# 1 Uvod

- Temeljni problemi *software developementa*:
  1. Problem **složenosti** softvera: tehnološka i složenost domene problema
  2. Problem **mijenjajućih zahtjeva**
- Različite vrste domena problema:
  1. Fizičke (npr. Proizvodni proces)
  2. Neopipljive (npr. Bankarski sustav)
  3. Vezane uz sam razvoj softvera (npr. IDE)
- Osnovni principi *Domain-driven* dizajna:
  - Fokus na domeni i njenoj pripadajućoj logici
  - Složeni dizajni domene bi trebali biti zasnovani na **modelu** (apstrakcija!)
- Model domene (keypoints):
  1. Model i implementacija oblikuju jedno drugo
  2. Model predstavlja temelj jezika koji se koristi
  3. Model prestavlja distilirano znanje
- Pri izgradnji modela domene osnovna zadaća je **ekstrakcija informacija**:
  - Od eksperata domene
  - Od postojećih korisnika
  - Od programera sa iskustvom na sličnim projektima
- Osnovni problemi razvoja softvera:
  1. *Waterfall* metoda ne radi - nema povratne veze (feedback)
  2. Zahtjevi korisnika se mijenjaju
- Rješenje je ***Iterative developement***:
  - Svaka iteracija zadovoljava odredjeni podskup zahtjeva sustava
  - Svaka iteracija se sastoji od skupa aktivnosti: dizajn, implementacija, testiranje
  - *Feedback* od korisnika nakon svake iteracije
- Iterativnim pristupom izbjegavamo *"Rush to code"* i *"Analysis paralysis"*

- Ključni element iterativnog procesa - **Refactoring**:
  - Mijenjanje dizajna ovisno o promjeni znanja/zahtjeva od domeni
  - Cilj: Poboljšanje strukture koda i njegove razumljivosti, bez promjena u ponašanju

# 2 Crash course in OOP

- Problem kod proceduralne paradigme:
  - Razdvojenost struktura ("stanje") i procedura ("ponašanje")
  - Rješenje: objedinjavanje podataka i funkcija koje djeluju nad tim podatcima u **razred/klasu**
- Razred/klasa je predložak iz kojeg se kreiraju (instanciraju) objekti koji imaju istu strukturu i ponašanje kao i razred iz kojeg su instancirani
- Objedinjavanjem smo dobili mogućnost dostizanja dva fundamentalna principa razvoja softvera:
  1. **Apstrakcija**
     - Apstrakcija označava esencijalne karakteristike objekta koje ga razlikuju od svih drugih vrsta objekata i koje definiraju njegove ošte konceptualne granice
     - Suština OO dizajna je modeliranje pravog skupa apstrakcija (Model domene)
     - Sustav je izgradjen kao zajednica objekata koji medjusobnom suradnjom realiziraju traženu funkcionalnost
  2. **Enkapsulacija**
     - Realizacija principa *Information hiding*-a
     - Pakiranje operacija i atributa koji predstavljaju stanje u klasu, tako da je stanje dostupno ili promjenjivo samo preko sučelja definiranog enkapsulacijom
     - Uvodi se **pravo pristupa** za elemente razreda: *private, public, protected*
- Time postižemo potpuno razdvajanje razreda na dva dijela:
  1. **Sučelje** - definira vanjski pogled obuhvaćajući našu apstrakciju o ponašanju razreda
  2. **Implementacija** - mehanizam kojim se realizira traženo ponašanje
- **Hijerarhija objekata** predstavlja rangiranje apstrakcija

- Dvije najvažnije hijerarhije:
  1. Struktura razreda - IS-A-KIND-OF odnos koji vodi na naslijedjivanje
  2. Struktura objekata - PART-OF odnos koji vodi na agregaciju i kompoziciju
- **Naslijedjivanje**:
  - Izvedena klasa preuzima čitav skup odgovornosti bazne klase, i dodaje svoje specifične detalje
  - Omogućava **polimorfizam** - rad s objektima preko pokazivača na baznu klasu
- Dva bitna koncepta polimorfizma:
  1. **Apstraktni razredi**
     - Predstavlja "nedovršenu" klasu
     - Predvidjeni za daljnje naslijedjivanje i ne mogu se instancirati
  2. **Razredi sučelja**
     - Predstavlja sve ono što razred "daje" prema van, tj. ono na što se vanjski razredi mogu osloniti
     - Definira skup operacija koje izvedeni razredi moraju implementirati
- Kreiranje objekata:
  - Alokacija memorije:
    1. Smještanje objekta na stog
    2. Smještanje objekta na gomilu (*heap*)
       * `new` - operator za kreiranje objekta na heap-u
       * `delete` - operator za brisanje s heap-a
       * C#/Java - *Garbage collector*
  - Inicijalizacija objekta (**Konstruktor**):
    * Posebna članska funkcija namijenjena inicijalizaciji stanja objekta prilikom njegovog kreiranja

# 3 Osnovni principi OO dizajna

- Objektno-orijentirana **analiza**
  - Podrazumijeva ispitivanje zahtjeva iz perspektive razreda i objekata nadjenih u vokabularu domene problema
  - Odgovara na pitanje *što* treba napraviti
- Objektno-orijentirani **dizajn**
  - Podrazumijeva provodjenje OO dekompozicije uz notaciju za opisivanje logičkih, fizičkih, statičkih i dinamičkih modela sustava
  - Odgovara na pitanje *kako* treba napraviti sustav
- Faktori lošeg dizajna:
  - Krutost - jedna promjena povlači brojne druge
  - Krhkost - promjena uzrokuje pogreške u drugim, konceptualno nepovezanim djelovima
  - Nepokretnost - sistem se teško razdvaja u komponente
  - Nepotrebna složenost
  - Nepotrebno ponavljanje
- Različite domene razreda:
  1. **Fondacijska domena**
     - Razredi s najširom mogućom primjenom
     - Fundamentalni (Int, Char), Strukturni (List), Semantički (Date, Time)
  2. **Arhitekturna domena**
     - Ograničene na pojedinu računalnu arhitekturu
     - *Machine-communication* razredi, *Database-manipulation* razredi, *Human-interface* razredi
  3. **Poslovna domena**
     - Unutar pojedine industrije
     - *Atribute/Role/Relationship* razredi
  4. **Aplikacijska domena**
     - Koriste se samo unutar jedne aplikacije
     - *Event-recognizer* i *Event-manager* razredi
- **Reusability** - Svojstvo programskog koda da se nakon implementacije može ponovno iskoristiti na nekom drugom mjestu (aplikaciji, sustavu)

- ***Encumberance* (Opterećenje)**
  - Mjeri ukupnu potpornu mašineriju razreda - Obuhvaća sve druge razrede na koje se dani razred mora osloniti pri obavljanju svog zadatka
  - **Direktno opterećenje** razreda jednako je kardinalnosti njegovog *Direct class-reference* skupa
  - **Indirektno opterećenje** razreda jednako je kardinalnosti njegovog *Indirect class-reference* skupa
  - *Encumberance* nam daje mjeru sofisticiranosti razreda (koliko se razred nalazi iznad fondacijske domene)
- **Enkapsulacija**
  - Skrivanje nekih detalja iza dobro definirane granice
  - Postoje različiti nivoi enkapsulacije: *level-0* - niz instrukcija, *level-1* - procedura, *level-2* - OO enkapsulacija, *level-3* - komponente i moduli
  - Pojmovi mjere kvalitete enkapsulacije:
    1. Cohesion (kohezija) - mjera "usmjerenosti" skupa operacija i atributa unutar razreda u ostvarivanju svrhe razreda
    2. Coupling (sprega) - mjera broja i snage veza izmedju razreda
- ***Connascence* (Medjuzavisnost)**
  - Princip mjerenja kohezije i vezivanja
  - Vrste medjuzavisnosti:
    1. Medjuzavisnost imena
    2. Medjuzavisnost po konvenciji
    3. Medjuzavisnost po algoritmu
    4. Medjuzavisnost po vrijednosti
    5. Ostale: medjuzavisnost pozicije, izvršenja, *timing*-a, identiteta, po tipu
  - Klasifikacija medjuzavisnosti:
    1. Statička - leksička struktura koda
    2. Dinamička - ovisi o *pattern*-u izvršavanja koda
- **Demeterov zakon**
  - *"Don't talk to strangers. Talk only to your immediate friends."*
  - Objekt smije pričati samo s objektima na koje ima direktne reference

- ***Cohesion*** **(Kohezija)**
  - Mjera medjupovezanosti svojstava (atributa i operacija) u javnom sučelju razreda
  - Tri problema kohezije u pridjeljivanju odgovornosti razredima:
    1. ***Mixed-instance*** - Razred ima neka svojstva koja su nedefinirana za neke objekte
    2. ***Mixed-domain*** - Razred sadrži element koji direktno opterećuje razred s ekstrinsičnim razredom neke druge domene
    3. ***Mixed-role*** - Razred sadrži element koji direktno opterećuje razred s ekstrinsičnim razredom u istoj domeni
  - Što je bolje definiran skup odgovornosti razreda, to on ima bolju koheziju
- ***Single-Responsibility Principle*** **(SRP)**
  - Razred treba imati (modelirati) samo jednu odgovornost
- ***Open-Closed Principle*** **(OCP)**
  - Softverski entiteti trebaju biti otvoreni za proširenje, ali zatvoreni za modifikaciju
  - Rješenje pomoću sučelja i apstraktnih razreda

**Prostor stanja i ponašanje**

- **Prostor stanja**
  - Razred bi trebao predstavljati uniformnu apstrakciju svojstava individualnih objekata koji pripadaju tom razredu
  - Prostor stanja razreda je skup svih dozvoljenih stanja objekata tog razreda (Stanje objekta je skup svih dozvoljenih vrijednosti atributa tog objekta)
  - Dozvoljeno **ponašanje** razreda je skup svih prijelaza izmedju stanja u prostoru stanja tog razreda koje objekt smije napraviti
  - Svaki objekt zauzima jednu točku u prostoru stanja. Objekti u istoj točki su identični
  - **Dimenzionalnost** - Broj atributa u prostoru stanja
  - Podrazredi mogu proširiti stanje dodavanjem novih atributa
- **Invarijante razreda**
  - Uvjet koji svaki objekt nekog razreda mora zadovoljavati u svakom trenutku (tj. predstavljaju ograničenja na prostor stanja)

- Ograničenja na pojedine operacije:
  1. *Precondition* - Uvjet koji mora biti zadovoljen prije izvršavanja funkcije
  2. *Postcondition* - Uvjet koji mora biti zadovoljen nakon izvršavanja funkcije
- ***Design by Contract***
  - Ukoliko pošiljatelj poruke garantira da su preduvjeti ispunjeni, ciljna operacija garantira da će *postcondition* biti ispunjen; u suprotnom ciljna operacija ima pravo odbiti izvršavanja

**Nasljedjivanje II**

- ***Liskov Substitution Principle* (LSP)**
  - Ponašanje programa mora ostati nepromijenjeno kada umjesto objekta baznog razreda podemtnemo objekt izvedenog razreda
- ***Design by Contract II***
  - Redefinicija operacije (u izvedenom razredu) može zamijeniti *preconditions* samo s istim ili slabijijm, a *postconditions* samo s istim ili jačim
- **Suptilnosti nasljedjivanja s *overload, override, hide***
  - *Overload* (Preopterećenje) - Definiranje druge funkcije s istim imenom unutar istog dosega (*scope*), ali s različitim parametrima
  - *Override* (Prekrivanje) - Definiranje iste funkcije u izvedenom razredu
  - *Hide* (Skrivanje) - Definiranje funkcije s istim imenom u unutrašenjm dosegu
- ***Design patterns***
  - Predstavljaju rješenja za često ponavljajuće dizajnerske probleme
  - npr. ***Template method, Strategy design***

# 4 Proces razvoja

- **Proces razvoja** (metodologija)
  - Kako izgraditi traženi sistem polazeći od skupa zahtjeva korisnika
  - Osnovne aktivnosti procesa razvoja:
    * Skupljanje zahtjeva i njihovo grupiranje
    * Kreiranje logičke strukture sustava (dizajn)
    * Kreiranje fizičke strukture sustava (arhitektura)
    * Implementacija
    * Testiranje
    * Instalacija
- Vrste procesa:
  - Standardni *waterfall*
  - *Rational Unified Process*
  - *Agile* metodologije (npr. Extreme programming)
  - ICONIX
- Karakteristike modernog procesa:
  1. Iterativan i inkrementalan - U svakoj iteraciji (pod)skup zahtjeva
  2. Pokretan zahtjevima (*Use-case driven*) - Konstanta interakcija s korisnicima
- ***Timeboxing***
  - (Fiksirano trajanje) Svaka iteracija ima dobro isplaniran cilj
- ***Use cases***
  - Kako opisati/formalizirati zahtjeve (Analiza)
  - Predstavljaju "priče" o korištenju sistema za postizanje nekog cilja
  - **Use case** opisuje ponašanje sistema u različitim uvjetima kod reakcije sistema na zahtjev nekog *stakeholdera*
  - **Use case** predstavlja niz povezanih scenarija koji opisuju učesnikovo korištenje sistema radi postizanja nekog cilja
- Djelovi *Use casea*:
  1. *Stakeholderi* i lista interesa - Sugerira i ograničava što sistem mora raditi

2. Preduvjeti (govore što mora biti istinito prije početka odvijanja scenarija) i garancije uspjeha (što mora biti istina po završetku *Use casea*)
3. Glavni uspješni scenario (*"Happy path"*)
4. Ekstenzije (Alternativni scenariji) - Predstavljaju grananje glavnog scenarija
5. Specijalni zahtjevi - Nefunkcionali zahtjevi koji se odnose na *Use case* (Najčešće vezani uz performanse, pouzdanost itd.)

- Vrste *Use casea*:
  1. ***Black-box*** - Ne opisuju unutarnje detalje sustava (Preporučeno)
  2. ***White-box*** - Opisuju unutarnje detalje sustava
- Kod analize zahtjeva za računalnu aplikaciju, fokusirati se na nivou **elementarnih poslovnih procesa (EBP)** - *User goal-level Use case*
- *Use case* **dijagrami**
  - Prikazuje odnose izmedju učesnika i *Use caseova* u sistemu
  - Sekundarni kod rada s *Use caseovima*

# 5 Razrada zahtjeva

- Uz *Use case* treba razraditi:
  - *Screen layout*
  - Dokumentaciju o pravilima u domeni
  - Arhitekturna pitanja
- Objekti predstavljaju osnovu našeg **modela domene**
  - Cilj: Kreirati softversku realnost tako da najbolje zadovoljava zahtjeve na naš sistem
  - Prvo odredjujemo **koncepte** *(Conceptual/domain objects)*
- Pretakanje konceptualnih objekata u konkretne objekte prema dizajnu:
  - Postaju *computer-related*
  - Koncepti se dijele u više objekata, ili više koncepata implementira jedan objekt
- **CRC kartice** (*Candidates, Responsibilities, Collaborations*)
  - Prikazuju skup odgovornosti objekta (*knowing* i *doing*)
- Pronalazak objekata u domeni - ***Discovery strategy***
  - *"Finding good objects means identifying abstractions that are part of your application's domain and its execution machinery."*
- Odabir kandidata za objekte - ***Search strategy***
  1. Imenovanje kandidata
  2. Karakterijzacija kandidata (pomoću stereotipova):
     - Information holder - posjeduje i daje informacije
     - Structurer - održava odnose medju objektima
     - Service provider - pruža *computing services*
     - Coordinator - reagira na dogadjaje i delegira zadatke
     - Controller - donosi odluke i upravlja tudjim zadacima
     - Interfacer - transformira informacije izmedju različitih djelova sistema