03. Objektově orientovaný návrh software, metodiky objektového návrhu, postupy rozvinutí technické specifikace na úrovni modulů do detailního objektového návrhu, návrhové vzory

1. Fáze návrhu

- o fázi návrhu (kdy je prováděn, jaký je jeho účel)
- návaznost návrhu na analýzu (upřesňování, rozšiřování)
- návrh (architektura) musí odpovídat funkčním a nefunkčním požadavkům (škálovatelnost, bezpečnost, stabilita, výkon, dostupnost, snadná údržba)
- návrhové UML diagramy, technická specifikace, revize návrhu

Postup:

1. funkční požadavky
2. use case, zpřesňují use case
3. identifikace kandidátů na analytické třídy za použití analytických vzorů
4. analýza interakcí mezi těmito analytickými třídami
5. interakčními diagramy (sekvenční, diagramy aktivit, stavové diagramy)
6. analytických tříd se upřesňují a rozšiřují na třídy návrhové (průchodnost relací, násobnost, viditelnost atributů, datové typy atributů, nové metody a atributy)
7. refaktorizace návrhu za použití návrhových vzorů GoF, GRASP a SOLID
8. implementační třídy

2. Koncepty OOP

- přístup OOP je mnohem **bližší našemu chápání světa**
- **snižuje komplexitu** problémů díky dekompozici
- umožňuje tvorbu znovupoužitelného **komponentového** kódu
- umožňuje **rozdělovat zodpovědnosti do menších celků** a omezovat globálních proměnných zavedením viditelnosti (private, public, protected)
- široká podpora v OO jazycích (Smalltalk) a hybridních jazycích (C#, Java, C++)

- Dědičnost
- Polymorfismus
- Delegace
- Zapouzdření
- Abstrakce
- Kompozice
- Objekty

3. Různé metodiky

- návrh by měl probíhat iterativně
- fáze návrhu vypadá v jednotlivých metodikách různě:

- v agilních metodikách je návrh neustále přítomný během implementace a často dochází k refaktorizaci
- v klasických metodikách jako RUP nebo waterfall probíhá návrh samostatně

- metodika předepisuje **CO, KDO, KDY a JAK** má dělat, proto ji můžeme chápat jako nástroj pro zvládnutí komplexních problémů a rozsáhlých projektů
- **MDD** (model driven development) – vývoj řízený modely, velká míra podrobností v modelu, lze generovat kód z modelů
- důležitá je podpora CASE nástrojů

4. Architektonické vzory

- logická a fyzická architektura (komponent diagrams, deployment diagrams)
- layers, klient-server, request/reply, publisher/subscriber, messaging, file exchange, shared database, RMI, SOA, pipes & filters, proxy, blackboard, broker

5. Deméterovo pravidlo

= doporučení pro OOD, které respektuje **princip minimální znalosti o zbytku systému**
- respektuje pravidlo low coupling
- objekt by měl činit jen minimální předpoklady o struktuře a chování ostatních objektů
- každý objekt komunikuje jen se svým nejbližšími sousedy, kde sousednost dvou objektů znamená schopnost jednoho objektu volat metody objektu druhého
- slouží k zachování znovu-použitelnosti
- metoda M objektu O může volat metody jen a pouze těchto objektů:
  - objektu O
  - parametrů metody M
  - objektů vytvořených v metodě M
  - objektů obsažených v objektu O

6. Postupy rozvinutí technické specifikace na úrovni modulů do detailního objektového návrhu

- GRASP, SOLID, Top-down decomposition, Bottom-up konstrukce SW (znovu-použitelné komponenty)

7. GRASP

= general responsibility assignment software patterns
- je sada doporučení, principů a vodítek, sloužících k vytvoření kvalitnějšího objektového návrhu
- zaměřuje se na **rozdělení zodpovědností** jednotlivým třídám a objektům
- výstupem jsou nejčastěji diagramy spolupráce v jazyce UML
  - **Information Expert** rozdělení zodpovědnosti na základě přístupu k informací
  - **Creator** – za vytvoření instance A je zodpovědná třída, která obsahuje třídu A, se skládá z třídy A, obsahuje všechny informace nutné k vytvoření třídy A, velmi úzce spolupracuje s třídou A
  - **Controller** realizace uživatelské akce, zpracování události
  - **Low Coupling** nízká závislost mezi třídami, žádný nebo minimální dopad změny jedné třídy na třídy ostatní, vysoký potenciál znovupoužití

- **High Cohesion** funkce každé třídy je snadno pochopitelná a popsatelná, každá třída do sebe soustředí data a s nimi související funkce, existence každé třídy je objektivně odůvodnitelná
- **Polymorphism** odlišné chování podle konkrétního typu objektu
- **Protected Variations** identifikace a stabilizace společného rozhraní několika tříd
- **Pure Fabrication** vylepšuje systémovou architekturu (zmenšení závislostí), pure fabrication třída nesouvisí s řešenou doménou
- **Indirection** komunikace mezi dvěma třídami zajištěná třetím prostředníkem kvůli snížení závislostí a zvýšení i znovupoužitelnosti
- **Information expert-** třídou zodpovědnou za nějakou akci má být třída, která má k provedení této akce nejvíce informací, třída by měla se svými daty pracovat sama

8. Návrhové vzory GoF

- vztah ke GRASP vzorům
- creational/structural/behavioral
- návrhový vzor je určitý obecný návrh, jak psát kód řešící danou část problematiky
- založené na OOP
- není to hotová knihovna nebo implementace v konkrétním programovacím jazyce, ale spíš obecný algoritmický zápis (i pomocí UML diagramů)

GoF je katalog 23 základních návrhových vzorů rozdělených do 3 kategorií:

Creational Patterns

- řeší otázky vytváření a předávání referencí objektů v systému
- upřednostňují flexibilní objektovou kompozici namísto „pevné“ dědičnosti
- příklady: Singleton, Factory, Builder, Prototype, Fly Weight, Library, Enum, Pool

Structural Patterns

- popisují, jak jsou třídy a objekty složeny do větších struktur
- příklady: Adapter, Decorator, Proxy, Facade, Composite, Přepravka

Behavioral Patterns (chování)

- věnují se komunikací mezi objekty a jejich interakcemi
-příklady: Observer, Command, Iterator, Strategy, Template Method, State, MVC