
# Úvod do návrhových vzorov

Návrhové vzory

# Konceptuálne modelovanie

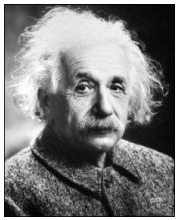
- Aký je rozdiel medzi **analýzou** a **návrhom**?
- Tieto činnosti často skončia rovnako, preto ich mnoho ľudí stotožňuje.
- Príklad na analýze simulátora hry snooker (pomocou prípadu použitia):
  - Hráč trafí tágom bielu guľu, ktorá sa následne pohybuje určitým smerom a rýchlosťou, narazí pod určitým uhlom do červenej gule, a tá sa potom tiež pohybuje určitým smerom a rýchlosťou.
- Ak by sme iba modelovali konkrétne situácie v systéme (ak sú gule rozmiestnené takto a trafím bielu takto, tak výsledok bude takýto), nikdy by ho nebolo možné modelovať úplne.

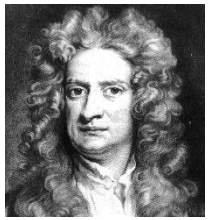
# Konceptuálne modelovanie

- Pre pochopenie systému je potrebné pozrieť sa hlbšie a odhaliť vzťahy medzi pohybom a rýchlosťou, hmotnosťou, hybnosťou, atď. – po pochopení týchto zákonitostí je vytvorenie simulátora jednoduchšie.
- Vytvorenie simulátora snookeru je jednoduché, lebo fyzikálne zákony sú nám známe. Ak by sme chceli byť naozaj dôkladný v paralele pri tvorbe SW, tak by sme ich pri ich procese museli sami odhaliť.
- **Konceptuálne modelovanie** – myšlienkový model, ktorý zjednodušuje skúmaný systém a tým ho robí pochopiteľným.

# Konceptuálne modelovanie

- Pre samotné modelovanie pohybu je možné využiť hneď dva modely  pohybu - jeden úplne presný (zatiaľ nič lepšie nemáme), jeden pre naše potreby postačujúci, ale nepresný.





- Ktorý model použijete a prečo? S využitím relativity by bol model určite presnejší, ale je to potrebné?
- **Modely nie sú dobré alebo zlé, sú iba viac alebo menej užitočné!**
- Čím viac je model flexibilnejší, tým je komplexnejší a ťažší na spravovanie, pochopenie.

# Konceptuálne modelovanie

- **Nepridávajte zbytočne funkčnosť, ktorú nebudete potrebovať!**
- Využívajte čo možno najjednoduchšie modely (často to nebudú tie, ktorí Vám ako prvé napadnú) – ľahšie sa udržujú.
- Pozor na myslenie v konkrétnom programovacom jazyku.
- **Konceptuálne modely sú spojené s rozhraniami (interface), nie konkrétnymi implementáciami (triedami)!!!**

# História

- Christopher Alexander (profesor architektúry na Kalifornskej univerzite v Berkeley) publikoval niekoľko kníh o vzoroch v architektúre (tá, čo stavia budovy ☺), pričom za prototyp kníh venujúcich sa vzorom v softvéroch je považovaná práve jeho kniha [1].
- Jeho myšlienku využitia vzorov rozšírili Kent Beck a Ward Cunningham.
- Začiatkom 90. rokov sa konferencia OOPSLA, kde Bruce Anderson viedol workshop, ktorý sa zaoberal myšlienkou vytvoriť „príručku“ pre softvérových architektov. Pre C++ vydal takú knihu Jim Coplien [2].
- Títo (a iní) ľudia tvorili Hillside group – skupina, ktorá sa zaoberala problematikou vzorov.

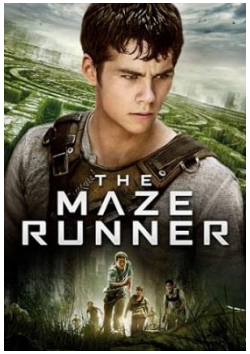
# História

- **Gang of Four** (GoF) – kniha [3] sa stala prelomová – vzory sa dostali do širokého povedomia.
- Takisto pomohla konferencia PLoP (Pattern Language of Programming), ktorú založila Hillside group v roku 1994. V tom istom roku bol založený Portland Pattern Repository.
- 1998 Martin **Fowler** vydal [4].

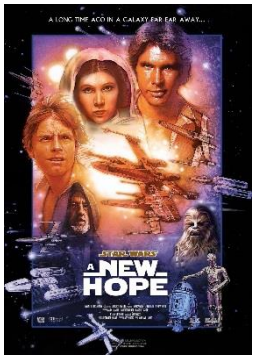
# Čo je to (návrhový) vzor?

- *„Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.“* Alexander
- *„The design patterns .. are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.“* GoF
- *„.. an idea that has been useful in one practical context and will probably be useful in others“.* Fowler

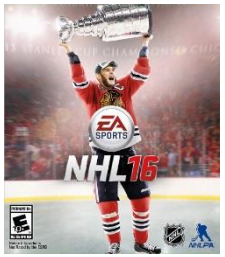
# Čo je to (návrhový) vzor?



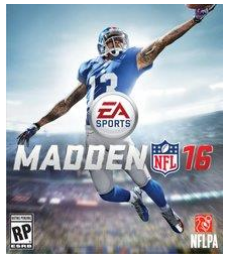


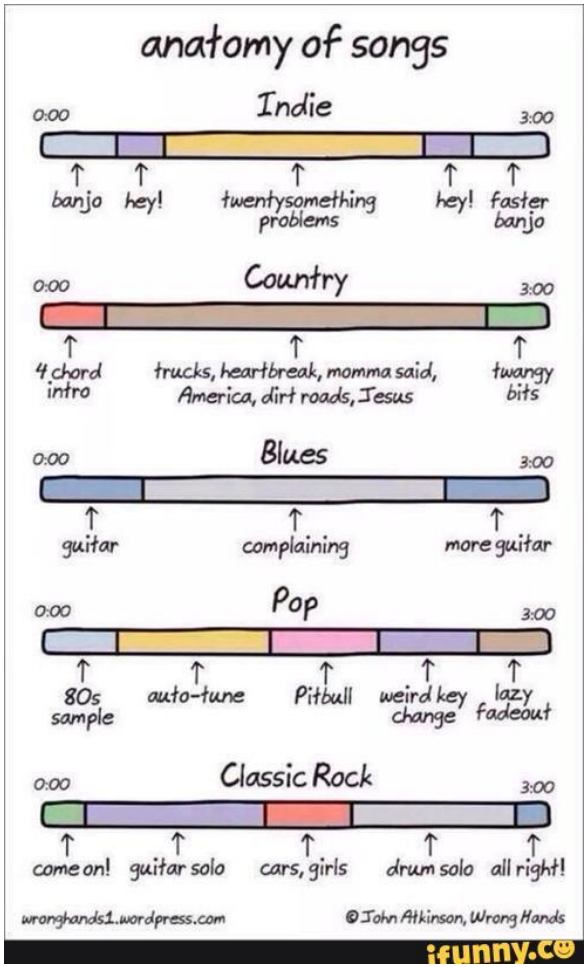








# Čo je to (návrhový) vzor?

- Návrhové vzory nie sú umelo vytvárané – vznikajú v praxi ako riešenia konkrétnych problémov. Ak je možné riešenie aplikovať aj na iné problémy, môžeme hovoriť o vzore.
- Fowler rozdelil vzory do dvoch kategórií:
  - **Analytické** (65 vzorov) – tvoria skupinu konceptov, ktorá reprezentuje všeobecnú koncepciu v business modelovaní.
  - **Podporné** (21 vzorov) – popisujú, ako aplikovať analytické vzory – návrhové vzory.
- **Návrhový vzor je počiatok riešenia problému, nie riešenie samotné.**

# Analytické vs. návrhové vzory

- Základné princípy ucelene prezentuje [5] (tu sú popísané základné koncepty).
- Analytické vzory sú z business domény. Pomáhajú správne pochopiť, a doplniť prípadné medzery v doménovej oblasti – kľúčová vlastnosť pre správny návrh systému.
- Autori v [6] mapujú niekoľko spôsobov ako aplikovať analytické a návrhové vzory a rozdeľujú ich na dva spôsoby:
  - Nesystematická aplikácia vzorov
    - Vzor aplikujem, ak vznikne nejaký problém – jednoúčelové riešenie situácie.
  - **Systematická aplikácia vzorov**
    - Existujú systematické techniky, ktoré definujú, ako vzory aplikovať.

# Systematická aplikácia vzorov

- Katalóg vzorov (Pattern Language)
  - Poskytuje vzory z konkrétnej oblasti.
  - Definuje vzťahy a súvislosti medzi vzormi.
  - Vzory sú väčšinou aplikované metódou *dosadenia do výsledného modelu*.
- Vývojový proces (Development Process)
  - Skladanie vzorov tvorí základ aplikačného modelu.
  - Definuje postup, modely, nástroje..

# Metóda dosadenia vzorov do výsledného modelu

1. Pomocou štandardných prostriedkov sa vytvorí model (analytický alebo návrhový)
2. Vyberú sa vhodní adepti vzorov (napr. podľa doménovej oblasti).
3. Následne sa vybrané adepti aplikujú na vhodné miesta v modeli – často dochádza k tomu, že vzor ponúkne navyše k požadovanej ešte dodatočnú funkčnosť (pozor na prvé pravidlo na tejto snímke!), čím sa zvýši flexibilita.
4. Nakoniec sa vyhľadajú účastníci vzoru, čím sa vzor vytvorí (vznikne jeho „inštancia“) a zasadí do aplikačného modelu.

# Pattern Oriented Analysis and Desing

- Metodika POAD bola detailne popísaná v [7], presne definuje predchádzajúci postup.
- Delí sa na tri etapy:
  1. **Analytická fáza** – na základe požiadaviek sa vyberú vhodné vzory pre konkrétnu modelovanú oblasť.
  2. **Návrh na vyššej úrovni** – inštancie vzorov, ktoré boli vybraní v predchádzajúcej fáze sa navzájom prepoja pomocou formalizovaného postupu.
  3. **Upresnenie návrhu** – rozvinie (doplní detaily, ktoré neboli pokryté) hrubý aplikačný model z predchádzajúcej fázy.

# Analytické vzory

- Fowler kategorizoval analytické vzory do 9 rôznych oblastí [4]:
  1. Accountability
  2. Observations and Measurements
  3. Observation for Corporate Finance
  4. Referring to Objects
  5. Inventory and Accounting
  6. Planning
  7. Trading
  8. Derivative Contracts
  9. Trading packages

# Časti návrhového vzoru

- Každý vzor by mal mať štyri základné časti:
  - **Meno** – popisný názov vzoru.
  - **Problém** – kedy je možné využiť vzor?
  - **Riešenie** – popisuje prvky, vzťahy medzi nimi, ich zodpovednosti a spoluprácu. Nepopisuje konkrétnu implementáciu, ale abstraktný popis!
  - **Dôsledky** – výhody a nevýhody využitia konkrétneho vzoru (môže klásť ďalšie požiadavky na systém, čo môže ovplyvniť jeho flexibilitu, rozšíriteľnosť alebo prenositeľnosť).
- Existuje mnoho členení– podľa Alexandra, Fowlera, GoF, Portlandská forma, Coplienova forma, forma POSA..

# Popis návrhového vzoru podľa GoF

- **Názov** a **klasifikácia** vzoru - (Pozri snímku vyššie).
- **Zámer** – na aký problém sa vzor zameriava.
- **Alias(y)** – aké iné názvy vzoru sú zaužívané.
- **Motivácia** – krátky prípad, ako vzor rieši daný problém.
- **Aplikovateľnosť** – aké problémy dokáže vzor riešiť.
- **Štruktúra** – UML diagram.
- **Účastníci** – aké objekty vystupujú vo vzore a aké majú úlohy.

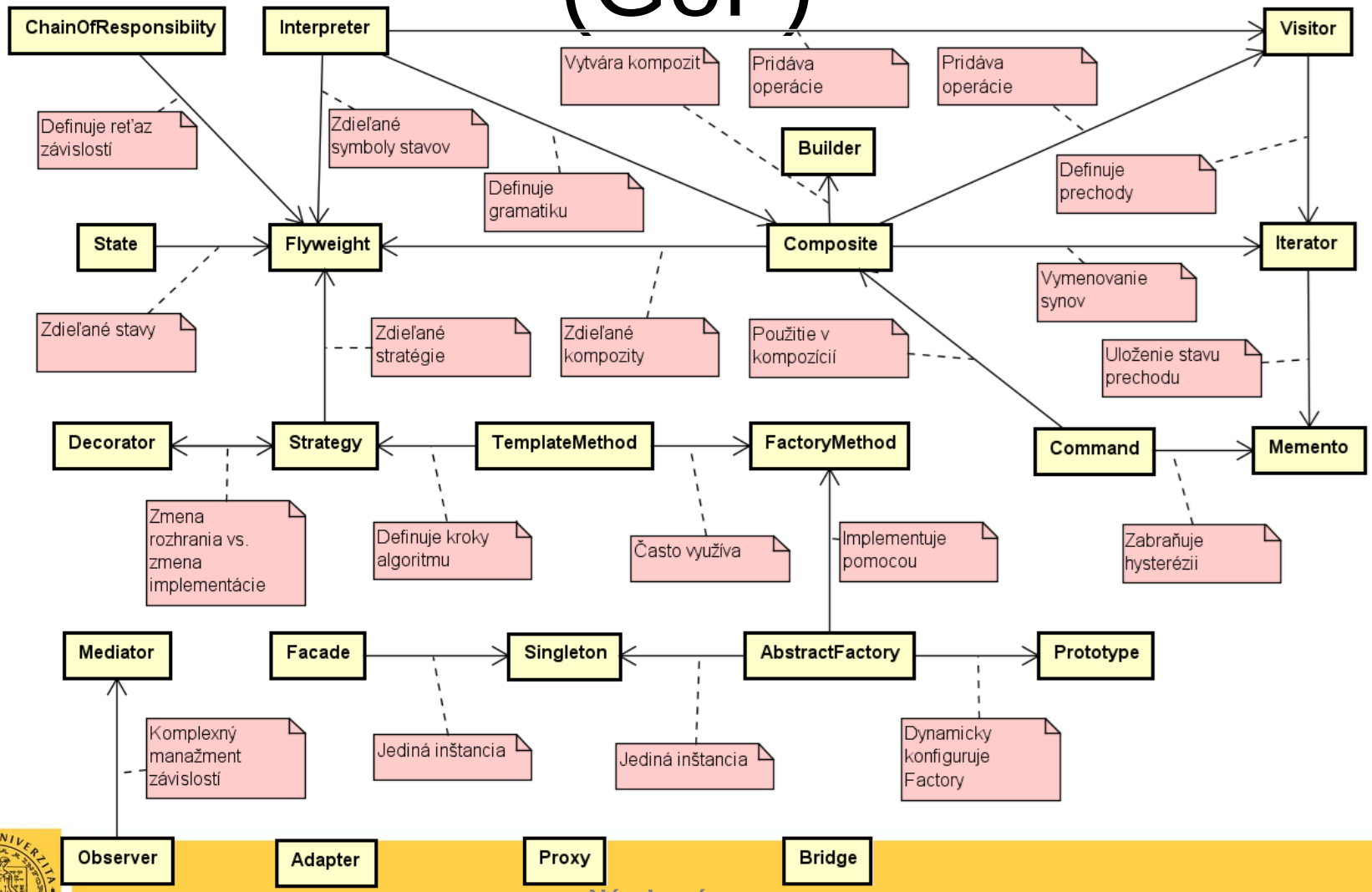
# Popis návrhového vzoru podľa GoF

- **Spolupráca** – Aké sú vzťahy a úlohy účastníkov.
- **Dôsledky** (Pozri snímku vyššie).
- **Implementácia** – na čo si treba dávať pozor, aké techniky sa dajú využiť.
- **Ukážkový kód** – C++, Smalltalk.
- **Prípady použitia** – príklady z iných domén (ako z domény aplikácie).
- **Príbuzné vzory** – vzory, ktoré sú späté s definovaným vzorom (s ktorými by sa mal používať), popis rozdielov.

# Návrhové vzory zavedené GoF

| Oblasť | | Účel: Tvorba objektov Creational | Účel: Štrukturálne Structural | Účel: Vzory správania Behavioral |
|---|---|---|---|---|
| Oblasť | Trieda | Factory method | Adapter | Interpreter<br>Template method |
| Oblasť | Inštancia | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Adapter<br>Bridge<br>Composite<br>Decorator<br>Facade<br>Flyweight<br>Proxy | Chain of responsibility<br>Command<br>Iterator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

# Spolupráca návrhových vzorov (GoF)



# Časté dôvody redesignu aplikácie (GoF)

1. **Vytvorenie objektov s priamym uvedením triedy**
   - `Auto a = new Auto()`
   - **Nie je** to programovanie voči rozhraniu – ak sa bude auto do budúcna vytvárať špeciálnejšie, bude treba upraviť väčšie množstvo kódu.
   - *Abstract factory*, *Factory method*, *Prototype*.
2. **Závislosť na konkrétnych správach**
   - Niekedy nemusí byť vhodné spoliehať sa na to, že správy sú implementované vždy konkrétnymi objektami – čo ak sa kompetencia reakcie na správu presunie?
   - *Chain of responsibility*, *Command*.

# Časté dôvody redesignu aplikácie (GoF)

3. **Závislosť na hardvérovej alebo softvérovej platforme**
   - Portovanie aplikácie môže byť ťažké, ak je navrhnutá na jednu konkrétnu platformu.
   - *Abstract factory*, *Bridge*.
4. **Závislosť na konkrétnej implementácii objektov**
   - Ak sa objekty spoliehajú na vnútorné organizovanie objektov, ktorí využíva, môže ich zmena spôsobiť zmenu závislých objektov.
   - `FRI.KI.varga.doSomething();` – čo ak zmením katedru?
   - *Abstract factory*, *Bridge*, *Memento*, *Proxy*.

# Časté dôvody redesignu aplikácie (GoF)

## 5. Závislosti na algoritmoch

- Zmena algoritmov (rozšírenie, optimalizácia, nahradenie) je veľmi častá.
- Ak sa objekt spolieha na konkrétnu podobu algoritmu, bude sa musieť tiež zmeniť.
- *Builder*, *Iterator*, *Strategy*, *Template method*, *Visitor*.

## 6. Úzke previazanie tried

- Ak majú triedy medzi sebou veľmi úzku závislosť, je ťažké použiť ich samostatne.
- Systém sa stáva jednoliatou ťažko udržateľnou, nepochopiteľnou masou.
- *Abstract factory*, *Bridge*, *Chain of responsibility*, *Command*, *Facade*, *Mediator*, *Observer*.

# Časté dôvody redesignu aplikácie (GoF)

7. **Rozširovanie funkčnosti tried dedičnosťou**
   - Pre správnu funkčnosť potomka musí byť dobre známa implementácia predka.
   - Môže sa stať, že predefinovanie jednej metódy má význam len ak sa predefinuje nejaká ďalšia metóda.
   - Pozor na kombinatorickú explóziu tried.
   - Kompozícia je často lepšie riešenie, ale nadmerné použitie zneprehľadní kód.
   - *Bridge*, *Chain of responsibility*, *Composite*, *Decorator*, *Observer*, *Strategy*.
8. **Nemožnosť upravovať triedy**
   - Zdrojové kódy nie sú k dispozícii alebo zásah spôsobí priveľmi zmien.
   - *Adapter*, *Decorator*, *Visitor*.

# Ako si zvoliť návrhový vzor (GoF)?

1. Zvážte, **ako** daný návrhový **vzor rieši problém**.
2. Pre zúženie výberu zvážte **zámer** návrhového vzoru.
3. Zvoľte vzor, ktorý vhodne **spolupracovať** s existujúcim návrhom.
4. Vzor musí byť určený na daný **účel**.
5. Aké by boli následky, ak by bolo potrebné **redesignovať** aplikáciu?
6. Vyberte prvky, od ktorých považujete, aby boli **variabilné**.

- **Vždy použite zdravý rozum!**

# Variabilné prvky „creational“ vzorov

| Návrhový vzor | Čo je možné efektívne upraviť? |
|---|---|
| *Abstract factory* | „rodiny“ produktov |
| *Builder* | spôsob vytvorenie kompozitu |
| *Factory method* | potomka triedy, ktorú metóda vytvára |
| *Prototype* | triedu vytváraného objektu |
| *Singleton* | inštanciu samotnej triedy |

# Variabilné prvky „structural“ vzorov

| Návrhový vzor | Čo je možné efektívne upraviť? |
|---|---|
| *Adapter* | rozhranie objektu |
| *Bridge* | implementáciu objektu |
| *Composite* | vnútornú štruktúru objektu |
| *Decorator* | schopnosti objektu bez nutnosti dedenia |
| *Facade* | rozhranie voči systému |
| *Flyweight* | náklady spojené s veľkosťou objektov |
| *Proxy* | prístup/umiestnenie objektu |

# Variabilné prvky „behavioral“ vzorov

| Návrhový vzor | Čo je možné efektívne upraviť? |
|---|---|
| *Chain of responsibility* | objekt, ktorý dokáže splniť požiadavku |
| *Command* | kedy a ako sa vykoná požiadavka |
| *Interpreter* | gramatiku a interpretáciu jazyka |
| *Iterator* | akým spôsobom prejde množinu agregovaných prvkov |
| *Mediator* | ako a ktoré prvky navzájom interagujú |
| *Memento* | ktoré prvky a kedy budú uložené mimo objektu |
| *Observer* | počet navzájom závislých objektov; ako závislé objekty ostanú aktuálne |
| *State* | stavy objektu |
| *Strategy* | algoritmus |
| *Template method* | kroky algoritmu |
| *Visitor* | operácie, ktoré vie objekt vykonať bez nutnosti zásahu do jeho kódu |

# „Ďalšie“ návrhové vzory

- Neexistujú iba vzory od GoF – sú najznámejšie a objektovými puristami uznávané za tie „pravé“.
- Niekoľko ďalších vzorových koncepcií popísal Rudolf Pecinovský [7].
- Niektoré ďalšie vzory:

| | | Oblasť | | |
|---|---|---|---|---|
| | | **Tvorba objektov**<br>**Creational** | **Štrukturálne**<br>**Structural** | **Vzory správania**<br>**Behavioral** |
| **Účel** | **Trieda** | Dependency injection<br>Simple factory method | Twin | |
| | **Inštancia** | Multiton<br>Object pool<br>RAII | Delegation<br>Marker | Null object |

# Literatúra


[1] ALEXANDER, Christopher, et al.: *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure)*. 1977.

[2] COPLIEN, J.O.: *Advanced C++ Programming Styles and Idioms.* Reading, MA: Addison-Wesley, 1992.

[3] GAMMA, E., HELM R., JOHNSON, R., VLISSIDES , J.: *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995.

[4] FOWLER, Martin.: *Analysis patterns: reusable object models*. Addison-Wesley Professional, 1997.

[5] BUHNOVÁ, Barbora. *Analytické vzory*. [online] s.d. [cit. september. 2016]- Dostupné z http://www.fi.muni.cz/~buhnova/PV167/Analyticke-vzory.pdf

[6] YACOUB, S., AMMAR, H.: *Pattern-Oriented Analysis and Design – Composing Patterns to Design Software Systems*. USA, Boston: Addison- Wesley, 2003.

[7] PECINOVSKÝ, Rudolf: *Návrhové vzory – 33 vzorových postupů pro objektové programování*. Computer Press, 2007. ISBN 978-80-251-1582-4

# Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu 5II132 Návrhové vzory (Design Patterns) na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline.
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené.

Ing. Michal Varga, PhD.
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline
Michal.Varga@fri.uniza.sk