**My first code review**

My job is to find problem

**Anti-pattern**

Nit picking

Design changes when the code work

Inconsistent feed back

The ghost reviewer

Ping pong review

**Developers hate code review**

**Code reviews are a massive waste of time** زمان زیادی رو هدر می ده

# Why?

* **Ensure code meets standards**
* **Find bugs**
* **Share knowledge**
* **Check code is understandable**
* **Ensure code does what it’s supposed to** اطمینان حاصل کنید کد حتما وظیفه اصلی اش را انجام می دهد
* **Collaborate on design** همکاری در طراحی
* **Evolve application code**کد برنامه را تکامل دهید

# When?

**When do you review?**

During implementation?

When it’s ready to merge?

After it’s been merged?

**When is the review complete?**

When everyone agrees?

When a gate keeper agrees?

When all comments are addressed?

# Who?

Who reviews the code?

Who signs it off?

# Where?

Pairing

Showing code to a colleague at a computer نشان دادن کد به یک همکار در کامپیوتر

Mob reviewing in a conference room

Remote screen sharing

In the IDE, checking out a commit or branch

Using code review software

# What to look for

* How does the new code fit with the overall architecture?
* Does the code follow [SOLID principles](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)), [Domain Driven Design](https://en.wikipedia.org/wiki/Domain-driven_design) and/or other design paradigms the team favours?
* What [design patterns](https://en.wikipedia.org/wiki/Software_design_pattern) are used in the new code? Are these appropriate?
* If the codebase has a mix of standards or design styles, does this new code follow the current practices? Is the code migrating in the correct direction, or does it follow the example of older code that is due to be phased out?
* Is the code in the right place? For example, if the code is related to Orders, is it in the Order Service?
* Could the new code have reused something in the existing code? Does the new code provide something we can reuse in the existing code? Does the new code introduce duplication? If so, should it be refactored to a more reusable pattern, or is this acceptable at this stage?
* Is the code over-engineered? Does it build for reusability that isn’t required now? How does the team balance considerations of reusability with [YAGNI](https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it)?

#### Readability & Maintainability

* Do the names (of fields, variables, parameters, methods and classes) actually reflect the thing they represent?
* Can I understand what the code does by reading it?
* Can I understand what the tests do?
* Do the tests cover a good subset of cases? Do they cover happy paths and exceptional cases? Are there cases that haven’t been considered?
* Are the exception error messages understandable?
* Are confusing sections of code either documented, commented, or covered by understandable tests (according to team preference)?

#### Functionality

* Does the code actually do what it was supposed to do? If there are automated tests to ensure correctness of the code, do the tests really test the code meets the agreed requirements?
* Does the code look like it contains subtle bugs, like using the wrong variable for a check, or accidentally using an and instead of an or?

#### Have you thought about…?

* Are there potential security problems with the code?
* Are there regulatory requirements that need to be met?
* For areas that are not covered with automated performance tests, does the new code introduce avoidable performance issues, like unnecessary calls to a database or remote service?
* Does the author need to create public documentation, or change existing help files?
* Have user-facing messages been checked for correctness?
* Are there obvious errors that will stop this working in production? Is the code going to accidentally point at the test database, or is there a hardcoded stub that should be swapped out for a real service?

[**https://blog.jetbrains.com/upsource/tag/what-to-look-for/**](https://blog.jetbrains.com/upsource/tag/what-to-look-for/)

**human reviewer should be doing what cannot be automated**

**understand the constraints** درک محدودیت ها

**why: knowledge sharing:**

purpose isn’t to reject the code

focus is on how easy it is to understand the code

When: At the end

Too late for design

Should have specific checks

# How?

Automate everything you can

Submitting for review

Reviews should be small

Annotate your code

Reviewing

Should be clear who is reviewing

Respond in a timely fashion

Checklist of what to look for

**Comments**

**Bear in mind *why, when* and *what***

**Be constructive سازنده** باش

**Be specific**

**Accept or reject or raise concern**

**Resolving**

**The goal is to accept the review**

**Should be clear who signs it off and when**

# Have clear objectives

**Clarity from understanding**

**Why, where, who, what, when, how**

**The goal is to sheep the code. NOT to prove how clever you are**