***CLEAN CODE RULES***

**#** Characteristics of **Bad Code** :

\* **Code that not work.**

\* **Code smell.**

\* **Code rot.**

**#** Characteristics of **Clean Code** :

\* **Reusable.**

\* **Maintainable.**

\* **Portability.**

\* **Understandable.**

\* Follow **Clean Code Rule.**

\* Easy to **Inherit.**

\* Easy to **Data Abstraction.**

1. Avoid **Bad Code**. Don’t write code in short time that’s why our code

become bad. Always maintain **“Boy Scout Rule”**.

2**. Use proper indentation** and space as need our code.

3. **Break our code** into **small** part like **class** or **function** as much as

possible.

4. Give **meaningful names** (variable, function. class, template):

\* **Intension reveling name** that’s why it intent its meaning and

work what it is done.

\* **Make** **meaningful distinction** means don’t collapse and overlap

similar word and don’t redundant names.

\* **Use pronounceable name** that’s why we can read and

understand what our name really mean.

\* **Use searchable names** that’s why we can search and find out

our required things when need.

\* **Avoid disinformation** about names that’s why we never confuse

about our works by its name and its work.

\* Use **name which vary** as much as possible.

\* **Avoid encoding** means don’t use name with encoding string.

\* **Use shorter name** as much as possible but be careful about its

meaning.

\* **Use noun** type name for **classes** and **objects**.

\* **Use verb** type names for **methods** / **function**.

\* **Avoid funny type name.**

\* **Avoid same word** for two purposes.

\* **Avoid mental mapping** means no one should not use other

name after reading your name.

\* **Don’t be cute**.

\* Try to maintain **one word per concept** principle.

5. **Function characteristics** :

\* **Function should be small** as much as possible but it should be

**readable** and **clear enough** that we can understand its work by

at a glance.

\* It should contain only **one or two** level of indentation or nesting.

\* A function should do **one and only** A specific task means a

function **do** only **one task**. If we can **divide** the function **logically**

it indicates the function do more than one task. Its called **Single**

**Responsibility Principle**.

\* Function should have **one level** of **Abstraction**.

\* Function should follow **Step Down Rule**.

\* Function name Should **verb** and **descriptive**.

\* Function should not have more than **three arguments**.

\* A function should not have any **side effect**. Its called **Lies**.

\* A function should not **duplicate**. We must follow the **DRY**

**principle**.

\* A function should either change object state or give information

of that object but not both, if it do it leads us in confusion.

6. **Comment :**

\* Try to **avoid comment** for bad code as can as possible.

\* Try to **rewrite the code** avoiding comment if possible.

\* Comments are always **failure**. Old comment is a **horrible**.

\* Use **good comment** if necessary likes **legal** comment,

**informative** comment, **clarification** etc.

\* **Update** your comment day by day **according** to your **code**

otherwise it leads our project in **disaster**.

\* **Avoid** **misleading** comment.

7. **Object and data structures** :

\* **Data Abstraction** in case of **process** and **entity**. It is the internal

**behavior** of objects.

\* **Object** is the **variable** of **new** **data** type and **data** **structure** is the

way of organized data.

\* Object **hide** the **data** from **public** but **data** **structure** **disclose** the

data in public.

\* Always maintain **Law** **of** **Demeter** which says that a method M of

object O can only access:

1. O itself

2. input arguments of M

3. Any object created in M

4. O’s instance variable

\* Each **unit** or method have a **limit** knowledge.

\* Method should **not** **know** about **inners** **information** of objects.

\* Try to maintain a **single** **dot** operation of code.

8. **Class:**

\* First declare the **variable** and **then** **function**. And always

maintain the serial **private** --> **protected** --> **public**.

\* Try to declare all the **variable** and **utility** **function** as **private** at

max **protected**.

\* It should be **small**.

\* It take only **one** responsibility.

\* There has only **one** reason to change the class.

\* Class should follow the **Single** **Responsibility** **Principle**.

\* The method of a class should **manipulate** **one** or **more** variable

of this class so that class become **cohesive**.

\* Always remember **Open** **Close** **principle** for class changes.

\* Should follow Dependency Inversion Principle.

9. **Error Handling:**

\* Don’t returns **error** **codes**, throw an **exception**.

\* Don’t **return** **null** pointer form method.

\* Don’t **pass** **null** pointer as argument of a method.

\* Try to throw a essential error **message** from a exception

handling.

10. **Formatting:**

1) Vertical Formatting:

\* It maintain **top** **down** approach.

\* At the **top** declare **name**, then more **important** **content**,

then **less** important **content** as much we down our code.

\* Maintain **openness** means separate the thoughts by a

single **blank** **line**.

\* **Density** maintainable means tightly related information

stay dense or closed.

\* **Distance** maintainable means vertical distance from one

content to another represent dependency or uses each

other.

\* **Caller** function should above the **callee** function and should

close to each other.

\* Maintain the **ordering** means important content stay above

and less comes last.

2. **Horizontal Formatting:**

\* Different content should separate by **whitespace**.

\* **Related** content should stay **closed** or dence.

\* Maintain **horizontal** **alignment**.