|  |
| --- |
| **S**ingle Responsibility Principle  a class should do one thing and therefore it should have only a single reason to change  **O**pen-Closed Principle  classes should be open for extension and closed to modification  **L**iskov Substitution Principle  subclasses should be substitutable for their base classes  **I**nterface Segregation Principle  is about separating the interfaces  **D**ependency Inversion Principle  classes should depend upon interfaces or abstract classes instead of concrete classes and functions |

**Rule of three** ("Three strikes and you refactor")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KISS** | **DRY** | **AHA** | **SSOT** | **SPOT** |
| **K**eep  **I**t  **S**imple, **S**tupid | **D**on't  **R**epeat  **Y**ourself | **A**void  **H**asty  **A**bstractions | **S**ingle  **S**ource  **O**f  **T**ruth | **S**ingle  **P**oint  **O**f  **T**ruth |

**C**omposition **O**ver **I**nheritance

objects with complex behaviors should contain instances of objects with individual behaviors

**A**void **P**remature **O**ptimization

**L**aw **o**f **D**emeter (principle of least knowledge)

**S**eparation **o**f **C**oncerns(SoC)

is an abstract version of single responsibility. Program should be designed with different containers, which should not have access to each other. Each piece of code is completely independent.

**Y**ou **A**ren't **G**oing **t**o **N**eed **I**t (YAGNI)

you should never code for functionality on the off chance that you may need something in the future

(you shouldn't try to solve a problem that doesn't exist)

Minimize coupling

Document Your Code

Transactions

ACID properties of transactions

Atomicity

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency (**Послідовність, Постійність, Узгодженість**)

Data is in a consistent state when a transaction starts and when it ends.

For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability (**Довговічність**)

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.