CoGrammar.com

Kinds of Code Review |

Code review can be used in various settings, including technical, educational, and assessment settings. This document describes some variants of code review, and some parameters that we take into account when doing code reviews.

Educational review |

These reviews focus on the *student*, in order to help the student learn. The reviewer will look for areas of weakness, and turn the code review into a mini personalised lesson, giving highly practical and actionable feedback to allow the student to improve their own knowledge.

The reviewer will focus on technological weakness, even if this is not specifically related to the student's assigned task (e.g. the reviewer can point out inefficiencies or bad style, even if the assignment did not explicitly ask the student to write efficient code in a good style).

Assessment review |

These reviews focus on the *submitted code*, in order to assess how well a student has followed task requirements. These reviews are generally shorter than our Educational reviews above. They focus on objectivity and quantitative assessment, following either one of our rubrics or one provided by our clients.

Reviewers will give grades based on a (custom) set of attributes, such as style, completeness, efficiency, communication, and attention to detail. Reviewers can also give an overall grade following a custom or standard grading system, such as 1-100% or A-F.

Technical review |

These reviews focus on the *submitted code*, in order to ensure that this code is production ready and bug-free. Many of these reviews are simply confirmations that the code is ready to be shipped, but reviewers also focus on bugs, inefficiencies and security vulnerabilities.

These reviews are often shorter than our education and assessment reviews, and consist mainly of pointing out flaws (without necessarily providing fixes), or suggesting code changes.

Code review parameters |

These are the primary factors we take into account when generating custom pricing for our code review as a service product.

Length of submission |

Longer submissions take more time to review. A good length for code submissions is often 100-500 lines of code, as this allows the reviewer to comment closely on each piece of submitted code. Longer submissions are more difficult for the reviewer to fully understand (they need to keep more information in their working memory), and as review length does not grow linearly with code submission length, reviews for shorter submissions can often go into more detail.

Complexity of submission and technologies used |

The complexity of a code submission is broken into two broad factors – the complexity of the *code* itself (such as use of algorithms, data structures, and other Computer Science fundamentals) and the complexity of the *technology* used.

Code complexity |

We use a tiered code reviewer system, which is closely linked to program complexity. A code review for a code submission that contains basic logic flows, input and output is easier to review than a code submission that makes substantial use of advanced computer science concepts like concurrency, parallelism, memory management, networking, statistics, compiling, etc.

Technology complexity |

If the technologies used are standard and popular, it is more likely that our reviewers will have direct expertise. For example, code submissions written in Python are easier to review than those written in Delphi. The same holds for frameworks used – code written using Django, Ruby on Rails, or React is easier to review than that written using Pyramid, Sinatra, or Deku.

The variety of technologies used within one code submission is also taken into account – a code review containing HTML, CSS, JavaScript, Python, YAML and bash is more difficult to review than a submission of equivalent length that contains only Python.

Various other factors are taken into account here, such as if the submission relies on a lot of obscure libraries, uses outdated versions, or follows non-standard style guides.

Available contexts |

There are different kinds of context that affect the time a reviewer needs to complete a code review, as well as the quality of the review.

- **Project context** if the entire coding project is contained in the review, it is easier for the reviewer to see how everything fits together. If the code review is part of a larger project, the reviewer needs to spend time guessing how this piece might fit in, or even spend time reading through a larger piece of the project (if available)

- **Task context** if the reviewer knows *exactly* the goals of the code review, it is easier to a) find mistakes and room for improvement and b) provide feedback in line with the original expectations. This context could be provided by the author of the code (i.e. docstrings and comments, and/or a README file, explaining what the code is *meant* to do, or it could be provided by a third-party (e.g. the assignment written by a teacher and provided to the student - the student could then submit the assignment doc along with their attempt at a solution).

- **Student context** if the reviewer reviews for the same person on an on-going basis, it is easier to give appropriate feedback, tailored for that coder's level. Failing this, other context can be provided along with a code submission, such as how many years experience the coder has, how much experience they have with the technologies used in the code submission, etc.

Review scope |

We offer different scopes of code review. This refers to what kind of errors or improvements reviewers will consider when doing the code review. For custom code reviews, we can do different levels of code review, which can include **any or all** of the following aspects in our reviews.

- **Correctness** - syntax and semantics. Our reviewers check for bugs, errors in logical thinking, edge cases, syntax errors, and check if the code submitted correctly matches the specification provided (if available).

- **Efficiency** - our reviewers discuss improvements to existing structures, even if the submitted code does what it's meant to do, it can often be improved.

- **Style** - code can be consistent in a number of ways – we can check, in increasing order of difficulty, for *internal* consistency (the submitted code doesn't use different styles in different parts), *external* consistency (the code matches best practices for that language, e.g. PEP8 for Python code), or *custom* style (the code should match a custom style guide provided)

- **Plagiarism** - we can check that the code hasn't been copied or stolen. We use proprietary technology and can detect plagiarised code even when effort has been made to disguise the plagiarism, such as through changing variable names.

- **Code smells** - these are common constructs that aren't necessarily bad on their own, but which often strongly suggest either a) a theoretical gap in the coder's knowledge or b) dependence on other code that is significantly flawed. Our reviewers can fill in gaps in the coder's knowledge by inferring what the coder doesn't yet know, or find potential room for improvement in other parts of the code base that are not part of the code submission.

- **Communication** - we look at how well the coder communicates their code. This includes the quality and quantity of documentation included, whether in comments, docstrings, a README file, or formal documentation, as well as how readable the code is (how well the coder communicates through the code itself).

- **Choice of technology** - a higher level review where the reviewer comments on the tools, languages, frameworks and constructs that the coder chose to use. For example, if the code submission contains a bunch of long bash scripts for system configuration, the reviewer might suggest that tooling such as Ansible could be considered instead.

Get in touch |

The above variations are not fully comprehensive, and at CoGrammar we customize the code review experience to exactly your needs. If you have more questions or more specialised needs, reach out to us at any time to be put in touch with a code review expert who can advise further.