# Reviewer Guidance

Since parts of reviews can be automated via linters and such, human reviewers can focus on architectural and functional correctness. Human reviewers should focus on:

- The correctness of the business logic embodied in the code.
- The correctness of any new or changed tests.
- The "readability" and maintainability of the overall design decisions reflected in the code.
- The checklist of common errors that the team maintains for each programming language.

Code reviews should use the below guidance and checklists to ensure positive and effective code reviews.

## General Guidance

### Understand the Code You are Reviewing

- Read every line changed.
- If we have a stakeholder review, it’s not necessary to run the PR unless it aids your understanding of the code.
- AzDO orders the files for you, but you should read the code in some logical sequence to aid understanding.
- If you don’t fully understand a change in a file because you don’t have context, click to view the whole file and read through the surrounding code or checkout the changes and view them in IDE.
- Ask the author to clarify.

### Take Your Time and Keep Focus on Scope

You shouldn't review code hastily but neither take too long in one sitting. If you have many pull requests (PRs) to review or if the complexity of code is demanding, the recommendation is to take a break between the reviews to recover and focus on the ones you are most experienced with.

Always remember that a goal of a code review is to verify that the goals of the corresponding task have been achieved. If you have concerns about the related, adjacent code that isn't in the scope of the PR, address those as separate tasks (e.g., bugs, technical debt). Don't block the current PR due to issues that are out of scope.

## Foster a Positive Code Review Culture

Code reviews play a critical role in product quality and it should not represent an arena for long discussions or even worse a battle of egos. What matters is a bug caught, not who made it, not who found it, not who fixed it. The only thing that matters is having the best possible product.

## Be Considerate

- Be positive – encouraging, appreciation for good practices.
- Prefix a “point of polish” with “Nit:”.
- Avoid language that points fingers like “you” but rather use “we” or “this line” -- code reviews are not personal and language matters.
- Prefer asking questions above making statements. There might be a good reason for the author to do something.
- If you make a direct comment, explain why the code needs to be changed, preferably with an example.
- Talking about changes, you can suggest changes to a PR by using the suggestion feature (available in GitHub and Azure DevOps) or by creating a PR to the author branch.
- If a few back-and-forth comments don't resolve a disagreement, have a quick talk with each other (in-person or call) or create a group discussion this can lead to an array of improvements for upcoming PRs. Don't forget to update the PR with what you agreed on and why.

## First Design Pass

### Pull Request Overview

- Does the PR description make sense?
- Do all the changes logically fit in this PR, or are there unrelated changes?
- If necessary, are the changes made reflected in updates to the README or other docs? Especially if the changes affect how the user builds code.

### User Facing Changes

- If the code involves a user-facing change, is there a GIF/photo that explains the functionality? If not, it might be key to validate the PR to ensure the change does what is expected.
- Ensure UI changes look good without unexpected behavior.

### Design

- Do the interactions of the various pieces of code in the PR make sense?
- Does the code recognize and incorporate architectures and coding patterns?

## Code Quality Pass

### Complexity

- Are functions too complex?
- Is the single responsibility principle followed? Function or class should do one 'thing'.
- Should a function be broken into multiple functions?
- If a method has greater than 3 arguments, it is potentially overly complex.
- Does the code add functionality that isn't needed?
- Can the code be understood easily by code readers?

### Naming/Readability

- Did the developer pick good names for functions, variables, etc?

### Error Handling

- Are errors handled gracefully and explicitly where necessary?

### Functionality

- Is there parallel programming in this PR that could cause race conditions? Carefully read through this logic.
- Could the code be optimized? For example: are there more calls to the database than need be?

- How does the functionality fit in the bigger picture? Can it have negative effects to the overall system?
- Are there security flaws?
- Does a variable name reveal any customer specific information?
- Is PII and EUII treated correctly? Are we logging any PII information?

## Style

- Are there extraneous comments? If the code isn't clear enough to explain itself, then the code should be made simpler. Comments may be there to explain why some code exists.
- Does the code adhere to the style guide/conventions that we have agreed upon? We use automated styling like black and prettier.

## Tests

- Tests should always be committed in the same PR as the code itself ('I'll add tests next' is not acceptable).
- Make sure tests are sensible and valid assumptions are made.
- Make sure edge cases are handled as well.
- Tests can be a great source to understand the changes. It can be a strategy to look at tests first to help you understand the changes better.

Last update: August 22, 2024