PRIKAZ CELOVITOG OKRUŽENJA MIGRACIJE SOFTVERSKIH SISTEMA SA OSVRTOM NA PRISTUPE MIGRACIJI 4GL APLIKACIJA

Veselin Gredić, *Vazduhoplovni Opitni Centar - Batajnica*

Sadržaj – *Kroz rad je opisana globalna okolina u kojoj se obavlja migracija softverskih sistema sa osvrtom na širi spektar tehničkih i problema upravljanja koji se moraju razmatrati. Dati su pojedini primeri pripremljenih "listi pitanja" čija je namena da se ispitaju elementi celovitog okruženja i dobiju odgovori na strategijska usmerenja, politiku organizacije i pojedinih celina kao i potreba kupaca za značajnim servisima. Kao primer opšte analize nasleđenog sistema opisani su pojedini pristupi za migraciju 4GL informacionih sistema.*

1. UVOD

Različite organizacije su osetile ogroman pritisak za proširenjem i unapređenjem svojih softverskih sistema pošto je jasna potreba da se odgovori novim zahtevima tržišta i ubrzanim tehnološkim promenama. Ovakav konstantan pritisak je upravljan :

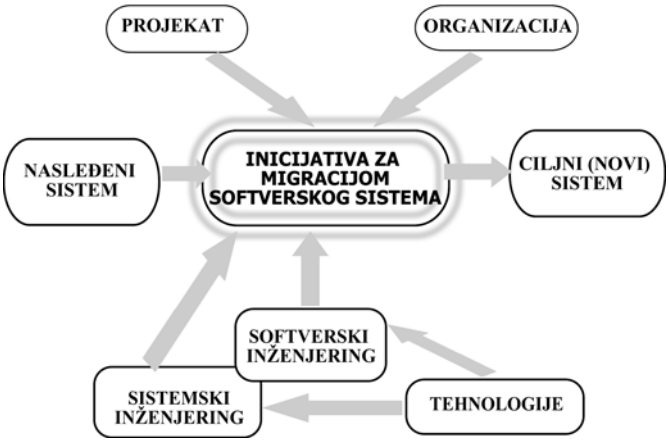
- pojačanim očekivanjima kupaca i potrebi da se odgovori novim proširenim standardima,
- potrebom da se poboljšaju performanse,
- potrebom objedinjavanja novih proizvoda sa sistemom,
- borbom sa novim verzijama softvera,
- potrebom prekida kontinuiranog zastarevanja hardvera i softvera.

U organizacijama gde nije uspostavljena disciplina razvoja softvera (softverskih procesa) često se daje akcenat na specifičan skup tehničkih pitanja. Međutim, u realnosti mnogi aspekti razvoja sistema ne odnose se striktno na tehničke probleme. Potrebno je povesti računa o širem spektru inženjerskih problema, povećanim potrebama kupaca, strateškim ciljevima i smerovima organizacije i njenim poslovnim procesima. Jedan od primera je i "problem 2000 godine (Y2K)" koji je opisan u [1]. Kriza izazvana ovim problemom je jednostavna, dobro poznata i zaokružena tehnička problematika, ali su se morali uzeti u obzir i aspekti organizacije (strateški, organizacioni i poslovni).

Kroz celovito okruženje prikazane su globalne karakteristike okoline u kojoj se obavlja razvoj sistema i dat je uvid u deo tehničkih i problema upravljanja koji se moraju razmatrati pri "intenzivnom" razvoju softverskih sistema. Nadalje dati su primeri listi pitanja (engl. "*chacklist*"), koji su kreirane kako bi ispitale strategijska usmerenja i politika organizacije, pojedinih celina i potreba kupca za značajnim servisima.

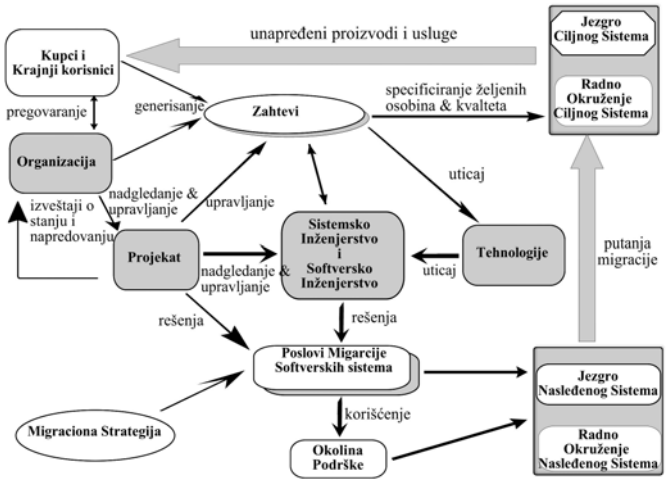
2. CELOVITO OKRUŽENJE

Celovito okruženje se sastoji od sedam elemenata koji predstavljaju oslonce za uspešnu migraciju softvera. Svaki element poseduje kritičan skup tehničkih i problema upravljanja koji su bitni za sačinjavanje sveobuhvatnog plana. Na Sl. 1 su prikazani elementi okruženja.



Sl.1. *Elementi celovitog okruženja*

Elementi celovitog okruženja se mogu primeniti na široku klasu problema migracije softverskih sistema. Raspored između elemenata i njihov odnos su u funkciji konkretnog okruženja, kulturoloških aspekata kao i tehničke i prakse rukovođenja. Na Sl. 2 je data arhitektura visokog nivoa za izvršavanje migracije softverskih sistema.



Sl.2. *Opšta arhitektura migracije softverskih sistema*

Na Sl.2. je ilustrovana uloga elemenata okruženja, njihovih relacija i načina na koji će se isti integrisati u životni ciklus procesa migracije softverskih sistema. Osenčeni objekti su elementi okruženja, a osnovne uloge su označene na strelicama koje označavaju vezu između elemenata. Ciklus počinje sa pregovaranjem (dogovaranjem, ubeđivanjem) između kupca i organizacije a završava sa unapređenim proizvodom i uslugom za krajnjeg korisnika. Žižna tačka predstavljenog pogleda je objekat koji je označen kao "Poslovi Migracije Softverskih sistema ". Konceptualno, transformacije neophodne za migraciju postojećeg sistema u željeni sistem su specificirane u formi skupa poslova koji predstavljaju dobru inženjersku praksu sistemskog i softverskog inžinjeringa. U praksi ovi poslovi su izvedeni iz

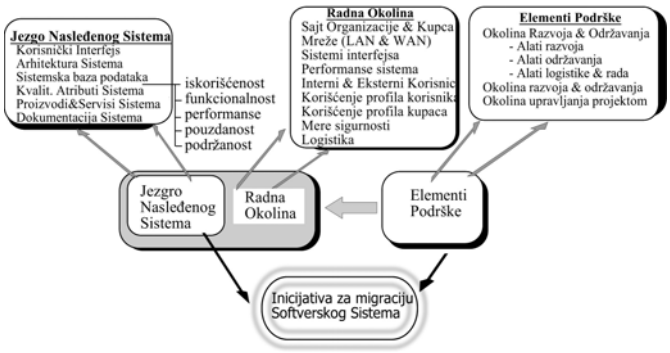
odabrane strategije migracije koja je obično opisana u planu projekta. Poslovi se mogu izvršavati od strane projektnog tima, tima sistemskih i softverskih inženjera, ostalih organizacionih celina ili strana koji imaju ugovore sa njima.

Primeri migracije softverskih sistema su : inkrementalna poboljšanja sistema, prebacivanje host aplikacija na novu platformu, kao i reversni inženjering sistema koji dodaje nove osobine sistemu (poboljšana funkcionalnost, poboljšanje zaštite, ili smanjena osetljivost na greške). U primeru prebacivanja *host* aplikacija na novu platformu poslovi migracije softverskog sistema mogu biti : analiza koda kojom bi se odredila zavisnost od nasleđenog sistema, poboljšanja koda u smislu iskoristivosti, razvoj I/O upravljačkih programa i izvršavanje regresivnih testova. U primeru reverznog inžinjeringa poslovi migracije softverskih sistema mogu uključiti : razvoj domenskog modela, razvoj koncepta rada, pravljenje prototipova, razvoj nove sistemske arhitekture i poboljšanje postojećih aplikacionih modula.

Ostali ne osenčeni objekti na Sl. 2 predstavljaju značajne čvorove u relacijama između elemenata okruženja. Čvorovi "Zahtevi" i "Strategija Migracije " igraju važnu ulogu u procesu planiranja migracije kao i vođenju dizajna sistema i implementacije.

3. POLAZNE OSNOVE ZA ANALIZU NASLEĐENOG SISTEMA

Migracija softverskih sistema počinje sa nasleđenim sistemom koji uključuje : jezgro nasleđenog sistema , njegovo radno okruženje i okruženje podrške (Sl. 3).



Sl.3. *Nasleđeni sistem*

Jezgro nasleđenog sistema je radni "softverski-intenzivni" sistem koji je kandidat za unapređenje. Kao što je prikazano na Sl. 3 sistem je opisan u formi tehničkih faktora kao što su njegova arhitektura, proizvodi i servisi, funkcionalnost, iskoristivost i ostali kvalitativni atributi. Izazov u disciplinovanoj migraciji softverskih sistema je u razumevanju funkcionalnosti, dizajna, rada i performansi nasleđenog sistema i u anticipiranju tipa izazova koji će biti tražen tokom životnog veka sistema. Nakon više godina održavanja, unapređivanja i usavršavanja nasleđenog sistema korisnički priručnici i dokumentacija dizajna sistema je često zastarela, netačna i ne odslikava tekuće osobine sistema i njegov rad.

Tehničke karakteristike i stanje sistema mogu se ispitati kroz inicijalnu listu pitanja :

- Da li sistem ima konfiguracioni dijagram ili dijagrami dizajna sistema ?
- Da li je softverska arhitektura i dizajn softvera dobro dokumentovana ?
- Da li su sistemski interfejsi i komunikacioni protokoli dobro dokumentovani ?
- Da li su funkcionalnost i rad sistema opisani korektno u korisničkoj i sistemskoj dokumentaciji ?
- Da li su svi korisnički interfejsi dokumentovani ?
- Da li bile sve softverske aplikacije i kritični algoritmi identifikovani ? Da li su analizirani ?
- Da li su svi softverski interfejsi, poruke i formati podataka identifikovani ?
- Da li su performanse sistema procenjene ? Da li su korišćeni *benchmark* programi?
- Da li je dostupan izvorni kod, biblioteke i skript fajlovi? Da li su aktuelni ?
- Da li postoji dokumentacija o fizičkim i logičkim rečnicima podataka ?
- Da li su zavisnosti koje nisu dokumentovane identifikovane ?
- Da li procenjena kompleksnosti i osetljivost sistema?
- Koliko je stabilan rad sistema ? Da li su zapisani nerešeni problemi i da li su pregledani zahtevi za promenama zbog informacija o trendu ?

Osnovni dokument radne okoline nasleđenog sistema može se dobiti kroz sledeći listu pitanja :

- Da li su svi kupci, njihovi sajtovi i korisničke grupe identifikovane ?
- Da li su svi proizvodi i servisi od kojih korisničke grupe zavise identifikovani ?
  Da li postoji profile za precizno karakterisanje tekućeg "opterećenja " sistema ?
- Da li su identifikovani svi eksterni ulazi/izlazi, sistemski fajlovi i procedure od kojih korisnik zavisi?
- Da li postoji radni scenario upotrebe kako bi se osiguralo opšte razumevanje osobina sistema i njegovog rada iz korisničke perspektive ?
- Da li postoji tačan i ažuriran konfiguracioni dijagram mreže koji specificira podsisteme i njihove interfejse ?
- Da li se mogu identifikovati svi vanjski interfejsi sistema i da li su dokumentovani ?
- Da li su identifikovani i dokumentovani svi softverski komunikacioni protokoli ?
- Da li su izmene u sigurnosti zahtevima potpune jasne projektnom timu ?
- Da li su navedene operacije (uloge i odgovornosti) logistike, podrške i administracije sistema ?
- Da li će rad nasleđenog sistema biti ustaljen kako bi omogućio adekvatno vreme za korisnike da omoguće obuku i potpuno obavljanje prelaza na predloženi sistem ?

Više elementa su uključeni u upravljanje, razvoj, održavanje i podršku sistema tokom njegovog životnog ciklusa. Okolina

podrške može biti mešavinu okoline za razvoj i održavanje, testiranje i integraciju, upravljanje projektima i ostalih funkcije podrške.

Kao i nasleđeni sistem, ova okolina se razvija tokom vremena u skladu sa zrelošću sistema i pojavljivanjem novih potreba. Naprimer, inicijalno fokus je usmeren preko alata za razvoj na proces razvoja i održavanja, ali nakon što nasleđeni sistem postane potpuno operativan fokus se preusmerava prema alatima za analizu i održavanje sistema, izvršavanje odabranih poboljšanja i testiranja sistema i svakodnevnoj podršci rada. Ipak kritično je održavanje ključnih mogućnosti originalnog alata za razvoj (da bi se efikasno i disciplinovano podržala poboljšanja i razvoj sistema).

Važno je razmotriti pitanje, u uspostavljanju okoline za održavanje, da li će biti odvojena okolina za integraciju, testiranje i podršku problemima ili će biti sadržana u radnom delu sistema. Još jedan aspekt okoline za podršku je stepen podrške automatizovanih alata za upravljanje projektom i funkcijama podrške kao što su upravljanje konfiguracijom i osiguranjem kvaliteta.

4. OPIS OKRUŽENJA 4GL APLIKACIJA

Mnoge komercijalne aplikacije zasnovane na RDBMS razvijene su uz upotrebu 4GL. Upotreba ovih alata smanjuje cenu razvoja informacionih sistema. Proizvođači alata često obezbeđuju i dodatne skupove alata kako bi dodatno smanjile cenu razvoja. Nedostatak ovakvih okolina je što su 4GL specifični za razliku od dobro definisanih standardnih okolina (kao što je npr. *COBOL* okruženje) što uslovljava zavisnost od proizvođača. Ova razlika je još značajni nedostatak ako proizvođač *DB*-a (baze) nudi tehnološki superiorno rešenje ili ako servis od strane proizvođača opada u smislu kvaliteta. U oba slučaja organizacija bi bila sprečena da migrira aplikacije na novu okolinu koju obezbeđuje drugi prodavac zbog velikih troškova koji su neophodni da se ponovo razviju aplikacije. Naprimer, neki prodavac može ponuditi *ugrade* svog proizvoda nudeći tehnologiju koja unapređuje proizvod kao što je podrška paralelnom procesiranju. Za neke organizacije unapređenje performansi koji nudi takav *DB* sistem obezbeđuje značajan napredak, tako da migracija aplikacija prema novom proizvođaču postaje opravdana.

Kada 4GL aplikacije nisu prenosive organizacija koja želi da promeni proizvođača mora ponovo da piše aplikacije. Slična situacija se pojavljuje kada se zbog stanja na tržištu (zbog komercijalnih razloga) proizvođač nađe u situaciji da odustane od dalje proizvodnje svog 4GL-a. Njegove konkurentne aplikacije neće moći da budu iskorišćene u savremenim *DB* tehnologijama i novim aplikacijama pošto unapređenja postoje kod drugih proizvođača. Nove aplikacije koje će biti razvijene upotrebom starijih tehnologija traže više resursa nego što bi se očekivalo u savremenim okruženjima za okoline koje nemaju perspektivu unapređenja.

4GL zasnovani IS se sastoje od korisničkog interfejsa, logike aplikacije i RDBMS. Iz korisničke perspektive sistem se sastoji iz formi, *report*-a i menija u određenoj hijerarhiji. Proizvođači obezbeđuju alate za razvoj aplikacija koji su spregnuti sa odgovarajućim RDBMS-om tako da pomažu bržem konstruisanju navedenih komponenti.

Alat formi se sastoji od dvije komponente nazvane *form dizajner* i *forms run-time* okolina . *Form dizajner* se koristi za konstrukciju formi koji su moduli sastavljeni od korisničkog interfejsa i od odgovarajuće logike aplikacije. Dizajner može automatski generisati korisnički interfejs za svaku formu koristeći informacija iz šeme baze podataka. Ekran se sastoji od polja koje odgovaraju kolonama u bazi podataka. Veličina i tip ulaza od strane korisnika se automatski proveravaju. Ovaj alat takođe služi kao programska okolina za proceduralni jezik i kao *run-time* okolina za aplikaciju.

Ovaj deklarativni tip formi daje više mogućnosti 4GL informacionom sistemu zato što obezbeđuje funkcionalnost koja je uključena u aplikaciju bez potrebe da se aplikacija programira.

5. MOGUĆI PRISTUPI MIGRACIJI 4GL APLIKACIJA

Prvi pristup, koji možemo nazvati **emulacija,** konvertuje svaku 4GL naredbu iz izvornog jezika u jednu ili više naredbi ciljnog jezika uz odgovarajuće semantičko predstavljanje svake pojedinačne izvorne naredbe. Svaka pojedinačna izvorna naredba može se izolovati. Ako postoji bilo kakva značajna semantička razlika između jezika onda karakteristike izvornog jezika su emulirane u ciljni jezik uz upotrebu rutina i biblioteka koje su posebno napisane za ovu svrhu. Ovim je obezbeđeno da emulacioni alat potpuno automatski konvertuje aplikaciju ali rezultujući kod može biti opširniji nego “originalni“ kod tj. kod koji je pisan ručno uz upotrebu ciljnog 4GL. Ovaj prikaz je takođe izložen riziku da jednostavno preslikavanje može rezultovati degradacijom performansi. Pristup emulacije je sličan rekonstruiranju koju su Chikosfsky i Cross [2] definisali kao “transformacija iz jedne predstavljene forme u drugu na relativno istom nivou apstrakcije, uz očuvanje karakteristika sistema (funkcionalnosti i semantike)“. Međutim pristup emulacije može zamenjivati jednu izvornih naredbi sa više ciljnih naredbi tj. pozivom emulacionih podprograma. Ciljni 4GL proizvod može biti posmatran kao da je na nižem nivou apstrakcije bez obzira što su izvorni i ciljni sistemi implementirani upotrebom 4GL.

Drugi pristup je migracija kroz ponovni dizajn (*design recovery* [2,3,4,5]) kojom se izvode informacije na nivou dizajna iz izvornih aplikacija i nakon toga takve informacije služe kao ulaz softverskim alatima koji generišu ciljnu aplikaciju. Problem ovog pristupa je što se ne mogu automatizovano izvući sve informacije izvornog sistema jer su neke od njih skrivene i nestandardne što za alate ove namene predstavlja problem. Stepen automatizacije koji se može postići ovakvim pristupom zavisi od logike i interfejsa alata koji treba da daju aproksimaciju nasleđene aplikacije. Potpuna funkcionalnost aplikacije može se postići nakon toga kroz uspostavljanje zahteva aplikacije koristeći dizajnerska svojstva novog alata.

Sledeći pristup možemo nazvati prepoznavanje šablona ili uzoraka. Ovim pristupom se prepoznaju uzorci koji se naknadno generišu odgovarajućim alatom koji je na višem nivou apstrakcije. Praksa pokazuje da se ovakvim pristupom mogu dobiti veoma brzo razultati koji su kvalitetni.

6. ZAKLJUČAK

Pažljivo razmatranje uloge i doprinosa predhodno prikazanih elemenata okruženja i razumevanje njihovih veza i konkretno datoj organizaciji (datom problemu) predstavlja važan korak u postizanju sveobuhvatnog i kordinisanog prilaza migraciji softverskih sistema. Jedan od načina da okruženje unapredi razumevanje ovih elemenata i njihovih međusobnih veza je preko razmatranja postojeće pozitivne prakse i uz analizu proverenih listi pitanja.

Stepen automatizacije koji se može postići upotrebom različitih tehnika i alata koji ih podržavaju pri migriranju sistema, kao što su 4GL aplikacije, obezbeđuje ubrzanje procesa. Ipak iskustva govore da su u procesu migracije softverskih sistema ključni faktor uspeha ljudi koji ga izvršavaju.

LITERATURA

[1] Smith, Dennis B.; Muller, Huasi A.; & Tilley, Scott R. The Year 2000 Problem: Issues and Implications (CMU/SEI-97-TR-002, ADA325361).Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1997.

[2] Chikofsky, E. and Cross, J., “Reverse Engineering and Design Recovery: A Taxonomy ”, IEEE Software, January, 1990,pp.13-18.

[3] Kozaczynski, W., “A Knowledge-Based Approach to Software System Understanding”, Proceedings ofthe Sixfh Conference on Knowledge-Based Soflware Engineering, Syracuse, NY, September, 1991, pp. 203-213.

[4] Ong, C. L., and Tsai, W. T., “Class and object extraction from imperitive code”, Journal of Object-Oriented Programming, March, 1993, pp. 58-68.

[5] Ning, J., Engberts, A. and Kozaczynski, W., Recovering Reusable Components from L.egacy Systems by Program Sementation”, Proceedings of the IEEE Working Conference on Reverse Engineering, Baltimore, MD, May, 1993, pp.64-72.

Abstract – Through this papers are describes enterprise framework for software system migrations with retrospections of wide specter technical and managements aspect that must be realized. Exemplary checklists are included to identify critical enterprise issues corresponding to each of the framework’s elements and obtain answers on strategic directions, policy of organizations or particular part and also customers and users needs for important services. Approach for migrations 4GL information system is presented like example of legacy systems global environment.

ENTERPRISE FRAMEWORK FOR SOFTWARE SYSTEM MIGRATIONS WITH APPROACH FOR LEGACY 4GL APLICATION MIGRATION

Veselin Gredić