**Code Konvention und Clean Code**

**Code Konvention**

**-**Methoden und Variablen in englischer Sprache

**-**Programmieren nach den C# code conventions:

<https://docs.microsoft.com/de-de/dotnet/csharp/fundamentals/coding-style/coding-conventions>

**Clean Code**

**General rules**

- Follow standard conventions.

- Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.

- Boy scout rule. Leave the campground cleaner than you found it.

- Always find root cause. Always look for the root cause of a problem.

**Design rules**

- Keep configurable data at high levels.

- Prefer polymorphism to if/else or switch/case.

- Separate multi-threading code.

- Prevent over-configurability.

- Use dependency injection.

- Follow Law of Demeter. A class should know only its direct dependencies.

**Understandability tips**

- Be consistent. If you do something a certain way, do all similar things in the same way.

- Use explanatory variables.

- Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place.

- Prefer dedicated value objects to primitive type.

- Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.

- Avoid negative conditionals.

**Names rules**

- Choose descriptive and unambiguous names.

- Make meaningful distinction.

- Use pronounceable names.

- Use searchable names.

- Replace magic numbers with named constants.

- Avoid encodings. Don't append prefixes or type information.

**Functions rules**

- Small.

- Do one thing.

- Use descriptive names.

- Prefer fewer arguments.

- Have no side effects.

- Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

**Comments rules**

- Always try to explain yourself in code.

- Don't be redundant.

- Don't add obvious noise.

- Don't use closing brace comments.

- Don't comment out code. Just remove.

- Use as explanation of intent.

- Use as clarification of code.

- Use as warning of consequences.

**Source code structure**

- Separate concepts vertically.

- Related code should appear vertically dense.

- Declare variables close to their usage.

- Dependent functions should be close.

- Similar functions should be close.

- Place functions in the downward direction.

- Keep lines short.

- Don't use horizontal alignment.

- Use white space to associate related things and disassociate weakly related.

- Don't break indentation.

**Objects and data structures**

- Hide internal structure.

- Prefer data structures.

- Avoid hybrids structures (half object and half data).

- Should be small.

- Do one thing.

- Small number of instance variables.

- Base class should know nothing about their derivatives.

- Better to have many functions than to pass some code into a function to select a behavior.

- Prefer non-static methods to static methods.

**Tests**

- One assert per test.

- Readable.

- Fast.

- Independent.

- Repeatable.

**Code smells**

- Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.

- Fragility. The software breaks in many places due to a single change.

- Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.

- Needless Complexity.

- Needless Repetition.

- Opacity. The code is hard to understand.