Lecture 09

ECE 1145: Software Construction and Evolution

Code Review

Announcements

- Iteration 3 due Oct. 3
  - Refactoring, Strategy Pattern
  - Introduce Strategy in several places – make a refactoring/test list and divide up the work
- Next week's assignment will be **code review**
  - Provide another group access to your Iteration 3 code (Release3)
  - Perform code review by Oct. 10
    - Meet with / contact the other group to clarify any questions, provide comments with appropriate context
    - Use time in class next week
- Then, Iteration 4 will be code quality improvements

Announcements

- Resources
  - Google Code Review Standards, What to look for
    - https://google.github.io/eng-practices/review/reviewer/standard.html
    - https://google.github.io/eng-practices/review/reviewer/looking-for.html
  - Compassionate – Yet Candid – Code Reviews, April Wensel https://www.youtube.com/watch?v=Ea8EiIPZvh0
  - Code Review Guidelines for Humans, Philipp Hauer https://phauer.com/2018/code-review-guidelines/
  - *Implementing Effective Code Reviews* (CH 11), Giuliana Carullo https://learning.oreilly.com/library/view/implementing-effective-code/9781484261620/html/

Questions for Today

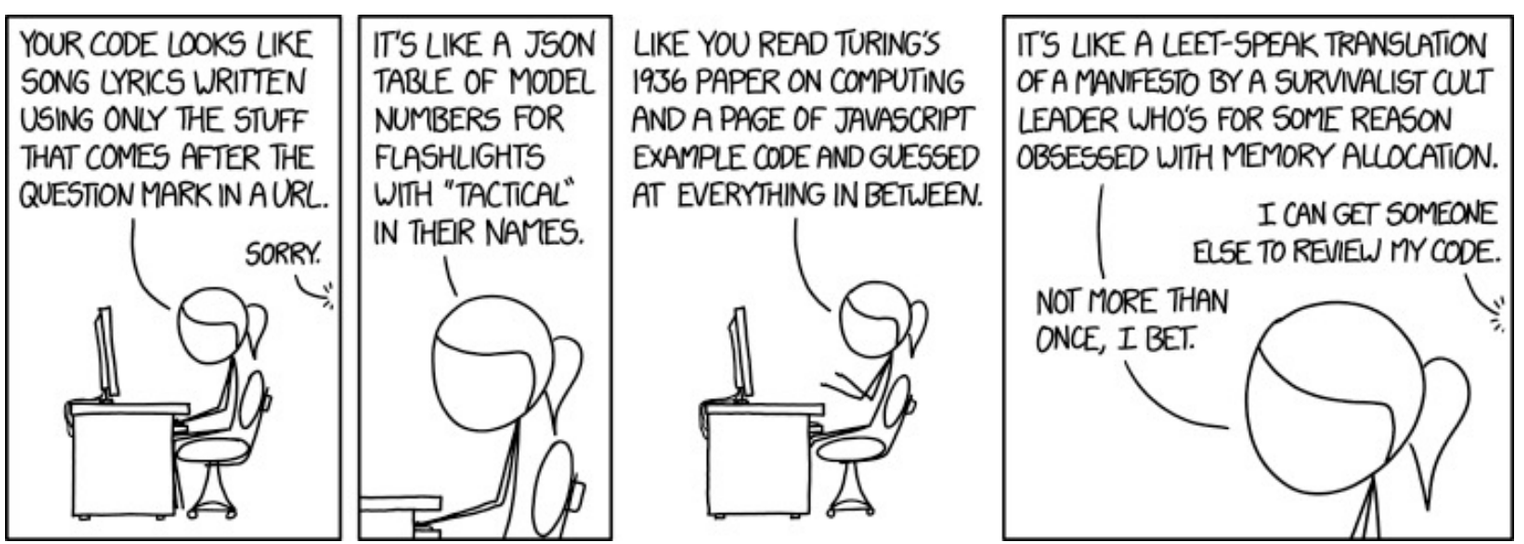
How do we ensure our code is understandable?

How do we conduct effective code reviews?

What is code review?

Let other people read your code (and vice versa) and provide suggestions or make changes.

Why code review?

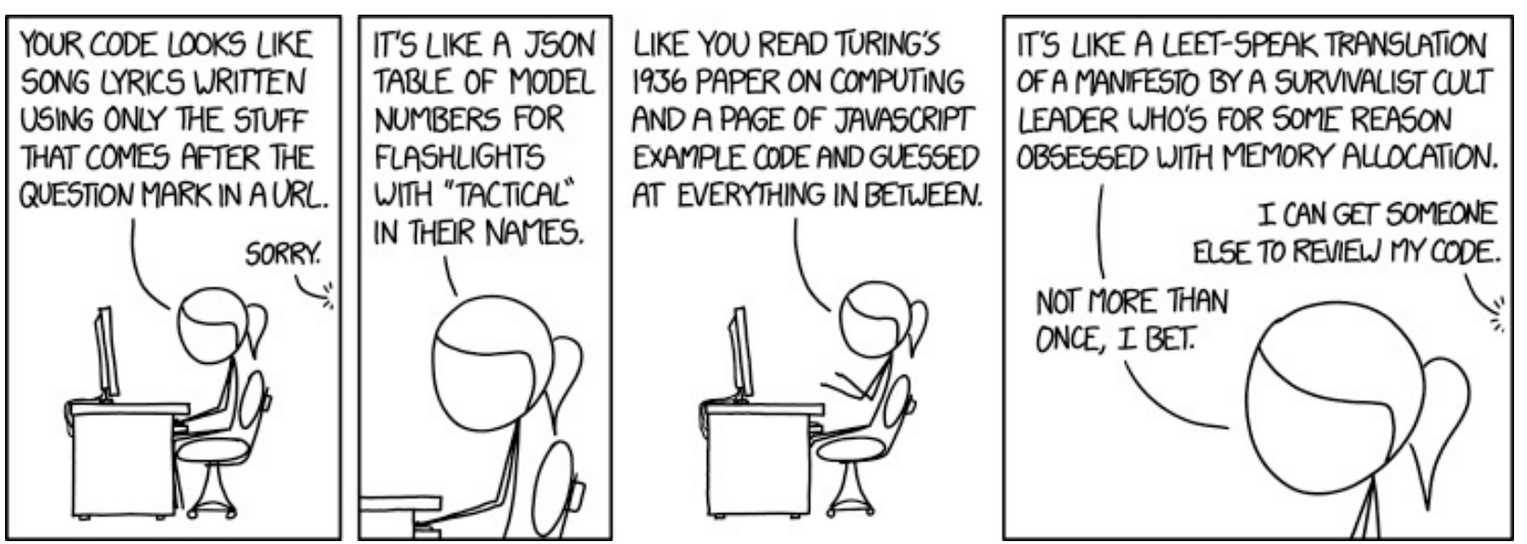


What is code review?

Let other people read your code (and vice versa) and provide suggestions or make changes.

Why code review?

→ Build a shared understanding of the code base / reach consensus on changes



What is code review?

Let other people read your code (and vice versa) and provide suggestions or make changes.

Why code review?

→ Build a shared understanding of the code base / reach consensus on changes

Benefits of code review:

- Knowledge transfer, establishing best practices
- Improving code clarity
- Identifying refactoring opportunities
- Sharing perspectives/generating new ideas

Sometimes refactoring is done during the code review

Methods of Code Review

When is code reviewed?

- Could be for every pull request, or just new/important features

Option 1. Reviewer looks at code without the original author present

- e.g., reviewing a pull request

Option 2. Reviewer looks at code alongside the original author

- Author provides context
- Reviewer can better understand author intent
- Author can better understand reviewer's comments
- If review includes refactoring, may involve **pair programming**

Code Review

There is no such thing as perfect code, only **better** code.

https://google.github.io/eng-practices/review/reviewer/standard.html

What to look for?

- Design
- Functionality
- Complexity
- Tests
- Naming
- Comments
- Style
- Consistency
- Documentation
- Every Line
- Context
- Good Things

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Design**:
  Do code interactions make sense?
  Do any changes integrate well?
  Is now a good time to add this functionality?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Functionality**:
  Does the code do what the developer intended?
  Is what the developer intended good for the users of this code?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Complexity**:
  Is the code more complex than it should be (check all levels: lines, functions, classes)?
  Can it be understood quickly?
  Will modifications risk introducing bugs?
  Is it too generic / over-engineered?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Tests**:
  Are the tests appropriate (unit, integration)?
  Are tests correct, sensible, and useful?
  Will tests actually fail if code is broken?
  Are tests separated appropriately?
  Are tests too complex?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Naming**:
  Are names descriptive yet concise?
  Is it clear what each item is or does?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Comments**:
  Are the comments clear and understandable?
  Are they necessary (why, not what)?
  Are they compensating for unclear code?
  Are any comments outdated?
  Don’t comment out sections of code! Just remove them.

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Style**:
  Does the code follow the appropriate style guide?
  For updates: Are functional changes mixed with style changes?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Consistency**:
  If not directly or specifically addressed in a style guide, is it at least consistent with existing code?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Documentation**:
  Are associated READMEs, docs included/updated?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Every Line**:
  Look at every line!
  Is it all understandable with a reasonable amount of time spent reading the code?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Context**:
  Does each piece of code make sense in the context of the system?
  For updates: Do changes overall improve the code rather than making it more complex, less tested, etc.?

https://google.github.io/eng-practices/review/reviewer/looking-for.html

What to look for?

- **Good Things**:
  Offer encouragement and appreciation for good practices!

https://google.github.io/eng-practices/review/reviewer/looking-for.html

Guidelines

SMART Code Review:

Be **S**pecific with defects

Have **M**easurable and **A**chievable suggested improvements

Give feedback that is **R**elevant to the type of review and the problem to be solved

Give feedback that is **T**ime bounded – set priorities and have specific actions to be taken to remove defects with a specified deadline

Implementing Effective Code Reviews, Giuliana Carullo

Guidelines

Specific
Measurable
Achievable
Relevant
Time bounded

“There are too many dependencies between Component 1 and Component 2. This is due to bad design. Change it.”

Guidelines

Specific
Measurable
Achievable
Relevant
Time bounded

“There are too many dependencies between Component 1 and Component 2. This is due to bad design. Change it.”

Example of better feedback:

“There are a lot of dependencies between Component 1 and Component 2, related to code smell **X**. The design could be improved by refactoring and applying design pattern **Y.** This is an important defect; consider fixing it by date **Z**.

Code Review for Humans

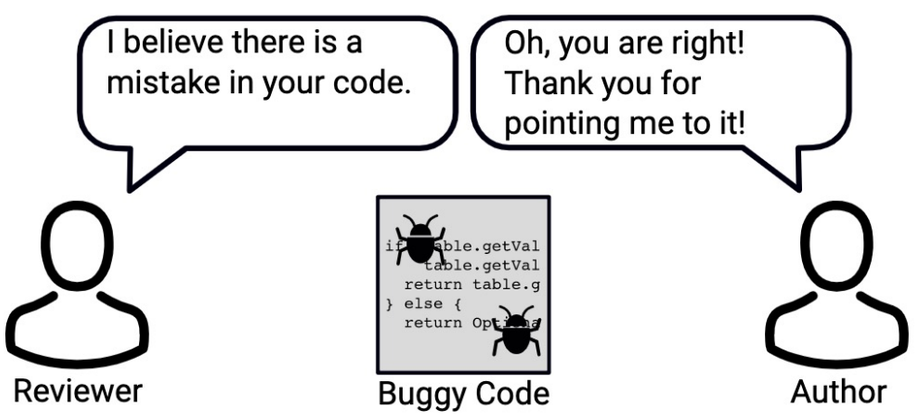
Code review guidelines for doing code reviews like a human.

https://phauer.com/2018/code-review-guidelines/

For the Author

Be Humble

- Have an open and humble mindset about feedback you will receive
- Everybody makes mistakes, everyone can improve
- We do our best to avoid mistakes with careful coding and testing
- But, accept that mistakes happen, and it is most important that they are noticed, acknowledged, and fixed



Making mistakes is accepted and admitting them is desired.

https://phauer.com/2018/code-review-guidelines/

For the Author

You are not your code

- Criticism of your code is not criticism of you!
- You are still a valuable team member even when you make mistakes (provided you acknowledge and fix them)
- We tend to place more value on code **we** have written – be aware of this bias and be willing to accept changes or removal of your code when necessary

You and the reviewer are on the same side

- You both want to improve the code, to make a better product

https://phauer.com/2018/code-review-guidelines/

For the Author

Seek out new perspectives

- Every developer has different background, assumptions, experience
- Identify implicit knowledge / assumptions, and make them explicit to make your code more understandable

Exchange best practices and experiences

- Code review is not just about the code
- Use the opportunity to exchange knowledge, establish/agree on best practices

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Use I-Statements

- Phrasing is important; frame your feedback as from your point of view
- You-statements can sound like absolute statements; at worst can seem like an attack on the author
- Avoid getting caught up in justifications/ego, focus on building a shared understanding to move forward on improving the code
- Not just applicable to code review!

Increase the acceptance of your feedback by using I-messages

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Talk about the code, not the coder

- Criticism on the code is harder to take personally, which will improve communication keep the focus on improving the code
- Also avoids the impulse to blame someone else for a mistake (collective code ownership)

> Wrong: "**You're requesting** the service multiple times, which is inefficient."

> Right: "**This code is requesting** the service multiple times, which is inefficient."

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Ask questions

- Code review is collaborative
- Be sincere – aim to understand the intention behind a design decision; don’t assume there was not a good reason
  - And if there was not a good reason, then you can decide together on a fix

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Ask questions

- Code review is collaborative
- Be sincere – aim to understand the intention behind a design decision; don't assume there was not a good reason
  - And if there was not a good reason, then you can decide together on a fix

> Right: "**This code is requesting** the service multiple times, which is inefficient."

Or: Why is this code requesting the service multiple times?

> Wrong: "This variable should have the name 'userId'."

> Right: "What do you think about the name 'userId' for this variable**?**"

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Structure your feedback: Observation – Impact – Request

- Observation: “This method has 100 lines”
- Impact: “This makes it difficult for me to grasp the logic of the method”
- Request: “How about extracting the low-level details into subroutines with descriptive names?”

https://phauer.com/2018/code-review-guidelines/

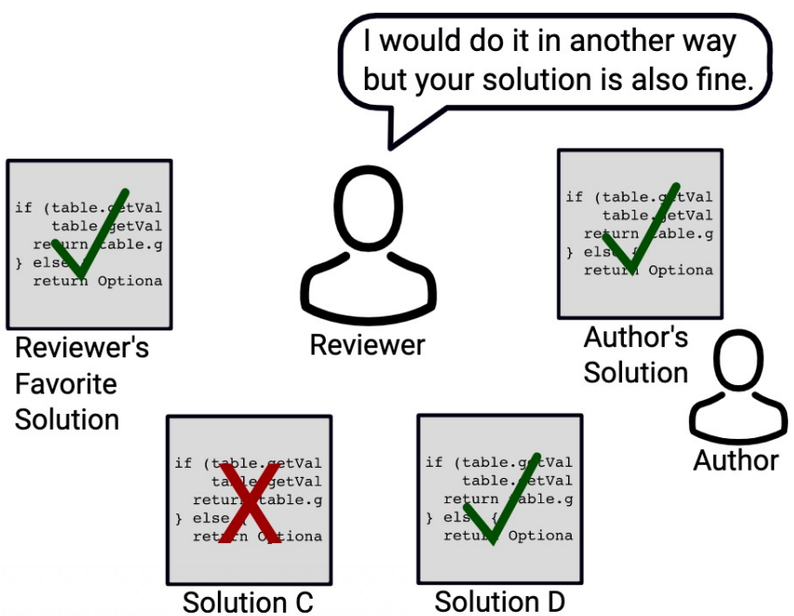
For the Reviewer

Accept that there are different solutions

- Maybe you have a favorite solution, but the author's may also be fine
- Distinguish between common best practices and your preferences
- Make compromises and be pragmatic

Don't criticize every line of code

- Don't exhaust the author - focus on changes that you think are most important and necessary
- If possible, take a break, iterate



Be aware of your personal taste, accept other solutions and make compromises

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Praise

- Don’t forget to comment on good code!
- Praise should be sincere, specific, concrete, and separate from criticism

https://phauer.com/2018/code-review-guidelines/

For the Reviewer

Before giving feedback, ask yourself:

- Is it true?
- Is it necessary?
- Is it kind?

https://phauer.com/2018/code-review-guidelines/

https://www.youtube.com/watch?v=Ea8EiIPZvh0

For the Reviewer

Before giving feedback, ask yourself:

- Is it true? Express your **opinion**, use I-statements
- Is it necessary? Focus on the most important changes, be **helpful**
- Is it kind?
  This does not mean no criticism! Don't be "fake nice," and don't be afraid to make suggestions.
  It is an important skill to provide useful criticism and feedback without shaming the author.

https://phauer.com/2018/code-review-guidelines/

https://www.youtube.com/watch?v=Ea8EiIPZvh0

Also applies to the author!

Ask yourself:

- Is it true?
- Is it necessary?
- Is it kind?

Reviewer: Given the dependencies, I'm concerned it will be difficult to modify this in the future.

Author: I think that's a trivial concern.

https://www.youtube.com/watch?v=Ea8EiIPZvh0

Also applies to the author!

Ask yourself:

- Is it true?
- Is it necessary?
- Is it kind?

Reviewer: Given the dependencies, I'm concerned it will be difficult to modify this in the future.

~~**Author**: I think that's a trivial concern.~~

Better: Thank you for raising that concern. I don't think it will be a problem because {reason}. What do you think?

https://www.youtube.com/watch?v=Ea8EiIPZvh0

Code Review Template

- Design, Functionality, Complexity
  - Strategy Pattern
- Tests
- Naming, Comments, Style, Consistency, Every Line, Good Things
  - Clean code table
- Specific, Measurable, Achievable, Relevant, Time-Bounded
  - Specific examples (Good Things) or specific suggested improvements
  - Time-bounded: Implied to be the code quality iteration deadline
- True, Necessary, Kind

Code Review Template

Guideline	**Example/Suggested Improvement**
Small	
Do One Thing	
One Level of Abstraction	
Use Descriptive Names	
Keep Number of Arguments Low	
Avoid Flag Arguments	
Have No Side Effects	
Command Query Separation	
Prefer Exceptions to Error Codes	
Don’t Repeat Yourself	
No arguments in method/class names	
Do the Same Thing, the Same Way	
Name Boolean Expressions	
Bail out Fast	