Princípy tvorby softvéru
Dizajnové princípy

Robert Lukoťka
lukotka@dcs.fmph.uniba.sk
www.dcs.fmph.uniba.sk/~lukotka

M-255

Analýza a dizajn - úrovne

- Architektúra
- Vysokoúrovňový analytický model
- ...
- Nízkoúrovňový analytický model
- Implementačný model

Analýza a dizajn

Complexity of most software systems is in the fact that in comprises of a large amount of simple tasks.

Čo chceme dosiahnuť

- Modularization
- Abstraction
- Information hiding
- Separation of interface and implementation
- Low Coupling
- High Cohesion
- Sufficiency, Completeness, Easy to understand, ...

Modularization

Modularizes requirements, implementation, test cases, ...

Abstraction

- Identify aspects that are relevant to the problem
- Different concepts in the problem domain may become identical
- Identify concepts that do not represent a concept in the problem domain that can be useful to simplify stuff

Design Verification and Validation

- Verification - Check if all design outputs meet design conditions imposed at the beginning of the process.
- Validation - Check if all design outputs meet the customer needs.

Paradigms - data and behaviour

- object oriented - data and behaviour combined

  One of the fundamental principles of object-oriented design is to combine data and behavior, so that the basic elements of our system (objects) combine both together. M.Fowler
- procedural - behaviour acts on separated data
- functional - based on immutable values and pure functions
- declarative - abstraction from how to do stuff (often via DSL)
- ...

Dizajnové princípy

Niektoré princípy nie sú závislé na paradigme

- YAGNI
- DRY
- Rule of 3

Dizajnové princípy - Anemic domain model

Objects have no or little behaviour.

M. Fowler, 2003:

> *"The fundamental horror of this anti-pattern is that it's so contrary to the basic idea of object-oriented designing; which is to combine data and process them together. The anemic domain model is just a procedural style design, exactly the kind of thing that object bigots like me ... have been fighting since our early days in Smalltalk. What's worse, many people think that anemic objects are real objects, and thus completely miss the point of what object-oriented design is all about."*

Dizajnové princípy - Anemic domain model

- May indicate lack of abstraction
- Not really an OO design
- This my be completely acceptable or even preferred in other design styles (e.g. procedural, functional design).

Dizajnové princípy - OO paradigma

Information hiding, Separation of interface and implementation

- Encapsulation - a way how to hide internal information
- Interface should not depend on the implementation

Low Coupling

Coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are. Disadvantages of high coupling:

- A change in one module usually forces a ripple effect of changes in other modules.
- Assembly of modules might require more effort and/or time due to the increased inter-module dependency.
- A particular module might be harder to reuse
- Hard to reuse, test
- Message transmission/translation/interpretation overhead
- ...

Coupling

- Information expert principle
  - Placing the responsibility on the class with the most information required to fulfill it.
  - Reduces coupling.
- Coupling strength, Coupling distance (high coupling between objects in the same package is more acceptable)

High Cohesion

Cohesion refers to the degree to which the elements inside a module belong together.

- Various metrics: e.g. LCOM4.
- Types of cohesion

Encapsulate what varies

Ak je niektorá funcionalita vystavéná častým zmenám požiadaviek

- vytvoriť objekt obsahujúci len toto správanie
- skryť za iterface(y)

Ešte výhodnejšia možnosť: abstrakcia

SOLID

- Single responsibility principle
- Open-closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

Single responsibility principle

A class should have only a single responsibility.

This implies:

- high cohesion
- almost necessarily LCOM4$=1$ for the class.

Strategy pattern to separate other responsibility.

Open-closed principle

A class/package should be open for extension, but closed for modification.

Tools:

- Inheritance
- Composition (preferred)

How to do composition and not create a direct dependency?

- Dependency injection (via constructor, via method)
- Factory method

Liskov substitution principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

Example:

- Square as a subclass of Rectangle is probably not a good idea.

Interface segregation principle

Many client-specific interfaces are better than one general-purpose interface.

Implications:

- Limits the impact of an interface change.

Dependency inversion principle

One should depend upon abstractions, not concretions.

- In procedural programming a good practice was that higher level modules depend on lower level modules.
- This creates a chain of dependencies
- This principle asks us to depend on abstractions - typically interfaces.
- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.
- If followed completely, there should be an interface for each potential dependence between classes, but this is not too practical.

Further maxims

1. Composition over inheritance
2. Objects are about behavior, not attributes
3. Strategy pattern - We may alway treat methods like attributes
4. Design for change, not to last

Procedural

- Break procedures and functions into units + give the units a structure
- Model data (very often in DBMS)

Functional

- Identify mutable states, builders
- Algebraic data types
- SOLID - aplicable with modifications
  - OCP - we can use higher order functionas
  - "LSP just says “predicates are contravariant" - Rúnar Óli

Niektoré zdroje

- https://www.slideshare.net/cristalngo/software-design-principles-57388843 Wikipedia - SOLID

Recommended:

1. Bob Martin: Bob Martin SOLID Principles of Object Oriented and Agile Design