HW18 Software Engineering Principles

Name: Leonel Mantuano

28 MANTUANO FERNANDEZ LEONEL FERNANDO

INTRODUCTION TO THE SOLID PRINCIPLES

They help the quality of the software in view of the developers.

SINGLE RESPONSIBILITY PRINCIPLE

The responsibilities of any component must have a single purpose, to maintain a single reason for change.

OPEN/CLOSED PRINCIPLE

Software entities can only have extensions, in other words they cannot be modified.
Implementations have pure method declarations.
Public methods can be grouped into abstractions, each with a unique purpose.
Can be encapsulated to vary secondary objects, each component decreasing complexity.
It has ways of roaming, aggregation and Parameterization.

INTERFACES, ABSTRACT CLASSES, PURE VIRTUAL CLASSES

Allows the declaration of methods with interface constraints.
Objects cannot be instantiated.

LISKOV SUBSTITUTION PRINCIPLE

An object will always need another object. Both objects may be able to do what the other can do.
The principle can increase reusability, extensibility, and maintainability.

DEPENDENCY INVERSION PRINCIPLE

Order in layers, the abstraction does not depend on the details but on the details of the abstractions.

DEPENDENCY INVERSION PRINCIPLE

High and low level modules depend on abstractions but do not depend on low level, increasing reusability and maintainability.

SOFTWARE ENGINEERING GOALS

To produce quality products with a low budget.

PROBLEM BACKGROUND

Three main paradigms are postulated: modularity, abstraction and encapsulation.

CORE PROBLEM

Programmers lack basic knowledge of multi-paradigm software development.

SOFTWARE ENGINEERING PRINCIPLES

The definition of the principle aids in the completion of the features to be realized.
The practice of idioms express algorithms in a coherent way that exemplify the patterns and principles of solution.

OBSERVATIONS RELATIVE TO MODULARITY

It consists of building modules by grouping abstractions with almost no dependencies, it facilitates the process of loosely coupled units.
It uses part of the encapsulation by hiding the components from the users, for an efficient development it uses coupling and cohesion.
For FP it depends on parameters
LPs consist of rules and facts
Each design decision must be localized in time.

OBSERVATIONS RELATIVE TO ABSTRACTION

The important thing is to keep the important details in the foreground, and leave out the useless ones.
Some abstraction artifacts show the public details.
For a developer creating a software abstraction leads to extensive and tedious analysis.
The most eye-catching problems are: Leaky, excessive abstraction and inefficient control of details.

PARADIGM-INDEPENDENT DEFINITION FOR ABSTRACTION

The declaration of a component may be part of the source code and may become dependent on it, its practices and criteria consist of meaningful tags and identifiers.

OBSERVATIONS RELATIVE TO ENCAPSULATION

There are three types of definition:
Grouping: lacking proposition, and confused with modularity.
Hiding: Linguistic construction
Data organization: Minimizes coupling.

PARADIGM-INDEPENDENT DEFINITION FOR ENCAPSULATION

Details are kept private to prevent access or modification, achieving greater reliability and avoiding accidental coupling.