54. OBJEKTOVĚ ORIENTOVANÝ NÁVRH

- jedná se o návrh SW či systému či architektury realizovaný s využitím objektové orientace a s ní spojených paradigmat
- funkcionální paradigma – základním stavebním kamenem jsou funkce, jejich abstrakce, aplikace, definice a volání
  - základem může být λ-kalkul a je to třeba Haskell nebo Lisp
- logické paradigma – základem jsou nějaké formule a fakta dané logiky, například Hornovy klauzule predikátové logiky 1. řádu a jazyk Prolog
- procedurální paradigma – základem jsou procedury a sekvenční vykonávání kódu včetně větvení, cyklů atd... – třeba jazyk C
- objektově orientované paradigma – základním stavebním kamenem je objekt, které zapouzdřují nějaká data, operace nad nimi a celkově nějaký funkční celek nebo část
  - objekty patří (jsou instancemi) do tříd kde může být definována také dědičnost tříd / objektů
  - objekty mají mezi sebou různé relace, závislosti a komunikují spolu a kooperují

(– objekty mohou být logicky členěny do modulů, komponent a balíčků)

- principy objektové orientace
  - objekt – prvky modelované reality, poskytují funkčnost (metody) a zapouzdřují data
    - reprezentují (abstrahují objekty reálného světa...)
  - abstrakce – objekty jsou jako black box a přes rozhraní komunikují s okolím, které nemusí znát jejich vnitřní realizaci
  - zapouzdření – objekty zapouzdřují data ke kterým nelze přistupovat přímo (způsobilo by nekonzistenci) ale poskytují rozhraní
  - kompozice – objekty se mohou skládat z jiných objektů
  - dědičnost – objekty dědí strukturu a vlastnosti a chování jiných objektů
    - nejčastěji dědičnost tříd, třeba i jejich instancí
    - strom dědičnosti
  - polymorfismus – mnohotvárnost
    - několik objektů poskytuje stejné rozhraní a pracuje se s nimi na venek stejně ale jejich konkrétní chování a to co vrací závisí na jejich implementaci
    - rozhraní zvíře a metoda udělej zvuk v implem. pes štěkne ale ptáček pípne

OO návrh

- na vstupu jsou diagramy případů použití (Use-Case) a další případy užití a doplňující specifikace a detaily případů užití a nějaký slovník pojmů a diagramy domény (nad jakými doménami bude SW stát) a úkolem OO návrhu je nalézt vhodné rozložení do tříd jejichž instance (objekty) budou zapouzdřovat data a chování a komunikovat spolu a ⇒ realizovat tak jednotlivé případy použití

Scanned with CamScanner

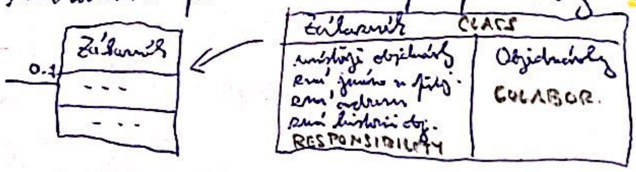
– kritická je schopnost umět navrhovat a aplikovat principy OO a OO paradigma
– jednotlivé případy použití které vznikly z fáze získání požadavků a jejich analýzu budou nyní reprezentovány objekty a třídami (ať už jedním, nebo skupinou spolupracujících)
– výstupem je diagram tříd, komponent, objektů, sekvenční diagram, diagram komunikace (spolupráce) a další v UML

RDD (Responsibility driven design – návrh řízený zodpovědností)

– návrh má nějaké cíle, kterých musí dosáhnout – cíle jsou například na základě případů užití a dalších požadavků na systém
– RDD rozkládá zodpovědnosti do jednotlivých tříd a objektů a vytváří tedy spolupracující množiny objektů (tříd) které mají nějaké zodpovědnosti, komunikují spolu a kooperují se účelem dosažení cílů návrhu
– rozdělení zodpovědnosti mezi objekty (třídy), či skupiny objektů
– zodpovědnost může být nějaká komplexní role nebo podúloha
– využívají se principy a doporučení GRASP – říká komu přiřadit jaké zodpovědnosti
– postup RDD:
  • vstupem jsou případy použití
  • vytvoří se doménový model
  • identifikují se zodpovědnosti vycházející z požadavků na SW a případů užití
  • zodpovědnosti se rozdělí mezi domény

Zodpovědnost za činnost
  – něco provádí, deleguje, koordinuje jiné prvky (objekty/třídy, ...)
Zodpovědnost za vědomost
  – zapouzdřuje a uchovává data (vědomosti)
  – zapouzdřuje objekty atd...

– CRC štítky tříd – udávají třídám zodpovědnosti a spolupracovníky = Class/Responsibility/Collaborator (identifikace, zodpovědnost, spolupracovníci)



GRASP

– doporučení, principy a pravidla k vytvoření kvalitnějšího OO návrhu
– přiřazují/rozdělují zodpovědnosti objektům (třídám)
– výstupem bývá diagram spolupráce (=komunikace) v UML

1. Informační expert – za akci nebo událost atd. má zodpovídat ten objekt, který o tom má nejvíce informací
2. Creator (tvůrce) – nové objekty se v OO tvoří velmi často – vytvářet by je měl objekt který ten vytvářený objekt obsahuje, skládá se z něj, má všechny informace potřebné k vytvoření, nebo velmi úzce s tím objektem bude spolupracovat
3. Controller (řadič) – přijímá a reaguje na události které v systému nastanou (myš. akce ...)
   – a deleguje událost jiným objektům které na ní reagují – zpracují data atd..

Scanned with CamScanner

4. **Low coupling (nízká závislost)** – nízká závislost mezi třídami, ideálně aby byly třídy co nejméně závislé mezi sebou aby byla snadná jejich znovupoužitelnost bez nutnosti dalších změn
- změny v jedné třídě by neměly mít velký vliv na jiné třídy
- třídy můžeme ze systému vyjmout a použít jinde => znovupoužitelnost

5. **High Cohesion (vysoká koheze – soudržnost)** – každá třída do sebe soustředí logicky související data a k nim metody, je pochopitelná a potřebná
- soustředění závislostí na jednom místě
- srozumitelnost, lepší udržovatelnost
- vedlejším produktem je Low Coupling – protože se logické části které spolu souvisí soustředí do jedné třídy – když chceme tak je povinnost jejich soudržnost tak nízká závislost (Low coupling)

6. **Polymorfismus** – chování se mění na základě konkrétní implementace

7. **Protected variations (chráněné variace)** – prvky systému jsou chráněné před jinými prvky systému protože pouze pracují s jejich rozhraním a změny v konkrétních implementacích jim nevadí

8. **Indirection – nepřímost** – snižuje závislost mezi dvěma třídami tak že mezi ně vloží prostředníka
- viz MVC, C prostředník

SOLID

S – single responsibility – třída by měla odpovídat pouze za jednu věc a ta odpovědnost by měla být tou třídou pokryta kompletně

O – open/closed – uzavřenost změnám ale otevřenost rozšíření – lze rozšiřovat funkčnost aniž by se sahalo do kódu – hookování, nové entity, dědičnost

L – liskov substitution (princip Liskové) – potomek je rozšířením rodiče a může jej zastupovat, tam kde se čeká ~~potomek~~ rodič může přijít ~~rodič~~ potomek – potomek musí mít stejné chování a vlastnosti jako rodič a může je rozšiřovat

I – interface segregation – rozbíjet rozhraní na co nejmenší aby třídy nemusely zbytečně implementovat rozhraní – metody co nepotřebují

D – dependency inversion – vyšší moduly by neměly záviset na nižších ale pouze na jejich abstrakcích
- závislosti by měly vést jedním směrem, a to od konkrétnějšího k abstraktnějšímu
- závislosti by měly být na rozhraních a abstraktních třídách ne na konkrétní implementaci

Návrhové vzory

(využívají se při OO programování a měl by na ně být brán ohled při OO návrhu)

vytvářecí:
- singleton – jenom jeden objekt dané třídy (instance) a to s globálním přístupem
  - statická metoda třídy vrátí singleton
- factory (továrna) – třída odpovídající za vytváření jiných objektů

strukturální:
- adaptér – adaptuje nekompatibilní rozhraní postarších objektů
- Facade (fasáda) – zastřešuje nějakou skupinu rozhraní a implementací – obalí je (skrytě podsystém) a dá jednotné rozhraní

chování:
- observer/listener – naslouchá událostem a reaguje na ně
  - asynchronní – třeba GUI
  - registruje se observer k nějakým událostem na které potom reaguje