1. UVOD

U ovom specijalističkom radu baviću se projektovanjem, razvojem i implementacijom baze podataka vezane za konkretan studijski primjer edukacije pravosudnih institucija, konkretno sudovi i tužilaštva u Republici Srpskoj.

U samom radu obradiću pored samog modela baze i koncepte ciklusa odnosno faze razvoja projekta, te samu implementaciju korisničkog interfejsa preko objektno orijentisanog programskog jezika C#.

Ovaj projekat možemo definisati kao sređen skup metoda, procesa i operacija za prikupljanje, čuvanje, obradu, prenošenje i distribuciju podataka u okviru jedne organizacije, uključujući i korisnike koji se tim aktivnostima bave. Ovakav sistem je, u stvari, model realnog sistema. On opisuje i mijenjanje realnog sistema. Cilj je prikupljanje i obezbjeđivanje relevantnih informacija iz realnog sistema i okruženja i transformacija istih u upravljačke informacije radi održavanje sistema, odnosno upravljanje samim sistemom.

2. ŽIVOTNI CIKLUS PROJEKTA

Neka od pitanja koja se najčešće postavljaju kod razvoja novog softvera su: Kako se trebaju alocirati odgovornosti na klase i objekte, kako bi objekti trebali sarađivati, šta bi klase trebale da rade, itd. Ovo su kritična pitanja koja se postavljaju kod projektovanja sistema, a do odgovora na ta pitanja se treba doći primjenom Objektno orijentisane analize i dizajna (OOA/D). S druge strane, na osnovu starih rješenja se mogu iskazati najbolje prakse ili paterni, te se kao takvi mogu koristiti kod budućeg razvoja softvera.

2.1 Larman metodologija

Upravo Larmanova metodologija treba da prikaže kako se na efikasan način mogu primjeniti znanja iz OOA/D i paterna.

U Larmanovoj metodologiji možemo razlikovati 4 faze:

Prva faza je **faza prikupljanja zahtjeva** u kojoj se definišu svojstva i uslovi koje sistem ili šire gledajući projekat mora da zadovolji. Glavni izazov kod analize zahtjeva je da se pronađe, poveže i zapamti (najčešće da se zapiše) šta je zaista potrebno u obliku koji je jasan klijentima i članovima razvojnog tima.

Svi zahtjevi se prema FURPS+ modelu mogu podijeliti u funkcionalne (**F**unctional), zahtjeve vezane za upotrebljivosti (**U**sability), pouzdanost (**R**eliability), performanse (**P**erformance) i podrživost (**S**upportability). Veoma česta podjela je i ona na funkcionalne i nefunkcionalne zahtjeve.Funkcionalni zahtjevi definišu zahtjevane funkcije sistema, dok nefunkcionalni definišu sve ostale.

U ovom radu ćemo se fokusirati na funkcionalne zahtjeve i na njihovo definisanje.

Funkcionalni zahtjevi se uglavnom predstavljaju preko slučajeva korišćenja. Slučajevi korišćenja predstavljaju tekstualnu priču koja se koristi za pronalaženje i pamćenje zahtjeva.

Za prikazivanje svih slučajeva korišćenja se koristi Model slučajeva korišćenja.On prestavlja skup svih slučajeva korišćenja, aktora i veza između aktora i slučajeva korišćenja.Aktor u tom kontekstu predstavlja nešto što ima ponašanje,bez obzira da li je to osoba, kompjuter ili organizacija. Model slučajeva korišćenja može opciono da uključuje i UML case dijagrame za prikazivanje imena slučajeva korišćenja i aktora i njihovih

veza. Na ovaj način može se dobiti koristan kontekstni dijagram sistema i njegovog okruženja.Ipak, moramo istaći da su slučajevi korišćenja prvenstveno tekstualni dokumenti, a ne dijagrami, te je modelovanje slučajeva korišćenja prvenstveno čin pisanja teksta,ne crtanja dijagrama.

Scenario je specifična sekvenca akcija i interakcija između aktora i sistema.Takođe se naziva i instanca slučajeva korišćenja. To je jedna konkretna priča o korišćenju sistema ili jedan put kroz slučaj korišćenja. Na osnovu ovoga možemo reći da slučaj korišćenja predstavlja kolekciju srodnih uspješnih i neuspješnih scenarija koji opisuju korišćenje sistema od strane aktora da bi ostvario svoj cilj.

Kao rezultat ove faze imaćemo u najmanju ruku prikazan model slučajeva korišćenja, pri čemu će za svaki slučaj korišćenja biti prikazan naziv slučaja korišćenja,svi aktori i učesnici,preduslovi i post uslovi, osnovni scenario, kao i jedan ili više alternativnih scenarija.

Druga faza je **faza analize**. Faza analize opisuje logičku strukturu i ponašanje softverskog sistema, pri čemu se ponašanje opisuje preko sistemskih dijagrama sekvenci koji se prave za svaki slučaj korišćenja koji smo identifikovali u prethodnoj fazi.

Sistemski dijagrami sekvenci ilustruju ulazne i izlazne događaje koji su u vezi sa sistemom koji se posmatra.Tekstualni opis slučajeva korišćenja i sistemski događaji koji se na osnovu njih mogu identifikovati predstavljaju osnovu za sistemske dijagrame sekvenci.Naime, slučajevi korišćenja opisuju kako spoljni aktori vrše interakciju sa sistemom. U toku ove interakcije, aktor generiše sistemske događaje za sistem, uglavnom zahtjevajući da određene sistemske operacije obrade događaj. UML uključuje dijagrame sekvenci pomoću kojih se mogu prikazati interakcije aktora i operacije koje se usljed toga iniciraju. Sistemski dijagram sekvenci je slika koja prikazuje, za jedan konkretni scenario slučaja korišćenja, događaje koje eksterni aktor generiše, njihov redosljed i međusistemske događaje. Svi sistemi se tretiraju kao crna kutija. Akcenat na ovim dijagramima je na događajima koji prelaze granice sistema od aktora ka sistemu. Inače, postoje tri tipa događaja na koje sistem mora odgovoriti:Eksterni događaj od strane aktora, vremenski generisani događaji i greške.Može se lako vidjeti da upravo eksterni događaji predstavljaju sistemske događaje, te je njihovo predstavljanje veoma važan dio analize ponašanja sistema.

Domenski model je najvažniji model u objektno orijentisanoj analizi. Domenski objekti su vizuelna reprezentacija konceptualnih klasa objekata i stvarnih situacija u domenu. Domenski model je takođe poznat kao konceptualni model, model domenskih objekata ili model objekata analize.Moramo naglasiti da je domenski model reprezentacija konceptualnih,a ne softverskih klasa. U skladu sa tim, možemo reći da je domenski model ilustrovan preko dijagrama klasa u kojima nisu definisane operacije. Ovi dijagrami pružaju konceptualnu perspektivu i prikazuju:

- Domenske objekte ili koncepualne klase
- Asocijacije između konceptualnih klasa
- Atribute konceptualnih klasa

Konceptualne klase predstavljaju atribute softverskog sistema, te stoga opisuju strukturu softverskog sistema.Konceptualne klase sastoje se od atributa, koji opisuju osobine klase.

Asocijacija je veza između konceptualnih klasa.Svaki kraj asocijacije predstavlja ulogu koncepta koji učestvuje u asocijaciji. Uloga sadrži ime, preslikavanje i navigaciju.

Atributi predstavljaju osobine koje se pridružuju konceptualnim klasama. Svaki atribut je vezan za jedan određeni tip podatka.

Slučajevi korišćenja su osnovni alati za opisivanje ponašanja sistema. Međutim, ponekad je potreban detaljniji opis ponašanja sistema, a u tu svrhu se koriste ugovori. Ugovori koriste formu u obliku preduslova i post uslova da detaljno opišu promjene nad objektima u domenskom modelu kao rezultat izvršenja sistemskih operacija. Glavni ulaz za ugovore predstavljaju sistemski dijagrami sekvence i domenski model.Zbog toga se takođe ugovori mogu koristiti kod projektovanja objekata, budući da prikazuju promjene koje se zahtjevaju u softverskim objektima ili bazi.

Oblik u kome se mogu prikazati ugovori je sljedeći:

- Operacija- ime operacije i njeni parametri
- Veze sa slučajevima korišćenja- slučajevi korišćenja u kojima se može ova operacija pojaviti
- Preduslovi – pretpostavke o stanju sistema ili objekata u domenskom modelu prije izvršenja operacije
- Post uslovi – stanje objekata u domenskom modelu nakon kompletiranja operacija

Na osnovu ranije izloženog, možemo da zakljucimo da kao rezultat ove faze dobijamo logičku strukturu i ponašanje softverskog sistema. Ponašanje je predstavljeno preko sistemskih operacija i ugovora vezanih za njih, dok je struktura opisana preko domenskog modela.

Faza projektovanja je treća faza u okviru ove metodologije.

U ovoj fazi se opisuje fizička struktura i ponašanje softverskog sistema.Ako bi se moglo reći da smo se u fazi prikupljanja zahtjeva i fazi analize nastojali da „odradimo prave stvari“, onda bismo za fazu projektovanja mogli reći da je glavni naglasak na tome da se „stvari odrade kako treba“. Možemo reći da je glavni zadatak u ovoj fazi da se izvrši adekvatno projektovanje arhitekture softverskog sistema. Prije nego nastavimo sa definisanjem softverske arhitekture moramo da se upoznamo sa konceptom logičke arhitekture.Logička arhitektura predstavlja organizaciju softverskih klasa u pakete, podsisteme i nivoe, pri čemu pod nivoom možemo smatrati grupu klasa, pakete ili podsistema sa jasno razgraničenim zaduženjima. U većini sistema nivoi su tako organizovani da viši nivoi pozivaju usluge nižiš nivoa. Tipični nivoi u objektno orijentisanim sistemima su:

- Korisnički interfejs
- Aplikaciona logika i domenski objekti koji predstavljaju softversku reprezentaciju domenskih koncepata
- Tehnički servisi – objekti i podsistemi opšte namjene koji pružaju tehničku podršku kao što je recimo komunikacija sa bazama podataka.Ovi servisi uglavnom ne zavise od aplikacija i mogu se koristiti na više različitih sistema.

Sada možemo da definišemo softversku arhitekturu kao skup odluka o organizaciji softverskog sistema i strukturnih elemenata i interfejsa od kojih je sistem sastavljen, ali i o ponašanju tih elemenata što se vrši kroz specifikaciju saradnje između elemenata i kompoziciju strukturnih i elemenata ponašanja u podsisteme. Možemo vidjeti da softverska arhitektura predstavlja konkretni prikaz logičke arhitekture sistema i načina na koji elementi sistema međusobno komuniciraju.

Za specifikaciju strukture i ponašanja softverskog sistema najčešće se koriste statični i dinamični modeli UML-a.

Najčešći statični UML dijagram je „Class diagram“ koji se koriste za ilustraciju klasa, interfejsa i njihovih asocijacija. U ovoj fazi se, na osnovu domenskih klasa, definišu softverske klase koje, za razlike od domenskih klasa, sadrže i potpise metoda. Kao što smo već istakli, najčešći način za njihovo prikazivanje je korišćenjem UML dijagrama klasa.