Some important principles, rules, guidelines that ensure a clean software design:

Principles:

* Loose Coupling — if classes use each other, they are coupled together. The less classes are coupled, the easier is to change them.
* High Cohesion — degree to which elements of a whole belong together. Components of the class should be highly cohesive.
* Locality — Changes, maintenance, extensions are only local. This leads to no harming whole environment.
* Removeable — Software Components should be easily removeable.
* Small Components — software system should be only of small components ideally each doing only one task.

Class Design:

* Single Responsibility Principle (SRP) — class should do only one task.
* Open Closed Principle (OCP) — class should be extended not modified.
* Liskov Substitution Principle (LSP) — child classes must be able to replace their super classes.
* Dependency Inversion Principle (DIP) — dependeny is reversed: high level components are free of low-level components.
* Interface Segregation Principle (ISP) — interfaces should be small: classes should not implement unnecessary methods.
* Cohesion Principles:
* Release Reuse Equivalency Principle (RREP) — only together releaseable components should be bundled together.
* Common Closure Principle (CCP) — classes that change together should be bundled together.
* Common Reuse Principle (CRP) — classes that are used together should be bundled together.

Coupling Principles:

* Acyclic Dependencies Principle (ADP) — no dependency cycles.
* Stable Dependencies Principle (SDP) — depend on direction of stability.
* Stable Abstractions Principle (SAP) — the more abstract, the more stable.

High-Level Architecture:

* Keep Configurable Data at High Levels — constants or config datas should be kept in high level.
* Don’t Be Inconsistent— have a convention, principle, rule or guidelines and always follow them.
* Prefer Polymorphism To If/Else or Switch/Case.
* Separate Multi-Threading Code — isolate multi-thread from rest of the code.
* Only one level of Abstraction per layer — stay conform to existing abstraction layers.
* Fields Not Defining State — fields holding data that does not belong to the state of the instance but are to hold temporary data. Use local variables or extract to a class abstracting the performed action.
* Micro Layers — avoid unnecessary design layers.
* Singletons / Service Locator — Make use of dependency injection.
* Base Classes Depending On Their Derivatives — Base classes should work with any derived class.
* Feature Envy — The methods of a class should be interested in the variables and functions of the class they belong to, and not the variables and functions of other classes. Using accessors and mutators of some other object to manipulate its data, is envying the scope of the other object ©.
* Unused Coupling — avoid unused dependencies, be greedy.
* Hidden Coupling — make sure that order of calls to different methods are correct.
* Transitive Navigation — (Law of Demeter), write isolated code. Classes should have access to only its direct dependencies.

Environment:

* Project Build Requires Only One Step.
* Executing Tests Requires Only One Step.
* Source Control System — Always use a source control system.
* Continuous Integration — Assure integrity with Continuous Integration.
* Overridden Logs— Do not override warnings, errors, exception handling