Objektově-orientovaný návrh

Z FITwiki

Obsah

Funkcionální vs. objektové paradigma

Funkcionální (označovaný také jako funkční, klasický, strukturovaný, procedurální, imperativní)
: základní jednotkou dekompozice složitého softwarového systému je **funkce** (také se používá označení proces). Základním modelem je diagram datových toků, který ukazuje procesy a datové toky mezi nimi.

Objektově orientovaný
: základní abstrakcí a tedy i jednotkou dekompozice je **objekt**. Složitý softwarový systém je členěn na balíky (package), komponenty nebo jiná seskupení tříd objektů, mezi kterými mohou existovat různé vztahy. V UML existuje řada prostředků pro objektově orientované modelování struktury a chování.

OO návrh

Podstata OO návrhu

Nalézt třídy a interakci jejich objektů, které budou realizovat chování reprezentované případy použití při respektování omezení daných nefunkčními požadavky.

Pro OO návrh je kritická znalost a schopnost aplikovat principy návrhu (ne znalost UML, nebo konkrétních technologií).

Vstupy OO návrhu

- Specifikace případů použití
- Doplňující specifikace

- Systémové diagramy sekvence (SSD)
- Kontrakty operací
- Slovník
- Model domény

Výstupy OO návrhu

- diagram tříd
- diagramy interakce
- ...

Návrh řízený zodpovědností

- RDD = Responsibility-driven design oblíbený způsob uvažování o návrhu tříd

návrh řízený zodpovědností je způsob návrhu založený na uvažování o zodpovědnostech a spolupráci, jejich přiřazování třídám. Hledání množiny spolupracujících zodpovědných objektů, které zajistí splnění cílů návrhu.

Postup RDD

- Požadavky specifikované jako use-case, systemove sekvenční diagramy, ...
- Vytvoření doménového modelu
- Identifikace odpovĚdností pro každý krok v use-case
- Přiřazení odpovědnosti doménám

Odpovědnost za činnost

- dělání něčeho
- spuštění nějaké akce u jiných objektů
- řízení a koordinace akcí jiných objektů

Odpovědnost za vědomost

- privátních zapouzdřených dat
- souvisejících objektů
- věcí, které lze odvodit nebo spočítat

GRASP

General Responsibility Assigment Software Patterns - pomůcka pro OO návrh založený na zodpovědnostech

(pozn. koheze = soudržnost)

Tvůrce (creator)

Kdo vytváří objekty?

třídě přiřazena zodpovědnost za vytváření instancí jiné třídy pokud platí, že ji:

- obsahuje
- agreguje
- má pro ni inicializační data
- zaznamenává

- hodně užívá

Přínos: Nezvyšuje vazbu, třída tvůrce velmi pravděpodobně již závisí na vytvářené třídě díky asociaci či zanoření.

Kdy nepoužít: Jsou-li speciální požadavky, které činí vytváření složité, pak raději použít návrhový vzor Factory, resp. Abstract Factory.

Informační expert (Information Expert)

Kdo má za co zodpovídat?

Zodpovědnost za nějakou akci přiřaď informačnímu expertovi - tj. třídě, která má k provedení této akce nejvíce informací, a proto i zodpovědnost

Přínos: Obvykle vede na nízkou vazbu a vysokou kohezi

Kdy nepoužít: Někdy by použití mohlo způsobit duplikaci nebo nárůst vazeb

Nízká vazba (Low Coupling)

Jak redukovat vliv změn?

Rozdělit zodpovědnosti tak, aby nezbytná vazba byla co nejmenší.

Zásady:

- nízká vazba (závislost) mezi třídami
- žádný nebo minimální dopad změny jedné třídy na třídy ostatní
- vysoký potenciál znovupoužití

Přínos: Obvykle vede na nízkou vazbu a vysokou kohezi

Kdy nepoužít: Silná vazba na stabilní a běžné prvky (např. standardní knihovnu) není problém

Řadič (Controller)

Který objekt za vrstvou GUI zpracovává požadavky (události) a koordinuje práci systému?

Přiřaď zodpovědnost objektu, který:

- reprezentuje celý systém (facade controlller) - vhodné pokud je jen málo operací
- reprezentuje scénář případu užití (use case/session controller) - pokud by jediný řadič vedl k malé kohezi

Doporučení:

- Řadič by neměl dělat mnoho - pouze deleguje a koordinuje.
- systémové operace ve vrstvě aplikační logiky ne v GUI

Vysoká koheze (High Cohesion)

Jak nechat objekty soustředěné, srozumitelné, spravovatelné (a jako boční efekt podporovat princip Low Coupling)?

Přiřaď odpovědnost, aby koheze zůstala vysoká. Použij tento princip pro vyhodnocování alternativ.

Přínos: Srozumitelnost, lepší udržovatelnost, zpravidla nízká vazba.

Kdy nepoužít: Někdy je oprávněné akceptovat nižší kohezi (např. soustředit zodpovědnost do jedné třídy z důvodu údržby, např. znalost SQL pro mapování na databázi).

Polymorfismus (Polymorphism)

Kdo je zodpovědný za chování měnící se podle typu?

Je-li chování závislé na typu, zodpovědnost dát polymorfním operacím.

Pouhá konstrukce (Pure Fabrication)

Komu přiřadit zodpovědnost pokud by její přiřazení existujícím třídám narušilo vysokou kohezi a nízkou vazbu?

Vytvoří se speciální třída, která se použije pro dodržení výše uvedených pravidel (pokud nejde aplikovat na existující třídy)

Nepřímost (Indirection)

Komu přiřadit zodpovědnost aby jsme se vyhnuli přímé závislosti?

Vytvoří se třída, která dělá prostředníka a zamezuje přímému propojení (architektura MVC)

Chráněné změny (Protected Variations)

Jak přiřadit zodpovědnosti tak, aby změny prvků neměly nechtěný efekt na další prvky?

Ochraňuje prvky systému před změnami jiných prvků tak, že zavede rozhraní. Prvky pracují pouze s rozhraním objektu a jeho chování se může měnit na základě použité instance.

SOLID

- **Single responsibility** - každé třídě by měla být přiřazena jediná funkce/odpovědnost
- **Open/Closed** - třída by měla být otevřená rozšiřování ale uzavřená modifikaci (v existující třídě se jen opravují chyby, nemění se její chování, aby se nerozbil kód který ji používá)
- **Liskov substition** - klient nadtřídy musí být schopný používat libovolnou její podtřídu aniž by ji znal (metody podtřídy musí fungovat stejně jako u nadtřídy aniž by klient s třídou dělal něco navíc)
- **Interface segregation** - klient by měl záviset jen na vlastnostech třídy, které využívá (třída bude implementovat více rozhraní)
- **Dependendency inversion**
  - Moduly vyšší úrovně (byznys logiky) by neměly záviset na modelech nižší úrovně. Oboje by mělo záviset na abstrakcích.
  - Abstrakce by neměly záviset na detailech. Detaily by měly záviset na abstrakcích.

Kategorie: Státnice MIS | Analýza a návrh informačních systémů | Státnice AIS | Státnice 2011

- Stránka byla naposledy editována 4. 6. 2018 v 10:13.