Do's for writing a code

1. Code formatting

While writing the code, check the code formatting to improve readability and ensure that there are no blockers:

a) Use alignments (left margin), proper white space. Also ensure that code block starting point and ending point are easily identifiable.

b) Ensure that proper naming conventions (Pascal, Camel Case etc.) have been followed.

c) Code should fit in the standard 14 inch laptop screen. There shouldn't be a need to scroll horizontally to view the code. In a 21 inch monitor, other windows (toolbox, properties etc.) can be opened while modifying code, so always write code keeping in view a 14 inch monitor.

d) Remove the commented code as this is always a blocker, while going through the code. Commented code can be obtained from Source Control (like SVN), if required.

2. Architecture

a) The code should follow the defined architecture:-

1. Separation of Concerns followed

   - Split into multiple layers and tiers as per requirements (Presentation, Business and Data layers).
   - Split into respective files (HTML, JavaScript and CSS).

2. Code is in sync with existing code patterns/technologies.
3. Design patterns: Use appropriate design pattern (if it helps), after completely understanding the problem and context.

3. Coding best practices

- No hard coding, use constants/configuration values.
- Group similar values under an enumeration (enum).
- Comments – Do not write comments for what you are doing, instead write comments on why you are doing. Specify about any hacks, workaround and temporary fixes. Additionally, mention pending tasks in your to-do comments, which can be tracked easily.
- Avoid multiple if/else blocks.

4. Non Functional requirements

a) Maintainability (Supportability) – The application should require the least amount of effort to support in near future. It should be easy to identify and fix a defect.

1. **Readability:** Code should be self-explanatory. *Get a feel of story reading, while going through the code*. Use appropriate name for variables, functions and classes. If you are taking more time to understand the code, then either code needs refactoring or at least comments have to be written to make it clear.

2. **Testability:** The code should be easy to test. Refactor into a separate function (if required). Use interfaces while talking to other layers, as interfaces can be mocked easily. Try to avoid static functions, singleton classes as these are not easily testable by mocks.
3. **Debuggability:** Provide support to log the flow of control, parameter data and exception details to find the root cause easily. If you are using Log4Net like component then add support for database logging also, as querying the log table is easy.
4. **Configurability:** Keep the configurable values in place (XML file, database table) so that no code changes are required, if the data is changed frequently.

b) Reusability

1. DRY (Do not Repeat Yourself) principle: The same code should not be repeated more than twice.
2. Consider reusable services, functions and components.
3. Consider generic functions and classes.

c) Reliability – Exception handling and cleanup (dispose) resources.

d) Extensibility – Easy to add enhancements with minimal changes to the existing code. One component should be easily replaceable by a better component.

e) Security – Authentication, authorization, input data validation against security threats such as SQL injectionsand Cross Site Scripting (XSS), encrypting the sensitive data (passwords, credit card information etc.)

f) Performance

1. Use a data type that best suits the needs such as String Builder, generic collection classes.
2. Lazy loading, asynchronous and parallel processing.
3. Caching and session/application data.

g) Scalability
– Consider if it supports a large user base/data? Can this be deployed into web farms?

h) Usability – Put yourself in the shoes of a end-user and ascertain, if the user interface/API is easy to understand and use. If you are not convinced with the user interface design, then start discussing your ideas with the business analyst.

The first step while assessing the code quality of the entire project is through a static code analysis tool. **Use the tools (based on technology) such as SonarQube, NDepend, FxCop, TFS code analysis rules. There is a myth that static code analysis tools are only for managers.**
Use plug-ins such as Resharper, which suggests the best practices in Visual studio.To track the code review comments use the tools like Crucible, Bitbucket and TFS code review process.