Hypothetically we have a code review check list in our team

|  |  |  |  |
| --- | --- | --- | --- |
|  | Metrics | Descriptions | principles |
|  | Test coverage | Is there a need to test more cases? |  |
| 1 | Abstraction |  | OOP basic principles |
| 2 | Encapsulation |  | OOP basic principles |
|  | Polymorphism |  | OOP basic principles |
|  | Inheritance |  | OOP basic principles |
|  | Association, Aggregation and Composition |  | OOP |
|  | Composition over inheritance |  | OOP |
| 1 | Single responsibility |  | Solid Principal |
|  | Open–closed |  | Solid Principal |
|  | Liskov substitution |  | Solid Principal |
|  | Interface segregation |  | Solid Principal |
|  | Dependency inversion |  | Solid Principal |
|  | Creator |  | GRASP Principles |
|  | Information Expert |  | GRASP Principles |
|  | Low Coupling |  | GRASP Principles |
|  | High Cohesion |  | GRASP Principles |
|  | Controller |  | GRASP Principles |
|  | Pure Fabrication |  | GRASP Principles |
|  | Polymorphism |  | GRASP Principles |
|  | Readability | Are there any redundant code and comments |  |
|  | **Security** | Does the code expose the system to a cyber attack |  |
|  | **Architecture** | Does the code use **encapsulation** and **modularization** to achieve separation of concerns |  |
|  | **Reusability** | Does the code use reusable components, functions, and services |  |
|  | Long Method |  | Code Smell |
|  | Large Class |  | Code Smell |
|  | Long Parameter List |  | Code Smell |
|  | Hidden dependency |  | Code Smell |
|  | Gang of four Design pattern |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | **Inspection rate** | The speed at which your team reviews a specific amount of code, calculated by dividing lines of code (LoC) by number of inspection hours | **code review metrics** |
|  | Defect rate | The frequency with which you identify a defect, calculated by dividing the defect count by hours spent on inspection | **code review metrics** |
|  | **Defect density** | The number of defects you identify in a specific amount of code, calculated by dividing the defect count by thousands of lines of code (kLOC). | **code review metrics** |
|  |  |  |  |
|  |  |  |  |

1. Regarding to avoid **Hidden Dependency** I suggest Constructor dependency injection for all of your class instead of Resolving Dependency by IOC container in the class constructor
2. Regarding **Single Responsibility** you can delegate the responsibility of AssignValuesFromArguments() to AppSettings object because it has already read setting from ConfigurationManager in ReadAllSettings()