# 10 Best Practices for Code Commenting

BY SAMIR BEHARA *on* FEBRUARY 25, 2016 • ( 4 )

2 Votes

Comments play an integral role in determining an application's readability. In a software development lifecycle, there comes a phase where every application needs to be maintained. With time, developers change and the application needs to be handled by a new set of people. Having a code that is easily understandable, makes life much easier.

Having worked in a number of projects, with varied complexity levels, I have learned the significance of code comments. In this article, I will be sharing few of the best practices of writing code comments and I hope that you will find it useful.

1. **Use clear and understandable naming conventions**

This is the first and foremost step for creating a reliable and maintainable application. You can make your code tell its own story by following this simple rule.

When you write code, make sure to give meaningful names to your methods, classes and variables. This helps in making the code self-explanatory and reduces the need of explicit commenting.

2. **Let your Unit Tests and Integration Tests do the talking**

Unit Tests checks if the discrete piece of code is doing what it is supposed to do whereas Integration Tests checks if different pieces of the modules are working together.

Your Unit Tests and Integration Tests ideally should explain what a particular block of code is doing. If you follow Test Driven Development, you will always be confident that you will have failing tests, if your code is incorrect.

Having good test coverage and writing well named Unit/Integration Tests, is the best way to make your code understandable and most importantly reliable. Unlike comments, Unit Tests can never be stale. They will give you immediate feedback when your code functionality changes.

3. **Follow Design Principles**

If you follow good design principles, your code becomes more organized and clear.

If you write a class, which has just a single responsibility, it makes the code much easier to understand. Similarly if you follow open closed principle, when requirements change, you will add new code and not change the old code that already works.

4. **Keep your comments meaningful and concise**

Comments are very essential from a maintainability standpoint. The comments should always explain WHY you are making a particular code change.

When you have a complex Business logic, you need to ensure to add proper comments explaining the logic behind the changes. This will help another developer to go through the code and understand WHY the changes were made. It is always a good practice to write the comments, as you write your code. Also make sure not to over comment.

5. **Leverage the benefits of XML Comments**

XML comments are a great way to create custom documentation for your source code. One of the best thing about using XML comments is that it provides information about the methods/variables when you hover over the type in Visual Studio IDE. This Intellisense support helps developers to have a better understanding of the code and increase their coding speed as well.

6. **Make use of TODO Comments**

While working on an application code, there are scenarios when you bump across an existing code which needs refactoring, has missing/improper comments, has not been covered by a unit test, or else there is a better way to implement the functionality. In all these cases, you would like to come back to it at a later point of time and fix it. TODO comments are apt to be used in such scenarios.

The Task List window in Visual Studio IDE lists all your TODO comments and helps you track all these tasks in a single place. You can evaluate them during your development or code review process and delete them as and when they are addressed.

7. **Generate MSDN Style documentation with SandCastle**

Sandcastle is a nice tool from Microsoft that helps you to generate great documentation with an MSDN look and feel. It pulls all the XML comments on methods/variables in your code and converts it into HTML format.

8. **Remove commented out codes**

In a world of source control systems, we don't need the code which is not being used. Commented out codes create confusion and reduces code readability. In the long run, it adds to the overall maintenance cost as well.

9. **Do not allow Out-of-date comments to increase your Technical Debt**

Ask any developer a simple question – What is one thing which changes most frequently?Most of the time the answer which you will hear is – 'Requirements'.

Now if the requirement changes, the code needs to change as well. And then what happens to the existing comments? Who changes them? What happens if we do not update those comments?

There is always a maintainability cost associated with code comments. When code changes, it is very important that the associated comments are updated as well. If your code and comments are not in sync, then it is a perfect recipe for disaster at some point in time.

10. **Make sure to verify comments during your Code Review process**

When you write an incorrect code, generally the compiler catches it and throws an error/warning. Also there is every chance that a Unit Test/Integration Test will fail immediately and provide you a feedback that something is wrong. However the same does not stand true in case of comments.

There is no automated way to verify the correctness of comments. This creates a necessity to validate their correctness during the Code Review process. We need to ensure that with every code change, the related comments are updated accordingly.

> ***Summing up this article, I think comments are indispensable for an application readability and maintainability. But we need to very diligent in keeping them updated, else they can very easily become a code smell and increase our technical debt over a period of time.***



‹ Code Analysis Improvements in Visual Studio 2015 Update 1
Live Query Statistics in SQL Server 2016 ›

Categories: C#, Coding Standards

## 4 replies

**youvebeengoonedRussell Gilbert**
January 20, 2017 • 3:52 am

Every code comment I make represents my failure to express myself clearly in code (see Uncle Bob for more detail). Don't like TODO comments. If some refactoring is needed either do it there and then or put a story in the backlog.

Like
Reply ↓

**zameb**
May 14, 2021 • 4:04 pm

Agree the first, disagree on the second. TODO comments are some of the few acceptable ones. Yes, putting on the backlog can be a good substitution for that.

But, for example. Today I´m implementing some cache functionality on the query interfaces. But there's another story for the command part. A few tests will have to be adjusted after this command part is implemented. Yup, maybe the command should have been done before… no, it is untesteable if we don't have the query part. Maybe we should have not split the query and command parts in 2 stories… no, it is a value for the user to have some cache improvements on the reads NOW and the command part just will add some details that are not very valuable right now.

Like
Reply ↓

## Trackbacks

1. Celebrating 1 year of DotNetVibes – dotnetvibes
2. 10 Best Practices for Code Commenting – dotnetvibes | Veteranos de las Nuevas Tecnologías

## Leave a Reply