A CODE REVIEW CHECKLIST QUESTIONNAIRE

Implementation

1. Does this code change do what it is supposed to do?
2. Can this solution be simplified?
3. Does this change add unwanted compile-time or run-time dependencies?
4. Was a framework, API, library, service used that should not be used?
5. Was a framework, API, library, service not used that could improve the solution?
6. Is the code at the right abstraction level?
7. Is the code modular enough?
8. Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?
9. Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?
10. Are there any best practices, design patterns or language-specific patterns that could substantially improve this code?
11. Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-close principle, Liskov Substitution Principle, Interface Segregation, Dependency Injection?

Logic Errors and Bugs

1. Can you think of any use case in which the code does not behave as intended?
2. Can you think of any inputs or external events that could break the code?

Error Handling and Logging

1. Is error handling done the correct way?
2. Should any logging or debugging information be added or removed?
3. Are error messages user-friendly?
4. Are there enough log events and are they written in a way that allows for easy debugging?

Usability and Accessibility

1. Is the proposed solution well designed from a usability perspective?
2. Is the API well documented?
3. Is the proposed solution (UI) accessible?
4. Is the API/UI intuitive to use?

Testing and Testability

1. Is the code testable?
2. Does it have enough automated tests (unit/integration/system tests)?
3. Do the existing tests reasonably cover the code change?
4. Are there some test cases, input or edge cases that should be tested in addition?

Dependencies

1. If this change requires updates outside of the code, like updating the documentation, configuration, readme files, was this done?
2. Might this change have any ramifications for other parts of the system, backward compatibility?

Security and Data Privacy

1. Does this code open the software for security vulnerabilities?
2. Are authorization and authentication handled in the right way?
3. Is sensitive data like user data, credit card information securely handled and stored? Is the right encryption used?
4. Does this code change reveal some secret information (like keys, usernames, etc.)?
5. If code deals with user input, does it address security vulnerabilities such as cross-site scripting, SQL injection, does it do input sanitization and validation?
6. Is data retrieved from external APIs or libraries checked accordingly?

Performance

1. Do you think this code change will impact system performance in a negative way?
2. Do you see any potential to improve the performance of the code?

Readability

1. Was the code easy to understand?
2. Which parts were confusing to you and why?
3. Can the readability of the code be improved by smaller methods?
4. Can the readability of the code be improved by different function/method or variable names?
5. Is the code located in the right file/folder/package?
6. Do you think certain methods should be restructured to have a more intuitive control flow?
7. Is the data flow understandable?
8. Are there redundant comments?
9. Could some comments convey the message better?
10. Would more comments make the code more understandable?
11. Could some comments be removed by making the code itself more readable?
12. Is there any commented out code?

Experts Opinion

1. Do you think a specific expert, like a security expert or a usability expert, should look over the code before it can be committed?

2. Will this code change impact different teams? Should they have a say on the change as well?