What is code review?

Find problems

~~Code review does not have to be on a software. Like screen sharing etc. or pair programming counts too.~~ If we want to do fully async and/or we have to audit the code (large organisations).

“Human reviewers should be doing what cannot be automated”

The goal is the accept the review. Make sure to code is good enough to go to production

Why review code?

--------------------------------------------------------

Multiple team members understand the changes

Check the quality / compliance to standards of the code

Catch security problems/ bugs early

~~Get early feedback on design approaches2~~

2 <https://www.youtube.com/watch?v=EjwD7Pi7J_0>

-----------------------------------------------------------------------

Ensure code meets standarts (what are the standards?)

Find bugs (most of them can be automated tho)

Share knowlage

Understandability of the code

When a code review should done?

Focuses are different

Gateway Reviews -> when code is complete -> more traditional -> someone says good to go or not -> if not fix these parts

* ~~Hard to fix if there is a fundamental design change~~

~~Knowledge sharing~~

* ~~When code in production~~
* ~~Focus: Readability, having everyone on the same page~~

~~Early Design Feedback~~

* ~~Before code~~

Quickly reviewing is important. 2

Author probably waiting for the review results 2

And if too much time passes, it might get hard to remember context of the changes. 2

Who should do the code review?

Reviewer choosen according to team guidelines. (What kind of guidelines) 2

Gataway or quality check -> one person and/or speclialist should check 2

Share the details of the implementation with the team -> anyone or everyone 2

------------------------------------------------------------------------------------------------------

Senior developer

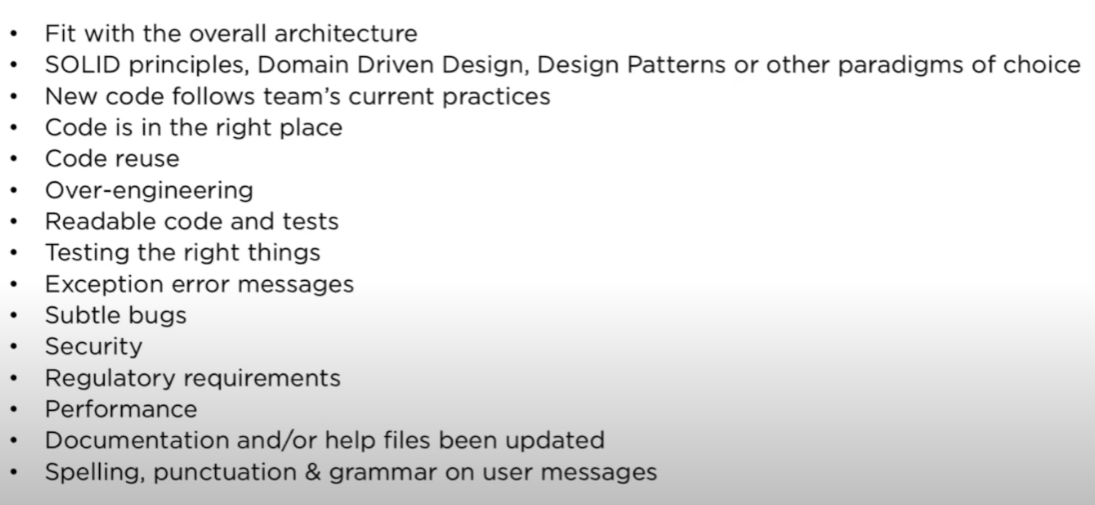
Junior developer -> share knowledge

What to look while reviewing?

Code review best practices video - webinar

What to look for in a code review -> book

Differ from the work flow. It depends on a lot of things.



Constraints are important. (eg Security or performance or latency can change application to application)

What are tools and common programs used for code reviewing?

<https://blog.jetbrains.com/upsource/tag/code-review-workflows/>

Upsource –Jetbrains

Crucible – Atlassian

Github

Azure Devops

Where can we introduce automation in code review?

<https://blog.jetbrains.com/upsource/tag/code-review-workflows/>

formatting should be automated. Code reviewer should not look for formatting problems. Such as curly brace places or method names etc.

who will be assigned to review? Can be automated 2

creation of reviews can be automated. Meaning which parts will be reviewed.2

Integration of CR with a BT tool, reviewers can easily look issue and explanations and discussions that this commit aims to achieve. 2 (issue tracker integration)

If the commit passes the automated tests or not. If not passed there is no need for review.

What artefact can code review can and should produce and how to organise them?

Comments and feedbacks kept in program (upsource)

What not to do?

Nit picking -> (eg. style)

Design changes when code works (implementation finished). If it is a gateway review. Other workflows might prevent it.

Too big reviews

Ping pong reviews

What are the results of the code review?

Accepted

~~Failed~~ Raise Consern

Conserns should be clear

Questions needs to be answered

Change needed -> change done (extra commit added to review) -> processing

More ?

Closing the review

What is the ideal review size?

Reviews should be small.

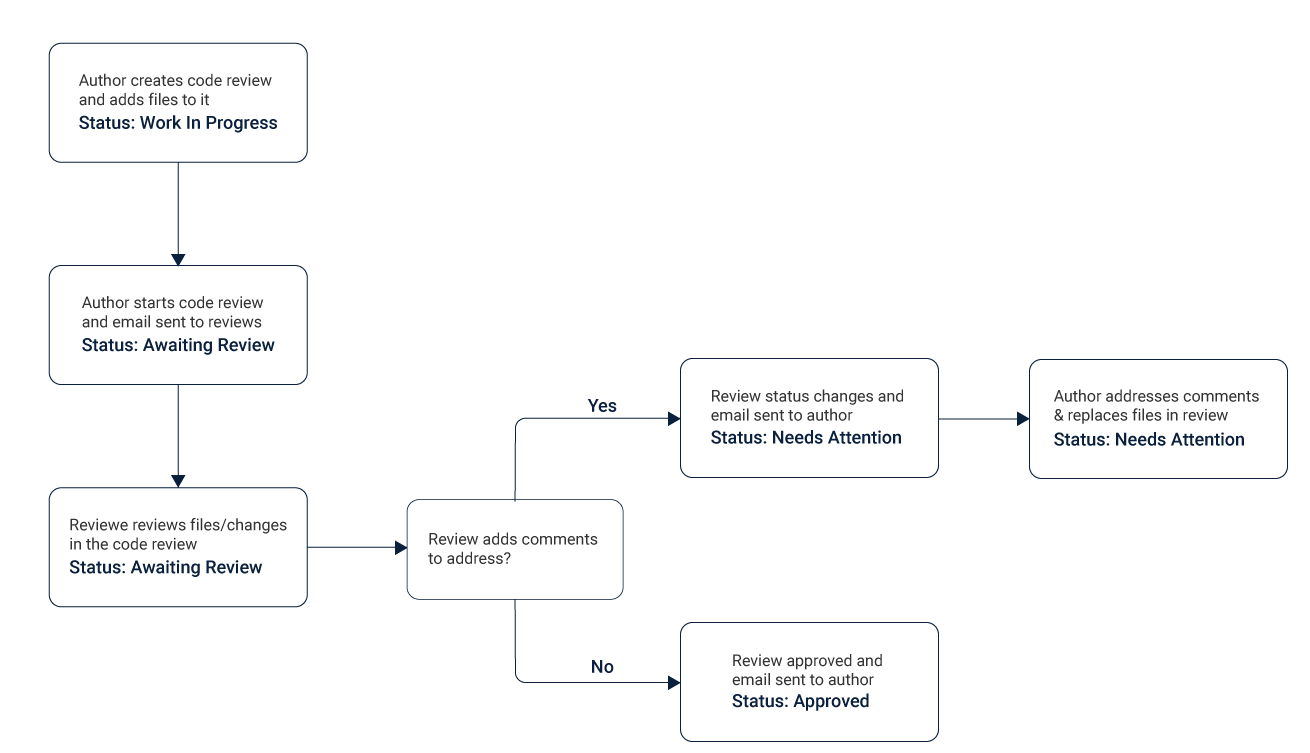
https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

When is the review complete?

Ideal review

* Should respond quickly.
* A review should be small.
* Review should be done when something cannot be automated.
* If done at the end, it should not change the design. In the end code should be more like “is this doing what is suppose to do” etc. Author should not write the code all over again. This is a waste of time.
* Ideally authors should annotate the code before review [smart bear]

https://medium.com/cuelogic-technologies/code-review-process-best-practices-3eeecab26ded



Examples Of Lightweight Techniques

~~Over-the-shoulder~~

~~One developer looks over the author’s shoulder as the latter runs through the code and suggests changes to be made.~~

~~This method is more appropriate when dealing with short snippets, and extreme accuracy is required regarding the results.~~

~~Email pass-around~~

~~As the name suggests, the author or the SCM system creates a hierarchical system and emails code to reviewers.~~

~~Pair Programming~~

~~Two authors develop the source code together at the same workstation and reverse engineer the results with the assumption that they would arrive at the same base kernels.~~

Tool-assisted

Obvious to the name, this means that authors and reviewers use specialised tools designed for peer code review.

https://devblogs.microsoft.com/appcenter/how-the-visual-studio-mobile-center-team-does-code-review/

Emoji code

The Emoji Code helps us separate well-meant suggestions, simple questions, and must-have requests that require code changes. It also adds an additional human component to the conversation, so we don’t forget there’s a human on the other side of the screen.

http://guides.beanstalkapp.com/code-review/guide-to-code-review.html

**Overview of the workflow**

* Developers use branches to implement features and bug fixes.
* Once branches are ready for testing, developers request code reviews.
* Other members of your team review code from the branches.
* A list of issues is compiled for each review.
* Developers commit additional changes to the branches to fix discovered issues.
* Code review of the branch gets approved.
* Branch is merged into master and shipped to production.

https://www.codegrip.tech/productivity/how-microsoft-does-its-code-review/

**Microsoft does it’s code review**

Preparing Code:

the developer then prepares a document for a reviewer. It explains the reason and meaning behind the changes done

Selecting Reviewer:

team leader or senior devs

Getting Feedback:

**Emoji Code**

**Improving code**

The aim of this step is to only satisfy the changes asked and not to add anything more. Once the developer feels that the changes are sufficient, they send the new code for reviewing which is also called revision. Either the revision will get approved by getting enough +1s and it then can be committed or the reviewer will send more changes repeating steps three and four.

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

Google’s CR guidelines

## How Fast Should Code Reviews Be?

**One business day is the maximum time it should take to respond** to a code review request

There is one time where the consideration of personal velocity trumps team velocity. **If you are in the middle of a focused task, such as writing code, don’t interrupt yourself to do a code review.**

## What Is An Emergency?

An emergency CL would be a **small** change that: allows a major launch to continue instead of rolling back, fixes a bug significantly affecting users in production, handles a pressing legal issue, closes a major security hole, etc.

In emergencies we really do care about the speed of the entire code review process, not just the [speed of response](https://google.github.io/eng-practices/review/reviewer/speed.html). In this case only, the reviewer should care more about the speed of the review and the correctness of the code (does it actually resolve the emergency?) than anything else. Also (perhaps obviously) such reviews should take priority over all other code reviews, when they come up.

Eray Hoca: Focus on the pull request code review

Most basic:

1. Ticket created
2. Assigned
3. Fixed and sent for review
4. Reviewer declines or accepts
5. If declined author fixes
6. Reviewer looks again
7. If accepted merge pull request.

Who reviews?

1. Senior developer? (Code reviewers)
2. (Only watch) junior developers (Code Watchers)

How long it should take?

“If you are not in the middle of a focused task, **you should do a code review shortly after it comes in.**

**One business day is the maximum time it should take to respond** to a code review request (i.e., first thing the next morning).” [Google src]

Not more then 400 LOC. [Smart bear]

Approx. 200 LOC [Google - [google.github.io/eng-practices/review/developer/small-cls.html](https://google.github.io/eng-practices/review/developer/small-cls.html)]

Daily how much ?

EXCEPTION: Emergencies.

Before review?

Build should be checked with CI/CD pipeline. Build should pass before reviewing. Reviewer should not test the code. Test should be automated. Maybe static code analysers should be in this stage.

[https://www.perforce.com/blog/qac/9-best-practices-for-code-review#:~:text=3.-,Don't%20Review%20Code%20For%20Longer%20Than%2060%20Minutes,(and%20in%20short%20sessions). Code analyzers]

Reviewer assignment might be automated. 2

Formatting should be automated. Code reviewer should not look for formatting problems. Such as curly brace places or method names etc.

Creation of reviews can be automated. Meaning which parts will be reviewed.2

Integration of CR with a BT tool, reviewers can easily look issue and explanations and discussions that this commit aims to achieve. 2 (issue tracker integration)