# Clean Code Cheat Sheet

**Loose Coupling**
Two classes, components or modules are coupled when at least one of them uses the other. The less these items know about each other, the looser they are coupled. A component that is only loosely coupled to its environment can be more easily changed or replaced than a strongly coupled component.

**High Cohesion**
Cohesion is the degree to which elements of a whole belong together. Methods and fields in a single class and classes of a component should have high cohesion. High cohesion in classes and components results in simpler, more easily understandable code structure and design.

**Change is Local**
When a software system has to be maintained, extended and changed for a long time, keeping change local reduces involved costs and risks. Keeping change local means that there are boundaries in the design which changes do not cross.

**It is Easy to Remove**
We normally build software by adding, extending or changing features. However, removing elements is important so that the overall design can be kept as simple as possible. When a block gets too complicated, it has to be removed and replaced with one or more simpler blocks.

**Mind-sized Components**
Break your system down into components that are of a size you can grasp within your mind so that you can predict consequences of changes easily (dependencies, control flow, …).

**Keep Configurable Data at High Levels**
If you have a constant such as default or configuration value that is known and expected at a high level of abstraction, do not bury it in a low-level function. Expose it as an argument to the low-level function called from the high-level function.

**Don’t Be Arbitrary**
Have a reason for the way you structure your code, and make sure that reason is communicated by the structure of the code. If a structure

appears arbitrary, others will feel empowered to change it.

**Be Precise**
When you make a decision in your code, make sure you make it precisely. Know why you have made it and how you will deal with any exceptions.

**Structure over Convention**
Enforce design decisions with structure over convention. Naming conventions are good, but they are inferior to structures that force compliance.

**Prefer Polymorphism To If/Else or Switch/Case**
“ONE SWITCH”: There may be no more than one switch statement for a given type of selection. The cases in that switch statement must create polymorphic objects that take the place of other such switch statements in the rest of the system.

**Symmetry / Analogy**
Favour symmetric designs (e.g. Load – Save) and designs that follow analogies (e.g. same design as found in .NET framework).

**Separate Multi-Threading Code**
Do not mix code that handles multi-threading aspects with the rest of the code. Separate them into different classes.

Source: https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf, acceced 18/01/2025