Principy objektového návrhu

Přednáška 8, LS 2013/2014

Principy objektového návrhu

- Cílem je vytvořit kvalitní návrh, který bude předcházet vzniku symptomů jako:
  - Ztuhlost - změna SW je obtížná
  - Křehkost - úprava způsobí problémy na jiných místech SW
  - Špatná znovupoužitelnost - vyčlenění určité části SW pro znovupoužití je složité či nemožné

Principy objektového návrhu

- SOLID
- DRY
- GRASP

SOLID

- Shrnuje základních pět principů, které by měly být dodrženy při objektovém návrhu, abychom předešli problémům při dalším vývoji.

- Nejedná se přímo o návrhové vzory, ale o principy.

S - Single Responsibility Principle (SRP)

Třídy by měly mít jedinou zodpovědnost / jediný důvod ke změně.

S - SRP

- Každá třída má zodpovědnost právě za jednu věc.
- Výstižný název třídy.
- Vodítko porušení: **Pokud nelze zodpovědnost třídy popsat jednou jednoduchou větou bez použití spojky „a“, pravděpodobně porušuje princip jedné zodpovědnosti.**

O - Open-Closed Principle (OSP)

Třídy by měly být otevřené pro rozšiřování, ale uzavřené pro změny.

O - OSP

- Rozšíření funkčnosti by mělo být možné pouze tím, že přidáme nový kód bez nutnosti zasahovat do kódu existujícího.

- Minimalizace toho, že při úpravách poškodíme jinou část SW spoléhající na původní kód.

- Klíčem je abstrakce a polymorfismus.

- Společná funkčnost v abstraktních třídách, konkrétnosti v potomcích.

L - LSP

- Viz přednáška 3.

- **Podtřídy by měly být zaměnitelné s jejich bázovými třídami.**

- **Odvozené třídy nesmí nikdy vyžadovat více a poskytovat méně než bázová třída.**

I - Interface Segregation Principle (ISP)

Více specifických rozhraní je lepší než jedno univerzální rozhraní.

I - ISP

- Závislost tříd pouze na rozhraních, která používají.
- Pokud má třída více různých typů uživatelů, pak bychom měli pro každý typ uživatele vytvořit vlastní rozhraní.

- **Uživatelé tříd pak závisí pouze na těch rozhraních, která používají.**

D - Dependency Inversion Principle (DIP)

Závislost by vždy měla být na abstraktním ne na konkrétním. Konkrétnější musí záviset na abstraktnějším a ne naopak.

D - DIP

- Všechny závislosti by měly být zaměřeny na rozhraní a abstraktní třídy, nikdy ne pouze na konkrétní implementaci.
- Dodržování vede k výrazné redukci závislostí v kódu.
- Viz vzor Abstract Factory

DRY

- Obecný princip “Don’t Repeat Yourself”, neboli neopakuj se.
  - **Každá dílčí znalost musí mít v systému jedinou, jednoznačnou, směrodatnou reprezentaci.**
- Princip DRY zahrnuje všechny zdroje informací v systému od zdrojového kódu, přes datové schéma po dokumentaci.

DRY

- Každý řádek kódu, který se dostane do aplikace, musí být udržován a je potenciálním zdrojem chyb v budoucnosti.

- Pokud je stejná funkčnost v programu roztroušena na více místech, pak při změně je nutné provést změnu všude, kde je tento kód duplikován.

- Vyvarovat se programování copy-paste-edit.

- Ne každá duplikace je porušením DRY.

- Podstatným hlediskem pro posuzování zda nějaká duplikace je porušením DRY je, zda dané úseky kódu spolu logicky souvisejí – zda vyjadřují stejnou informaci.

GRASP

- General Responsibility Assignment Software Patterns

- Vodítka jak přidělit zodpovědnosti (kdo-co bude dělat).

Zodpovědnost

- **Zodpovědností (responsibility)** rozumíme závazek nebo povinnost prvku (subsystému, třídy, objektu) něco:
  - **Dělat** – udělat něco sám, vyvolat akci jiného prvku a nebo tyto akce koordinovat
  - **Vědět** – znát svoje (privátní) data, znát související prvky a nebo umět něco odvodit či agregovat

Přidělování zodpovědností

- Přidělování zodpovědností (responsibility assignment) je pak určování, která komponenta bude mít kterou zodpovědnost.

- Je to zásadní a netriviální krok navrhování, který vyžaduje značnou kreativitu a zvažování mnoha obvykle protichůdných faktorů.

- Prvním krokem při přidělování zodpovědností je jejich definice. Ta závisí na tom, s jakou granularitou se na systém díváme.

- **Zodpovědnost není to samé co metoda**. Konkrétní metody jsou implementovány, aby byla splněna zodpovědnost.

GRASP

- Jde o sadu principů, které slouží jako **pomůcka při rozhodování** o přidělování zodpovědností třídám-objektům-systémům.

- Při návrhu není možné jednotlivé principy uvažovat samostatně, protože často jsou jejich požadavky proti sobě.

- Vede ke zvážení všech pro a proti a z pohledu všech principů.

GRASP - Principy

- Jednotlivé principy GRASP mají různou úroveň abstraktnosti. Některé z nich jsou pouze obecné zásady. Jiné představují konkrétní doporučovaná řešení.

GRASP - Principy

- Protected variations – Chráněné změny
- High cohesion - Vysoká soudržnost
- Low coupling – Slabá provázanost
- Pure fabrication – Pouhá konstrukce
- Polymorphism
- Indirection - Nepřímé vazby
- Information expert – Informační expert/Expert
- Creator – Tvůrce
- Controller

Protected Variations

- Chráněné změny.

- Většina systémů se rozvíjí a neustále se tedy v nich provádějí změny.

- Každá změna představuje určité riziko, že ostatní části systému přestanou fungovat.

- Jak na to: **Identifikujte místa pravděpodobných změn a nestability a zodpovědnosti přiřaďte stabilním rozhraním, která kolem nich vytvoříte.**

Protected Variations

- Jeden z nejobecnějších principů.
- **Čím méně stabilní určité části systému jsou, tím důležitější je dodržovat tento princip.**
- Návaznost na LSP, Open-Closed Principle, Deméteřin zákon.

High Cohesion

- Vysoká soudržnost. Jedná se o obecný princip.
- Pokud náš návrh obsahuje prvky s nízkou soudržností, nelze se vyhnout řadě problémů.
  - Těžká pochopitelnost jednotlivých prvků, míchání nesouvisejících věcí.
  - Obtížná znovupoužitelnost.

High Cohesion

- Výhody:
  - Třídy s vysokou soudržností se snadněji chápou
  - Lze je snadno udržovat
    - Jsou stabilnější a méně často se mění
    - Je podpořena znovu-použitelnost
- Důvodem nízké soudržnosti bývá často příliš hrubá granularita zodpovědností.
- Příčinou může být postupné přidávání zodpovědností.

High Cohesion

- Hrubou metrikou, kterou doporučuje Robert C. Martin, je podívat se na to kolik každá její metoda používá instančních proměnných.

- Pokud mnoho metod používá pouze malou část instančních proměnných, je to známka toho, že třída je málo soudržná.

High Cohesion

- Podle Craiga Larmana soudržnost můžeme rámcově rozdělit na několik úrovní:
  - **Velmi nízká soudržnost** – třída je zodpovědná za dvě nebo více nesouvisejících funkčních oblastí (například zajišťuje business logiku i práci s databází).
  - **Nízká soudržnost** – třída je zodpovědná za mnoho funkcí v jedné funkční oblasti. Obsahuje velké množství metod a pomocného kódu (příkladem je třída, která sama zodpovídá za všechny operace s databází). Takovéto třídy by měly být rozděleny na více menších vzájemně spolupracujících tříd.
  - **Vysoká soudržnost** – Třída má přiměřené množství zodpovědností v jedné funkční oblasti a na provedení úkolů spolupracuje s dalšími třídami.
  - **Přiměřená soudržnost** – Třída má několik zodpovědností menšího rozsahu, které všechny souvisejí s účelem třídy, ale ne spolu navzájem. Toto je typický případ Controllerů a objektů na vyšších úrovních abstrakce (objektů představujících celý subsystém).

High Cohesion

- Související principy: Princip jedné odpovědnosti, Princip oddělení rozhraní

Low Coupling

- Slabá provázanost.
- **Provázanost (coupling)** – je míra toho, jak moc je jeden prvek (třída, subsystém, modul, …) propojen s dalšími prvky, tedy zda o nich má informace nebo na ně spoléhá.
- **Zodpovědnosti přiřazujte tak, aby provázanost zůstala co nejmenší.**
- Určitá míra provázanosti mezi třídami je normální a nezbytná.

Low Coupling

- **Content Coupling** - prvek závisí na interních datech jiného prvku
- **Common Coupling** - prvky sdílejí společná globální data
- **External Coupling** - dva prvky používají stejný datový formát, rozhraní, protokol k nějakému externímu prvku
- **Control Coupling** - jeden prvek řídí interní fungování jiného prvku tím, že mu předává informace o tom co má dělat
- **Stamp Coupling** - více prvků používá stejnou datovou strukturu, z níž každý využívá jen určitou část. Příkladem je předávání celého záznamu do metody, která z něj používá pouze jednu položku.
- **Data Coupling** - Prvky sdílejí data například prostřednictvím parametrů metod a jejich návratových hodnot a předávána jsou pouze data, která jsou využita. V ideálním případě by všechny provázanosti měly mít tuto podobu.

Polymorphism

- **Pokud chování závisí na typu objektu (třídě), přiřaďte zodpovědnost za toto chování pomocí polymorfických metod třídě, na které toto chování závisí.**

- Použito např. v: State, Strategy, Command, Proxy, …

Pure Fabrication

- Jedná se o vytvoření **pouhého konstrukčního prvku**.
- Použije se v případě, kdy přiřazení odpovědnosti některé třídě představující objekt z problémové domény by narušilo soudržnost (cohesion) této třídy, zvýšilo její provázání (coupling) nebo porušilo jiné principy.
- **Třídy představující pure fabrication nereprezentují nic, co by existovalo v rámci problémové domény.**
- Použito ve většině návrhových vzorů.

Indirection

- Jak přiřadit zodpovědnosti, pokud se potřebujeme vyhnout přímé vazbě?

- Přiřaďte zodpovědnost prostředníkovi, takže nebudou muset být prvky propojeny přímo.

- Většina problémů může být vyřešena přidáním další úrovně nepřímosti.

- Poznámka: Většina problémů s výkonem může být vyřešena ubráním nějaké úrovně nepřímosti. :-)

Information Expert

Zodpovědnost přidělte informačnímu expertovi – prvku, který má informace potřebné pro splnění této zodpovědnosti.

Information Expert

- Existují případy, kdy řešení **vyplývající z použití principu Informační expert jsou nevhodná**. Obvykle kvůli problémům s vysokou provázaností a nebo nízkou soudržností.

Creator

- Tvůrce se zabývá otázkou, komu přidělit zodpovědnost za vytváření instancí objektů. Jeho znění je:
    - Přiřaďte třídě B zodpovědnost za vytváření instancí třídy A, pokud platí jedno nebo více z následujících:
        - B je agregátor objektů A
        - B obsahuje objekty A
        - B uchovává záznamy o objektech A
        - B úzce spolupracuje s objekty A
        - B má inicializační data pro A (B je informační expert pro inicializaci A)

Creator

- Dobrým kandidátem na vytváření objektů jsou třídy, které slouží jako kontejnery pro vytvářené objekty.

- Použití ve vzorech: Factory Method, Abstract Factory, Builder

Controller

- Kdo by měl být zodpovědný za zpracování systémových událostí?

- **Systémová událost (system event)** je událost, která je generována nějakým vnějším aktérem.

- **Systémové operace (system operation)** jsou pak ty činnosti systému, které probíhají jako reakce na systémové události.

- Součástí našeho systému přijímající systémové události je jeho uživatelské rozhraní. Tato součást by ale **neměla být místem, kde bude probíhat zpracování těchto událostí**.

Controller

- Podle principu controller bychom měli vytvořit speciální třídu nebo třídy, které budou za **zpracování systémových událostí zodpovědné**.

- Controller není součástí ani uživatelského rozhraní ani doménové vrstvy.

- Jde o objekty představující Pure fabrication vložené mezi tyto dva subsystémy.

- V podstatě tvoří rozhraní systému z pohledu uživatelského rozhraní (Facade).

Controller

- Uživatelské rozhraní pak při zachycení systémové události pouze volá metody controlleru, který zařídí jejich zpracování.

- Obvykle zpracování neprovádí sám, ale deleguje jej na další součásti systému v doménové vrstvě.

- **Controller tedy zpracování pouze deleguje a koordinuje.**

Literatura

- Slajdy vytvořeny dle seriálu: http://www.zdrojak.cz/serialy/principy-objektove-orientovaneho-navrhu/

- Robert C. Martin: Čistý kód, Computer Press a.s., 2009, ISBN-978–80–251–2285–3