  
}
```

Usually at this point the interviewer will ask you about **how good** this solution is.

Example interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
boolean sumsToTarget (int[]arr, int k) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[i] + arr[j] == k) {  
                return true;  
            } } }  
    return false;  
}
```

Usually at this point the interviewer will ask you about **how good** this solution is. $O(n^2)$ time!

Example interview problem: be clever

- once you have something that works, try to come up with a clever solution

Example interview problem: be clever

- once you have something that works, try to come up with a clever solution
- e.g., for the two-sum problem, you might come up with:

```
boolean sumsToTarget (int[]arr, int k) {  
    HashSet < Integer > values = new HashSet < Integer > ();  
    for (int i = 0; i < arr.length; i++) {  
        if (values.contains (k - A[i])) return true;  
        values.add (A[i]);  
    }  
    return false;  
}
```

Example interview problem: be clever

- once you have something that works, try to come up with a clever solution
- e.g., for the two-sum problem, you might come up with:

```
boolean sumsToTarget (int[]arr, int k) {  
    HashSet < Integer > values = new HashSet < Integer > ();  
    for (int i = 0; i < arr.length; i++) {  
        if (values.contains (k - A[i])) return true;  
        values.add (A[i]);  
    }  
    return false;  
}
```

Why is this better?

Example interview problem

- there are lots of possible solutions to the problem
- part of your goal while you're interviewing is showing that you understand the trade-offs between them
- think of interviewing as a **microcosm** of software engineering:
 - if you don't show them you know it, they'll assume you don't

Example interview problem

- there are lots of possible solutions to the problem
- part of your goal while you're interviewing is showing that you understand the trade-offs between them
- think of interviewing as a **microcosm** of software engineering:
 - if you don't show them you know it, they'll assume you don't
 - implication: even though the interview problem is **small and simple**, you show try to **show all the steps** of the software engineering process

Do Not Forget!

Even though the problem is small, you should:

- perform requirements elicitation (ask questions!)
- ask about both functional and non-functional properties
- talk about process considerations
 - e.g., mention maintainability when relevant
- write **good quality code**, including e.g., documentation, tests, etc.
 - mention things you'd be thinking about if this was part of a real system

Interviewing mistakes

- #1 Practicing on a computer
- #2 Not rehearsing behavioral questions
- #3 Not doing a mock interview
- #4 Trying to memorize solutions
- #5 Not solving problems out loud
- #6 Rushing
- #7 Sloppy coding (bad style)
- #8 Not testing
- #9 Fixing mistakes carelessly
- #10 Giving up

Interviewing mistakes

#1 Practicing on a computer

#2 Not rehearsing behavioral questions

#3 Not doing a mock interview

#4 Trying to memorize solutions

#5 Not solving problems out loud

#6 Rushing

#7 Sloppy coding (bad style)

#8 Not testing

#9 Fixing mistakes carelessly

#10 Giving up

Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!
 - What is your greatest weakness?
 - Tell me about a time you missed a deadline.
 - Tell me about a time you experienced a conflict with a teammate.

Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!
 - What is your greatest weakness?
 - Tell me about a time you missed a deadline.
 - Tell me about a time you experienced a conflict with a teammate.

It's easy to sound unimpressive if you haven't thought about your answers ahead of time.

Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

What is your greatest weakness?

How can they tell about a time you missed a deadline.

if you are nice? about a time you experienced a conflict with a

Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

What is your greatest weakness?

How can they tell about a time you missed a deadline.

if you are nice? about a time you experienced a conflict with a

lunch!

Interviewing: the other side

Interviewing: the other side

- Choose the technical problem you ask carefully
 - Common solution: use the “best” interview question you’ve ever been asked
 - Alternative: base the problem on something you personally had to deal with at work
- Think through all the possible solutions to the problem
- Remember that it’s stressful for the person being interviewed!

Interviewing: does it work?

- The answer is that we don't really know
- Technical interviews haven't been studied in depth
- But they're the industry standard, so we have to deal with them
- Open area of research!

Teams/Interviewing Takeaways

- How you organize your team can have a big impact on your productivity
- Communication is key
- For the group project, especially, make sure you decide on how you'll make decisions (no one is the manager!)
- Interviewing is a microcosm of software engineering
 - Show the interviewer what you know, even if it seems like too much for the problem at hand

Code Review

Today's agenda:

- Finish slides on interviewing
- **Reading Quiz**
- What is code review (and why we do it)
- How to do a code review (with empirical evidence)
- Good and bad examples of code review comments

Reading Quiz: Code Review

Q1: **TRUE** or **FALSE**: Because aspects of software design are usually a pure style issue or just a personal preference, the Google style guide says that it is okay to ignore reviewer feedback that you disagree with, as long as you are polite.

Q2: If you ask a developer to explain a piece of code that you don't understand while reviewing, that should usually result in:

- A. a comment in the code review tool that explains the code
- B. rewriting the code to make it more clear
- C. a code comment explaining the code

Reading Quiz: Code Review

Q1: **TRUE** or **FALSE**: Because aspects of software design are usually a pure style issue or just a personal preference, the Google style guide says that it is okay to ignore reviewer feedback that you disagree with, as long as you are polite.

Q2: If you ask a developer to explain a piece of code that you don't understand while reviewing, that should usually result in:

- A. a comment in the code review tool that explains the code
- B. rewriting the code to make it more clear
- C. a code comment explaining the code

Reading Quiz: Code Review

Q1: **TRUE** or **FALSE**: Because aspects of software design are usually a pure style issue or just a personal preference, the Google style guide says that it is okay to ignore reviewer feedback that you disagree with, as long as you are polite.

Q2: If you ask a developer to explain a piece of code that you don't understand while reviewing, that should usually result in:

- A. a comment in the code review tool that explains the code
- B. rewriting the code to make it more clear
- C. a code comment explaining the code

Code Review

Today's agenda:

- Finish slides on interviewing
- Reading Quiz
- **What is code review (and why we do it)**
- How to do a code review (with empirical evidence)
- Good and bad examples of code review comments

What is code review?

Definition: In a *code review*, another developer examines your proposed change and explanation, offers feedback, and decides whether to accept it.

What is code review?

Definition: In a *code review*, another developer examines your proposed change and explanation, offers feedback, and decides whether to accept it.

- There is significant **tool support** for “modern” code review
 - We’ll talk about this in more depth later in this lecture

Analogy: writing

Compare the effectiveness of:

- spell checking your own writing
- reading and editing your own writing
- having your writing be edited by someone else

Analogy: writing

Compare the effectiveness of:

- spell checking your own writing
- reading and editing your own writing
- having your writing be edited by someone else

Professional writers have **editors**; professional software engineers have **code reviewers**

What is(n't) “modern” code review?

- Historically, “code review” used to refer to what we now call *code inspection* or *holistic code review*.

What is(n't) “modern” code review?

- Historically, “code review” used to refer to what we now call *code inspection* or *holistic code review*.

Definition: a *holistic code review* is a code review of an entire component of a software system as a whole.

What is(n't) “modern” code review?

- Historically, “code review” used to refer to what we now call *code inspection* or *holistic code review*.

Definition: a *holistic code review* is a code review of an entire component of a software system as a whole.

- Typically, “code inspection” suggests that a team of reviewers is involved, while “holistic code review” suggests a single reviewer (but these are connotations, not rules)

What is(n't) “modern” code review?

- Historically, “code review” used to refer to what we now call **code inspection** or **holistic code review**.

Definition: a **holistic code review** is a code component of a software system as a whole

- Typically, “code inspection” suggests a reviewer is involved, while “holistic code review” suggests a reviewer (but these are connotations)

History fact: there was a lot of interest (and research) into code inspection in the 80s/90s (at the same time that Waterfall was the dominant methodology)

So then what is modern code review?

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

Definition: a **modern code review** is a review of a set of proposed changes to a codebase, typically performed by another developer who is already familiar with the code being changed

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

Definition: a **modern code review** is a review of a set of proposed changes to a codebase, typically performed by another developer who is already familiar with the code being changed

- Inductive argument for code quality:
 - if $v(n)$ is good, and the diff between $v(n)$ and $v(n+1)$ is good, then $v(n+1)$ is good

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

Definition: a **modern code review** is a review of proposed changes to a codebase, typically by a peer developer who is already familiar with the codebase

One reason you should care about this lecture: you are **required** to do modern code reviews on **all code** that you write for your group project

- Inductive argument
 - if $v(n)$ is good, and the difference between $v(n)$ and $v(n+1)$ is good, then $v(n+1)$ is good

Modern code review: intuition

- “Given enough eyeballs, all bugs are shallow.” – Linus's Law

Modern code review: intuition

- “Given enough eyeballs, all bugs are shallow.” – Linus's Law
- Reviewer has:
 - different background, different experience
 - no preconceived idea of correctness
 - no bias because of “what was intended”

Modern code review: intuition

“Breadth of experience in an individual is essential to creativity and hence to good engineering. ... Collective diversity, or diversity of the group - the kind of diversity that people usually talk about - is just as essential to good engineering as individual diversity. ... Those **differences in experience are the "gene pool" from which creativity springs.**”

– Bill Wulf, National Academy of Engineering President

Modern code review: the most common analysis

- Modern code review is considered a **best practice almost everywhere** in industry

Modern code review: the most common analysis

"All code that gets submitted **needs to be reviewed** by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian to do code reviews, and obviously, **we spend a good chunk of our time reviewing code.**"

- Amanda Camp, Software Engineer, Google

Modern code review: the most common analysis

“At Yelp we use review-board. An engineer works on a branch and commits the code to their own branch. The reviewer then goes through the diff, adds inline comments on review board and sends them back. The **reviews are meant to be a dialogue**, so typically comment threads result from the feedback. Once the reviewer's questions and concerns are all addressed they'll click "Ship It!" and the author will merge it with the main branch for deployment the same day.”

- Alan Fineberg, Software Engineer, Yelp

Modern code review: the most common analysis

“At Wizards we use Perforce for SCM. I work with stuff that manages rules and content, so we try to commit changes at the granularity of one bug at a time or one card at a time. Our team is small enough that you can designate one other person on team as a code reviewer. Usually you look at code sometime that week, but it depends on priority. **It's impossible to write sufficient test harnesses** for the bulk of our game code, so **code reviews are absolutely critical.**”

- Jake Englund, Software Engineer, MtGO

Modern code review: the most common analysis

"At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, she submits it to this tool, which will notify the person or people she has asked to review the change, along with others that may be interested in the change – such as people who have worked on a function that got changed. At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."

Ryan McElroy, Software Engineer, Facebook

Modern code review: the most common analysis

- Modern code review is considered a **best practice almost everywhere** in industry
- While each place has their own way of doing reviews, the broad strokes are common between companies

Modern code review: benefits

Modern code review: benefits

- > 1 person has seen every piece of code
 - **Insurance** against author's disappearance (recall: bus factor)
 - **Accountability** (both author and reviewers are accountable)

Modern code review: benefits

- > 1 person has seen every piece of code
 - **Insurance** against author's disappearance (recall: bus factor)
 - **Accountability** (both author and reviewers are accountable)
- Forcing function for **documentation** and code improvements
 - Authors must articulate their decisions
 - Prospect of a review raises your quality threshold

Modern code review: benefits

- > 1 person has seen every piece of code
 - **Insurance** against author's disappearance (recall: bus factor)
 - **Accountability** (both author and reviewers are accountable)
- Forcing function for **documentation** and code improvements
 - Authors must articulate their decisions
 - Prospect of a review raises your quality threshold
- Inexperienced personnel get **experience** without hurting quality
 - Pairing them up with experienced developers
 - Can learn by being a reviewer as well

Modern code review: benefits

- > 1 person has seen every piece of code
 - **Insurance** against human error (human factor)
 - **Accountability** (but not quantifiable)
- Forcing function for code quality
 - Authors must articulate their intent
 - Prospect of a review
- Inexperienced person can learn
 - Pairing them up with an experienced person
 - Can learn by being a reviewer as well

Non-goal: assessing whether the author is good at their job

- managers/HR **shouldn't** be involved in code review

Modern code review: benefits by the numbers

Modern code review: benefits by the numbers

- Average defect detection rates higher than testing
- 11 programs developed by the same group of people
 - First 5 without reviews: average 4.5 errors / 100 LoC
 - Remaining 6 with reviews: average 0.82 errors / 100 LoC
 - Errors reduced by > 80%.
- IBM's Orbit project: 500,000 lines, 11 levels of inspections. Delivered early with 1% of the predicted errors.
- After AT&T introduced reviews, 14% increase in productivity and a 90% decrease in defects.

(From Steve McConnell's [Code Complete](#))

Code Review

Today's agenda:

- Finish slides on interviewing
- Reading Quiz
- What is code review (and why we do it)
- **How to do a code review (with empirical evidence)**
- Good and bad examples of code review comments

Author checklist before sending out a CL

Author checklist before sending out a CL

- Review it yourself

Author checklist before sending out a CL

- Review it yourself
- Make sure the diff is clean

Author checklist before sending out a CL

- Review it yourself
- Make sure the diff is clean

Avoid:

- extraneous whitespace changes
- debugging code
- commented-out code
- style guide violations
- undocumented code
- etc.

Author checklist before sending out a CL

- Review it yourself
- Make sure the diff is clean
- Choose the right reviewers

Author checklist before sending out a CL

- **Review it yourself**
- Make sure the diff is **clean**
- Choose the **right reviewers**

Factors to consider in a reviewer:

- availability (how many reviews are they already working on?)
- code ownership
- code expertise
- readability

Author checklist before sending out a CL

- **Review it yourself**
- Make sure the diff is **clean**
- Choose the **right reviewers**

Factors to consider in a reviewer:

- availability (how many reviews are they already working on?)
- code ownership
- code expertise
- **readability**

Aside: “readability”

How to do a code review: Google's principles

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.
- On matters of style, the **style guide** is the absolute authority

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.
- On matters of style, the **style guide** is the absolute authority
- Aspects of software design are **almost never** a pure style issue or just a personal preference.

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.
- On matters of style, the **style guide** is the absolute authority
- Aspects of software design are **almost never** a pure style issue or just a personal preference.
 - weigh options on **principles**, not simply by personal opinion

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.
- On matters of style, the **style guide** is the absolute authority
- Aspects of software design are **almost never** a pure style issue or just a personal preference.
 - weigh options on **principles**, not simply by personal opinion
- If no other rule applies, then the reviewer may ask the author to be **consistent** with what is in the current codebase

How to do a code review: Google's principles

- Technical facts and data **override** opinions and personal preferences.
- On matters of style, the **style guide** is the absolute authority
- Aspects of software design are **almost never** a pure style issue or just a personal preference.
 - weigh options on **principles**, not simply by personal opinion
- If no other rule applies, then the reviewer may ask the author to be **consistent** with what is in the current codebase
- reviewers should favor approving a CL once it is in a state where it definitely **improves the overall code health** of the system

How to do a code review: Google's principles

- I'll add one more:
 - **Don't be a jerk**

What to look for in a code review

- Design/complexity:

What to look for in a code review

- Design/complexity:
 - Does this change **belong** in *this* code, or somewhere else?

What to look for in a code review

- Design/complexity:
 - Does this change **belong** in *this* code, or somewhere else?
 - Are all the parts of the change **related** enough, or should this really be two (or more) PRs

What to look for in a code review

- Design/complexity:
 - Does this change **belong** in *this* code, or somewhere else?
 - Are all the parts of the change **related** enough, or should this really be two (or more) PRs
 - Is it **easy to understand** how the parts fit together?

What to look for in a code review

- Design/complexity:
 - Does this change **belong** in *this* code, or somewhere else?
 - Are all the parts of the change **related** enough, or should this really be two (or more) PRs
 - Is it **easy to understand** how the parts fit together?
 - Does each unit of code (class, method, etc.) follow **good code-level design principles**?

What to look for in a code review

- Design/complexity:
 - Does this change **belong** in *this* code, or somewhere else?
 - Are all the parts of the change **related** enough, or should this really be two (or more) PRs
 - Is it **easy to understand** how the parts fit together?
 - Does each unit of code (class, method, etc.) follow **good code-level design principles**?
 - Is it **over-engineered**?

Aside: over-engineering

Definition: a design is *over-engineered* if it anticipates problems for which will never occur in practice

Aside: over-engineering

Definition: a design is *over-engineered* if it anticipates problems for which will never occur in practice

- over-engineered code is **harder to change**, because it has abstractions that aren't necessary

Aside: over-engineering

Definition: a design is *over-engineered* if it anticipates problems for which will never occur in practice

- over-engineered code is *harder to change*, because it has abstractions that aren't necessary
- defense against over-engineering: do not add an abstraction to deal with a *purely theoretical* problem
 - demand to see evidence that a problem actually exists!

Aside: over-engineering

Definition: a design is *over-engineered* if it anticipates problems for which will never occur in practice

- over-engineered code is **harder to change**, because it has abstractions that aren't necessary
- defense against over-engineering
 - deal with a **purely theoretical** demand to see evidence

Large danger of over-engineering in code reviews: designing for changes that you **don't know** whether you will need. Advice: err on the side of not supporting such changes

What to look for in a code review

- Functionality/testing:

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?
 - Are there **tests**? Are the tests testing the right thing?

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?
 - Are there **tests**? Are the tests testing the right thing?
 - Is it a change to a user interface? If so, has someone **actually looked** at the new UI?

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?
 - Are there **tests**? Are the tests testing the right thing?
 - Is it a change to a user interface? If so, has someone **actually looked** at the new UI?

Especially relevant for **course projects**, since Covey.Town is UI-heavy

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?
 - Are there **tests**? Are the tests testing the right thing?
 - Is it a change to a user interface? If so, has someone **actually looked** at the new UI?
 - Is the code doing something **difficult to understand** (such as concurrency)?

What to look for in a code review

- Functionality/testing:
 - Is it clear what the change is **supposed** to do?
 - Are there **tests**? Are the tests testing the right thing?
 - Is it a change to a user interface? If so, has someone **actually looked** at the new UI?
 - Is the code doing something **difficult to understand** (such as concurrency)?
 - If so, pay extra attention and **prove to yourself** that it is correct.

How to write code review comments

How to write code review comments

- **Be kind**, courteous, and respectful.

How to write code review comments

- **Be kind**, courteous, and respectful.
- **Explain** your reasoning.

How to write code review comments

- **Be kind**, courteous, and respectful.
- **Explain** your reasoning.
- **Balance** giving explicit directions with just pointing out problems and letting the developer decide.

How to write code review comments

- **Be kind**, courteous, and respectful.
- **Explain** your reasoning.
- **Balance** giving explicit directions with just pointing out problems and letting the developer decide.

“In general it is the **developer’s responsibility** to fix a CL, not the reviewer’s”

How to write code review comments

- **Be kind**, courteous, and respectful.
- **Explain** your reasoning.
- **Balance** giving explicit directions with just pointing out problems and letting the developer decide.
- **Insist** developers simplify code or add code comments instead of just explaining the complexity to you.

How to write code review comments

- **Be kind**, courteous, and respectful.
- **Explain** your reasoning.
- **Balance** giving explicit directions with just pointing out problems and letting the developer decide.
- **Insist** developers simplify code or add code comments instead of just explaining the complexity to you.

“Explanations written only in the code review tool are not helpful to **future code readers**”

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:
 - **Must Fix**: I don't think I can approve this CL until this problem is fixed, even if everything else is perfect.

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:
 - **Must Fix**: I don't think I can approve this CL until this problem is fixed, even if everything else is perfect.

Usually authors treat comments without a severity level as **must fix**

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:
 - **Must Fix**: I don't think I can approve this CL until this problem is fixed, even if everything else is perfect.
 - **Nit**: This is a minor thing. Technically you should do it, but it won't hugely impact things.

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:
 - **Must Fix**: I don't think I can approve this CL until this problem is fixed, even if everything else is perfect.
 - **Nit**: This is a minor thing. Technically you should do it, but it won't hugely impact things.
 - **Optional**: I think this may be a good idea, but it's not strictly required.

How to write code review comments: severity

- Label comments with their **severity**, to avoid misunderstandings:
 - **Must Fix**: I don't think I can approve this CL until this problem is fixed, even if everything else is perfect.
 - **Nit**: This is a minor thing. Technically you should do it, but it won't hugely impact things.
 - **Optional**: I think this may be a good idea, but it's not strictly required.
 - **FYI**: I don't expect you to do this in this CL, but you may find this interesting to think about for the future.

Common mistakes to avoid as a reviewer

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.

If you get **pushback** on a suggestion,
take the time to understand why

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.

Try to get back to the author within
“**one business day**”, whatever that
means for your team

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**

Common mistake: “LGTM” everything
for the sake of speed

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**
- Being **inconsistent**

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**
- Being **inconsistent**

I've had reviewers ask for one thing, which I do, and then ask for something completely different a week later. **Read your previous review!**

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**
- Being **inconsistent**
- Letting complexity through with a promise to **clean up** later

Common mistakes to avoid as a reviewer

- Being **condescending**, especially if you're wrong.
- Taking **too long** to complete a review.
- Being **too lax**
- Being **inconsistent**
- Letting complexity through with a promise to **clean up** later

Doesn't usually happen! If the problem is serious, insist on fixing it now!

Common mistakes to avoid as an author

Common mistakes to avoid as an author

- Respond to **every comment**

Making a code change counts as a response!
Don't write "fixed" or similar on every comment.

Common mistakes to avoid as an author

- Respond to **every comment**
- If you fix something in one place, **fix it everywhere**

As a reviewer, it is very tedious to point out every place that an author has made the same mistake.

Common mistakes to avoid as an author

- Respond to **every comment**
- If you fix something in one place, **fix it everywhere**
- Assume **good faith**

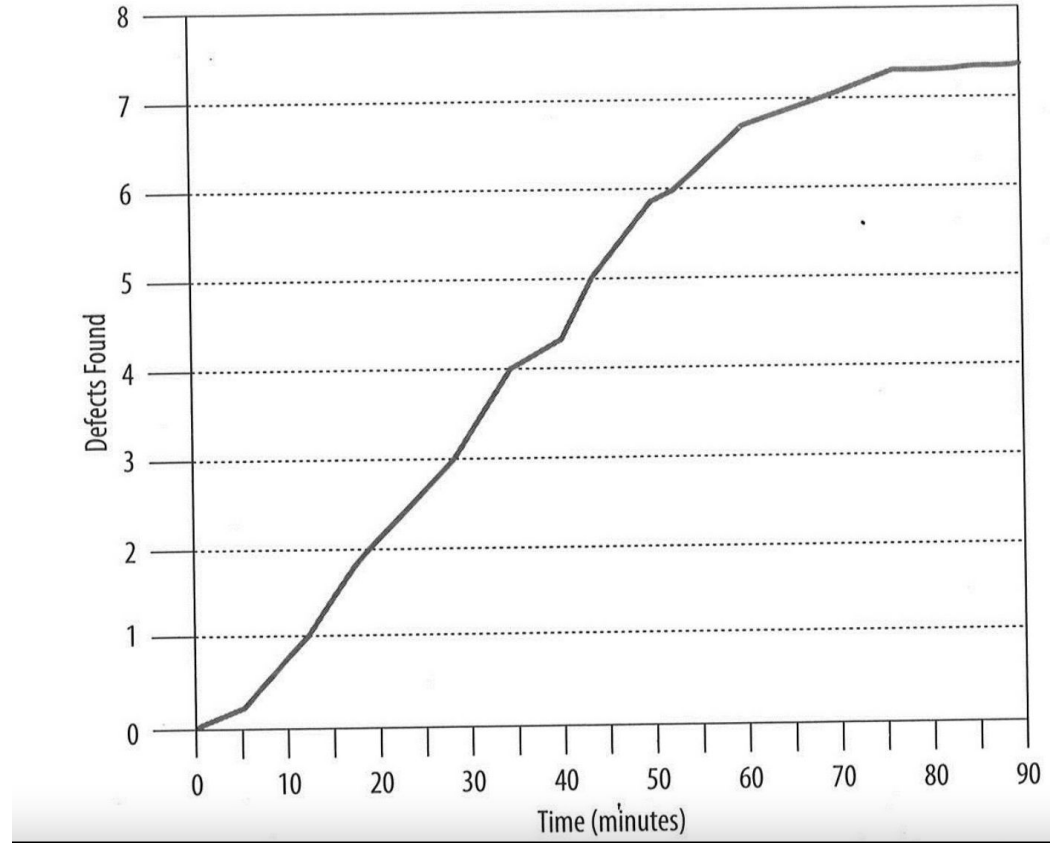
Common mistakes to avoid as an author

- Respond to **every comment**
- If you fix something in one place, **fix it everywhere**
- Assume **good faith**
- Address comments **by changing the code**, not by explaining in the review tool

Empirical guidelines for code review

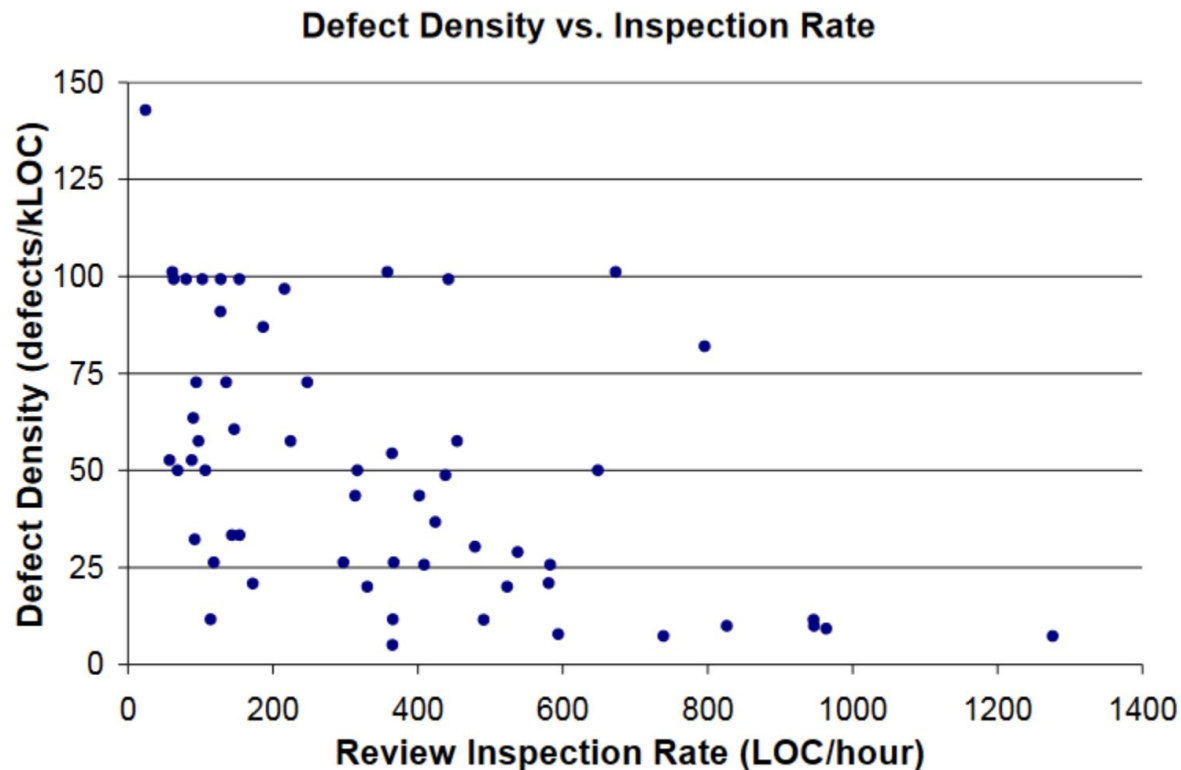
Empirical guidelines for code review

- **Recommendation:**
Do not exceed 60
minute session
- **Reason:** focus
fatigue



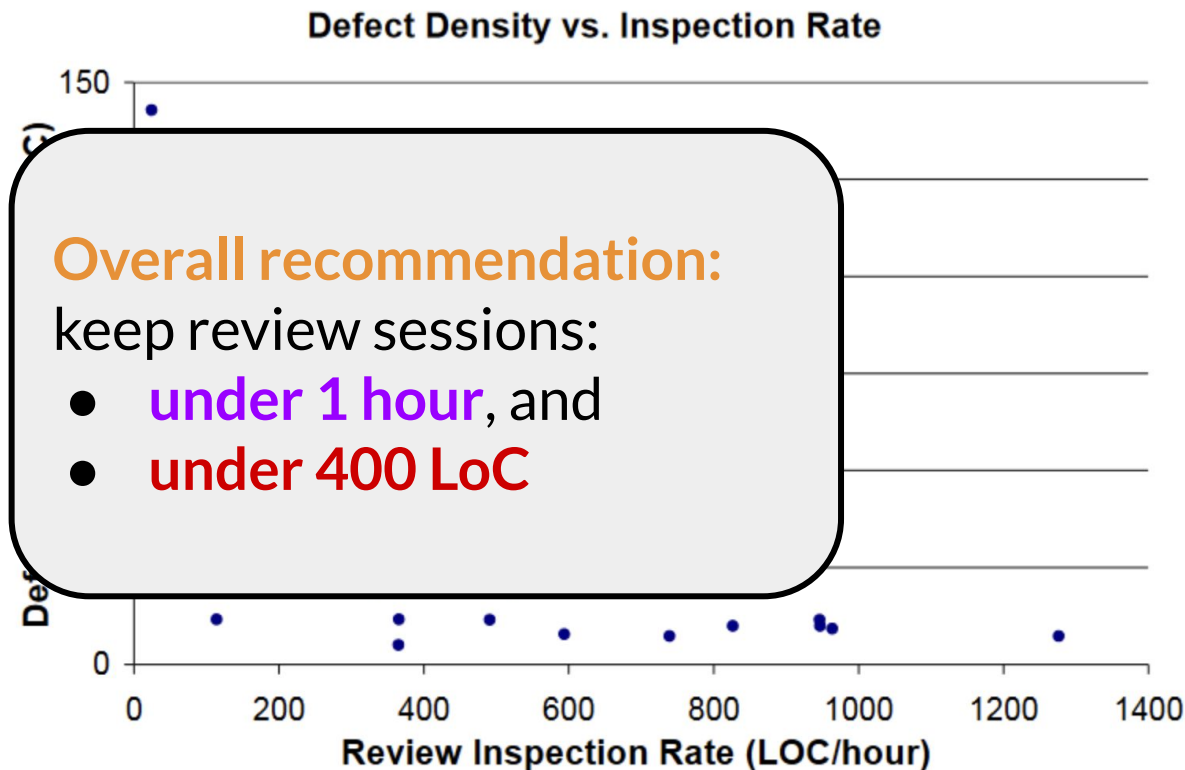
Empirical guidelines for code review

- **Recommendation:**
Don't review more than 400 LoC per hour
- **Reason:** at faster paces, reviews get too shallow

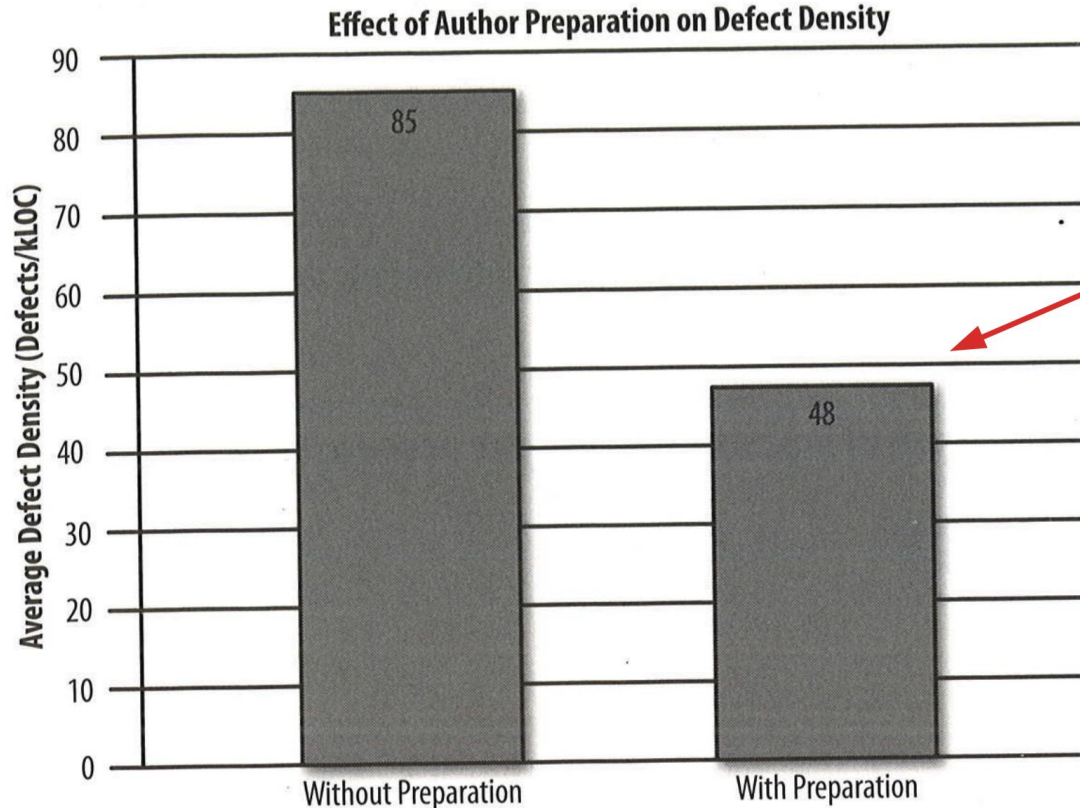


Empirical guidelines for code review

- **Recommendation:**
Don't review more than 400 LoC per hour
- **Reason:** at faster paces, reviews get too shallow



Empirical guidelines for code review



Important to
review your own
code before giving
it to others

Code Review

Today's agenda:

- Finish slides on interviewing
- Reading Quiz
- What is code review (and why we do it)
- How to do a code review (with empirical evidence)
- **Good and bad examples of code review comments**

Example comment: good or bad?

[Many of the examples in the following slides borrowed from Sandya Sankarram's ["Unlearning toxic behaviors in a code review culture"](#)]

Example comment: good or bad?

108 + videos: [],

ssnkr 2 minutes ago
extra space

Reply...

Start a new conversation

109 + navItems: [],

ssnkr 2 minutes ago
extra space

Reply...

Start a new conversation

110 + currentChannel: '927',

ssnkr 2 minutes ago
extra space

Reply...

Start a new conversation

111 + pages: [],

112 + loading: false,

ssnkr 2 minutes ago
extra space

Reply...

Example comment: good or bad?



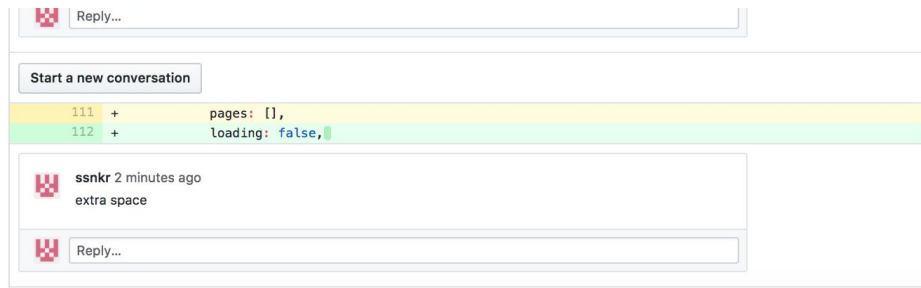
Example comment: good or bad?



ssnkr commented 2 minutes ago



Looks like you checked in some trailing spaces on several lines of your change set. Our style guide specifies no trailing whitespace. Can you take a look at this?



Example comment: good or bad?



BETTER: consolidate the comment in one place rather than repeating yourself

ssnkr commented 2 minutes ago

Looks like you checked in some trailing spaces on several lines of your change set. Our style guide specifies no trailing whitespace. Can you take a look at this?



Example comment: good or bad?

108

+

videos: [],



ssnr 2 minutes ago



Example comment: good or bad?

108

+

videos: [],



ssnr 2 minutes ago



BAD! frankly, this
is just rude. Use
your words!

Example comment: good or bad?



ssnr commented 2 minutes ago



LGTM 100 🎉

Example comment: good or bad?



ssnr commented 2 minutes ago

LGTM 100 🎉

OK: emojis and similar “casual” language should only be used to praise, never to criticize

Example comment: good or bad?



anon-reviewer

I don't mean we're mean-spirited. I just mean that we are merciless. You'll notice that I left the comment "Beep!" on the imports of every file you touched. What I meant was, "Your imports violate our standard convention — we order them by built-ins, then third party, and then project level," but that was too much to type on every file.

Example comment: good or bad?



anon-reviewer

I don't mean we're mean-spirited. I just mean that we are merciless. You'll notice that I left the comment "Beep!" on the imports of every file you touched. What I meant was, "Your imports violate our standard convention" and then project l

VERY BAD!

rude, condescending, and sarcastic.
Be helpful, not antagonistic

Example comment: good or bad?



anon-reviewer

This breaks when you enter a negative number. Can you please address this case?

Example comment: good or bad?



anon-reviewer

This breaks when you enter a negative number. Can you please address this case?

GOOD: straight to the point, politely points out a technical problem

Takeaways

- Code review is one of the best ways to prevent defects
 - You must do it during the course project (I will check!)
- Be nice as both an author and a reviewer
 - Respect each other and each other's time
- One thing I'll look for when assessing your group project is the quality of your code reviews
 - If you're unsure, you can ask the course staff to review your reviews (in office hours)