Princípy tvorby softvéru
Designové princípy

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

M-255

Analýza a dizajn - úrovne

- Architektúra
- Vysokoúrovňový model
- Nízkoúrovňový model
- Implementačný model

Analýza a dizajn

- Zložitosť väčšiny SW projektov spočíva v tom, že projekt sa skladá z veľkého množstva jednoduchých úloh.

Čo chceme dosiahnuť

- Modularizácia
- Abstrakcia
- Skrývanie informácií
- Oddelenie interfacu a implementácie
- Low Coupling
- High Cohesion
- Postačujúcosť, úplonosť, jednoduchosť, ...

Modularizácia

Delí požiadavky, implementáciu, testy, ...

Abstraction

- Identifikovať aspekty systému relevantné pre problém.
- Pojmy, ktoré sú v doméne rôzne sa môžu stať v našom modeli identické.
- Identifikovať pojmy, ktoré nereprezentujú koncept z domény, ale môžu byť užitočné.

Verifikácia a validácia dizajnu

Aj dizajn možno verifikovať"

- Verifikácia dizajnu - Overenie, že výstup dizajnu spĺňa požiadavky kladené na design (toto sa uskutoční oveľa ľahšie).
- Validácia - Overenie, že výstup dizajnu spĺňa potreby zákazníka.

Dizajnové princípy

Niektoré princípy nie sú závislé na paradigme:

- YAGNI
- DRY
- Rule of 3

Dizajnové princípy - Anemický model domény

Objekty maju málo, alebo žiadne správanie.

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 - Anemický model domény

- Môže identifikovať nedostatok abstrakcie
- Nie je to OO design
- Môže byť akceptovateľný alebo dokonca preferovaný v paradigmách (e.g. procedural, functional design).

Dizajnové princípy - OO paradigma

Skrývanie informácií, Separácia interfacu a implementácie

- Enkapsulácia - OO spôsob skrývania informácií
- Interface by nemal závisieť od implementácie (určite nie na úrovni zdrojového kódu, a len málo na logickej úrovni)

Low Coupling

Coupling (previazanosť) je stupeň závislosti medzi softvérovými modulmi; miera toho ako teste prepojené sú dva moduly navzájom. Nevýhody veľkej previazanosti:

- Zmena v jednom module môže vyvolať lavínu zmien v ostatných moduloch.
- Ťažšie skladanie modulov (napr. nasadenie novej verzie po zmene).
- Ťažkosti pre znovupoužitie modulu
- Ťažšie až nemožné samostatné testovanie modulu
- Výkonová strata spôsobená prenosom, prekladom, a interpretáciou informácií (Môže byť veľmi relevantné napríklad pri microservicoch)
- ...

Low Coupling

- Information expert princíp - Heuristika na nízku previazanosť
  - Zodpovednosť má trieda, ktorá má informácie na vykonanie činnosti.
- Coupling strength, Coupling distance (väčšia previazanosť medzi "blízkymi" triedami je akceptovateľnejšia)
  - Napr. silné previazanie výpočtu miezd na použitú databázu je veľmi nežiadúce.

Vysoká kohézia

Kohézia - ako veľmi veci, ktoré sú v jednom module patria k sebe.

- Príklad triviálnej exaktnej metriky: e.g. LCOM4.

Encapsulate what varies

Ak je niektorá funcionalita vystavená častým zmenám požiadaviek / používa sa súčasne vo viacerých verziách

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

Ešte výhodnejšia možnosť: abstrakcia

Video - Bob Martin - Čo je to OO design

Bob Martin - SOLID Principles of OO and Agile Design
(12:30-34:45, 37:45-52:30)

SOLID

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

Single responsibility principle

Trieda by mala mať jednu a práve jednu zodpovednosť.

Toto implikuje:

- vysoká kohézia
- LCOM4$= 1$
- trieda má zodpovednosť

Ako separovať zodpovednosti? Napr. strategy pattern.

Open-closed principle

Trieda by mala byť otvorená pre rozširovanie ale uzavretá pre modifikáciu.

Nástroje:

- Dedenie
- Skladanie (väčšinou lepšie)

Ako zložiť objekty bez vytvorenia explicitnej závislosti?

- Dependency injection (cez konštruktor/cez metódu)
- Factory method

Liskov substitution principle

Objekty v programe by mali byť nahraditeľné inštanciami ich subtypov bez zmeny korektnosti programu.

Štandardný príklad:

- Štvorec nie je podtrieda Obdĺžnika.

Interface segregation principle

Viac klient-špecifických interfacov je lepších ako jeden veľký všeobecný interface.

Dôsledky:

- Limituje dosah zmeny interfacu.

Dependency inversion principle

Moduly by mali byť závislé od abstrakcií a nie od konkrétností.

- Videli sme video . . .
- Priníp žiada závisieť od abstrakcií - nimi sú typicky interfacy (v duck typing jazykoch ani nemusia byť súčasťou zdrojového kódu).
- Vysokoúroňové moduly nemajú záviesieť na nízkoúrovňových. Oba moduly majú závisieť od obstrakcie (ktorá napr popisuje potreby komunikácie medzi modulmi).
- Abstrakcia nemá záležať na detajloch. Detajly majú zavísieť na abstrakcii.
- Napríklad pri príklade na architektúru sme robili interfacy - abstrakcie a nezaoberali sme sa detajlami modulov, interfacy prišli prvé.
- Úplne dodržiavanie princípu spôsobí, že je interface, medzi každými dvoma triedami. Toto nemusí byť vždy praktické, ale

Ďalšie vybrané formulky.

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.

Niektoré zdroje

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