Name: Angel Guaman
NRC: 7490
Matter: object oriented programming
Teacher: Edison Lascano

Theme: Code Quality

Introduction to the solid principles

Overview of the solid principles

- Single responsibility príncipl
- Open/Closed principle
- Interface segregation principle
- Dependency inversión principle

First five principle

- That claim has not been justifield or widely accepted
- However, whether they are the forst five principles is not very imporant

Desing Problem

- He program must produce maxes that
- Each room has at last one door to an adjoinhing rrom
- A door between two romos

Single responsibility principle

Core ideas

- Every class should be responsable for a single parto f the systems functionality
- A class properties should be narrowly aligned with that responsiblity
- The more genereal principle of “COHESION” which says that the any component (method class , sub-system,ect)

Following this principle can help

- Increase reuse and maintainablity
- Reduce complexity, even thougth the number of classses migth increase

Open/Closed principle

Core ideas

- Original definitions

- A class is open i fis still available for extensión
- A class is closed if it available for use by other class

Interfaces,Abstract classes

- Inheritance allows a specialization (a derived class)

- Data members
- Method declarations
-Method definitions

- An interface is like a base class

- It has no methods implementation
- Java supports interfaces

- The modern Open/Closed principlee encourages developers to

- Derive concrete classes from these abstract components

Liskov Substitution principle

- An 5 object must be able to do everything any T objetc can do

- Co-variance of return type for a method in s
- Post-conditions for a method cannot be weakened is s
- invariants of the super-type must be preserved

- When implementing a specialization , Widget , of ,some ,Product, ensure that

Interface Segregation Principle

- Foundational concept

- An interface represents public methods if a component

- Java does support iterface directly
- An interface is a window or portal into the funcionality of a component
- Core idea
- The public methods of a component can be groped by purpose or responsilibrt

Dependency Inversion

- Organize the system into layers: some layers, like reusable librearies or frameworks
- Components from the abstract layers should not depende on components from the detalis layers
- Abstractions should not dependo n details
- Following the depedency inversión

-Increase Reusability
-Increase Maintainalibity

UNIFYING DEFINITIONS FOR MODULARITY, ABSTRACTION, AND ENCAPSULATION AS A

STEP TOWARD FOUNDATIONAL MULTI-PARADIGM SOFTWARE ENGINEERING

PRINCIPLES

SOFTWARE ENGINEERING GOALS

- Software engineers aim to build quality products on time anwithin Budget
- Some Desirable Qualities:

• understandability• testability • maintainability • efficiency

• reliability •security • extensibility • openness • interoperability

• reusability

COMMON PARADIGMS

• Object orientation (OO) • Aspect orientation (AO)

• Logic programming (LP) • Genetic programming (GP)

• Functional programming (FP) • Structured program (SP)

The all PROBLEM BACKGROUND is Modularity, abstraction, and encapsulation have value in all these common software development paradigms, albeit to different degrees

There are also many other proposed principles that overlap and break up the ideas differently

CORE PROBLEM

• There are no general, unifying definitions, especially for multiparadig softwaredevelopment

- The principles are hard to teach

- Programmer often don’t understand the core principles, and therefore don’t benefit from their guidance, especially in multi-paradigm software development

Propose a template for documenting principles that

• Allows a principle’s definition to go beyond just communicating the underlying conce• provides a basis for assessing adherence to the principle and a foundation for teachinthe principle to programmers

CONTRIBUTIONS OF THIS INITIAL PAPER

The hoped-for contributions of the unified definitions include

providing:

• a starting point for formulating research questions for MPSD

• a foundation for designing and conducting empirical studies

• a basis for defining metrics that can systematically assess quality for MPSD

Software Engineering Principle:

1) a truth or proposition that supports reasoning about the desirable characteristics of a software system

2) a rule for creating software with certain desirable characteristics

3) an aspect of software design that imparts certain desirable characteristics

- that leads to and supports reasoning about desirable characteristics, such as maintainability, efficiency, openness, reusability, etc.
- then the degree to which a software engineer adheres to P should predicate the degree to which Q is present in the software artifacts.

PRINCIPLES VS.BEST PRACTICES, PATTERNS, AND IDIOMS VS. DESIRABLE CHARACTERISTICS

Principles should should give developers ways to

• Reason about design decisions

• Assess whether or how well a design either conforms to a principle

• Balance choices between conflicting objectives and design alternatives.

OBSERVATIONS RELATIVE TO MODULARITY

• The first principle of Robert Martin’s SOLID principles is a restatement of High Cohesion

• OLID principles overlap to some degree with basic idea of Low Coupling are actually specific best practices for achieving Low Coupling in certain contexts

PARADIGM-INDEPENDENT DEFINITION FOR

MODULARITY

Practices and Criteria:

• Localization of design decisions • Low Coupling

• High Cohesion• Modular Reasoning

PARADIGM-INDEPENDENT DEFINITION FORMODULARITY

•Developers must ensure every predicate represents a single idea or responsibility.

•This is done by defining a predicate and set of rules for each design decision.

OBSERVATIONS RELATIVE TO ABSTRACTION

•From a process perspective, abstraction is the act of bringing certain details to the forefront while suppressing all others.

•Abbott et al. described an abstraction as the “reification and conceptualization of a distinction”

OBSERVATIONS RELATIVE TO ENCAPSULATION

• Three categories of existing definition for encapsulation:

• The bundling of data with operations

• The hiding decisions behind logical barriers

• The organization of components to minimize ripple effects

· Abstraction and encapsulation might be considered duals of each other, but one cannot subsume the other because the mechanisms for doing each are different

- Proposed drafts of paradigm-independent definitions for the MAE principles

Showed that these definitions are non-redundant and complimentary