PROJEKTOVANJE I IMPLEMENTACIJA SOFTVERA ZA PODRŠKU I PRAĆENJE RADA ŠAHOVSKIH KLUBOVA

Diplomski rad

Prezentacija skraćene verzije

Miloš Kecman
36/01-02 Profesor informatike

Зрењанин, 2007.

1. Uvod

- Projektovanje softvera je daleko osetljivija i važnija aktivnost u životnom ciklusu softvera od same izrade softvera. Objektno orijentisani pristup, koji je prikazan u ovom radu uvodi koncepte koji olakšavaju modelovanje realnog sveta i omogućavaju ponovnu upotrebu koda, ali samo pod uslovom da je softverski sistem dobro isprojektovan.
- Dobar projekat softvera znači ne samo u potpunosti zadovoljenje trenutnih zahteva nego i sposobnost softvera da se lako prilagodi novim budućim zahtevima korisnika. Zato su fleksibilnost i lakoća održavanja softvera jedan od ključnih principa kojima se teži u objektno orijentisanom programiranju.

2. Metodologije

- Osamdesetih godina programeri su počeli sa opisivanjem kontrolisanog procesa razvoja softvera, od opisa zadatka do održavanja gotovog proizvoda. Ovo je dovelo do stvaranja **strukturne metodologije** kao što je **SSADM** (Strukturna Sistem Analiza i Dizajn Metod). Metodologija predstavlja opis koraka kroz koje razvojni tim treba da prođe, a u cilju stvaranja visoko-kvalitetnog sistema [2].

- Devedesetih godina, objektno orijentisano programiranje preuzima dominaciju i projektanti počinju da rade na **Objektno-Orijentisanim Metodologijama**, prijemčivijim objektno -orijentisanom programerskom stilu. Rane OOM uključuju *Booch* metod [Booch 93], *Objectory* metod [Jacobson 92] i OMT [Rumbaugh 91], nastaje i ***Rational Unified Process*** kao i metodologija koja sve više dobija na popularnosti ***eXtreme Programming*** [3] **.**

2.1. Metodologije i mali projekti

- Metodologija pomaže pri nametanju discipline prilikom kodiranja.
- Prolaženjem kroz osnovne metodološke korake povećava naše razumevanje problema, poboljšavajući kvalitet našeg rešenja.
- Omogućava nam uočavanje konceptualnih i praktičnih grešaka pre nego što ih unesemo u izvorni kod.
- Na svakom nivou metodologija specifikuje naš sledeći korak.
- Metodologija nam pomaže da proizvedemo proširiv kod (koji se lako menja), kod koji se može koristiti za rešenje i drugih problema i koji je lakši za pronalaženje grešaka (jer je dobro dokumentovan).

2.2. Veliki razvojni projekti

- Dokumentacija: Metodologija obuhvata postupak dokumentovanja svake faze razvojnog truda, tako da za završeni sistem nije nepristupačno monolitan.
- Smanjenje vremena pristupa: Pošto su poslovni tokovi, aktivnosti, uloge i zavisnosti razumljiviji, javlja se manja mogućnost da proces stoji u redu za čekanje resursa.
- Povećava se šansa za isporukom na vreme i bez probijanja budžeta.
- Princip ponovljivosti: Pošto imamo dobro definisane aktivnosti, slični projekti bi trebalo da se isporučuju sa sličnim vremenskim rokom i sa sličnim cenama. Ako proizvodimo slične sisteme iznova za različite kupce možemo težiti ka metodologiji koja će se koncetrisati na jedinstvene aspekte poslednjeg projekta eventualno možemo automazizovati delove projekta i prodavati ih drugim proizvodnim timovima.

2.2.1. SSADM

Strukturna Sistem Analiza i Dizajn Metod

- **Analiza sistema**

Poznata i kao „Studija izvodljivosti“. Koristi Dijagram Toka Podataka za opis rada sistema i vizualizaciju poznatih problema.

Obuhvata sledeće korake:

- **Razvoj *Business Activity Model*-a**. Poslovni događaji i pravila se istražuju kao ulaz u specifikaciju novog automatizovanog sistema.
- **Pronalaženje (Ispitivanje) i definisanje zahteva**. Identifikuju se problemi u vezi sa okruženjem koji će biti rešeni od strane novog sistema.
- **Ispitivanje procesiranja**. Istražuju se tokovi informacija u saradnji sa postojećim uslugama i opisuju u formi **M**odela **T**oka **P**odataka. U datom stadijumu MTP predstavlja tekuće servise (usluge) sa svim nedostacima.
- **Ispitivanje tekućih podataka**. U ovom koraku se identifikuje i opisuje struktura podataka sistema nezavisno od načina trenutnog skladištenja i organizovanja podataka.Ovaj korak rezultira kreiranjem modela podataka koji podržava tekuće servise (usluge).
- **Izvođenje logičkog pogleda**. Razvija se logički pogled na dati sistem kako bi se omogućilo razumevanje problema tekućeg sistema.

2.2.1. SSADM
Strukturna **S**istem **A**naliza i **D**izajn **M**etod

- **Opis poslovne specifikacije**
  - Poznata i kao logička etapa specifikacije sistema. Sastoji se iz dva dela. Prvi deo je **istraživanje postojećeg okruženja** i obuhvata identifikovanje zahteva sistema i modelovanje poslovnog okruženja. Modelovanje čini kreiranje DTP i **L**ogičke **S**trukture **P**odataka za procese i strukture podataka koji su deo sistema. Drugi deo se sastoji u **odabiru i izgradnji jedne od 6 Poslovnih Opcija Sistema** (Business Systems Options).

2.2.1. SSADM

Strukturna Sistem Analiza i Dizajn Metod

- **Detaljna specifikacija sistema**

Poznata kao specifikacija zahteva. Obuhvata sledeće korake:

- **Definisanje zahtevanog procesiranja sistema**. Ovaj korak teži ka prilagođavanju zahteva kako bi reflektovali odabran BSO , opisali zahtevani sistem u pogledu tokova podataka sistema, i definisali korisničke uloge novog sistema.
- **Razvoj modela podataka**. Ovaj korak se odvija paralelno sa prethodnim. Logički model podataka datog okruženja se prosiruje kako bi podržao sva procesiranja u odabrana u BSO.
- **Izvođenje funkcija sistema**. Za vreme paralelnog definisanja podataka i procesa, dodatni događaji su identifikovani, što je dovelo do ažuriranja postojećih i definisanja novih funkcija.
- **Razvoj specifikacije korisničkih poslova**. Razvijen je Work Practice Model kako bi se dokumentovalo razumevanje poslova korisnika u potpunosti.
- **Poboljšanje modela podataka.** Poboljšanje kvaliteta logičkog modela podataka datog sistema primenom relacione analize podataka (poznate i kao normalizacija)
- **Razvoj prototipa specifikacije**. Opisuje odabrane delove sistema u prilagodljivoj formi demonstracije za korisnika.
- **Razvoj specifikacija obrade** (procesiranja).
- **Potvrda sistemskog cilja** (pravca sistema)

2.2.1. SSADM
Strukturna **S**istem **A**naliza i **D**izajn **M**etod

- **Logički dizajn podataka**
  - Poznat i kao logička specifikacija sistema. Vrši se odabir opcija koje su tehnički naj izvodljivije. Razvojno/implementaciona okruženja se specificiraju na osnovu ovog izbora.
- **Logički dizajn procesa**
  - Takođe poznat kao logička specifikacija sistema. Vrši se ažuriranje logičkog dizajna i procesa. Moguća je i specifikacija dijaloga.
- **Fizički dizajn**
  - Cilj ove faze je specifikacija fizičkog dizajna podataka i procesa, korišćenjem rečnika i karakteristika odabranog fizičkog okruženja i inkorporiranih standarda.

2.2.1. SSADM
Strukturna **S**istem **A**naliza i **D**izajn **M**etod

Tri najbitnije tehnike koje se koriste u SSADM su:

- **Logičko modelovanje podataka** – identifikacija, dokumentovanje, modelovanje zahteva sistema koji se dizajnira.
- **Modelovanje toka podataka** - identifikacija, dokumentovanje, modelovanje prenosa podataka u informacionom sistemu.
- **Modelovanje ponašanja entiteta** - identifikacija, dokumentovanje, modelovanje događaja koji utiču na svaki entitet i sekvenci u kojima se „okidaju“ događaji.

2.2.2. Objektno orijentisana metodologija

Namera OOM nije da propisuje pravila: projektanti imaju brojne mogućnosti i slobode izbora oko korištenja dijagrama ili nekih drugih artifakta. Međutim, projektni tim mora izabrati jednu metodologiju i složiti se oko artifakta koje će stvarati, pre nego što počne bilo kakvo detaljno planiranje ili popis.

Statičko modelovanje obuhvata delovanja nad logičkim i fizičkim delovima sistema i načinu na koji će oni biti povezani.

Dinamičko modelovanje je posvećeno izboru načina kolaboracije statičkih delova.

Grubo govoreći, statički model opisuje konstrukciju i inicijalizaciju sistema, dok dinamički model opisuje kako će se sistem ponašati za vreme rada. Uglavnom se proizvede jedan statički i jedan dinamički model u toku svake faze modelovanja.

2.2.2. Objektno orijentisana metodologija

- **UML, RUP i XP**
  - Godine 1996, *Jacobson* i *Rumbaugh* pridružili su se Rational Corporation (osnovane od strane *Booch*-a), i razvili set notacija koje su postale poznate kao ***Unified Modeling Language*** (UML). Neki projektanti svrstavaju UML kao notaciju za *brainstorming* i dokumentovanje na visokom nivou, drugi pak svrstavaju UML u slikovni programski jezik, koji generiše nov kod iz postojećeg koda.
  - **Rational Unified Process** je spiralni, iterativni, inkrementalni metod nastao od tvoraca UML-a. Može se reći da je to metodologija koja izdvaja najbolje aspekte UML-a.
  - Još jedna popularna metodologija je **eXtreme Programming [*Beck 99*]** smatra se brzom,efikasnom metodologijom jer dozvoljava i reaguje na promene. XP se odlikuje dvema osnovnim idejama: par programera i projektovanje navođeno testiranjem. Programeri rade, pre svega, na poboljšanju kvalieta softvera i na testiranju ali tako da test bude napisan i pre samog koda.

2.2.3. Klasične faze u razvoju softvera

- **Spisak zahteva**
  - otkriva šta želimo da postignemo sa novim softverom i ima dva aspekta:
    - Poslovno modelovanje (Business modeling) – razumevanje konteksta u kom će novi softver raditi
    - Funkcionalna specifikacija – definisanje mogućnosti softvera. Šta će novi softver raditi, a šta neće.
- **Analiza**
  - omogućava da spoznamo „sa čime imamo posla“. Pre nego što dizajniramo rešenje moramo biti svesni relevantnosti, svojstava i veza.
- **Dizajn**
  - U ovoj fazi razmatramo kako rešiti problem. Donosimo zaključke bazirane na iskustvu, proceni, intuiciji, kakav ćemo softver napisati i kako ćemo ga razviti.

2.2.3. Klasične faze u razvoju softvera

- **Specifikacija**
  - Kao osnova za dizajniranje test softvera za ispitivanje sistema
  - Za demonstraciju korektnosti,ispravnosti softvera
  - Za dokumentaciju softverskih komponenti i okvira u kojima se mogu koristiti od strane trećeg lica
  - Za opis ponovnog korištenja koda u okviru drugih aplikacija

Faze koje spadaju u klasične faze razvoja softvera, a u ovom radu ih neću podrobno opisivati jesu testiranje, uvođenje i održavanje.

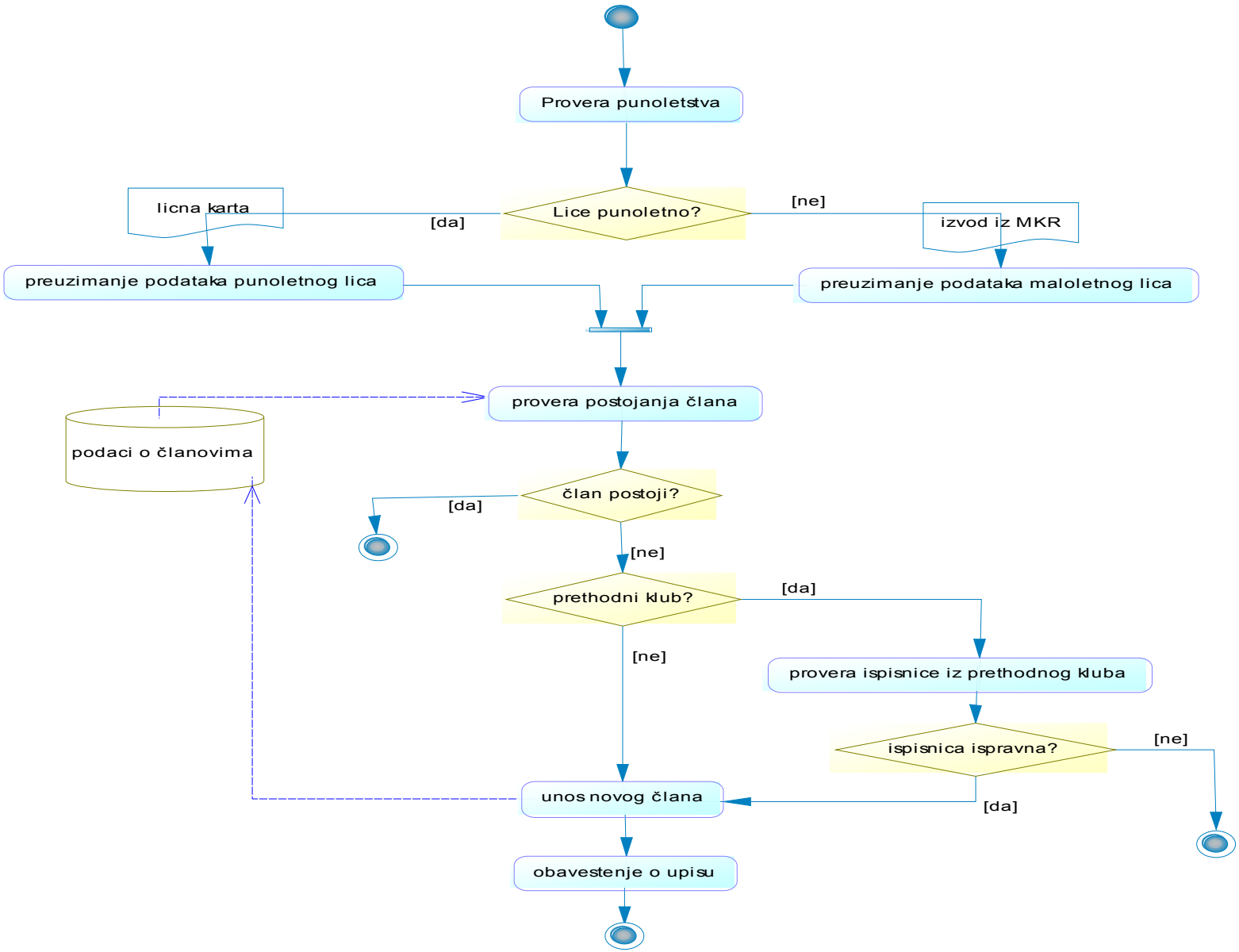
3. Stvaranje sistema

- Pre nego što uopšte i počnemo sa razmatranjem softvera koji ćemo napisati, moramo istražiti poslove u kojima će se naš softver primenjivati – bez razumevanja posla, teško da možemo očekivati da će naš softver poboljšati poslovanje.
- Kada smo razumeli poslove i dokumentovali naše shvatanje poslova u **business requirements** (poslovni zahtevi), moramo razmisliti o svrsi softvera u odnosu na korisnike. Bez razumevanja **system requirements** (sistemskih zahteva), rizikujemo da stvorimo kod koji neće biti nikome od koristi.

3.1. Poslovni model

- ***Business model*** (poslovni model) je prethodnica modeliranju funkcionalnosti sistema.Poslovni model može biti jednostavan kao dijagram klasa, pokazujući veze između poslovnih entiteta – ponekad se tretira kao **domain model** (područni model).

  - **Business Process Model** (BPM) se koristi za izradu poslovnog modela sistema i predstavlja složeniju alternativu Use Case modelima. BPM je konceptualni model koji omogućava opis poslovne logike i pravila sa korisničke tačke gledišta. Prikazuje interakciju između procesa, tokova, poruka i kolaboracionih protokola od jedne ili više početnih tačaka do nekoliko potencijalnih krajnjih tačaka.



UML dijagram Business Process Model-a

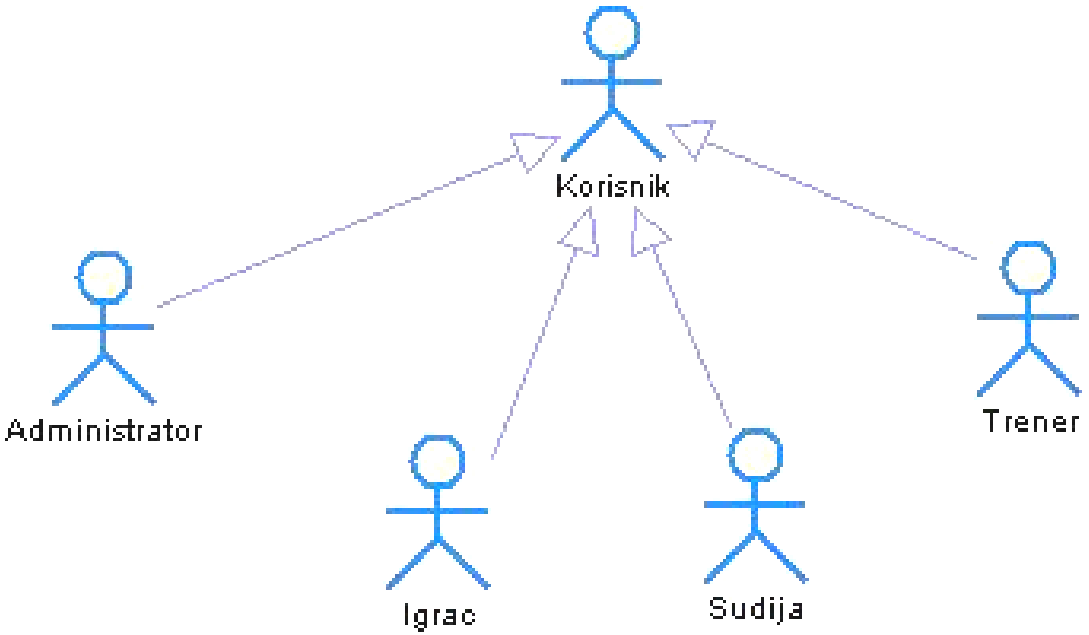
3.2. Use Case

Nakon dokumentovanja posla u BPM, opšte je prihvaćeno da se zahtevi sistema prezentuju i preko Use Case modela. Razlog za to je što se Use Case relativno lako kreira za prikaz sponzorima [2].

Use Case treba da sadrži:

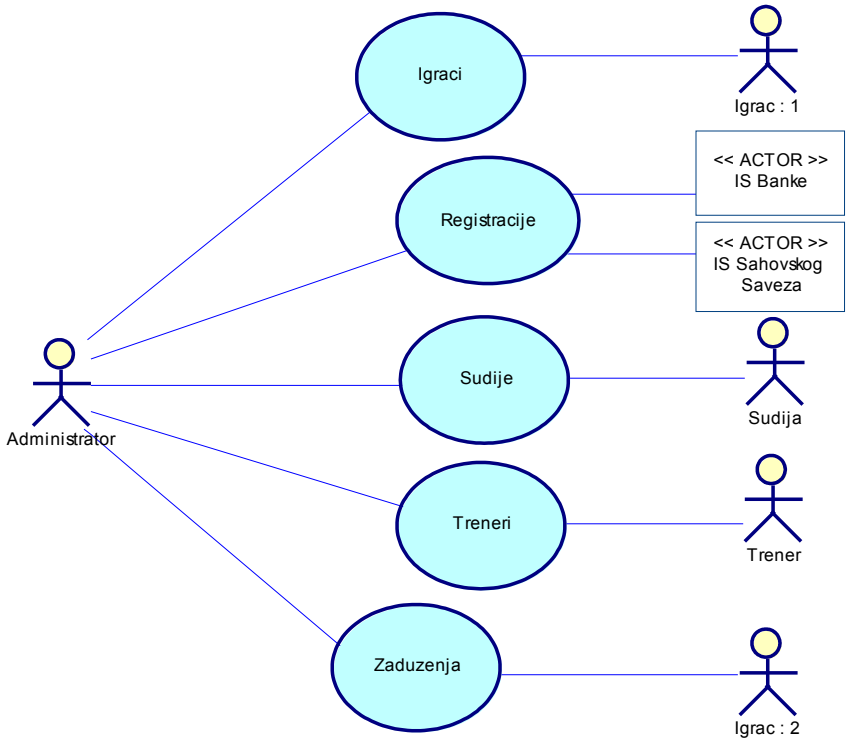
- listu učesnika (sa opisima)
- listu slučajeva korištenja (sa opisima)
- dijagrame slučajeva korištenja
- detalje slučajeva korištenja
- pregled slučajeva korištenja
- dodatne zahteve (sistemski zahtevi koji ne pripadaju nijednom posebnom Use Case dijagramu)
- Skicu korisničkog interfejsa
- Poboljšan rečnik pojmova
- Prioritete slučajeva korištenja

3.2. Use Case



UML dijagram Use Case-a liste učesnika

3.2. Use Case



UML dijagram Use Case-a glavnog menia

4. Analiza

Proces analize može sadržati sledeće korake koji se mogu ponavljati dok se ne postigne zadovoljavajući nivo za projektante i „sponzore“.

- Koristi se model zahteva sistema da bi se pronašli kandidati klasa koji opisuju objekte koji mogu biti relevantni za sistem i zapisuju se na dijagramu klasa
- Pronalaženje veza (asocijacija, agregacija, kompozicija i nasleđivanje) za klase
- Pronalaženje atributa za klase
- Prolaženje kroz sistemski *use case* , proveravajući da li je podržan od strane objekata koje imamo, fino podešavanje klasa, atributa i veza odakle sledi da *use case* realizacija prouzrokuje operacije za kompletiranje atributa.
- Ažuriranje rečnika podataka i nefunkcionalnih zahteva ukoliko je neophodno.

4. Analiza

Statička analiza

- Statičko modelovanje uređuje logičke i fizičke delove sistema i način na koji će biti spojeni. Odnosno, opisuje kako ćemo konstruisati i inicijalizovati sistem.

Sastoji se od:

- identifikovanja klasa
- identifikovanja relacija među klasama
- crtanja klasnih i objektnih dijagrama
- kreiranja atributa

4. Analiza

Dinamička analiza obuhvata:

- Use Case realizaciju – koriste se komunikacioni i sekvencni dijagrami
- Granice, entitete i kontrolere – koriste se komunikacioni dijagrami
  - učesnik – osoba ili sistem van granice sistema
  - granica – objekat na ivici sistema (između sistema i učesnika)
  - entitet – objekat u okviru sistema koji predstavlja poslovni koncept
  - kontrolor – uslužni objekat koji omogućava sledeće usluge
    - kontrolu sistemskih procesa
    - stvaranje novih entiteta
    - obnovu postojećih entiteta
- Elemente komunikacionih dijagrama
- Dodavanje operacija klasama
- Odgovornosti
  - Svaki objekat treba da ima tačno jedan posao (ili ulogu) ako postoji objekat zadužen za više od jednog posla tada je očigledno da nam je potrebno više objekata. Za objekte sa jedinstvenim setom odgovornosti se kaže da imaju jaku koheziju, što je željeni cilj.
- Modelovanje stanja

5. Projektovanje

- Cilj ka kome se teži u etapi projektovanja jeste **stvaranje dobrog – kvalitetnog proizvoda**. Do kvalitetnog proizvoda se dolazi kombinacijom dobrog projekta i dobre realizacije.Ukoliko se bilo koji od ova dva uslova ne ispuni krajnji proizvod neće biti kvalitetan.
- Na nastanak kvalitetnog softvera utiče nekoliko faktora koji se mogu **grupisati u spoljašnje i unutrašnje faktore.** Kvaliteti poput brzine i jednostavnosti korišćenja, a koje krajnji korisnici mogu direktno da iskuse, predstavljaju spoljašnje faktore kvaliteta; dok npr. modularnost i čitljivost softvera predstavljaju unutrašnje faktore, faktore koji su odlika izvornog teksta softvera.

5. Projektovanje

U spoljašnje faktore ubrajaju se:

- **Korektnost** – *mogućnost softverskog proizvoda da tačno izvršava svoje zadatke koji su definisani njihovom specifikacijom*.
- **Robusnost** – *mogućnost softvera da na odgovarajući način reaguje na nenormalne situacije.*
- **Proširivost** – *lakoća prilagođavanja softvera promenama specifikacije.*
- **Višekratna upotreba** – *mogućnostsoftverskih elemenata da služe u konstruisanju više različitih aplikacija.*
- **Kompatibilnost** – *lakoća kombinovanja softverskih elemenata jednih sa drugim.*
- **Efikasnost** – *mogućnost softvera da što manje zahteva hardverske resurse.*
- **Prenosivost** – *lakoća prenošenja softvera u različita hardverska i softverska okruženja*
- **Jednostavnost upotrebe** - *jednostavnost upotrebe je lakoća sa kojom osobe razli itog obrazovanja i kvalifikacija mogu da nauče da koriste softverske proizvode i primene ih u rešavanju problema. Ona podrazumeva i jednostavnost instalacije, održavanja i nadgledanja*
- **Funkcionalnost** – *dijapazon mogućnosti koje pruža sistem*
- **Blagovremenost** – *mogućnost softvera da bude završen kada ga korisnici žele.*

5.1. Dizajniranje arhitekture

Koraci u dizajnu sistema:

- odabir topologije sistema: na koji način će hardver i procesi biti distribuirani
- izbor tehnologije (programski jezici, baze podataka, protokoli,...)
- dizajniranje konkurentnosti : više korisnički, procesorski,računarski;
- dizajniranje sigurnosti: postoje mnogi aspekti sigurnosti koje moramo
- obezbediti i kontrolisati
- odabir delova podsistema: u većini slučajeva proizvodimo posebne aplikacije
- koje rešavaju određene probleme i moramo im obezbediti efektnu komunikaciju
- deljenje podsistema na slojeve ili druge podsisteme
- odluke o načinu komuniciranja mašina, podsistema i slojeva

5.1.1. Logička i fizička arhitektura

Ispravno dizajnirana logička n-slojna arhitektura obezbeđuje sledeće prednosti:

- logički organizovan kod
- lakšu manipulaciju
- bolje ponovno iskorištenje koda
- bolje iskustvo projektantskog tima
- visoku jasnoću prilikom kodiranja

Sa druge strane, ispravno dizajnirana fizička n-slojna arhitektura obezbeđuje sledeće prednosti:

- performanse
- skalabilnost
- nedostatak tolerancije
- bezbednost

5.1.1. Logička i fizička arhitektura

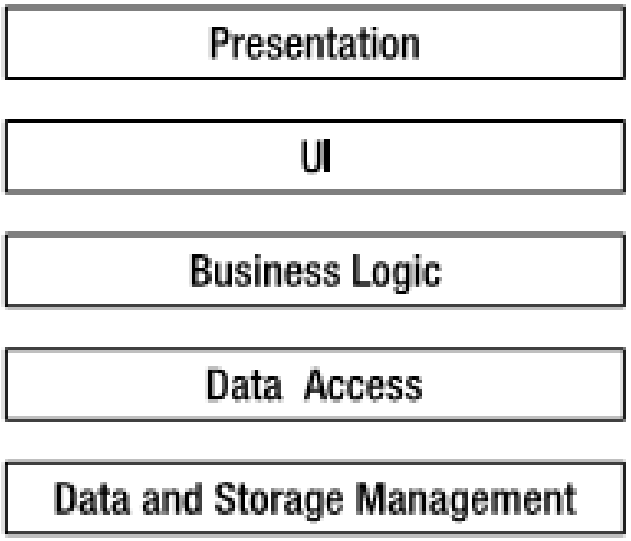
N- slojna arhitektura jedino pojednostavljuje proces kod velikih aplikacija ili kompleksnih okruženja.N- slojnu arhitekturu bi trebalo koristiti u slučajevima kada:

- aplikacija je velika ili kompleksna
- aplikacija je jedna od mnogih sličnih ili srodnih aplikacija i prilikom kombinovanja nastaje velika ili kompleksna aplikacija
- okruženje je veliko ili kompleksno

Tradicionalno, najzastupljenija n-slojna arhitektura je bila sastavljena od 3 sloja: interfejsa, poslovne logike i sloja podataka.

U novije vreme sve više se koristi 4 do 6 slojna logička arhitektura.

5.1.1. V-slojna logička arhitektura



5.1.2. Implementacija logičke arhitekture

Petoslojnu logičku arhitekturu [8] je moguće konfigurisati u jedan, dva, tri, četiri ili pet fizičkih slojeva u cilju poboljšanja performansi, skalabilnosti, bezbednosti ili nulte tolerancije.

Optimalne performanse- pametni klijent

Ovaj način implementacije je korišten prilikom projektovanja i implementacije SPPR Šahovskih Klubova.

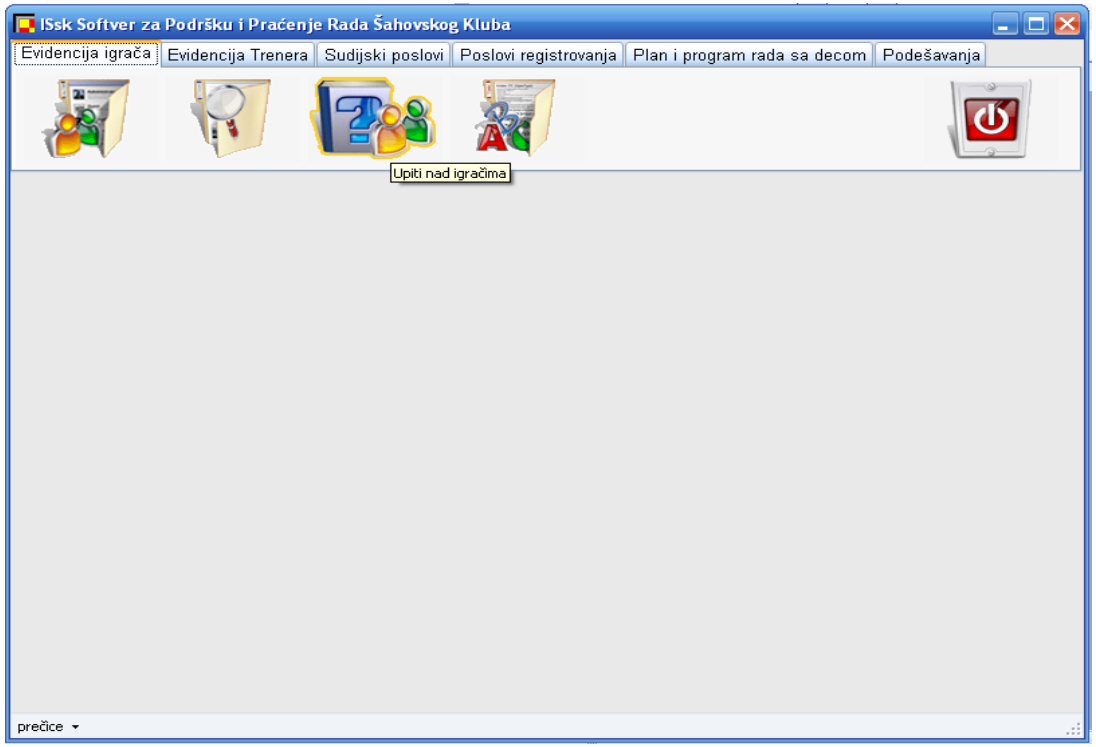
- Implementacija obično podrazumeva Windows Forme za prezentacioni i UI sloj , sa kodom poslovne logike i sloja pristupa podacima koji se izvršava u istom procesu i obraća se *Access (JET)* ili *Microsoft SQL Server Express* bazi podataka.Činjenica da je sistem upošljen na samo jednom fizičkom sloju ne kompromituje logičku arhitekturu i odvajanje kao što je i prikazano na slici ispod.

5.2. Dizajn podsistema

Vodeći se principima dizajniranja sistema, dizajn podsistema možemo definisati preko:

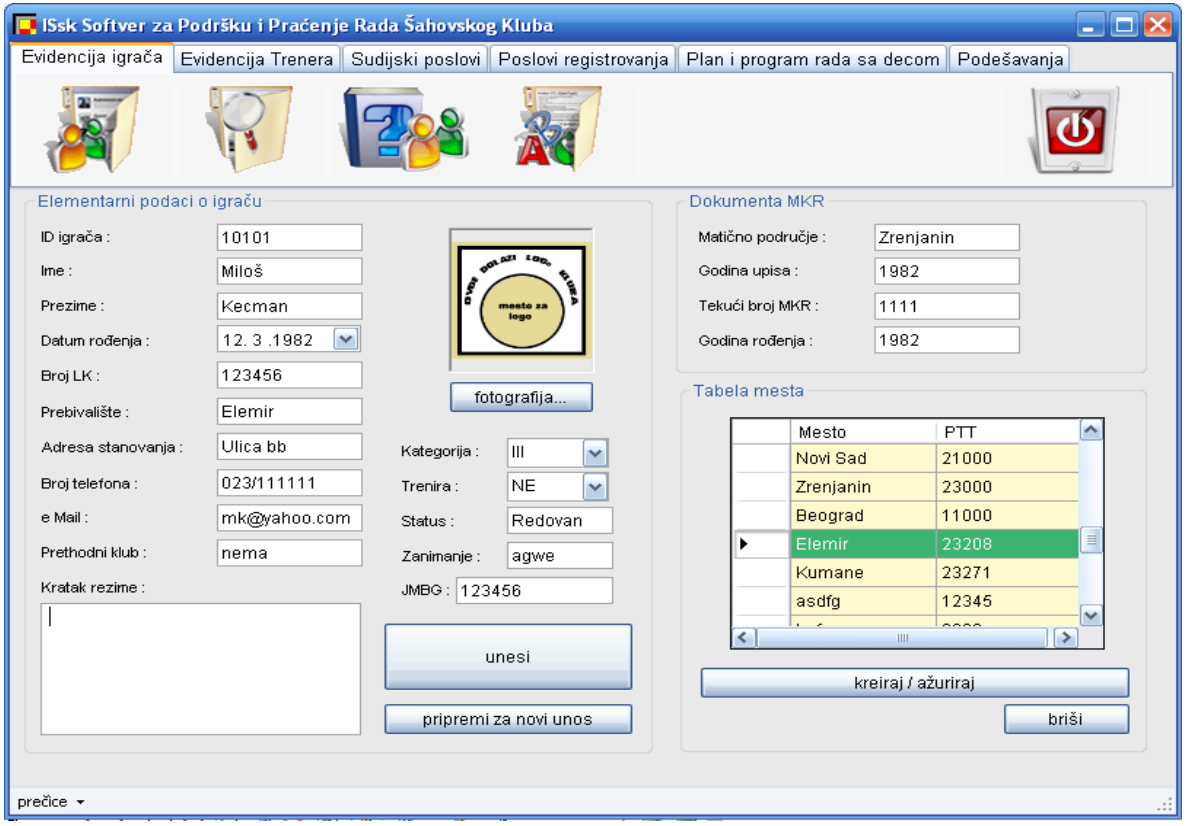
- dizajna klasa i polja poslovnog sloja koristeći klasni model iz postupka analize kao vodič
- načina skladištenja perzistentnih podataka i dizajna skladišta
- izgleda korisničkog interfejsa na osnovu skica proizvedenih u toku postupka analize
- prelaza kroz slučajeve korišćenja sa referencama na dizajn korisničkog interfejsa oslanjajući se na poslovne servise koji moraju biti podržani u srednjem sloju
- razvoja poslovni servisa u serverske objekte (distribuiran dizajn sistema)
- definisanja mera koje su potrebne za osiguranje konkurentnosti sistema (na osnovu poslovnih pravila koja kontrolišu pristup sistemu: korisničko ime, šifra,...) i paralelnog procesiranja

6. Struktura aplikacije



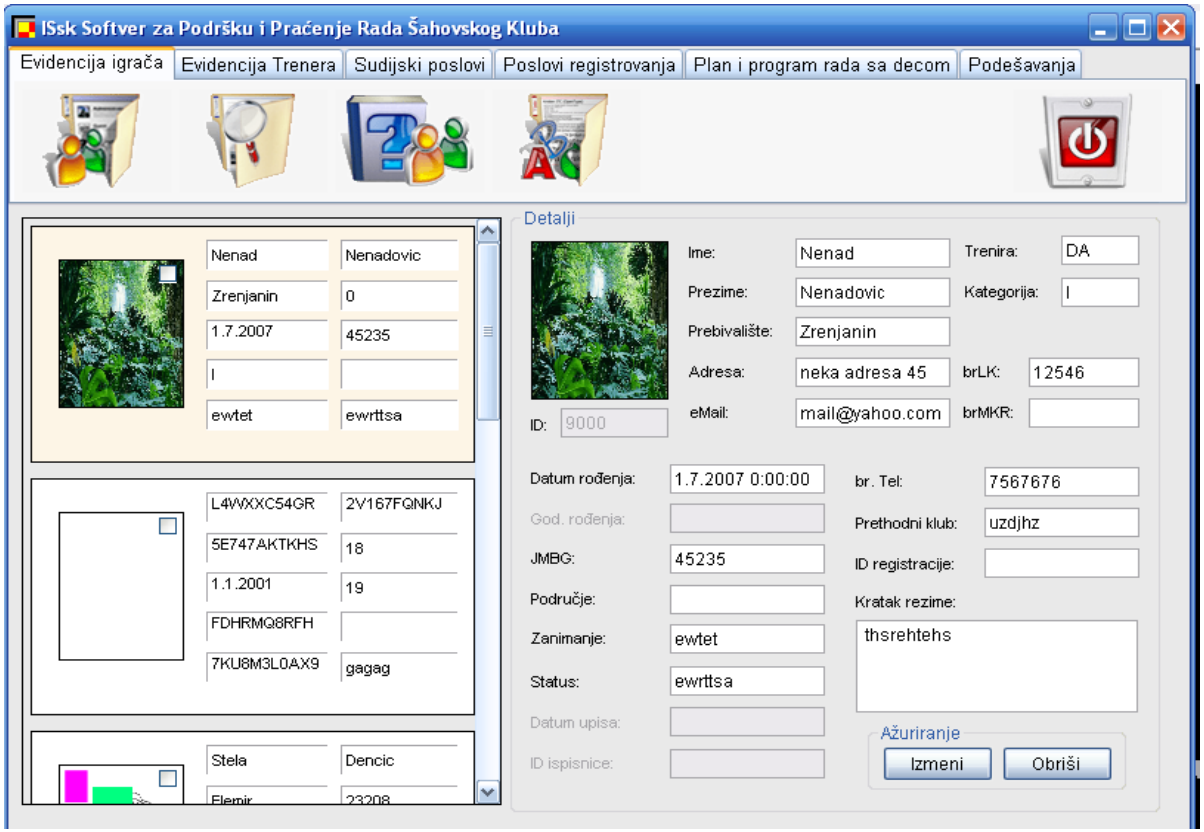
Glavni meni

6. Struktura aplikacije



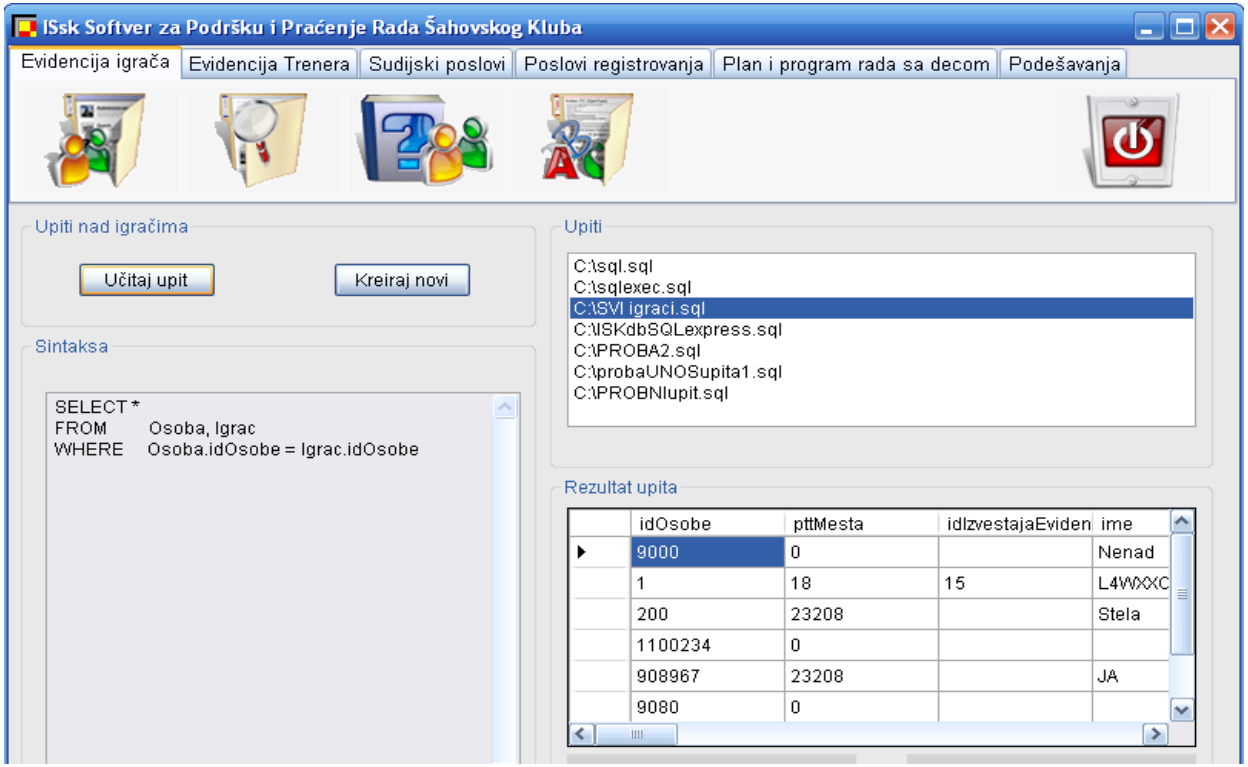
Unos igrača

6. Struktura aplikacije



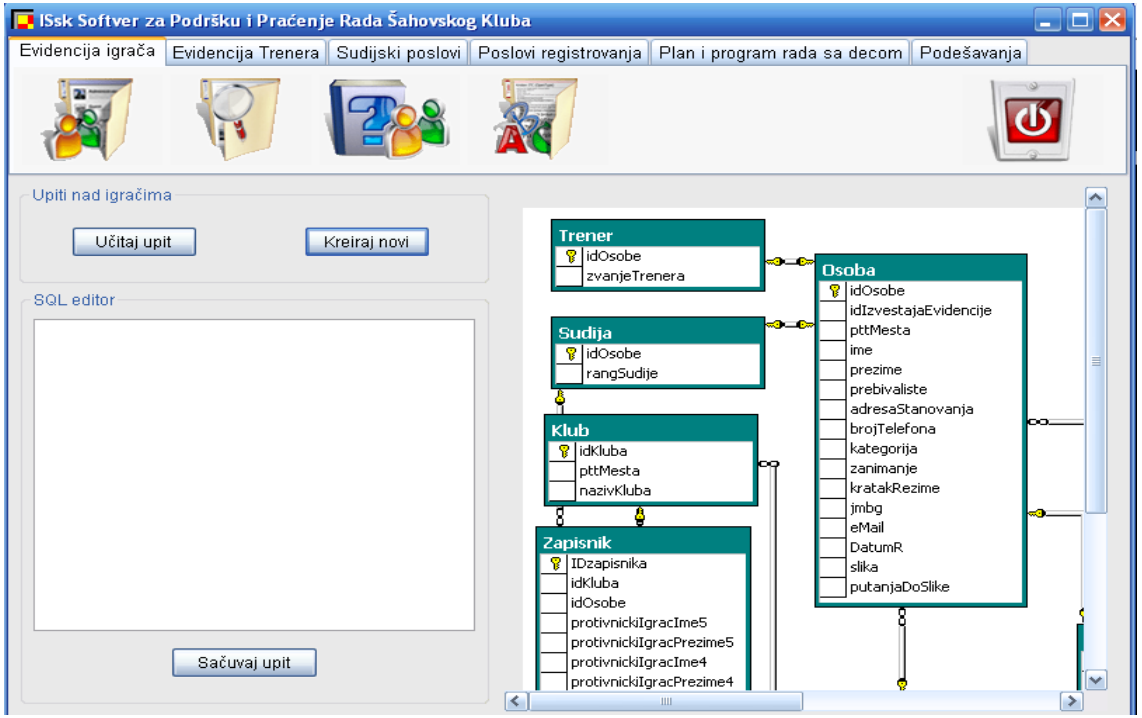
Pregled igrača

6. Struktura aplikacije



Upiti nad igračima

6. Struktura aplikacije



Upiti nad igračima - kreiranje upita

Literatura

- [1] Eric J. Naiburg ,Robert A. Maksimchuk „UML for Database Design“ ,July 01,2001
- [2] Mike O'Docherty „Object-Oriented Analysis and Design“, John Wiley & Sons 2005
- [3] By Dr. Neil Roodyn „eXtreme .NET: Introducing eXtreme Programming Techniques to .NET Developers“ ,Addison Wesley Professional, December 2004
- [4] Pejić Jovan „Kriptografski i stenografski postupci“, diplomski rad, Tehnički fakultet “Mihajlo Pupin”, Zrenjanin, 2006.
- [5] Dr. Radosav Dragica, „Softversko inženjerstvo I“, Tehnički fakultet “Mihajlo Pupin”, Zrenjanin, 2005.
- [6] Dr. Radosav Dragica, „Softversko inženjerstvo II“, Tehnički fakultet “Mihajlo Pupin”, Zrenjanin, 2005.
- [7] Dr.Radulović Biljana, Ljubica Eremić, Kazi Zoltan, „Odabrana poglavlja projektovanja informacionih sistema“, Tehnički fakultet “Mihajlo Pupin”, Zrenjanin, 2002.
- [8] Rockford Lhotka „Expert C# 2005 Business Objects“ , Apress 2006
- [9] Stephen C. Perry „Core C# and .NET“ , September 2005