10 FAULTY BEHAVIORS OF CODE REVIEW

LEMi ORHAN ERGiN
co-founder @ craftbase

```php
function register()
{
if (!empty($_POST)) {
$msg = '';
if ($_POST['user_name']) {
if ($_POST['user_password_new']) {
if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
if (strlen($_POST['user_password_new']) > 5) {
if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
$user = read_user($_POST['user_name']);
if (!isset($user['user_name'])) {
if ($_POST['user_email']) {
if (strlen($_POST['user_email']) < 65) {
if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
create_user();
$_SESSION['msg'] = 'You are now registered so please login';
header('Location: ' . $_SERVER['PHP_SELF']);
exit();
} else $msg = 'You must provide a valid email address';
} else $msg = 'Email must be less than 64 characters';
} else $msg = 'Email cannot be empty';
} else $msg = 'Username already exists';
} else $msg = 'Username must be only a-z, A-Z, 0-9';
} else $msg = 'Username must be between 2 and 64 characters';
} else $msg = 'Password must be at least 6 characters';
} else $msg = 'Passwords do not match';
} else $msg = 'Empty Password';
} else $msg = 'Empty Username';
$_SESSION['msg'] = $msg;
}
return register_form();
}
```

```php
function register()
{
if (!empty($_POST)) {
$msg = '';
if ($_POST['user_name']) {
if ($_POST['user_password_new']) {
if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
if (strlen($_POST['user_password_new']) > 5) {
if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
$user = read_user($_POST['user_name']);
if (!isset($user['user_name'])) {
if ($_POST['user_email']) {
if (strlen($_POST['user_email']) < 65) {
if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
create_user();
$_SESSION['msg'] = 'You are now registered so please login';
header('Location: ' . $_SERVER['PHP_SELF']);
exit();
} else $msg = 'You must provide a valid email address';
} else $msg = 'Email must be less than 64 characters';
} else $msg = 'Email cannot be empty';
} else $msg = 'Username already exists';
} else $msg = 'Username must be only a-z, A-Z, 0-9';
} else $msg = 'Username must be between 2 and 64 characters';
} else $msg = 'Password must be at least 6 characters';
} else $msg = 'Passwords do not match';
} else $msg = 'Empty Password';
} else $msg = 'Empty Username';
$_SESSION['msg'] = $msg;
}
return register_form();
}
```

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

LEMi ORHAN ERGiN

co-founder, Craftbase

founder, Software Craftsmanship Turkey

alumni, Sony, eBay, ACM, iyzico

programming, since 2001 with love ❤️

based, Istanbul, Turkey

speakerdeck.com/lemiorhan

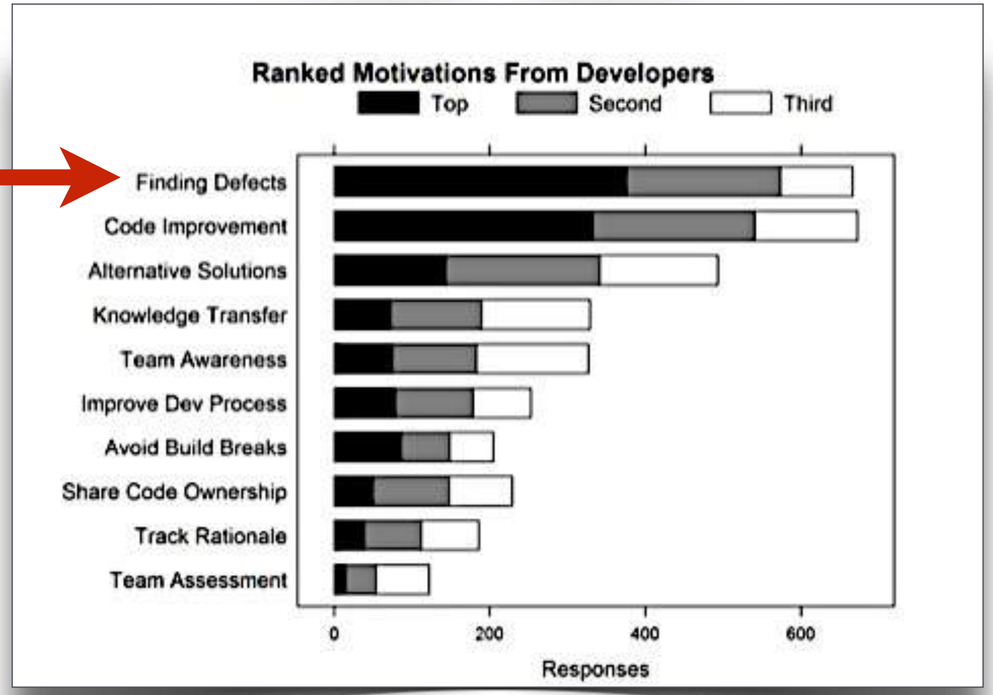
@lemiorhan

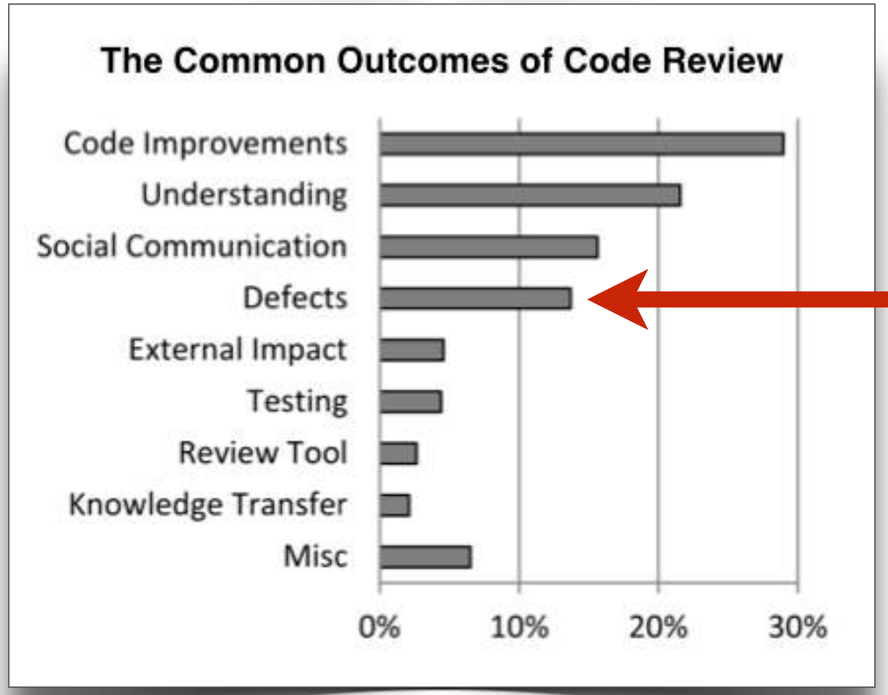

WARNING
NEVER DEPLOY TO PRODUCTION
UNLESS THE CODE IS REVIEWED
& APPROVED BY SOMEONE ELSE

Benefits of Code Review

Benefits of Code Review

Theory **Practice**





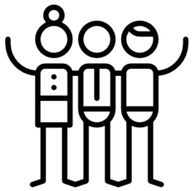
Quotes and figures are from Alberto Bacchelli and Christian Bird,
“Expectations, Outcomes, and Challenges of Modern Code Review”, May 2013,
Proceedings of the International Conference on Software Engineering.

https://ieeexplore.ieee.org/document/6606617

Benefits of Code Review

1

Common Ownership



The whole team is fully responsible to create a well-crafted software. It is about **sharing responsibility** of failures and successes and owning results together.

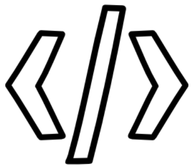
2

Knowledge Sharing

We learn how others design and implement software. We see new ways of coding, **alternative solutions** and know-how about the domain.

3

Better Code



Code becomes more optimized, **cleaner and better** in both security and performance. Since it is reviewed by another brain, suggestions improve code.

4

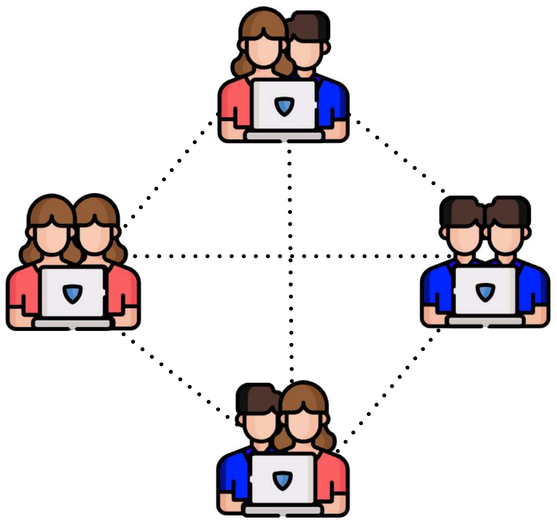
Finding Defects

Much far before the code goes to production, we can **notice bugs** just by reading the code. That is cheap, much cheaper than expected.

Efficient Code Review Techniques

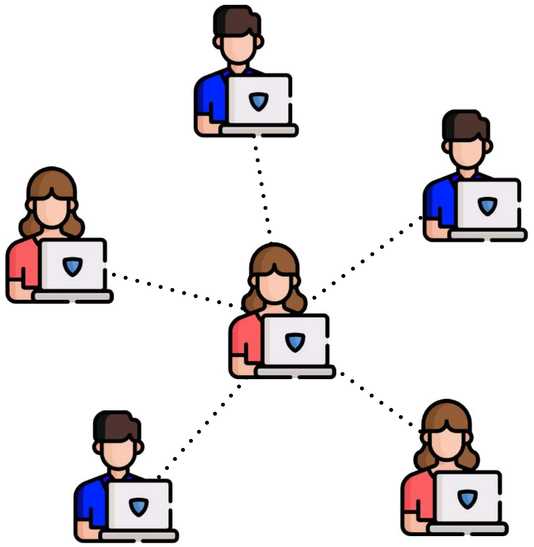
Collaborative Teams

knowledge is shared
no task assignments to individuals
everyone is responsible for quality



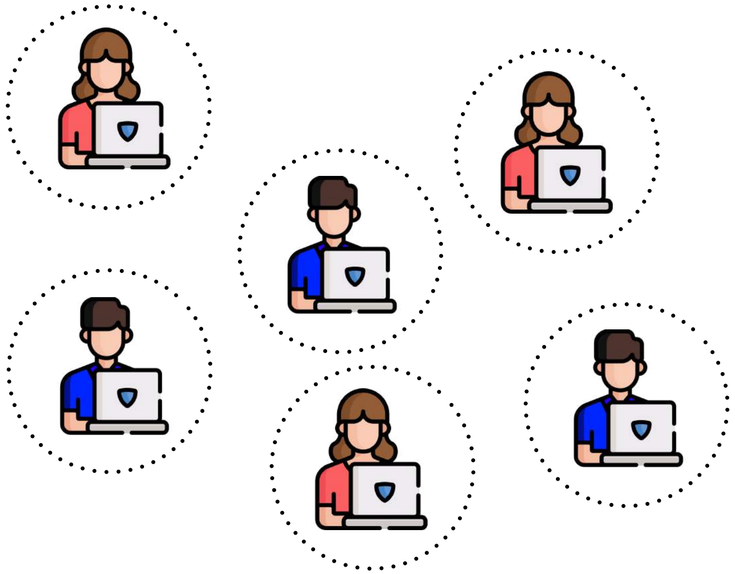
Gate Keepers' Teams

someone owns the project
tech lead is the main gate for quality
few people knows everything



Group of Silos

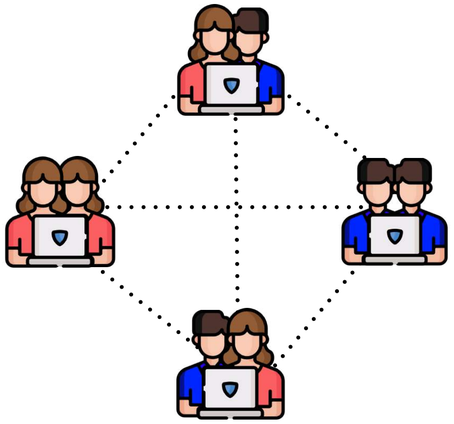
each individual works on its own
everyone knows its own domain
everyone is responsible for its domain



Efficient Code Review Techniques

Collaborative Teams

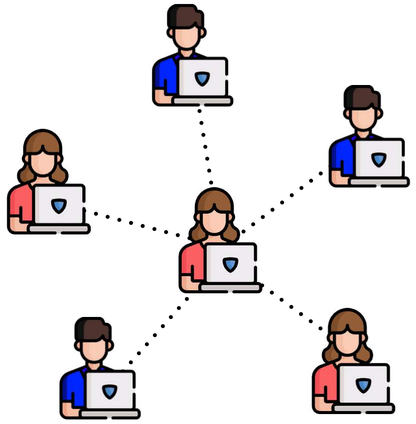
knowledge is shared
no task assignments to individuals
everyone is responsible for quality



mob programming
pair programming
code review sessions

Gate Keepers' Teams

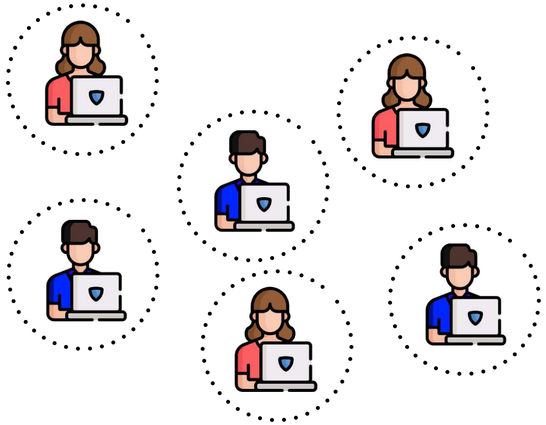
someones owns the project
tech lead is the main gate for quality
few people knows everything



pull requests

Group of Silos

each individual works on its own
everyone knows its own domain
everyone is responsible for its domain

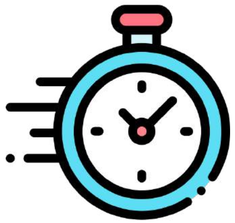


code review can never be as efficient as expected

Code reviews should be lightweight

Fast

small change sets, takes time indeed, but not very time consuming for reviewers



Early

feedback is much valuable when it is provided in early stages of development

Clear

reviewers can understand the code and deliver to-the-point feedback

Prioritized

not all change requests are urgent

4 STEPS OF CODE REVIEW

AND 10 FAULTY BEHAVIORS

Prerequisites

Opening

Review

Closing

no agreements in the team

Discussing again and again on the same topics
Reviewing broken code
Writing comments does not change anything
Endless discussions, no conclusions

make agreements in the team

Use tools to make you follow guidelines

Use tools like CI, style checker and static code analysis

Have standards

Agree on standards on conventions and process, and improve it continuously

Share responsibilities

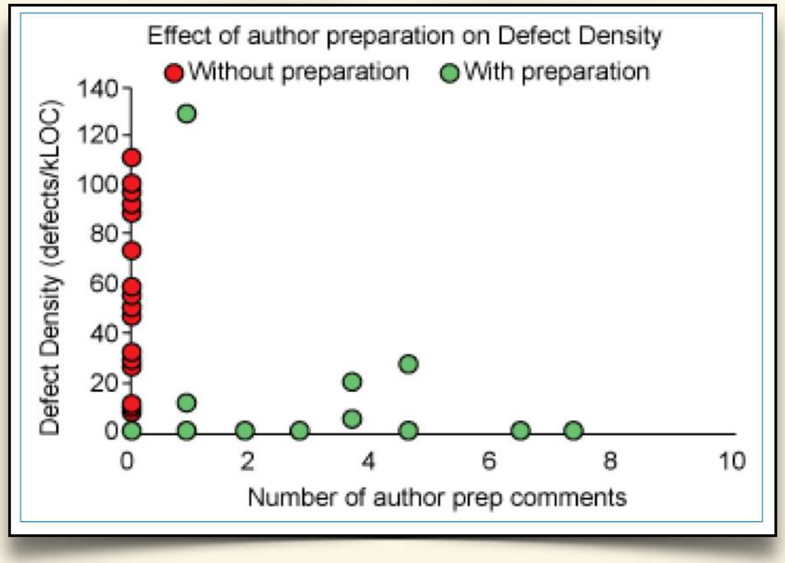
Reviewers have the same responsibility for the bugs as the authors

Review by yourself again

Statistics say that author review beforehand decreases the number of defects dramatically and leads to cleaner code and safer review process

before asking for a review
the authors has to review by themselves

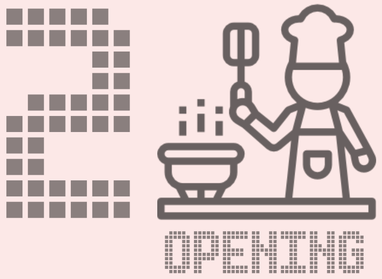
"Each person typically makes the same
15-20 mistakes over and over again."



from the book "Best Kept Secrets of Peer Code Review"
by Jason Cohen, Steven Teleki and Eric Brown

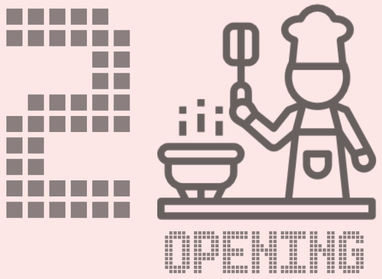
https://smartbear.com/resources/ebooks/best-kept-secrets-of-code-review

- ☑ check if code is working as expected
- ☑ format code
- ☑ remove dead code
- ☑ run tests and fix failing ones
- ☑ add new tests for missing cases
- ☑ reorganize code
- ☑ add comments for required places
- ☑ remove warnings and errors
- ☑ check usage of exceptions and logs
- ☑ remove hardcoded values
- ☑ add todos to required places
- ☑ refactor once more



ambiguous content to review

A big whale of code is waiting in the review
No information at commit messages or PR descriptions
Author comments required for clarification
Old comments are gone. Where are my comments?



better PRs lead better reviews

Keep change set small

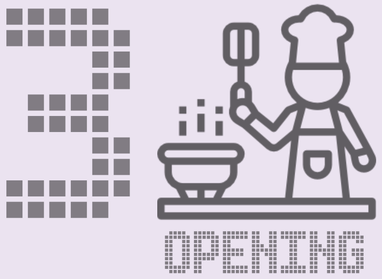
Big change sets distracts focus. Keep it small to complete in max 30 mins.

Better messages

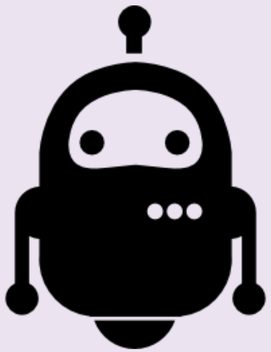
Commit messages and PR descriptions should contain as much information as it could be, such as design decisions and test scenarios.

Prefer face to face

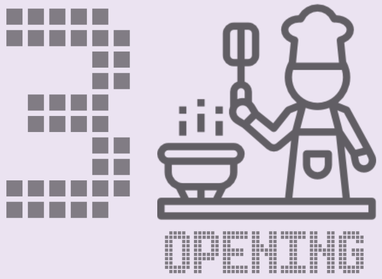
You may still have content to review ambiguous to others. Ask for a code review session and discuss the content face to face.



wrong reviewer selection



Waiting feedback from too many people
Always selecting the same people as the reviewer
Randomly selecting reviewers and receive nothing
Who the hell should I open my review to?



make the review load bearable

Select max 2-3 reviewers

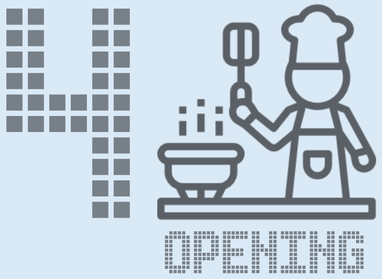
If you open to the whole team, expect to get feedback from max 2 people.

Select from old authors

By looking at the git history, you can find who touched the code you worked. Then you can open reviews to them.

Open reviews not to the same people

Remember that each review takes approx. 30 mins average. It means it takes time. So show respect to people’s workload.

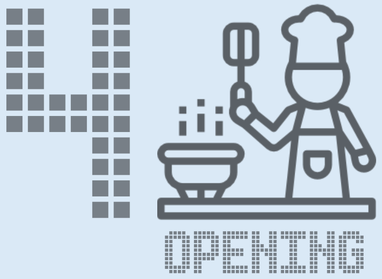


too late for asking for feedback

Opening to code review after merging back to source
Ask for review after 1 month work
Deadline has arrived but you haven’t opened PR yet
Code review branches in everywhere in git

Gather feedback before it is merged to master

Asking for feedback after deploying to production is meaningless



Do not wait to open PR to ask for feedback

Implement in pairs or as mob

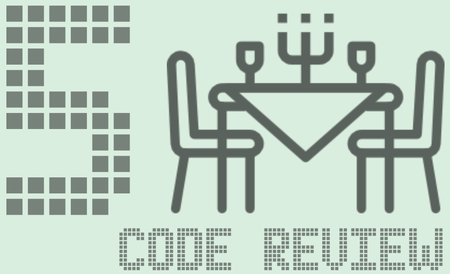
Prefer implementing not alone if you develop somewhere risky and requires immediate feedback.

Call people for review sessions

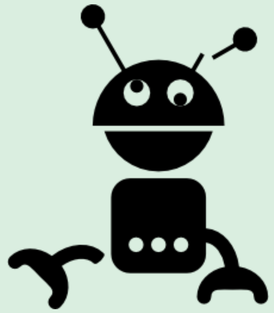
You do not need to complete development of the feature and push to upstream. Come together with colleagues and review.

Review regularly

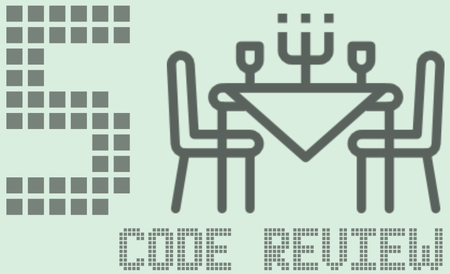
Everyday you should look at the commits delivered or PRs opened. Before or after daily is the best time for it.



cannot understand
what the change set is doing



Having no tests. Cannot understand what code really does.
Not knowing where to look while reviewing.
Tons of syntactical feedback, but nothing from business logic.
I don’t know the code better than the author.



make the review process more structured

Have a checklist of topics to review

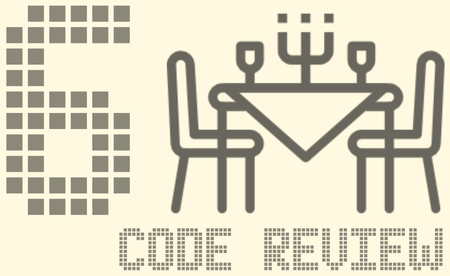
It is important to look at where to look in correct mindset. Prepare a checklist of topics and review the code with each.

Write automated tests

Minds cannot compile or run the code. Without tests, code is simply a puzzle to solve with our imagination.

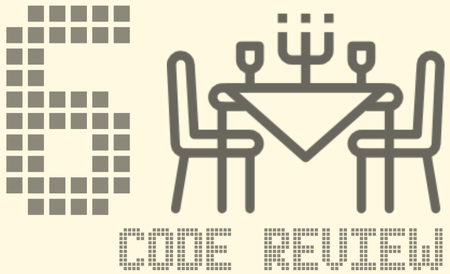
Write description about the business logic

It would not be possible to have reviewers knowledgable about the business as you are. Give information about the business logic at PR descriptions.



this is my code, my expertise

I am the only one who works on X domain.
Not touching someone else's code.
Improvement suggestions are never applied.
I know the code better than anyone. How dare you are?



this is our code, our expertise

Work with your colleagues

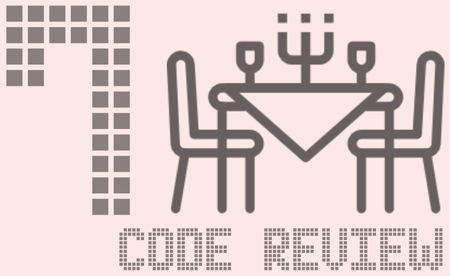
There is nothing such as “your code”. You are a silo right now and you are in danger. Share knowledge with your colleagues.

Discuss your feelings with your team

If you feel no one can give valuable feedback due to lack of knowledge, share your feelings with your team and find a way to improve knowledge and responsibility sharing.

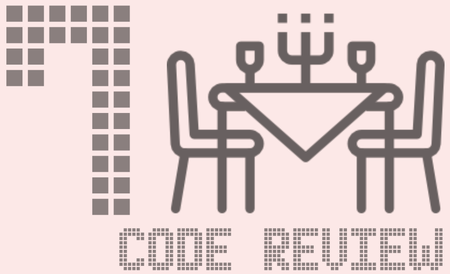
Apply criticals now, add `//todo` for others

If you get an improvement suggestion during code review, apply them now. Of course nice to haves could wait a little bit longer:)



trying to prove
others are stupid

I know better than others. They code the legacy.
“This code is totally wrong. Haven’t you read the book?”
Women is not biologically suitable for programming.
Others try to destroy my reputation.



leave your ego, be kind and do not make it personal

Prime Directive: Egoless Programming

Accept that you are not your code, you will make mistakes and someone else will know more than you.

Be kind

Be gentle, keep tone positive. Be thankful. Point out good stuff.
No need for jokes, the words “always”, “never”, “mine”, “yours”.

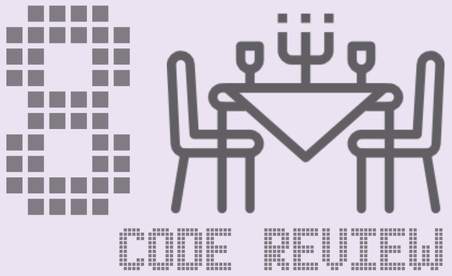
Respect the legacy code

Keep in mind that many bad decisions were taken due to limitations of the past. So respect them and try to understand conditions of the past and reasons of the decisions.

BE CAREFUL WITH YOUR WORDS

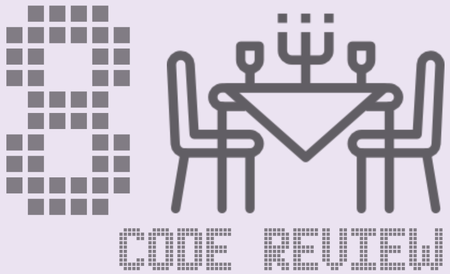
Jokes
Gender based comments
Direct "you"
Words "always", "never", "hate"
"mine", "yours"

"I didn't understand. Can you clarify?"
"We usually do it in this way"
"What do you think about naming this XYZ?"
"I'm not sure - let's look at it again."
"Do you have special purpose for your decision?"
"Good point, thank you"
"I feel it is not showstopper but ..."



not able to convince people
about your review comments

Writing lots of comments and getting no response.
I am sure I am right but I cannot convince my solution is better
We discuss but never conclude on my proposals
Always hearing “this is how we do in this team”



sell your ideas like people sell products

Support your idea with arguments

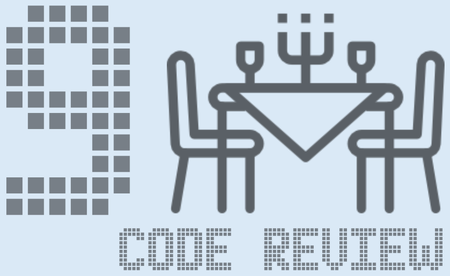
Collect code snippets, links, tutorials, and any resource that can give detailed explanation about your comment.

Make your idea feel like a better alternative

Never judge the decisions. Instead show your idea in all aspects and make people feel like a better alternative solution.

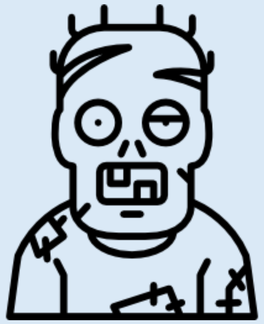
Prefer discussions over comments

Still face to face communication is the best way to find the best solution and improve.

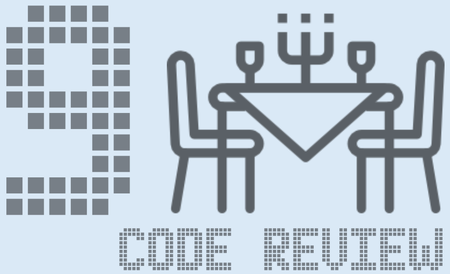


people play dead

when I create a PR and ask for feedback



No time for code review due to work load
PRs stay open unmerged for too long
Asking a question but getting no response
Everyone asks feedback but no-one gives one



no-one can play dead while communicating face to face

Shout out loud if you need urgent feedback

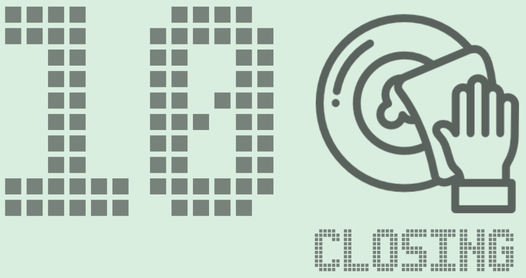
If you need help and instant feedback from your teammates, simply go and ask for it.

Make code review a ritual

In order to make code review a habit, make it a regular ritual, like “the review hour” on the last hour of every work day.

Go and review code together

If you get no response from the author, do not wait. Just go to her/him and review code together.

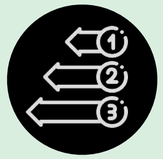


premature closing

Merging the PR without reviewing due to deadline pressure
Merging after getting one dummy acceptance
Merging even though the discussions are going on
Managers order to merge the unreviewed code

10

never merge a PR with ongoing discussions



Be aware of priorities

Not all change requests are urgent or showstopper. Some PRs can be merged with accepting the debt to eliminate later.

Never allow long discussions

If ongoing discussions seems to last long, meet together and discuss face to face.

Ask for immediate help

If your code stayed unreviewed for a while and the deadline comes, ask from someone else for an immediate feedback.

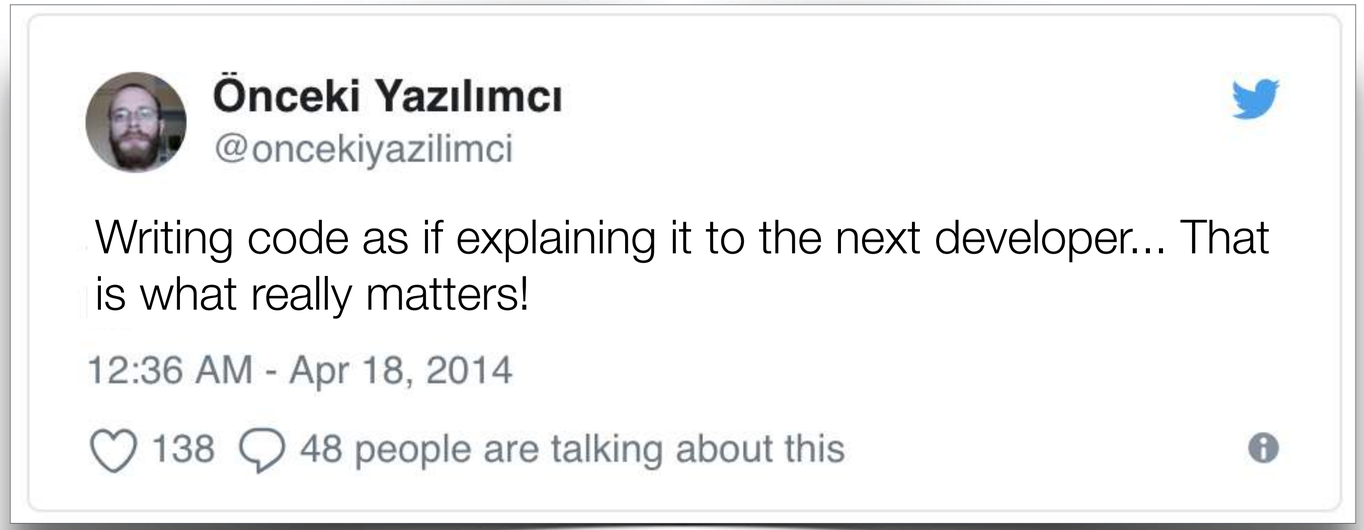
10 FAULTY BEHAVIORS of CODE REVIEW

1. no agreements in the team
2. ambiguous content
3. wrong reviewer selection
4. too late for asking for feedback
5. cannot understand what the change set is doing
6. this is my code, my expertise
7. trying to prove others are stupid
8. not able to convince people about our review comments
9. people play dead when I create a PR and ask for feedback
10. merging PRs with ongoing discussions

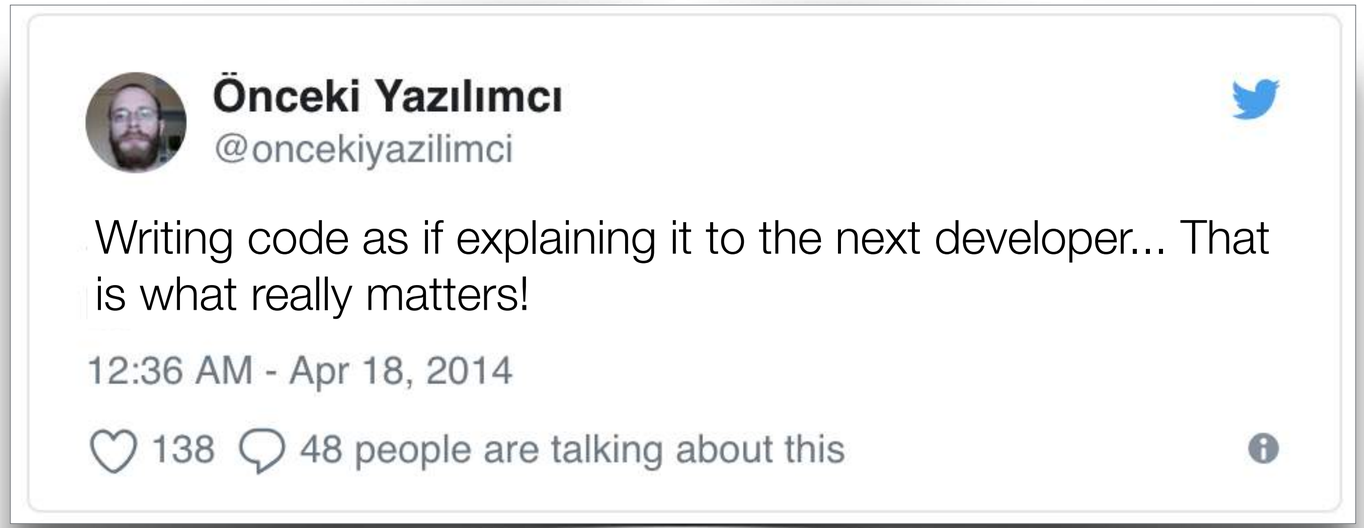
good programmers are the ones
who care the code they produce

do retrospectives

talk about code review process
as soon as you feel something goes wrong



Original text in Turkish: Sonraki yazılımcıya anlatır gibi kod yazmak... Bütün mesele bu!
https://twitter.com/oncekiyazilimci/status/456909019685605376



Original text in Turkish: Sonraki yazılımcıya anlatır gibi kod yazmak... Bütün mesele bu!
https://twitter.com/oncekiyazilimci/status/456909019685605376