Code Review Process

Brief

Code review, a software quality assurance activity in which one or several humans check a source-code mainly by viewing and reading chunks and parts. And one or several developers perform required actions to refactor source-code. [Read More]

Mainly, code review done either manual or by using tools, however, it can be done by several ways in order to improve code quality which are mentioned below:

- Formal Inspection
- Regular change-based code review
- Peer-to-peer code review
- Use Code review tools

This code review process covers the following aspects at a very high level:

- Over all architecture, patterns and practices
- Maintainability
- Code Readability
- Quality of code
- Internationalisation/Localisation (Optional)
- Security
- SQL optimisation aspects

Quick Review Checklist

- ☐ Proper Naming conventions and practices should be followed as per coding standards. (To check and verify the coding standards we can use Resharper tool.)
- ☐ Build the solution and check for error/warning. It should not be any project warnings.
- ☐ Perform code analysis (with enabling all settings) by clicking on “Build -> Run Code Analysis on solution” and remove or fix all errors and warnings.
- ☐ Remove unassigned using statements.
- ☐ Remove unused and unreachable code if any.
- ☐ Remove unused methods for the classes.
- ☐ Null check needs to be performed wherever applicable to avoid Null Reference Exception.
- ☐ Use centralized exception handling and logging, remove try/catch clauses from code if added.
- ☐ Method has less number of lines of code. Approx. 30 to 40 lines only.
- ☐ Remove/Avoid nested for/foreach loops and nested if conditions as much as possible.

- ☐ Use anonymous types if code is going to be used only once. (i.e. var)
- ☐ Use access specifiers to define scopes. (private, public, internal, protected)
- ☐ Mark classes as sealed or static or abstract.
- ☐ Use constant and readonly wherever applicable.
- ☐ Use StringBuilder instead of string for concatenation.
- ☐ Avoid type casting and type conversions as much as possible. It impacts performance of application.

Detailed Code Review

Architectural Patterns and Practices

The application should use a variety of architectural patterns and common industry practices such as the ones mentioned below:

- Model View Controller (MVC)
- Dependency Injection
- Object Relational Mapper (ORM) to talk to the database

Maintainability

For the application to be maintainable, it has to have the following attributes:

- Code must be readable, and easy to understand
- Application must use good architectural patterns
- Implementation and code must be consistent throughout
- Design must ensure that it has low coupling and high cohesion

Code Quality

- The code should follow standard coding and naming conventions.
- For instance, parameters should not start with an underscore, should not use camel case. It is recommended that standard naming conventions be used. Good resource for naming convention can be found here. A tool like ReSharper will help in checking code conventions while coding.
- Do not use names of interfaces/class as variable names. For instance, avoid the code shown below:

```
        IExceptionLog IExceptionLog = new ExceptionLog();
```

- Remove unnecessary “Using” statements at the beginning of a file
- Use implicitly typed local variables wherever possible. (See this link)
- Remove redundant qualifiers for classes that have been included via the Using directive.
  For example, instead of using System.Net.HttpStatusCode.NotAcceptable in code, just use HttpStatusCode.NotAcceptable.

- Use namespaces to match folder structure.
- While using LINQ, replace statements that check for a “Count” to be greater than 0, with a “Any” statement. In addition, instead of using Where and SingleOrDefault together, use just SingleOrDefault as shown in the snippet below:

```
        applicationUsersToken.SingleOrDefault(t => t.UserId == userId && t.AuthToken ==
request.Headers.Authorization.Scheme);
        //instead of
        applicationUsersToken.Where(t => t.UserId == userId && t.AuthToken ==
request.Headers.Authorization.Scheme).SingleOrDefault();
```

- Use ?? Operator instead of (x == null) ? …: …
- Use object initialisers where possible.
- Remove redundant catch exceptions, as shown below:

```
        catch(Exception)
        {
            throw;
        }
```

- Avoid swallowing exceptions, as shown in the code snippet below:

```
        try
        {
            Uow.Save();
        }
        catch (Exception)
        {
        }
```

- Remove hardcoded strings and numbers. It should be converted into constant strings.
- Code has potential issues with closure, while using LINQ queries. Use temporary local variables to address this issue.
- Some of the foreach statements can be converted to LINQ expressions to make the code simple.
- Some of the for statements can be converted to foreach statements for code readability.
- Avoid “this.” in code to reduce code clutter.
- JavaScript coding conventions should also be addressed –
    - Remove hardcoded values,
    - Reuse of parameter variable names that appear in the outer scope,
    - Missing semi-colon for line ending,
    - Re-check code for common coding mistakes such as the use of == instead of =.
- Ensure that async methods have a matching await in its implementation.
- If some portion of code are repeatedly in use, extract it as method and reuse it as generic method.
- There should be code consistency, if you use int32 or int for integer values, use string instead of String. It should be same throughout the system.
- Usage of out and ref keywords should be avoided as per code analysis.

- Object initialization can be simplified by inline variable declarations.
- Do not leave debugging code. Remove debuggers.
- some of the heavy process code execution, it should run asynchronously. (like, bulk email sending)
- Utilities / Common functionalities should be at central location.
- Date time parsing should be proper, do not convert date time to string or string to date time.
- HTML code should not contain inline or internal CSS codes. CSS and JS code must be bundled.

Memory Usage

- Memory leaks in the system can lead to slowing down of the application over a period of time and lead to crashes. Usage of the Disposable pattern can ensure that items are properly garbage collected. It is highly recommended that memory profiler be run to check for memory leaks.

  For example, the code creates a SqlConnection in the constructor, but the class does not implement IDisposable, where the connection object is disposed. This could lead to a potential memory leak.
- Dispose unmanaged resources like file I/O, Network resources, use using blocks for unmanaged codes.

Security

- Perform a quick security check for common OWASP Top 10 issues, (see here) it is recommended that the application always be run over https.
- Elements in the cookie should be encrypted – this is deemed unnecessary while running the application over https, and overly complicates the implementation. This can safely be removed if running the application over https.
- Always encrypt sensitive details like password.

Internationalization and Localization

- It would include creating multiple resources for storing strings related to that region, and ensuring that the correct number, currency and date format are used.

SQL Optimization

- Review database schema design, normalization, foreign key usage, relationships.
- Database tables should be normalized and proper naming conventions should be used.
- Check for inefficient queries.
- Remove use of sub-queries in the statements and conditions.
- Replace possible where conditions into joins wherever applicable.

- Proper alias should be given to columns in stored procedures and views.
- Check for improper usage of ROW_NUMBER () in queries.
- Use proper joins in queries.
- Avoid use of ‘*’ in select clause. Also multiple queries with ‘in’ clause have been written. Instead of using long queries, it should be separated within chunk and called upon requirements. Use of inline queries should also be avoided. Sometimes, there might be a risk of SQL injection. Database queries should be placed in the separate layer. We found that database queries are placed in the same page throughout the application.
- Try and avoid using Union All.
- Remove redundant/improper conditions from where clause like “Where 1=2”.