Razvoj poslovnih aplikacija pristupom linija za proizvodnju softvera

Roško, Zdravko

Sveučilište u Zagrebu

FAKULTET ORGANIZACIJE I INFORMATIKE

Zdravko Roško

RAZVOJ POSLOVNIH APLIKACIJA PRISTUPOM LINIJA ZA PROIZVODNJU SOFTVERA

DOKTORSKI RAD

Varaždin, 2015.

Sveučilište u Zagrebu

FACULTY OF ORGANIZATION AND INFORMATICS

Zdravko Roško

BUSINESS APPLICATIONS DEVELOPMENT BASED ON SOFTWARE PRODUCT LINES APPROACH

DOCTORAL THESIS

Varaždin, 2015.

PODACI O DOKTORSKOM RADU

I. AUTOR

Ime i prezime	Zdravko Roško
Datum i mjesto rođenja	12. studenog 1962., Sonković
Naziv fakulteta i datum diplomiranja na VII stupnju	Ekonomski Fakultet Zagreb, 10. studenog 1988.
Sadašnje zaposlenje	Adriacom software d.o.o.

II. DOKTORSKI RAD

Naslov	Razvoj poslovnih aplikacija pristupom linija za proizvodnju softvera
Broj stranica, slika, tabela, priloga, bibliografskih podataka	265 stranica, 98 slika, 50 tablica, 1 prilog, 197 bibliografskih podataka
Znanstveno područje i polje iz kojeg je postignut doktorat znanosti	Društvene znanosti / Informacijske i komunikacijske znanosti
Mentori ili voditelji rada	Prof. dr. sc. Vjeran Strahonja
Fakultet na kojem je obranjen doktorski rad	Fakultet organizacije i informatike Varaždin
Oznaka i redni broj rada	

III. OCJENA I OBRANA

Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena tema	3. ožujka 2015.
Datum predaje rada	15. lipnja 2015.
Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena pozitivna ocjena rada	
Sastav povjerenstva koje je rad ocijenilo	Dr.sc. Danijel Radošević Dr.sc. Vjeran Strahonja Dr.sc. Mladen Varga
Datum obrane doktorskog rada	31. kolovaza 2015.
Sastav povjerenstva pred kojim je rad obranjen	Dr.sc. Danijel Radošević Dr.sc. Vjeran Strahonja Dr.sc. Mladen Varga Dr.sc. Zlatko Stapić Dr.sc. Dragutin Kermek
Datum promocije	

Sveučilište u Zagrebu

FAKULTET ORGANIZACIJE I INFORMATIKE

Zdravko Roško

RAZVOJ POSLOVNIH APLIKACIJA PRISTUPOM LINIJA ZA PROIZVODNJU SOFTVERA

DOKTORSKI RAD

Mentor: prof. dr. sc. Vjeran Strahonja

Varaždin, 2015.

Sveučilište u Zagrebu

FACULTY OF ORGANIZATION AND INFORMATICS

Zdravko Roško

BUSINESS APPLICATIONS DEVELOPMENT BASED ON SOFTWARE PRODUCT LINES APPROACH

DOCTORAL THESIS

Supervisor: prof. dr. sc. Vjeran Strahonja

Varaždin, 2015.

SAŽETAK

Planski pristup ponovnoj upotrebi (engl. *reuse*) kod razvoja softvera, koji se naziva „linija za proizvodnju softvera“ (engl. *software product lines*), uspješno se primjenjuje u poslovnim sektorima kao što su mobilna telefonija, kućna elektronika, auto industrija, brodogradnja, avio industrija, vojna industrija, medicinska oprema, itd. Područje primjene ovog pristupa nije ograničeno samo na navedene industrije, već se odnosi na razvoj softvera općenito, pa tako i na razvoj poslovnih aplikacija. Međutim, primjena „linija za proizvodnju softvera“ u razvoju poslovnih aplikacija nije uobičajena. Studije slučaja i znanstvena literatura iz ovog područja, uglavnom opisuju primjenu ovog pristupa u navedenim sektorima, dok područje primjene u razvoju poslovnih aplikacija nije dovoljno istraženo. U ovoj disertaciji definiraju se: (1) funkcionalni zahtjevi referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera, (2) implementiraju se artefakti referentne arhitekture, (3) provjerava korisnost (engl. *usefulness*) referentne arhitekture, te (4) predlažu i provjeravaju nove metrike za mjerenje utjecaja referentne arhitekture na održavljivost linije za proizvodnju softvera. Istraživanje je provedeno kroz primjenu artefakata predložene referentne arhitekture u jednoj financijskoj instituciji.

Znanstveni doprinos odnosi se unaprjeđenje referentne arhitekture poslovnih aplikacija prema pristupu linija za proizvodnju softvera s ciljem njihovog jeftinijeg i bržeg razvoja i održavanja, te povećanja njihove kvalitete. Za mjerenje kvalitete poslovnih aplikacija razvijen je i potvrđen novi model i metrike za ocjenu održavljivosti linije za proizvodnju softvera za poslovne primjene.

Ključne riječi: referentna arhitektura, linije za proizvodnju softvera, poslovne aplikacije, programski okvir, uzorci oblikovanja, poslovne komponente, metrike arhitekture.

ABSTRACT

A software product lines as a planned approach to reuse in software development, has been successfully applied in business domains such as mobile phones, home electronics, automobile industry, shipbuilding, airline industry, military industry, medical equipment, etc. The use of this approach is not limited to those industries, but it could also be applied to the software development in general, including the development of business applications, which is not usual. Case studies and scientific literature in this area, mainly describe the use of this approach in mentioned domains, while the area of the business applications development has not been addressed enough. This dissertation defines: (1) functional requirements of reference architecture for business applications based on software product lines approach, (2) implementation of the reference architecture artifacts, (3) evaluate the usefulness of the reference architecture, and (4) propose and validate novel metrics to measure the impact of reference architecture on software product lines maintainability. The research was done using the proposed reference architecture artifacts in a financial institution.

The scientific contribution is related to the improvement of reference architecture for business applications based on software product lines approach, with the goal to decrease cost and time of their development and maintenance, and increase their quality. To measure the quality of business applications we have developed and validated a new model and metrics for assessing the maintainability of the software product line for business applications.

Keywords: reference architecture, software product lines, business applications, framework, design pattern, business components, architecture metrics.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. UVOD

U uvodnom poglavlju opisuje se predmet istraživanja, motivacija autora kod izbora područja istraživanja, pregled dosadašnjih istraživanja, istraživačka pitanja, ciljevi i hipoteze, metodologija i znanstveni doprinos istraživanja te daje pregled strukture rada po poglavljima.

1.1. Predmet istraživanja

Pristup razvoju softvera pod nazivom ***linije za proizvodnju softvera*** (engl. *software product lines*) predstavlja niz softverski intenzivnih sustava, koji dijele više zajedničkih funkcionalnosti potrebnih za zadovoljavanje specifičnih potreba nekog područja, a razvijaju se na temelju zajedničkog skupa osnovnih komponenata na zadani način (Clements & Northrop, 2002b). Ovaj pristup smatra se najučinkovitijim pristupom u području ponovne upotrebe (engl. *reuse*) jer omogućava: 50% brži razvoj softvera, 70% smanjenje izvornog programskog koda, značajno smanjenje cijene koštanja, 50% smanjenje grešaka, značajno smanjenje troškova održavanja, 20-30% smanjenje troškova resursa (F. J. Linden, Schmid, & Rommes, 2010). Poslovni softver ili ***poslovne aplikacije*** predstavljaju skup računalnih programa koje koriste poslovni korisnici za obavljanje različitih poslovnih funkcija. U ovom radu, pojam poslovnih aplikacija obuhvaća aplikacije koje čine integralne informacijske sustave poduzeća i drugih sličnih organizacija koje su prevelike i kompleksne da bi ih mogli koristiti za individualne potrebe. Značajan broj projekata razvoja poslovnih aplikacija ne uspijeva ostvariti zadane vremenske, funkcionalne, te kriterije cijene i kvalitete (Abbott, 2000; Nash, 2000; Nasir & Sahibuddin, 2011; Tsai i ostali, 2011), stoga istraživačka i poslovna zajednica traže nove načine za poboljšanje produktivnosti razvoja i kvalitete poslovnih aplikacija. U tijeku razvoja potpuno novog softverskog proizvoda, moguće je koristiti postojeće artefakte ako su razvijeni na način koji omogućava njihovu ponovnu upotrebu kod razvoja novih, sličnih proizvoda. Međutim, ova mogućnost se nedovoljno koristi u praksi (Van Vliet, 2008). Porastom kompleksnosti softverskih sustava značajno raste i interes za ponovnom upotrebom artefakata od kojih se sustavi sastoje, i to već u ranoj fazi njihovog razvoja (Reinhartz-Berger, Dori, & Katz, 2009). Linije za proizvodnju softvera se koriste u poslovnim sektorima kao što su: mobilna telefonija, kućna elektronika, medicinska oprema, auto industrija; u području poslovnih aplikacija, koje je ujedno i najveće pojedinačno područje u razvoju softvera, ovaj pristup se rijetko primjenjuje. Jedan od razloga za takvo stanje je vjerojatno i u činjenici da su u početnom razdoblju tijekom uvođenja linija za proizvodnju softvera u neku organizaciju prisutni troškovi koji su značajniji

u odnosu na situaciju kada se ovaj pristup ne koristi (Cohen, 2001). Početni troškovi se odnose na razvoj referentne arhitekture, ponovno upotrebljivih komponenata, potrebnih alata i na prilagodbu organizacije novom pristupu razvoja i održavanja softvera. Nakon početne faze koja završava negdje u vrijeme razvoja trećeg ili četvrtog proizvoda (Pohl, Böckle, & van der Linden, 2005) troškovi razvoja svakog slijedećeg proizvoda se značajno snižavaju uslijed korištenja prethodno razvijenih artefakata referentne arhitekture i ponovno upotrebljivih aplikacijskih komponenata.

U ovom radu će se definirati, opisati i razviti konkretni artefakti referentne arhitekture na općenitoj razini apstrakcije za poslovne aplikacije (Cloutier i ostali, 2010), prema pristupu linija za proizvodnju softvera, koja ne rješava specifičnosti pojedinog poslovnog sektora već predstavlja horizontalni okvir koji se kasnije može proširiti i koristiti za razvoj aplikacija za pojedinačne poslovne sektore, vertikalno. Implementacija jedinstvene referentne arhitekture i njenih artefakata za poslovne aplikacije općenito ima za cilj olakšavanje razvoja poslovnih aplikacija u različitim poslovnim sektorima. Valjanost predložene referentne arhitekture provjeriti ćemo u praksi kroz studiju slučaja u jednoj financijskoj instituciji.

Prikupljene podatke metrika izvornog programskog koda na temelju studije slučaja, iskoristiti ćemo za provjeravanje i potvrdu valjanosti novih metrika i predloženog modela za mjerenje utjecaja referentne arhitekture na održavljivost (engl. maintainability) linije za proizvodnju softvera za poslovne primjene.

1.2. Motivacija za istraživanje

Glavna karakteristika današnjih i budućih softverskih sustava je stalno povećavanje njihove veličine i kompleksnosti (SEI, 2006). Što se tehnologija više razvija i pruža više mogućnosti, povećava se broj i kompleksnost funkcionalnih korisničkih zahtjeva i nefunkcionalnih karakteristika funkcija koje sustavi obavljaju kao što su distributivnost, decentralizacija, sigurnost, pouzdanost i drugo. Posljedica takve situacije je sve veća složenost procesa razvoja softverskih sustava, među kojima je najviše poslovnih sustava, što u konačnici značajno povećava troškove njihovog razvoja i održavanja.

U današnjim poslovnim modelima mnogih vrsta organizacija, softver predstavlja ključni a često i strateški element poslovnog modela. Značajan broj i kompleksnost korisničkih zahtjeva kod razvoja poslovnih aplikacija predstavlja veliki izazov za razvoj suvremenih informacijskih sustava, posebno u ograničenim vremenskim i financijskim okolnostima. Znanstveno je

potvrđeno da se pristupom linija za proizvodnju softvera mogu savladavati navedeni izazovi potičući učinkovitu ponovnu upotrebu artefakata u razvoju softvera (Clements & Northrop, 2002b; F. J. Linden i ostali, 2010; Pohl, Böckle, & Linden, 2005).

Jedna od najperspektivnijih tehnika za oblikovanje velikih i složenih softverskih sustava je definiranje njihove arhitekture (R.N. Taylor & Hoek, 2007). Arhitektura softverskog sustava je skup glavnih dizajnerskih odluka koje se odnose na softverski sustav. Glavni ciljevi korištenja softverske arhitekture odnose se na izolaciju tehničkih detalja niske razine, te da posluži kao pomoć u boljem razumijevanju sustava. Definiranje arhitekture može se odnositi samo na jedan sustav ili na više sličnih sustava koji pripadaju sličnoj poslovnoj i tehnološkoj okolini. Zajedničke karakteristike i sličnosti većeg broja sustava najprihvatljivije je definirati putem referentne arhitekture.

Glavni motiv za ovo istraživanje je današnja praksa razvoja poslovnih aplikacija koja se uglavnom temelji na dobro razvijenoj *inter-organizacijskoj* upotrebi postojećih artefakata kao što su: Eclipse 4, Eclipse Riena, Eclipse Scout programski modeli; Spring, Struts, Unisys 3D Blueprints i IBM Insurance Application Architecture okviri (engl. f*rameworks*); Hibernate, iBatis, TopLink, JPA komponente za uparivanje objekata sa relacijskim modelom; Websphere, Weblogic, Tomact aplikacijski poslužitelji, i drugo. Međutim, ponovna upotreba artefakata koji su razvijeni unutar organizacije, *intra-organizacijskih* artefakata, nije razvijena u dovoljnoj mjeri u većini organizacija (Bosch, 2009a), posebno kod razvoja poslovnih aplikacija.

Kada su u pitanju istraživanja o mogućnostima primjene pristupa linija za proizvodnju softvera u pojedinim poslovnim sektorima, uobičajena je podjela na poslovne sektore poput: telekomunikacija, medicine, proizvodnje, bankarstva, i drugo. Iz iskustva smatramo da postoji dovoljno opravdanosti da se razina apstrakcije kod razmatranja primjene navedenog pristupa ponovnoj upotrebi, premjesti sa potpuno vertikalne podjele (prethodno navedeni sektori), najprije na višu razinu poslovnih aplikacija u obliku referente arhitekture, horizontalno, a zatim na nižu razinu pojedinog sektora (npr. bankarstvo, osiguranje) vertikalno; definiranjem i razvojem specifičnih ponovno upotrebljivih komponenata za svaki vertikalni sektor posebno. Prepoznavanje poslovnih aplikacija na generičkoj razini, kao domene za primjenu linija za proizvodnju softvera, bez isključivog uzimanja u obzir konkretnih poslovnih sektora kod definiranja funkcionalnih zahtjeva njihove referentne arhitekture, omogućava razvoj standardnih funkcionalnosti referentne arhitekture koja je zajednička većini poslovnih aplikacija. Standardne funkcionalnosti se odnose na: upravljanje transakcijama, upravljanje izvorima podataka, pristup izvorima podataka, komunikaciju između sustava, sigurnost,

validaciju podataka, internacionalizaciju, čuvanje podataka (engl. *caching*), enkripciju, oblikovanje i izgled grafičkih sučelja, formatiranje izvješća, interoperabilnost, itd.

Dodatna motivacija kod izbora područja ovog istraživanja potaknuta je nedostatkom referentnih arhitektura koje eksplicitno podržavaju razvoj poslovnih aplikacija na generičkoj horizontalnoj razini apstrakcije. Izvor motivacije je i prilično uobičajeni neučinkovit način podjele artefakata i aktivnosti kod razvoja poslovnih aplikacija koji ne uzima u obzir prednosti podjele rada razvojnih programera na klijent-poslužitelj principu te organizaciju procesa razvoja na principu aplikacije-infrastruktura.

Kako bi se pokušalo dati odgovor na neka od motivacijskih pitanja, a uslijed nedostatka primjera primjene prikladnih metrika za mjerenje kvalitete artefakata referentne arhitekture u kontekstu primjene pristupa linija za proizvodnju, u ovom radu predložiti će se upotreba metrika pod nazivom „p*latform responsibility*“ za mjerenje održavljivosti komponenata poslovnih aplikacija, indirektno i mjerenja kvalitete referentne arhitekture. Navedene metrike se mogu koristiti kao „rani“ indikator buduće kvalitete poslovnih aplikacija koje se temelje na nekoj referentnoj arhitekturi, a u cilju poboljšanja njihove kvalitete te predviđanja i smanjivanja resursa potrebnih za održavanje poslovnih aplikacija.

1.3. Dosadašnja istraživanja

Koncept korištenja linija za proizvodnju softvera prvi je predložio D.L. Parnas 1976. godine (Parnas, 1976), na temelju ideje E.W. Dijkstre iz 1970. godine (Dijkstra, Dijkstra, & Dijkstra, 1970). Značajnija istraživanja područja linija za proizvodnju softvera počela su razvojem tehnologija koje omogućavaju ponovnu iskoristivost programskog koda i razvoj temeljen na komponentama. Prvi značajniji rezultat istraživanja u ovom području je definiranje *Feature Oriented Domain Analysis* (FODA) (Kang, Cohen, Hess, Novak, & Peterson, 1990) metode za analizu sustava s ciljem poboljšanja ponovne upotrebe arhitekture i komponenata sustava. U Europi je pokrenuto nekoliko znanstvenih projekata s ciljem unaprjeđenja razvoja pristupom linija za proizvodnju softvera: ARES, Praise, ESAPS, CAFE i FAMILIES (F. Van der Linden, 2002). U američkom Software Engineering Institute (SEI) provedena su značajna istraživanja za potrebe ministarstva obrane (Clements & Northrop, 2002b). Međutim, u počecima su se organizacije više fokusirale na procesne aspekte i uvođenje samog pristupa u organizacije a manje na tehnološke aspekte i učinkovitost linija za proizvodnju softverskih proizvoda (Clements & Northrop, 2002b). Proteklih desetak godina nekoliko većih organizacija počelo je koristiti pristup linija za proizvodnju softvera u procesu razvoja programskih proizvoda.

Područje razvoja poslovnih aplikacija pristupom linija za proizvodnju softvera nije bilo predmet znanstvenih istraživanja u mjeri koliko se to odnosi na ostala područja. Osim organizacijskih troškova, najveći izazov kod uvođenja ovog pristupa u organizacije odnosi se na razvoj referentne arhitekture (engl. *reference architecture*) kao temeljnog artefakta za razvoj novih proizvoda. Referentna arhitektura za poslovne aplikacije sadrži glavne komponente za buduće aplikacije te definira njihov odnos korištenjem slojeva (engl. *layers*) u kojima su te komponente implementirane (Brunner & Zimmermann, 2012). Referentna arhitektura linije za proizvodnju softvera u biti služi za implementaciju funkcionalnosti (engl. *features: aspects, capability*) koje se koriste u više proizvoda u kasnijim fazama. Proizvodi koji uključuju određene karakteristike konfiguriraju se tako što se u njih ugrađuju slojevi referentne arhitekture unutar kojih su te karakteristike implementirane (Batory, Cardone, & Smaragdakis, 2000). U centru same referentne arhitekture je razvojni okvir (engl. *framework*) kojeg čini skup uglavnom apstraktnih klasa u kojima se implementiraju zajednički algoritmi aplikacijskih komponenata koje se temelje na tom okviru (Johnson & Foote, 1988). Primjena okvira kod linija za proizvodnju softvera u području poslovnih aplikacija omogućava razdvajanje ponovno upotrebljivih komponenata na generičku razinu za korištenje u više poslovnih sektora, te na razinu specifičnog poslovnog sektora. Korištenje prethodno razvijenih zajedničkih i varijabilnih komponenata (engl. *commonality and variability*) u razvoju proizvoda omogućava se na način da ih budući proizvodi mogu putem konfiguriranja koristiti ili izostaviti.

Pristup linija za proizvodnju softvera sve više se koristi za „*mission-critical*“ i „*safety-critical*“ sustave u automobilskoj i avionskoj industriji (Weiss, 2008). Linije za proizvodnju softvera sve više se razvijaju i na način koji obuhvaća više organizacija (Bosch, 2009a). Alati za konfiguraciju proizvoda razvijaju se u smjeru koji obuhvaća više repozitorija u kojima se čuvaju modeli varijabilnosti i koji pripadaju različitim organizacijama (Dhungana i ostali, 2011). Istraživanje na primjeru četiri organizacije koje su implementirale *Enterprise resource planning* (ERP) poslovne sustave (Hamza, Martinez, & Alonso, 2010) pokazalo je da je veliki broj zajedničkih funkcija u ERP sustavima moguće definirati na većoj razini apstrakcije, na razini ERP domene prema pristupu linija za proizvodnju softvera. Osim istraživanja mogućnosti primjene linija za proizvodnju softvera u ERP domeni, ostala slična istraživanja (Czarnecki, Grünbacher, Rabiser, Schmid, & Wąsowski, 2012; Dhungana i ostali, 2011; Hamza i ostali, 2010; Nöbauer, Seyff, Dhungana, & Stoiber, 2012) uglavnom se odnose na područje konfiguriranja komponenata poslovnih informacijskih sustava koji koriste ERP za potporu standardnim poslovnim procesima u organizacijama. Iz navedenih istraživanja može se

zaključiti da postoji potreba daljnjeg istraživanja primjene pristupa linija za proizvodnju softvera u razvoju poslovnih aplikacija, što je ujedno i glavni cilj ovog istraživanja.

Najviše istraživanja, osim onih koja se odnose na strukturne aspekte referentne arhitekture, odnosi se na metrike koje se mogu koristiti u pojedinim fazama životnog ciklusa linije za proizvodnju softvera. Metrike za mjerenje atributa kvalitete referentne arhitekture kao što je metrika kvalitete dizajna (Martin, 1994a), metrike za mjerenje održavljivosti strukture i arhitekture linije za proizvodnju softvera (Rahman, 2004), te metrike kompleksnosti (Junior, Gimenes, & Maldonado, 2013) ne obuhvaćaju i područje kvalitete same implementacije referentne arhitekture i njene odgovornosti u preuzimanju kompleksnosti na sebe kada je u pitanju kompleksnost aplikacija koji ju koriste. Stoga se u ovom radu predlažu nove metrike za mjerenje odgovornosti referentne arhitekture za kvalitetu aplikacija koje se na njoj temelje.

U području razvoja poslovnih aplikacija pristupom linija za proizvodnju softvera, važan moment je takozvana „rana faza“, prije razvoja većeg broja aplikacija, u kojoj se mogu koristiti metrike mjerenja kvalitete referentne arhitekture kako bi se otklonili rizici eventualnog razvoja novih proizvoda na slaboj podlozi, ukoliko referentna arhitektura ne zadovoljava potrebne karakteristike. Predložene nove metrike pod nazivom odgovornost okvira platforme (engl. *platform responsibility*) namijenjene su za mjerenje kvalitete referentne arhitekture, ranog indikatora buduće kvalitete proizvoda koji se temelje na referentnoj arhitekturi.

Vjerojatni razlozi male primjene linija za proizvodnju softvera, kao pristupa razvoju poslovnih aplikacija, možda leže i u malom broju znanstvenih istraživanja koje se odnose na ovo područje. Znanstvena istraživanja ovog pristupa u području razvoja poslovnih aplikacija, a posebno istraživanja specifičnih metrika i referentne arhitekture, mogu biti značajan poticaj većoj primjeni linija za proizvodnju softvera u praksi.

1.4. Istraživačka pitanja

Motivacija i dosadašnja istraživanja ukazuju na potrebu razvoja referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera. Poslovne aplikacije različitih poslovnih sektora imaju zajedničke karakteristike koje se mogu definirati u okviru jedinstvene referentne.

Slijedeća istraživačka pitanja predstavljaju središnji dio ovog istraživanja na koja ćemo odgovor saznati definiranjem funkcionalnih zahtjeva, implementacijom predloženih artefakata i provjerom valjanosti implementiranih artefakata i predloženih metrika referentne arhitekture:

(P1) *Koje funkcionalne zahtjeve treba zadovoljiti referentna arhitektura za poslovne aplikacije prema pristupu linija za proizvodnju softvera?*

(P2) *Može li se razviti referentna arhitektura koja zadovoljava te zahtjeve?*

(P3) *Da li je referentna arhitektura koja zadovoljava te zahtjeve korisna u praksi?*

(P4) *Postoji li povezanost između predloženog modela PR (engl. Platform Responsibility) za mjerenje održavljivosti linije za proizvodnju softvera za poslovne aplikacije i standardnog modela MI (engl. Maintainability Indeks), kada se oba modela koriste za mjerenje održavljivosti identičnih softverskih komponenata?*

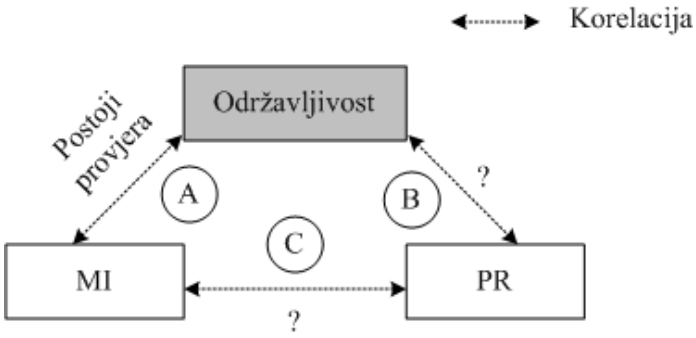
Odgovor na prvo istraživačko pitanje (P1) temeljiti će se na istraživanju postojećeg teorijskog okvira i analizi postojećih referentnih arhitektura iz područja razvoja poslovnih aplikacija, uključujući i praktično iskustvo autora u razvoju poslovnih aplikacija u nekoliko poslovnih sektora. Dodatno, prikupljanje informacija na temelju ispitivanja stručnjaka putem anketnog upitnika koji imaju iskustvo u istraživanju i praksi, povećati će pouzdanost u relevantnost definiranih funkcionalnih zahtjeva referentne arhitekture.

Drugo istraživačko pitanje (P2) odnosi na oblikovanje i razvoj artefakata referentne arhitekture, prvenstveno programskog okvira (engl. *framework*), te ostalih manje zahtjevnih artefakata kao što su pomoćni alat (engl. *plugin*), opis arhitekture (engl. *architectural description*) i referentna aplikacija (engl. *reference application*).

Odgovor na treće istraživačko pitanje (P3) svakako predstavlja najveći izazov. Jedini način da se dokaže korisnost predloženog pristupa odnosi se na primjenu razvijenih artefakata referentne arhitekture u praksi metodom studije slučaja i prikupljanjem podataka. Provjera korisnosti (engl. *usefulness*) odnosi se na dvije vrste provjere; upotrebljivosti (engl. *usability*) i primjenjivosti (engl. *utility*): „*Primjenjivost sustava odnosi se na pitanje da li sustav može obavljati funkcije koje treba obavljati, a upotrebljivost na pitanje kvalitete obavljanja tih funkcija za korisnika*“ (Nielsen, 1994). Provjera primjenjivosti artefakata referentne arhitekture provesti će se metodom studije slučaja u jednoj financijskoj instituciji, dok će se provjera upotrebljivosti ispitati putem metode kontroliranog eksperimenta kojeg ćemo provesti među korisnicima artefakata referentne arhitekture, razvojnim programerima poslovnih aplikacija.

Slika 1 ilustrira postupak koji koristimo kod četvrtog istraživačkog pitanja (P4); odnosi se na mjerenje povezanosti između predloženog modela PR i standardnog modela MI za provjeru

održavljivosti linije za proizvodnju softvera. Povezanost između održavljivosti i MI (označena kao A) znanstveno je provjerena (Oppedijk, 2008).



Slika 1 MI i PR metrike u odnosu na održivost sustava

Glavna namjera razvoja vlastitog modela (PR) je njegova upotreba za provjeru održavljivosti aplikacijskih komponenata linije za proizvodnju softvera. Za očekivati je da postoji slična korelacija između održavljivosti i PR modela (označena kao B) kao što postoji između održavljivosti i MI modela (označena kao A). Ukoliko se to potvrdi, pretpostavka je da postoji statistički značajna povezanost rezultata mjerenja MI i PR modela (označena kao C). Statistička analiza odnosa između MI i PR modela, (označeno kao C) je predmet ovog istraživačkog pitanja. Operacionalizacija koncepta mjerenja uključuje upotrebu PR i MI metrika kao dviju zavisnih varijabli te određivanje nekoliko standardnih metrika izvornog programskog koda koje će se koristiti kao nezavisne varijable.

1.5. Ciljevi i hipoteze istraživanja

Na temelju prethodnih istraživanja autora, istraživanja literature i postojećih referentnih arhitektura čije je kratki pregled dan u uvodnom poglavlju, a detaljnije je obrađen u poglavljima 2 i 3, te na temelju iskustva autora u području istraživanja, identificirani su ciljevi ovog doktorskog rada. Glavni cilj ovog istraživanja je razvoj i provjera artefakata i metrika referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera. Cilj **C1** odnosi se na funkcionalne zahtjeve predložene referentne arhitekture:

> (**C1**) Definirati i provjeriti funkcionalnosti i strukturne aspekte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera.

(**C1.1**) Analizirati postojeća istraživanja iz područja referentnih arhitektura za poslovne aplikacije i identificirati postojeće izazove za referentne arhitekture.

(**C1.2**) Analizirati postojeće referentne arhitekture za poslovne aplikacije.

(**C1.3**) Definirati funkcionalnosti referentne arhitekture.

(**C1.4**) Ispitati mišljenje iskusnih arhitekata i istraživača o relevantnosti definiranih funkcionalnosti referentne arhitekture.

Na temelju prethodno definiranih funkcionalnosti, slijedeći cilj **C2** se odnosi na implementaciju predložene referentne arhitekture:

(**C2**) Odrediti i razviti artefakte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera koji zadovoljavaju definirane funkcionalne zahtjeve (funkcionalnosti).

(**C2.1**) Opisati razvijene artefakte referentne arhitekture.

(**C2.2**) Prikazati način implementacije definiranih funkcionalnosti kroz razvoj artefakata referentne arhitekture.

Da bi se procijenila valjanost ovog istraživanja, razvijene artefakte treba primijeniti u praksi te analizirati njihovu korisnost (engl. *usefulness*). To je ujedno i motivacija za postavljanje **C3**:

(**C3**) Provjeriti razvijene artefakte referentne arhitekture i njihovu korisnost u praksi.

(**C3.1**) Provjeriti upotrebljivost (engl. *usability*) razvijenih artefakata referentne arhitekture.

(**C3.2**) Primijeniti i koristiti razvijene artefakte referentne arhitekture u praksi putem longitudinalne studije slučaja s ciljem provjere njihove primjenjivosti (engl. *utility*).

(**C3.3**) Postići zrelost referentne arhitekture razine „4“ (engl. *Variants Products*) prema modelu za zrelost linija za proizvodnju softvera (engl. *Family Evaluation Framework*) (F. Van Der Linden, Bosch, Kamsties, Känsälä, & Obbink, 2004).

Predloženi model i metrike za procjenu održavljivosti aplikacijskih poslovnih komponenata koje primjenjuju artefakte referentne arhitekture motivacija su za postavljanje cilja **C4**:

(**C4**) Provjeriti povezanost rezultata PR (engl. *Platform Responsibility*) i MI (engl. *Maintainability Index*) modela za mjerenje održavljivosti aplikacijskih komponenata linije za proizvodnju softvera.

(**C4.1**) Izmjeriti jakost veza među modelima PR i MI za mjerenje održavljivosti aplikacijskih poslovnih komponenata u jednoj financijskoj instituciji.

(**C4.2**) Pronalaženje adekvatnog modela (linearne kombinacije) koji se sastoji od skupa nezavisnih varijabli (metrika izvornog programskog koda) koje procjenjuju nepoznatu regresijsku funkciju zavisne varijable PR.

Na temelju navedenih ciljeva postavljene su dvije hipoteze koje provjeravamo:

H1: Nova referentna arhitektura prema pristupu linija za proizvodnju softvera je korisna (engl. *usefull*) za proizvodnju softvera za poslovne primjene.

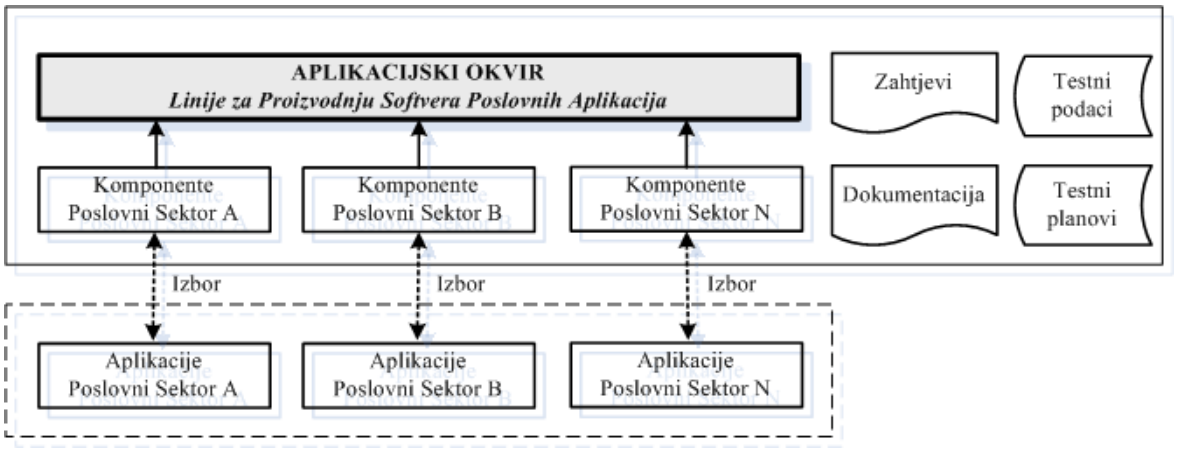
H2: Rezultati određivanja održavljivosti poslovnih komponenata (engl. *maintainability*) informacijskog sustava metrikom „odgovornosti platforme“ (engl. *Platform Responsibility*) podudaraju se s rezultatima dobivenim uz pomoć alternativne metrike.

1.6. Kontekst istraživanja

Linija za proizvodnju softvera sastoji se od različitih ponovno upotrebljivih elemenata (artefakata), kao što su: aplikacijski okvir, poslovne komponente, testni planovi, dokumentacija, itd. Navedeni elementi pripadaju različitim logičkim razinama, primjerice, zajedničkoj referentnoj arhitekturi, nekom poslovnom sektoru, poslovnim aplikacijama, itd. Glavni artefakt predložene referentne arhitekture je ***aplikacijski okvir*** kojeg zajednički dijeli više poslovnih sektora; predstavlja predložak arhitekture svih poslovnih aplikacija. Aplikacijski okvir može se definirati samo za jedan poslovni sektor, što je uobičajeno, međutim, veliki broj zajedničkih funkcionalnosti koje imaju poslovne aplikacije koje pripadaju različitim poslovnim sektorima postaju redundantne ukoliko se poslovne aplikacije razvijaju u istoj organizaciji. Broj aplikacija koje se razvijaju u istoj organizaciji i broj korisničkih zahtjeva za izmjenama brzo se povećava; razvoj i održavanje velikog broja aplikacija može postati vrlo kompleksan i skup. Glavni izazov u takvoj situaciji predstavlja definiranje, razvoj i primjena aplikacijskog okvira u kontekstu njegove upotrebe za više poslovnih sektora (slika 2).

Glavni fokus ovog istraživanja, aplikacijski okvir u navedenom kontekstu, trebao bi biti jednostavan za upotrebu kod razvoja aplikacija u više poslovnih sektora, smanjiti kompleksnost razvoja poslovnih aplikacija preuzimanjem „odgovornosti“ za interakciju sa vanjskim sustavima, omogućiti jednostavniju evoluciju poslovnih aplikacija uslijed učestalih korisničkih zahtjeva i upotrebe novih tehnologija, poboljšati održavljivost aplikacija koje ga primjenjuju.

U poglavlju 5 opisali smo predloženi aplikacijski okvir, a u poglavlju 6 opisali smo njegovu upotrebu kroz studiju slučaja koja obuhvaća 16 poslovnih aplikacija.



Slika 2 Fokus ovog istraživanja (aplikacijski okvir)

1.7. Metodologija istraživanja

Osim suvremenih metoda i tehnika programskog inženjerstva, u istraživanju će se koristiti i opće istraživačke metode: ***znanost o oblikovanju*** (engl. *design science*) (Hevner, March, Park, & Ram, 2004a; V. K. Vaishnavi & Kuechler Jr, 2007; V. Vaishnavi & Kuechler, 2004), studija slučaja, metoda ispitivanja i eksperiment. Postupak istraživanja se temelji na općem okviru ciklusa oblikovanja (engl. *general design cycle framework*) (V. Vaishnavi & Kuechler, 2004) koji će se koristiti kao predložak procesa istraživanja u informacijskim sustavima. Proces istraživanja odvija se u ponavljajućim koracima: ***definiranje problema, prijedlog rješavanja, razvoj artefakata, provjera artefakata, zaključak***. Za svaki procesni korak koristi se jedan ili više uzoraka (engl. *pattern*) koji se odnose na metodu *znanost o oblikovanju* (V. K. Vaishnavi & Kuechler Jr, 2007) a koji u biti opisuju detaljnu proceduru koju je potrebno slijediti u tijeku istraživanja. Detaljna razrada metodologije i nacrta ovog istraživanja opisana je u poglavlju 4 „Nacrt istraživanja“.

1.8. Znanstveni doprinosi

Temeljni znanstveni doprinosi ove disertacije odnose se na istraživačko područje linija za proizvodnju softvera i na organizacije koje namjeravaju ili već koriste taj pristup pri razvoju poslovnih aplikacija.

Cilj ove disertacije je pomaknuti istraživanje referentnih arhitektura za poslovne aplikacije, jedan korak dalje. Glavni znanstveni doprinos ovog rada koji predstavlja nadogradnju prethodnih istraživanja, odnosi se na razvijenu i poboljšanu referentnu arhitekturu za poslovne aplikacije različitih poslovnih sektora. Referentna arhitektura se temelji na postojećim teorijskim osnovama koje su primjenom suvremenih metoda i pristupa za razvoj korištene za definiranje, oblikovanje, razvoj i validaciju njenih artefakata s krajnjim ciljem da budu:

- Pojednostavljeni, fleksibilni, proširivi i otvoreni.
- Općeniti u odnosu na različite poslovne sektore.
- Prilagođeni načinu rada dionika procesa razvoja poslovnih aplikacija.
- Neovisni o vrsti programskih alata koji se koriste u razvoju poslovnih aplikacija.

Jednostavnost i fleksibilnost omogućavaju lakšu, jeftiniju i bržu nadogradnju i prilagodbu poslovnih aplikacija novim tehnologijama, programskim alatima, te komponentama drugih dobavljača koje se ubrzano i često mijenjaju. Istraživanje u cilju definiranja, razvoja i provjere predložene referentne arhitekture uključuje analizu teorijskog područja, te sintezu prikupljenih rezultata i praktičnih spoznaja. Ključni znanstveni doprinosi ovog istraživanja su:

- Pregled literature, istraživanja i stanja u području referentnih arhitektura za poslovne aplikacije, te u području pristupa i metoda razvoja uz pomoć ponovno iskoristivih komponenata. Provedena analiza može pomoći drugim istraživačima pri pronalaženju postojećih istraživanja u ovom području.
- Jasno definirane i provjerene funkcionalnosti referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera, horizontalno, za više poslovnih sektora, novo i različito od uobičajene primjene pristupa samo za jedan poslovni sektor. Definirani funkcionalni i strukturni aspekti mogu koristiti drugim istraživačima koji istražuju razvoj poslovnih aplikacija prema pristupu linija za proizvodnju softvera, te za praktičare koji žele provjeriti korisnost razvijenih artefakata ili predloženog pristupa.

- Razvijeni artefakti (okvir, referentna aplikacija, opis arhitekture, alat), njihova primjena u jednom poslovnom sektoru, vertikalno, kao primjer obuhvaćanja specifičnosti jednog poslovnog sektora pomoću zajedničke horizontalne arhitekture. Organizacije koje razvijaju poslovne aplikacije prema pristupu linija za proizvodnju softvera, u različitim sektorima, mogu primijeniti razvijene artefakte za vlastiti razvoj.
- Razvoj i potvrda novog modela i metrika za ocjenu održavljivosti linije za proizvodnju softvera za poslovne aplikacije u odnosu na način upotrebe i stupanj korištenja referentne arhitekture.
- Provjera primjenjivosti referentne arhitekture na studiji slučaja primjene u razvoju poslovnih aplikacija. Predloženi pristup razvoju poslovnih aplikacija i razvijeni artefakti su korisni u praksi, što je potvrđeno u studiji slučaja. Rezultati studije slučaja mogu biti zanimljivi istraživačima u ovom području i organizacijama koje namjeravaju koristiti pristup linija za proizvodnju softvera za razvoj poslovnih aplikacija.
- Unaprjeđenje referentne arhitekture poslovnih aplikacija na temelju provođenja nekoliko ciklusa: definiranja, oblikovanja, razvoja i provjere artefakata s ciljem omogućavanja njenog korištenja pri razvoju poslovnih aplikacija.

Disertacija se ne odnosi na slijedeće:

- Referentnu arhitekturu svih poslovnih aplikacija. Današnji poslovni modeli pojedinih organizacija mogu uključivati i aplikacije koje imaju specifične tehničke i poslovne zahtjeve koji nisu obuhvaćeni ovim istraživanjem.

1.9. Struktura disertacije

Istraživanje referentne arhitekture za poslovne aplikacije u ovoj disertaciji temelji se na metodi *znanost o oblikovanju* što direktno utječe i na samu strukturu disertacije. Kao što je navedeno u poglavlju 1.5, glavni ciljevi ovog istraživanja odnose se na definiranje, razvoj i provjeru referentne arhitekture, što ujedno određuje artefakte ovog istraživanja koji se pojavljuju u obliku koncepta, modela i primjerka referentne arhitekture.

Tablica 1 prikazuje strukturu disertacije koja je podijeljena u četiri cjeline prema metodologiji istraživanja opisanoj u poglavlju 1.7.

Cjelina I proteže se kroz drugo i treće poglavlje a odnosi se na teorijski okvir istraživanja u kojem su objašnjeni osnovni pojmovi u područjima razvoja poslovnih aplikacija i pristupa njihovom razvoju koji se zove linija za proizvodnju softvera.

Cjelina II obuhvaća definiranje funkcionalnih zahtjeva (funkcionalnosti) i strukturnih aspekata referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera, prijedlog njihovog rješavanja te njihovo oblikovanje i razvoj.

Cjelina III podijeljena je u pet dijelova, opisuje validaciju referentne arhitekture i dovođenje u vezu s hipotezama koje su postavljene. Prvi dio opisuje eksperiment kojim se ispituje upotrebljivost referentne arhitekture. Drugi dio se odnosi na studiju slučaja primjene referentne arhitekture u jednoj financijskoj instituciji. Treći dio odnosi se na ocjenjivanje evolucije okvira referentne arhitekture kao temelja za procjenu pogodnosti za njegovo poboljšanje. Četvrti dio opisuje primjenu novog modela za predviđanje održavljivosti (promjenjivosti) aplikacijskih komponenata linije za proizvodnju softvera upotrebom metrika izvornog programskog koda. Peti dio obuhvaća validaciju referentne arhitekture prema jednom od modela zrelosti za referentnu arhitekturu linija za proizvodnju softvera.

Cjelina IV odnosi se na sažetak rezultata provedenog istraživanja, navođenje konačnih zaključaka izvedenih u istraživanju, raspravu o znanstvenim doprinosima istraživanja u cilju boljeg razumijevanja istraživanja, njegova ograničenja, te prijedlog budućih istraživanja.

Tablica 1 Struktura disertacije po cjelinama

Cjelina	**Naziv poglavlja**	**Broj poglavlja**
I. Teorijski okvir	Poslovne aplikacije	2
	Linije za proizvodnju softvera	3
II. Razvoj referentne arhitekture na temelju provedenog istraživanja	Razvoj referentne arhitekture	5
III. Validacija referentne arhitekture	Validacija	6
IV. Zaključak	Zaključak	7

2. POSLOVNE APLIKACIJE

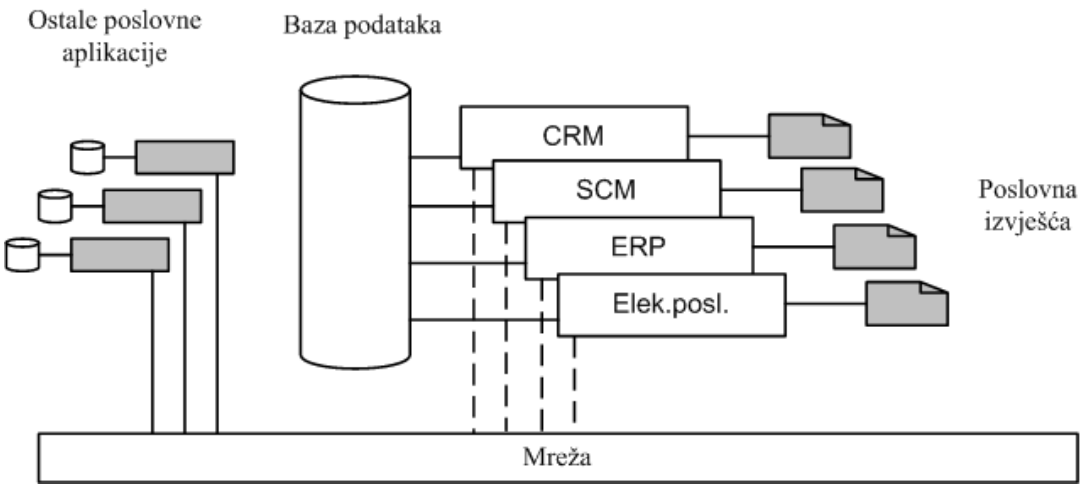
U ovom poglavlju određujemo osnovne pojmove područja poslovnih aplikacija, povijesni razvoj poslovnih aplikacija; pojašnjavamo važnost arhitekture i referentne arhitekture kao temeljne organizacije informacijskih sustava, način definiranja i razvoja referentne arhitekture; opisujemo metode za provjeru kvalitete arhitekture i referentne arhitekture; opisujemo životni ciklus poslovnih aplikacija te otkrivamo prepoznatljive trendove u razvoju poslovnih aplikacija.

Svrha ovoga poglavlja je istraživanje metoda razvoja i provjere kvalitete nove referentne arhitekture čiji razvoj i validacija je glavni cilj ovoga rada, što je direktno povezano sa slijedećim postavljenim ciljevima: (C1.1) analizirati postojeća istraživanja iz područja referentnih arhitektura za poslovne aplikacije i identificirati postojeće izazove za referentne arhitekture, (C1.2) analizirati postojeće referentne arhitekture za poslovne aplikacije, (C1.3) definirati funkcionalnosti referentne arhitekture, (C2.1) opisati razvijene artefakte referentne arhitekture, (C4.1) izmjeriti jakost veza među modelima PR i MI za mjerenje održavljivosti aplikacijskih poslovnih komponenata u jednoj financijskoj instituciji.

2.1. Opis i definicija poslovnih aplikacija

Poslovne aplikacije obuhvaćaju standardne poslovne sustave za podršku temeljnim poslovnim procesima kao što su: *Enterprise Resource Planning* (ERP) koji se odnosi na poslovne informacijske sustave koji su dizajnirani s ciljem integracije i optimizacije poslovnih procesa i transakcija u organizacijama; *Customer Relationship Management* (CRM) sustav za upravljanje odnosima s kupcima; *Supply Chain Management* (SCM) sustav za upravljanje nabavnim lancem; sustave za elektroničko poslovanje.

Osim navedenih standardnih poslovnih sustava koji obuhvaćaju svega 70 posto potreba organizacija (Bosilj Vukšić & Kovačić, 2004, str. 58), poslovne aplikacije obuhvaćaju i ostale slične i/ili pomoćne aplikacije koje se integriraju u jedinstveni informacijski sustav organizacija; uglavnom imaju korisnička sučelja, poslovnu logiku, baze podataka, sastoje se od logičkih zadataka ili transakcija (engl. *logical unit of work*), oblikovane su na način da se mogu integrirati ili povezati sa drugim aplikacijama, primjenjuju se u različitim mrežnim okruženjima (*intranet, internet*), osiguravanju primjerene sigurnosne karakteristike i administraciju; koriste se za podršku i povezivanje poslovnih procesa u organizacijama, kao i za vanjske poslovne procese kojima se organizacije povezuju s ostalim poslovnim partnerima.



Slika 3 Kontekst pojma poslovnih aplikacija[1]

Poslovni procesi mogu se opisati kao niz logički povezanih aktivnosti koje koriste resurse poduzeća, a čiji je krajnji cilj zadovoljenje potreba kupaca za proizvodima ili uslugama odgovarajuće kvalitete i cijene, u adekvatnom vremenskom roku, uz istodobno ostvarivanje neke vrijednosti (Bosilj Vukšić & Kovačić, 2004, str. 9).

Konkurentnost modernih poslovnih organizacija značajno ovisi o poslovnom softveru i informacijsko komunikacijskoj tehnologiji. Glavna motivacija za korištenje poslovnog softvera proizlazi iz mogućnosti da se poveća profit smanjivanjem troškova ili ubrzavanjem poslovnog ciklusa, pojača financijska kontrola, poboljša pouzdanost poslovnih procesa. Razlikujemo dvije glavne vrste poslovnih aplikacija:

- Interaktivne: koje imaju korisničko sučelje pomoću kojega korisnici unose, mijenjaju, ili čitaju poslovne podatke. Odzivno vrijeme nakon svake korisničke interakcije sa sučeljem poslovne aplikacije je relativno kratko, svega nekoliko sekundi. Konstantna raspoloživost, očuvanje integriteta podataka i dobre performanse, glavne su značajke ove vrste poslovnih aplikacija.
- *Batch*: poslovne aplikacije koje se izvode relativno duže od interaktivnih aplikacija; bez potrebe za korisničkom interakcijom, pri čemu se sekvencijalno izvršava jedan ili više programa koji učitavaju ulazne datoteke, obrađuju poslovne podatke, te stvaraju nove ili ažuriraju postojeće izlazne datoteke. *Batch* programi se najčešće izvode noću i obrađuju podatke koji se najčešće po danu prikupljaju putem interaktivnih aplikacija.

[1] http://russellobrien.hubpages.com – prilagođeno ovom istraživanju

Interaktivne aplikacije pojavile su se negdje 1960-tih, najprije u obliku komandnih linija, zatim u obliku tekstualnih hiperveza (engl. *text-base hyperlinks*) te u konačnici u obliku grafičkih elemenata (engl. *windows, menus, radio buttons, check boxes*).

Batch aplikacije su se pojavile prije interaktivnih, negdje 1950-tih godina, najčešće su se koristile za obradu računovodstvenih podataka. Usprkos njihovoj dugoj povijesti, *batch* aplikacije i dalje imaju važnu ulogu u većini poslovnih organizacija. I najnoviji sustavi obično sadrže bar jednu ili više *batch* aplikacija za obradu poslovnih podataka, generiranje izvješća, ispis dokumenata, itd.

Poslovne aplikacije u početku su se uglavnom razvijale u poslovnim organizacijama koje su ih koristile, najčešće tako što bi programeri proučili poslovne zahtjeve, zatim bi prema njihovom razumijevanju poslovnih problema razvijali aplikacije. Tijekom vremena, proces razvoja polovnih aplikacija se izmijenio, tako da glavnu ulogu u tom procesu igraju poslovni korisnici umjesto programera. Danas se aplikacije uglavnom kupuju, naručuju prema specifičnim zahtjevima ili unajmljuju od velikih proizvođača softvera ili pružatelja usluga; oprema za izvođenje poslovnih aplikacija i čuvanje poslovnih podataka, u početku se nalazila na lokacijama poslovnih organizacija koje su opremu koristile, kasnije zbog razvoja tehnologije i poslovnih ušteda, na njihovim centralnim lokacijama, danas sve više na iznajmljenim lokacijama u drugim organizacijama koje su specijalizirane za pružanje usluga poslužitelja na udaljenoj lokaciji (engl. *cloud hosting*).

Način korištenja i vrste poslovnih aplikacija koje organizacije u svom poslovanju upotrebljavaju najčešće ovisi o veličini poslovne organizacije:

- Male organizacije uglavnom koriste unaprijed razvijene i na tržištu raspoložive aplikacije za: računovodstvo, uredsko poslovanje, komunikaciju itd.
- Srednje organizacije koriste više različitih aplikacija, od računovodstvenih, aplikacija za upravljanje odnosima s klijentima, upravljanje ljudskim potencijalima, trgovinu, odobravanje kredita itd.
- Velike organizacije (engl. *enterprise*) osim navedenih, koriste i aplikacije za upravljanje poslovnim procesima, upravljanje resursima organizacija (engl. *enterprise resource planning*) itd.

2.2. Povijesni razvoj poslovnih aplikacija

Poslovne aplikacije pojavljuju se i koriste u praksi od 1950-tih godina, te od tada do danas neprestano evoluiraju u tehnološkom i u pogledu uloge koju imaju u poslovanju organizacija. U prošlosti se informatika uglavnom koristila kao tehnička podrška redovitom poslovanju, danas ju smatramo partnerom poslovnom dijelu organizacije i važnim sredstvom za ostvarivanje strateških poslovnih ciljeva. Znanstveno područje čiji predmet istraživanja su poslovne aplikacije, pojavljuje se 1960-tih godina, najprije pod nazivom *Management Information Systems* (MIS), a kasnije i sve do danas pod nazivom *Information Systems* (IS). Informacijski sustavi (IS) nastali su kao spoj računalnih znanosti (engl. *computer science*), menadžmenta, teorije organizacije, operacijskih istraživanja i računovodstva (G. B. Davis & Olson, 1984)[2]. Svako od navedenih sastavnih područja ili disciplina ima svojstven utjecaj na primjenu informacijske tehnologije u organizacijama, međutim, niti jedno od njih se nije bavilo isključivo njenom primjenom. Informacijski sustavi, kao novo znanstveno područje, nastalo je upravo za to da proučava primjenu informacijskih tehnologija u organizacijama. Razvoj i rast područja informacijskih sustava kroz nekoliko zadnjih desetljeća, očituje se na razne načine. Primjerice, paralelno sa rastom informacijskih sustava, pojavljivale su se i istraživačke zajednice koje se bave istraživanjem informacijskih sustava, a intenzitet istraživanja se je naglo povećavao. Takav razvoj doveo je do pojave novih znanstvenih časopisa, konferencija, mnoštva knjiga i literature o informacijskim sustavima. Uz takav intenzivan rast jednog znanstvenog područja, bilo je za očekivati da će se područje informacijskih sustava utvrditi kao takvo u akademskom i praktičnom smislu, međutim to se još nije u potpunosti dogodilo (Hirschheim & Heinz, 2010).

Povijesni razvoj poslovnih aplikacija usko je povezan sa znanstvenim područjem informacijskih sustava; pojava novih tehnologija izravno je utjecala na poslovne potrebe organizacija, i obrnuto, potrebe poslovnih organizacija utjecale su na područja istraživanja informacijskih sustava. Povijest poslovnih aplikacija možemo podijeliti u nekoliko razdoblja, ovisno o tehnologiji, načinu njihovog razvijanja ili korištenja. Pojavom nove tehnologije i eventualno novog načina njene primjene u poslovnim aplikacijama, ne znači ujedno i da se starije poslovne aplikacije više neće koristiti u praksi; neka poslovna rješenja na starijim tehnologijama se još uvijek koriste u mnogim organizacijama. Povijesna razdoblja razvoja i

[2] Ostala područja za koja se smartra da su značajno utjecala na razvoj ovog područja su kibernetika i teorija sustava. Discipline poput sociologije, ekonomije, psihologije, antropologije, političkih znanosti, filozofije i arhitekture su također imale značajan utjecaj na razvoj informacijskih sustava (G. B. Davis & Olson, 1984).

korištenja poslovnih aplikacija, koja se uvelike preklapaju, možemo najkraće sažeti u tri razdoblja u kojima se pretežito koristila neka od navedenih tehnologija:

- Centralno računalo (1950 - 1980)
- Osobno računalo (1980 - 1990)
- Internet (1990 - do danas)

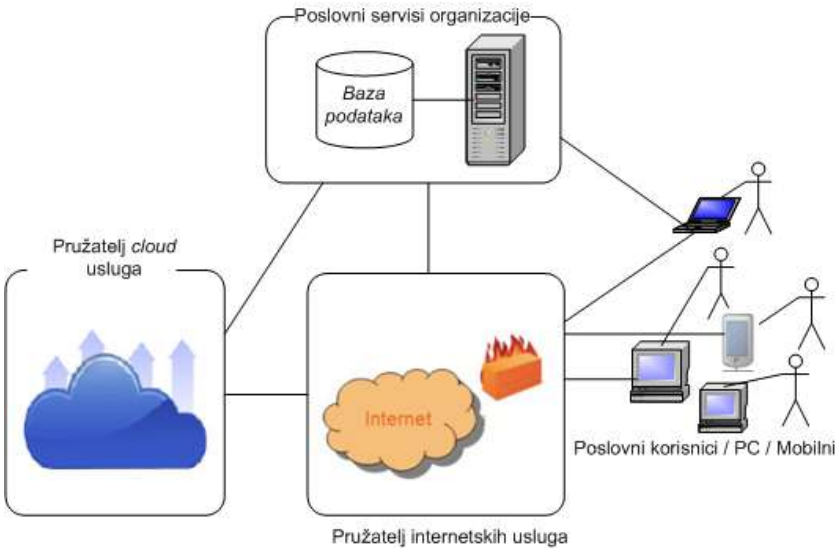
U prvom razdoblju dominiralo je centralno računalo (engl. *mainframe),* programiranje je bilo kompleksno, interakcija sa korisnikom svedena je na minimum (npr. pokretanje programa za obradu plaća); nešto kasnije se počinju koristiti interaktivne aplikacije, transakcijski sustavi koji se odlikuju radom u realnom vremenu.

U drugom razdoblju početkom 1980-tih godina pojavila su se osobna računala (engl. *personal computers*); na početku su se koristila uglavnom kao pomoćna računala za obradu teksta, tabličnih podataka, grafičko uređivanje i slično, početkom 1990-tih godina osobna računala povezivala su se sa ostalim centralnim sustavima, to vrijeme se smatra početkom klijent-poslužitelj (engl. *client-server*) razdoblja koje je karakteristično najviše po tome što su se poslovne aplikacije počele razdvajati na dva dijela; klijentski dio na kojem se nalazi poslovna aplikacija, te na poslužitelj na kojem se nalazi baza podataka.

Razdoblje koje još traje (Internet), karakterizira korištenje internet preglednika za pristup i izvođenje poslovnih aplikacija. Poslovna logika aplikacija i poslovni podaci se uglavnom nalaze na udaljenim lokacijama, broj korisnika je skoro pa neograničen, nije nužna instalacija poslovnih aplikacija na računala korisnika. Za razvoj poslovnih aplikacija najčešće se koriste programski jezici Java, C#, PHP, ASP, te gotove komponente i programski okviri kao što su Struts, Spring, Hibernate, i drugi. Korisnici poslovnih aplikacija više nisu samo poslovni korisnici unutar organizacija, nego su to sada i poslovni korisnici koji sustavu pristupaju iz svoga doma, te klijenti i potencijalni klijenti kojima je omogućen pristup informacijskom sustavu preko interneta. Slika 4 prikazuje pojednostavljeni poslovni sustav kakav se danas najčešće koristi u poslovnim organizacijama.

Veliki broj poslovnih aplikacija danas se potpuno eksternalizira izvan organizacija; pružatelji usluga na udaljenim lokacijama, komercijalnim oblacima (engl. *cloud computing*) obavljaju poslove nadzora, administriranja, kontrole sigurnosti, pa čak i samog razvoja poslovnih aplikacija. Korisnici unutar poslovnih organizacija imaju pristup tim aplikacijama gotovo na istovjetan način kao da su dio njihove lokalne infrastrukture ili sve češće putem hardverske virtualizacije (engl. *virtualization*) na način da se od korisnika „skriva“ pravi hardver i softver.

Organizacije pružateljima usluga korištenje aplikacija i ostalih resursa plaćaju prema obimu korištenja (engl. *pay per use*).



Slika 4 Korištenje interneta za rad s poslovnim aplikacijama

Već danas, možemo tvrditi da su mobilni uređaji na putu da obilježe slijedeće četvrto razdoblje razvoja i korištenja poslovnih aplikacija; u budućnosti možemo očekivati da će povezivanje različitih vrsta uređaja u jedinstveni sustav biti glavno obilježje još jednog potpuno novog razdoblja.

2.3. Arhitektura informacijskih sustava

U ovom pod poglavlju istražujemo arhitekturu sustava poslovnih aplikacija u kontekstu uvođenja i primjene linija za proizvodnju softvera koju nazivamo „referentna arhitektura“; istražujemo proces definiranja arhitekture, način i metode za prikupljanje zahtjeva za razvoj arhitekture; analiziramo postojeće definicije i definiramo vlastitu definiciju referentne arhitekture za kontekst koji uključuje više poslovnih sektora; analiziramo razliku između funkcionalnih zahtjeva poslovnih aplikacija i funkcionalnih zahtjeva referentne arhitekture. Svrha istraživanja arhitekture je povezana sa postavljenim ciljem ovog istraživanja: (C1) definirati i provjeriti funkcionalnosti i strukturne aspekte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera.

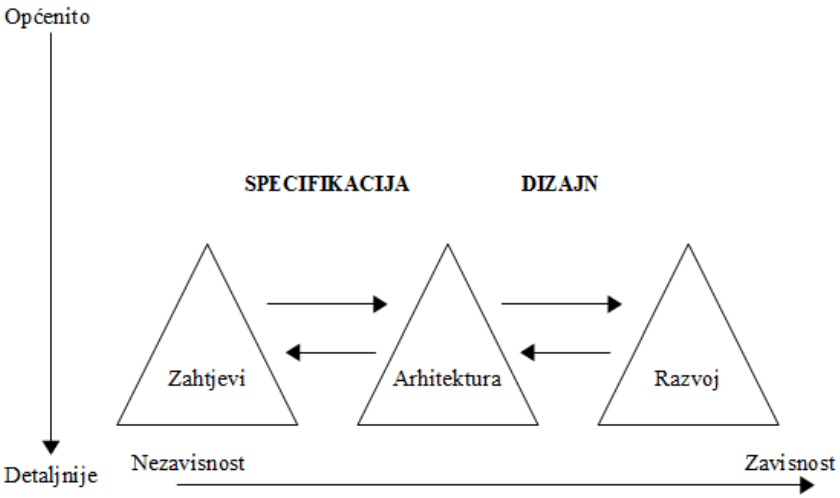
2.3.1. Proces definiranja arhitekture

Arhitektura informacijskog sustava je način na koji se povezuju poslovne aplikacije, tehnološka platforma i komunikacijske mreže u funkcionalan i učinkovit informacijski sustav. Važnost arhitekture poslovnih aplikacija u kontekstu razvoja informacijskih sustava, izrazito se naglašava u literaturi iz područja softverskog inženjerstva (Bass, Clements, & Kazman, 2003; Richard N. Taylor, Medvidovic, & Dashofy, 2009).

Prema standardu ISO/IEC/IEEE 42010:2011 arhitektura sustava je „*temeljna organizacija sustava koji se sastoji od komponenata, njihovog odnosa jednih prema drugima i prema okolini, te načela za njihovo oblikovanje i razvoj*“ (Käköla & Duenas, 2006, str. 248). Sustav se u ovom kontekstu odnosi na skup komponenata koje su organizirane na način koji omogućava izvršavanje specifičnih funkcija. Pojam sustava obuhvaća pojedinačne aplikacije, podsustave, grupu sustava, linije za proizvodnju softvera, kompletnu organizaciju, i druge slične grupacije. Sustav postoji kako bi ostvario jedan ili više zadataka u svome okruženju. Razvoj arhitekture ima za cilj kreiranje jedinstvenog okruženja u organizaciji (standardiziranje hardverskog i softverskog sustava) koji je usko povezan sa poslovnim procesima i poslovnom strategijom organizacije, promoviranje ponovne upotrebe postojećih artefakata i iskustava u vođenju projekata, te razvoj softvera koji će omogućiti poboljšanje i pojeftinjenje poslovanja organizacije.

Definiranje arhitekture započinje vrlo rano, često i prije nego što su zahtjevi i opseg sustava u potpunosti definirani. Zbog te činjenice, definiranje arhitekture je po prirodi dosta fleksibilno i u mnogome ovisi o vještinama i iskustvu arhitekta. Na samom početku definiranja nisu u potpunosti poznata veličina i opseg sustava, područja kompleksnosti, potencijalni rizici, te eventualni konflikti između zahtjeva pojedinih dionika. Slika 5 prikazuje proces definiranja arhitekture koji se odvija između dvije aktivnosti, specifikacije zahtjeva i dizajna, i to sa sve više detalja kako se projekt završava.

Specifikacija zahtjeva i definiranje arhitekture, kao i razvoj i definiranje arhitekture, su međusobno isprepleteni za vrijeme trajanja projekta, sa sve više detalja tijekom prolaska vremena rada na projektu. Iako su specifikacija zahtjeva, definiranje arhitekture i razvoj potpuno različite aktivnosti, jedno na drugo imaju veliku utjecaj te se ne smiju promatrati odvojeno.



Slika 5 Kontekst definiranja arhitekture (Rozanski & Woods, 2011)

Za uspjeh procesa definiranja arhitekture velika važnost se pridaje pridržavanju načela kao što su važnost korisničkih zahtjeva, intenzivna komunikacija sa korisnicima, pragmatičnost u okvirima realnih mogućnosti, fleksibilnost, tehnološka neutralnost itd.

2.3.2. Specifikacija zahtjeva arhitekture sustava

Prikupljanje zahtjeva za razvoj arhitekture sustava, uvijek započinje od glavnih dionika (engl. *stakeholders*) i njihovih potreba. Na temelju tih zahtjeva arhitekt dobije početne naznake opsega sustava za kojeg se razvija arhitektura, te na temelju njihove detaljne analize definira kontekst sustava pomoću kojeg se određuje njegov položaj i odnos s vanjskim elementima s kojima je nužno povezan; to su uglavnom njegovi korisnici ili drugi sustavi. Kontekst sustava predstavlja temelj za definiranje funkcionalnih i nefunkcionalnih zahtjeva i određivanje prioriteta među njima. Odgovornost za kvalitetu arhitekture budućeg sustava nalaže arhitektima da osim funkcionalnih zahtjeva, čije detaljno razumijevanje je ključno za definiranje uspješne arhitekture, detaljno analiziraju i potencijalne rizike te definiraju nefunkcionalne zahtjeve poput performansi, sigurnosti, uporabljivosti i slično. Proces specifikacije zahtjeva završava kada se zahtjevi detaljno dokumentiraju, te na temelju zajedničkog pregleda sa dionicima, potvrde kao potpuni u odnosu na stvarne potrebe dionika.

Specifikacija zahtjeva jednog sustava razlikuje se u odnosu na specifikaciju zahtjeva koja se odnosi na više srodnih sustava, utoliko što u prvom slučaju svih zahtjevi trebaju biti obuhvaćeni u sustavu, dok se u drugom slučaju zahtjevi dijele na one koji su zajednički svima srodnim

sustavima od onih koji su specifični za pojedini sustav u grupi srodnih sustava. U slučaju kada se radi o grupi srodnih sustava, osim zajedničkih zahtjeva nužno je identificirati još dvije grupe zahtjeva, alternativne i opcionalne. Opcionalni zahtjevi odnose se samo na neke od srodnih sustava, dok se alternativni odnose na srodne sustave za koje se oni odabiru između više alternativa.

2.3.3. Referentna arhitektura sustava

Rational Unified Process[3] (RUP) definira „referentnu arhitekturu kao unaprijed definirani predložak ili skup predložaka, djelomično ili potpuno implementiranih, oblikovanih, spremnih za upotrebu u konkretnom poslovnom i tehničkom kontekstu, skupa sa pripadajućim artefaktima koji omogućavaju njenu uporabu“. Referentna arhitektura u kontekstu linija za proizvodnju softvera predstavlja „ skup glavnih dizajnerskih odluka sa eksplicitno definiranim točkama varijacija koje se istovremeno mogu primjenjivati na više sličnih sustava, tipično unutar neke aplikativne domene“ (Richard N. Taylor i ostali, 2009, str. 588). Navedenu definiciju u ovom radu proširujemo i redefiniramo na slijedeći način:

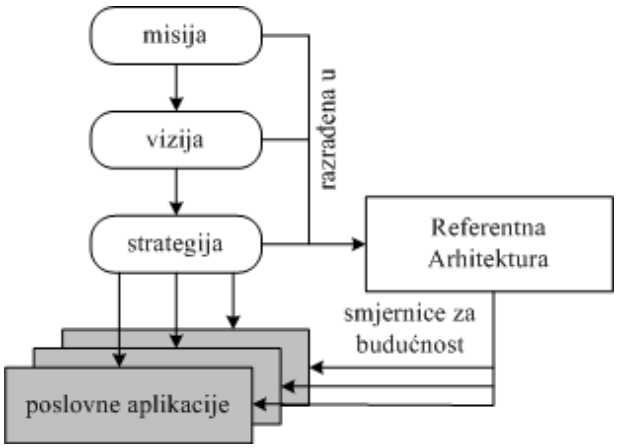
Referentna arhitektura je skup glavnih dizajnerskih odluka sa eksplicitno definiranim točkama varijacija koje se istovremeno mogu primjenjivati na više sličnih sustava, unutar neke aplikativne domene ili unutar više aplikativnih domena ukoliko se razina apstrakcije referentne arhitekture prilagodi njihovim zajedničkim karakteristikama.

Definiranje arhitekture može se odnositi samo na jedan sustav ili na više sličnih sustava koji pripadaju sličnoj poslovnoj i tehnološkoj okolini. Zajedničke karakteristike i sličnosti većeg broja sustava najprihvatljivije je definirati putem referentne arhitekture. Referentna arhitektura je relativno novi pojam, međutim mnogi arhitekti koji razvijaju kompleksne sustave često su koristili taj naziv. Razlike u arhitekturi budućih sustava koji se temelje na referentnoj arhitekturi su specifične u odnosu na standardnu arhitekturu pojedinačnih aplikacija, te ih je potrebno na poseban definirati i dokumentirati.

Slika 6 prikazuje najčešći kontekst u kojem se referentna arhitektura koristi u poslovnim organizacijama. Referentna arhitektura je usko povezana sa poslovnom strategijom organizacija i predstavlja smjernice za budući razvoj aplikacija za cijelu organizaciju.

[3] http://www-306.ibm.com/software/awdtools/rup/

U kontekstu linija za proizvodnju softvera, definicija referentne arhitekture se odnosi na osnovnu arhitekturu u kojoj se definira dizajn na najvišoj razini, a koji se odnosi na sve njezine aplikacije (Pohl, Böckle, & Linden, 2005, str. 124). U referentnoj arhitekturi linija za proizvodnju definiraju se komponente koje su zajedničke za više aplikacija, specificiraju se varijacijske točke za konfiguriranje raspoloživih opcija koje se koriste kod razvoja aplikacija, definiraju se i ograničenja arhitektura aplikacija koje se na njoj temelje.



Slika 6 Kontekst referentne arhitekture (Cloutier i ostali, 2010)

2.3.4. Funkcionalni zahtjevi referentne arhitekture za poslovne aplikacije

U kontekstu referentne arhitekture, definicija funkcionalnih zahtjeva razlikuju se od uobičajene definicije funkcionalnih zahtjeva korisničkih aplikacija. Primjerice, funkcionalni zahtjev neke poslovne aplikacije "*Sustav treba omogućiti korisniku pregled trenutnog stanja na bankovnom računu*", predstavlja funkciju koju sustav obavlja za korisnika poslovne aplikacije. Korisnici referentne arhitekture su uglavnom razvojni programeri, dok se njeni funkcionalni zahtjevi odnose na funkcije koje je referentna arhitektura u stanju obaviti, poput izvršavanja transakcija, čuvanja podataka u privremenoj memoriji i slično. Osim funkcionalnih zahtjeva koji specificiraju "što" referentna arhitektura radi, potrebno je definirati i nefunkcionalne zahtjeve kojima se specificira tražena kvaliteta obavljanja tih funkcija, poput brzine odzivnog vremena, razine sigurnosti, stabilnosti i slično. Zahtjeve referentne arhitekture dijelimo na one koji su značajni (engl. *architecturally significant requirement*) i ostale zahtjeve. "Značajno", u ovom kontekstu se odnosi na moguće troškove eventualne zamjene u budućnosti koje može uzrokovati predmetni zahtjev (Bosch, 2009a).

2.4. Razvoj referentne arhitekture poslovnih aplikacija

U ovom pod poglavlju istražujemo načine oblikovanja, razvoja i opisa referentne arhitekture, te istražujemo postojeće referentne arhitekture. Svrha ovih istraživanja je povezana sa postavljenim ciljevima ovog istraživanja: (C1.1) analizirati postojeća istraživanja iz područja referentnih arhitektura za poslovne aplikacije i identificirati postojeće izazove za referentne arhitekture, (C1.2) analizirati postojeće referentne arhitekture za poslovne aplikacije, (C2.1) opisati razvijene artefakte referentne arhitekture.

2.4.1. Dizajn referentne arhitekture

Razni autori iz područja referentnih arhitektura u svojim istraživanjima potvrdili su poseban značaj oblikovanja referentne arhitekture, posebno u kontekstu linije za proizvodnju softvera. Angelov je u svome radu analizirao kontekst, ciljeve i dizajn referentne arhitekture te utvrdio pet pripadnosti vrstama referentne arhitekture koje imaju veću vjerojatnost za uspjeh, potvrdio je i važnost upotrebe alata za razvoj referentne arhitekture (Angelov, Grefen, & Greefhorst, 2012). Losavio je kroz studiju slučaja oblikovanja referentne arhitekture uz pomoć novog konceptualnog modela (engl. *Domain Quality View*) potvrdio specifičnost referentne arhitekture u kontekstu linija za proizvodnju softvera u odnosu na razvoj standardne referentne arhitekture (Losavio & Matteo, 2013). Nakagawa u svome radu naglašava važnost skupa značajnih karakteristika referentne arhitekture, te predstavlja model (engl. *Reference Architecture Model*) u svrhu poboljšanja razumijevanja važnosti referentne arhitekture i komponenta kod kojih se sastoji, kako bi se potaknulo oblikovanje, razvoj i primjena takvih referentnih arhitektura (Nakagawa, Oquendo, & Becker, 2012).

Definiranje arhitekture odnosi se na proces oblikovanja glavnih dijelova sustava, te na donošenje značajnih odluka vezanih uz njegov dizajn. Isto tako, oblikovanje arhitekture se može promatrati kao strateško oblikovanje, dok se detaljni dizajn koji se definira kasnije, u fazi razvoja, može promatrati kao taktičko oblikovanje. Proces definiranja arhitekture temelji se prikupljenim i prioritetiziranim zahtjevima, odvija se u dva postepena koraka:

- Kreiranje logičke arhitekture: je prvi korak između specifikacije zahtjeva i razvijenog sustava u konačnici, pri čemu se izostavljaju eventualni detalji o tehnologiji koja će se koristiti u razvoju ili kasnije u tijeku korištenja sustava.

- Kreiranje fizičke arhitekture: drugi je korak za vrijeme kojeg se na temelju definirane logičke arhitekture uzimaju u obzir i tehnološki detalji na kojima će se temeljiti razvoj i korištenje sustava u praksi.

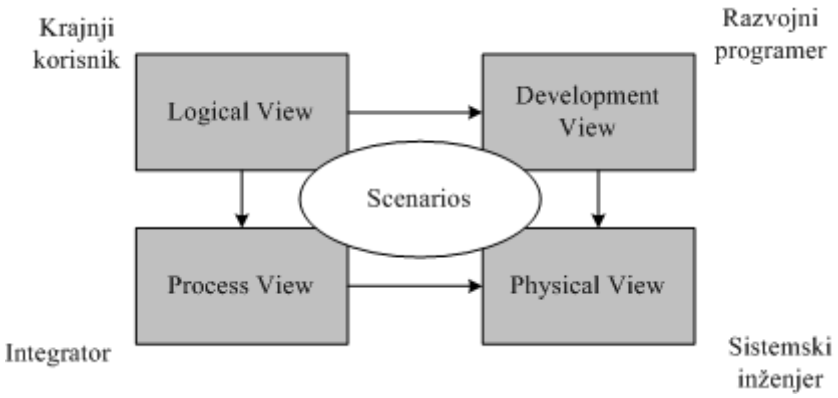
U definiranju arhitekture koriste se predlošci za arhitekturu (engl. *architectural patterns*) s ciljem lakšeg razumijevanja arhitekture od strane svih dionika, iskorištavaju se rješenja koja već postoje za probleme koji se ponavljaju. Osim predložaka, važno je istaknuti da se za definiranje arhitekture također koriste i stilovi arhitekture (engl. *architectural style*), koji se odnose na skup glavnih odluka kod oblikovanja arhitekture; u našem slučaju, jedan od stilova koji primjenjujemo odnosi se na slojeve (engl. *layered*).

Za definiranje arhitekture koriste se i mnoge metode, pristupi i alati, kao što su: Rational Unified Process (Jacobson, Booch, Rumbaugh, Rumbaugh, & Booch, 1999), Kruchten 4+1 perspektiva (P. B. Kruchten, 1995), Simensov Four Views model, Architecture Tradeoff Analysis Method (Kazman i ostali, 1998), Zachman razvojni okvir (Zachman, 1987).

2.4.2. Opis referentne arhitekture

Opis referentne arhitekture obuhvaća nekoliko dokumenta koji se odnose na opseg i kontekst arhitekture, zahtjeve, ograničenja, principe, te različite aspekte; nazivaju se zajedničkim nazivom *architecture description*. Za opis sadržaja u dokumentima arhitekture koristi se jezična forma koju razumiju dionici, dok je glavni cilj tih dokumenata prikazati dionicima da su njihovi zahtjevi uzeti u obzir. Sadržaj dokumenata referentne arhitekture treba biti ažuran u odnosu na stvarno stanje, te mora odisati jasnoćom, ispravnošću i preciznošću.

Standard ISO/IEC/IEEE 42010:2011 preporučuje dokumentiranje arhitektura prikladno interesima dionika, ali ne propisuje nikakav određeni skup aspekata (engl. *views*). Arhitekti biraju aspekte ovisno o specifičnostima pojedinog projekta i interesima dionika referentne arhitekture. Rational Unified Process koristi 4+1 aspekta prema (P. B. Kruchten, 1995), ostali modeli opisa arhitektura koji se najčešće koriste u praksi odnose se na Siemens model koji koristi 4 aspekta (Hofmeister, Nord, & Soni, 2000), te Zachman model od 6 aspekata za 5 vrsta dionika (Zachman, 1987). Slika 7 prikazuje opis referentne arhitekture poslovnih aplikacija koji se najčešće prikazuje pomoću 4+1 aspekta, pri čemu je svaki od njih prikladan posebnoj grupi dionika i njihovim specifičnim interesima.



Slika 7 Aspekti arhitekture prema 4+1 modelu (Kruchten, 1995)

Opis arhitekture se uglavnom realizira grafički uz pomoć nekoliko vrsta *Unified Modeling Language* (UML) dijagrama te upotpunjuje prikladnim tekstualnim opisom. *Logical* aspekt opisuje sustav uzimajući u obzir funkcionalne zahtjeve korisnika i njihovu realizaciju u obliku funkcija koje sustav može obavljati za korisnika. *Development* aspekt opisuje organizaciju softvera u razvojnom okruženju sa aspekta dionika koji sudjeluju u njegovom razvoju. *Process* aspekt prikazuje funkcionalne i neke od nefunkcionalnih zahtjeva u obliku procesa za vrijeme izvođenja aplikacija. *Physical* aspekt prikazuje raspored hardvera i odgovarajućeg softvera u sustavu. *Scenarios* aspekt se koristi za sveobuhvatni opis funkcioniranja sustava pomoću dinamičkih dijagrama i niza opisnih rečenica koje obuhvaćaju najvažnije funkcionalnosti sustava. Osim perspektiva, ovisno o interesima dionika i specifičnim potrebama projekta, arhitekti mogu koristiti i druge aspekte za opis arhitekture, primjerice *data view*, *project tasks view*, i slično.

2.4.3. Postojeće referentne arhitekture

Referentna arhitektura pojedinačnih poslovnih aplikacija najčešće se u praksi organizacija koje razvijaju poslovne aplikacije temelji na nekom od postojećih programskih okvira ili platformi, primjerice na: *Roma Application Framework, OpenXava, JBoss Seam, Vaadin, Spring*, i drugima. Navedeni programski okviri omogućavaju ubrzani razvoj poslovnih aplikacija, razdvajanje poslovne logike i tehnoloških komponenata koje se upotrebljavaju, te jednostavniju eventualnu zamjenu komponenata koje se indirektno koriste u aplikacijama.

Osim navedenih programskih okvira koji obuhvaćaju sve slojeve poslovnih aplikacija (prezentacijski sloj, sloj poslovne logike i sloj pristupa podacima), mnoge organizacije koriste

programske okvire koji obuhvaćaju samo neke od slojeva poslovnih aplikacija, primjerice *Hibernate* okvir za pristup podacima, *Struts* okvir za prezentacijsku logiku, i slično.

Za razliku od pojedinačnih poslovnih aplikacija za koje postoje raspoloživa rješenja kada je u pitanju referentna arhitektura, organizacije koje razvijaju veći broj poslovnih aplikacija prema pristupu linija za proizvodnju softvera, svoju referentnu arhitekturu uglavnom razvijaju samostalno. Vlastita referentna arhitektura ne isključuje korištenje postojećih programskih okvira i komponenata, već ih povezuje u zajedničku cjelinu skupa sa platformom linije za proizvodnju softvera i s arhitekturom pojedinačnih poslovnih aplikacija.

2.5. Provjera kvalitete softvera

U ovom pod poglavlju istražujemo metode, metrike i tehnike analize kvalitete referentne arhitekture i arhitekture općenito. Svrha ovog istraživanja je povezana sa postavljenim ciljem istraživanja: (C4.1) izmjeriti jakost veza među modelima PR i MI za mjerenje održavljivosti aplikacijskih poslovnih komponenata u jednoj financijskoj instituciji.

2.5.1. Definicija kvalitete softvera

Definicija kvalitete softvera odnosi se na razinu do koje sustav, komponenta, ili proces zadovoljavaju postavljene zahtjeve; razinu do koje sustav, komponenta, ili proces zadovoljavaju potrebe ili očekivanja korisnika (IEEE 610.12-1990, 2005). Prvi dio ove definicije definira kvalitetu softvera za ne-interaktivne sustave, kakva je i referentna arhitektura, dok se drugi dio odnosi na interaktivne sustave kao što su interaktivne poslovne aplikacije. Jedan od ključnih problema u razvoju aplikacija danas, odnosi se na njihovu kvalitetu. Ključni čimbenik u kvaliteti aplikacija koje zajednički koriste istu referentnu arhitekturu je kvaliteta same referentne arhitekture (Dobrica & Niemelä, 2000). Provjera kvalitete referentne arhitekture, pored ostalog, ima za cilj predviđanje kvalitete aplikacija koje se na njoj temelje, i to često i prije nego se aplikacije počnu razvijati (Kazman, Bass, Abowd, & Webb, 1993). Analiza kvalitete arhitekture može se provoditi odmah nakon što su donesene glavne odluke o konceptu arhitekture, i kasnije, sve dok se arhitektura primjenjuje u praksi. Primjerice, prema mišljenju mnogih iz ovog područja, valjanost referentne arhitekture koja se koristi u više aplikacija prema pristupu linija za proizvodnju softvera, potrebno je detaljno provjeriti negdje u razdoblju između njene primjene u trećoj i četvrtoj aplikaciji. Ovisno o periodu kada se validacija provodi, koriste se različite tehnike i metode. Rezultati provjere

kvalitete koji u pravo vrijeme ukazuju na područja koja je potrebno poboljšati, mogu se koristiti za poboljšanje procesa razvoja softvera, te poslužiti za komunikaciju sa dionicima ili kao sredstvo za ublažavanje potencijalnih rizika. U konačnici, relevantnu ocjenu kvalitete referentne arhitekture mogu dati samo njezini korisnici, bilo da su to razvojni programeri koji koriste artefakte referentne arhitekture u razvoju poslovnih komponenata ili aplikacija, krajnji korisnici aplikacija, administratori i drugi dionici.

2.5.2. Metode i tehnike analize kvalitete arhitekture

Analiza arhitekture se provodi s ciljem pronalaženja odgovora na pitanje, da li arhitektura zadovoljava zadane kriterije kvalitete na temelju specificiranih zahtjeva. Tehnike koje se pri tome koriste mogu se svrstati u dvije grupe: *ispitivanje* i *mjerenje* (Abowd, Bass, Clements, Kazman, & Northrop, 1997; Kazman i ostali, 1998). Tehnika ispitivanja može se koristiti za analizu bilo kojeg od atributa kvalitete arhitekture, i to korištenjem *upitnika, scenarija ili kontrolne liste*. Tehnika mjerenja odnosi se na kvantitativno mjerenje neke od karakteristika arhitekture, pri čemu se koriste *metrike, simulacije, prototipovi i iskustva*; provodi se s ciljem pronalaženja specifičnih vrijednosti atributa kvalitete arhitekture, te još uvijek nema široku primjenu kao što je ima tehnika ispitivanja. Ovisno o kompleksnosti arhitekture i troškovima analize, mogu se koristiti razne tehnike, najvažnije među njima posebno ćemo opisati.

Validacija korištenjem scenarija, scenarijo u ovom kontekstu predstavlja jednostavnu, jasnu i sažetu rečenicu kojom se definira situacija u kojoj se sustav može naći za vrijeme korištenja ili izmjene, zajedno sa definiranom reakcijom sustava u toj situaciji (Rozanski & Woods, 2011). Slično kao i kod specifikacije zahtjeva, scenariji se dijele u dvije grupe, *funkcionalne* i *nefunkcionalne*. Primjerice, funkcionalni scenarijo definira interakcije korisnika sa sustavom na koje sustav treba moći pravilno reagirati; proces koji se događa završetkom poslovne godine, ili kada preko vanjskog sučelja pristignu podaci koje sustav treba obraditi. Nefunkcionalni scenarijo odnosi se na reakciju sustava kada primjerice, broj aktivnih korisnika postane veći od predviđenog broja, način zaštite sigurnosti podataka u slučaju sigurnosne ugroženosti sustava i druge situacije za koje sustav treba biti dizajniran.

Prezentacija je najjednostavnija forma validacije, odnosi se na prezentaciju predložene arhitekture pred dionicima koji su sudjelovali u specifikaciji zahtjeva. Sama po sebi, prezentacija ne može poslužiti u svrhu validacije arhitekture ukoliko se ne pripremi na način koji od dionika zahtjeva sudjelovanje, razmišljanje i kritički pristup. Sama prezentacija i nije toliko efikasna ukoliko se u isto vrijeme ne koristi i neka od drugih tehnika koje ju nadopunjuju.

Recenzija je organiziranje grupe dionika kojima je prethodno dostavljena dokumentirana arhitektura i drugi relevantni artefakti kako bi ju stranicu po stranicu analizirali, te dali svoje mišljenje i preporuke.

Prototip je razvijeni sustav koji uključuje samo neke od funkcionalnosti, koristi se kako bi se ublažili eventualni tehnološki rizici, najčešće za provjeru novih tehnologija ili pristupa.

Skeleton sustava za razliku od prototipova koji se najčešće uništavaju nakon validacije, predstavlja prvu inačicu budućeg sustava sa minimalnim brojem funkcija. Međutim, taj minimalni broj treba omogućiti provjeru kompletne strukture sustava, od početka do kraja.

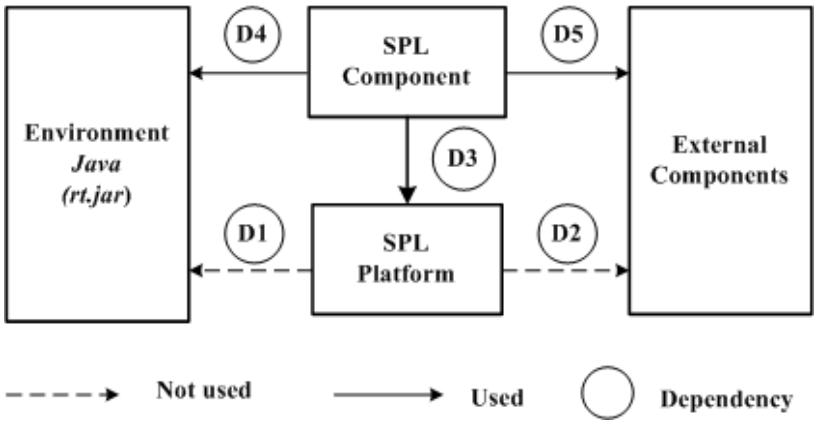
Neke od najčešće korištenih metoda za validaciju arhitekture, kao što su *Architecture Tradeoff Analysis Method* (ATAM) (Kazman i ostali, 1998), *Software Architecture Assesment Method* (SAAM) (Kazman, Bass, Webb, & Abowd, 1994), i *Holistic Product Line Architecture Assessment* (HoPLAA) (Olumofin & Mišić, 2007) temelje se na scenarijima kao tehnici za validaciju arhitekture.

2.5.3. Metrike kvalitete referente arhitekture

Kvalitetu softverskih proizvoda možemo mjeriti pomoću dvije vrste mjera, *unutarnjih* i *vanjskih*. Unutarnje mjere opisuju niz statičnih unutarnjih osobina koje se mogu izmjeriti, dok se vanjske mjere više fokusiraju na softver kao „crnu kutiju“ i opisuju vanjske atribute koji se mogu izmjeriti. Tablica 2 prikazuje karakteristike prema ISO/IEC 25010:2011 standardu pomoću kojih se može mjeriti kvaliteta softverskih artefakata. Za mjerenje kvalitete artefakata referentne arhitekture može se koristiti i metoda *Evolution Identification Using Historical Information* (Mattson & Bosch, 1999) za pronalaženje i karakterizaciju svojstava razvojnih okvira, njegovih podsustava i komponenata koja su podložna promjenama, promatrajući povijesne podatke o promjenama nastalim tijekom razvoja i održavanja.

U ovom radu koristimo metrike za provjeru kvalitete referentne arhitekture, a posebno *Maintainability/Changeability* karakteristike za mjerenje kvalitete komponenata poslovnih aplikacija. Nekoliko empirijskih istraživanja se bavi istraživanjem područja održavanja (engl. *maintainability*) i promjenjivosti (engl. *changeability*) artefakata linija za proizvodnju softvera (Bagheri & Gasevic, 2011), (Briand, Bunse, & Daly, 2001), (Aldekoa, Trujillo, Mendieta, & Díaz, 2006), (Mari & Eila, 2003), (Tizzei, Dias, Rubira, Garcia, & Lee, 2011). Međutim, samo jedno istraživanje (Rosko, 2013a) koliko nam je poznato, bavi se promjenjivošću komponenata u kontekstu primjene referentne arhitekture i njihove ovisnosti o vanjskim komponentama.

Slika 8 prikazuje metrike za vanjske i unutarnje ovisnosti komponenata koje mogu poslužiti za procjenu utjecaja promjena u okruženju na poslovne komponente, budući da promjene unutar jedne komponente direktno utječu na sve druge ovisne komponente.



Slika 8 Metrike ovisnosti u kontekstu linije za proizvodnju softvera (Rosko, 2013a)

Ovisnost može postojati između komponenata koje se koriste u aplikacijama i referentne arhitekture (platforme) ili o vanjskim komponentama koje nisu pod kontrolom organizacije. Pretpostavljamo da je promjenjivost neke komponente bolja i pod većom kontrolom internog razvoja, kada je manji broj direktnih ovisnosti o vanjskim komponentama. Ova pretpostavka je na tragu preporuka dobrog modularnog dizajna, kojima je cilj, postići visoki stupanj unutarnje kohezije, i manji stupanj ovisnosti o vanjskim komponentama.

Za validaciju referentne arhitekture linija za proizvodnju softvera, osim atributa kvalitete prikazanih u tablici 2 dodatno se može koristiti i *availability* kao atribut kvalitete kojim se mjere mogućnosti promjene ili proširenja referentne arhitekture.

Maintainability Indeks

Maintainability Indeks (MI) je metrika koja je nastala kao rezultat traženja modela za brzi i jednostavan način procjene održavljivosti softvera upotrebom nekoliko najčešće korištenih i lako mjerljivih i raspoloživih metrika (Oman & Hagemeister, 1992). Izračun MI je složen, uključuje niz različitih metrika: HEFF (kumulativni Halstead Effort svih dijelova klase), NOTM (broj metoda u klasi), TCC (ukupnu vrijednost Cyclomatic Complexity metrike za sve metode u klasi), NOS (ukupni broj Java kompletiranih naredbi). Osim vrijednosti metrika, u formuli za izračun vrijednosti MI se koriste i znanstveno potvrđeni težinski faktori koji su izračunati na temelju korelacije sa subjektivnim procjenama održavljivosti koje su napravili

eksperti koji se bave održavanjem softvera. Vrijednosti metrika se grupiraju u tri dijela koji se zatim koriste za izračun krajnje vrijednosti MI, formula (1).

$$\begin{aligned} \textbf{\textit{EffortPart}} &= 3.42 * log(HEFF/NOMT) \\ \textbf{\textit{CyclomaticPart}} &= 0.23 * log(TCC/NOMT) \\ \textbf{\textit{LinesPart}} &= 16.2 * log(NOS/NOMT) \\ \boldsymbol{MI} &= 171 - effortPart - cyclomaticPart - linesPart \end{aligned} \quad (1)$$

Rezultat mjerenja MI je broj koji ukazuje na razinu održavljivosti. Što je veći MI neke softverske komponente, veća je njena održavljivost. Komponente s vrijednošću MI manjom od 65 je teško održavati, komponente sa vrijednošću MI između 65 i 85 imaju prosječnu održavljivost, komponente sa vrijednošću MI iznad 85 imaju izvrsnu održavljivost. MI metrika se često koristi u praksi za procjenu održavljivosti softvera, znanstveno je potvrđena za proceduralne i za objektno orijentirane programske jezike (Coleman, Ash, Lowther, & Oman, 1994; Misra, 2005; Oppedijk, 2008; Schach, Jin, Wright, Heller, & Offutt, 2002; Welker & Oman, 1995; Zhou & Xu, 2008).

U softverskom inženjerstvu općenito je prihvaćen stav da arhitektura utječe na atribute kvalitete sustava kao što su promjenjivost (engl. *changeability*) ili performanse. Međutim, na temelju našeg iskustva vjerujemo da se to odnosi i na upotrebljivost sustava koji koriste predmetnu arhitekturu ili na upotrebljivost samih artefakata referentne arhitekture čiji korisnici su razvoji programeri poslovnih aplikacija.

Tablica 2 Atributi kvalitete prema ISO/IEC 25010:2011 standardu

Atributi kvalitete	**Podgrupe**
Functional suitability Stupanj do kojeg softver obavlja funkcije koje zadovoljavaju zadane i implicirane potrebe, kada se softver koristi pod određenim uvjetima.	*Suitability, Accuracy, Interoperability, Security, Compliance*
Reliability Stupanj do kojeg sustav ili komponente obavljaju određene funkcije u skladu s propisanim uvjetima za određeno vremensko razdoblje.	*Maturity, Fault Tolerance, Recoverability, Compliance*
Operability Stupanj do kojeg proizvod ima karakteristike koje omogućavaju njegovo lakše razumijevanje, savladavanje i korištenje, ako se koristi pod određenim uvjetima.	*Appropriatenes, Recognisability, Ease of use, Learnability, Attractiveness, Technical accessibility, Compliance*
Performance efficiency Performanse u odnosu na korištenje resursa u zadanim uvjetima.	*Time Behaviour, Resource Utilisation, Compliance*

Security Stupanj zaštite informacija i podataka, tako da ih neovlaštene osobe ili sustavi ne mogu čitati ili mijenjati, na način da ovlaštenim osobama ili sustavima nije zabranjen pristup.	*Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity, Compliance*
Compatibility Stupanj u kojem dva ili više sustava ili komponenata mogu razmjenjivati informacije i / ili obavljati svoje funkcije, dok koriste isto hardversko i softversko okruženje.	*Replaceability, Co-existence, Interoperability, Compliance*
Maintainability Stupanj učinkovitosti i efikasnosti s kojim se produkt može mijenjati.	*Modularity, Reusability, Analyzability, Changeability, Modification stability, Testability, Compliance*
Transferability Stupanj do kojeg sustav ili komponenta može biti djelotvorno i učinkovito prenesen s jednog hardvera, softvera ili nekog drugog okruženja na neki novi hardver, softver ili okruženje.	*Portability, Adaptability, Installability, Compliance*

2.6. Životni ciklus poslovnih aplikacija

Poslovne organizacije, da bi bile konkurentne, nužno usklađuju svoje poslovne strategije sa suvremenim tehnologijama i alatima koji po svojoj prirodi vrlo brzo zastarijevaju. Nekada suvremene tehnologije i alati te informacijski sustavi koji su razvijeni i održavaju se pomoću njih, u mnogim poslovnim organizacijama još uvijek predstavljaju najvažnije dijelove informacijskog sustava. Iako takvi sustavi imaju veliku važnost, tijekom vremena njihova kompleksnost i osjetljivost se uvećava i poskupljuje njihovo održavanje, što se najčešće događa u poslovnim sektorima kao što su: bankarstvo, financije, javna uprava, obrana, industrijska proizvodnja, telekomunikacije, transport, trgovina i u nekim drugim poslovnim sektorima.

Uprave i ostali visoki menedžment u poslovnim organizacijama uviđaju da zastarjeli sustavi postaju preveliki poslovni, strateški, tehnološki, financijski i operativni rizik kojeg treba pro aktivno ublažavati. Ova disertacija predstavlja jedno od mogućih rješenja za ublažavanje navedenih rizika, pro aktivno, i to na dva moguća načina, modernizacijom arhitekture postojećih poslovnih aplikacija ili razvojem novih poslovnih aplikacija uz korištenje predloženog pristupa. Modernizacija arhitekture postojećih poslovnih aplikacija pruža poslovnim organizacijama priliku za potpunije razumijevanje same arhitekture, definiranje budućeg načina razvoja i transformaciju ključnih poslovnih sustava. Modernizacija postojećih poslovnih sustava u svojoj biti nije više taktička i neobavezna, već ima stratešku, dugoročnu,

važnu i sveobuhvatnu ulogu, te služi kao sredstvo usklađivanja poslovne strategije i informacijske tehnologije u organizacijama.

2.6.1. Razvoj poslovnih aplikacija

Poslovne aplikacije kao i ostali softver razvijaju se korištenjem raznih modela, metoda, tehnika i pristupa. Proces razvoja poslovnih aplikacija se uglavnom odvija korištenjem takozvanog vodopad (engl. *waterfall*) modela koji se u praksi pokazao dosta ograničenim, međutim u zadnje vrijeme on se sve više zamjenjuje sa novim pristupima koji se temelje na kombinaciji iterativnog (engl. *iterative*), spiralnog (engl. *spiral*) i agilnih (engl. *agile*) modela. Trenutno se najviše koriste agilne metode; broj istraživačkih radova iz područja agilnih metoda potvrđuje veliki interes znanstvene zajednice i organizacija za njihovom primjenom (Dingsøyr, Nerur, Balijepally, & Moe, 2012).

Od velikog broja metoda, navodimo samo neke koji se najčešće koriste u praksi.

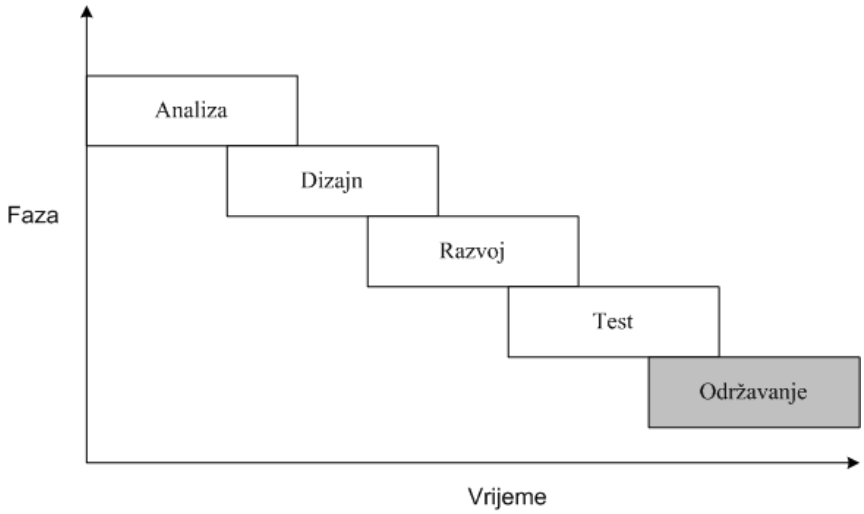
Rational Unified Process (RUP): iterativni okvir procesa razvoja softvera kojeg je definirala *Rational Software Corporation*, dio IBM-a od 2003. godine. RUP se može prilagoditi specifičnostima pojedinih organizacija i projektnih timova koji mogu odabrati elemente procesa koje su prikladni za njihove potrebe. Proces se sastoji od 4 glavne faze: **osnutka** (engl. *inception*), **razrade** (engl. *elaboration*), **izgradnje** (engl. construction), **tranzicije** (engl. *transition*) (P. Kruchten, 2004).

Scrum je iterativni, inkrementalni i agilni proces za razvoj softvera. Glavni princip ovog procesa odnosi se na ulogu korisnika i predviđanje mogućnosti da korisnik bilo u kojoj fazi projekta može promijeniti svoje zahtjeve koji se prema tradicionalnim procesima razvoja ne mogu na efikasan način uzeti u obzir. Scrum podrazumijeva podjelu uloga na projektu na 3 glavne uloge: **vlasnik proizvoda** (engl. *product owner*), **razvojni tim** (engl. *development team*), **voditelj tima** (engl. *scrum master*) (Beck i ostali, 2001).

U životnom ciklusu poslovnim aplikacija, ključnu ulogu ima proces njihovog razvoja kojeg možemo sažeti u nekoliko glavnih koraka koji su zajednički za sve navedene metode. Prvo se analizira poslovni problem i radi specifikacija zahtjeva korisnika, zatim oblikovanje na temelju tih zahtjeva, nakon toga razvoj i testiranje.

Slika 9 prikazuje faze procesa razvoja poslovnih aplikacija. Proces je obično mnogo kompleksniji, primjerice, dizajn se najčešće dijeli na fazu oblikovanja arhitekture i fazu detaljnog dizajna, testiranje se također dijeli na nekoliko faza. Slika 9 prikazuje faze koje se

odvijaju sekvencionalno, međutim, ovisno o projektu, aktivnosti se najčešće preklapaju; moguće je započeti implementaciju jednog dijela aplikacije dok su ostali djelovi još u fazi oblikovanja.



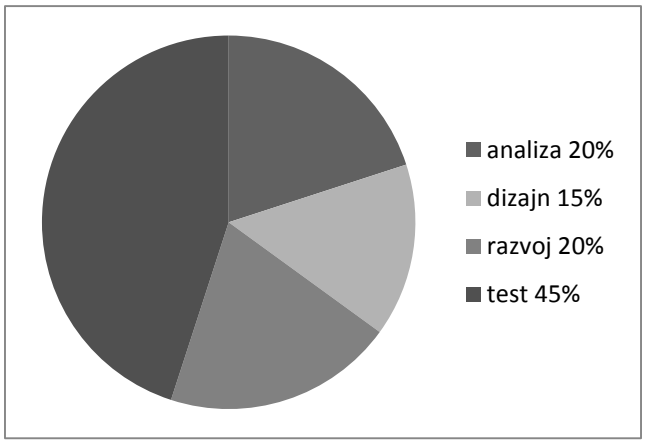
Slika 9 Procesni koraci u razvoju aplikacija

Analiza: Glavni zadatak ove faze je detaljno razumijevanje poslovnog problema, korisničkih zahtjeva i okruženja u kojem će buduća aplikacija funkcionirati. Sastavni dio analize za veće projekte razvoja je obično i studija izvodljivosti, pomoću koje se analiziraju ekonomski i tehnološki aspekti predloženog rješenja za poslovni problem. Rezultati faze analize se dokumentiraju u *specifikaciji zahtjeva* koja se kasnije koristi u mnogim drugim aktivnostima kao što je dizajn, korisničko testiranje, pisanje korisničke dokumentacije.

Dizajn: U fazi dizajna definira se model aplikacije koji ako se kasnije implementira u nekom od programskih jezika, predstavlja rješenje postavljenog poslovnog problema. Poslovni problem se razdjeljuje na manje dijelove, na komponente, specificiraju se glavne funkcije komponenata i njihova sučelja. Odluke koje se donose u ranim fazama dizajna imaju značajan utjecaj na kvalitetu sustava na koji se primjenjuju. Kada se dizajn odnosi na više srodnih aplikacija ili ako služi kao skeleton za razvoj ponovno upotrebljivih komponenta, nazivamo ga *referentna arhitektura,* te ga je potrebno detaljno dokumentirati u obliku *opisa arhitekture* (engl. *architecture description*). Oblikovanje aplikacije dokumentira se u *tehničkoj specifikaciji* koja se koristi kao početna točka za razvoj, te za neke druge aktivnosti kao što je primjerice, definiranje integracijskog testiranja.

Razvoj: Odnosi se na implementaciju pojedinačnih komponenata. Početna točka razvoja je upoznavanje sa tehničkom specifikacijom komponente koja se razvija. U razvoju se koriste različiti pristupi, paradigme, alati i tehnike. Konačni rezultat ove faze je izvodivi program.

Test: Testiranje kao aktivnost ne započinje, kako se obično smatra, nakon razvoja. Prve aktivnosti vezane uz testiranje započinju već u fazi analize zahtjeva korisnika. Predmet testiranja određuje i vrstu testiranja; zahtjevi korisnika povezani su sa korisničkim testiranjem, tehničke specifikacije sa integracijskim testom, izvodivi programi sa *unit* testom, itd. Slika 10 ilustrira grubu procjena trajanja svake od aktivnosti u standardnom procesu razvoja nove aplikacije.

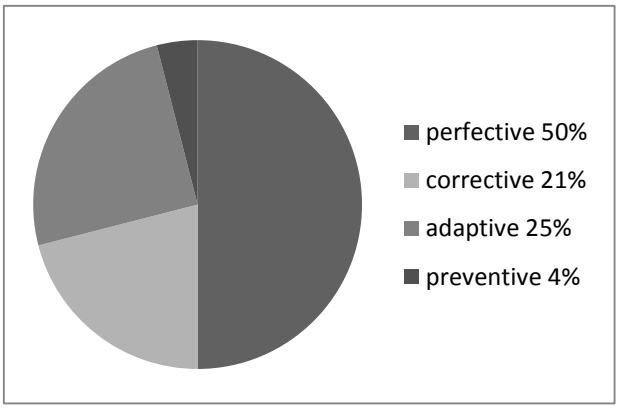


Slika 10 Relativni napor u procesu razvoja aplikacija (Van Vliet, 2008)

Cilj svakog projekta razvoja poslovnih aplikacija je izgraditi sustav koji radi i koji je pouzdan, unutar zadanih granica, koji zadovoljava poslovne ciljeve prema zahtjevima korisnika u prihvatljivom vremenu i po opravdanoj cijeni. Međutim, mnogi projekti nisu bili uspješni zbog nerazumijevanja suštine organizacije i poslovanja, zbog loše izvedbe uslijed pogrešaka u procesu razvoja, ili zbog lošeg vođenja i koordinacije projekta. Glavni procesni koraci zajednički su svim modelima razvoja. Razvoj aplikacija je kompleksan i subjektivan proces koji se može značajno razlikovati čak i u situacijama kada se razvija u potpunosti ista aplikacija. Njegovu kompleksnost čini veliki broj mogućih opcija u izboru timova, njihovih vještina, metoda, tehnologije, razvojnih alata, i drugih resursa, a ponajviše eventualni nedostatak standarda. Međutim, ukoliko se u procesu razvoja uvažavaju postojeća iskustva, planiranje, upravljanje rizicima, komunikacija sa klijentima i znanstvene činjenice, neke od kompleksnosti mogu se lakše savladati.

2.6.2. Održavanje poslovnih aplikacija

Nakon završetka razvoja i isporuke aplikacije korisnicima potrebno je ispravljati eventualne greške, raditi prilagodbe novim okolnostima, proširivati njene funkcionalnosti i poboljšavati neke od karakteristika. Prema IEEE610/1990 standardu, *održavanje* aplikacija je proces mijenjanja sustava ili komponente nakon kompletiranog razvoja u svrhu popravljanja grešaka, poboljšanja performansi ili drugih karakteristika, te prilagodbe promjenama u okruženju. Navedene aktivnosti se svode na *održavanje* aplikacija koje nosi velike troškove u odnosu na druge aktivnosti u procesu razvoja. Smatra se da održavanje predstavlja negdje od 50 do 70 posto ukupnih troškova ako se promatra cijeli životni ciklus neke aplikacije (Jalote, 2008). Održavanje aplikacija može se podijeliti na četiri podgrupe: ispravljanje grešaka (engl. *corective*), prilagodba okruženju (engl. *adaptive*), poboljšanje karakteristika i proširivanje funkcionalnosti (engl. *perfective*), te na preventivno održavanje (engl. *preventive*). Slika 11 prikazuje odnos potrebnog napora za svaku od navedenih aktivnosti održavanja.



Slika 11 Distribucija aktivnosti održavanja (Van Vliet, 2008)

Slika 11 prikazuje podatke iz 1970-tih godina, međutim, kasnija istraživanja (V. Basili i ostali, 1996; Nosek & Palvia, 1990; Schach, Jin, Yu, Heller, & Offutt, 2003) pokazuju da se situacija nije značajnije mijenjala. Promjene u poslovanju organizacija i tehnološke promjene u okruženju su neizbježne, stoga održavanje poslovnih aplikacija zahtjeva ažurna znanja iz oba područja, poslovanja i tehnologije.

2.7. Trendovi razvoja poslovnih aplikacija

Povijesno gledano, poslovne aplikacije razvijale su se na početku u organizacijama koje su ih koristile, predstavljale su poslovnu prednost za te organizacije; kasniji trendovi svode se na izdvajanje razvoja iz organizacija koje koriste poslovne aplikacije. Danas možemo zaključiti da poslovne aplikacije same po sebi ne predstavljaju poslovnu prednost za organizacije koje ih koriste, već se ta prednost odnosi na poslovne modele pomoću kojih organizacije obavljaju svoje poslovanje.

Softversko inženjerstvo kao disciplina koja se bavi razvojem poslovnih aplikacija nalazi se na značajnoj prekretnici, prelasku sa integracijski orijentiranog pristupa prema kompozicijski orijentiranom pristupu. Tradicionalno, glavni fokus softverskog inženjerstva odnosio se na centraliziranu i manualno upravljanu integraciju raznih dijelova sustava. Danas se sve više koristi decentralizirani pristup sa višim stupanj automatske integracije, primjerice u procesima testiranja, instalacije, itd. Decentralizacija je posljedica korištenja postojećih komponenata i servisa koji nisu razvijeni u organizaciji koja razvija poslovne aplikacije; otvorenih tehnologija i rješenja (engl. *open source*), komponenata koje su razvili vanjski razvojni programeri (engl. *eco software developers*), vanjskih servisa (npr. tečajne liste, cijene dionica).

Sam razvoj poslovnih aplikacija danas se odvija uglavnom izvan poslovnih organizacija; sve više se izvode na udaljenim lokacijama (engl. *cloud*); aplikacije imaju ugrađenu podršku za interoperabilnost sa ostalim sustavima (npr. javna uprava, regulatorske organizacije); korisnici najčešće nisu prošli obuku za korištenje aplikacija; aplikacije podržavaju mobilne i internacionalne korisnike.

3. LINIJE ZA PROIZVODNJU SOFTVERA

U ovom poglavlju određujemo osnovne pojmove područja linija za proizvodnju softvera, istražujemo povijesni razvoj ponovne upotrebe, opisujemo linije za proizvodnju softvera iz četiri uobičajene perspektive (organizacija, proces, arhitektura, poslovanje), opisujemo metode razvoja i modele njihove zrelosti, te navodimo neke od studija slučaja uvođenja ovog pristupa u praksu.

Svrha ovoga poglavlja je istražiti kontekst definiranja nove referentne arhitekture kod uvođenja linija za proizvodnju softvera u organizacije, što je povezano sa postavljenim ciljem ovog istraživanja: (C1.3) definirati funkcionalnosti referentne arhitekture, (C3.3) postići zrelost referentne arhitekture razine „4“ prema modelu za zrelost linija za proizvodnju softvera FEF (engl. *Family Evaluation Framework*).

3.1. Opis i definicija linija za proizvodnju softvera

Iako nije izumitelj linija za proizvodnju, Henry Ford je 1913. godine postojeću praksu korištenja pokretnih traka (engl. *assembly line*) unaprijedio i prvi uveo u proizvodnju automobila s ciljem da poveća produktivnost i smanji cijenu automobila. Razvoj sustava za proizvodnju opisuje (Vladimír, 2014); način proizvodnje tijekom vremena se značajno mijenjao; u počecima, kada je proizvodnja bila prilagođena jednom kupcu, proces proizvodnje bio je potpuno drugačiji od procesa kakav se koristio kasnije, kada se značajno povećao broj kupaca koji su mogli kupiti isti proizvod. Ovakav razvoj situacije na tržištu doveo je do uvođenja linija za proizvodnju. Međutim, uvođenjem linija za proizvodnju smanjena je mogućnost izbora raznolikih opcija koje neki proizvod može imati.

U proizvodnji softvera situacija je slična kao i u proizvodnji ostalih proizvoda. Početkom 1990-tih u softverskoj industriji počeo se koristiti pristup linija za proizvodnju softvera (engl. *software product lines)* SPL. Tako se umjesto razvoja softvera svaki put iz početka, počela uvoditi praksa iskorištavanja prethodno razvijenih artefakata. Umjesto razvoja softverskih sustava uvijek na isti način, omogućeno je da se u obzir uzimaju i pojedinačni korisnički zahtjevi pri izboru velikog broja raspoloživih opcija. SPL pristup omogućava masovnu proizvodnju, tako da se pojedinačni proizvodi sastavljanju korištenjem ponovno upotrebljivih komponenata, dok se s druge strane zadržava mogućnost prilagodbe pojedinačnim zahtjevima korisnika. Primjerice, alat za razvoj softvera Eclipse i operativni sustav Linux te njihove inačice

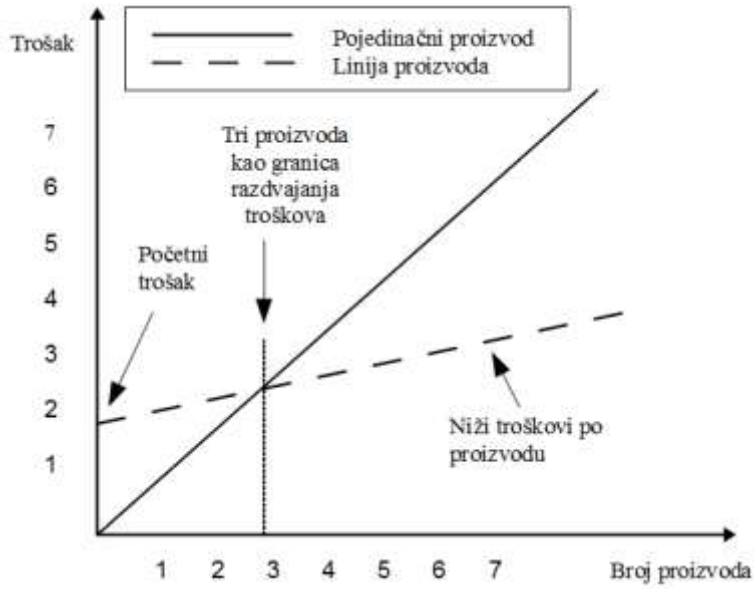
koje su prilagođene grupama različitih korisnika, proizvodi kao što su iPod, iPhone, iPad, (Apple Inc.) predstavljaju primjere vrlo uspješne primjene linija za proizvodnju softvera.

Prilagođavanje korisnicima

SPL pristup omogućava prilagodbu softverskih proizvoda specifičnim korisničkim zahtjevima na način da se u okviru mogućnosti koje su predviđene kao varijabilne točke (engl. *variability points*), svaki proizvod maksimalno prilagodi specifičnim poslovnim potrebama korisnika.

Smanjeni troškovi

Vjerojatno najznačajniji razlog uvođena SPL pristupa u organizacije je evidentno smanjivanje troškova realizacije proizvoda. Najznačajnije smanjenje troškova odnosi se na uštede koje se postižu korištenjem unaprijed definiranih komponenata, umjesto trošenja na dizajn i razvoj svakog proizvoda iz početka. Prije mogućnosti korištenja unaprijed definiranih komponenata, potrebno je utrošiti značajna sredstva za njihov razvoj. Osim toga, za učinkovitu ponovnu upotrebu postojećih artefakta, potrebno je na sustavan način planirati njihov razvoj i korištenje. Slika 12 ilustrira situaciju kada linija za proizvodnju ima manje od četiri proizvoda, pri čemu su troškovi linije relativno visoki, i situaciju gdje se ti troškovi značajno smanjuju kada linija ima veći broj proizvoda (Pohl, Böckle, & van der Linden, 2005).



Slika 12 Troškovi razvoja korištenjem linija za proizvodnju softvera

Negdje u vrijeme razvoja trećeg proizvoda troškovi se izjednačuju za te dvije situacije; tu fazu možemo nazvati „kraj rane faze“. Prema novijim empirijskim istraživanjima točka presijecanja se nalazi negdje između trećeg i četvrtog proizvoda (Parra, 2011). Na poziciju točke presijecanja utječu i druge okolnosti i faktori kao što su: poslovna domena, iskustvo osoblja, raspoloživa financijska sredstva, struktura organizacije, vrsta korisnika, strategija implementacije, te opseg SPL-a.

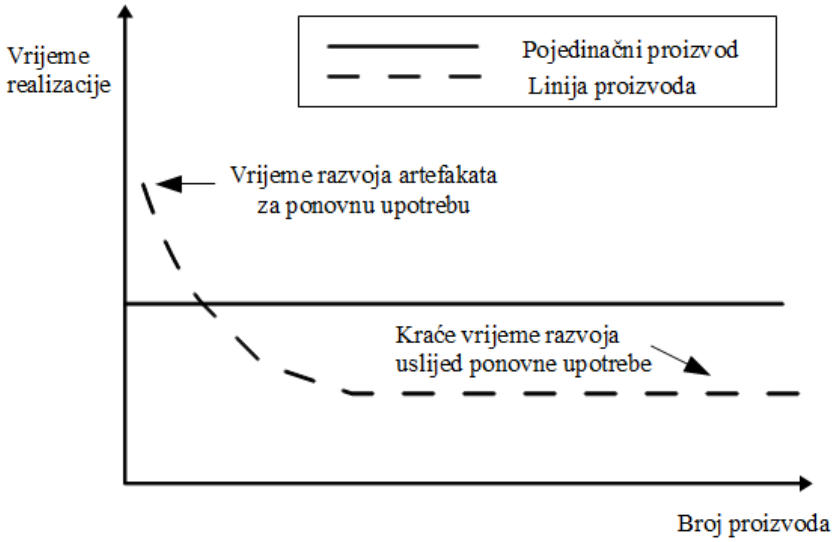
Poboljšanje kvalitete

Budući da se artefakti za ponovnu upotrebu koriste u skoro svim proizvodima, to implicira intenzivno testiranje tih artefakata, povećanje mogućnosti otkrivanja i korekcije grešaka, te njihovo korištenje u različitim situacijama, što značajno doprinosi kvaliteti proizvoda koji ih koriste.

Vrijeme realizacije

Realizacija pojedinačnih proizvoda predstavlja veći trošak i zahtjeva duže vrijeme od realizacije masovnih softverskih proizvoda. Međutim, ukoliko se korisniku omogući izbor između unaprijed definiranih komponenata, proizvođač može vrlo brzo, korištenjem gotovih komponenata, proizvesti i traženi pojedinačni proizvod. U situacijama kada ne postoje unaprijed definirane komponente koje odgovaraju korisničkim zahtjevima, razvoj proizvoda na temelju postojećih komponenata ipak značajno ubrzava razvoj u odnosu na razvoj iz početka. Vrijeme realizacije proizvoda najčešće je najvažniji pokazatelj njegove uspješnosti. Kada organizacije razvijaju pojedinačne proizvode, potrebno vrijeme je uglavnom konstanta[4], a najviše se odnosi na sam razvoj proizvoda. Međutim, kod primjene pristupa linija za proizvodnju softvera, vrijeme realizacije je u početku značajno duže, zbog razvoja artefakata za ponovnu upotrebu. Slika 13 prikazuje da se nakon uspješnog razvoja artefakata za ponovnu upotrebu vrijeme realizacije značajno smanjuje, budući da tada novi proizvodi mogu koristiti prethodno razvijene artefakte.

[4] U praksi postoje razlike u potrebnom vremenu za razvoj jednog proizvoda, međutim u svrhu prikazivanja značajnih razlika između razvoja jednog proizvoda i linije proizvoda, ova pretpostavka je dovoljno pouzdana.



Slika 13 Vrijeme realizacije (Pohl, Böckle, & Linden, 2005, str. 11)

Uspješna primjena linije za proizvodnju softvera, kao planskog i sistematskog pristupa ponovnoj upotrebi, mnogim organizacijama omogućila je postizanje njihovih poslovnih ciljeva, povećanje produktivnosti, smanjivanje troškova razvoja, brži izlazak na tržište, veću pouzdanost softvera, te konkurentsku prednost (Kang, Sugumaran, & Park, 2010).

3.2. Povijesni razvoj ponovne upotrebe softvera

Od samih početaka razvoja softvera bilo je evidentno da se troškovi razvoja, broj softverskih grešaka, te broj neuspješnih projekata konstantno, godinu za godinom, povećava. Mnoge organizacije su smatrale da se ti negativni trendovi mogu ublažiti ili potpuno okrenuti tako da se na jedan sustavan način prakticira ponovna upotreba (engl. *reuse*). Ponovna upotreba je proces razvoja softvera na način da se iskorištavaju postojeći artefakti radije nego da se softver razvija od samog početka (Krueger, 1992). Proces ponovne upotrebe se zagovara kao mehanizam za skraćivanje vremena potrebnog za razvoj, za povećanje produktivnosti te za smanjivanje broja grešaka (Holmes & Walker, 2012). U samim počecima pažnja kod ponovne upotrebe se svodila na manje cjeline pa je i sam uspjeh ponovne upotrebe ostao ograničen na pojedinačne slučajeve. Glavni ciljevi ponovne upotrebe su uglavnom bili automatiziranje razvoja softvera u pojedinačnim područjima korištenjem specifičnih programskih jezika za određenu domenu (Van Deursen, Klint, & Visser, 2000) i generatora izvornog programskog koda (Czarnecki & Eisenecker, 2000).

Pojam ponovne upotrebe u praksi razvoja poslovnih aplikacija dijelimo na dvije vrste: na dobro razvijenu *inter-organizacijsku* upotrebu resursa kao što su operativni sustavi, baze podataka, aplikacijski poslužitelji, itd., te na znatno slabije razvijenu *intra-organizacijsku* ponovnu upotrebu artefakata koji se razvijaju unutar organizacije (Bosch, 2009b). Predmet ovog istraživanja odnosi se na ponovnu upotrebu artefakata za razvoj poslovnih aplikacija koji se razvijaju unutar organizacija.

Ideja o intenzivnoj praksi korištenja ponovne upotrebe resursa u razvoju softvera postoji toliko dugo kao i sam razvoj softvera (F. J. Linden i ostali, 2010), dok prva znanstvena istraživanja iz područja ponovne upotrebe softvera potječu sa NATO Software Engineering Conference iz 1968. godine, koja se ujedno smatra kao godina nastajanja programskog inženjerstva. Tema navedene konferencije je bila „*Softverska kriza - problem razvoja velikih, pouzdanih sustava na kontroliran i isplativ način*“. McIlroy je kao autor najznačajnijeg rada na toj konferenciji iz područja ponovne upotrebe, u svome radu *Mass Produced Software Components* (McIlroy, 1968) prvi predložio upotrebu knjižnice komponenata za numeričke izračune, obradu teksta, i alokaciju memorijskog prostora. Godinama kasnije, veliki broj znanstvenika još uvijek smatra ponovnu upotrebu tehnikom za poboljšanje procesa razvoja softvera. Usprkos očekivanjima, ponovna upotreba još uvijek nije postala uobičajena praksa kod razvoja softvera (Krueger, 1992). Navedeni problemi primjene tehnika ponovne upotrebe ponukali su mnoge znanstvenike da se i dalje intenzivno bave ovim područjem. Većina znanstvenih istraživanja se odnosi na pristup unaprijed planirane ponovne upotrebe: objektno-orijentirano nasljeđivanje, softverske komponente, linije za proizvodnju softvera, koji pored prednosti ima i značajne nedostatke (Holmes & Walker, 2012). Za razliku od planirane ponovne upotrebe, programeri se u praksi najčešće nalaze u situaciji da im je novi razvoji zadatak već odnekuda poznat, ili su ga sami već ranije programirali ili imaju pristup izvornom programskom kodu koji obuhvaća funkcionalnost razvojnog zadatka. U takvoj situaciji programer se može odlučiti za jednu od tri opcije: razvoj iz početka, prepravljanje postojećeg rješenja kako bi se moglo koristiti na ponovno upotrebljiv način, napraviti kopiju postojećeg rješenja i prilagoditi je zadatku. Svaka od tri nabrojene opcije ima značajnih nedostataka u praksi (Kapser & Godfrey, 2008; Opdyke, 1992). Međutim, za učinkovito korištenje ponovne upotrebe potreban je strateški pristup koji u obzir uzima jedinstvene zahtjeve i mogućnosti koje ova tehnika pruža (Abran, Moore, Bourque, Dupuis, & Tripp, 2001, str. 8–1).

Ponovna upotreba u organizaciji se može „dogoditi“ slučajno, neplanski, ili može biti rezultat planskog pristupa koji ujedno podrazumijeva i druge značajne promjene u načinu razvoja

softvera. Potrebni su dodatni napor i troškovi kako bi se razvijao, testirao, i dokumentirao ponovno upotrebljivi softver. U procesu razvoja nužno je uvesti nove procesne aktivnosti čija svrha je pronalaženje prilika i načina ponovne upotrebe postojećih komponenata. Troškovi razvoja softvera tj. njihova struktura se mijenja uslijed činjenice da se troškovi i eventualne koristi ponovne upotrebe ne odnose samo na jedan projekt. Planski pristup zahtjeva značajno investiranje u samom početku, dok se koristi od njega mogu očekivati tek kasnije, kod razvoja novih proizvoda ili pokretanja novih projekata. Prema (Leach, 2012) postoje mnogi primjeri uspješne primjene tehnika ponovne upotrebe:

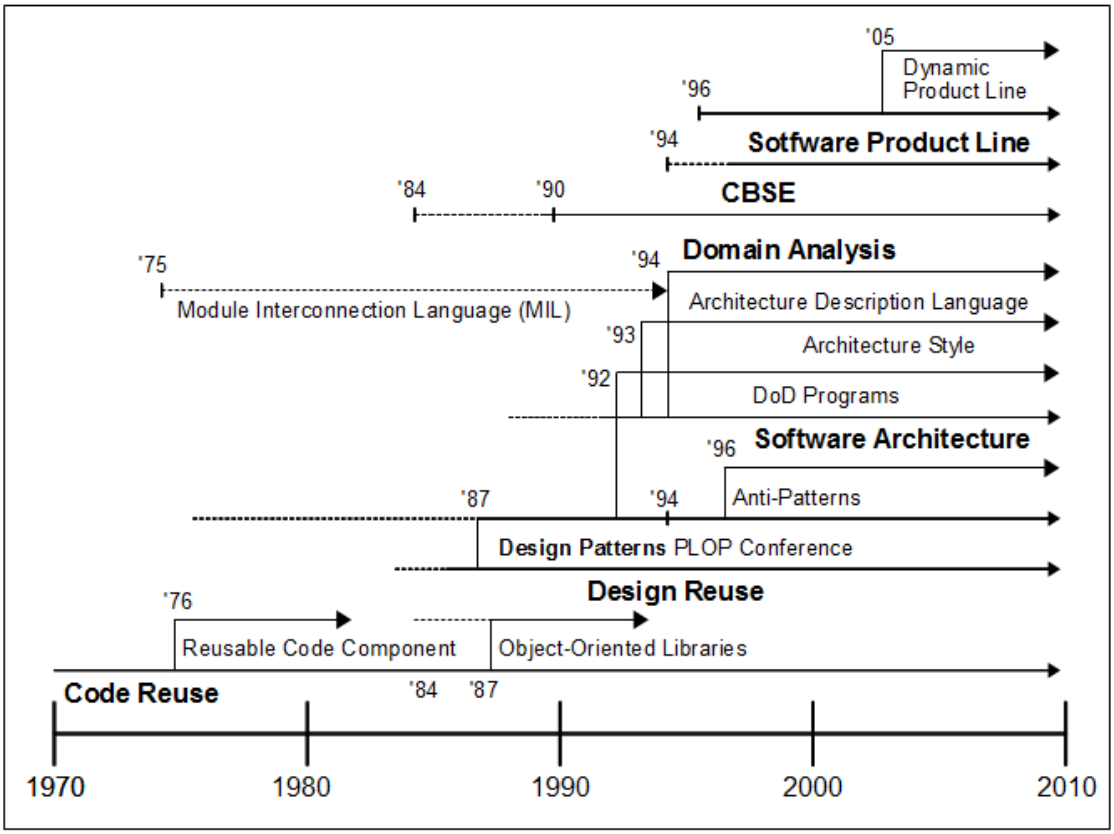
- Na preko tri stotine projekata u zrakoplovnoj industriji povećana je produktivnost za 20%, smanjen je broj pritužbi klijenata za 20%, za 25% su skraćeni rokovi za ispravljanje grešaka, te je za 25% smanjeno vrijeme razvoja sustava.
- U istraživanju na uzorku japanske industrije utvrđeno je povećanje produktivnosti 15-50%, 20-35% smanjenje broja pritužbi klijenata, 20% smanjenje u troškovima obrazovanja, 10-50% smanjenje vremena za razvoj sustava.
- NASA je smanjila potreban napor za razvoj i cijenu razvoja softvera za 75%.
- Na projektu razvoja simulatora za američku mornaricu povećana je produktivnost za 200% u broju linija izvornog programskog koda.

Na navedenim primjerima vidimo da je ponovna upotreba softvera tehnika koja može imati značajan pozitivni utjecaj na razvoj softvera općenito.

Poznate su mnoge tehnike iz područja ponove upotrebe: aplikativni okviri (engl. *frameworks*), uzorci dizajna (engl. *design patterns*), komponente, generatori izvornog koda, itd. Navedene tehnike nisu prikladne za velike sustave koji se u današnje vrijeme složenih sustava nužno trebaju integrirati sa sustavima iz drugih poslovnih domena kako bi podržavali uobičajene korisničke zahtjeve. Slika 14 prikazuje povijest ponovne upotrebe od jednostavnog korištenja postojećeg izvornog koda do linija za proizvodnju softvera.

Pristup koji se smatra najučinkovitijim u području ponovne upotrebe, linija za proizvodnju softvera, umjesto standardne ponovne upotrebe individualnih komponenata, omogućava ponovnu upotrebu softverske arhitekture i pripadajućih joj artefakata: korisničkih zahtjeva, referentne arhitekture, dijagrama modela, korisničke dokumentacije, projektnih i testnih planova, korisničkih sučelja, testne dokumentacije, i drugih artefakata. Softverska industrija

sve više prepoznaje strateški karakter i važnost linija za proizvodnju softvera (Clements & Northrop, 2002c).



Slika 14 Povijest ponovne upotrebe (Kang i ostali, 2010, str. 5)

3.3. Opis pristupa linija za proizvodnju softvera

Linije za proizvodnju softvera predstavljaju pristup koji se uspješno koristi u raznim područjima, omogućavajući dionicima optimiziranje procesa proizvodnje i raspoloživih resursa kroz identifikaciju i ponovnu uporabu zajedničkih artefakata koji se koriste u više proizvoda. Primjerice, u proizvodnji mobilnih telefona Nokia koristi liniju za proizvodnju mobilnih telefona. Cannon i Hewlett Packard koriste isti pristup za proizvodnju pisača. U softverskom inženjerstvu, ponovna upotreba gotovih artefakata predstavlja veliki napredak. Ideja intenzivnog iskorištavanja gotovih artefakata u razvoju softvera, jednako je stara kao i smo softversko inženjerstvo (F. J. Linden i ostali, 2010). Međutim, budući da su prvi napori u tom smjeru bili obično usmjereni na ponovnu upotrebu manjih dijelova softvera za vrijeme razvoja, rezultati tih napora bili su prilično ograničeni. Koncept pod nazivom „obitelji proizvoda“ (engl. *product families*) prvi puta se spominje 1976. godine (Parnas, 1976) na

temelju ranijih ideja E.W. Dijkstra (Dijkstra i ostali, 1970) vezano za varijabilnost nefunkcionalnih karakteristika softvera. Najčešće se smatra da porijeklo linija za proizvodnju softvera seže do 1977. godine kada je u Japanu pod nazivom *Toshiba Software Factory* kod razvoja softvera za električne generatore korišten pristup sličan današnjem pristupu linija za proizvodnju softvera (Matsumoto, 1987). Međutim, tek nakon 15 godina, početkom 1990-tih godina koncept linije za proizvodnju softvera je široko prihvaćen, a mnoge organizacije su počele na sustavan način pristupati ovom problemu. Primjerice, Fhilips je predstavio metodu pod nazivom *Buildng Block* početkom 1990-tih (F. J. Van Der Linden & Muller, 1995); jedan od prvih znanstvenih radova iz ovog područja (Kang i ostali, 1990) kojim je predstavljena metoda za analizu funkcionalnosti Feature Oriented Domain Analysis (FODA) objavljen je 1990. godine; Europska Unija je financirala niz znanstvenih projekta iz ovog područja:

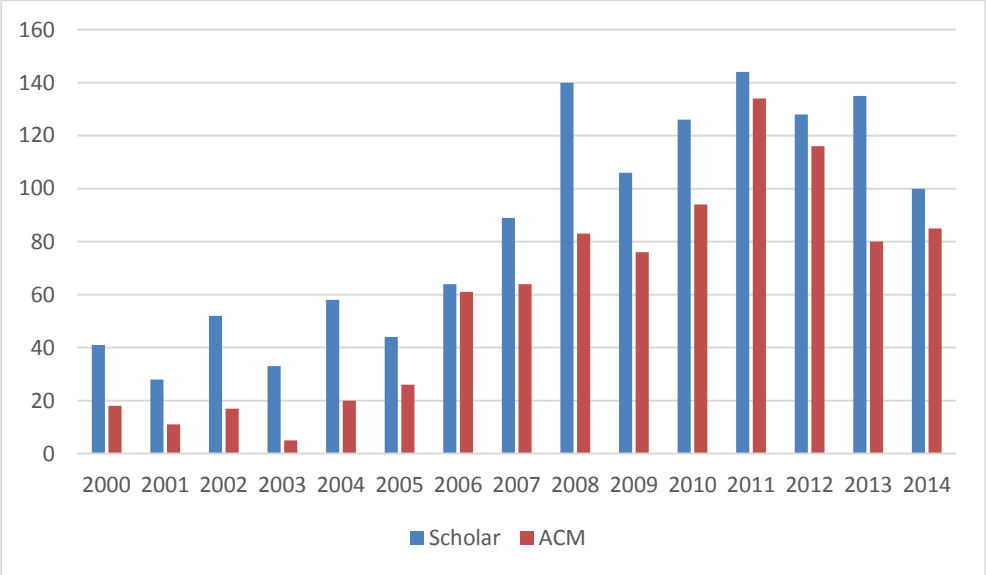
- ARES (1995) – Architectural Reasoning for Embedded Systems.
- Praise (1998) – Product-line Realization and Assessment in Industrial Settings.
- ESAPS (1999) – Engineering Software Architectures, Processes and Platforms.
- CAFE (2001) – From Concepts to Application in system-Family Engineering.
- FAMILIES (2003) – Fact-based Maturity through Institutionalisation, Lessons-learned and Involved Exploration of System-family engineering.

Navedeni projekti značajno su doprinijeli sustavnom istraživanju ovog područja i praksi primjene linija za proizvodnju softvera u Europi. U isto vrijeme u SAD-u najveći doprinos popularizиranju ovog pristupa dao je institut Software Engineering Institute (SEI), najčešće vezano uz projekte vladinih organizacija.

U znanstvenoj literaturi linije za proizvodnju softvera počinju se značajnije pojavljivati tek od kasnih 1990-tih godina. Pretraživanje *ACM Digital Library* i *Google Scholar*, elektroničkih baza sa vjerojatno najvećim brojem znanstvenih i stručnih naslova, koje smo proveli u okviru ovog istraživanja, daje nam pregled broja objavljenih radova po godinama. Filtere kod pretraživanja znanstvenih radova definirali smo na slijedeći način: s točnim izrazom „*software product lines OR software product line*“ u naslovu članka; vrati članke datirane u razdoblju od jedne godine (od 2000. godine do 2014. godine); isključi patente i citate; datum pretraživanja je 3.4.2015. godine.

Slika 15 prikazuje da je najveći broj radova objavljen u razdoblju od 2008. do 2013. godine. Interes znanstvene zajednice vezano za područje linija za proizvodnju softvera najčešće se

odnosi na: knjižnice artefakata (engl. *asset libraries*), alate i metode, referentne arhitekture, generatore izvornog koda, upravljanje konfiguracijama, mjerenje i eksperimentiranje, te poslovne i organizacijske izazove. Posebno je aktualno područje upravljanja varijabilnošću (engl. *variability management*), integracija sa aspekt-orijentiranom tehnologijom, te razvoj referentnih arhitektura poput .NET, EJB, i Service Oriented Architecture (SOA).



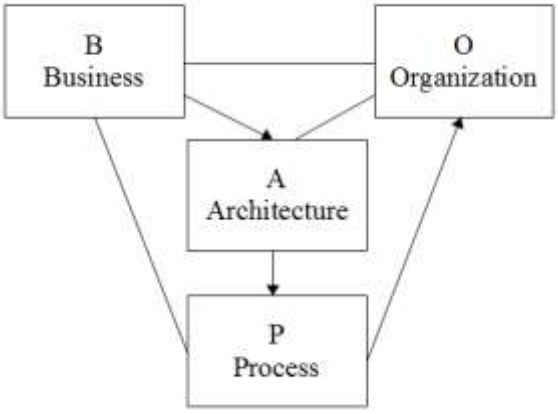
Slika 15 Broj objavljenih znanstvenih i stručnih radova

Za opis pristupa linije za proizvodnju softvera u ovom radu koristimo BAPO (engl. ***B**usiness **A**rchitecture, **P**rocess, **O**rganization*) model. BAPO model se već više puta koristio u kontekstu linija za proizvodnju softvera (F. J. Linden i ostali, 2010, str. 16; Wijnstra, 2002). Slika 16 ilustrira BAPO model koji područje softverskog inženjerstva prikazuje iz četiri različite perspektive: poslovna (engl. b*usiness*), arhitektura (engl. *architecture*), proces (engl. *process*), organizacija (engl. *organization*).

- *Poslovna*: troškovi i dobit od softverskih proizvoda, strateški način uvođenja u organizaciju.
- *Arhitektura*: u kontekstu linija za proizvodnju softvera, gdje više aplikacija dijeli zajedničku arhitekturu, arhitekturu nazivamo i referentna arhitektura koja predstavlja tehnička sredstva za razvoj softvera s ciljem potpore poslovanju organizacije, pri čemu je glavni fokus na obuhvatu zajedničkih funkcionalnosti aplikacija i upravljanju njihovim razlikama na efektivan način. Definiciju arhitekture u ovom radu smo proširili

(poglavlje 2.3.3) kako bi obuhvatili više poslovnih sektora za koje razvijamo predloženu referentnu arhitekturu (RABA).

- *Proces*: uloge, odgovornosti i odnosi u procesu razvoja softvera.
- *Organizacija*: organizacijske strukture za razvoj i održavanje softvera.



Slika 16 Perspektive – *BAPO* (F. J. Linden i ostali, 2010, str. 16; Wijnstra, 2002)

Različite perspektive su međusobno isprepletene. Promjene u jednoj uzrokuju nužne promjene u drugoj perspektivi. Najutjecajnija je poslovna perspektiva. Arhitektura uzima u obzir poslovne aspekte, ugrađuje ih u svoju strukturu i principe. Proces omogućava razvoj softvera na temelju definirane arhitekture. Organizacija je način rasporeda odjela i osoblja prema odgovornostima za poslovanje, arhitekturu i proces.

3.3.1. Poslovna perspektiva

Iz iskustva primjene pristupa linija za proizvodnju softvera u mnogim organizacijama, možemo tvrditi da ovaj pristup ima značajan utjecaj na poslovanje tih organizacija (Ahmed & Capretz, 2007). Pristup predstavlja osnovu za efikasniji, jeftiniji, kvalitetniji i produktivniji razvoj aplikacija za cjelokupno tržište neke organizacije. Sveobuhvatnost ovog pristupa pretpostavlja da je usklađenost sa poslovnom strategijom organizacije ključan čimbenik koji umnogome utječe na uspjeh organizacija koje ga uvod u praksu (Wijnstra, 2002). Glavni ciljevi uvođenja odnose se na definiranje zajedničke referentne arhitekture za više aplikacija, postizanje optimalnije organizacije resursa, te na uspostavu efikasnijih procesa za razvoj i održavanje aplikacija.

Osim standardnog načina uvođenja ovog pristupa u praksu, organizacije sve više prihvaćaju trend uključivanja drugih dionika izvan svojih organizacijskih granica u razvoj referentne arhitekture i komponenata, što zapravo predstavlja prijelaz na prošireni pristup koji se najčešće naziva „*software ecosystem*" (Bosch, 2009b; Jansen & Cusumano, 2013). *Software ecosystem* pristup koji uključuje vanjske dionike u razvoj proizvoda omogućava organizacijama brži razvoj inovativnih rješenja, povećanje vrijednosti postojećih proizvoda, i mnoge druge prednosti.

Odluku o uvođenju potrebno je donijeti na temelju valjanih poslovnih razloga. Oni su ključni čimbenik u odlučivanju koje treba uzeti u obzir kod razmatranja slijedećih pitanja (F. J. Linden i ostali, 2010) :

- Da li se ovaj pristup uvodi u organizaciju ili ne?
- Kakav je njegov utjecaj na učinkovitost poslovanja organizacije?
- Koji proizvodi će biti uključeni u proces?
- Koje funkcionalnosti će imati ti proizvodi?
- Kojim redoslijedom će se proizvodi plasirati na tržište?
- Koje funkcionalnosti proizvoda su zajedničke za sve a koje su specifične za pojedine proizvode?
- Kada i kako uvesti ovaj pristup u organizaciju?
- Na koji način će se razvijati primjena pristupa u organizaciji?

Odgovori na navedena pitanja najviše ovise o ciljevima i tržištu proizvoda na koje se odnosi linija za proizvodnju softvera. U praksi je relativno teško uvesti *linije za proizvodnju softvera*; razlozi poput uobičajenog otpora razvojnih timova, poteškoća u razumijevanju paradigme linija za proizvodnju, nefleksibilnosti organizacijske strukture, utječu na nepredvidivost potrebnog vremena i napora. Neke studije slučajeva su pokazale da tranzicija pod takvim okolnostima može trajati od 2 do 5 godina. Međutim, u situacijama gdje su se na primjeren način otklanjale navedene prepreke, tranzicija je trajala vrlo kratko, svega nekoliko mjeseci (Krueger, 2006). Iste studije slučajeva pokazale su da su vrijeme i napor za tranziciju manji ukoliko se ponovna upotreba temelji na dobro oblikovanim postojećim artefaktima umjesto na razvoju potpuno novih artefakata. Mnogi pokušaji uvođena ne uspijevaju postići zacrtane ciljeve, ili u potpunosti propadnu zbog neadekvatne analize raznih utjecaja i eventualnih problema (Wijnstra, 2002).

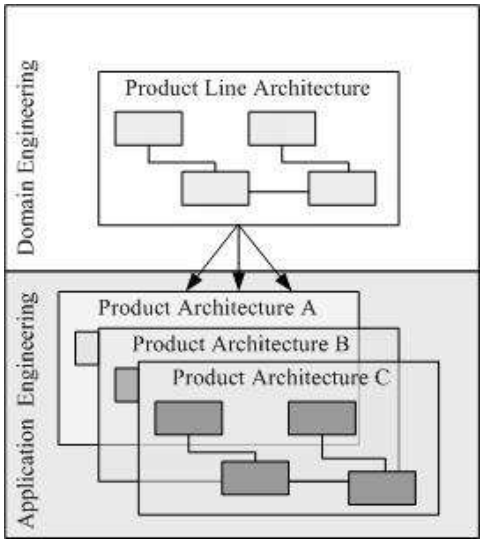
Početni troškovi uvođenja te dugi rok potencijalnih isplativosti zahtjeva strateški pristup i potpunu usklađenost sa poslovnom strategijom organizacije. Jedan od ključnih koraka kod pripreme uvođenja je procjena rizika, kako bi se na temelju nje odredile mjere za smanjivanje potencijalnih rizika kod uvođenja pristupa u organizaciju. Pored procjene eventualnih rizika uvođenja, važno je provesti analizu zrelosti organizacije za uvođenje *linije za proizvodnju softvera* korištenjem neke od metodika kao što su *Family Evaluation Framework* (FEF) ili SEI *Product Line Technical Probe* (PLTP).

Veliki potencijalni utjecaj linija za proizvodnju softvera na poslovanje organizacija i obrnuto upućuje na to da bi u idealnoj situaciji poslovna strategija trebala biti pokretač odluka u području tehnologija za razvoj softvera, dok bi se s druge strane poslovna strategija trebala biti pod utjecajem tehnoloških prilika ili teškoća koje se eventualno mogu pojaviti.

3.3.2. Perspektiva arhitekture

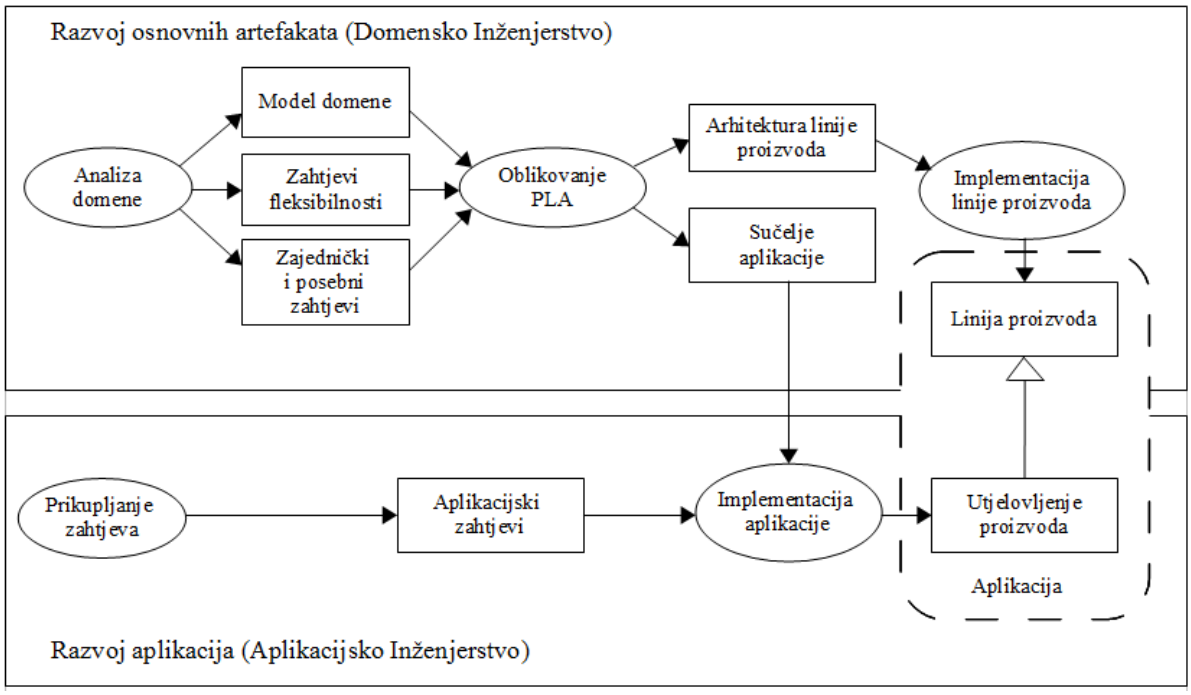
Arhitektura softverskog proizvoda ili aplikacije nastaje na temelju funkcionalnih i nefunkcionalnih korisničkih zahtjeva. Za razliku od pojedinačnih aplikacija, arhitektura linije za proizvodnju softvera (engl. *product line architecture*) odnosi se na grupu aplikacija koje imaju sličnu funkcionalnost i strukturu. *Product line architecture* (PLA) predstavlja „*temeljnu organizaciju sustava koji se sastoji od komponenata, njihovih međusobnih odnosa i odnosa prema okolini, te principa koji se koriste pri njihovom razvoju i evoluciji*“ (IEEE1471,2000). Proces razvoja zajedničke arhitekture prema pristupu linija za proizvodnju softvera omogućava efikasniju raspodjelu resursa za razvoj i efikasniju ponovnu upotrebu razvijenih komponenata kod razvoja novih proizvoda. PLA ima ključnu ulogu kod razvoja proizvoda, budući da predstavlja apstrakciju proizvoda koji se na temelju nje mogu razvijati, te stoga što u sebi uključuje sve zajedničke i posebne karakteristike linije za proizvodnju softvera (Junior i ostali, 2013). PLA definira način organizacije komponenata i njihovu povezanost, principe i smjernice za implementaciju i razvoj proizvoda (IEEE, 2000).

Slika 17 prikazuje proces u kojem nastaje referentna arhitektura (engl. *reference architecture*) kao rezultat aktivnosti koje nazivamo „domensko inženjerstvo“ (engl. *domain engineering*) u okviru procesa razvoja prema pristupu linija za proizvodnju softvera. Glavni artefakti ovog procesa su osnovni elementi (engl. *core assets*) koji osim komponenata uključuju i druge ponovno upotrebljive artefakte. Skup osnovnih elementa referentne arhitekture također nazivamo i „platforma“ (engl. *platform*).



Slika 17 Definiranje arhitekture proizvoda (Parra, 2011)

Slika 18 prikazuje osnovne aktivnosti procesa razvoja referentne arhitekture i aplikacija. Svrha procesa koji se naziva „razvoj osnovnih artefakata“ (engl. *domain engineering)* je definiranje, oblikovanje i implementacija artefakata referentne arhitekture za poslovno područje na koje se odnosi linija za proizvodnju softvera. Ukoliko se to odnosi na poslovne aplikacije općenito, za više poslovnih sektora, potrebno je razinu apstrakcije prilagoditi zajedničkim karakteristikama njihovih aplikacija. Situacija u kojoj veći broj aplikacija pripada različitim poslovnim sektorima, za razliku od većine slučajeva kada se linija za proizvodnju softvera odnosi samo na jedan poslovni sektor, značajno je zahtjevniji slučaj po pitanju referentne arhitekture. Kada je u pitanju širi djelokrug aplikacija koje koriste istu referentnu arhitekturu, govorimo o aplikacijskoj populaciji (engl. *product population*) prema definiciji (Van Ommering, 2002). Referentna arhitektura za aplikacijsku populaciju tijekom vremena često zahtjeva prilagodbu ili eventualnu podjelu na više linija ukoliko se prilagodbom razine apstrakcije ne mogu ostvariti isti ciljevi. Komponente poslovnih aplikacija koje su namijenjene rješavanju konkretne poslovne logike pojedinog poslovnih sektora, razvijaju se u okviru procesa koji se naziva „razvoj aplikacija“ (engl. *application engineering*), a ukoliko se koriste u više aplikacija, održavamo ih u procesu koji se naziva „razvoja osnovnih artefakata“.



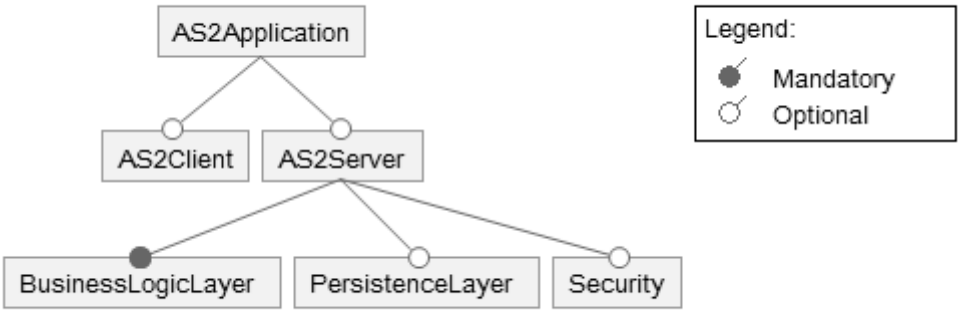
Slika 18 Aktivnosti procesa razvoja i njihov odnos (Tirila, 2002, str. 9)

Referentna arhitektura se koristi kao artefakt ponovne upotrebe kod razvoja aplikacija u sklopu aktivnosti koju nazivamo „aplikativno inženjerstvo“ (engl. *application engineering*). U tom procesu referentna arhitektura se može proširiti kako bi se definirala specifična arhitektura pojedinačne aplikacije. Proširivanje se radi u skladu sa varijacijskim točkama (engl. *variation points*) pomoću kojih se definiraju pozicije u referentnoj arhitekturi koje omogućavaju prilagodbu arhitekture i njenih raspoloživih funkcija svakoj aplikaciji posebno (Jacobson, Griss, & Jonsson, 1997). Funkcionalnosti koje su zajedničke za sve proizvode implementiraju se u samoj referentnoj arhitekturi, dok ostale funkcionalnosti koje su specifične za pojedine proizvode, nisu nužno dio referentne arhitekture, već je njihova implementacija u referentnoj arhitekturi samo unaprijed predviđena.

Dijagrami funkcionalnosti

Za analizu zahtjeva referentne arhitekture uobičajeno se koristi koncept funkcionalnost (engl. *feature*) i dijagram funkcionalnosti (engl. *feature diagram*) koje je prvi uveo (Kang i ostali, 1990). Slika 19 ilustrira primjer upotrebe dijagrama funkcionalnosti pomoću kojeg se definira organizacija funkcionalnosti koje su zajedničke za više proizvoda. Pomoću dijagrama funkcionalnosti moguće je povezivati zahtjeve korisnika sa artefaktima referentne arhitekture i proizvodima koji te artefakte koriste. Tehnički gledano, to se odnosi na zahtjeve, dizajn,

programiranje, testiranje, sve do kraja procesa. Osim funkcionalnosti referentne arhitekture, ovim konceptom se mogu obuhvatiti i komponente koje ne pripadaju referentnoj arhitekture već se odnose na komponente poslovne logike, što u konačnici omogućava njihovu selekciju pri razvoju aplikacija za krajnje korisnike.



Slika 19 Primjer dijagrama funkcionalnosti

Vrednovanje referentne arhitekture

Vrednovanje referentne arhitekture je važan korak procesa njenog oblikovanja i razvoja. Za formalno vrednovanje referentne arhitekture najčešće se koriste standardne metode ATAM, CBAM i SAAM[5] pri čemu je potrebno najviše pažnje posvetiti načinu i mehanizmima za upravljanje varijabilnošću (engl. *variability*). Ukoliko se to odnosi na kontekst poslovnih aplikacija općenito, iz više poslovnih sektora, potrebno je ispitati da li je predložena arhitektura previše općenita, i da li se pokušava riješiti previše funkcija odjednom, na pogrešnoj razini apstrakcije?

Referentna arhitektura po svojoj prirodi je promjenjiva, potrebno ju je prilagoditi novim tehnološkim, tržišnim i zahtjevima korisnika. Njena kvaliteta najviše ovisi o tome da li ju je lako prilagoditi tim zahtjevima. Osim izmjena postojećih i dodavanja novih, često je potrebno ukloniti neke od postojećih funkcionalnosti. Proces održavanja postojeće referentne arhitekture zahtjeva planski pristup i dokumentiranje promjena kako ne bi došlo do njene erozije (engl. *architectural erosion*) koja se događa ukoliko se uslijed velikog broja malih izmjena one sustavno ne dokumentiraju. Isto tako, ukoliko prilagodba referentne arhitekture novim

[5] Architecture Trade-off Analysis Method (ATAM), Cost Benefit Analysis Method (CBAM) i Software Architecture Analysis Method (SAAM)

zahtjevima postane preskupa, to ujedno znači i kraj referentne arhitekture kao takve, koja tada može poslužiti jedino kao izvor potrebnog znanja za razvoj nove referentne arhitekture.

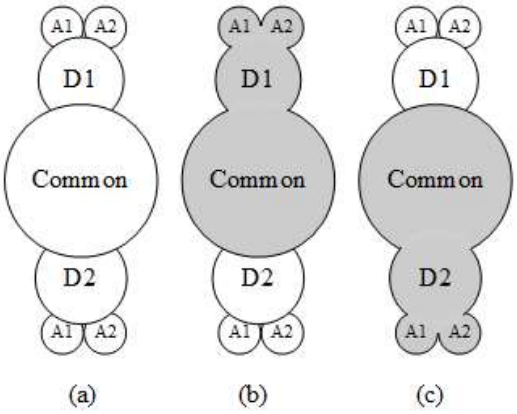
3.3.2.1. Arhitektura i poslovne aplikacije

Poslovne aplikacije se međusobno razlikuju po različitim karakteristikama kao što je veličina, namjena, kompleksnost, potencijalni korisnici. Razlike postoje i u svakom kompleksnijem softverskom sustavu, primjerice, u auto industriji, mobilnoj telefoniji. Automobili i mobilni telefoni su proizvodi dvaju potpuno različitih poslovnih sektora (engl. *domain*) koji zahtijevaju drugačije vještine dionika koji sudjeluju u razvoju njihovih softverskih sustava. Svaki od navedenih poslovnih sektora koristi specifične procese, tehnologiju, principe, metode i alate koji su se godinama usavršavali. Mala je vjerojatnost da se neka od rješenja iz jednog poslovnog sektora koriste u drugome. Svaki poslovni sektor se dalje može dijeliti na pod sektore (engl. *subdomains*). Primjerice, u auto industriji, različiti pod sektori mogu obuhvaćati osobne, putničke i teretne automobile. Međutim, unutar jednog poslovnog sektora i njegovih pod sektora, postoji i veliki broj zajedničkih karakteristika koje se preklapaju te ih je potrebno obuhvatiti na učinkovit način. Pristup koji se za to najčešće koristi, naziva se *Domain Specific Software Engineering* (DSSE) koji u obzir uzima postojeća dostignuća u specifičnom poslovnom sektoru (Bryant, Gray, & Mernik, 2010).

Poslovne aplikacije koje pripadaju različitim poslovnim sektorima, također imaju veliki broj zajedničkih karakteristika koje se mogu obuhvatiti na jedinstveni način. Umjesto da se za svaki poslovni sektor razvija posebna referentna arhitektura, rješenje se može pronaći u prilagodbi jedinstvene referentne arhitekture svakom od poslovnih sektora. Takvo rješenje odnosi se na referentnu arhitekturu koja opisuje koncept razvoja poslovnih aplikacija, knjižnicu komponenata te metodu za izbor i konfiguraciju raspoloživih komponenata.

Slika 20 pomoću Vennovog dijagrama (Richard N. Taylor i ostali, 2009, str. 594) prikazuje arhitekturu linije za proizvodnju poslovnih aplikacija koja obuhvaća dva poslovna sektora. Veliki krug u sredini predstavlja zajedničku referentnu arhitekturu i njene komponente. Manji krug „D1“ predstavlja poslovne komponente koje su specifične za jedan poslovni sektor, primjerice trgovinu. Manji krug „D2“ predstavlja poslovne komponente koje su specifične za drugi poslovni sektor, primjerice turizam. Slika 20 (a) prikazuje kontekst korištenja zajedničke referentne arhitekture. Na slici 20 (b) istaknut je kontekst koji obuhvaća poslovni sektor D1: uniju zajedničke referentne arhitekture, komponenata poslovnog sektora D1 i aplikacije A1,

A2. Slika 20 (c) ilustrira kontekst koji obuhvaća poslovni sektor D2: uniju zajedničke referentne arhitekture, komponenata poslovnog sektora D2 i aplikacije A1, A2.



Slika 20 Koncept PLA za poslovne aplikacije

Za prilagođavanje standardne referentne arhitekture kontekstu linija za proizvodnju najčešće se koristi tehnika koju nazivamo varijabilnost (engl. *variability*).

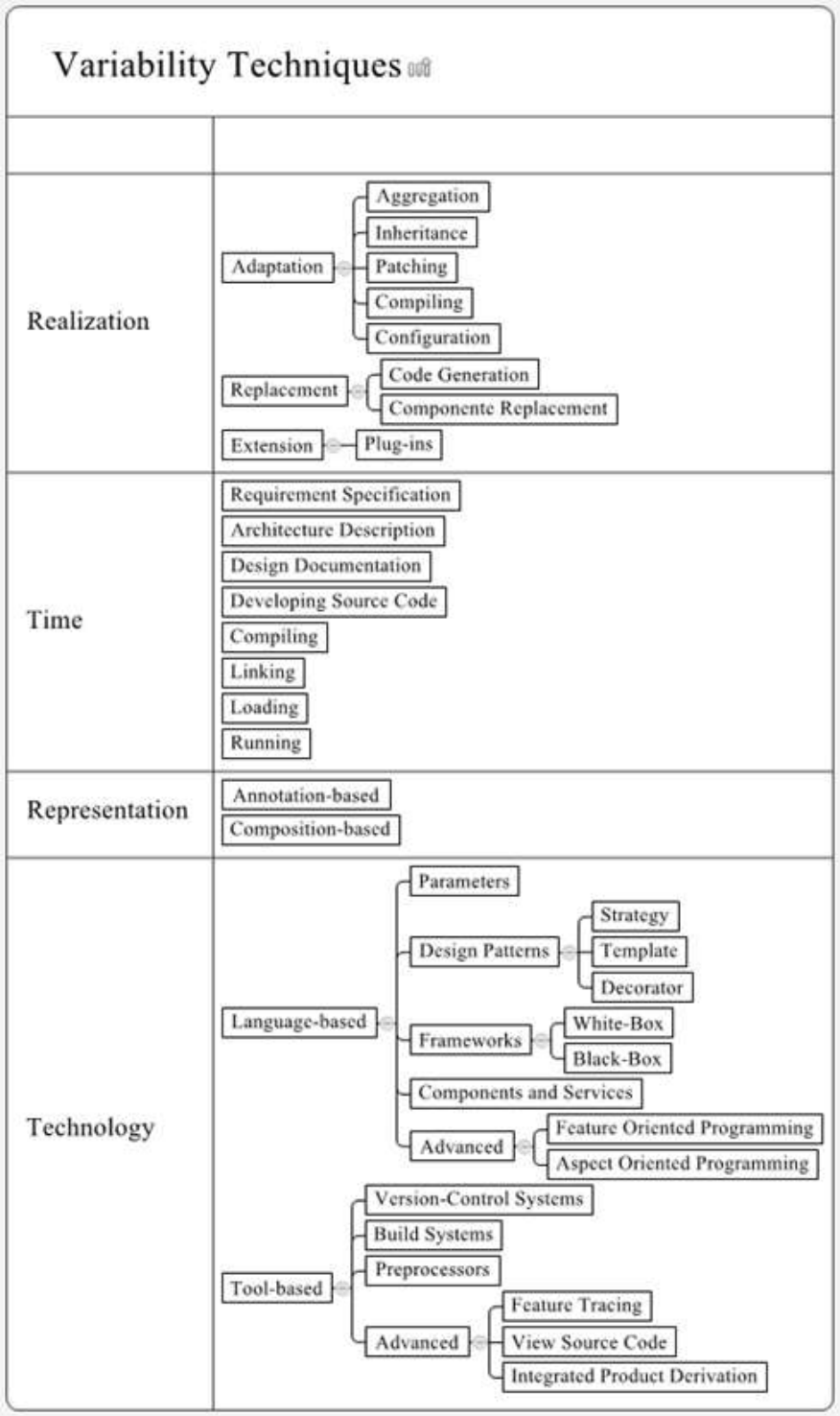
3.3.2.2. Varijabilnost

Razvoj i održavanje linija za proizvodnju softvera značajno se oslanja na upotrebu tehnika varijabilnosti pomoću kojih se upravlja sa razlikama između proizvoda koji se temelje na zajedničkoj referentnoj arhitekturi; na način da se odluke o dizajnu odgađaju za kasnije faze razvoja proizvoda i njihovog korištenja. Varijabilnost u kontekstu linija za proizvodnju softvera je ključna karakteristika referentne arhitekture, definiramo je kao „*sposobnost softverskog sustava ili artefakta da se učinkovito proširuje, mijenja, prilagođavao ili konfigurira za upotrebu u određenom kontekstu*“ (Svahnberg, Van Gurp, & Bosch, 2005); opisuje se pomoću točaka varijabilnosti i raspoloživih varijanti. U ovom radu varijabilnost referentne arhitekture ima poseban značaj jer se odnosi na širi kontekst upotrebe, za više poslovnih sektora, stoga smo glavne točke varijabilnosti definirali na horizontalnoj razini referentne arhitekture koja se odnosi na različite poslovne sektore. Takav pristup varijabilnosti nije uobičajen u području linija za proizvodnju softvera, najčešće se varijabilnost definira za jedan poslovni sektor (vertikalno), na nižoj razini apstrakcije, što predstavlja manju kompleksnost u odnosu na horizontalnu razinu na kojoj definiramo varijabilnost u ovom radu.

Varijabilnost korištenja softverskih komponenata postaje sve značajnija karakteristika u području razvoja softvera. Nekada je softver bio relativno statičan, za svaku izmjenu bilo je potrebno mijenjati izvorni programski kod, danas to više nije prihvatljivo; uz pomoć modernih tehnika oblikovanja i pristupa razvoju softvera, moguće je vremensko odgađanje odluke o tome koje komponente i kakve karakteristike treba imati pojedini softverski proizvod. Tipičan primjer odgađanja takve odluke odnosi se na primjenu pristupa linija za proizvodnju softvera. Umjesto da se unaprijed definiraju proizvodi koji će se razvijati, u kontekstu linije za proizvodnju softvera, unaprijed se definira i implementira samo referentna arhitektura i skup ponovno upotrebljivih komponenata, kako bi se kasnije omogućilo konfiguriranje većeg broja proizvoda koji će koristiti te komponente i naslijediti tu referentnu arhitekturu. Referentna arhitektura je mjesto gdje se implementira varijabilnost linije za proizvodnju softvera pomoću varijabilnih točaka (engl. *variation points*). Osim standardnih varijabilnih točaka pomoću kojih omogućavamo izbor funkcija koje neka aplikacija može imati, moguće je definirati i varijabilne točke referentne arhitekture za odabir njenih strukturnih elemenata, za svaku aplikaciju posebno (Thiel & Hein, 2002). Slika 21 ilustrira „mentalnu mapu“ tehnika varijabilnosti koju smo u okviru ovog istraživanja definirali na temelju analize različitih tehnika koje se najčešće koriste u praksi; nazivi tehnika varijabilnosti na slici su obilježeni na izvornom jeziku iz dva razloga: jednostavnijeg prepoznavanja u referentnoj literaturi; jer ne postoji uvriježeni prijevod za veliki broj relativno novih pojmova. Implementaciju varijabilnosti možemo promatrati iz nekoliko različitih perspektiva:

- Način realizacije (engl. *realization*): razlikujemo tri različite tehnike realizacije varijabilnosti u referentnoj arhitekturi: prilagodba, zamjena, proširenje.
- Vrijeme primjene (engl. *time*): različite tehnike varijabilnosti omogućavaju primjenu u različito vrijeme, primjerice, u vrijeme kompiliranja, pokretanja aplikacije, za vrijeme izvođenja.
- Reprezentacija u kodu (engl. *representation*): koriste se dva pristupa, prvi (engl. *annotation*) i drugi (engl. *composition-based*). Izvorni programski kod može sadržavati iste elemente za sve aplikacije, pri čemu se tek u vrijeme izvođenja prepoznaju elementi koji su odabrani kao opcije u aplikaciji koja se izvodi (npr. *preprocessor* programskog jezika C). Drugi način odnosi se na odabir komponenata koje pripadaju aplikaciji prije njenog pokretanja (npr. *plug-in* komponente u Eclipse alatu za razvoj aplikacija). Isto tako moguća je kombinacija ovih dviju tehnika u praksi.

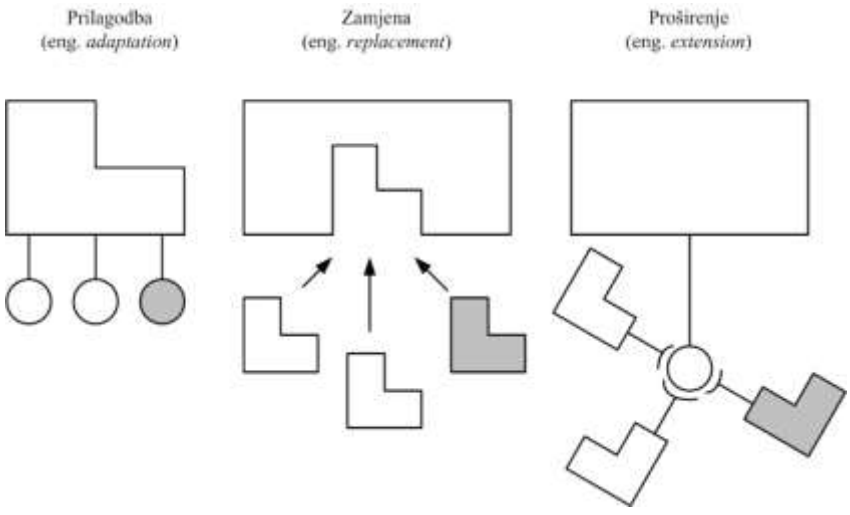
- Tehnologija koja se koristi (engl. *technology*): varijabilnost se može implementirati uz pomoć programskih jezika unutar izvornog programskog koda ili uz pomoć alata za konfiguraciju aplikacija.



Slika 21 Tehnike varijabilnosti

Slika 22 prikazuje tehnike načina realizacije (engl. *realization*) varijabilnosti arhitekture koje se dijele na slijedeće tehnike:

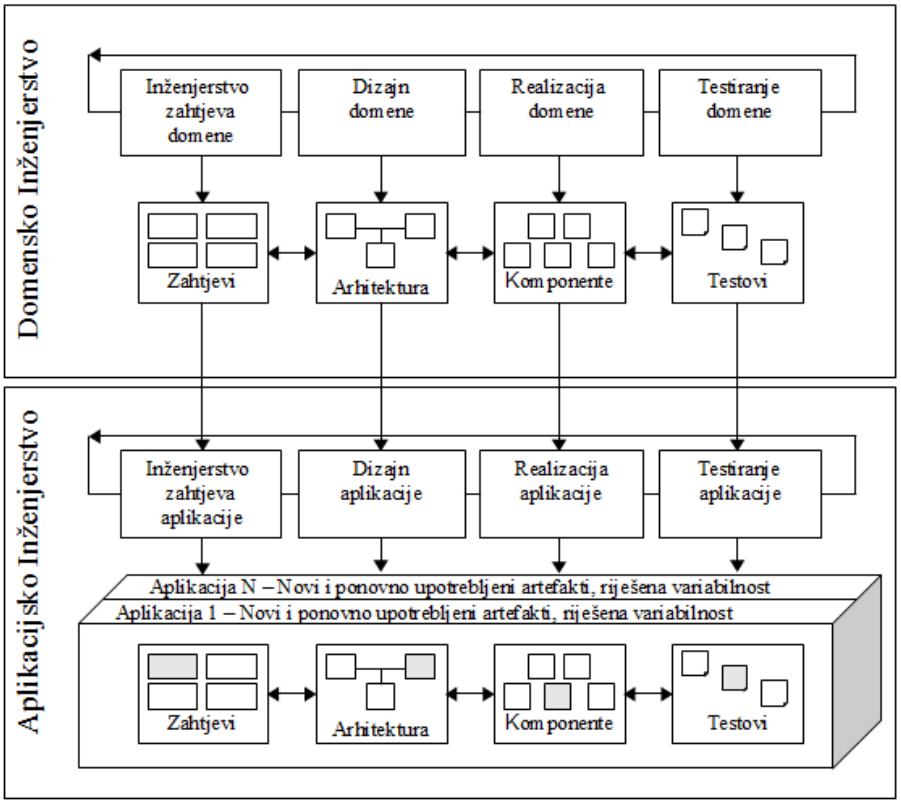
- Prilagodba (engl. *adaptation*): ova tehnika podrazumijeva postojanje samo jedne implementacije neke komponente koja je pritom oblikovana na način da je omogućena njena prilagodba specifičnim potrebama nekog od proizvoda koji ju koriste putem implementacije unaprijed definiranog sučelja (engl. *interface*).
- Zamjena (engl. *replacement*): u situaciji kada imamo nekoliko implementacija iste komponente koja je razvijena prema specifikacijama referentne arhitekture, ovom tehnikom omogućeno je da proizvod koristi neke od postojećih implementacija ili da se kroz aktivnosti aplikativnog inženjerstva razvije potpuno nova komponenta prema istim specifikacijama.
- Proširenje (engl. *extension*): putem ove tehnike omogućava se dodavanje novih komponenta tako što referentna arhitektura unaprijed definira generičko sučelje za dodavanje različitih vrsta komponenata. Ova tehnika se razlikuje od tehnike prilagodbe po tome što se tehnikom prilagodbe definira što komponenta radi dok se način implementacije može razlikovati, a tehnikom proširenja se umjesto samo jedne može dodati više komponenata bez unaprijed definirane funkcije koju će obavljati.



Slika 22 Tri tehnike realizacije varijabilnosti (F. J. Linden i ostali, 2010, str. 41)

3.3.3. Perspektiva procesa

Proces razvoja aplikacija prema pristupu linija za proizvodnju odvija se u dvije faze: fazu domenskog inženjerstva (engl. *domain engineering*) i fazu aplikacijskog inženjerstva (engl. *application engineering*) (F. J. Linden i ostali, 2010). Domensko inženjerstvo obuhvaća nekoliko aktivnosti; identifikaciju zajedničkih i posebnih karakteristika zadanog skupa aplikacija, implementaciju zajedničkih osnovnih artefakata za njihovo korištenje u razvoju aplikacija na ekonomičan način koji ujedno omogućava varijabilnost svake od aplikacija u odnosu na raspoložive funkcionalnosti artefakata koji se u njima koriste. U fazi aplikacijskog inženjerstva, pojedinačne aplikacije se razvijanju pomoću osnovnih artefakata koji su prethodno razvijeni u fazi domenskog inženjerstva. Domensko inženjerstvo odnosi se na razvoj **za** ponovnu upotrebu (engl. *development for reuse*) koje obuhvaća artefakte koji se uobičajeno nazivaju *infrastruktura* linije za proizvodnju, dok se aplikacijsko inženjerstvo odnosi na razvoj **sa** ponovnom upotrebom (engl. *development with reuse*). Rezultat domenskog inženjerstva je zajednička referentna arhitektura i ponovno upotrebljive komponente. Slika 23 prikazuje proces razvoja aplikacija u navedene dvije faze.



Slika 23 Dvije faze razvoja aplikacija (Pohl, Böckle, & Linden, 2005)

Uvođenje pristupa linija za proizvodnju softvera u organizaciju je isplativo samo ukoliko su koristi od ponovne upotrebe zajedničkih artefakata (infrastrukture) u fazi aplikacijskog inženjerstva veće od napora i troškova njihovog razvoja u fazi domenskog inženjerstva (Deelstra, Sinnema, & Bosch, 2005). Domensko i aplikacijsko inženjerstvo su komplementarni procesi čije aktivnosti se nužno ne provode u zadanom redoslijedu. Primjerice, moguće je razvijati osnovne artefakte na način da se oni identificiraju na temelju analize postojećih aplikacija. Ti isti artefakti se kasnije mogu koristiti za razvoj drugih aplikacija. Drugi način odnosi se na razvoj osnovnih artefakata iz početka. U tom slučaju domensko inženjerstvo prethodi procesu aplikacijskog inženjerstva. Svaki osnovni artefakt razvija se na način da se unaprijed planira njegova upotreba u više poslovnih aplikacija.

3.3.3.1. Domensko inženjerstvo

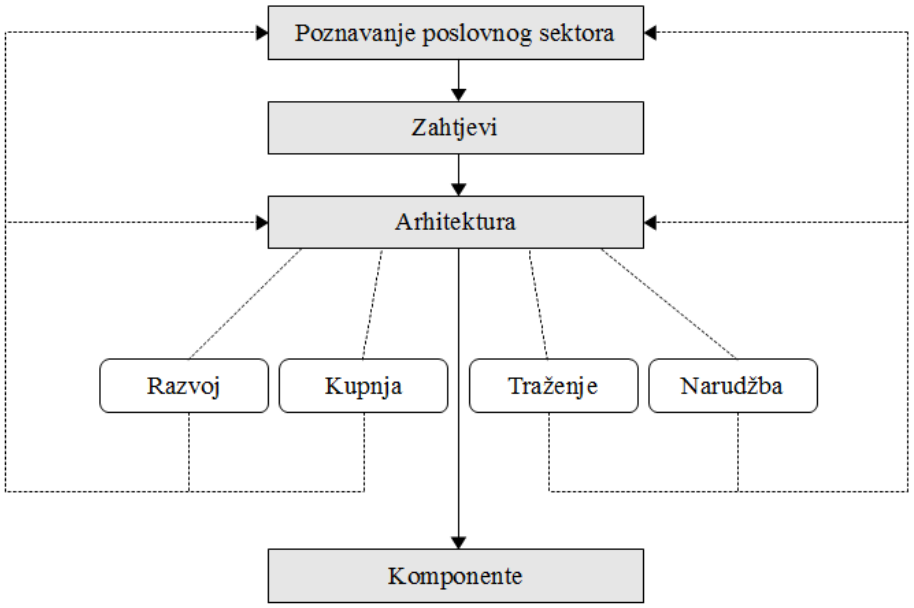
Domensko inženjerstvo je proces unutar kojeg se razvijaju osnovni artefakti koji zajedno sačinjavaju platformu linije za proizvodnju softvera. Platforma definira zajedničke komponente za aplikacije (engl. *commonality*) i pozicije varijabilnosti (engl. *variability*) na kojima se omogućava kombiniranje artefakata i izbor raspoloživih opcija. Platforma se sastoji od referentne arhitekture, osnovnih komponenata, testne dokumentacije, itd. Prema (Pohl, Böckle, & Linden, 2005) glavni ciljevi procesa domenskog inženjerstva su:

- Definiranje zajedničkih artefakta i varijabilnosti linije za proizvodnju softvera;
- Definiranje njenog opsega, skupa aplikacija koje se razvijaju na zajedničkoj platformi;
- Razvoj ponovno upotrebljivih artefakata koji omogućavaju planiranu varijabilnost i njihovo kombiniranje za razvoj aplikacija.

Ukoliko se linija za proizvodnju odnosi na više poslovnih sektora, domensko inženjerstvo se osim razvoja platforme koja je zajednička za više poslovnih sektora, odnosi i na razvoj komponenata poslovnih aplikacija koje se koriste za više aplikacija nekog od poslovnih sektora. Primjerice, linija za proizvodnju poslovnih aplikacija za dva poslovna sektora, turizam i trgovinu, sastoji se od platforme koja je zajednička za oba poslovna sektora i dvije grupe poslovnih komponenata, za svaki poslovni sektor po jednu, koje se koriste za razvoj više od jedne aplikacije svakog od navedenih poslovnih sektora.

Proces implementacije osnovnih artefakata referentne arhitekture započinje proučavanjem poslovnog sektora, nastavlja se prikupljanjem zahtjeva, te odlučivanjem o implementaciji definiranih zahtjeva: vlastitom razvoju (engl. *make*), kupnji (engl. *buy*), pronalaženju rješenja

u postojećim aplikacijama (engl. *mine*) ili povjeravanju razvoja (engl. *commission*) nekoj drugoj organizaciji, ilustracija na slici 24.



Slika 24 Proces odlučivanja o razvoju komponenata (Clements & Northrop, 2002c)

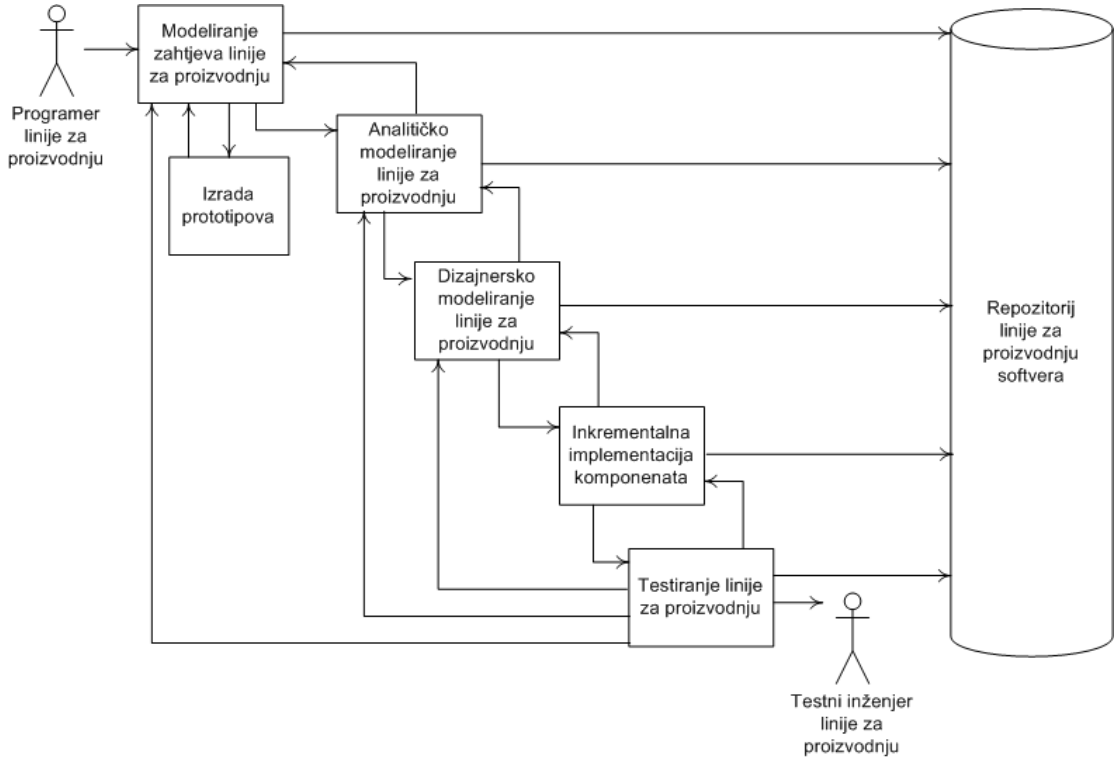
Vlastiti razvoj (engl. *make*): artefakt se razvija u vlastitoj organizaciji. Glavna prednost ove opcije odnosi se na razinu kontrole koju organizacija ima nad razvijenim artefaktom, posebno ako artefakt predstavlja inovaciju u odnosu na raspoložive slične artefakte na tržištu.

Kupnja (engl. *buy*): artefakt se kupuje na tržištu ili preuzima rješenje otvorenog programskog koda ukoliko je to isplativije od vlastitog razvoja. Primjerice, to se odnosi na operativne sustave (npr. Windows, Linux), razvojne alate (npr. Eclipse) ili neke druge komponente (npr. Spring security, Hibernate).

Traženje rješenja (engl. *mine*): artefakt se pronalazi u nekom od postojećih sustava unutar organizacije; potrebno ga je prilagoditi za ponovnu upotrebu, što ponekad zahtjeva veliki napor kojeg je prije prilagodbe potrebno procijeniti u odnosu na druge opcije.

Povjeravanje (narudžba) razvoja (engl. *commision*): razvoj artefakta se nakon specifikacije zahtjeva dodjeljuje nekoj drugoj organizaciji. Najveći rizik ove opcije odnosi ne na potencijalne razlike u razumijevanju specifikacija zahtjeva između dionika koji sudjeluju u procesu. Kako bi se ublažio taj rizik, potrebno je dobro i detaljno specificirati zahtjeve i uspostaviti pravovremenu komunikaciju sa organizacijom kojoj je dodijeljen zadatak razvoja artefakta.

Slika 25 ilustrira faze procesa domenskog inženjerstva; u fazi definiranja zahtjeva koriste se *use case* model i *feature diagram* pomoću koji se dokumentiraju *commonality* i *variability* na razini cijele linije za proizvodnju softvera; u fazi analize razvijaju se statički i dinamički modeli, najčešće pomoću *object interaction diagram* i dijagrama klasa; u fazi oblikovanja koriste se modeli koji su nastali u fazi analize (naglasak na problemima koji se rješavaju) za definiranje pod sustava i komponenata pomoću modela oblikovanja (naglasak na rješenjima problema), koriste se predlošci ili uzorci za arhitekturu koji obuhvaćaju strukturu i dinamiku arhitekture; u fazi implementacije se prema definiranim modelima iz prethodne faze razvijaju komponente u inkrementalnim fazama, prema planiranim prioritetima, u tri pod faze: detaljni dizajn, kodiranje i *unit test*; zadnja faza je faza testiranja u kojoj se izvode najmanje dva testa: funkcionalni i integracijski test. Rezultat ovih faza je definirana referentna arhitektura, komponente koju ujedno nazivamo i platforma.

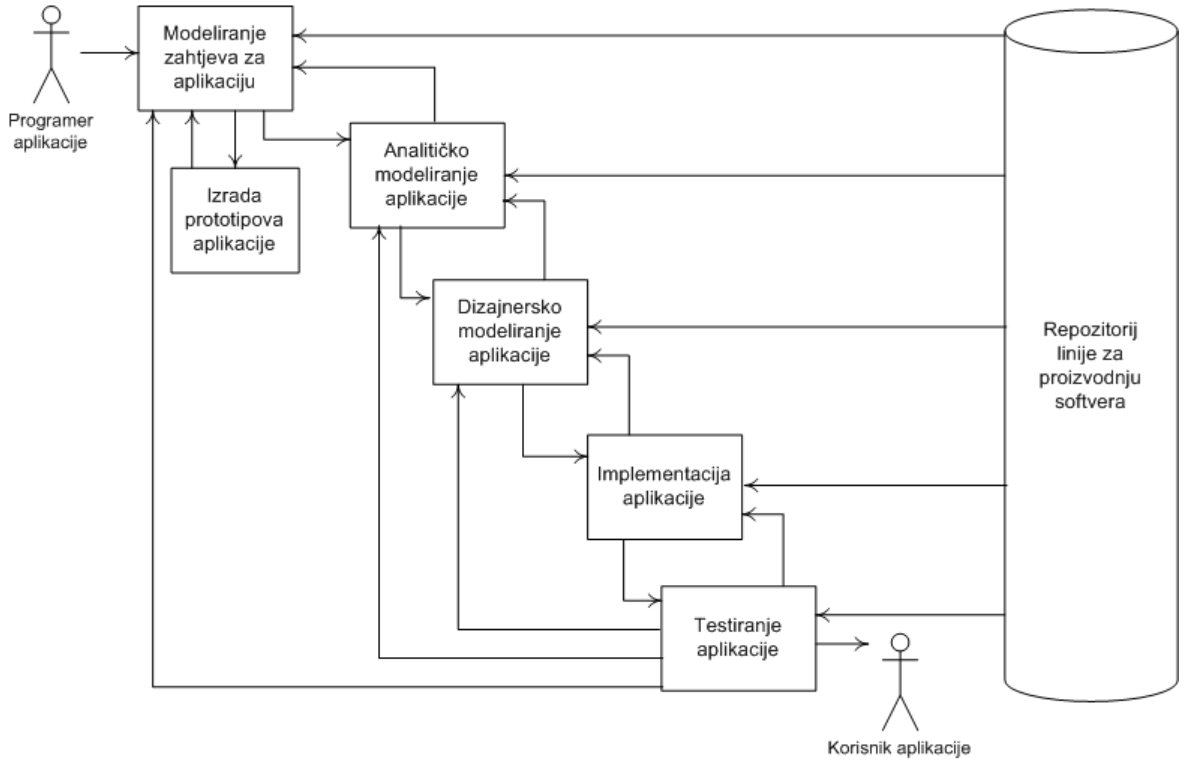


Slika 25 Proces razvoja osnovnih artefakata (Gomaa, 2005)

Platforma linije za proizvodnju softvera koristi se u više aplikacija. Greške u radu komponenata koje se nalaze u platformi direktno se odnose i na svaku od aplikacija koje ih koriste. Stoga je posebno važno osigurati potrebnu razinu kvalitete platforme izvođenjem raznih vrsta testiranja.

3.3.3.2. Aplikacijsko inženjerstvo

Aplikacijsko inženjerstvo odnosi se na proces razvoja aplikacija kombiniranjem osnovnih artefakata prethodno definiranih u procesu domenskog inženjerstva. U ovom procesu aplikacije linije za proizvodnju softvera se razvijaju korištenjem osnovnih artefakata i konfiguriranjem postojećih točaka varijabilnosti. Slika 26 ilustrira razdvajanje procesa razvoja aplikacija od procesa razvoja osnovnih artefakata koji su definirani u repozitoriju zajedničkih komponenata. Za vrijeme razvoja aplikacija odvija se povratna komunikacija sa timom u domenskom inženjerstvu u cilju redefiniranja postojećih artefakata ili kreiranja novih na temelju potreba ili iskustva za vrijeme razvoja novih aplikacija.



Slika 26 Proces razvoja aplikacija (Gomaa, 2005)

Procesa aplikacijskog inženjerstva odvija se također u fazama; u fazi definiranja zahtjeva koriste se *use case* model, odabiru se postojeće funkcionalnosti (obavezne, alternativne) koje su prethodno definirane u domenskom inženjerstvu pomoću *feature diagram* modela, realiziraju se točke varijabilnosti (engl. *variability points*); u fazi analize definiraju se statički i dinamički modeli aplikacije koji u obzir uzimaju samo komponente koje se koriste u aplikaciji; u fazi oblikovanja definira se arhitektura aplikacije na temelju referentne arhitekture linije za

proizvodnju softvera, određuju parametri za konfiguraciju varijabilnih točaka; u fazi implementacije koriste se postojeće komponente iz repozitorija komponenata i razvijaju se nove komponente ukoliko one ne postoje, u tri pod faze: detaljni dizajn, kodiranje i *unit test*; u fazi aplikacijskog testiranja izvode se dva testa za svaki *use case*: funkcionalni test (engl. *black box*) i integracijski test (engl. *white box*).

Ukoliko se linija za proizvodnju softvera odnosi na više poslovnih sektora, aplikacijsko inženjerstvo se odnosi na korištenje postojećih i na razvoj novih komponenata poslovnih aplikacija. Nove komponente su kandidati za uvrštavanje u skup ponovno upotrebivih komponenata kroz proces domenskog inženjerstva, koje se kasnije mogu koristiti u drugim aplikacijama istog poslovnih sektora. Sve komponente koje su specifične samo za jednu aplikaciju razvijaju se i održavaju u procesu aplikacijskog inženjerstva.

3.3.4. Perspektiva organizacije

Organizacija razvoja softvera podrazumijeva raspored aktivnosti, uloga i odgovornosti na konkretne ljude i na organizacijsku strukturu. Iz perspektive procesa, organizacija linija za proizvodnju softvera predstavlja strukturu i odgovornosti za osnovne artefakte i aplikacije. Organizacijska struktura linije za proizvodnju softvera određuje tko i gdje će (Clements & Northrop, 2002b):

- Razvijati i održavati referentnu arhitekturu;
- Dizajnirati, razvijati i održavati osnovne artefakte;
- Razvijati aplikacije korištenjem osnovnih artefakata;
- Definirati proces razvoja i način praćenja sukladnosti sa definiranim procesima;
- Održavati produkcijsku okolinu unutar koje se razvijaju aplikacije;
- Pratiti trendove, tehnološke promjene, i druge okolnosti koje mogu utjecati na budućnost razvoja linije za proizvodnju.

Organizacije mogu izabrati pristup „svi na istoj lokaciji“ ili pristup razdvajanja timova na više lokacija. Oba pristupa imaju prednosti i nedostatke. Pri odlučivanju o tomu da li je potrebno timove razdvajati važno je u obzir uzeti slijedeće faktore (Clements & Northrop, 2002b):

- Broj proizvoda. Naime broj kanala komunikacije se uvelike povećava ukoliko se aktivnost razvoja osnovnih artefakata izvodi u svakom timu koji je odgovoran za

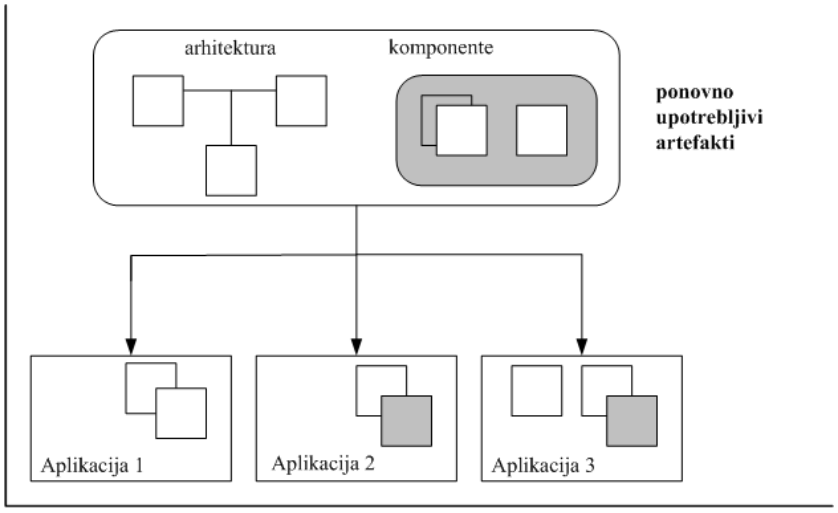
specifičnu aplikaciju. S druge strane, organiziranje jednog tima koji je odgovoran za osnovnu infrastrukturu koju koriste sve aplikacije, smanjuje broj kanala komunikacije koja se odvija između timova za razvoj aplikacija i jednog centralnog tima za razvoj osnovnih artefakata.

- Novi razvoj ili uporaba postojećih aplikacija. U uvjetima gdje se osnovni artefakti razvijaju iz postojećih komponenata, postoje opravdani razlozi da isti timovi koji rade na održavanju ili razvoju aplikacija, sudjeluju i u izradi osnovnih artefakata.
- Način financiranja timova za izradu osnovnih artefakata.
- Potreban napor kod iskorištavanja ponovno uporabljivih komponenata; koliko je potrebno da se iz postojećih komponenata napravi nova aplikacija. Ukoliko taj napor nije velik, tada je opravdano odvojiti tim za razvoj komponenata, u suprotnom integrirani timovi su bolje rješenje.
- Učestala promjenjivost osnovnih artefakata povećava potrebu da posebni tim upravlja sa promjenama, umjesto da to rade timovi za razvoj aplikacija.
- Paralelni ili sekvencijalni razvoj aplikacija. Ukoliko se aplikacije razvijaju sekvencijalno tada integrirani timovi imaju više smisla nego u slučaju da se aplikacije razvijaju paralelno, u tom slučaju postoji jača potreba za odvajanje tima za razvoj osnovnih artefakata.

Glavna pažnja kod primjene pristupa linija za proizvodnju softvera poklanja se tehničkim i procesnim područjima, dok se za organizacijski model podrazumijeva da se sastoji od jedne organizacijske jedinice koja je odgovorna za razvoj i održavanje arhitekture i osnovnih artefakata, te nekoliko aplikativnih organizacijskih jedinica koje razvijaju aplikacije korištenjem definirane arhitekture i osnovnih artefakata. Analizom nekoliko organizacija koje koriste ovaj pristup (Bosch, 2001), prepoznata su četiri organizacijska modela:

Razvojni odjel (engl. *development department*): organizacijski model sastoji se samo od jedne organizacijske jedinice. Svaki član organizacijske jedinice može raditi na razvoju osnovnih artefakata i na razvoju aplikacija koje koriste osnovne artefakte. Razvoj se organizira po projektima koji se mogu svrstati u kategorije: infrastrukturni projekti, koji imaju za zadatak razvoja osnovnih artefakata infrastrukture ili njihovih novih inačica, te aplikativni projekti, koji imaju za zadatak razvoj aplikacija ili njihovih novih inačica korištenjem osnovnih artefakata

infrastrukture. Ovaj organizacijski model je u primjeni u organizaciji u kojoj provodimo istraživanje kroz studiju slučaja (poglavlje 6), stoga ga posebno prikazujemo na slici 27.



Slika 27 Razvojni odjel (Bosch, 2001)

Prema istraživanju (Bosch, 2001) model je primjenjiv na organizacije koje imaju manje od 30 osoba koje rade na poslovima usko povezanima uz softver. Ukoliko organizacija ima više od 30 osoba direktno povezanih uz razvoj softvera, općenito je potrebno napraviti reorganizaciju, neovisno o primjeni linija za proizvodnju softvera.

Poslovna jedinica (engl. *business unit*): organizacijski model u kojem je svaka organizacijska jedinica odgovorna za jedan podskup sličnih aplikacija. Ponovno upotrebljive komponente i osnovni artefakti infrastrukture se razvijaju u bilo kojem dijelu organizacijske jedinice i stavljaju se na raspolaganje ostalim organizacijskim jedinicama. Svaka organizacijska jedinica slobodno mijenja ili proširuje osnovne artefakte infrastrukture, testira ih, dostavlja ih ostalim organizacijskim jedinicama za njihovo korištenje. Ovaj model se primjenjuje u organizacijama koje imaju od 30 do 100 osoba direktno povezanih uz razvoj softvera.

Odjel za razvoj infrastrukture (engl. *domain engineering unit*): organizacijski model koji dijeli organizaciju na organizacijsku jedinicu koja se isključivo bavi razvojem i održavanjem osnovnih artefakata infrastrukture (*domain engineering unit*) i na druge organizacijske jedinice koje se bave razvojem aplikacija (*product engineering unit*). Ovaj organizacijski model je prikladan za velike organizacije, zahtjeva intenzivnu komunikaciju između organizacijskih dijelova za razvoj aplikacija koji su u čestoj komunikacijskoj vezi sa klijentima, i od

organizacijskog dijela za razvoj osnovnih artefakata koji nema komunikacijsku vezu sa klijentima ali mu je potrebno razumijevanje korisničkih zahtjeva. Ovaj model je prikladan za primjenu u organizacijama od preko 100 osoba direktno povezanih uz razvoj softvera.

Hijerarhijski odjel za razvoj infrastrukture (engl. *hierarchical domain engineering unit*): osim veličine, izražene u broju osoba direktno povezanih uz razvoj softvera, često su i tehnički razlozi povod za uvođenje organizacijskog modela koji uključuje još jednu organizacijsku razinu. To se može odnositi na implementaciju jedne ili više specijaliziranih linija za proizvodnju softvera, te ovisno o veličini i kompleksnosti dodatna linija se može organizirati kao *poslovna jedinica* ili kao *odjel za razvoj infrastrukture*. To u biti znači da ovaj model može u sebi uključivati i druge modele organizacije. Ovaj oblik organizacije odgovara samo velikim organizacijama koje imaju nekoliko linija za proizvodnju softvera. Primjena ovog modela zahtjeva visoke troškove. Model je primjenjiv samo u velikim organizacijama koje imaju proizvode sa dugačkim vijekom trajanja; kompleksnost uvođenja ovog modela zahtjeva veću razinu zrelosti organizacije u području upravljanja projektima za razvoj softvera.

3.4. Metode razvoja

Linije za proizvodnju softvera predstavljaju specifičan pristup za koji je bilo potrebno razviti specifične metode i alate kako bi se lakše rješavali problemi koji proizlaze iz prakse uvođenja i održavanja ovog pristupa u organizacijama. Od velikog broja predloženih metoda, navodimo samo neke koji se najčešće koriste u praksi.

Product Line Software Engineering (PuLSE) (DeBaud, Flege, & Knauber, 1998): sveobuhvatna metoda koja podržava konceptualizaciju, razvoj, korištenje i evoluciju linija za proizvodnju softvera.

Feature-Oriented Domain Analysis (FODA) (Kang i ostali, 1990): metoda za pronalaženje i prikaz funkcionalnosti ili svojstava koji su zajednički za više aplikacija, u fazi specifikacije zahtjeva, s ciljem njihove ponovne upotrebe.

Feature-Oriented Reuse Method (FORM) (Kang i ostali, 1998): proširena FODA metoda za pronalaženje i prikaz funkcionalnosti ili svojstava koji su zajednički za više aplikacija, u fazi dizajna; propisuje način razvoja osnovnih artefakata arhitekture i komponenata za ponovnu upotrebu.

Component-Oriented Platform Architecting (COPA): metoda za razvoj arhitekture linija za proizvodnju elektroničkih proizvoda, koju je razvio Filips Research Lab za internu upotrebu.

Family-Oriented Abstraction, Specification, and Translation (FAST) (Weiss & others, 1999): proces koji se koristi kada organizacija razvija veći broj inačica softverskih aplikacija koje dijele zajedničke artefakte.

Komponentenbasierte Anwendungsentwicklung (KobrA) "*component-based application development*" (Atkinson, 2002): definira kompletan proces koji uključuje aktivnosti i artefakte, podržava Model Driven Architecture (MDA) pristup.

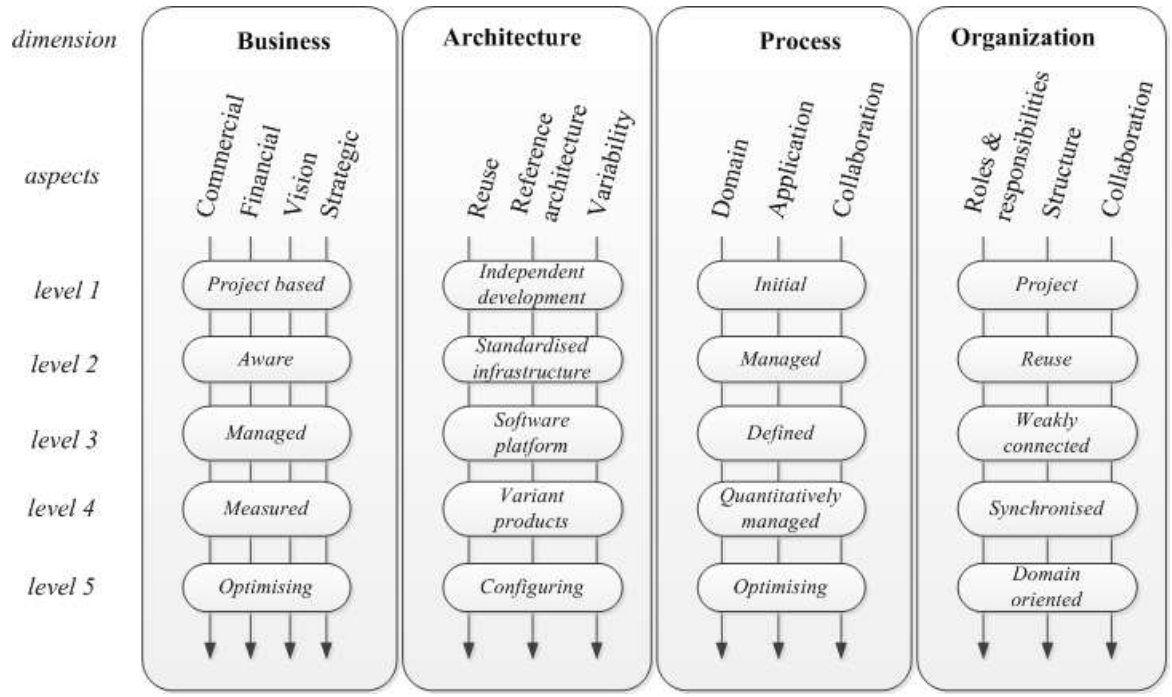
Quality-driven Architecture Design (QADA) (Purhonen, Niemelä, & Matinlassi, 2004): metoda za dizajn i provjeru arhitekture uslužno orijentiranih (engl. *service-oriented*) sustava.

3.5. Modeli zrelosti

Organizacije provode procjenu zrelosti linija za proizvodnju softvera kako bi se utvrdila učinkovitost u primjeni ovog pristupa u praksi, za usporedbu sa drugim sličnim organizacijama, te kao podloga za donošenje odluka o uvođenju ili poboljšavanju primjene ovog pristupa u organizaciji. Rezultati procjene zrelosti mogu upućivati na to da je linija za proizvodnju softvera na željenoj razini zrelosti, ili da ona to nije; u svakom slučaju organizacije mogu rezultate procjene zrelosti koristiti za predlaganje inicijativa za poboljšanje ili za usklađivanje procesa, organizacije ili arhitekture sa strateškim poslovnim ciljevima organizacije.

Metoda koja se najčešće koristi u praksi za provjeru zrelosti linije za proizvodnju softvera naziva se *Family Evaluation Framework* (FEF) (F. Van Der Linden i ostali, 2004). Ova metoda se temelji na BAPO dimenzijama koje obuhvaćaju četiri područja: poslovanje, arhitekturu, procese i organizaciju. FEF nije prva metoda za procjenu zrelosti linija za proizvodnju softvera, ona u sebi uključuje većinu karakteristika metoda koje su joj prethodile: *Product Line Technical Probe* (PLTP) (Clements & Northrop, 2002b), kojom su obuhvaćena područja razvoja aplikacija, razvoja osnovnih artefakata te područje upravljanja, dok je njezin glavni cilj procjena spremnosti neke organizacije za uvođenje i uspješnu primjenu linije za proizvodnju softvera; *Capability Maturity Model Integrated* (CMMI) metoda za ispitivanje zrelosti organizacije u području softverskog inženjerstva; *Maturity and Evolution in Software Product Lines* (Bosch, 2002); *Quantitative Model of the Value of Architecture in Product Line Adoption* (Schmid, 2004); *Product Line Potential analysis in Software Product Lines* (Fritsch & Hahn, 2004).

Slika 28 prikazuje FEF dimenzije i područja metode koja uključuje procjenu zrelosti za svaku od četiri BAPO dimenzije, neovisno jednu od druge. Svaka dimenzija obuhvaća niz aspekata, i nekoliko razina zrelosti.



Slika 28 Family Evaluation Framework (FEF)

Razine zrelosti od 1 od 5 pokazuju stupanj na kojem se organizacija nalazi u svakom od četiri područja. Viši stupanj zrelosti podrazumijeva da organizacija zadovoljava sve uvjete iz nižih stupnjeva.

U ovom radu, u poglavlju validacije artefakata (6.6) koristimo područje arhitekture za procjenu zrelosti predložene referentne arhitekture za poslovne aplikacije. Procjena zrelosti arhitekture uglavnom se odnosi na procjenu zrelosti odnosa referentne arhitekture i arhitekture aplikacija. Razina zrelosti se procjenjuje u odnosu na: razinu ponovne upotrebe osnovnih artefakata i komponenata (engl. *asset reuse level*), razinu utjecaja referentne arhitekture na arhitekturu aplikacija (engl. *reference architecture*), te na način implementacije varijabilnosti u referentnoj arhitekturi (engl. *variability management*).

3.6. Studije slučajeva uvođenja

Premda je pristup linija za proizvodnju softvera teoretski potvrđen, mnoge organizacije su ga uspješno uvele i koriste ga u praksi, to još uvijek ne garantira njegovo uspješno uvođenje i primjenu u svakoj organizaciji. Svaka organizacija ima svoje specifičnosti, stoga je potrebno analizirati postojeća iskustva drugih sličnih organizacija prije donošenja odluke o načinu i opsegu primjene ovog pristupa u vlastitoj organizaciji. U ovom poglavlju predstaviti ćemo nekoliko slučajeva uvođenja i primjene ovog pristupa u praksi različitih vrsta organizacija. Primjerice, *Philips Medical Systems* uspostavio je tim od preko 1000 razvojnih programera, razvio osnovne artefakte iz postojećih aplikacija, smanjio ukupno potreban napor od 25-50%, ubrzao isporuku softvera; *Siemens* je djelomično uveo ovaj pristup što je omogućilo povećanje ponovne upotrebe komponenata za 75% (F. J. Linden i ostali, 2010). Studije slučaja uvođenja i primjene pristupa linija za proizvodnju softvera u različite vrste organizacija ukazuju na to da se pristup može implementirati u razne vrste organizacija i poslovnih sektora, te da može sa velikom vjerojatnošću značajno poboljšati razvoj i održavanje softvera općenito.

Kao dio istraživanja u okviru disertacije analizirane su studije slučaja objavljene u relevantnoj literaturi. Istraživanje je obuhvatilo 3 studije slučaja. Određeni su zajednički atributi. Rezultati su prikazani u tablicama koje slijede.

AKVAsmart[6]	
Veličina:	6-10 razvojnih programera.
Način uvođenja:	Zamjena postojećeg sustava.
Poboljšanja:	- Smanjenje broja linija izvornog koda za 70%. - Jedinstveni izgled aplikacija. - Zajednička platforma i isti izgled izvornog koda. - Lakša ponovna upotreba, održavanje i integracija.
Poslovanje:	Pristup se koristi za planiranje novih proizvoda.
Arhitektura:	Razvijen je okvir referentne arhitekture i pomoćni alat (engl. *plug-in*).
Proces:	Proces razvoja je definiran na fleksibilan način, koriste se elementi RUP-a, SCRUM-a i XP modela.
Organizacija:	Jedna organizacijska jedinica radi na svim zadacima. Najviše napora se odnosi na razvoj osnovnih artefakata koji su namijenjeni za korištenje u nekim od specifičnih proizvoda.

[6] Studija slučaja autora Magne Johnson, Magne Syrnstad (F. J. Linden, Schmid, & Rommes, 2010)

Nokia Mobile Phones: primjenjuje ovaj pristup kako bi poboljšala kvalitetu i smanjila troškove, za upravljanje kompleksnošću sustava.

Nokia Mobile Phones[7]	
Veličina:	Preko 1000 razvojnih programera.
Način uvođenja:	Strateška odluka koja se temelji na primjeni postojećih artefakata.
Poboljšanja:	- Bolje razumijevanje evolucije softvera. - Bolji uvid u mogućnosti ponovne upotrebe.
Poslovanje:	Razvoj arhitekture utječe na poslovne potrebe.
Arhitektura:	Modeli arhitekture olakšavaju njenu evoluciju.
Proces:	Specifičan pristup razvoju i dokumentiranju arhitekture.
Organizacija:	Zahtjevi različitih dionika u procesu se uzimaju u obzir.

Philips Consumer Electronics: za razvoj televizijskih sustava, u svrhu upravljanja sve zahtjevnijim softverom. Uvođenje pristupa je zahtijevalo značajne organizacijske promjene.

Philips Consumer Electronics[8]	
Veličina:	250 razvojnih programera.
Način uvođenja:	Nova arhitektura, reverzni inženjering postojećeg koda.
Poboljšanja:	- Samo jedna linija za proizvodnju softvera za sve srednje i velike *Philips* televizije. - Omogućava efikasnu konfiguraciju od tržišta zahtijevanih varijabilnih karakteristika. - Razvoj softvera više nije „usko grlo“ kod razvoja novih proizvoda. - Nakon 6 godina arhitekturu nije bilo potrebno mijenjati. Starije arhitekture su trajale najviše 5 godina.
Poslovanje:	Namjera je omogućiti podršku za od tržišta zahtijevanu varijabilnost, u isto vrijeme održavati kvalitetu proizvoda i u budućnosti omogućiti kombinaciju više proizvoda u jednom.
Arhitektura:	Arhitektura koristi pristup kompozicije, *Koala* komponentni model.
Proces:	Promjena od projektne organizacije u produktnu organizaciju.
Organizacija:	Organizacija je od produktno orijentirane preoblikovana u jedinstvenu organizaciju sa dva tima, jedan za razvoj osnovnih komponenata i drugi za razvoj proizvoda (produkata).

[7] Studija slučaja autora Claudio Riva, Jinali Xu (F. J. Linden i ostali, 2010)
[8] Studija slučaja autora Rob van Ommering (F. J. Linden i ostali, 2010)

4. NACRT ISTRAŽIVANJA

U ovom poglavlju opisujemo proces i metodologiju istraživanja, opis korištenih metoda za istraživanje, instrumenata istraživanja, način prikupljanja podataka, plan istraživanja, strategiju validacije referentne arhitekture te navodimo objavljene radove autora iz područja istraživanja.

Nacrt istraživanja obuhvaća slijedeće ciljeve ovog istraživanja: (C1.4) ispitati mišljenje iskusnih arhitekata i istraživača o relevantnosti definiranih funkcionalnosti referentne arhitekture, (C3.1) provjeriti upotrebljivost razvijenih artefakata referentne arhitekture, (C3.2) primijeniti i koristiti razvijene artefakte referentne arhitekture u praksi putem longitudinalne studije slučaja s ciljem provjere njihove primjenjivosti, (C4) provjeriti povezanost rezultata PR i MI modela za mjerenje održavljivosti aplikacijskih komponenata linije za proizvodnju softvera.

4.1. Proces istraživanja

Odgovori na postavljenja istraživačka pitanja (poglavlje 1) uglavnom se odnose na implementaciju i provjeru valjanosti referentne arhitekture za poslovne aplikacije. Implementacija se temelji na definiranim funkcionalnim zahtjevima koji su rezultat istraživanja postojećeg teorijskog okvira, analize postojećih referentnih arhitektura, te praktičnog iskustva autora u razvoju poslovnih aplikacija. Potvrdu relevantnosti inicijalno predloženih funkcionalnih zahtjeva referentne arhitekture dobili smo ispitivanjem stručnjaka koji imaju iskustvo u praksi i istraživanju arhitekture. Provjera valjanosti referentne arhitekture uključuje provjeru primjenjivosti u praksi putem studije slučaja (engl. *case study*) i provjeru upotrebljivosti metodom kontroliranog eksperimenta (engl. *controll experiment*).

Osim navedenih metoda, za provjeru povezanosti rezultata dvaju modela (MI i PR) za mjerenje održavljivosti softverskih komponenata, koristimo i statističke metode korelacije i multiple linearne regresije pomoću kojih u odnos stavljamo više kvantitativnih nezavisnih varijabli, metrika izvornog programskog koda, i dvije zavisne varijable, za svaki od dvaju modela po jednu.

4.2. Metode istraživanja

Traženje prikladne krovne znanstvene metode za ovo istraživanje nije bilo trivijalno, budući da glavni ciljevi ovog istraživanja nisu formulirani na tradicionalan način koji se najčešće koristi u istraživanju informacijskih sustava. Za razliku od uobičajenih istraživanja čije se glavno težište odnosi na razvoj teorije i njenu provjeru, ovo istraživanje osim teorijskog karaktera uključuje implementaciju, provjeru i primjenu artefakata u praksi. Na prvi pogled, rad na referentnoj arhitekturi za poslovne aplikacije može izgledati kao istraživanje teorijskog karaktera. Međutim, iako je istraživanje referentne arhitekture poslovnih aplikacija svakako i teorijskog karaktera, to ne znači da je i znanstveni doprinos ovog istraživanja isključivo i samo teorijskog karaktera. Istraživanja čiji je znanstveni doprinos isključivo teorijskog karaktera imaju glavno težište na razumijevanju određenog predmeta istraživanja i na traženju odgovora na pitanje "zašto" određenog fenomena koji je predmet tog istraživanja (Whetten, 1989) .

Motiv za ovo istraživanje nije razumijevanje samog fenomena referentne arhitekture, već je to i traženje rješenja koje će dati odgovor na postavljena istraživačka pitanja u cilju pronalaženja kvalitetnog načina organizacije i prikladnih vrsta odnosa između ključnih dijelova referentne arhitekture koji bi omogućio optimalni razvoj poslovnih aplikacija. Iz navedenog razmatranja može slijediti i pitanje o karakteru ovog istraživanja; može li se traženje rješenja kvalitetne referentne arhitekture promatrati kao standardni postupak implementacije programskih rješenja. Pitanje ima smisla, najviše stoga jer istraživanje u ovoj disertaciji nema uobičajeni teorijski doprinos. Možemo li prema tome tvrditi da pristup kojeg koristimo pri traženju rješenja u području referentne arhitekture za poslovne aplikacije jest znanstveni pristup? U pronalaženju odgovora na postavljeno pitanje o tome da li se ovo istraživanje može smatrati znanstvenim istraživanjem, potrebno je pronaći i prikladnu krovnu metodu koju možemo primijeniti u ovoj disertaciji. Činjenica je da postoji znanstvena metoda pod nazivom „znanost o oblikovanju*"* (engl. *design science*) koja se koristi u području informacijskih sustava (March & Smith, 1995) i koja se može koristiti pri razvoju referentne arhitekture za poslovne aplikacije. Suštinu ove metode najbolje su opisali (Fuller & Kuromiya, 1992): "*Funkcija metode znanosti oblikovanja je rješavanje problema uz pomoć novog artefakta čija će raspoloživost utjecati na spontano odbacivanje postojećih artefakata za rješavanje istovjetnih problema. Primjerice, kada ljudi imaju potrebu prelaziti sa jedne strane rijeke na drugu, kao znanstvenik koji koristi ovu metodu, dizajnirao bih most koji bi sasvim sigurno spontano i zauvijek odbacio prelaženje rijeke plivanjem sa jedne strane na drugu*".

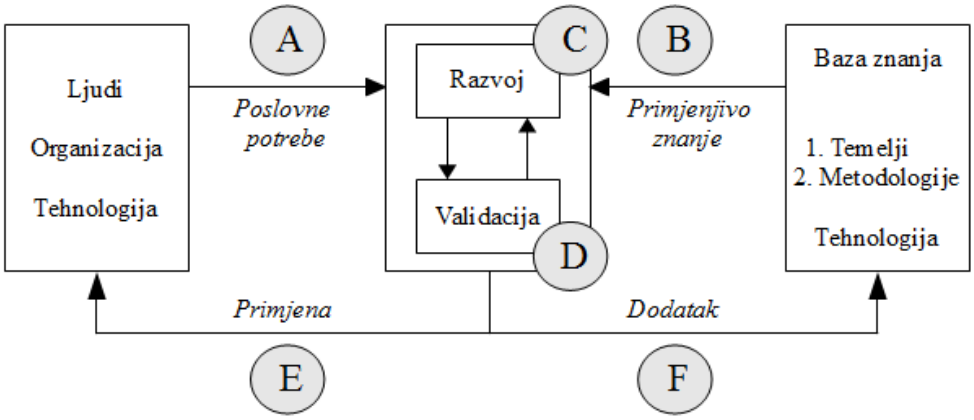
Primijenjeno na ovu disertaciju, primjena metode *znanost o oblikovanju* se odnosi na oblikovanje referentne arhitekture za poslovne aplikacije pristupom linija za proizvodnju softvera. Njena svrha je olakšavanje razvoja i održavanja poslovnih aplikacija na novi i drugačiji način, odbacujući postojeće pristupe skupljeg i manje produktivnog razvoja i održavanja. Ova tvrdnja ima i uporište u istraživanju (Chen & Ho, 2002) kojim se potvrđuje primjena *znanosti o oblikovanja* u istraživanju informacijskih sustava kao primjerene metode kojom se znanje primjenjuje u rješavanju praktičnih problema.

Osim standardnih znanstvenih metoda koje koristimo u ovoj disertaciji, kao ključne metode pored *znanosti o oblikovanju*, koristimo metodu studije slučaja za provjeru primjenjivosti referentne arhitekture u jednoj financijskoj instituciji, kontrolirani eksperiment za provjeru upotrebljivosti referentne arhitekture, statističke metode korelacije i linearne regresije za provjeru povezanosti dvaju modela (MI i PR) za mjerenje održavljivosti aplikacijskih poslovnih komponenta, te metodu ispitivanja putem anketnog upitnika za potvrđivanje relevantnosti definiranih funkcionalnih zahtjeva referentne arhitekture.

4.2.1. Znanost o oblikovanju (engl. *design science*)

Znanost o oblikovanju prvi puta se pojavljuje 1963. godine, kao sistematski način oblikovanja (Fuller & McHale, 1967). Na popularnost ove metode najviše je utjecala knjiga pod nazivom *The Sciences of the Artificial* (Simon, 1969), koja se smatra glavnim pokretačem razvoja više sustavnih i formalnih metoda oblikovanja koje se odnose na razna područja kao što su arhitektura, prostorno planiranje, medicina i računalne znanosti. Istraživanja koja koriste metodu *znanost o oblikovanju* zovu se i istraživanja za unaprijeđenije (engl. *improvement research*), dok se sama bit ovog pristupa odnosi na način rješavanja problema i poboljšavanja karakteristika artefakata koji su predmet istraživanja. *Znanost o oblikovanju* ima za cilj stvaranje artefakata koji ljudima služe za određenu svrhu, za razliku od prirodnih i društvenih znanosti, koje pokušavaju razumjeti stvarnost (Au, 2001). Jedna od najčešće citiranih definicija istraživanja koja se temelje na ovoj metodi glasi: „*Istraživanja prema metodi znanosti o oblikovanju su istraživanja u kojima istraživač (oblikovatelj) odgovara na pitanja koja se odnose na ljudske probleme na način da kreira artefakte, te na taj način doprinosi znanosti*“ (Hevner & Chatterjee, 2010). Slika 29 prikazuje postupak istraživanja prema ovoj metodi koji započinje definiranjem konkretne poslovne potrebe za koju ne postoji rješenje sa potrebnim karakteristikama, niti u praksi niti u postojećim istraživanjima. Na temelju postojećih znanja, prakse i tehnologije razvija se novi artefakt ili poboljšavaju karakteristike postojećeg artefakta.

Razvoj i validacija artefakta odvija se u ciklusima sve dok se ne postignu definirane potrebne karakteristike. Razvijeni ili poboljšani artefakti i novo stečena znanja koriste se dvostruko, kao dodatak postojećem znanju te u obliku primjene u praksi za rješavanje poslovnih problema. U području informacijskih znanosti, glavni utjecaj na korištenje ove metode u znanstvenim istraživanjima imao je znanstveni rad *Design science in Information Systems research* (Hevner, March, Park, & Ram, 2004b).



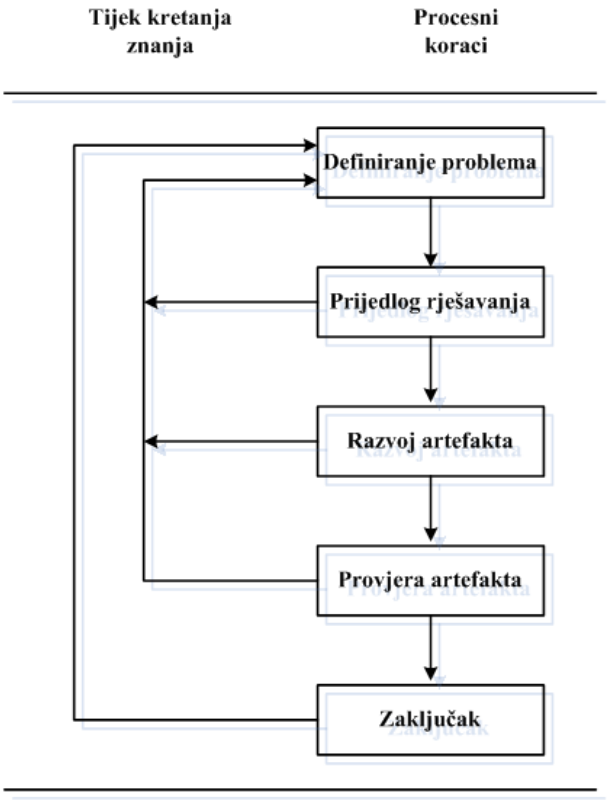
Slika 29 Postupak istraživanja prema metodi *znanost o oblikovanju*[9]

Većinu istraživanja u području informacijskih sustava karakteriziraju dvije glavne paradigme: (engl. *behavioral science*) i (engl. *design science*). Istraživanja koja se odnose na razvoj i provjeru teorije pomoću koje se objašnjava ili predviđa ponašanje ljudi ili organizacija svrstavamo u eksplanacijska istraživanja. Konfirmacijska istraživanja, s druge strane, mogu koristiti metodu *znanost o oblikovanju* koja proširuje navedeni kontekst eksplanacijskih istraživanja na način da se osim ljudi i organizacije u kontekst istraživanja dodaje i izrada novih ili poboljšanih postojećih artefakata kao što su: koncepti, modeli, metode i primjerci raznih artefakata koji se mogu razvijati (Hevner i ostali, 2004b). Svrha razvoja koncepta, modela, metode i primjerka je njihovo korištenje za rješavanje specifičnih problema, kao što je u našem istraživanju poboljšani razvoj poslovnih aplikacija. Koncepte koristimo da bi proširili ili redefinirali standardne pojmove u nekom području, te da pomoću njih možemo jasnije opisati probleme u tom području. Model se koristi za prikaz ili izražavanje odnosa među konceptima, pa tako u našem istraživanju modele koristimo u oblikovanju odnosa između glavnih dijelova referentne arhitekture. Metoda je niz postupaka koje koristimo u cilju obavljanja specifičnog

[9] (Hevner, March, Park, & Ram, 2004b)

zadatka. Osnova za provođenje metode su koncepti i modeli koji su prethodno nastali. U našem istraživanju koristimo nekoliko standardnih metoda za razvoj, validaciju i dokumentiranje referentne arhitekture. Primjerak, kao vrsta artefakta, odnosi se na konkretnu realizaciju nekog prethodno definiranog koncepta i modela korištenjem prikladne metode za njegov razvoj. Referentna arhitektura za poslovne aplikacije koja je predmet ovog istraživanja spada u ovu vrstu artefakta.

Istraživanja u kojima se koristi metoda *znanost o oblikovanju* imaju za svrhu bolje razumijevanje, objašnjenje i poboljšanje informacijskih sustava. Predmeti takvih istraživanja su artefakti poput algoritama, korisničkih sučelja, programskih jezika i metoda oblikovanja informacijskih sustava. Takva istraživanja sastoje se od dvije glavne aktivnosti, razvoja i validacije artefakata. Aktivnost razvoja odnosi se na razvoj koncepta, modela, metode i primjerka čija svrha je pružanje dokaza da se artefakt može razviti. Validacija se odnosi se na definiranje kriterija ili prihvaćanje standardnih pravila za validaciju artefakta, u našem slučaju referentne arhitekture za poslovne aplikacije, te provjeru zadovoljavanja postavljenih kriterija ili standardnih pravila. Paralelno sa razvojem i validacijom, promatramo *kako* i *zašto* artefakt radi u okruženju u kojem se koristi.



Slika 30 Opći okvir ciklusa oblikovanja (V. K. Vaishnavi & Kuechler Jr, 2007, str. 59)

Slika 30 prikazuje postupak kojeg koristimo u ovom istraživanju koji se odvija u ponavljajućim ciklusima, prilagođen je istraživanju prema metodi *znanost o oblikovanju*, zove se opći okvir ciklusa oblikovanja (engl. *general design cycle framework).* Postupak se odvija slijedom od definiranja problema, prijedloga rješavanja, razvoja artefakata, provjere artefakata i zaključka, ponavlja se sve dok se ne postignu postavljeni istraživački ciljevi. Za svaki procesni korak koristimo jedan ili više uzoraka (engl. *pattern*) koji se odnose na metodu *znanost o oblikovanju* (V. K. Vaishnavi & Kuechler Jr, 2007). Uzorci u ovom kontekstu predstavljaju opis detaljne procedure koju je potrebno slijediti za vrijeme rada na nekom od procesnih koraka u vrijeme istraživanja prema metodi *znanost o oblikovanju.* Tablica 3 sadrži uzorke metode *znanost o oblikovanju* koje koristimo u ovom istraživanju.

- *Problem Area Identification* je metoda za pronalaženje općenitih istraživačkih problema koji su od interesa za istraživača ili istraživačku zajednicu u nekom području.
- *Problem Formulation* je metoda za identifikaciju specifičnih istraživačkih problema, te definiranja istraživačkih pitanja i ciljeva istraživanja.

Tablica 3 Uzorci *znanosti o oblikovanju* koje koristimo u ovom istraživanju

Procesni korak	**Uzorak znanosti oblikovanja**
Definiranje problema	*Problem Area Identification* *Problem Formulation* *Leveraging Expertise* *Industry and Practice Awareness* *Solution Scope Mismatch* *Being Visionary*
Prijedlog rješavanja	*Empirical Refinement* *General Solution Principle*
Razvoj artefakta	*Abstracting Concept* *Hierarchical design* *Elegant Desing*
Provjera artefakta	*Experimentation* *Using Metrics*
Zaključak	*Novelty and Significance* *Use of Examples*

- *Leveraging Expertise* se koristi za izbor istraživanja za vrijeme kojeg će se moći iskoristiti stručnost istraživača u području istraživanja.
- *Industry and Practice Awareness* je postupak kojim se aktivno prati stanje u industriji i praksi u području istraživanja, primjerice u korištenju programskih jezika, baza podataka, operativnih sustava, i slično.
- *Solution Scope Mismatch* je metoda za analizu eventualne primjene postojećih rješenja za postavljeno istraživačko pitanje u situacijama povećanja opsega ili povećanja kompleksnosti istraživačkog problema.
- *Being Visionary* je metoda za planiranje poboljšanja postojećih rješenja za istraživački problem.

- *Empirical Refinement* je metoda za razvoj artefakata kao rješenja za istraživački problem kroz nekoliko iteracija, razvoj, empirijsko promatranje, prepravljanje.
- *General Solution Principle* konstruiranje generičkog rješenja za neku vrstu problema.
- *Abstracting Concept* kreiranje generičkog rješenja iz postojećih parcijalnih rješenja pomoću metode apstrakcije.
- *Hierarchical design* metoda za oblikovanje kompleksnog sustava na način da se sustav podijeli na manje dijelove radi lakše kontrole.
- *Elegant Desing* oblikovanje artefakta na općenit način, da ne ovisi o okruženju u kojem se koristi; artefakt će funkcionirati čak i kada se okruženje promjeni.
- *Experimentation* upotreba eksperimenta za validaciju postavljenih hipoteza koje su povezane sa artefaktima koji se odnose na rješenja istraživačkih pitanja.
- *Using Metrics* korištenje standardnih metrika kao pomagala kod validacije predloženog rješenja za postavljena istraživačka pitanja.
- *Novelty and Significance* u zaključku se naglašava značajnost i novost istraživanja.
- *Use of Examples* za bolje razumijevanje korisnosti istraživanja koriste se konkretni primjeri primjene artefakata u praksi.

Zaključno, koncepti, modeli, metode i primjerci razvijaju se kako bi se kasnije koristili za rješavanja specifičnih problema. Razvijeni artefakti postaju predmeti istraživanja čiju valjanost je potrebno znanstveno provjeriti kako bi se utvrdilo da li je postignuto bilo kakvo poboljšanje.

4.3. Plan istraživanja

Plan istraživanja objašnjava način na koji se metode istraživanja (studija slučaja, eksperiment, i ispitivanje), tehnike prikupljanja podataka (upitnici i repozitorij izvornog programskog koda), i tehnike obrade i analize podataka (statistička analiza) koriste u ovom radu.

Tablica 4 prikazuje detaljan plan istraživanja i način na koji je provedeno ovo empirijsko istraživanje.

Tablica 4 Plan istraživanja

<table>
<tr><th>Tijek kretanja znanja</th><th>Procesni koraci</th><th colspan="3">Znanstveni doprinosi</th><th>Poglavlje</th></tr>
<tr><td rowspan="7"></td><td>Definiranje problema</td><td colspan="3">Pregled literature</td><td>Poglavlja: 2,3,5</td></tr>
<tr><td>Prijedlog rješavanja</td><td colspan="3">Relevantni funkcionalni zahtjevi</td><td>Poglavlje 5</td></tr>
<tr><td>Razvoj artefakata</td><td colspan="3">1. Opis referentne arhitekture
2. Okvir referentne arhitekture
3. Referentna aplikacija
4. Pomoćni alat za razvoj</td><td>Poglavlje 5</td></tr>
<tr><td rowspan="4">Provjera artefakata</td><td>Metoda</td><td>Prikupljanje podataka</td><td>Analiza podataka</td><td rowspan="4">Poglavlja: 5,6</td></tr>
<tr><td>Studija slučaja</td><td>Baza podataka korisničkih transakcija i zahtjeva za izmjenama, repozitorij izvornog koda (4 inačice).</td><td>Statistička analiza, korelacijska analiza, linearna regresija</td></tr>
<tr><td>Eksperiment</td><td>Upitnik</td><td>Komparativna analiza</td></tr>
<tr><td>Ispitivanje</td><td>Upitnik</td><td>Komparativna analiza</td></tr>
<tr><td>Zaključak</td><td colspan="3"></td><td>Poglavlje 7</td></tr>
</table>

4.3.1. Definiranje problema

Definiranje problema istraživanja je obavljeno u okviru prethodnih istraživanja i pripreme prijave ove doktorske disertacije. Međutim, sukladno postupku istraživanja koji se koristi, kod definiranja problema istraživanja ponavljati će se pretraživanje literature iz područja referentnih arhitektura poslovnih aplikacija s ciljem jasnog razumijevanja potrebnih funkcionalnih zahtjeva referentne arhitekture. Pretraživanje literature će se rukovoditi slijedećim istraživačkim pitanjima:

- Koje referentne arhitekture za poslovne aplikacije postoje (P1 i P2)?
- Koji su ključni funkcionalni zahtjevi referentne arhitekture za poslovne aplikacije (P1)?
- Koje su to prikladne metode za istraživanje korisnosti (engl. *usefulness*) referentne arhitekture poslovnih aplikacija u praksi (P3)?
- Koje metrike za mjerenje kvalitete referentne arhitekture postoje, te koje od njih se mogu koristiti za mjerenje održavljivosti linija za proizvodnju softvera (P4)?

4.3.2. Prijedlog rješavanja

Očekuje se da će analiza literature ukazati na osnovne funkcionalne, strukturne i druge zahtjeve prema referentnim arhitekturama, a posebno prema referentnim arhitekturama za razvoj ponovno iskoristivih komponenata za poslovne aplikacije. Pregledom literature, analizom postojećih referentnih arhitektura za poslovne aplikacije, te temeljem iskustva autora ovog istraživanja dobiti će se dovoljno informacija o tomu koje sve karakteristike referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera treba implementirati. Pouzdanost u relevantnost definiranih funkcionalnih zahtjeva referentne arhitekture potvrditi će se na temelju ispitivanja stručnjaka koji imaju iskustvo u istraživanju i praksi, putem anketnog upitnika.

4.3.3. Razvoj artefakta

Polazeći od karakteristika referentne arhitekture definiranih u prethodna dva koraka, potrebno je odgovoriti i na pitanje; kako iskoristiti navedeno znanje za razvoj artefakta referentne arhitekture. U ovoj fazi se obavlja glavnina dizajna artefakta. Tehnički dio razvoja obaviti će se korištenjem suvremenih ali i u praksi dokazanih tehnika i alata kao što su: Eclipse 4, Java 7, Aspect Oriented, Patterns, Model Driven, Component Based, Object Oriented, Generative

Programming, itd. Nove spoznaje i rezultati razvoja artefakata u ovoj fazi mogu zahtijevati analizu alternativnih načina rješavanja problema, odnosno ponavljanje prvog i drugog koraka.

4.3.4. Provjera artefakata

Artefakti referentne arhitekture će se provjeravati empirijskim metodama „kako bi utvrdili kako dobro artefakti rade“ (Siegmund i ostali, 2012; V. K. Vaishnavi & Kuechler Jr, 2007; Wohlin, Höst, & Henningsson, 2003). Za provjeru da se referentna arhitektura može primjenjivati u praksi (engl. *utility*) koristi će se longitudinalna studija pojedinačnog slučaja tipa „poboljšanja“ (engl. *improving*) u jednoj financijskog instituciji. Za procjenu dovoljne upotrebljivosti (engl. *usability*) referentne arhitekture koristiti će se kontrolirani eksperiment (engl. *controlled experiments*) u kojem će sudjelovati nekoliko razvojnih programera koji referentnu arhitekturu koriste u praksi. U organizaciji u kojoj će se provoditi studija slučaja, prikupljati će se kvantitativni i kvalitativni podaci te ih statistički analizirati i koristiti (korelacija, multipla linearna regresija, deskriptivna statistika) kako bi se, pored ostalog, utvrdili faktori održavljivosti pojedine aplikacijske poslovne komponente koja koristi referentnu arhitekturu, te za potvrdu valjanosti predloženog modela (PR) za provjeru održavljivosti linije za proizvodnju softvera i povezanosti sa standardnim modelom (MI).

Osim provjere artefakta, mjerenjem i analizom metrika napraviti će se i usporedba artefakta sa poznatim referentnim vrijednostima (Mattson & Bosch, 1999), te procjena zrelosti referentne arhitekture linije za proizvodnju softvera pomoću metode *Family Evaluation Framework* (FEF).

Tablica 5 Strategija validacije referentne arhitekture

Istraživačka pitanja	Pristup	Validacija	Znanstveni doprinos
(**P1**) *Koje funkcionalne zahtjeve treba zadovoljiti referentna arhitektura za poslovne aplikacije prema pristupu linija za proizvodnju softvera?*	Istraživanje postojećeg teorijskog okvira i analiza postojećih referentnih arhitektura; praktično iskustvo autora u razvoju poslovnih aplikacija.	Znanost o oblikovanu (engl. *design science*), definiranje problema (engl. *awarness of the problem*) i prikladni predlošci (poglavlje 5).	Terminološko određenje osnovnih pojmova u području istraživanja. Definirani i provjereni funkcionalni zahtjevi referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera.
	Definiranje funkcionalnih zahtjeva referentne arhitekture i njihova revizija na temelju ispitivanja stručnjaka.	Ispitivanje (engl. *survey*) stručnjaka putem anketnog upitnika.	
(**P2**) *Može li se razviti referentna arhitektura koja zadovoljava te zahtjeve?*	Iterativno (u više ciklusa) oblikovanje i implementacija artefakata referentne arhitekture prema definiranim zahtjevima.	Opis referentne arhitekture, te objašnjenje načina na koji su definirani zahtjevi ugrađeni u artefakte referentne arhitekture.	Razvijeni artefakti referentne arhitekture za poslovne aplikacije prema definiranim funkcionalnim zahtjevima.
(**P3**) *Da li je referentna arhitektura koja zadovoljava te zahtjeve korisna u praksi?*	Potvrda upotrebljivosti referentne arhitekture u praksi.	Kontrolirani eksperiment *(*engl. *controll experiment).*	Unaprjeđenje referentne arhitekture poslovnih aplikacija na temelju provođenja nekoliko ciklusa: definiranja, oblikovanja, razvoja i provjere artefakata.
	Procjena stabilnosti okvira referentne arhitekture.	Provjera stabilnosti (engl. *stability assessment*) prema (Bosch i Mattsson).	
	Korištenje referentne arhitekture u praksi i prikupljanje podataka o korištenju, npr. metrike izvornog programskog koda, broj korisničkih transakcija.	Studija slučaja (*engl. case study*).	Provjera primjenjivosti referentne arhitekture na studiji slučaja primjene u razvoju poslovnih aplikacija.
(**P4**) *Postoji li povezanost između predloženog modela PR za provjeru održavljivosti linije za proizvodnju softvera i standardnog modela MI?*	Korištenje metrika izvornog programskog koda za mjerenje povezanosti između modela PR za provjeru održavljivosti linije za proizvodnju softvera i modela MI.	Deskriptivna statistika, korelacija (*Pearson*).	Razvoj i potvrda novog modela (PR) i metrika za ocjenu održavljivosti linije za proizvodnju softvera za poslovne aplikacije u odnosu na način upotrebe i stupanj korištenja referentne arhitekture u aplikacijskim poslovnim komponentama.
	Pronalaženje adekvatnog modela (linearne kombinacije) PR za mjerenje održavljivosti komponenata poslovnih aplikacija.	Linearna regresija.	

4.3.5. Zaključak

Ovo istraživanje daje teoretski i praktični doprinos u svakoj od pet faza istraživanja. U fazi definiranja problema odrediti će se stanje i problemi koji se odnose na referentnu arhitekturu poslovnih aplikacija. Posebno će se istražiti pristup poznat kao *linije za proizvodnju softvera*, koje nisu dovoljno istražene u području razvoja poslovnih aplikacija, pri čemu detaljna analiza tog pristupa može biti značajan teoretski doprinos. U fazi prijedloga rješenja doći će se do podataka o važnim karakteristikama koje referentna arhitektura treba posjedovati. U fazi razvoja artefakta razviti će se primjerak referentne arhitekture za koju se očekuje da će u primjeni dati dobre rezultate na temelju standardnih mjerenja, te prema metrikama koja će biti predložene u okviru ovog istraživanja. U fazi provjere artefakta provjeriti će se korisnost referentne arhitekture i njen utjecaj na održavljivost aplikacijskih poslovnih komponenata. Na temelju provedenih istraživanja, u završnoj fazi izvesti će se ocjene i zaključci te preporuke za daljnja istraživanja. Također, podaci koje će se dobiti u fazi provjere mogu poslužiti kao osnova za daljnje istraživanje u području razvoja poslovnih aplikacija prema pristupu linija za proizvodnju softvera.

4.3.6. Studija slučaja

Za provjeru točnosti hipoteze (H1) koristimo studiju slučaja (engl. *case study*) kako bi istražili primjenjivost (engl. *utility*) artefakata referentne arhitekture putem njihove primjene u praksi za razvoj poslovnih aplikacija. Cilj validacije primjenjivosti artefakata je dokazivanje da je referentna arhitektura primjenjiva za razvoj poslovnih aplikacija u jednom od poslovnih sektora. Validacija se provodi kroz primjenu artefakata za razvoj 16 poslovnih aplikacija, promatranje njihovog korištenja, analizu interakcija korisnika sa poslovnim aplikacijama. Metoda studije slučaja je postupak kojim se izučava neki pojedinačni slučaj iz određenoga znanstvenog područja, ali ne predstavlja znanstvenu metodu u pravom smislu te riječi, jer se samo na temelju rezultata promatranja više slučajeva mogu izvući određene zakonitosti (Zelenika, 2011). Promatranje i analiziranje jednog slučaja metodom studija slučaja ne dovodi do stvaranja zakonitosti poopćavanjem koje možda i postoje, već je ono često povod za šira i dublja istraživanja. Za razliku od kvantitativnih metoda istraživanja, poput anketa, koje su fokusirane na pitanja *tko, što, gdje, koliko*, studija slučaja uobičajena je strategija kada se postavljaju pitanja *kako* i *zašto*.

Slučaj kojeg analiziramo u ovom radu odnosi se na jednu financijsku instituciju koja koristi predloženu referentnu arhitekturu; obrađujemo ga longitudinalno pri čemu su objekti istraživanja (engl. *object of the study*) poslovne komponente koje se koriste u razvoju poslovnih aplikacija. Glavnina podatka za istraživanje nalazi se u repozitoriju izvornog programskog koda poslovnih aplikacija i artefakata referentne arhitekture za razdoblje od tri godine. Longitudinalnu studiju (ponavljanje istraživanja nakon svake nove inačice referentne arhitekture) koju za istraživanje fenomena linija za proizvodnju softvera preporučuje (Runeson, Host, Rainer, & Regnell, 2012, str. 101) koristimo kako bi ublažili nedostatak ove metode vezano za nemogućnost poopćavanja rezultata istraživanja.

Glavni povod za korištenje metode studija slučaja najčešće se odnosi na poboljšanje nečega u organizaciji ili na projektu; u znanstvenom istraživanju povod se najčešće odnosi na znanstveni doprinos u vidu novih otkrića. Predložena referentna arhitektura predstavlja oboje, priliku za poboljšanje postojeće inačice, te doprinos u vidu novo predloženih artefakata i pristupa razvoju poslovnih aplikacija. Tip studije slučaja kojeg koristimo može se smatrati poboljšavajućim tipom (engl. *improving*) budući je glavni cilj ovog istraživanja poboljšanje postojeće referentne arhitekture. Metodu studija slučaja smo odabrali kako bi postavljene hipoteze u prvom poglavlju provjerili u praksi. Prilikom prikupljanja podataka u analizi slučaja koristimo protokol istraživanja (engl. *case study protocol*) i nekoliko načina čuvanja prikupljenih podataka koji sadrže sve dokaze konačnih zaključaka kojima potkrjepljujemo pouzdanost ovog istraživanja.

4.3.7. Ispitivanje

Metodu ispitivanja koristimo za ispitivanje mišljenja stručnjaka iz prakse kako bi povećali pouzdanost u primjerenost predloženog rješenja za referentnu arhitekturu poslovnih aplikacija. Glavni ciljevi korištenja ovog instrumenta istraživanja su pronalaženje odgovora na pitanje (P1), provjera važnosti svake od funkcionalnih karakteristika, razumijevanje njihovog relativnog značaja, te pronalaženje dodatnih funkcionalnosti koje su eventualno izostavljene u početnom skupu funkcionalnih zahtjeva. Pitanja u upitniku su uglavnom pitanja zatvorenog tipa - nude se mogući odgovori, sa najviše pet ponuđenih intenziteta, dok je samo jedno od pitanja otvorenog tipa. Uzorak ispitanika je izabran na temelju analize javno dostupnih profila korisničkih grupa na trenutno najznačajnijoj društvenoj mreži *LinkedIn* koja, među ostalima, okuplja i stručnjake iz područja istraživanja ovog rada.

4.3.8. Eksperiment

Za provjeru točnosti hipoteze (H1) koristimo kvazi-eksperiment (engl. *qasi-experiment*) kako bi istražili upotrebljivost (engl. *usability*) artefakata referentne arhitekture ispitivanjem njihovih korisnika, razvojnih programera. Cilj mjerenja upotrebljivosti je dokazivanje da je referentna arhitektura u dovoljnoj mjeri upotrebljiva. Mjerenje se provodi ispitivanjem 5 razvojnih programera koji su koristili ili koriste predloženu referentnu arhitekturu za razvoj poslovnih aplikacija. Upitnik za provjeru upotrebljivosti se sastoji od četiri grupe istraživačkih pitanja prema (Nielsen, 1994). U postupku analize rezultata koristiti će se metoda komparativne analize.

4.4. Objavljeni radovi

Tablica 6 prikazuje objavljene radove autora za vrijeme trajanja ovog istraživanja koji su usko povezani sa nekom od aktivnosti u procesu razvoja artefakata referentne arhitekture.

Tablica 6 Objavljeni radovi povezani sa istraživanjem

Razvoj artefakata	***Provjera artefakata***
(OR3) (Roško, 2014a)	*(OR1)* (Roško & Strahonja, 2014)
(OR5) (Rosko, 2013b)	*(OR2)* (Roško, 2014b)
(OR6) (Rosko, 2012)	*(OR3)* (Roško, 2014a)
(OR7) (Rosko, 2011)	*(OR4)* (Rosko, 2013a)
(OR8) (Rosko, 2010a)	
(OR9) (Rosko & Konecki, 2008a)	
(OR10) (Roško, 2007)	

(OR1) A Case Study of Software Product Line for Business Applications Changeability Prediction (Roško & Strahonja, 2014) opisuje primjer upotrebe predloženih metrika izvornog programskog koda i modela PR za mjerenje održavljivosti komponenata poslovnih aplikacija u jednoj financijskoj instituciji.

(OR2) Predicting the Changeability of Software Product Lines for Business Application (Roško, 2014b) analizira međusobne ovisnosti odnosno povezanosti rezultata standardnog modela (MI) i predloženog modela (PR) za procjenu održavljivosti softverskih komponenata u kontekstu linija za proizvodnju softvera.

(OR3) Case Study: Refactoring of Software Product Line Architecture-Feature Smells Analysis (Roško, 2014a) opisuje proces prepravljanja postojećih inačica referentne arhitekture i poslovnih komponenata s ciljem njihovog poboljšanja.

(OR4) Assessing the Responsibility of Software Product Line Platform Framework for Business Applications (Rosko, 2013a) predlaže nove metrike za procjenu održavljivosti linija za proizvodnju softvera.

(OR5) Business applications architecture model based on software product line approach (Rosko, 2013b) opisuje način definiranja funkcionalnih zahtjeva referentne arhitekture za poslovne aplikacije, te predlaže način njihove implementacije i provjere njihove valjanosti.

(OR6) *Strategy Pattern as a Variability Enabling Mechanism in Product Line Architecture* (Rosko, 2012) prikazuje način upravljanja sa varijabilnošću referentne arhitekture linija za proizvodnju softvera putem korištenja uzorka oblikovanja.

(OR7) Source Code Organization for 3-tier OLTP Architecture Systems (Rosko, 2011) opisuje predloženi način organizacije izvornog programskog koda u kontekstu razvoja interaktivnih poslovnih aplikacije prema pristupu linija za proizvodnju softvera.

(OR8) Backward-Forward Transaction Service Design Pattern (Rosko, 2010a) predlaže novi način upravljanja sa transakcijama poslovnih aplikacija u slučajevima kada se vrijeme jedne transakcije proteže na duži vremenski period od standardnih transakcija.

(OR9) Dynamic Data Access Object Design Pattern (Rosko & Konecki, 2008a) odnosi se na prijedlog novog uzorka oblikovanja za upravljanje programskom logikom pristupa podacima.

(OR10) Sovereign Value Object Design Pattern (Roško, 2007) predlaže sveobuhvatni način čuvanja i prijenosa poslovnih podataka u vidu objekata unutar aplikacija, od izvora podataka do njihovog korištenja i spremanja.

5. RAZVOJ REFERENTNE ARHITEKTURE

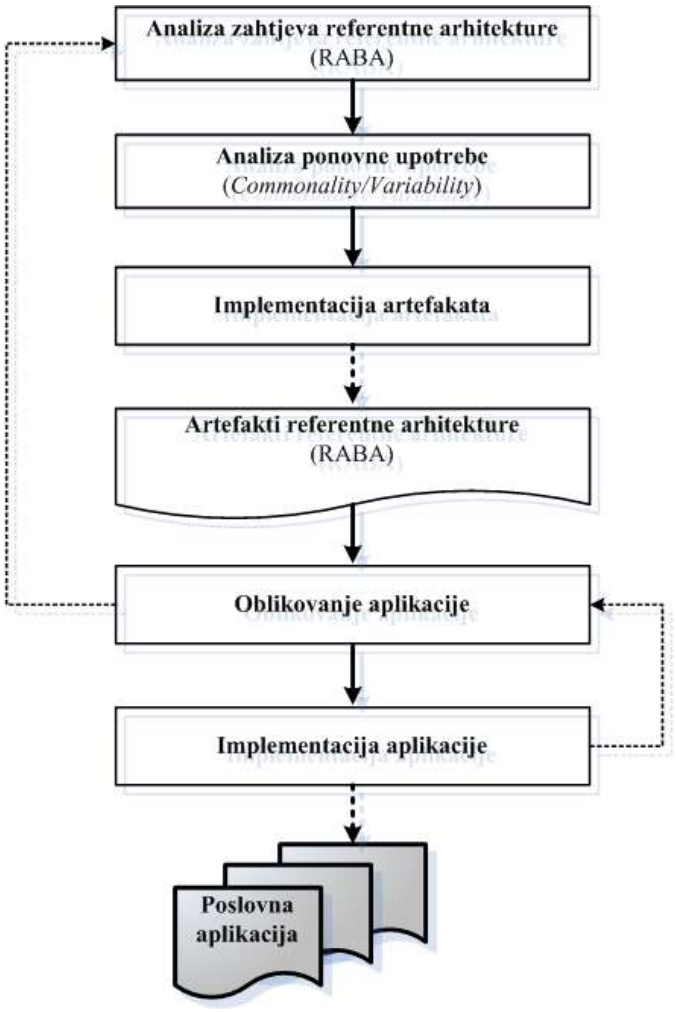
U ovom poglavlju predstavljamo predloženu referentnu arhitekturu za poslovne aplikacije (RABA), predlažemo inicijalne funkcionalne zahtjeve njenog aplikacijskog okvira, provjeravamo njihovu relevantnost ispitivanjem stručnjaka iz područja istraživanja, opisujemo oblikovanje i razvoj artefakata referentne arhitekture, analiziramo metrike postojećih inačica okvira referentne arhitekture u cilju njihovog poboljšanja, opisujemo podjelu okvira na podsustave, prikazujemo upotrebu uzoraka dizajna kod oblikovanja okvira te opisujemo način korištenja artefakata referentne arhitekture pri razvoju poslovnih aplikacija.

Ovim poglavljem obuhvaćeni su slijedeći ciljevi ovog istraživanja: (C1.3) definirati funkcionalnosti referentne arhitekture, (C1.4) ispitati mišljenje iskusnih arhitekata i istraživača o relevantnosti definiranih funkcionalnosti referentne arhitekture, (C2) odrediti i razviti artefakte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera koji zadovoljavaju definirane funkcionalne zahtjeve.

5.1. Referentna arhitektura za poslovne aplikacije (RABA)

Referentna arhitektura za poslovne aplikacije (RABA) koja se predlaže u ovom radu, primjenjuje se za razvoj i održavanje poslovnih aplikacija u više različitih poslovnih sektora. Primjena ima karakter vertikalnog prilagođavanja pojedinog poslovnog sektora horizontalnoj referentnoj arhitekturi te njeno eventualno proširivanje koje je specifično za pojedini poslovni sektor. Budući da se referentna arhitektura razvija prema pristupu linija za proizvodnju softvera, moguće je da se u tijeku njenog prilagođavanja pojedinom poslovnom sektoru razviju i komponente koje se mogu koristiti i u drugim poslovnim sektorima; takve komponente se u procesu razvoja prepoznaju i uvrštavaju u osnovne artefakte na horizontalnoj razini kako bi se mogli primjenjivati i u ostalim poslovnim sektorima za razvoj poslovnih aplikacija. Slika 31 prikazuje proces razvoja referentne arhitekture i proces razvoja poslovnih aplikacija koje primjenjuju predloženu referentnu arhitekturu.

Razvoj artefakata predložene referentne arhitekture (RABA) odvijao se u nekoliko faza; za rezultat imamo definiranu zadnju inačicu referentne arhitekture (V4) koju koristimo u praksi. U procesu definiranja i razvoja artefakata referentne arhitekture koristili smo pored drugih metoda softverskog inženjerstva i nekoliko predložaka metode znanost o oblikovanju (engl. *design science*), tablica 3.



Slika 31 Proces razvoja aplikacija i artefakata referentne arhitekture

5.2. Definiranje problema (engl. *awarness of problem*)

Tehnike softverskog inženjerstva najčešće se u organizacije uvode iz poslovnih razloga, zbog poboljšanja kvalitete, cijene i vremenskog razdoblja izrade ili održavanja poslovnih aplikacija, dok je utjecaj samih tehnika na formuliranje poslovnih strategija organizacija uglavnom neznatan. S druge strane, *Software Product Line Engineering* (SPLE) pristup razvoju poslovnih aplikacija, kao sveobuhvatna tehnika koja se odnosi na cijeli niz aplikacija, značajno utječe na definiranje poslovnih strategija organizacija (F. J. Linden i ostali, 2010, str. 21). Poslovni razlozi su ti koji utječu na donošenje odluka o tome da li uvesti SPLE pristup u organizaciju, u kojem opsegu, za koje poslovne aplikacije, koje funkcionalnosti će imati te aplikacije, te kako organizirati razvoj i održavanje aplikacija u organizaciji; stoga je nužna usklađenost poslovne

strategije organizacije sa SPLE pristupom razvoju i održavanju aplikacija u cilju značajnijeg unaprjeđenja sveukupnog poslovanja organizacije.

Jedinstvena arhitektura za više poslovnih aplikacija najvažniji je dio informacijskog sustava u kojem aplikacije zajednički koriste unaprijed definirane funkcionalnosti. Takva jedinstvena arhitektura značajno određuje funkcionalnosti koje poslovne aplikacije potencijalno mogu imati, te je ujedno i glavni činitelj u određenju atributa kvalitete cjelokupnog informacijskog sustava. Zajednička arhitektura linije za proizvodnju softvera naziva se referentna arhitektura (engl. *reference architecture*). Referentna arhitektura definira stil, organizaciju sastavnih komponenata, njihovu povezanost, te glavne principe koje poslovne aplikacije od nje nasljeđuju. Korištenje referentne arhitekture za poslovne aplikacije moguće je proširiti na područje koje obuhvaća više poslovnih sektora (engl. *domains*), te pronaći zajedničke funkcionalne karakteristike koji se mogu koristiti pri razvoju poslovnih aplikacija općenito, bez ograničenja na konkretan poslovni sektor.

Identifikacija područja istraživanja (engl. *Problem Area Identification*)

Odgovore na istraživačko pitanje P1 („*Koje funkcionalne zahtjeve treba zadovoljiti referentna arhitektura za poslovne aplikacije prema pristupu linija za proizvodnju softvera?*“) temeljimo na istraživanju postojećeg teorijskog okvira i analizi postojećih raspoloživih referentnih arhitektura iz područja razvoja poslovnih aplikacija, uključujući i praktično iskustvo autora u razvoju poslovnih aplikacija u nekoliko različitih poslovnih sektora poput telekomunikacija, prometa, turizma, proizvodnje, farmaceutike, financija i bankarstva.

U praksi uvođenja SPLE pristupa u organizacije, uobičajeno je obuhvatiti samo jedan poslovni sektor, primjerice bankarstvo, pri čemu se referentna arhitektura i njeni artefakti definiraju u tako zadanim okvirima. Međutim, poslovne aplikacije imaju veliki broj zajedničkih funkcionalnosti, bez obzira kojem poslovnom sektoru pripadaju. Takav skup zajedničkih funkcionalnih zahtjeva moguće je definirati u obliku artefakata referentne arhitekture za poslovne aplikacije općenito.

Formulacija problema (engl. *Problem Formulation*)

Problem smo formulirali na temelju uočene potrebe za objedinjavanjem zajedničkih funkcionalnosti referentne arhitekture poslovnih aplikacija koje pripadaju različitim poslovnim sektorima u organizacijama koje se bave razvojem poslovnih aplikacija za više poslovnih sektora istovremeno. Osim toga potrebno je pronaći način zadovoljavanja funkcionalnih zahtjeva koji su specifični samo za pojedine poslovne sektore. Tako objedinjeni zajednički

funkcionalni zahtjevi u obliku referentne arhitekture, prema pristupu linija za proizvodnju softvera, omogućili bi organizacijama postizanje potrebnih karakteristika kvalitete, cijene i rokova za razvoj i održavanje poslovnih aplikacija; na taj način značajno umanjujući rizike koji se odnose na navedene karakteristike. Ostali funkcionalni zahtjevi koji su specifični samo za pojedine poslovne sektore, mogu se zadovoljiti razvojem poslovnih komponenata na temelju prethodno definirane referentne arhitekture. Način formulacije problema navodi na zaključak da se radi o razvojnom umjesto o istraživačkom problemu, međutim, problem koji obuhvaća definiranje, razvoj i provjeru artefakata referentne arhitekture uz pomoć metode znanost o oblikovanju (engl. *design science*), pri čemu otkrivamo slučajeve kada artefakti ne funkcioniraju „u skladu sa teorijom“, možemo smatrati znanstvenim problemom.

Iskorištavanje stručnosti u području istraživanja (engl. *Leveraging Expertise*)

Navedeni problem i pristup njegovom rješavanju u razvoju poslovnih aplikacija poznati su autoru iz praktičnog iskustva u razvoju aplikacija u nekoliko različitih poslovnih sektora. Organizacije koje razvijaju ili održavaju poslovne aplikacije najčešće nisu u stanju pratiti poslovne potrebe i mogućnosti organizacija koje aplikacije naručuju ili koriste. Glavni izazovi koji se pri tome nalaze pred organizacijama koje razvijaju ili održavaju takve sustave odnose se na definiranje jedinstvene referentne arhitekture i organizaciju pojedinih aktivnosti, uloga u procesima i podjelu odgovornosti.

Praćenje stanja i trendova u području istraživanja (engl. *Industry and Practice Awareness*)

Kod razvoja poslovnih aplikacija moguće je, osim vlastitog razvoja potrebnih komponenata, koristiti i veliki broj gotovih komponenata drugih proizvođača koje služe za pristup, spremanje, obradu i prikaz podataka. Specifično za *Java* okruženje, takve komponente potječu iz različitih izvora i od raznih proizvođača, primjerice: *Apache.org, Sourceforge.net, Sun.com, Microsoft.com, Smartclient.com, Highchart.com.* Kompleksnost takvog okruženja povećava i činjenica da se takve komponente, njihova okolina i potrebni alati koji se koriste za njihovu pripremu učestalo mijenjaju. Poslovne aplikacije same po sebi također imaju različite inačice u primjeni, u isto vrijeme i to potencijalno u više poslovnih organizacija.

Problemi koji nastaju u praksi zbog navedenih činjenica odnose se na otežano održavanje razvijenih aplikacija i na njihovo eventualno proširivanje na temelju poslovnih potreba organizacija koje ih koriste. S druge strane, razvoj poslovnih aplikacija postaje kompleksan jer zahtjeva široki spektar znanja razvojnih programera koja najčešće nisu raspoloživa, dok

korisnici poslovnih aplikacija češće nego nekad traže izmjene u poslovnim aplikacijama i to u sve kraćim rokovima.

Osim navedenih tehnoloških i poslovnih odrednica koje uvjetuju njihov razvoj, poslovne aplikacije danas koriste razne na zahtjev raspoložive udaljene resurse (engl. *cloud computing*) kao što su: geografski podatci, spremišta podataka, prevoditelji, analitički podatci, sigurnosni servisi i ostalo. Isto tako, korisnici poslovnih aplikacija sve više traže pristup aplikacijama koristeći *web* i mobilne kanale, istovremeno uz postojeće kanale pristupa.

Motivacija za ovo istraživanje temelji se na dugogodišnjoj potrebi omogućavanja ponovne upotrebe postojećih komponenata referentne arhitekture u razvoju novih poslovnih aplikacija u kontekstu koji obuhvaća različite poslovne sektore. Problem nije samo povezan sa arhitekturom poslovnih aplikacija već se odnosi i na organizaciju i proces razvoja zajedničkih komponenata i aplikacija koje te komponente koriste.

Nepodudarnost postojećih rješenja (engl. *Solution Scope Mismatch*)

Postojeće referentne arhitekture kao što su *Spring, Scout, Vadiin, SmartGwt, Roma Framework, Openxava*, zadovoljavaju većinu zahtjeva koje poslovne aplikacije imaju, dok s druge strane otežavaju integraciju sa postojećim sustavima, zahtijevaju široki spektar znanja od razvojnog programera za poslovne aplikacije, značajno onemogućavaju zadovoljavanje specifičnih zahtjeva poslovnih korisnika, te predstavljaju potencijalni rizik za organizaciju u slučaju njihovog nestanka sa tržišta. Definiranje vlastite referentne arhitekture ne isključuje korištenje komponenata iz postojećih raspoloživih referentnih arhitektura ili drugih raspoloživih komponenata raznih proizvođača. Za razliku od korištenja postojećih, vlastita referentna arhitektura omogućava potpunu kontrolu nad procesima razvoja i održavanja poslovnih aplikacija, značajno umanjuje tehnološke rizike. S druge strane, vlastiti razvoj osim poslovno utemeljene odluke, zahtjeva posjedovanje značajnog iskustva i specifičnih znanja koja su potrebna za ublažavanje eventualnih rizika.

Vizionarski pristup (engl. *Being Visionary*)

Definiranje i razvoj vlastite referentne arhitekture za razvoj poslovnih aplikacija općenito, koja bi se mogla proširivati za pojedine poslovne sektore, omogućila bi potpunu kontrolu i upravljanje eventualnim rizicima, bilo na način da se korite komponente iz postojećih referentnih arhitektura, njihovom zamjenom u određenim uvjetima, bilo razvojem vlastitih rješenja. Postavlja se pitanje razine apstrakcije na koju se referentna arhitektura odnosi. Organizacije koje razvijaju poslovne aplikacije za više poslovnih sektora susreću se sa

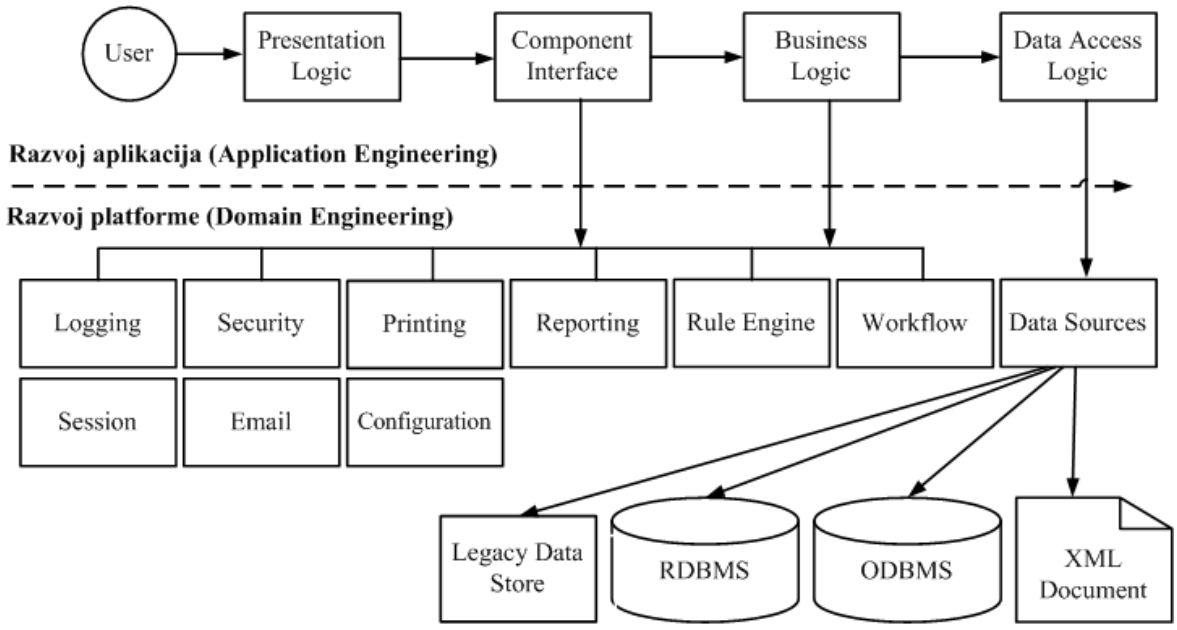
ponavljajućim funkcionalnostima koje je zbog budućeg efikasnijeg održavanja i proširivanja potrebno organizirati na učinkovit način. Predlažemo rješenje koje se temelji na postavljanju apstrakcije na horizontalnoj razini koja obuhvaća skup funkcionalnosti koje se mogu primijeniti u aplikacijama iz različitih poslovnih sektora, dok specifične zahtjeve pojedinog poslovnog sektora možemo definirati na vertikalnoj razini apstrakcije koja je specifična za svaki pojedini poslovni sektor.

5.3. Prijedlog rješavanja (engl. *suggestion*)

Predložena referentna arhitektura i artefakti temelje se na konceptu prema kojem je najvažnije da organizacije definiraju vlastiti kontekst za korištenje komponenata od kojih se sastoje njihove poslovne aplikacije. Sredstvo za stvaranje takvog konteksta je aplikacijski okvir (engl. *application framework*) (Crnkovic & Larsson, 2002, str. 14). Naše predloženo rješenje odnosi se na objektno orijentirani klasni okvir (engl. *object-oriented class framework*) koji je prvenstveno odgovoran za kontrolu načina rada povezanih komponenata i za rješavanje kompleksnih interakcija sa vanjskim sustavima, nazivamo ga i programski okvir ili okvir referentne arhitekture. Prema (Szyperski, 2002, str. 546) takav okvir sadrži skup klasa i samo dio njihovih interakcija koji je zajednički za veći broj aplikacija u području njegove primjene. Aplikacije ili njihove komponente nasljeđuju klase iz okvira, pri čemu su razvojni programeri aplikacija potpuno upoznati sa detaljima okvira u cilju boljeg razumijevanja i predlaganja njegovog kontinuiranog unaprjeđenja. Ovakva vrsta aplikacijskog okvira u teoriji se naziva otvoreni okvir (engl. *whitebox*), međutim, u kontekstu organizacije razvoja prema pristupu linija za proizvodnju softvera, preciznije bi bilo nazvati ga stakleni okvir (engl. *glassbox*) budući da se njegovim razvojem bavi samo jedan dio organizacije (engl. *domain engineering*) dok su ostali dionici upoznati sa njegovim izvornim kodom. U predloženom konceptu razlikujemo dvije grupe komponenata: poslovne komponente i generičke infrastrukturne komponente. Poslovna komponenta predstavlja implementaciju autonomnog poslovnog koncepta ili procesa koji sadrži potrebne artefakte za njegovu reprezentaciju, implementaciju, i autonomnu instalaciju kao ponovno upotrebljivog elementa nekog većeg distribuiranog informacijskog sustava (Herzum & Sims, 2000). Infrastrukturne komponente koje sadrže programski kod za rješavanje tehničkih kompleksnosti dijelimo na vlastite, komercijalne (engl. *commercial off-the-shelf*) i nekomercijalne (engl. *open source*). Komponente koje su sastavni dijelovi poslovnih aplikacija, definiraju se na optimalnoj razini apstrakcije, imajući u vidu

njihovu pripadnost nekom poslovnom podsustavu, ponovno ih se koristi u razvoju drugih poslovnih aplikacija na način koji je poslovno, organizacijski i tehnološki najučinkovitiji.

Slika 32 ilustrira predloženi koncept podjele odgovornosti kod razvoja poslovnih aplikacija na dva područja, razvoj aplikacija i razvoj platforme; drugi dio predloženog koncepta odnosi se na strukturu poslovnih aplikacija. Osnovu predložene strukture čine slojevi (engl. *layers*), prema istoimenom uzorku za arhitekturu (Buschmann, Henney, & Schmidt, 2007, str. 185), koji značajno umanjuju napor i cijenu razvoja aplikacija (Zweben, Edwards, Weide, & Hollingsworth, 1995).

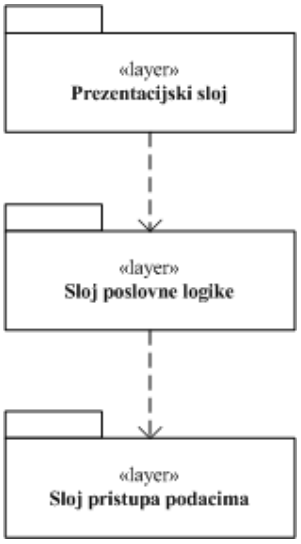


Slika 32 Koncept razdvajanja razvoja aplikacija i infrastrukture (platforme)

Slika 33 ilustrira predloženi koncept slojeva aplikacija koje dijelimo na tri logičke cjeline: *prezentacijski sloj*, *sloj poslovne logike* i *sloj pristupa podacima*. Prezentacijski sloj sadrži programske elemente koji su odgovorni za prikazivanje podataka i raspoloživih funkcija korisniku aplikacije; sloj poslovne logike je odgovoran za implementaciju poslovne logike i poslovnih pravila; sloj za pristup podacima sadrži programske elemente za pristup izvorima podataka kao što su primjerice relacijske baze. Ovakav koncept podrazumijeva da komunikacija između slojeva aplikacija može ići samo u smjeru od višeg prema nižem sloju (engl. *strict layering*). Osim značajnih prednosti takvog ograničenja, kao što je olakšano i postepeno razumijevanje, sloj po sloj, postoje i manje važni njegovi nedostaci (Szyperski, 2002, str. 162).

Prednost primjene slojeva u arhitekturi poslovnih aplikacija, kao i aplikacija općenito odnosi se na slijedeće:

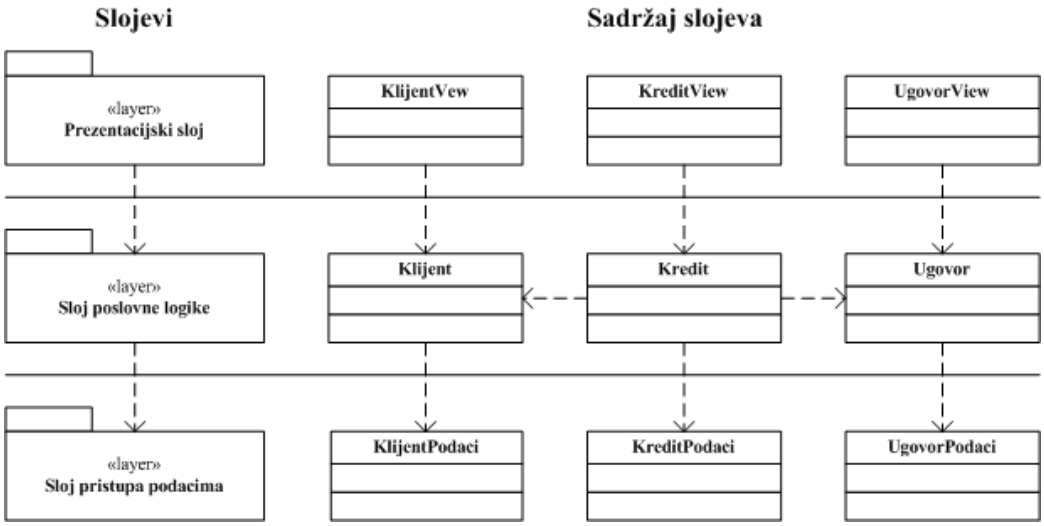
- Umanjivanje njihove kompleksnosti
- Olakšavanje održavanja
- Izolaciju najmanje stabilnih elementa
- Lakše pronalaženje komponenata koje su najčešće predmet ponovne uporabe
- Efikasniju organizaciju timova za razvoj s obzirom na njihovu specijalizaciju



Slika 33 Slojevi prema odgovornosti

Podjela na tri logička sloja koju smo prikazali odnosi se na organizaciju strukturu poslovnih aplikacija prema odgovornosti svakog pojedinog sloja. Ovakva podjela može olakšati razvoj i održavanje poslovnih aplikacija budući da su pojedine odgovornosti izolirane jedna od druge.

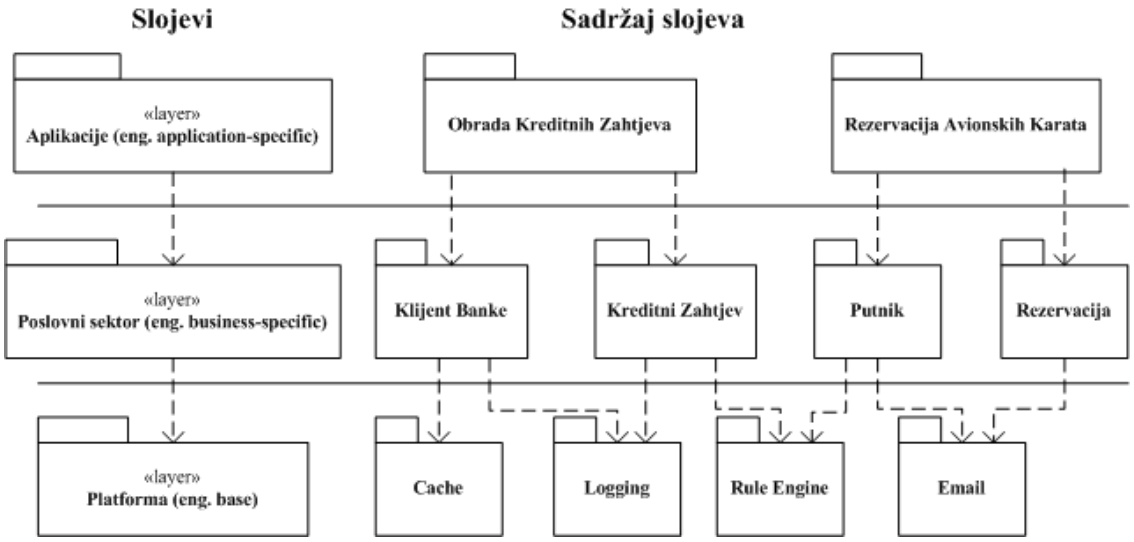
Slika 34 ilustrira način implementacije svake od navedenih odgovornosti kao zasebnog sloja. Važno je napomenuti da logička podjela na slojeve kakvu predlažemo nije nužno i fizička podjela, primjerice, prezentacijski sloj može se izvoditi unutar istog procesa kao i sloj poslovne logike, ili pak unutar drugog procesa ukoliko to korisnik poslovne aplikacije zahtjeva. Pored podjele slojeva prema odgovornosti, slojeve predlažemo podijeliti i prema razini ponovne upotrebe (engl. *reuse*) u strukturi aplikacije, i to prema pristupu linija za proizvodnju softvera. Takav način podjele predložio je (Jacobson i ostali, 1997).



Slika 34 Primjer slojeva prema podjeli odgovornosti

Linije za proizvodnju softvera za poslovne aplikacije, kod kojih se na temelju iste platforme mogu razvijati komponente i aplikacije za različite poslovne sektore, prikladne su za takvu vrstu podjele. Slika 35 prikazuje primjer strukture poslovnih aplikacija podijeljene prema razini ponovne upotrebe komponenata od kojih se aplikacije sastoje. Najniži sloj (engl. *base*) sadrži komponente koje se mogu koristiti pri razvoju aplikacija za više različitih poslovnih sektora. Srednji sloj (engl. *business specific*) sastoji se od komponenata koje su specifične za poslovni sektor ili organizaciju, primjerice komponente za bankarski sektor grupiraju se posebno u odnosu na komponente za druge poslovne sektore. Najviši sloj (engl. *application specific*) sadrži komponente koje pripadaju konkretnoj aplikaciji ili projektu dok je mogućnost njihove ponovne upotrebe vrlo mala.

Ovo istraživanje odnosi se na razvoj artefakata prema predloženom konceptu. Artefakti referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera uključuju: okvir referentne arhitekture, pomoćni alat (engl. *plug-in*), opis arhitekture i referentnu aplikaciju. Glavni ciljevi ovakvog pristupa uključuju: definiranje modela poslovnih aplikacija visoke razine (engl. *high-level*), implementaciju standardnih rješenja za ponavljajuće probleme, te pojednostavljivanje razvoja poslovnih aplikacija. Da bi razvoj poslovnih aplikacija bio efektivan, razvojni ciklus se mora skratiti, što implicira na potrebu za standardnom platformom za poslovne aplikacije općenito. Takvu platformu nije moguće, niti je poslovno opravdano, razviti odjednom kako bi se mogla iskoristiti za razvoj prve aplikacije, već je njen razvoj potrebno provoditi kontinuirano i paralelno sa razvojem aplikacija.



Slika 35 Primjer slojeva prema razini ponovne upotrebe

Princip općenitog rješavanja problema (engl. *General Solution Principle*)

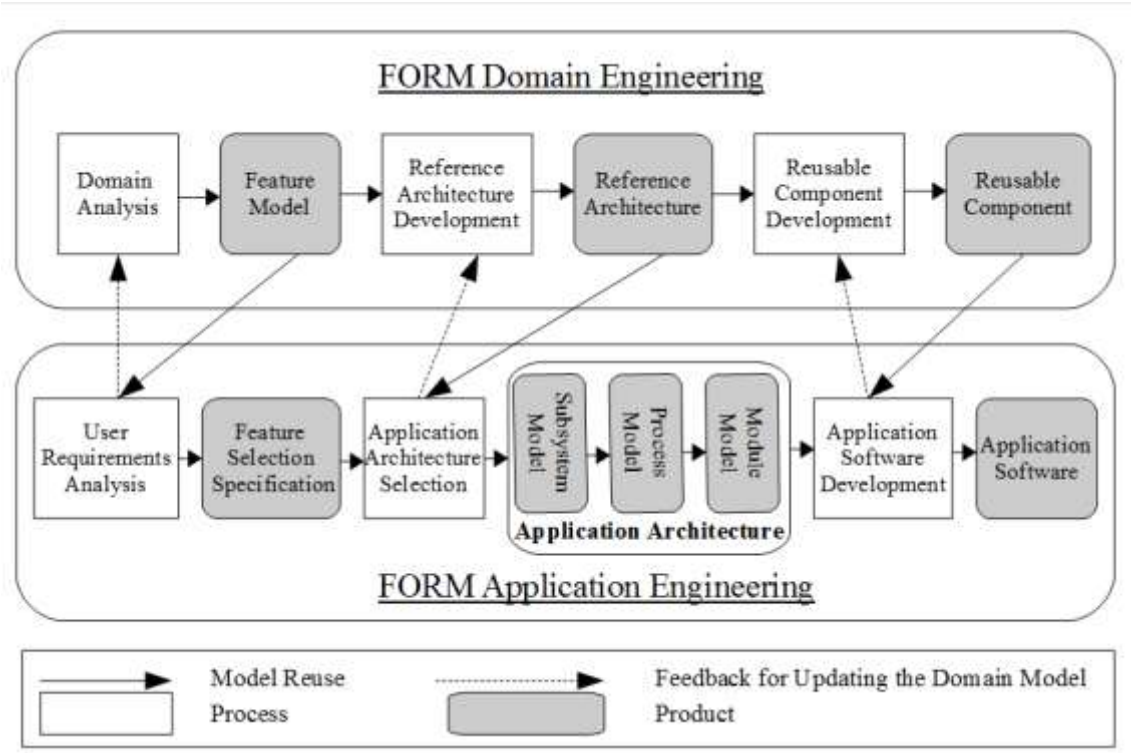
Organizacije koje samostalno razvijaju poslovne aplikacije za više poslovnih sektora najčešće koriste standardne gotove komponente postojećih referentnih arhitektura, međutim poslovni zahtjevi najčešće se ne mogu realizirati bez vlastitog razvoja koje u najmanju ruku služi za izolaciju raznih tuđih komponenata zbog mogućnosti njihove zastare ili nedovoljne funkcionalnosti. Stoga, predlažemo definiranje jedinstvene referentne arhitekture za poslovne aplikacije općenito koja predstavlja generalni koncept pomoću kojeg možemo riješiti zajedničke probleme poslovnih aplikacija na jednom mjestu, u okviru artefakata referentne arhitekture prema pristupu linija za proizvodnju softvera za poslovne aplikacije.

5.3.1. Funkcionalni i nefunkcionalni zahtjevi referentne arhitekture

Osim metode znanost o oblikovanju, kao krovne metode ovog istraživanja, u procesu razvoja artefakata referentne arhitekture koristimo i *Feature Oriented Reuse Method* (FORM) (Kang i ostali, 1998) metodu pomoću koje na sustavan način analiziramo i pronalazimo elemente ponovne upotrebe u obliku funkcionalnosti ili svojstava (engl. *feature*). Rezultate te analize koristimo za razvoj nove inačice artefakata referentne arhitekture (V4) i drugih komponenata linije za proizvodnju softvera koju istražujemo. U fazi analize definirali smo početni skup dviju grupa zahtjeva koje referentna arhitektura treba zadovoljiti: **nefunkcionalne** i **funkcionalne**.

Sličan proces smo koristili kod definiranje zahtjeva referentne arhitekture za nekoliko prethodnih inačica (V1,V2,V3) u razdoblju od 2 godine.

Iskustvo stečeno u praktičnoj primjeni referentne arhitekture pri razvoju poslovnih aplikacija koristili smo kako bi redefinirali postojeće zahtjeve. U istom razdoblju od 2 godine došlo je do značajnih promjena u tehnološkim potrebama korisnika poslovnih aplikacija temeljem kojih smo definirali potpuno nove zahtjeve za poboljšanje postojećih artefakata referentne arhitekture.



Slika 36 FORM metoda (Kang i ostali, 1998)

Tablica 7 prikazuje popis utvrđenih i predloženih nefunkcionalnih zahtjeva (ograničenja i osobina) referentne arhitekture koju razvijamo. Navedeni nefunkcionalni zahtjevi koji su definirani u fazi analize, rezultat su korisničkih zahtjeva u nekoliko prethodnih faza razvoja poslovnih aplikacija na temelju predložene referentne arhitekture (RABA) i njenih prethodnih inačica, analize tehnoloških mogućnosti sustava i vanjskih komponenata koje se direktno koriste u referentnoj arhitekturi, te analize postojećih sličnih referentnih arhitektura (*Spring, Scout, Vadiin, SmartGwt, Roma Framework, Openxava*).

Tablica 7 Nefunkcionalni zahtjevi referentne arhitekture

Broj	**Nefunkcionalni zahtjevi**
1	Podrška za razvoj aplikacija za više poslovnih sektora (domena)
2	Veliki broj korisnika aplikacija (istovremeno izvođenje velikog broja transakcija)
3	Istovremena podrška za više pozadinskih (engl. *backend*) sustava
4	Interoperabilnost sa više klijentskih platformi (operativni sustavi, kanali pristupa)
5	Visoka raspoloživost aplikacija (engl. *high availability*)
6	Razdvajanje aplikacija na klijent i server ili spajanje u jednu izvodivu cjelinu
7	Odzivno vrijeme za jednostavne transakcije ispod 3 sekunde
8	Konstantna raspoloživost sustava aplikacija (engl. *availability*)
8	Interoperabilnost sa drugim sustavima autentikacije (LDAP, Active Directory)
9	Održavljivost aplikacija na razini prosječnih kompetencija razvojnih programera
10	Dokumentiranost (izvorni kod, arhitektura, testiranje, korisnička dokumentacija)
11	Skalabilnost sustava aplikacija na razini pozadinskih sustava (engl. *scalability*)
12	Zaštita povjerljivosti podataka prema razini autorizacije korisnika
13	Fleksibilnost kod zamjene korištenih ili zastarjelih tehnoloških rješenja
14	Otvorenost i prilagodljivost za *open source* komponente i alate
15	Podrška za klijentske aplikacije koje koriste standardne *Web* protokole

Funkcionalni zahtjevi odnose se na podsustave referentne arhitekture **klijent** i **server**; zahtjevi referentne arhitekture za klijent uključuju: podršku za razvoj korisničkih sučelja prema predlošku *Model View Controller* (MVC), standardiziranu podlogu za aplikacije (engl. *desktop*), standardizirane predloške korisničkih sučelja (engl. *master detail, spredsheet, wizard, filter dataset, etc.*), formatiranje podataka za prikaz, podršku za slanje i primanje datoteka (engl. *file upload/download*), podršku za grafički prikaz podataka (engl. *charts*), itd.

Dio funkcionalnih zahtjeva implementira se u zajedničkom podsustavu (engl. ***common***), a odnose se primjerice na funkcionalne zahtjeve kao što su: obrada XML datoteka, čuvanje podataka (engl. *cache*), validacija poslovnih podataka, upravljanje poslovnim pravilima (engl. *rule engine*), upravljanje iznimkama (engl. *exceptions*), itd.

Tablica 8 prikazuje glavne funkcionalne zahtjeve (ponašanje sustava) referentne arhitekture za podsustav **server**. Pored navedenih zahtjeva, implementacija servera obuhvaća još nekoliko

manje značajnih funkcionalnih zahtjeva, primjerice: upravljanje obradama (engl. *jobs scheduler*), konfiguriranje aplikacija, itd.

Tablica 8 Funkcionalni zahtjevi referentne arhitekture za **server**

Broj	**Funkcionalni zahtjevi**
S1	Prijenos i čuvanje podataka u jedinstvenom obliku - *Data Value Objects*
S2	Smještanje glavnine poslovne logike u klase istoga tipa - *Business Objects*
S3	Smještanje logike pristupa podacima u klase istoga tipa - *Data Access Objects*
S4	Jedinstveno upravljanje zahtjevima sa klijenta - *Server Request Handlers*
S5	Jedinstveno upravljanje poslovnim transakcijama - *Remote Object Invokers*
S6	Uniformna reprezentacija svih izvora podataka - *Data Sources*
S7	Uniformno upravljanje sa spajanjem na izvore podataka - *Connection Pool*
S8	Priprema i prikazivanje izvještaja u više različitih formata - *Reporting Service*
S9	Konfiguracija sigurnosnih postavki - *Security Authorization and Authentication*
S10	Upravljanje transakcijama - *Transaction*
S11	Upravljanje sesijama - *Session*
S12	Pregledavanje podataka na strani klijenta u više koraka - *Value List Handler*

Funkcionalni i nefunkcionalni zahtjevi su uglavnom neovisni jedni od drugih, kao rezultat toga, možemo pretpostaviti da je moguće razviti sustav koji zadovoljava funkcionalne zahtjeve dok s druge strane sustav može ne zadovoljavati ograničenja i kvalitetu koji su definirani u obliku nefunkcionalnih zahtjeva. Budući da su nefunkcionalni zahtjevi vrlo kritični kod razvoja referentnih arhitektura, definirali smo ih na samom početku definiranja i razvoja najnovije inačice (V4) referentne arhitekture. Funkcionalne zahtjeve referentne arhitekture promatramo iz perspektive njihovih korisnika, razvojnih programera za poslovne aplikacije, za razliku od standardnog konteksta u kojem se funkcionalni zahtjevi najčešće promatraju iz perspektive krajnjih korisnika poslovnih aplikacija.

5.3.2. Početni skup funkcionalnih zahtjeva

Na temelju analize koju smo proveli u poglavlju 2 i 3 definirali smo početni skup funkcionalnih zahtjeva referentne arhitekture u svrhu traženja odgovora na pitanje (P1) „*Koje funkcionalne zahtjeve treba zadovoljiti referentna arhitektura za poslovne aplikacije prema pristupu linija*

za proizvodnju softvera“. Početni skup funkcionalnih zahtjeva upotpunili smo na temelju provedenog istraživanja u koje su bili uključeni stručnjaci s iskustvom u razvoju poslovnih aplikacija, definiranju referentnih arhitektura i korištenju linija za proizvodnju softvera.

Predstavljamo upitnik i rezultate ispitivanja koje smo proveli kako bi potvrdili njihovu valjanost te kako bi saznali više o relativnoj važnosti početnog skupa funkcionalnih zahtjeva referentne arhitekture.

Pregledom literature, analizom instrumenata koji mjere slične koncepte, razgovorom sa stručnjacima iz područja razvoja poslovnih aplikacija, te iskustvom autora, prikupili smo početni skup funkcionalnih karakteristika i sastavili tvrdnje za mjerenje stavova ispitanika prema važnosti pojedine funkcionalne karakteristike koju koristimo u upitniku.

Tablica 9 Funkcionalni zahtjevi – podloga za upitnik

Broj	**Funkcionalni zahtjevi**	**Opcije (varijabilnost)**
S1	*Data Value Objects*	*Single Value Objects, List of Value Objects, Meta data*
S2	*Business Objects*	POJO, EJB
S3	*Data Access Objects*	JDBC, JMS, CICS, iSeries, FTP, File, JSON, NoSQL
S4	*Server Request Handlers*	REST, HTTP, JMS, RMI, RPC
S5	*Remote Object Invokers*	AS2, EJB, JTA
S6	*Data Sources*	JDBC, JMS, CICS, iSeries, FTP, Hbase, MongoDB
S7	*Connection Pool*	JDBC, JMS, CICS, AS400 (iSeries), JMS
S8	*Reporting Service*	*Jasper, Excel, Word,* PDF
S9	*Security Authorization and Authentication*	LDAP, AD, RBAC
S10	*Transaction*	JTA, *Compensating Service Transaction,* AS2
S11	*Session*	RDBMS, *Cookie, Memory*
S12	*Value List Handler*	JDBC

Funkcionalne karakteristike implementiraju se u obliku komponenata okvira referentne arhitekture. Podjela na komponente subjektivnog je karaktera, međutim u razvoju poslovnih aplikacija prilično se ustalila podjela na komponente navedene u tablici 9 koju ovdje koristimo kao podlogu za upitnik kojeg smo uputili stručnjacima u području razvoja referentnih arhitektura.

5.3.3. Upitnik za istraživanje mišljenja stručnjaka

Sastavljanje upitnika i istraživanje mišljenja stručnjaka iz prakse proveli smo kako bismo povećali pouzdanost u primjerenost predloženog rješenja za referentnu arhitekturu poslovnih aplikacija. Glavni ciljevi korištenja ovog instrumenta istraživanja su pronalaženje odgovora na pitanje (P1), provjera važnosti svake od funkcionalnih karakteristika, razumijevanje njihovog relativnog značaja, pronalaženje dodatnih funkcionalnosti koje su eventualno izostavljene u početnom skupu funkcionalnih zahtjeva. Na temelju naše početne definicije funkcionalnih zahtjeva, paralelno sa provođenjem ovog istraživanja, radili smo na oblikovanju i razvoju najnovije inačice referentne arhitekture (V4). Prilagodbu tako razvijenih artefakata referentne arhitekture rezultatima ovog istraživanja planirali smo provesti u zadnjem ciklusu (iteraciji) razvoja. Ovakav iterativni postupak rezultat je primjene metode *znanost o oblikovanju* (engl. *design science*) koju primjenjujemo kao krovnu metodu u ovom radu.

5.3.3.1. Postupak ispitivanja

Sastavljanje upitnika

Pitanja u upitniku su uglavnom pitanja zatvorenog tipa - nude se mogući odgovori, dok je samo jedno od pitanja otvorenog tipa: „Koji su to dodatni funkcionalni zahtjevi koje smatrate važnima (engl. *what other capabilities do you regard as important*)“, na koje se očekuje jednostavan odgovor u obliku mišljenja ili prijedloga dodatnih funkcionalnosti koje bi trebala obuhvatiti predložena referentna arhitektura. Za pitanja koja mjere kvantitativnu varijablu, poslužili smo se oblikom zatvorenih pitanja sa najviše pet ponuđenih intenziteta.

(Pitanje 1)

Koliko godina radnog iskustva imate u slijedećim područjima?

a) Razvoju poslovnih aplikacija

b) Oblikovanju, razvoju i održavanju softverske arhitekture

c) Korištenju pristupa linija za proizvodnju softvera

Namjena prvog pitanja je bila dobivanje općeg dojma o iskustvu ispitanika u tri relevantna područja. Glavni cilj ovog pitanja je bilo definiranje granica broja godina relevantnog iskustva koje će se uzeti u obzir. Naša pretpostavka je da je iskustvo u razvoju poslovnih aplikacija potrebno za razumijevanje glavnih funkcionalnih zahtjeva referentne arhitekture, te da bi

ispitanici trebali imati iskustvo u oblikovanju, razvoju i održavanju softverske arhitekture. Granice potrebnog broja godina iskustva definirali smo na osnovi provedenog istraživanja: samo upitnici s odgovorima ispitanika s najmanje **tri** godine iskustva u razvoju poslovnih aplikacija te s najmanje **jednom** godinom iskustva u oblikovanju, razvoju i održavanju softverske arhitekture, uzeti će se u obzir i analizirati. Iskustvo u korištenju linija za proizvodnju softvera je također relevantno ali nije i nužno za razumijevanje potrebnih funkcionalnih zahtjeva.

(Pitanje 2)

Ocijenite koliku važnost dajete svakoj od funkcionalnosti referentne arhitekture? (engl. *how important do you rate the following capabilities of reference architecture*?)

Svrha drugog pitanja je bila provjera važnosti svake od predloženih funkcionalnih karakteristika, te razumijevanje njihovog relativnog značaja. Kako bi pojednostavnili upitnik, prvo smo objasnili kontekst artefakta (okvira referentne arhitekture) kojeg razvijamo, te smo uz pomoć dijagrama svojstava (engl. *feature diagram*) i prikazom korištenih uzoraka oblikovanja (engl. *design patterns*) opisali svaki od funkcionalnih zahtjeva.

Tablica 10 Upitnik za ocjenu važnosti karakteristika referentne arhitekture

-2	-1	1	2	*Koliku važnost dajete svakoj od funkcionalnosti referentne arhitekture?*
				Data Value Objects
				Business Objects
				Data Access Objects
				Server Request Handlers
				Remote Object Invokers
				Data Sources
				Connection Pool
				Reporting Service
				Security Authorization and Authentication
				Transaction
				Session
				Value List Handler

Stav prema važnosti pojedine funkcionalnosti (varijable koju mjerimo), klasificiran je u dvosmjernoj kategoriji (određen je pozitivno ili negativno). Intenzitet stavova prema predloženim funkcionalnostima referentne arhitekture mjerimo ocjenama na ljestvici stavova: -2 (totalno nebitno), -1 (nebitno), 1 (važno), 2 (jako važno).

(Pitanje 3)

Koji su to dodatni funkcionalni zahtjevi koje smatrate važnima?
(engl. *what other capabilites do you regard as important*?)

Svrha ovog pitanja je bilo pronalaženje dodatnih funkcionalnih karakteristika koje još nisu bile uzete u obzir u pitanju broj dva.

(Pitanje 4)

Slažete li se da postoji potreba da se definira posebna domena linija za proizvodnju softvera za poslovne aplikacije općenito?
(engl. *do you agree that there is a need to define a software product line domain for business applications in general*?)

Ispitanici

Uzorak ispitanika je izabran na temelju analize javno dostupnih profila korisničkih grupa na trenutno najznačajnijoj društvenoj mreži *LinkedIn* koja, među ostalima, okuplja i stručnjake iz područja istraživanja ovog rada. Kod pretraživanja korisničkih grupa za analizu, koristili smo ključne riječi „*software product line(s)*“, „*software architecture*“, „*information systems*“ i „*business applications*“. Osim pretraživanja po ključnim riječima, korisničke grupe smo pretraživali prema članstvu najznačajnijih autora i stručnjaka iz ovog područja (*Paul Clements, Jan Bosch, Linda Northrop, Frank van der Linden, Klaus Schmid, Erich Gamma*).

Nakon detaljne analize njihovih profila odabrali smo pet korisničkih grupa:

- *A - Software Product Line Conference*
- *B - Software Product Lines*
- *C - Software Design Patterns And Architecture*
- *D - Software Architecture / Techpost Media*
- *E - Software Architecture*

Ovaj ne probabilistički uzorak svrstava se u model "uzoraka prema prosudbi stručnjaka" (engl. *expert sample*) te se preporuča ukoliko se anketa treba provesti među stručnjacima iz neke uske specijalnosti (Žugaj, Dumičić, & Dušak, 2006, str. 176).

Pripremljeni upitnik smo uz pomoć javno dostupnog servisa (engl. *docs.google.com*) proslijedili ispitanicima preko portala društvene mreže *LinkedIn*.

5.3.3.2. Analiza rezultata istraživanja

Povratni rezultati

Na proslijeđeni upitnik odgovorio je vrlo mali broj od 5 članova, što je neočekivano mali broj. Razlog malom broju odgovora na upitnik vjerojatno leži u nemogućnosti davanja ispravnih odgovora prije prethodnog proučavanja konteksta primjene predloženih funkcionalnosti što zahtjeva dodatno vrijeme i interes ispitanika. Međutim, smatramo da to značajno ne utječe na vjerodostojnost istraživanja iz razloga što su odgovori u svim upitnicima podudaraju i stoga što je primarna svrha upitnika bila potvrda predloženih funkcionalnih zahtjeva a ne njihovo prikupljanje. Na temelju granice potrebnog broja godina iskustva koju smo prethodno definirali, u obzir smo uzeli svih 5 popunjenih upitnika. Tablica 11 prikazuje rezultate odgovora na pitanja: 1, 2 i 4.

Rezultati odgovora na pitanje 1.

Prosječno iskustvo ispitanika u različitim relevantnim područjima kreće se od preko 6 do preko 19 godina, što ukazuje na to da ispitanici imaju dovoljno iskustva u razvoju poslovnih aplikacija i definiranju arhitekture, da njihova mišljenja možemo sa pouzdanjem uzeti u obzir.

Rezultati odgovora na pitanje 2.

Odgovori na drugo pitanje ukazuju na to da su svi funkcionalni zahtjevi rangirani kao važni, prosječna ocjena je bila **1,25**. Budući da se stavovi prema važnosti pojedinih funkcionalnosti klasificiraju u dvosmjernoj kategoriji (pozitivno ili negativno), te da je relevantnost svih funkcionalnih zahtjeva koje mjerimo ocjenama na ljestvici stavova od -2 (totalno nebitno) do 2 (jako važno) ocjenjena prosječnom **pozitivnom** ocjenom (iznad nule), sve predložene funkcionalnosti smatramo relevantnima.

Relativna važnost

Samo jedan član rangirao neki of funkcionalnih zahtjeva (*Reporting Service*) kao -2 (totalno nebitno). Neki od funkcionalnih zahtjeva ocijenjeni su kao važniji u usporedbi s drugima. Šest

je funkcionalnih zahtjeva koje su neki od članova rangirali kao nebitan (-1). Kao najvažniji zahtjevi rangirani su *Data Access Object* i *Server Request Handler*, prosječna ocjena 1,8. Najmanje važan zahtjev je *Value List Handler*, ocjena 0,4.

Tablica 11 Rezultati odgovora na pitanja: 1, 2 i 4

	Odgovori na pitanja članova korisničkih grupa					
Pitanja	***Član 1***	***Član 2***	***Član 3***	***Član 4***	***Član 5***	***Prosjek***
1(a)	20	42	8	4	25	19,8
1(b)	7	22	4	4	15	10,4
1(c)	15	14	2	0	3	6,8
2(1)	1	2	-1	2	2	1,2
2(2)	1	2	1	2	1	1,4
2(3)	1	2	2	2	2	**1,8**
2(4)	1	2	2	2	2	**1,8**
2(5)	1	2	1	2	1	1,4
2(6)	2	2	-1	2	1	1,2
2(7)	1	2	-1	2	1	1,0
2(8)	2	-1	-2	2	2	0,6
2(9)	2	1	2	2	1	1,6
2(10)	2	2	-1	2	1	1,2
2(11)	1	2	1	2	1	1,4
2(12)	1	-1	-1	2	1	0,4
4	2	5	5	5	5	4,4

Rezultati odgovora na pitanje 3.

Na pitanje je odgovoreno sa 4 nove potencijalne funkcionalnosti (**SPA, Lambda, Identity & Access Management, JSON**) koje bi referentna arhitektura za poslovne aplikacije trebala podržavati. **Lambda** je tehnika prenošenja dijela programskog koda u cilju kasnijeg izvođenja, predstavlja novu funkcionalnost *Java 1.8* platforme; u ovom istraživanju koristimo *Java 1.7* platformu, stoga nismo bili u mogućnosti koristiti tu tehniku kako bi poboljšali neke od karakteristika referentne arhitekture. Single Page Applications (**SPA**) tehnika odnosi na strategiju prihvata i izvođenja korisničkih sučelja u *Web* aplikacijama; korisničko sučelje se

odjednom prihvati u pregledniku, prezentacijska logika izvodi se samo u pregledniku, neki standardni okviri (Angular, Durandal, and Ember) ugradili su ovu funkcionalnost; u ovom istraživanju koristimo GWT okvir za razvoj korisničkih sučelja koji trenutno ne podržava ovu funkcionalnost, stoga je nismo uzeli u obzir. **Identity & Access Management** odnosi se na funkcionalnost koju smo djelomično implementirali kao *Security* (RBAC) komponentu, jedan dio funkcionalnosti za upravljanje pristupom resursima aplikacija planiramo implementirati u novoj inačici artefakata, kada za to bude postojala poslovna potreba.

Od predloženih funkcionalnosti u obzir smo uzeli i implementirali samo **JSON (JavaScript Object Notation)**, jezično neutralni format za prijenos podataka; funkcionalnost smo ugradili u komponente koje koriste *Client Request Handler* i *Server Request Handler* komponente, prije slanja i nakon prihvata podataka od klijenta prema poslužitelju.

Rezultati odgovora na pitanje 4.

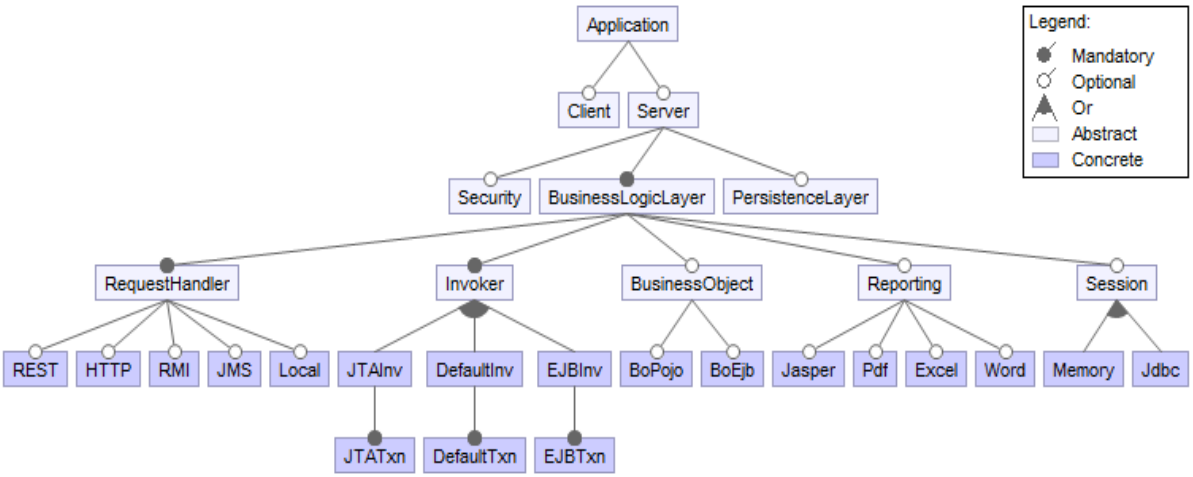
Prosječna ocjena odgovora na pitanje „*Da li se slažete da je potrebno definirati domenu za poslovne aplikacije općenito*“ bila je **4,4** (mogući odgovori od 1 do 5). Ispitanici smatraju da je definiranje domene za razvoj poslovnih aplikacija na općenitoj (horizontalnoj) razini prema pristupu linija za proizvodnju softvera **potrebno**. Ovaj pristup se uspješno koristi u velikom broju drugih sektora; mali uzorak stručnjaka koje smo ispitali i njihovo mišljenje, predstavlja još jednu potvrdu relevantnosti ovog istraživanja koje se odnosi na primjenu pristupa linija za proizvodnju softvera u praksi razvoja i održavanja u sektoru poslovnih aplikacija.

5.3.4. Konačni funkcionalni zahtjevi referentne arhitekture

Na temelju inicijalnog skupa funkcionalnih zahtjeva koje smo predložili istraživanjem literature i postojećih referentnih arhitektura, iskustva autora, te na temelju rezultata ispitivanja stručnjaka iz područja arhitektura, definirali smo konačni skup funkcionalnih zahtjeva za referentnu arhitekturu poslovnih aplikacija. Glavni artefakt referentne arhitekture, aplikacijski okvir, podijelili smo na tri podsustava (engl. *client, server, common*), sam podsustav **server** i njegove funkcionalnosti također smo podijelili na dva sloja, sloj poslovne logike i sloj pristupa podacima. Slika 37 ilustrira funkcijski dijagram sloja poslovne logike koji se uobičajeno koristi za prikaz funkcionalnih karakteristika sustava u kontekstu linija za proizvodnju softvera.

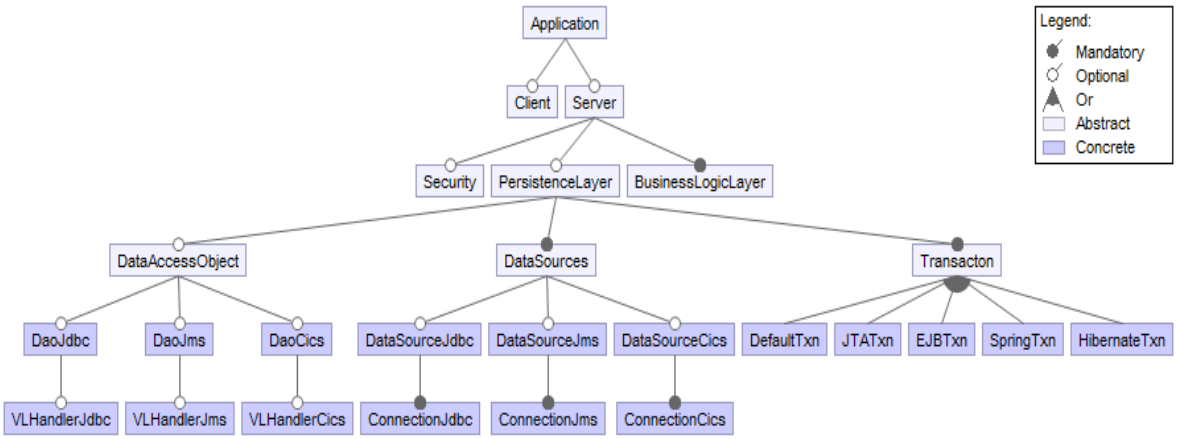
Slika 38 ilustrira funkcijski dijagram sloja pristupa podacima, dio je funkcionalnosti koje ima aplikacijski poslužitelj neovisno o poslovnom sektoru za koji se aplikacije razvijaju.

Ostale funkcionalne karakteristike, osim funkcija servera, primjerice funkcije prezentacijskog sloja referentne arhitekture, nisu dio detaljne analize u ovom radu. Njihov razvoj i primjena u praksi su također predmet ovog istraživanja, međutim, glavni predmet detaljnijeg istraživanja, primjene i korištenja u fazi validacije u ovom radu su funkcije servera, njihova primjena u razvoju poslovnih aplikacija, utjecaj njihovih atributa kvalitete na održavljivost aplikacijskih poslovnih komponenata, itd.



Slika 37 Dijagram funkcionalnosti za sloj poslovne logike

Kontekst razvoja poslovnih aplikacija prema pristupu linija za proizvodnju softvera, podrazumijeva upravljanje varijabilnošću raspoloživih funkcija. Funkcijski dijagrami služe i za prikaz varijabilnih točaka koje koristimo za izbor pojedinih funkcionalnosti kod razvoja svake od poslovnih aplikacija. Dijagram jasno prikazuje funkcije koje su obavezne, opcionalne ili alternativne. Mehanizam realizacije varijabilnosti funkcija aplikacijskog okvira koji koristimo u primjeni su konfiguracijski parametri pomoću kojih definiramo izbor i osnovne parametre funkcija koje se zahtijevaju bilo od strane korisnika, bilo zbog tehničkih ograničenja.



Slika 38 Dijagram funkcionalnosti za sloj pristupa podacima

5.3.5. Predloženi artefakti referentne arhitekture

Aplikacijski okvir sa svojim podsustavima predstavlja temeljni artefakt predložene referentne arhitekture. Jedna od najčešćih i najuspješnijih tehnika za implementaciju arhitekture linija za proizvodnju softvera je korištenje programskih okvira (engl. *framework*) (Bosch, 2000; Fayad, Schmidt, & Johnson, 1999; Johnson & Foote, 1988). Programski okvir se sastoji od klasa koje čine kostur buduće aplikacije; sadrži točke varijabilnosti koje se konfiguriraju za svaku aplikaciju posebno. Jedan od glavnih ciljeva razvoja aplikacijskog okvira je pojednostavljenje razvoja poslovnih aplikacija. Aplikacijski programeri koji koriste okvir za razvoj poslovnih aplikacija ne moraju proučavati detalje tehnologija koje okvir koristi za specifičnu svrhu kao što je serijalizacija, komunikacija, refleksija (engl. *reflection*). Navedeni detalji su ugrađeni u okvir na način da se aplikacijski programeri koji koriste okvir mogu skoro isključivo baviti programiranjem poslovne logike i oblikovanjem poslovnih aplikacija.

Predloženi koncept je pragmatičan i pojednostavljen pristup razvoju poslovnih aplikacija, a temelji se na tri poznate ideje: *Software Product Line Reference Architecture*, *Model-Driven Development* (MDD), *Pass-Through Business Objects Components*.

MDD koristimo za generiranje osnovne strukture sučelja poslovnih komponenata i klasa za čuvanje i prijenos poslovnih podataka na temelju poslovnog modela glavnih entiteta neke aplikacije ili poslovne komponente. Izmjene generiranih artefakata potrebno je provoditi isključivo kroz izmjene u poslovnim modelima.

Navedena ideja *Pass-Through Business Objects Components* odnosi se na koncept prema kojemu objekti poslovne logike ne sadrže poslovne podatke (osim u iznimnim slučajevima) niti su povezani direktno sa izvorima podataka (engl. *object-relational mapping*), već isključivo služe za programiranje poslovne logike na temelju podataka koje su dobili sa klijenta ili iz nekoga drugog izvora podataka.

Osim okvira referentne arhitekture, predlažemo definiranje još tri artefakta: referentnu aplikaciju (engl. *reference application*) koja se koristi kao predložak za razvoj novih aplikacija, pomoćni alat (engl. *plug-in*) za Eclipse razvojno okruženje kojeg koristimo za povećanje produktivnosti razvojnih programera za poslovne aplikacije, i opis arhitekture prema ISO/IEC/IEEE 42010 standardu.

5.4. Razvoj artefakata referentne arhitekture (engl. *development*)

Odgovor na istraživačko pitanje P2 („*Može li se razviti referentna arhitektura koja zadovoljava definirane zahtjeve?*“) uključuje oblikovanje i razvoj programskog okvira (engl. *framework*) i drugih artefakata prema definiranim zahtjevima na temelju definiranih odgovora na istraživačko pitanja (P1).

Apstrahiranje koncepta (engl. *Abstracting Concept*)

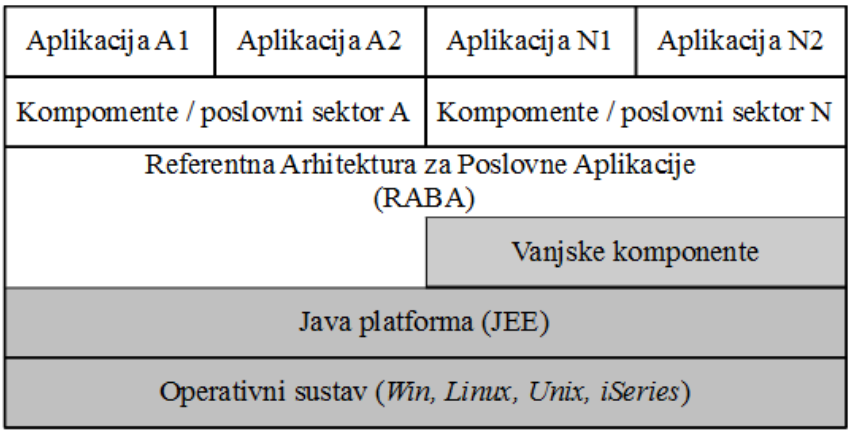
Poslovne aplikacije unutar neke organizacije uglavnom upotrebljavaju standardizirane *inter-organizacijske* komponente: operativne sustave, baze podataka, uredske aplikacije, *web* preglednike, programske platforme; njihova ponovna upotreba je vrlo rasprostranjena, dok je ponovna upotreba *intra-organizacijskih* komponenata razvijenih u samoj organizaciji koja razvija poslovne aplikacije vrlo rijetka.

Za razvoj poslovnih aplikacije najčešće se koriste dvije standardne tehnološke platforme: **.***NET* i *Java EE*. Navedene platforme su temelji na kojima su se razvijali razni pomoćni alati, gotove komponente i ostali artefakti potrebni za razvoj poslovnih aplikacija. Organizacije se uglavnom odlučuju za jednu od navedenih platformi, te ovisno o platformi, izabiru pomoćne alate i gotove komponente; ovisno o poslovnom sektoru kojem pripadaju, organizacije razvijaju referentne arhitekture i aplikacije. Međutim, poslovne aplikacije i njihove referentne arhitekture u raznim sektorima za koje se aplikacije razvijaju imaju veliki broj zajedničkih karakteristika koje se mogu implementirati u jedinstvenu i ponovno upotrebljivu referentnu arhitekturu. Predloženu referentnu arhitekturu u ovom radu temeljimo na iskustvu i rješenjima koje smo primijenili u nekoliko poslovnih sektora, bankarstvo, telekomunikacije, osiguranje, itd.

U ovom radu odlučili smo se koristiti *Java EE* platformu koja je starija i zrelija od *.NET* platforme, ima više raspoloživih gotovih komponenata i besplatnih alata za korištenje, neovisna je od operativnog sustava na kojem se koristi za razliku od *.Net* platforme koja se uglavnom može koristiti isključivo na *Windows* operativnim sustavima.

Hijerarhijski dizajn (engl. *Hierarchical Design*)

Poslovne aplikacije po svojoj prirodi mogu biti vrlo jednostavne ali i vrlo kompleksne. Razvoj i održavanje kompleksnih aplikacija moguće je pojednostavniti tako da se njihova struktura organizira prema poznatoj strategiji „podijeli pa vladaj“ (engl. „*divide and conquer*“). Predloženi sustav podjele poslovnih aplikacija odnosi se na situaciju kada se aplikacije razvijaju za više poslovnih sektora, na temelju iste referentne arhitekture, korištenju vanjskih komponenata drugih proizvođača, na jedinstvenoj platformi koja se izvodi na više operativnih sustava. Slika 39 ilustrira predloženu hijerarhiju odnosa između najvažnijih dijelova poslovnih aplikacija koje koriste istu referentnu arhitekturu.



Slika 39 Predložena hijerarhijska struktura poslovnih aplikacija

Podjela sustava na podsustave temelji se na principu koji podrazumijeva da je interakcija među podsustavima slabijeg intenziteta nego što su interakcije komponenata unutar samih podsustava. Prema predlošku metode znanost o oblikovanju (*Hierarchical Design*) kojeg primjenjujemo, podsustave smo oblikovali prema slijedećem postupku:

1. Podijelili smo sustav na tri glavne cjeline (referentna arhitektura, poslovne komponente, aplikacije). Svaki od podsustava je značajno manji od originalnog sustava, poslovnih aplikacija koje uključuju sva tri navedena podsustava.

2. Za svaki podsustav provjerili smo da li već postoji slično oblikovanje koje se može iskoristiti (npr. uzorak dizajna), te ga iskoristili ako postoji.
3. Ako se bilo koji podsustav može oblikovati bez daljnjeg razdvajanja na manje dijelove, oblikovali smo ga; inače, krenuli smo od prvog koraka u ovom postupku za oblikovanje podsustava.
4. Oblikovali smo interakcije među podsustavima tako da sustav u cjelini zadovoljava postavljene kriterije.

Kod razdvajanja sustava na podsustave prema navedenom postupku nastojali smo smanjiti broj poveznica, interakcija i broj ovisnosti među podsustavima. Primjenom navedenog postupka i principa, oblikovali smo arhitekturu poslovnih aplikacija koja se sastoji od podsustava koji su organizirani na hijerarhijski način, tako da je svaki podsustav dovoljno neovisan o drugim podsustavima. Glavna prednost korištenja ovog predloška odnosi se na značajno smanjivanje kompleksnosti oblikovanja poslovnih aplikacija.

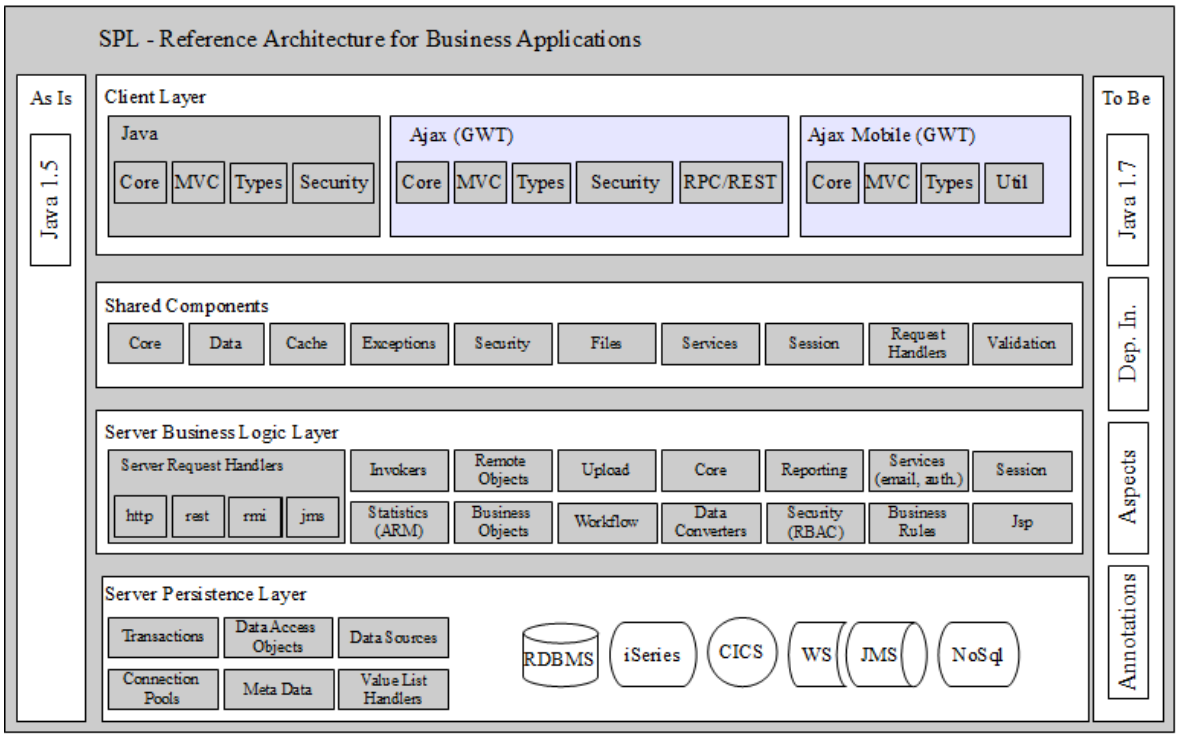
5.4.1. Prepravljanje postojeće inačice RABA okvira (engl. *refactoring*)

Postupak kojeg koristimo u ovom istraživanju odvija se u ponavljajućim ciklusima, stoga smo i prije definiranja funkcionalnih zahtjeva najnovije inačice referentne arhitekture (V4) proveli prepravljanje (engl. *refactoring*) njenih postojećih artefakata (V3) na temelju analize metrika izvornog koda i na temelju procjene rizika tehnološkog zastarijevanja. Prepravljanje postojeće inačice aplikacijskog okvira i aplikacijskih komponenata koje ga koriste bilo je prvenstveno nužno kako bi se umanjili rizici zastarijevanja postojećih rješenja u odnosu na nove inačice internet preglednika, operativnih sustava, *Java EE* platforme, razvojnih alata, vanjskih komponenata, baza podataka. Osim navedenog umanjivanja rizika, prepravljanje je imalo za cilj i iskorištavanje dodatnih mogućnosti koje pružaju nove inačice navedenih tehnoloških elemenata u svrhu poboljšanja atributa kvalitete sustava u cjelini. Prema definiciji (Fowler, 1999), prepravljanje predstavlja proces izmjena postojećeg sustava na način da se ne mijenja njegovo vanjsko ponašanje dok s druge strane doprinosi poboljšanju unutrašnje strukture sustava. Međutim, u kontekstu linija za proizvodnju softvera, ova definicija nije potpuno prikladna jer ne uzima u obzir veći broj aplikacija i njihov odnos sa zajedničkom referentnom arhitekturom. Proširena definicija prepravljanja koja vodi računa o kontekstu linija za proizvodnju softvera (Alves i ostali, 2006) odnosi se samo na prepravljanje modela funkcionalnosti ili svojstava (engl. *feature models*) i ne obuhvaća druge specifičnosti linija za

proizvodnju softvera, stoga u ovom radu koristimo prošireni pojam prepravljanja; „promjene u strukturi linija za proizvodnju softvera kako bi se poboljšale mogućnosti konfiguriranja, olakšalo razumijevanje i umanjio napor potreban za provođenje izmjena, bez vidljivih izmjena u ponašanju aplikacija“. Prepravljanje zahtjeva izmjene u malim koracima, detaljan test i dokumentiranje izmjena kako bi se sačuvala postojeća funkcionalnost i osiguralo se od mogućih novih grešaka.

Empirijska dorada (engl. *Empirical Refinement*)

Proces prepravljanja u ovom radu odnosi se na artefakte referentne arhitekture: okvir referentne arhitekture, pomoćni alat (Eclipse plug-in) i 9 poslovnih aplikacija. Slika 40 prikazuje arhitekturu okvira referentne arhitekture koji se sastoji od tri podsustava: klijent (engl. *client*) koji se koristi za razvoj prezentacijskog sloja poslovnih aplikacija, zajednički (engl. *shared, common*) za razvoj svih dijelova aplikacija, i server (engl. *server)* za razvoj poslovne logike i pristupa podacima. Glavne aktivnosti prepravljanja odnose se na migraciju artefakata sa postojeće inačice *Java 1.5* na novu *Java 1.7* inačicu, uvođenje Ajax (GWT) platforme za razvoj mobilnih i *web* aplikacija te primjenu tehnika kao što su anotacije (engl. *annotations*) i aspekti (engl. *aspects*).



Slika 40 Ciljana referentna arhitektura

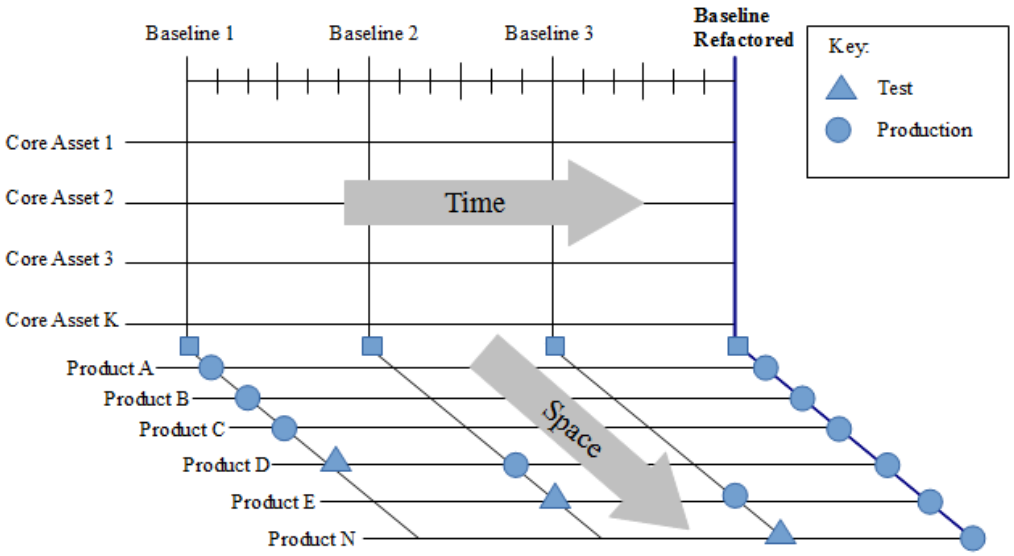
Način prepravljanja postojećeg sustava za koji smo se odlučili ubraja se u kategoriju preventivnih (engl. *preventive*) i prilagodnih (engl. *adaptive*) izmjena prema (Seacord, Plakosh, & Lewis, 2003). Prilagodne izmjene bile su potrebne zbog uvođenja novih inačica programskih jezika (npr. *Java 1.7*), operativnih sustava, razvojnih alata, baza podataka, vanjskih komponenata, itd. Preventivne izmjene imale su za cilj poboljšanje održavljivosti i pouzdanosti komponenata sustava. Za razliku od prilagodnih izmjena, preventivne izmjene pro aktivno utječu na poboljšanje atributa kvalitete referentne arhitekture i aplikacija koje ju koriste. Glavni naglasak tijekom prepravljanja bio je na poboljšanju atributa promjenjivosti (engl. *changeability*) i stabilnosti (engl. *stability*) referentne arhitekture. Promjenjivost se odnosi na utjecaj izmjena komponenata referentne arhitekture na druge ovisne komponente te na aplikacije koje ju koriste. Broj ovisnosti o vanjskim komponentama u tipičnom okruženju u kojem se poslovne aplikacije koriste, s vremenom se povećava, što utječe na održavljivost i stabilnost sustava poslovnih aplikacija. Promjene koje se nužno tijekom vremena događaju na vanjskim komponentama možemo tumačiti kao potencijalno smanjivanje stabilnosti i održavljivosti sustava. Stoga, smatramo korisnim te promjene amortizirati uz pomoć referentne arhitekture, tako da te promjene ne utječu direktno na poslovne aplikacije. Proces prepravljanja uključivao je artefakte referentne arhitekture, poslovne komponente, poslovne aplikacije. Veliki broj promjena u tijeku ovog procesa te njihova sinkronizacija sa ostatkom sustava, provode se s ciljem poboljšanja sustava, stoga je posebno važno promjene uvoditi planski, sustavno i postepeno. Na slici 41 djelomično preuzetoj iz (Krueger, 2010, str. 9) prikazane su tri inačice sustava prije prepravljanja, i inačica V4 (engl. *baseline*) koja je rezultat provedenog prepravljanja.

Osim sinkronizacije glavnih inačica (V1,V2,V3,V4) sa ostatkom sustava koje nastaju uglavnom jednom godišnje, proces sinkronizacije referentne arhitekture i ostatka sustava uključuje i njene mjesečne inačice. Prepravljanje inačica predstavlja planiranu „to-be“ inačicu ciljane referentne arhitekture i svih drugih komponenata i artefakata cijelog sustava.

U procesu prepravljanja i razvoja najnovije inačice sudjelovao je tim u kojem su najvažnije bile slijedeće uloge (neki članovi tima obavljali su više od jedne uloge):

- Tehnički arhitekt (engl. *technical architect*)
- Voditelj projekta (engl. *project manager*)
- Poslovni analitičar (engl. *business analyst*)
- Programer korisničkih sučelja (engl. *presentation-tier developer*)

- Programer poslovne logike (engl. *business logic developer*)
- Tester



Slika 41 Inačice SPL sustava (Krueger, 2010)

Proces prepravljanja postojećeg sustava odvijao se u nekoliko iterativnih koraka primjenom agilne *Feature Driven Development* (FDD) metode i standardnih procesnih koraka:

- Analiza
- Dizajn
- Implementacija
- Test

5.4.1.1. Analiza

1) Analiza funkcionalnosti referentne arhitekture i aplikacija

Analizom smo identificirali 25 funkcija na serveru i 26 funkcija na klijentu referentne arhitekture, 49 poslovnih komponenata na serveru i 60 na klijentu koje su se koristile za razvoj 9 poslovnih aplikacija. Za prepravljanje smo odabrali samo funkcije koje se odnose na server dok smo prepravljanje ostalih funkcija koje se odnose na prezentacijski

sloj odgodili za vrijeme nakon što se prema planu u inačici V4 završi novi dio okvira referentne arhitekture za podršku razvoja *web* aplikacija. Analiza obuhvaća pronalaženje funkcija i točaka varijabilnosti koje se ne koriste u sustavu.

2) *Analiza povezanosti komponenata pomoću Dependency Structure Matrix* (DSM)

DSM tehniku za analizu strukturne povezanosti (engl. *dependency*) komponenata ili podsustava koristili smo za analizu dvaju podsustava referentne arhitekture (*Common* i *Server*) sa svrhom poboljšanja modularnosti i adaptabilnosti razvojnog okvira, te za pronalaženje interakcija koje mogu biti eventualna prepreka u procesu njihovog korištenja pri razvoju poslovnih aplikacija. Podsustav za razvoj prezentacijskog sloja (*Client*) nismo analizirali budući da postojeću inačicu (V3) za razvoj *Java SWT* korisničkih sučelja nismo planirali mijenjati, već smo se odlučiti za razvoj novih podsustava za *web* i mobilne aplikacije. Povezanost komponenata sustava može biti statička (za vrijeme kompiliranja) ili dinamička (za vrijeme izvođenja). Za analizu statičke povezanosti koristili smo alat Cambridge Advance Modeler (2010) i JArchitect V4.0.0 © 2013 CodeGears dok smo analizu dinamičke povezanosti (npr. *Java reflection*) koja također postoji u sustavu radili manualno.

Slika 42 prikazuje rezultat statičke analize povezanosti komponenata podsustava *Common* (V3) okvira referentne arhitekture. Prazna ćelija ukazuje na to da ne postoji povezanost između komponente koja se nalazi na horizontalnoj liniji i komponente koja se nalazi na vertikalnoj liniji. Ostale ćelije, crna, zelena i plava imaju slijedeća značenja:

Zelena ćelija označava broj klasa u vertikalnoj liniji koje koristi komponenta u horizontalnoj liniji.

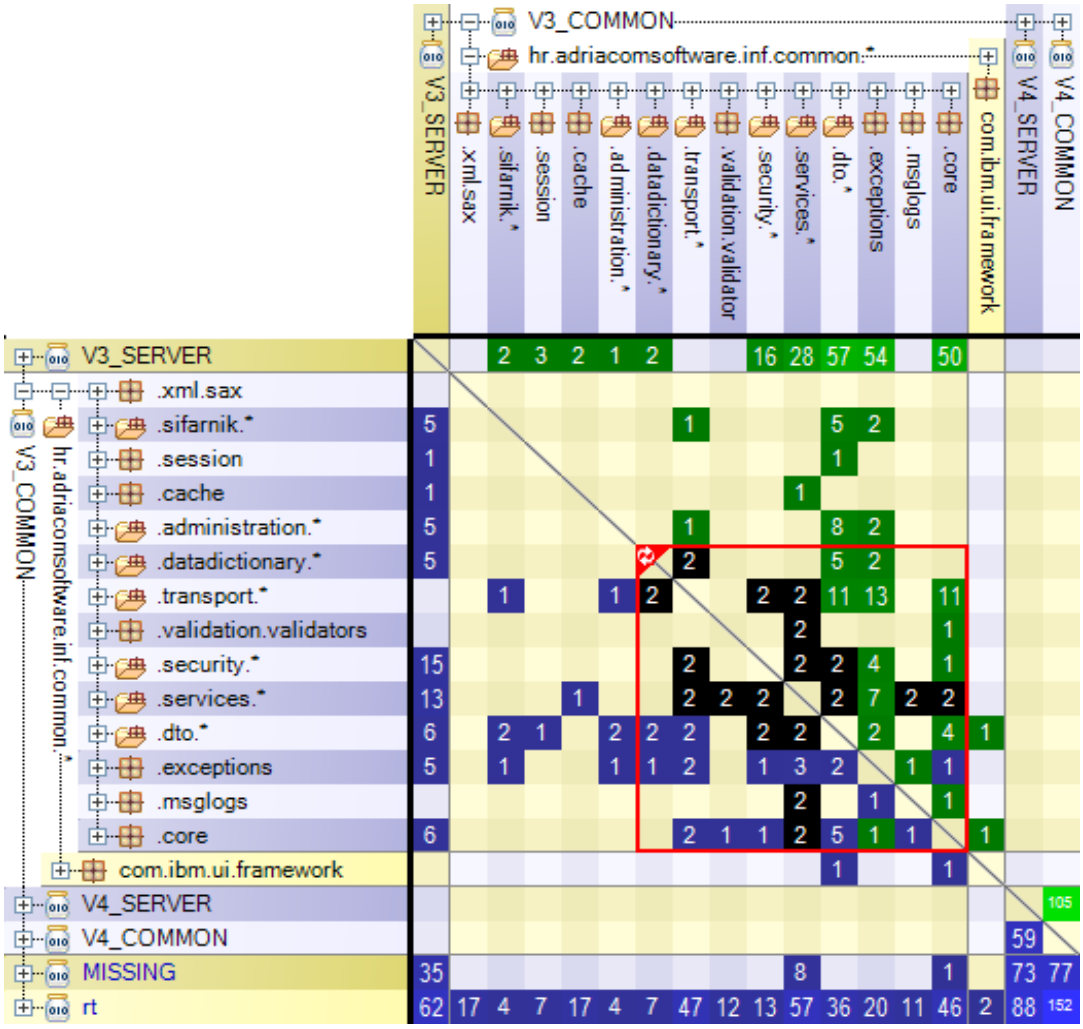
Plava ćelija označava broj klasa u horizontalnoj liniji koje su korištene od strane klasa komponente u vertikalnoj liniji.

Crna ćelija označava broj međusobno ovisnih klasa između komponente u vertikalnoj i komponente u horizontalnoj liniji.

Crveni okvir označava kružnu (engl. *cycle*) povezanosti komponenata, koja se može sastojati od zelenih, plavih ili crnih (direktna povezanost) ćelija.

DSM tehnika omogućava nam analizu kvalitete upotrebe pojedinih tehnika i stilova softverske arhitekture kod njihovog oblikovanja, primjerice primjene slojeva (engl. *layers*). Potpuna slojevitost arhitekture podrazumijeva da su sve plave ćelije ispod crte

dijagonale a sve zelene iznad crte dijagonale. Također, broj crnih ćelija koje ukazuju na broj međusobno ovisnih komponenata, ukazuje na nepravilno oblikovane komponente koje je potrebno prepraviti. Broj međusobno povezanih komponenata je **9**, što možemo protumačiti kao nužnu potrebu prepravljanja tih komponenata, dok su ostale ćelije prilično dobro raspoređene u odnosu na dijagonalnu crtu. Cilj nam je postići značajniju povezanost referentne arhitekture sa vanjskim komponentama i sa *Java* okružjem kako bi umanjili povezanosti poslovnih komponenata koje primjenjuju referentnu arhitekturu sa vanjskim komponentama i *Java* okružjem.



Slika 42 DSM podsustav *Common* (V3)

Slika 43 prikazuje rezultat statičke analize povezanosti komponenata podsustava *Server* (V3) okvira referentne arhitekture. Broj kombinacija međusobno povezanih komponenata je **3**, potrebno ih je prepraviti u novoj inačici V4. Broj komponenata u

podsustavu je **16**, za očekivati je da će taj broj narasti u novoj inačici V4, stoga je za usporedbu DSM strukture međuovisnosti moguće promatrati i relativno, u odnosu na broj komponenata.

*3) Analiza vanjskih (*engl. *third-party) komponenata*

Vanjske komponente koje se koriste u sustavu analizirali smo s obzirom na njihovu stabilnost, primjerenost, pouzdanost i rizik zastarijevanja. Najveći broj komponenata je potrebno osvježiti sa njihovom novom stabilnom inačicom, dok smo za jednu komponentu utvrdili da je proizvođač više ne podržava (IBM AUIML za razvoj korisničkih sučelja). Slika 42 i slika 43 također prikazuju broj poveznica između podsustava *Common* (V3) i između podsustava *Server* (V3) i vanjskih komponenata (**9** i **35)**.

4) *Analiza atributa kvalitete* (engl. *quality attributes*) *i metrika*

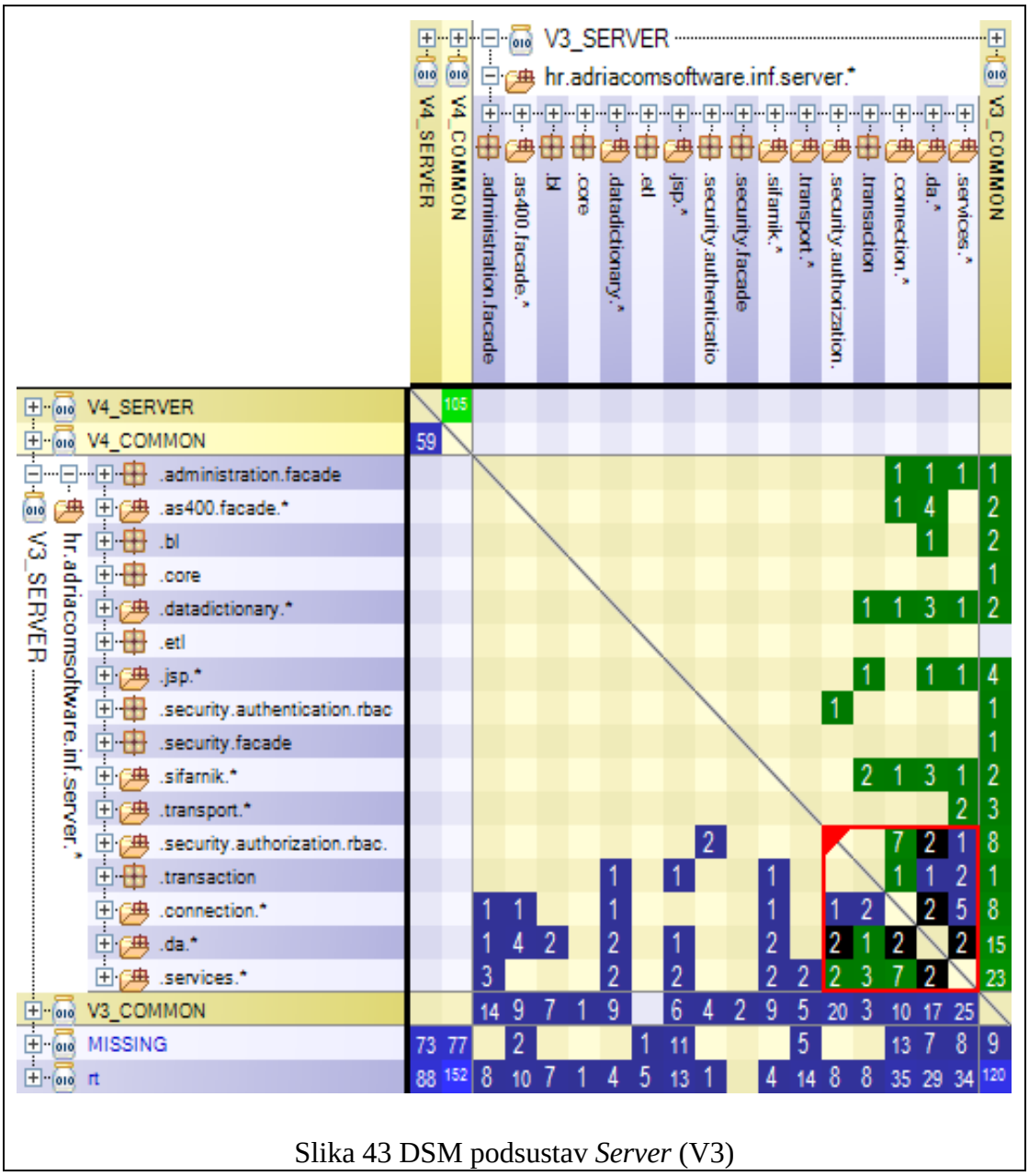
Prethodne inačice aplikacijskog okvira (V1, V2, V3) analizirali smo sa aspekta stabilnosti, relativnog opsega promjena, te niza metrika izvornog programskog koda kako bi se odučili za vrste prepravljanja koje je potrebno provesti u cilju poboljšanja atributa kvalitete referentne arhitekture.

5) *Analiza izvornog koda postojećih artefakata*

Pomoću Eclipse razvojnog alata i njegovih dodataka (engl. *plugins*) za analizu izvornog programskog koda pretraživali smo izvorni kod kako bi pronašli: *depricated code, warning statements, error statements, dead code, similar code, large classes, long methods, bugs, unused classes*. Proces prepravljanja podrazumijeva eliminiranje što većeg broja upozorenja te ponovno oblikovanje onih klasa za koje se utvrdi da ne zadovoljavaju postavljene kriterije (npr. broj linija izvornog programskog koda po klasi).

6) *Analiza statističkih podataka o korištenju postojećih poslovnih komponenata*

Za aplikacije koje korisnici koriste u obavljanju poslovnih funkcija u organizaciji vodi se evidencija o aktivnosti svakog korisnika. Analizom tih podataka evidentirali smo nekoliko komponenata koje se ne koriste, nekoliko komponenata čije odzivno vrijeme ne zadovoljava postavljene kriterije, itd.



Slika 43 DSM podsustav *Server* (V3)

7) Identifikacija aspekata

Pronalaženje potencijalnih aspekata uključuje analizu arhitekture i oblikovanja promatrane inačice (V3), manualni pregled izvornog programskog koda, korištenje alata za identifikaciju ponavljajućeg koda, te poznavanje programske strukture prethodne inačice od strane autora potencijalnih aspekata. Pronalaskom sličnog programskog koda koji se ponavlja na više mjesta, te pronalaženjem potencijalnih idioma aspekata iz raspoložive literature iz područja aspektno orijentiranog programiranja, identificirali smo buduće aspekte. Broj aspekata koje smo identificirali nije konačan broj mogućih

aspekata u promatranoj inačici, već je njihov broj i vrsta rezultat primijenjenog postupka i predložene arhitekture koji imaju svoja ograničenja.

Osim analize postojeće inačice (V3) artefakata referentne arhitekture, analizirali smo i neke nove funkcionalnosti koje će biti dio najnovije inačice aplikacijskog okvira (V4). Temeljem analize može se odlučiti da se umjesto vlastitog razvoja koristi postojeća komponenta nekog drugog proizvođača. Glavni rezultat procesa analize su definirana vanjska sučelja komponenata okvira referentne arhitekture.

5.4.1.2. **Dizajn** (engl. *Elegant Design*)

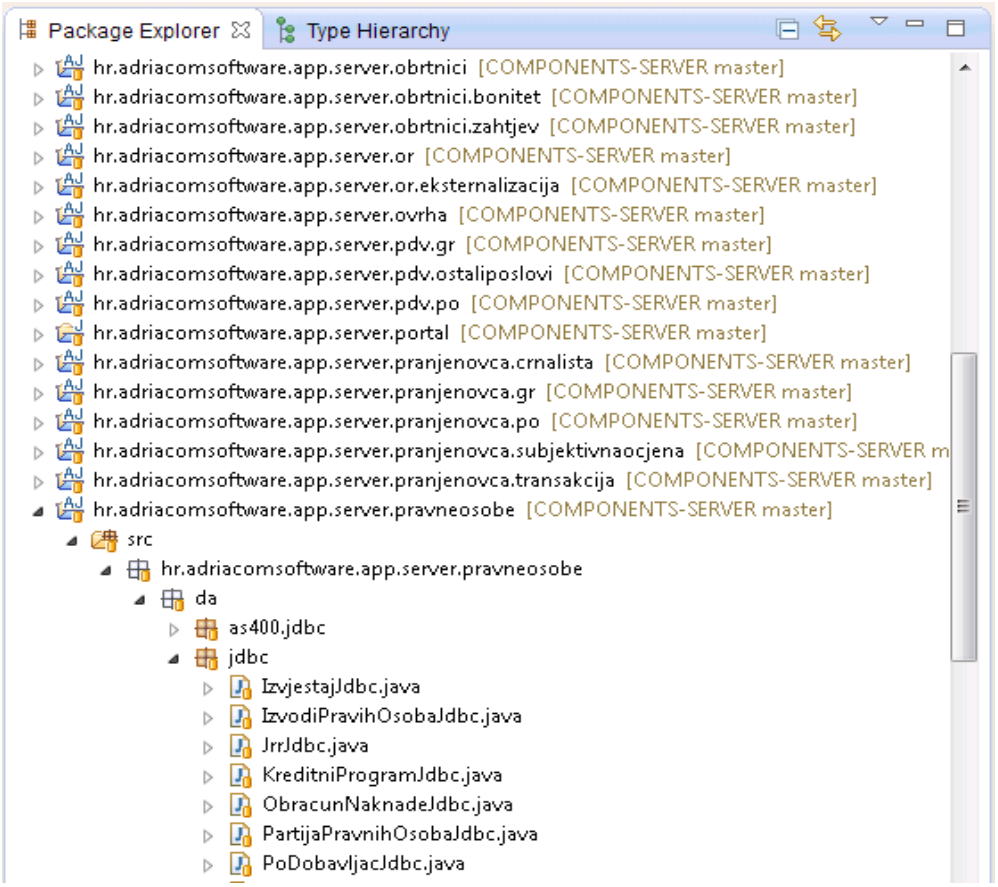
Za oblikovanje novih funkcionalnosti i izmjene postojećih komponenata koristili smo Unified Modeling Language (UML) i nekoliko vrsta dijagrama (*Use Case, Class, Sequence, Package, Deployment*). Modeliranje objekata, primjena uzoraka oblikovanja, i definiranje pripadnosti pojedinom sloju referentne arhitekture, glavne su aktivnosti u procesu oblikovanja. Funkcionalnosti se implementiraju u obliku komponenata koje svrstavamo u jedan od tri podsustava (*Common, Client, Server*); odluku o tome kojem podsustavu pripada komponenta donosimo na temelju predviđanja njenog korištenja; ako se komponenta može koristiti na serveru ili na klijentu, smještamo je u podsustav *Common*. Nakon određivanja pripadnosti pojedinom podsustavu referentne arhitekture, komponente se svrstavaju u neki od logičkih slojeva, sloj poslovne logike, sloj pristupa podacima, prezentacijski sloj, komunikacijski sloj (klijent-server). Neke od funkcionalnosti zahtijevaju definiranje modela podataka koji se spremaju u bazu podataka, za njih izrađujemo prikladan model ovisno o vrsti baze podataka koja će se koristiti (npr. relacijska baza podataka).

Komunikacija između klijenta i servera ovisi o vrsti klijenta i karakteristikama platforme na kojoj se planira izvođenje poslovnih aplikacija. Glavni cilj oblikovanja komunikacijskog mehanizma je osigurati transparentnost komunikacije, tako da prezentacijski sloj i sloj poslovne logike, koji posredno direktno komuniciraju, ne ovise o tipu komunikacije koji je definiran kao varijabilna točka i može se odnositi na izbor više mogućih mehanizama. Osim različitih mehanizama komunikacije, ovaj sloj odnosi se i na oblikovanje mehanizama za uravnoteženo opterećenje (engl. *load balancing*), visoku raspoloživost (engl. *high availability*), rezervnu opciju u slučaju pada sustava (engl. *failover*), upravljanje sigurnošću komunikacije.

5.4.1.3. Implementacija

Za razvoj aplikacijskog okvira, poslovnih komponenata i aplikacija, te većine ostalih artefakata, koristili smo **Eclipse** razvojno okruženje i nekoliko njegovih dodataka (engl. *plug-ins*). Aplikacijske komponente smo podijeli na dvije grupe, klijentske i poslužiteljske, za svaku komponentu kreirali smo *Eclipse* projekt, te na taj način omogućili efikasnije upravljanje procesima razvoja, održavanja, kompozicije aplikacija, testiranje komponenata, čuvanje izvornog koda. Prema agilnoj metodi koju koristimo (FDD) određujemo redoslijed razvoja i/ili prepravljanja komponenata (funkcionalnosti), za svaku komponentu implementiramo najprije sloj pristupa podacima (ukoliko postoji), zatim redoslijedom sve do prezentacijskog sloja. Kod programiranja koristimo definirane standarde za imenovanje artefakata, usvojene principe arhitekture. Nakon završetka razvoja aplikacijskog okvira referentne arhitekture, potrebno je prepraviti i prilagoditi postojeće poslovne aplikacije i poslovne komponente novoj referentnoj arhitekturi, uzimajući u obzir usvojene tehnike kao što su aspekti, anotacije i slično.

Slika 44 ilustrira način organizacije projekata razvoja poslovnih komponenta u kojima se implementira poslovna logika i logika pristupa podacima koji primjenjuju aplikacijski okvir najnovije inačice referentne arhitekture (V4).



Slika 44 Projekti poslužiteljskih komponenata

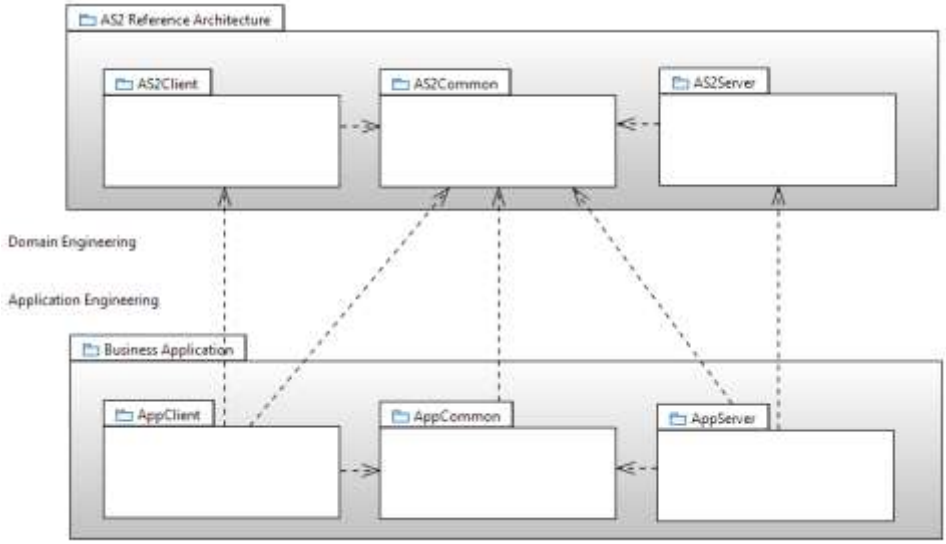
5.4.1.4. Testiranje

Nakon što su razvijene i/ili prepravljene komponente okvira referentne arhitekture i prepravljene postojeće poslovne komponente i aplikacije, uključujući i obavljeni programerski test (engl. *unit test*), tehnički arhitekt na projektu provodi ostale vrste testiranja koje se odnose na integraciju komponenata, performanse sustava, sigurnost, stabilnost, provjerava metrike izvornog programskog koda i metrike koje se odnose na oblikovanje i arhitekturu, osigurava primjenu definiranih standarda i usvojenih principa. Nakon implementacije nove inačice referentne arhitekture, tehnički arhitekt dodatno promatra stanje sustava u primjeni, prikuplja podatke o korištenju, performansama, sigurnosnim propustima, itd.

5.4.2. Podsustavi (*client, common, server*)

Referentna arhitektura (RABA) dijeli se na tri glavna podsustava: ***Client***, ***Common*** i ***Server***. Poslovne aplikacije koje se temelje na referentnoj arhitekturi, također se sastoje od istih

podsustava. Podjela na navedene podsustave ne implicira i način instalacije poslovnih aplikacija (primjerice klijent na jednoj lokaciji a server na drugoj) već predstavlja logičku podjelu programskih elemenata od kojih se sastoji okvir referentne arhitekture i aplikacija koje ga primjenjuju. Slika 45 ilustrira koncept podjele podsustava referentne arhitekture.



Slika 45. Podsustavi (moduli)

Nazivi podsustava i drugih elementa sadrže prefiks „**AS2**“ kojim smo se poslužili kako bi jednostavnije razlikovali vlastite od ostalih elemenata referentne arhitekture.

Okvir referentne arhitekture uglavnom se sastoji od klasa koje su rezultat objektno-orijentiranog oblikovanja prema predlošcima oblikovanja koje smo intenzivno koristili. Osim klasa, glavne elemente okvira referentne arhitekture predstavljaju anotacije (engl. *annotations*) i aspekti (engl. *aspects*) koje smo koristili za programiranje ponavljajućih programskih dijelova.

5.4.3. Anotacije i aspekti referentne arhitekture

Java anotacije (engl. *annotations*) koristimo za opis meta podataka, karakteristika pojedinih komponenata, metoda, varijabli, parametara ili klasa, koje kasnije koristimo u vrijeme izvođenja za odlučivanje o pokretanju predviđenih aktivnosti ili za prepoznavanje vrste pojedinih objekata. Iako postoje različiti pristupi korištenju i brojnosti anotacija, u predloženoj referentnoj arhitekturi koristimo samo mali broj anotacija, najčešće kao zamjenu za konfiguracijske datoteke (npr. *XML, properties*). Tablica 12 prikazuje anotacije koje koristimo i njihov kratki opis.

Tablica 12 Anotacije okvira referentne arhitekture

Anotacija	Opcije / opis
@AS2Factory	Obilježava klase koje kreiraju objekte drugih klasa.
@ AS2ResultSet	Klasa koja sadrži više serija poslovnih podataka.
@ AS2Singleton	Obilježava klase koje imaju samo jedan objekt.
@ AS2ValueObject	Klasa koja sadrži poslovne podatke.
@ AS2Validation	Metoda za koju se radi validacija ulaznih parametara.
@ AS2ValidationField	Polja za koja se radi provjera ispravnosti sadržaja.
@AS2Invoker	AS2, CTS, JTA, EJB, SPRING
@AS2BusinessObject	PASSTHROUGH, CACHEABLE, LOOPBACK
@AS2DataAccessObject	JDBC, AS400, JMS, CICS
@AS2FacadeMandatory	Metode za koje se provjeravaju obavezni parametri.
@AS2FacadeServer	Poslovna komponenta na serveru.
@AS2Interceptors	BEFORE, AFTER
@AS2ReportServer	Komponenta za izvješća na serveru.
@AS2ValueListHandler	Metoda za koju se podaci šalju u serijama na klijet.
@AS2CanRun	Klase koje se mogu samostalno pokrenuti.
@AS2FacadeMethodSecured	Metoda komponente za koju se radi provjera sigurnosti.
@AS2FacadeSecured	Komponenta za koju se radi provjera sigurnosti.

Funkcionalnosti koje su rasprostranjene u velikom broju klasa, čije ponavljanje predstavlja značajno povećanje redundancije programskog koda i otežava održavanje klasa, oblikovali smo pomoću aspekata (engl. *aspects*), prema pravilima aspektno orijentiranog programiranja. Priroda poslovnih aplikacija, uključujući i aplikacijski okvir kojega koristimo, zahtjeva jednostavnost kod razumijevanja njihovog izvornog koda, stoga, iako postoje različita mišljenja i pristupi kod primjene aspekata, smatramo da je njihov broj potrebno ograničiti samo na nefunkcionalne zahtjeve poslovnih aplikacija. Primjerice, za evidenciju aktivnosti korisnika u cilju osiguravanja sigurnosne pretpostavke neporecivosti poslovnih transakcija, mjerenja odzivnog vremena pojedinih komponenata ili slojeva aplikacije, internacionalizacije programskih rješenja, itd. Tablica 13 prikazuje aspekte koje koristimo u predloženoj referentnoj arhitekturi i njihov kratki opis.

Tablica 13 Aspekti referentne arhitekture

Naziv aspekta	Opis
AS2LoggingAspect	Ispis *log* poruka u aplikacijskim komponentama.
AS2ExceptionAspect	Upravljanje iznimkama u aplikacijskim komponentama.
AS2I18NAspect	Internacionalizacija korisničkih i sistemskih poruka.
AS2ResponseAspect	Evidencija odzivnog vremena poslovnih transakcija.
AS2FacadeSecurityAspect	Upravlja pravima pristupa poslovnim komponentama.
AS2FacadeValidationAspect	Provjerava poslovna pravila kod pristupa poslovnim komponentama uz pomoć anotacija.
AS2ActivityLog	Evidentira poslovne aktivnosti korisnika aplikacija.

5.4.4. Komponente referentne arhitekture i uzorci oblikovanja

Okvir referentne arhitekture sadrži veliki broj komponenata, raspodijeljenih na podsustave. Podsustav zajedničkih komponenata (***Common***) sadrži implementirane funkcionalnosti koje se ne koriste isključivo za razvoj prezentacijske logike poslovnih aplikacija (npr. validaciju unesenih podataka) kao što su funkcionalnosti podsustava ***Client***, ili s druge strane za razvoj poslovne logike (engl. *business logic*) i/ili logike pristupa podacima (engl. *data access logic*) kao što su funkcionalnosti podsustava ***Server***, već se mogu koristiti za razvoj svih dijelova poslovnih aplikacija. Kriteriji podjele pojedinačnih funkcionalnosti na podsustave referentne arhitekture odnosi se upravo na navedene zahtjeve i/ili ograničenja.

Tablica 14 prikazuje zajedničke komponente referentne arhitekture koje se koriste za razvoj aplikacijskih komponenata i/ili aplikacija, bilo na klijentu ili na serveru, u kontekstu primjene uzoraka za oblikovanje pri njihovoj implementaciji.

Tablica 14 Glavne zajedničke komponente

Naziv	**Primijenjeni uzorci oblikovanja**
Data Value Objects	*Data Transfer Object, Command, Observer*
Core	
Application Response Management	
Reporting Service	*Factory Method, Strategy*
Business Rule Engine Service	
Data Cache	*Cache*
Data Validation	
Data Services (Format, Parse, Xml, JSON)	
Email	
Exceptions	
File	
i18n	
Logging	*Strategy*
Client Request Handlers	*Client Request Handler, Proxy*
Types	

Session	
Validators	
Formatting	
Security	*Factory*

Tablica 15 prikazuje komponente aplikacijskog okvira referentne arhitekture podsustava ***Server*** koje se koriste za razvoj aplikacijskih komponenata i/ili aplikacija koje se izvode na serveru, u kontekstu primjene uzoraka za oblikovanje pri njihovoj implementaciji. Funkcionalnosti su implementirane kao vlastite komponente, međutim, način njihovog oblikovanja podrazumijeva jednostavnu zamjenu vlastitih sa nekim od standardnih rješenja otvorenog izvornog koda (engl. *open source*) ili rješenja nekog drugog proizvođača, ukoliko takav zahtjev postoji.

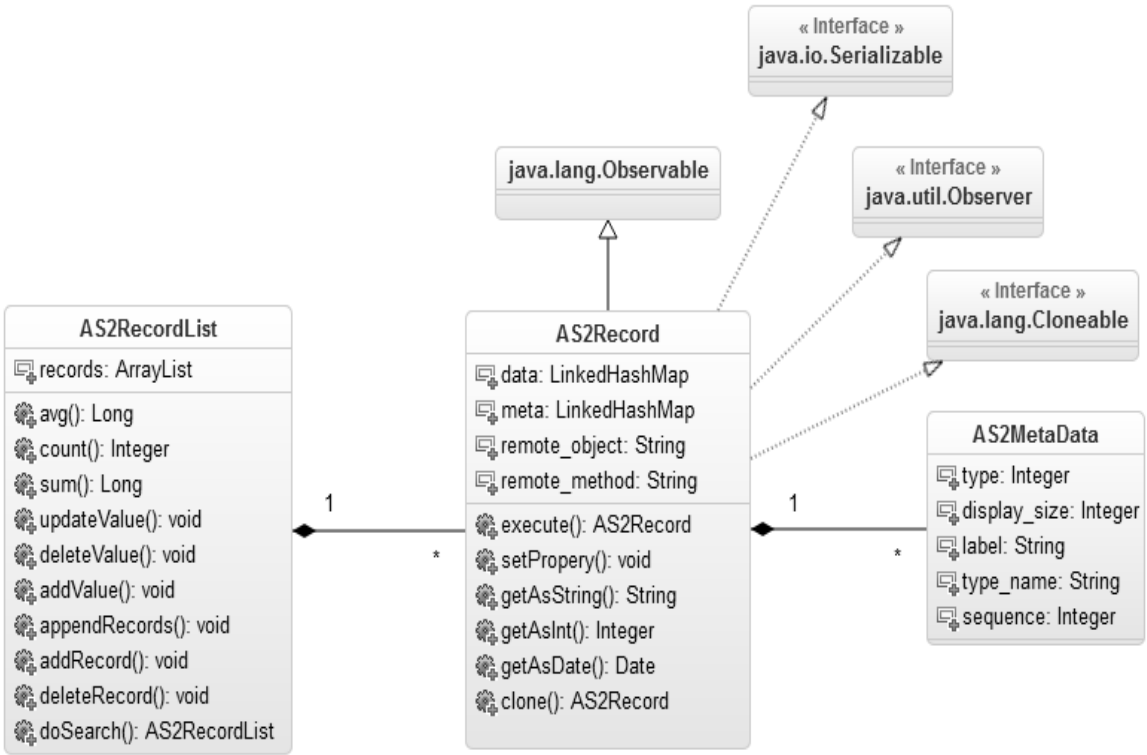
Tablica 15 Glavne serverske komponente

Naziv	**Uzorci oblikovanja**
Server Request Handlers	*Server Request Handler, Factory Method, Strategy*
Remote Object Invokers	*Invoker*
Business Object	*Factory Method, Strategy, Business Object, Facade*
Data Access Object	*Factory Method, Strategy, Data Access Object*
Data Sources	*Template*
Data Sources Connections	*Adapter*
Role Based Access Control	*Strategy, Adapter*
Transaction Service	*Factory Method, Adapter*
Session Service	*Factory Method, Strategy*
Value List Handler	*Value List Handler*

5.4.4.1. Data Value Objects

Poslovni podaci koji se obrađuju u aplikacijama intenzivno se prenose od korisničkog (engl. *user interface*) sučelja do spremišta podataka i obrnuto. Na tom putu podaci najčešće mijenjaju format te sudjeluju u raznim obradama na nekom od standardnih slojeva poslovnih aplikacija (korisničko sučelje, poslovna logika, pristup podacima, spremiše podataka). Vrlo često su

programske greške u aplikacijama uzrokovane upravo načinom čuvanja i prijenosa podataka u aplikacijama (npr. *NullPointerException*). Osim programskih grešaka, u praksi je često potrebno raditi izmjene programskih sučelja (engl. *method signature*) zbog same prirode poslovnih aplikacija ili zbog zahtjeva korisnika, a koje često otežavaju održavanje i proširivanje postojećih aplikacija.



Slika 46 *Value Object* - dijagram klasa

Namjena komponente *Data Value Objects* odnosi se upravo na rješavanje navedenih eventualnih problema. Pristup podacima koji se čuvaju u *AS2Record* objektima se odvija korištenjem *get* i *set* operacija koje u sebi sadrže kontrolni programski kod koji umanjuje mogućnost grešaka (npr. *Null Pointer Exception*). Drugo, većina ili gotovo sva programska sučelja (engl. *method signature*) kao ulazni ili izlazni parametar koriste klase *AS2Record* ili klase njene nasljednice (npr. KlijentRecord, KreditRecord) te na taj način doprinose stabilnosti programskog sučelja koje kao takvo olakšava održavanje i proširivanje postojećih aplikacija. Dizajn ove komponente temelji se na upotrebi nekoliko standardnih uzorka, *Command, Observer, Record Set, Value Object, Data Transfer Object, Meta Data Mapping, Service Data Objects i Sovereign Value Object* (Fowler, 2002, str. 401,306,508,486; Gamma, Helm, Johnson,

& Vlissides, 1995, str. 233,293; Rosko, 2007). Komponenta se koristi u svim slojevima poslovnih aplikacija. Ukoliko se na klijentu koristi programski jezik Java i standardni uzorak *Model View Controller* (MVC) ova komponenta služi kao *model* u kojem se smještaju poslovni podaci. Podaci sa korisničkog sučelja se prenose na server u istom (u obliku modela) ili transformiranome obliku, te se na serveru koriste za pozivanje *Remote Object* (Façade) poslovnih operacija pozivanjem metode *execute* ili drugih raspoloživih servisa (npr. *Rule Engine*). *Value object* komponenta se koristi unutar sloja za pristup podacima kao objekt iz kojeg se podaci prenose u spremište podataka te kao objekt za prihvat i trajno čuvanje (engl. *caching*) učitanih podataka (AS2RecordList). Osim poslovnih podataka, komponenta se koristi i za čuvanje meta podataka koji najčešće služe na klijentu za dinamički izbor kontrola, validatora podataka na korisničkom sučelju, ili na server za obradu poslovne logike i validaciju podataka prije njihovog spremanja u spremišta podataka.

```
package hr.as2.inf.data;

import hr.as2.inf.core.AS2Object;
import hr.as2.inf.server.anotations.AS2ValueObject;

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.lang.reflect.Method;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Observable;
import java.util.Set;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

@AS2ValueObject
public class AS2Record extends Observable implements AS2RecordInterface {
	private static final long serialVersionUID = -3464838910583253080L;
	protected LinkedHashMap<String, Object> _data = new LinkedHashMap();
	protected LinkedHashMap<String, Object> _meta = new LinkedHashMap();
	private boolean _dummy = false;
	private String _id = String.valueOf(System.currentTimeMillis());
	private String _remoteObject;
	private String _remoteMethod;
	private final PropertyChangeSupport _pcs = new PropertyChangeSupport(this);
	private static Log log = LogFactory.getLog(AS2Record.class);

	public AS2Record() {
	}
	public LinkedHashMap<String, Object> getProperties() {
		if (_data == null)
			_data = new LinkedHashMap<String, Object>(DEFAULT_COLLECTION_SIZE);
		return _data;
	}
	public AS2Record clone() {
		try {
			return (AS2Record)AS2Object.deepClone(this);
		} catch (Exception e) {
			return this;
		}
	}
	@Override
	public void update(Observable arg0, Object arg1) {
		setChanged();
	}
```

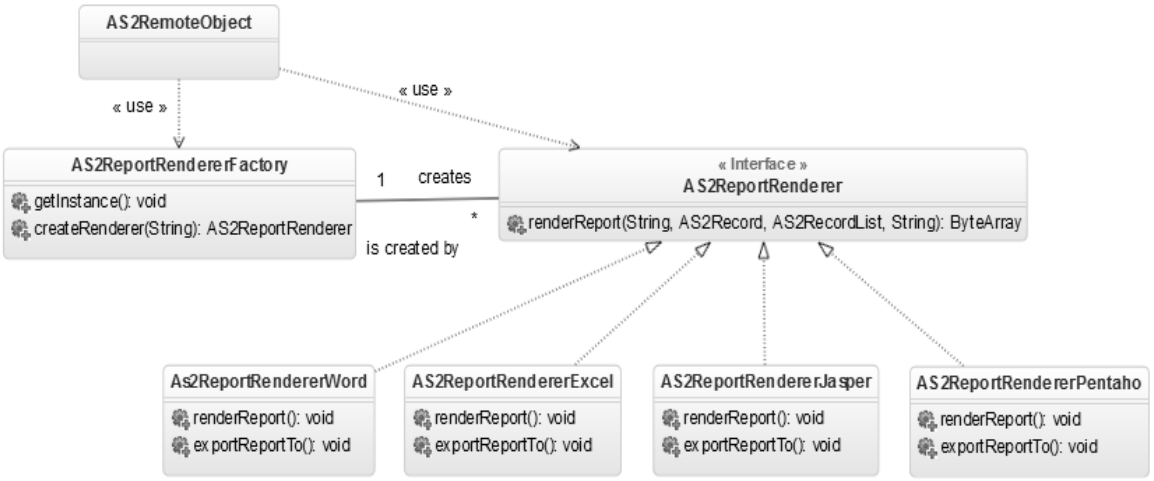
```
        public Object getAsObject(String name) {
                return getProperties().get(name);
        }
        public Object getProperty(String name) {
                return getProperties().get(name);
        }
        public byte[] getAsBytes(String name) {
                return (byte[]) getProperties().get(name);
        }
        public AS2Record execute() throws Exception {
                Class facade_class = Class.forName((String) this.getRemoteObject());
                Class parameter[] = new Class[] { this.getClass() };
                AS2Record response = new AS2Record();
                // Call Remote Object - Method here (Command design pattern implementation)
                return response;
        }
}
```

Slika 47 Dio izvornog koda za klasu *AS2Record*

5.4.4.2. Reporting Service

Poslovne aplikacije najčešće koriste više različitih formata za prikaz poslovnih izvješća prema krajnjim korisnicima aplikacija. Formati poslovnih izvješća mogu biti tekstualnog, tabličnog, grafičkog, i raznih drugih oblika. Isto tako često je potrebno pretvarati izvješća iz postojećeg formata u drugi željeni format, primjerice, izvješće u formatu *xlsx* (*Excel*) potrebno je pretvoriti u *pdf* (Acrobat) format, i to bez dodatnih korisničkih napora. Razlog zbog kojeg je *Reporting Service* komponenta, kao dio platforme, potrebna za razvoj poslovnih aplikacija odnosi se na smanjivanje rizika eventualnog prelaska sa postojećeg na novi izvještajni sustav (npr. ukoliko se *Jasper* izvješća se zamjenjuju sa *Excel* izvješćima); poboljšanje učinkovitosti razvojnih programera za poslovne aplikacije uslijed postojanja unaprijed razvijenih gotovih komponenata pomoću kojih se rješavaju kompleksni zadaci poput internacionalizacije, pretvaranja u druge formate, pokretanje vanjskih servisa za manipuliranje sa izvješćima, itd.

Tipičan scenarijo upotrebe ove komponente odnosi se na pretvaranje poslovnih podataka, koji potječu iz nekog od raznih izvora podataka na serveru, iz oblika objekata klase *AS2RecordList* u korisniku razumljivo izvješće. Izvješće se generira prema predlošku unaprijed definiranog *Jasper* izvješća, prikazuje prema krajnjem korisniku na *web* sučelju pomoću zadanog čitača na korisnikovom računalu. Korisnik uz pomoć aplikacije na strani klijenta specificira poslovnu komponentu (*Remote Object/Remote Method*) kao izvor podataka za izvještaj, unosi potrebne dodatne podatke, šalje naziv predloška izvještaja (u koliko predložak postoji) te vrstu formata za prikaz izvješća.



Slika 48 Dijagram klasa - *Reporting Service*

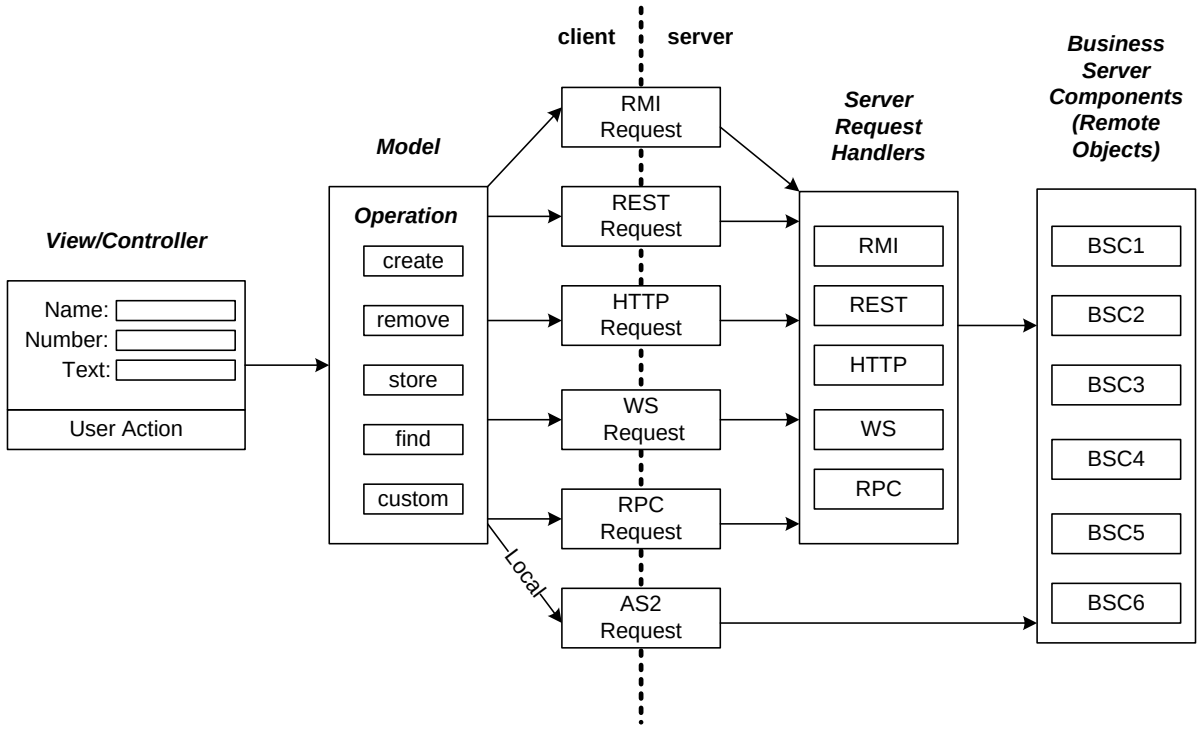
Dizajn ove komponente temelji se na upotrebi dva standardna uzorka, *Factory Method i Strategy* (Gamma i ostali, 1995, str. 87,315). Korištenje komponente odvija se preko poziva *AS2ReportRendererFactory* objektu koji na temelju parametra vrste formata izvješća stvara objekt *AS2ReportRenderer* kojeg pozivni objekt koristi za generiranje izvješća. Upotreba *Strategy* uzorka ima za cilj istovremeno korištenje više različitih izvještajnih rješenja, te ublažavanje rizika prelaska sa postojećeg na eventualno novo izvještajno rješenje.

5.4.4.3. Server Request Handlers

Za razliku od prije samo nekoliko godina, današnje poslovne aplikacije služe se sa raznim udaljenim servisima, pozivaju ih kako bi dobili, obradili ili provjerili potrebne poslovne podatke. U toj komunikaciji koriste se različiti, uglavnom standardizirani načini komunikacije, dok s druge strane te iste aplikacije trebaju omogućiti pristup drugim udaljenim aplikacijama. Važna komponenta u takvom okruženju su *Server Request Handler* objekti koji na centraliziran način upravljaju sa tom komunikacijom, zaprimaju pozive prema poslovnim komponentama (*Remote Objects*) te ih prosljeđuju prema za to određenom *Invoker* objektu za daljnju obradu.

RABA definira okvir za implementaciju više različitih stilova daljinskog pozivanja poslovnih objekata na serveru. Jedan od stilova komunikacije klijenta prema poslovnim objektima na serveru je i predloženi *Local* (slika 49) stil komunikacije koji koristimo u dva slučaja: za

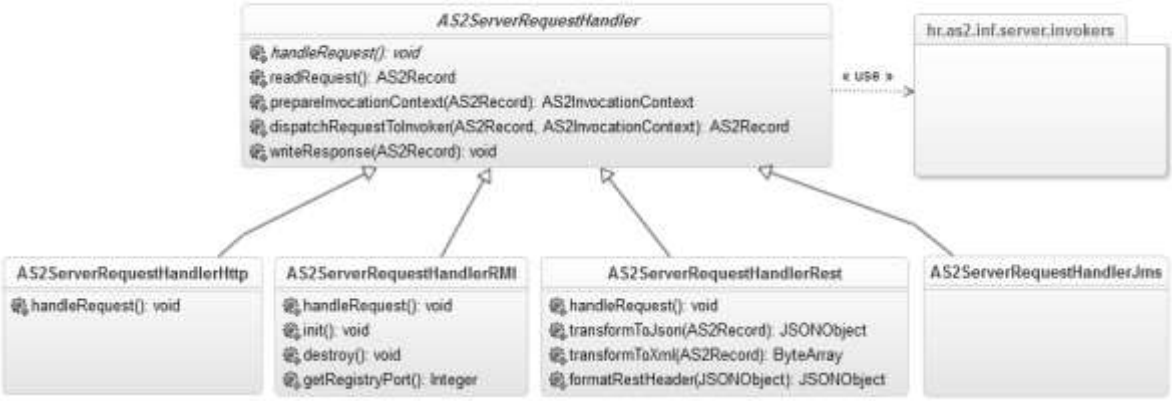
vrijeme razvoja poslovnih aplikacija u razvojnom alatu (npr. Eclipse) te u slučaju kada se poslovna aplikacija koristi bez upotrebe aplikacijskog poslužitelja (engl. *fat client*).



Slika 49 Kontekst upotrebe *Server Request Handler* komponente

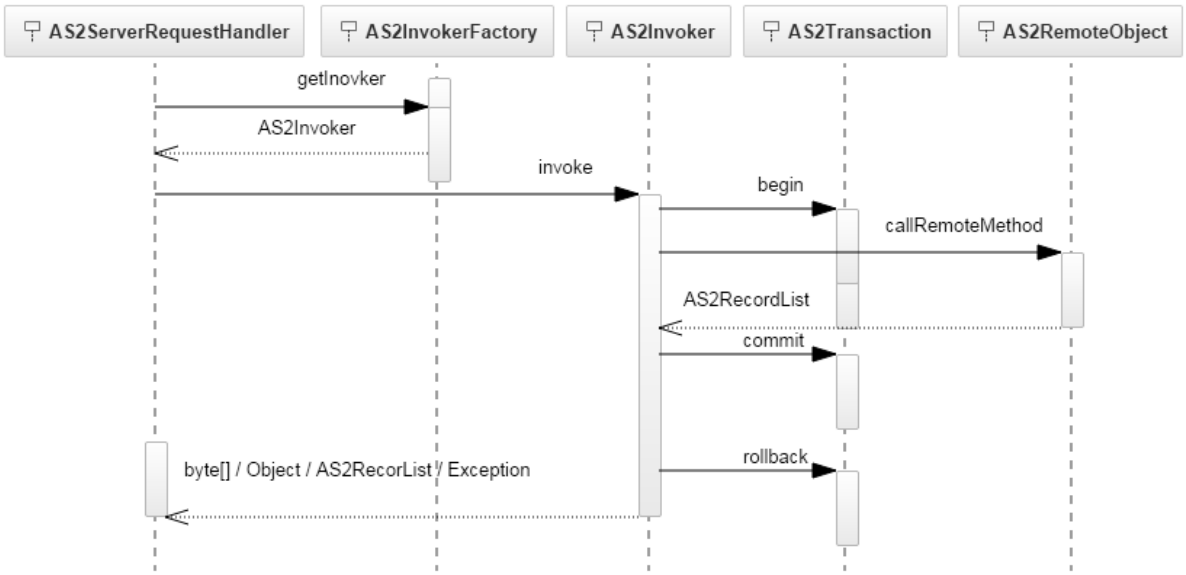
Za poslovne aplikacije za koje je potrebno osigurati uravnoteženo opterećenje (engl. *load balancing*), visoku raspoloživost (engl. *high availability*), rezervnu opciju u slučaju pada sustava (engl. *failover*) i neke druge standardne funkcije, RABA omogućuje korištenje drugih raspoloživih rješenja bez potrebe za značajnijim izmjenama u konfiguraciji sustava.

Dizajn ove komponente temelji se na upotrebi nekoliko standardnih uzorka, *Server Request Handler, Protocol Plug-ins, Marshaller, Remoting Error, InvocationContext* (Völter, Kircher, & Zdun, 2005, str. 51,55,63,133). Korištenje komponente odvija se preko poziva klijenta *AS2ServerRequestHandler* objektu koji primljene podatke priprema za daljnje slanje prema zadanom *Invoker* objektu. Pozivi sa klijenta odnose se na različite vrste protokola, primjerice JavaScript Object Notation (JSON) protokol, poziv sa podacima se pretvara djelomično u *AS2Record* objekt a djelomično u *AS2InvocationContext* objekt te prosljeđuje dalje prema *AS2Invoker* objektu.



Slika 50 *Server Request Handlers* – dijagram klasa

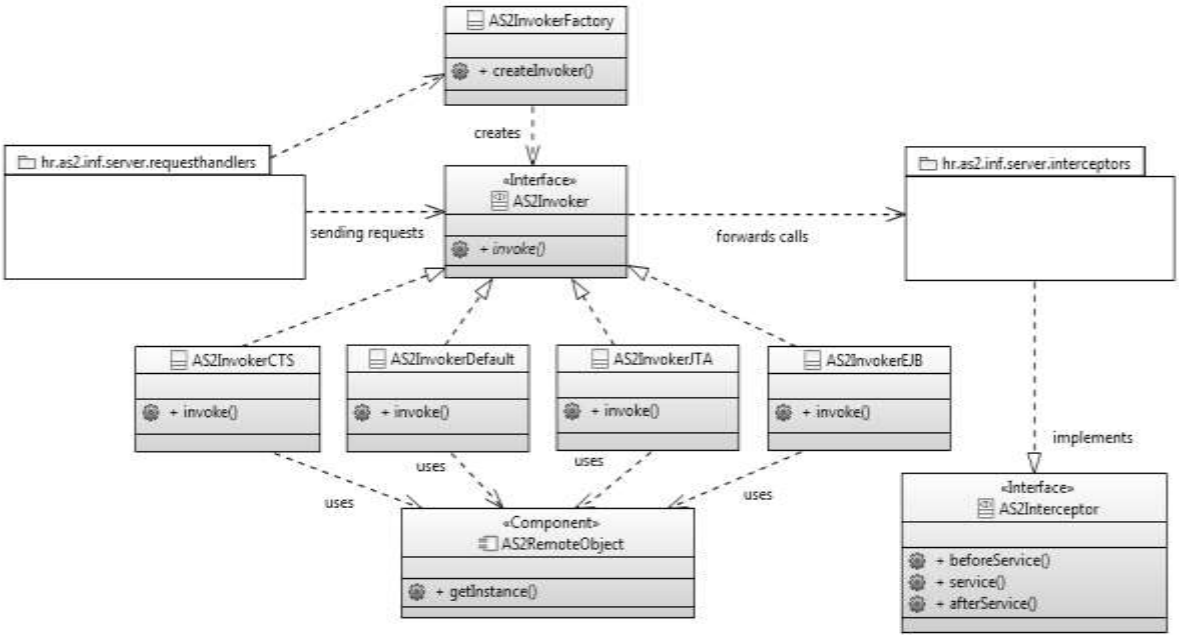
Nakon završetka poziva, poslovni podaci ili podaci o greški (engl. *exception*) transformiraju se putem istog protokola prema klijentu koji je uputio poziv. Ova komponenta je ključna komponenta okvira referentne arhitekture, tako je rangirana i od strane stručnjaka koje smo ispitali putem upitnika; stoga ćemo je prikazati i pomoću dijagrama slijeda. Slika 51 prikazuje redoslijed poziva glavnih objekata u trenutku kada je server zaprimio poziv sa klijenta.



Slika 51 *Server Request Handler* – dijagram redoslijeda poziva na serveru

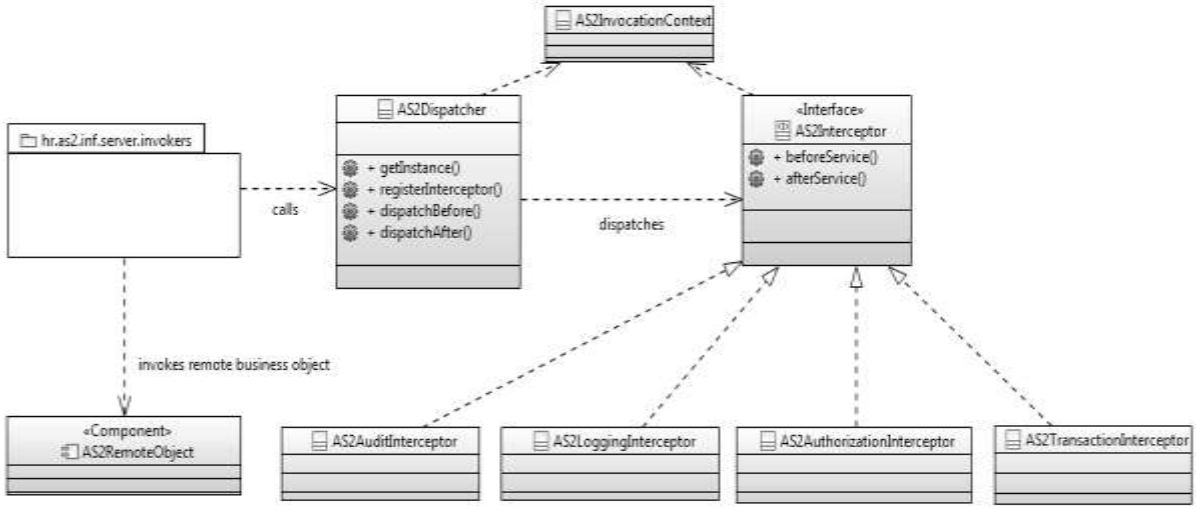
5.4.4.4. Remote Object Invokers

Udaljeni poslovni objekti koji se nalaze na serveru obavljaju razne vrste poslovnih transakcija, poput spremanja podatka u bazu, pripreme izvješća u raznim oblicima te provjere poslovnih podataka. Jednom kada zahtjev za pozivom prema poslovnom objektu stigne na server, nakon njegovog pretvaranja u definirani oblik (*AS2Record, AS2InvocationContext*), moguće je proslijediti ga na neki od različitih *Invoker* objekata. U predloženoj arhitekturu podržavamo nekoliko vrsta ovih objekata. Primjerice, uobičajeno je da se poslovna transakcija izvodi na način da se podaci spremaju u bazu i odmah postaju raspoloživi i drugim korisnicima sustava. Takva vrsta transakcije svrstava se u ACID (*Atomicity, Consistency, Isolation, and Durability*) vrstu poslovnih transakcija. Međutim, postoji i poslovna potreba da se podaci na serveru prikupljaju u nekoliko vremenski razdvojenih koraka, čuvaju se, te tek nakon zahtjeva korisnika koji ih je poslao, postaju raspoloživi drugim korisnicima. U takvom slučaju ne radi se o vrsti transakcije ACID, već se ta vrsta transakcije naziva „duga“ transakcija za koju se koristi uzorak CTS (engl. *Compensating Service Transaction)*. RABA podržava takvu vrsta transakcija ukoliko se server konfigurira da koristi jedan od definiranih objektata ove vrste (*AS2InvokerCTS*) te ukoliko klijent kod poziva koristi za to definirana pravila.



Slika 52 *Invokers* – dijagram klasa

Dizajn ove komponente temelji se na upotrebi nekoliko standardnih uzorka *Invoker*, *Compensating Service Transaction, Backward-Forward Transaction Service, Factory Method, Strategy* (Gamma i ostali, 1995, str. 107,315; Rischbeck & Erl, 2009a, str. 631; Rosko, 2010a; Völter i ostali, 2005, str. 43). Korištenje komponente inicira *AS2ServerRequestHandler* objekt koji primljene podatke sa klijenta priprema i skupa sa *AS2InvocationContext* objektom šalje prema *Invoker* objektu. *Invoker* poziva poslovne objekte na način da preko *Java Reflection* funkcionalnosti koja je implementirana u *AS2Record.execute()* metodi pozove ciljani poslovni objekt i proslijedi mu podatke koji su stigli sa klijenta uz ostale podatke koji se nalaze u *AS2InvocationContext* objektu a koji se odnose na podatke o korisniku, njegovoj sesiji, itd. Upotrebom *Strategy* uzorka, RABA omogućava korištenje više različitih *Invoker* objekata, iako se u praksi najčešće koristi samo jedan od njih. Prije i poslije poziva poslovnih objekata, potrebno je omogućiti prosljeđivanje pozivnih i povratnih podataka drugim servisima poput *Logging, Audit, Application Response Management, Authorization, Transaction*, itd. U tu svrhu RABA definira pomoćne komponente koje možemo svrstati u grupu *Inerceptor* komponenata, slika 53.



Slika 53 *Interceptors* – dijagram klasa

Interceptor komponenta temelji se na korištenju uzorka *Invocation Interceptor* (Völter i ostali, 2005, str. 130) kako bi se omogućilo korištenje većeg broja dinamički konfiguriranih servisa. Umjesto ove komponente istu funkcionalnost moguće je postići i korištenjem aspekata (engl. *aspects*), međutim, centralizirani način koji podrazumijeva da svi pozivi prema serveru prolaze kroz istu komponentu (jedan od *Invoker* objekata) daju nam za pravo odabrati ovakav način.

Aspekte i njihovu prednost u odnosu na ostale tehnike koristimo samo u slučajevima kada ne postoji centralizirani već raspršeni način korištenja određenih servisa.

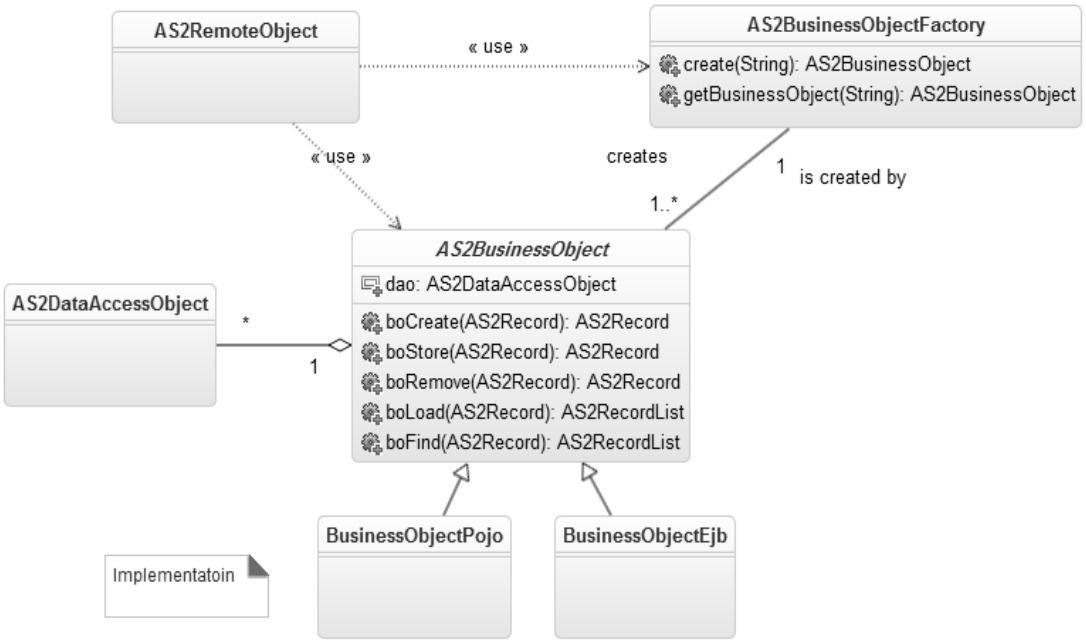
5.4.4.5. Business Object

U kontekstu predložene arhitekture, pod poslovnim objektima (engl. *business objects*) podrazumijevamo aplikacijski sloj za obradu poslovnih podataka koji je neovisan o izvoru tih podataka i obrnuto, pri čemu izvori podataka ne ovise o poslovnim objektima osim u slučaju eventualnog registriranja zbog obavijesti o poslovnim događajima (transakcijama izmjena podataka). Glavna svrha ovako izoliranog aplikacijskog sloja je pojednostavljivanje razvoja i održavanja poslovne logike, te ublažavanje eventualnih rizika kojima je poslovno okruženje podložno uslijed čestih poslovnih i tehnoloških izmjenama. Standardna definicija poslovnih objekata opisuje ih kao objekte koji najbliže prikazuju poslovni model u obliku entiteta, procesa ili događaja koji su značajni za poslovanje (Casanave, 1997). Tipični primjeri poslovnih objekata su: klijent, proizvod, račun, kupnja, narudžba, transakcija, knjiženje, itd. Predloženi način korištenja poslovnih objekata temelji se na definiciji Business Application Architecture (BAA) koju definira Object Management Group (OMG, 1996). Iako se najčešće pojedinačni poslovni objekt odnosi na jednu tabelu u bazi podatka, poslovne objekte je potrebno promatrati u širem kontekstu, primjerice, poslovni objekt se može koristiti za obradu podataka koji se ne spremaju u bazu podataka, ili s druge strane za svoju obradu kombiniraju podatke iz nekoliko tabela. Isto tako, poslovni objekti mogu poslužiti za pristup starijim (engl. *legacy*) sustavima u isto vrijeme dok pristupaju i relacijskoj bazi podataka, te na taj način omogućiti eventualni budući prelazak sa starijeg na relacijski model baze podataka. Poslovni objekti su također dobra podloga za uvođenje procesnih aplikacija u organizaciju.

Predložena arhitektura prvenstveno podržava Plain Old Java Object (POJO) vrstu poslovnih objekata koji ne sadrže poslovne podatke (engl. *stateless*), osim iznimno, i to za vrijeme trajanja poslovne transakcije, te koji se dodatno mogu svrstati u prolaznu vrstu poslovnih objekata (engl. *passthrough*). S druge strane, predložena arhitektura je otvorena i za Enterprise Java Bean (EJB) vrstu poslovnih objekata ukoliko je za to postoji korisnički zahtjev ili tehnička potreba.

Poslovne aplikacije ovu komponentu intenzivno koriste, najčešće na način da aplikacija na klijentu šalje poslovne podatke za spremanje ili upit u bazu podataka koje zaprima poslovni objekt, obrađuje ih i najčešće ih prosljeđuje dalje prema bazi podataka. Sama validacija ispravnosti podataka nije nužno dio ovog procesa, ali ukoliko to poslovni uvjeti nalažu, primjerice u slučaju da je klijent neka druga aplikacija koja poslovne podatke ne provjerava,

takva mogućnost je predviđena na način da se aktivira servis presretač (engl. *interceptor*) za validaciju podataka na temelju definiranih pravila i meta podataka.



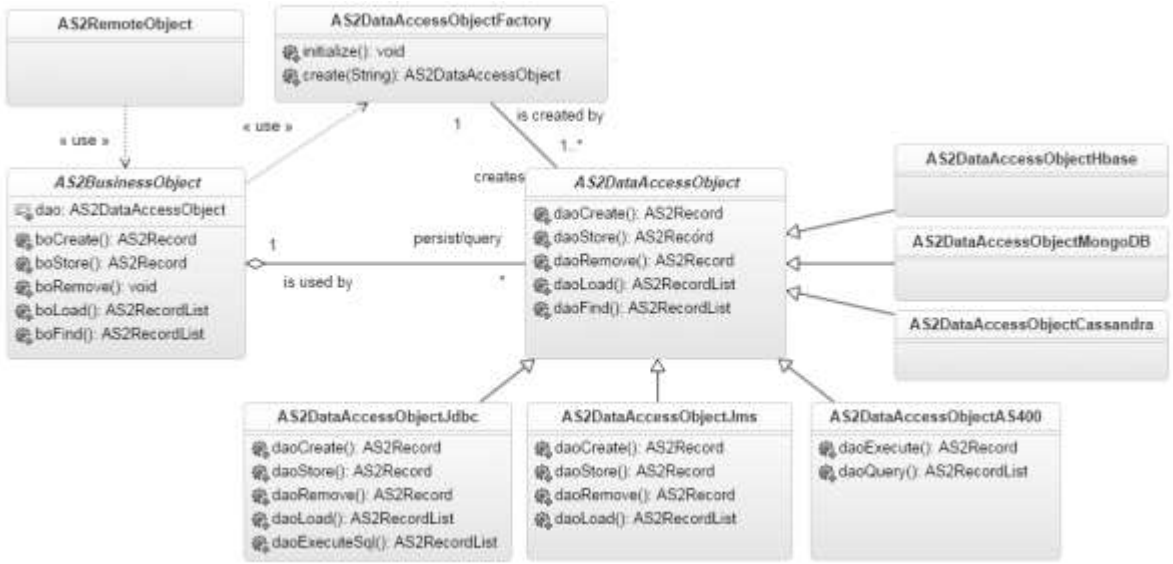
Slika 54 *Business Objects* – dijagram klasa

Dizajn ove komponente temelji se na upotrebi uzorka *Business Object*, *Façade*, *Factory Method*, *Strategy* (Alur, Malks, Crupi, Booch, & Fowler, 2003; Gamma i ostali, 1995, str. 107,185,315). Korištenje komponente inicira *AS2Invoker* objekt koji primljene podatke sa klijenta dostavlja do poslovne komponente (engl. *Remote Object*) koja zatim kombinira jedan ili više poslovnih objekata *AS2BusinessObject* kako bi se dovršila poslovna transakcija. Osim klasične obrade poslovnih podataka, ova komponenta omogućava pozivanje dodatnih servisa poput *Rule Engine* servisa na način da prikupljene podatke od strane klijenta i/ili drugih izvora koristi za pozivanje navedenog servisa u svrhu validacije postavljenih poslovnih pravila.

5.4.4.6. Data Access Object

Pristup poslovnim podacima uglavnom se u praksi rješava pomoću alata za uparivanje poslovnih objekata sa tabelama u relacijskim bazama podataka (engl. *object relational mapping)*. Neki od tih alata su: Hibernate, TopLink, iBatis, OpenJPA, Apache JDO. Međutim, osim Java Data Objects (JDO) pristupa koji je također ograničen na tri vrste izvora podataka

(File, OODB, RDMS), nije nam poznato da postoji sveobuhvatni pristup koji omogućava poslovnim objektima da pristupaju ostalim vrstama spremišta podatka na jedinstven način. Osim toga, svaki od navedenih alata zahtjeva detaljno poznavanje prije nego ga se može koristiti za razvoj poslovnih aplikacija, što je moguće ublažiti ukoliko se definira komponenta za pristup podacima koja navedene alate može ali i ne mora koristiti. Kod korištenja ove komponente poslovni podaci isključivo se prenose u obliku *AS2Record* i *AS2RecordList* objekata. Za obradu korisničkih zahtjeva za pristupom samo dijelu podataka, bilo u smjeru naprijed ili nazad, koristi se *Value List Handler* uzorak i komponenta *AS2ValueListHandler* pomoću koje se ta funkcionalnost realizira.



Slika 55 *Data Access Objects* – dijagram klasa

Dizajn ove komponente temelji se na upotrebi nekoliko uzoraka *Data Access Object, Factory Method, Strategy* i *Dynamic Data Access Object* (Alur i ostali, 2003; Gamma i ostali, 1995, str. 107,315; Rosko & Konecki, 2008a). Korištenje ove komponente inicira *AS2Business* objekt koji podatke za upit ili spremanje dostavlja do *AS2DataAccessObject* objekta kako bi se oni spremili u neko od spremišta podataka ili kako bi se koristili za upit. U slučajevima kada je spremište podataka relacijska baza, ova komponenta omogućava dinamičko uparivanje primljenih podataka sa klijenta sa meta podacima u bazama podataka kako bi se operacije *Create, Read, Update, Delete* (CRUD) potpuno automatizirale. Poslovni objekti koji iniciraju korištenje ove komponente nisu povezani direktno sa *AS2DataAccessObject* objektima, što omogućava njihovu zamjenu ukoliko je to potrebno. Primjerice, u situaciji kada poslovni objekt

koji se koristi za obradu podataka o kupcu pristupa sustavu iSeries (AS400) i koristi *AS2DataAccessObjectAS400* za pristup podacima, zbog eventualnog premještanja podataka o kupcima na novu relacijsku bazu podataka, biti će potrebno jedino izraditi novu *AS2DataAccessObjectJdbc* klasu dok ostali programski nije potrebno mijenjati. Tablica 16 prikazuje glavne metode *Data Access Object* klase.

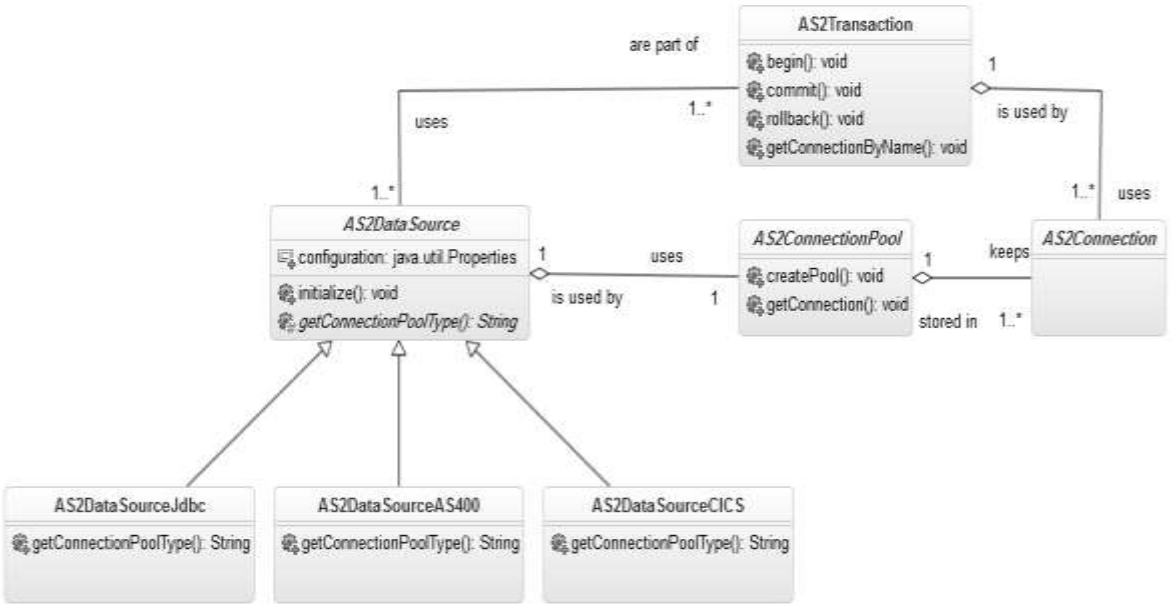
Tablica 16 Glavne metode *Data Access Object* klase

Metoda	**Opis**
daoCreate	Sprema podatke u spremište podataka po prvi put.
daoStore	Ažurira podatke u spremištu podataka prema ključu.
daoRemove	Briše podatke iz spremišta podataka prema ključu.
daoLoad	Čita podatke iz spremišta podataka prema ključu.
daoFind	Čita podatke iz spremišta podataka prema zadanom kriteriju.
daoCreateMany	Sprema skup podatka u spremište podataka po prvi put.
daoRemoveMany	Briše skup podatka iz spremišta podataka prema ključu.

5.4.4.7. Data Sources

Poslovne aplikacije potencijalno mogu pristupati velikom broju vanjskih izvora podataka. Za pristup svakom od njih potrebno je raspolagati sa nizom parametara poput korisničkog identiteta, adrese računala, i drugih konfiguracijskih podataka.

Ova komponenta može se koristiti i za čuvanje (engl. *cache*) podataka koji su prethodno učitani iz izvora podataka u slučajevima kada je zbog poboljšanja performansi sustava to potrebno. Dizajn ove komponente temelji se na upotrebi uzoraka *Template i Caching* (Gamma i ostali, 1995, str. 325; Kircher & Jain, 2004, str. 83). Komponenta usko je povezana sa *Transaction* i *Data Source Connection* komponentama. Naime, za vrijeme trajanja poslovne transakcije, *Data Access Object* može zahtijevati vezu (engl. *connection*) prema izvoru podataka i po nekoliko puta za redom. Brigu o čuvanju i dodjeljivanju za to potrebne jedne te iste veze vodi upravo *Data Source* komponenta.

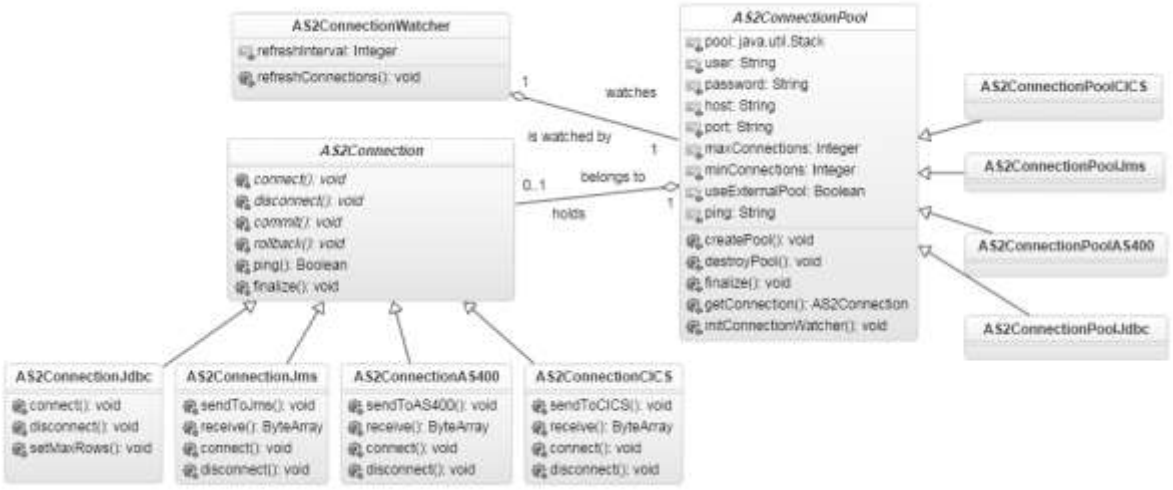


Slika 56 *Data Sources* – dijagram klasa

5.4.4.8. Data Sources Connections

Za pristupanje izvorima podataka koriste se veze (engl. *connections*), dok se za njihovo čuvanje zbog njihove ponovne upotrebe koristite spremišta veza (engl. *pool*). Spremišta veza za pristup izvorima podataka podržavaju svi aplikacijski serveri, međutim, to se uglavnom odnosi samo na iste vrste izvora podataka (npr. relacijske baze). Za podršku poslovnim aplikacijama koje imaju potrebu pristupati različitim vrstama izvora podataka (npr. IBM CICS), definirali smo ovu komponentu koja na vrlo jednostavan način omogućava upravljanje kompleksnim poslovnim transakcijama koje uključuju sve potencijalne vrste izvora podataka koji podržavaju funkciju transakcija.

Dizajn ove komponente temelji se na upotrebi uzoraka *Adapter, Pooling* (Gamma i ostali, 1995, str. 139; Kircher & Jain, 2004, str. 97). Veze (engl. *connection*) prema izvoru podataka čuvaju se u objektima klase *AS2ConnectionPool*, dok se *AS2ConnectionWather* brine o održavanju njihove ispravnosti i vremenskih ograničenja poput isteka vremena (engl. *time out*) i slično.



Slika 57 *Data Sources Connections* – dijagram klasa

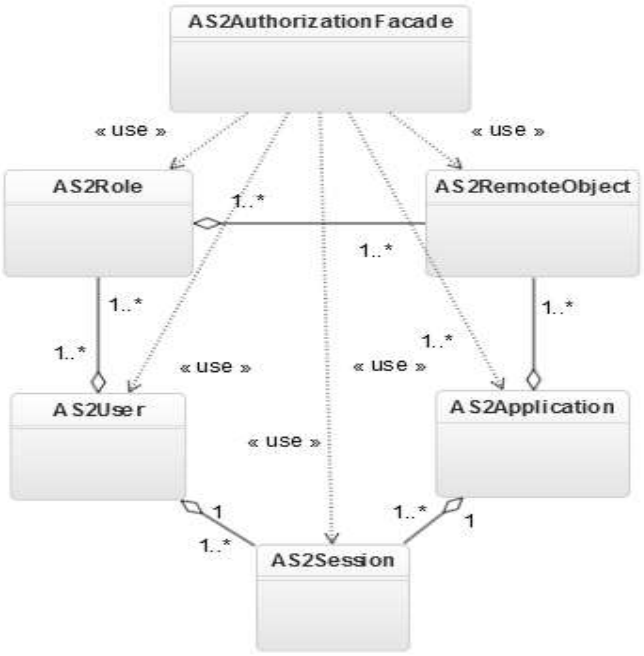
5.4.4.9. Role Based Access Control

Prava pristupa poslovnim funkcijama neke aplikacije definira sama organizacija koja aplikaciju koristi. Za definiranje prava pristupa koriste se dvije standardne sigurnosne funkcije autentikacija (engl. *authentication*) i autorizacija (engl. *authorization*). Obje funkcije provjere sigurnosti, moguće je delegirati standardnim vanjskim komponentama (npr. LDAP). Međutim, delegiranje upravljanja autorizacijom, za razliku od autentikacije, najčešće je kompleksno i neizvodivo na način koji odgovara prirodi poslovnih aplikacija.

Komponentu za autentikaciju definirali smo kako bi omogućili korisnicima poslovnih aplikacija da koriste vanjski servis za autentikaciju ukoliko ga posjeduju, a kao rezervnu mogućnost definirali smo način autentikacije preko korisničkih identiteta u relacijskoj bazi podataka.

Slika 58 *Authentication* – dijagram klasa

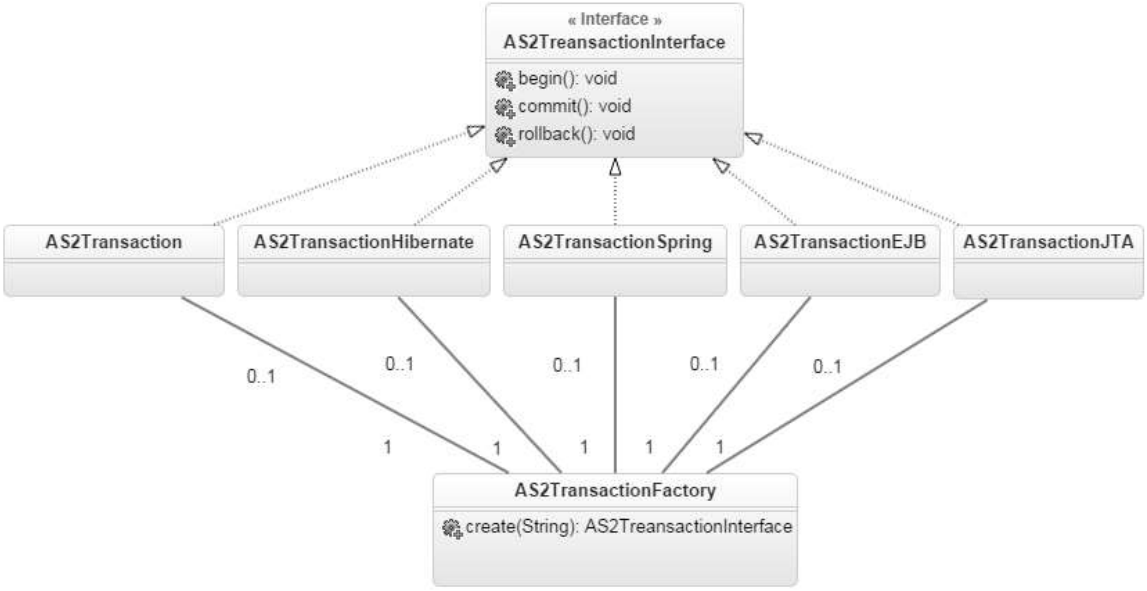
Za razvoj komponente za autorizaciju korisničkih prava koristili smo pristup pod nazivom Role Base Accessed Control (RBAC). Model autorizacije korisnika koji mogu pristupati u više poslovnih aplikacija, korištenjem više različitih uloga prikazan je na slici 59.



Slika 59 *Authorization* – dijagram klasa

5.4.4.10. Transaction Service

Način upravljanja poslovnim transakcijama kojeg predlažemo, odnosi se na vlastito rješenje, kako bi omogućili potpunu fleksibilnost kod korištenja različitih vrsta izvora podataka, ne samo relacijskih baza koje su najčešće jedine koje su podržane od strane aplikacijskih poslužitelja. Drugi razlog razvoja vlastitog rješenja, odnosi se na mogućnost izvođenja "dugih" transakcija prema uzorku *Compensating Service Transaction* za koje, koliko nam je poznato, ne postoji standardno definirano rješenje. Osim toga, predloženo rješenje je lako zamjenjivo nekim od raspoloživih rješenja, ukoliko se organizacija za to odluči. Koordinacija poslovnih transakcija provodi se na centralnom mjestu, direktno u objektima klase *Invoker* ili se ista delegira nekom od *Interceptor* objekata.

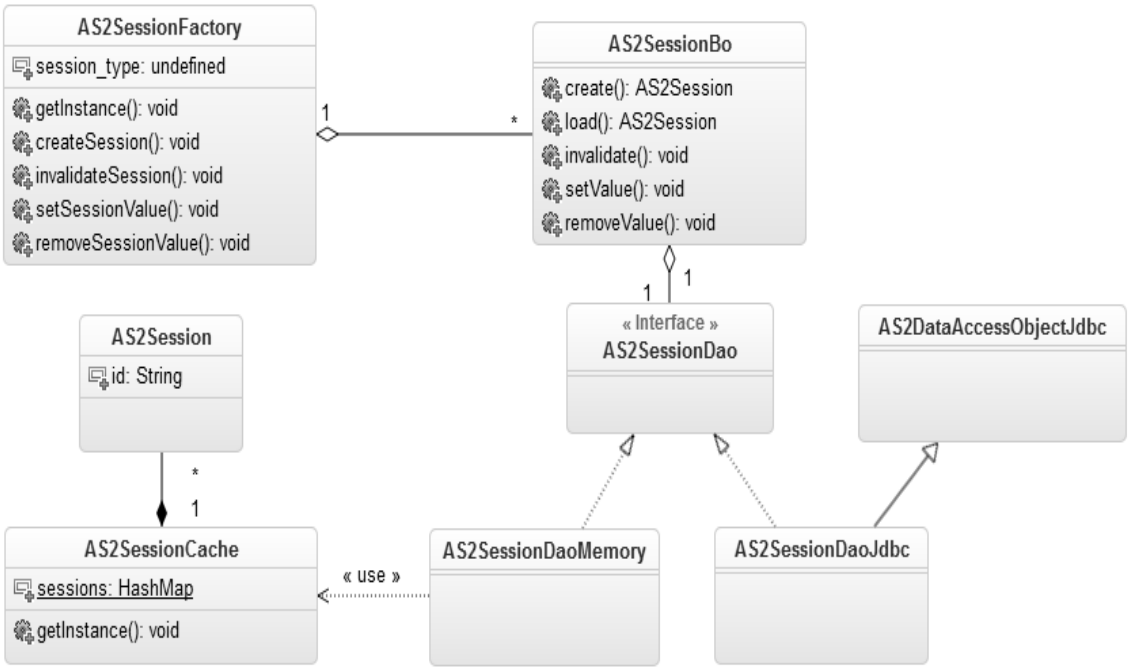


Slika 60 *Transaction Service* – dijagram klasa

Dizajn ove komponente temelji se na upotrebi uzoraka *Coordinator i Compensating Service Transaction* (Kircher & Jain, 2004, str. 111; Rischbeck & Erl, 2009a, str. 631). Poslovne transakcije potencijalno uključuju više izvora podataka, pri čemu svakog od njih predstavlja jedan objekt klase *AS2Connection* koji se u ovom kontekstu naziva učesnik (engl. *participant*) a koji zajedno sa drugim objektima iste vrste sudjeluje u kompletiranju jedne poslovne transakcije.

5.4.4.11. Session Service

Kod svake prijave u sustav korisniku se dodjeljuje jedinstveni identitet koji je povezan sa korisničkim identitetom. U isto vrijeme, jedan korisnik može se prijaviti više puta u istu aplikaciju na istom ili drugom računalu, pri tome se za svaku prijavu generira drugi identifikator prijave (engl. *session*). Aplikacije koje se razvijaju na temelju predložene arhitekture ne izvode se nužno u okruženju aplikacijskog poslužitelja (npr. *fat client*), te je za takve okolnosti potrebno koristiti vlastitu komponentu. U drugim slučajevima, kada se koristi neki od *Java* aplikacijskih poslužitelja, omogućeno je pojednostavljeno korištenje raspoloživih vanjskih komponenata za upravljanje sa prijavama (sesijama) prema mogućnostima poslužitelja koji se koristi (npr. *File, Memory-Single, RDBMS, Cookie, Memory-Cluster*).

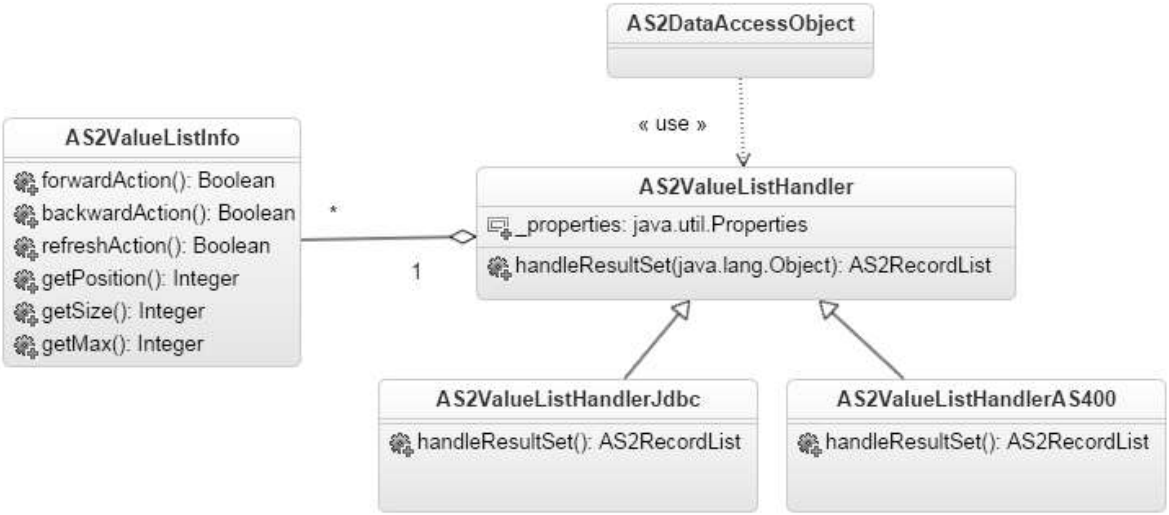


Slika 61 *Session Service* – dijagram klasa

5.4.4.12. Value List Handler

Value List Handler je mehanizam koji se koristi na serveru kako bi se klijentima omogućilo pregledavanje masovnih podataka u više koraka, umjesto svih od jednom, što je najčešće neizvodivo uslijed ograničenja hardvera ili otežane preglednosti podataka. U slučajevima kada izvor podataka podržava takvu mogućnost, ova komponenta služi samo kao koordinator, a u

drugim slučajevima koristi se za upravljanje korisničkim zahtjevima za prikaz podataka u oba smjera (naprijed i/ili nazad).

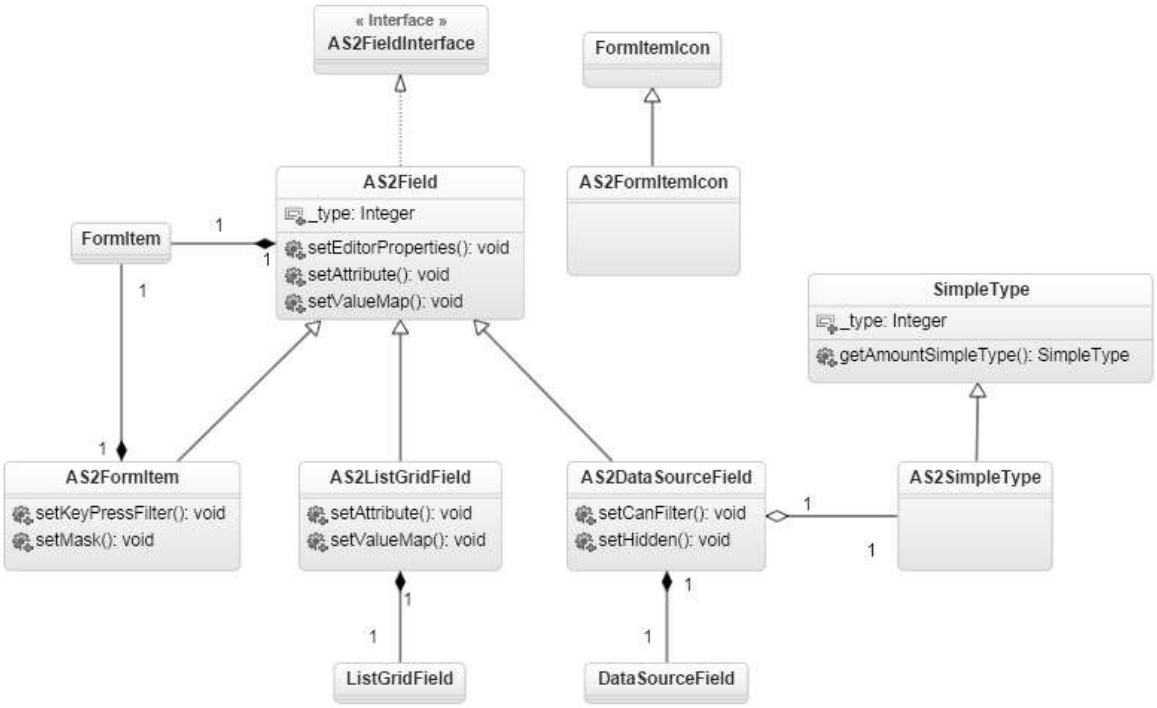


Slika 62 *Value List Handler* – dijagram klasa

Dizajn ove komponente temelji se na upotrebi uzorka *Value List Handler* (Alur i ostali, 2003). Komunikacija od klijenta do ove komponente sadrži podatke o trenutnoj poziciji na klijentu, broju traženih novih slogova, maksimalnom broju slogova koje klijent može prihvatiti, itd. Ovi podaci pripadaju grupi ne-poslovnih podataka koji se prenose u objektima klase *ASInvocationContext* koji se transformiraju u objekt klase *AS2VlueListInfo*, unutar ove komponente, kako bi ih se moglo ispravno koristiti.

5.4.4.13. Uzorci dizajna na klijentu

Komponente od kojih se sastoje poslovne aplikacije koje se izvode na klijentu dijelimo na tri grupe, ovisno o kanalu pristupa aplikacija za koje su te komponente namijenjene: Java SWT, SmartGWT, i mobilne. Glavni predložak oblikovanja kojeg koristimo pri razvoju tih komponenata je *Model View Controller* (MVC). *Model* predstavljaju sve klase koje služe za prihvat i čuvanje poslovnih podataka u kontekstu korisničkog sučelja; uglavnom se za svaki poslovni entitet kreira jedna klasa ovog tipa. *View* i *Controller* su dio predloška kojeg implementiramo kao jednu klasu koja podatke iz objekata klase *Model* prikazuje u korisničkom sučelju ili prenosi u sloj poslovne logike na poslužitelju. Osim navedenog predloška, na klijentu se najčešće koriste i uzorci oblikovanja *Proxy, Client Request Handler, Command*.



Slika 63 Osnovne klase za razvoj SmartGWT *web* komponenata

5.4.5. Vrste korisničkih formi

Vrste korisničkih sučelja koje primjenjujemo u razvoju poslovnih aplikacija, odabrali smo na temelju korisničkih potreba i zahtjeva koji se najčešće traže od strane poslovnih korisnika. Kako bi omogućili jednostavniji razvoj novih aplikacija, njihovo lakše održavanje, generiranje programskog koda, odabrali smo samo neke od 12 standardnih predložaka za korisnička sučelja poslovnih aplikacija[10]. Predloške: ***Master/Detail***, ***Search/Result***, ***Form***, ***Spredsheet***, implementirali smo u obliku osnovnih komponenata (klasa) okvira referentne arhitekture. Kod razvoja poslovnih aplikacija ili zajedničkih komponenata prezentacijskog sloja, koristimo ih putem nasljeđivanja. Osnovne funkcionalnosti implementirane su u naslijeđenim klasama, dok se za potrebe poslovnih komponenata ili aplikacija te funkcionalnosti mogu preraditi prema zahtjevima korisnika.

[10] http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns

5.4.6. Varijabilnost (engl. *variability*)

Uz pomoć tehnika varijabilnosti određujemo koje od poslovnih komponenata se uključuju ili ne uključuju u pojedinu poslovnu aplikaciju. Osim odabira komponenata, varijabilnost se odnosi i na izbor raspoloživih funkcionalnosti okvira referentne arhitekture, odabir raspoloživih opcija i vanjskog izgleda poslovnih aplikacija, i slično. Mehanizmi realizacije varijabilnosti definirani su na razini referentne arhitekture koja određuje pozicije gdje su varijacije moguće ili nisu. Trenutna inačica predložene referentne arhitekture ne podržava potpuno automatizirano provođenje varijabilnosti, upravljanje varijabilnošću eksplicitno je definirano na razini referentne arhitekture. **Realizacija** varijabilnosti odnosi se na primjenu nekoliko tehnika:

- Prilagodba: tehniku prilagodbe (engl. *adaptation*) koristimo kod implementacije korisničkih sučelja poslovnih aplikacija; svaka aplikacija ima svoje specifičnosti (npr. naziv, logotip, izgled, itd.).
- Proširenje: tehniku proširenja (engl. *extension*) koristimo kod dodavanja novih komponenata klijentskog dijela poslovnih aplikacija; komponente se dodaju prema unaprijed definiranom generičkom sučelju, unosom imena glavne klase (sučelja) komponente u konfiguracijsku datoteku. Datoteka se učitava kod pokretanja aplikacije, naziv klase se koristi za stvaranje objekta komponente i njegovo dodavanje u aplikaciju.
- Zamjena: tehniku zamjene (engl. *replacement*) koristimo najčešće, i to kod odabira mehanizma za pristup podacima (npr. JDBC, JMS), unosimo ime klase (eng. *data access object*) i vrste pristupa u konfiguracijsku datoteku; kod odabira mehanizma za komunikaciju sa serverom (npr. RMI, REST) pomoću konfiguracijskih parametara određujemo način komunikacije, lokaciju servera; kod izbora mehanizma za upravljanje konekcijama sa izvorima podataka; kod izbora formata poslovnih izvješća (npr. Excel, pdf); za izbor mehanizma upravljanja sa transakcijama, itd.

Vrijeme provođenja varijabilnosti odnosi se na dvije tehnike koje koristimo:

- Vrijeme pripreme programskih artefakata za instalaciju (engl. *linking*); koristimo konfiguracijske datoteke koje služe kao ulaz u standardne alate za pripremu.
- Vrijeme pokretanja aplikacija (engl. *loading*); uz pomoć konfiguracijskih datoteka određuju se komponente ili opcije koje su potrebne za rad aplikacija.

Način **reprezentacije** varijabilnosti koji primjenjujemo odnosi se na (engl. *composition-based*) pristup, funkcionalnosti se implementiraju kao ponovno upotrebljive komponente,

jedna komponenta po funkciji. Kod pripreme aplikacija, odabiru se komponente i njihovi konfiguracijski parametri kako bi se dobila izvodiva aplikacija.

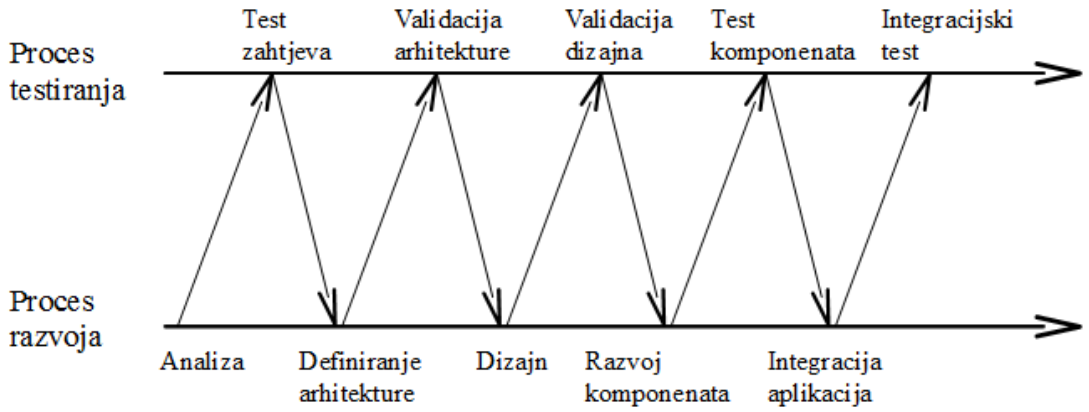
Tehnologija koju primjenjujemo kod provođenja varijabilnosti uključuje:

- Parametre u konfiguracijskim datotekama.
- Predloške (uzorke) oblikovanja (engl. *strategy, template, decorator*).
- Aspektno orijentirano programiranje (engl. *aspects*) koristimo za odabir sveobuhvatnih funkcionalnosti (npr. *logging*).

5.5. Testiranje artefakata

Glavne prednosti uvođenja pristupa linija za proizvodnju softvera u organizacije odnose se na smanjivanje troškova, brži izlazak na tržište i poboljšanje kvalitete softvera. Ove prednosti mogu se postići samo ukoliko su i atributi kvalitete softverskih proizvoda poput pouzdanosti i točnosti od samoga početka u fokusu organizacije koja primjenjuje ovaj pristup. Jedan od načina na koji organizacije postižu zadane atribute kvalitete jeste testiranje artefakata. U kontekstu linija za proizvodnju softvera, testiranje obuhvaća aktivnosti, od validacije zahtjeva na početku do korisničkog testa na kraju. Sam opseg testiranja varira od testiranja cjelokupne linije za proizvodnju softvera, pojedinačnih aplikacija, pa do testiranja individualnih komponenata koje mogu biti dio neke od aplikacija. Budući da troškovi pripreme i izvođenja testiranja mogu biti veliki skoro kao i troškovi razvoja, ponovna upotreba artefakata koji se stvaraju u procesu testiranja može značajno doprinijeti smanjenju ukupnih troškova.

Organizacije koriste razne procese za razvoj poput spiralnog, iterativnog, inkrementalnog; testiranje je potrebno prilagoditi konkretnom procesu za razvoj kojeg organizacija primjenjuje. Proces razvoja generira artefakte koji se u procesu testiranja validiraju, proces testiranja generira testne rezultate koji se ponovno prema potrebi koriste u procesu razvoja. Slika 64 ilustrira povezanost procesa testiranja sa procesom razvoja i njihovu međusobnu ovisnost.



Slika 64 Proces testiranja i razvoja (McGregor, 2001, str. 3)

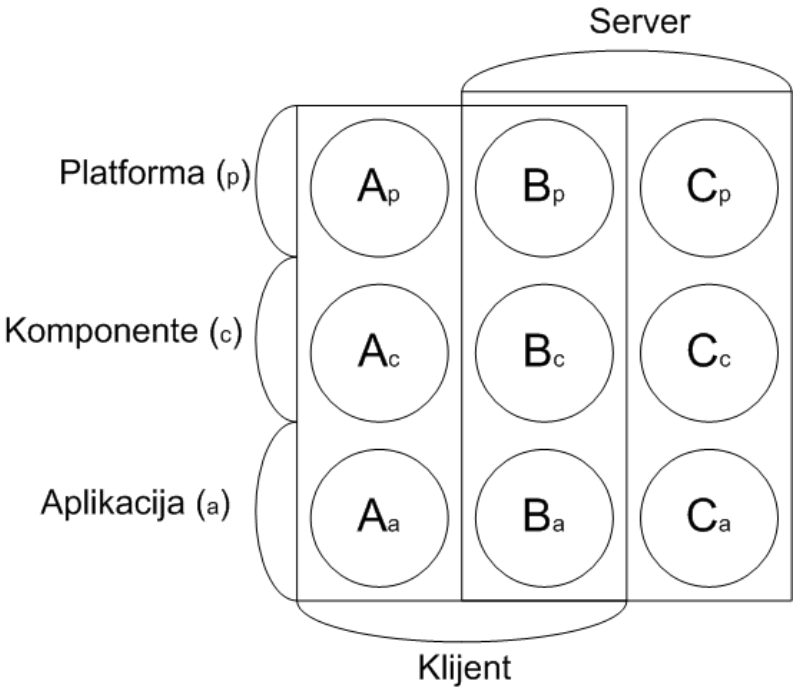
Najvažniji artefakti koji nastaju u procesu testiranja su: planovi testiranja, testni slučajevi (engl. *test cases*), testni podaci, skripte za testiranje, rezultati testiranja i slični drugi artefakti. Planiranje i izvođenje testiranja započinje na razini linije za proizvodnju softvera, zatim se ta razina spušta do aplikacija i na kraju do samih poslovnih komponenata.

Artefakte referentne arhitekture, poslovne komponente i aplikacije testirali smo tek nakon razvoja; najprije programerskim testom svake komponente posebno (engl. *unit test*), zatim integracijskim testom posebno klijentskih a posebno serverskih komponenata, sistemskim testom poslovnih aplikacija; na kraju su krajnji korisnici na temelju pripremljenih planova za testiranje sami proveli korisničko testiranje za sve poslovne aplikacije. Osim navedenih vrsta testova koje redovito provodimo, kod prepravljanja inačice V3 i razvoja nove inačice V4 koja je obuhvatila skoro sve komponente, bilo je potrebno provesti i detaljan regresijski test (engl. *regression test*) postojećih aplikacija i komponenata.

5.6. Korištenje artefakta za razvoj poslovnih aplikacija

Poslovne aplikacije koje koriste referentnu arhitekturu koju predlažemo, sastoje se od elemenata triju glavnih skupova artefakata. Slika 65 prikazuje način sastavljanja poslovnih aplikacija kombiniranjem elemenata triju skupova. Prvi skup se odnosi na okvir referentne arhitekture (platforma) koji se u cijelosti ugrađuje u svaku poslovnu aplikaciju. Drugi skup (komponente) čine komponente koje se na temelju korisničkih zahtjeva odabiru i ugrađuju

pojedinačno u poslovne aplikacije. Treći skup (aplikacija) se odnosi na specifične komponente svake od poslovnih aplikacije koje se ugrađuju samo u jednu poslovnu aplikaciju.



Slika 65 Elementi skupova od kojih se sastoji aplikacija

Platforma predstavlja skup osnovnih komponenata okvira referentne arhitekture koji se u cjelini ugrađuje u svaku poslovnu aplikaciju, dijelimo ga na tri podskupa (podsustava):

- ***Client*** **(A_p)** je podskup kojeg čine elementi pomoću kojih se razvijaju komponente prezentacijskog sloja poslovnih aplikacija: $A_p = \{a_p1, a_p2, a_p3, \dots\}$
- ***Common*** **(B_p)** je podskup kojeg čine elementi pomoću kojih se razvijaju komponente svih slojeva poslovnih aplikacija (prezentacijski, sloj poslovne logike i sloj pristupa podacima): $B_p = \{b_p1, b_p2, b_p3, \dots\}$
- ***Server*** **(C_p)** je podskup kojeg čine elementi pomoću kojih se razvijaju komponente sloja poslovne logike i sloja pristupa podacima: $C_p = \{c_p1, c_p2, c_p3, \dots\}$

Poslovne komponente čini skup elemenata koji se prema izboru, te prema principima varijabilnosti i korisničkim zahtjevima, ugrađuju u poslovne aplikacije, dijelimo ih na tri podskupa:

- ***Client*** **(A_c)** je podskup kojeg čine komponente koje se mogu koristiti u prezentacijskom sloju aplikacija: $A_c = \{a_c1, a_c2, a_c3, \dots\}$
- ***Common*** **(B_c)** je podskup kojeg čine komponente koje se mogu koristiti u svim slojevima aplikacija: $B_c = \{b_c1, b_c2, b_c3, \dots\}$
- ***Server*** **(C_c)** je podskup kojeg čine komponente koje se mogu koristiti u sloju poslovne logike i sloju pristupa podacima: $C_c = \{c_c1, c_c2, c_c3, \dots\}$

Aplikacija predstavlja skup specifičnih komponenata za svaku pojedinu aplikaciju koji se ne koriste u ostalim aplikacijama, ugrađuju se u poslovne aplikacije, dijelimo ih na tri podskupa:

- ***Client*** **(A_a)** je podskup kojeg čine elementi koji se dodaju u prezentacijski sloj poslovnih aplikacija: $A_a = \{a_a1, a_a2, a_a3, \dots\}$
- ***Common*** **(B_a)** je podskup kojeg čine elementi koji se dodaju u prezentacijski i slojeve poslovne logike i pristupa podacima: $B_a = \{b_a1, b_a2, b_a3, \dots\}$
- ***Server*** **(C_a)** je podskup kojeg čine elementi koji se dodaju u slojeve poslovne logike i pristupa podacima: $C_a = \{c_a1, c_a2, c_a3, \dots\}$

Poslovne aplikacije sastoje se od cijelog skupa komponenata platforme, odabranih zajedničkih poslovnih komponenta i odabranih komponenata koje su specifične za svaku pojedinu poslovnu aplikaciju. Odabrane poslovne komponente mogu se prikazati pomoću izraza:

$$A'_c = \{a_{ci} \in A_c : a_{ci}\ odabrane\ komponente - klijent\}$$

$$B'_c = \{b_{ci} \in B_c : b_{ci}\ odabrane\ komponente - klijent\ i\ server\}$$

$$C'_c = \{c_{ci} \in C_c : c_{ci}\ odabrane\ komponente - server\}$$

Odabrane specifične komponente za svaku aplikaciju mogu se prikazati pomoću izraza:

$$A'_a = \{a_{ai} \in A_a : a_{ai}\ odabrane\ komponente - klijent\}$$

$$B'_a = \{b_{ai} \in B_a : b_{ai}\ odabrane\ komponente - klijent\ i\ server\}$$

$$C'_a = \{c_{ai} \in C_a : c_{ai}\ odabrane\ komponente - server\}$$

U konačnici svaka poslovna aplikacija koja se temelji na predloženom konceptu i referentnoj arhitekturi sastoji se samo od elemenata koji su potrebni za funkcioniranje aplikacije, te se može prikazati pomoću izraza:

$$PA = (A_p \cup B_p \cup C_p) \cup ({A'}_c \cup {B'}_c \cup {C'}_c) \cup ({A'}_a \cup {B'}_a \cup {C'}_a)$$

Dio aplikacije koji se izvodi na klijentu može se prikazati izrazom:

$$PAC = (A_p \cup B_p) \cup ({A'}_c \cup {B'}_c) \cup ({A'}_a \cup {B'}_a)$$

Dio aplikacije koji se izvodi na serveru može se prikazati izrazom:

$$PAS = (B_p \cup C_p) \cup ({B'}_c \cup {C'}_c) \cup ({B'}_a \cup {C'}_a)$$

Konfiguracija varijabilnosti za pojedinu aplikaciju izvodi se u vrijeme pakiranja aplikacije na temelju odabira komponenata koji je zapisan u *XML* datoteci. Odabir je također moguće uraditi uz pomoć alata za konfiguraciju proizvoda, međutim, inačica V4 referentne arhitekture nije još prilagođena niti jednom od postojećih alata za konfiguraciju.

5.6.1. Razvoj i arhitektura poslovnih aplikacija

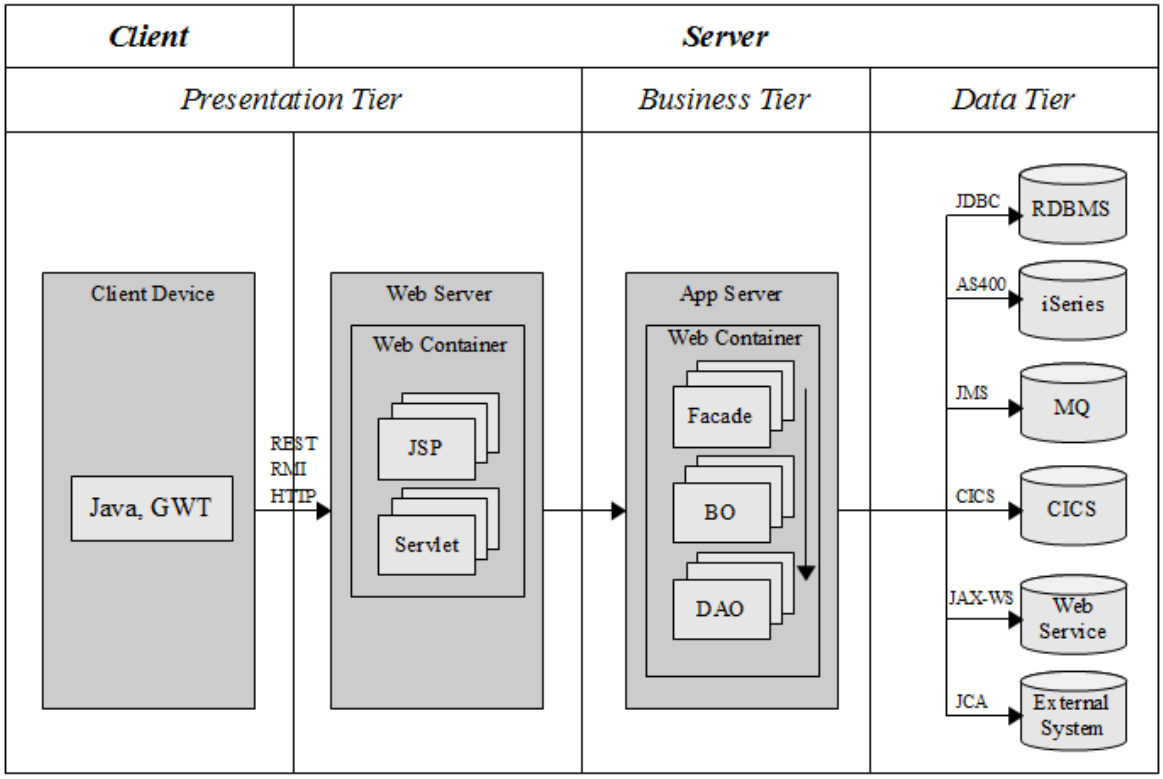
Razvoj poslovnih aplikacija koje se razvijanju na temelju predložene referentne arhitekture započinje analizom korisničkih zahtjeva. Razvojni programeri poslovnih aplikacija isključivo se bave poslovnim problemima, rješavaju ih razvojem poslovnih komponenata i aplikacija; neke od poslovnih komponenta postaju djelom linije za proizvodnju softvera, dok se artefakti koji su specifični za pojedine poslovne aplikacije ne dodaju u repozitorij poslovnih komponenata. Nove komponente razvijaju se samo ukoliko ne postoje komponente koje zadovoljavaju korisničke zahtjeve u repozitoriju raspoloživih komponenata. Izgled aplikacija, izvori podataka, komunikacijski kanali, postojeće poslovne komponente, i mnoge druge karakteristike definirane su kao točke varijabilnosti, nije ih potrebno programirati već se njihova primjena definira putem konfiguracije za svaku poslovnu aplikaciju. Korisnička sučelja novih komponenata, nove komponente i specifičnosti pojedine poslovne aplikacije predmet su razvoja kojeg obavljaju razvojni programeri poslovnih aplikacija za vrijeme projekata razvoja svake pojedine aplikacije, najčešće korištenjem tehnike generiranja programskog koda.

Arhitektura poslovnih aplikacija

Poslovne aplikacije nasljeđuju svoju arhitekturu od referentne arhitekture poslovnih aplikacija (RABA). Svaka specifičnost neke poslovne aplikacije oblikuje se najprije na razini aplikacije, nakon provjere može postati dio referentne arhitekture, kako bi je mogle koristiti i ostale poslovne aplikacije. Međutim, dizajn poslovnih aplikacija uglavnom se odnosi na oblikovanje onih dijelova aplikacije koji sadrže poslovne podatke i druge karakteristike koje su specifične za samo jednu poslovnu aplikaciju. Predložena arhitektura poslovnih aplikacija je fleksibilna,

ne ograničava primjenu specifičnih rješenja, međutim, postavljene granice nije preporučljivo prelaziti osim ako se radi o komponentama, potencijalnim kandidatima koje će se u budućnosti dodati među artefakte referentne arhitekture.

Slika 66 prikazuje glavne elemente oblikovanja poslovnih aplikacija koje koriste predloženu referentnu arhitekturu za poslovne aplikacije (RABA).



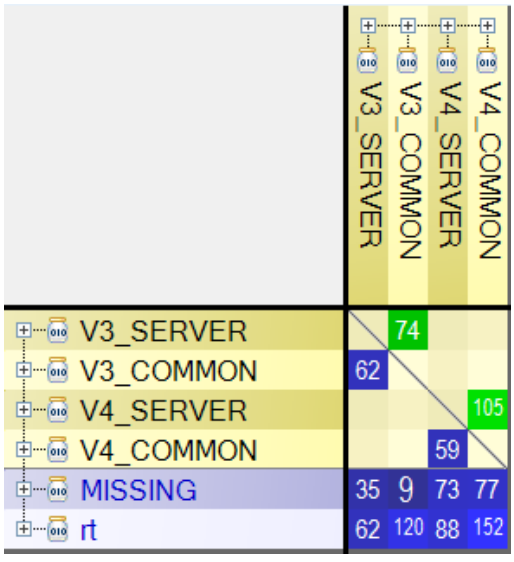
Slika 66 Predložena arhitektura poslovnih aplikacija

Poslovne aplikacije sastoje se od komponenata korisničkog sučelja koje se definiraju prema predlošku *Model View Controller* (MVC), sučelja poslovnih komponenata (*Facade*), poslovnih objekata za primjenu poslovne logike *Business Object* (BO), klasa za pristup izvorima podataka *Data Access Object* (DAO), klasa za prijenos podataka, grafičkih elemenata. Ostali elementi poslovnih aplikacija, uglavnom nisu predmet rada razvojnih programera poslovnih aplikacija, već su eventualno dio procesa razvoja okvira referentne arhitekture i ostalih ponovno upotrebljivih komponenata.

5.7. Okvir referentne arhitekture (V4)[11]

Inačica V4 okvira referentne arhitekture sastoji se od 5 podsustava: *Common, Server, Client* SWT, *Client* GWT, *Client Mobile*. Razvoj i primjena okvira referentne arhitekture u financijskoj instituciji kontinuirani je proces koji je uvjetovan prvenstveno korisničkim zahtjevima a zatim i tehnološkim uvjetima. Inačica V3 koristila se u 9 poslovnih aplikacija, nova inačica V4 koja uključuje nove *Web* i mobilne komponente, koristi se u 16 poslovnih aplikacija unutar financijske institucije u kojoj provodimo ovo istraživanje.

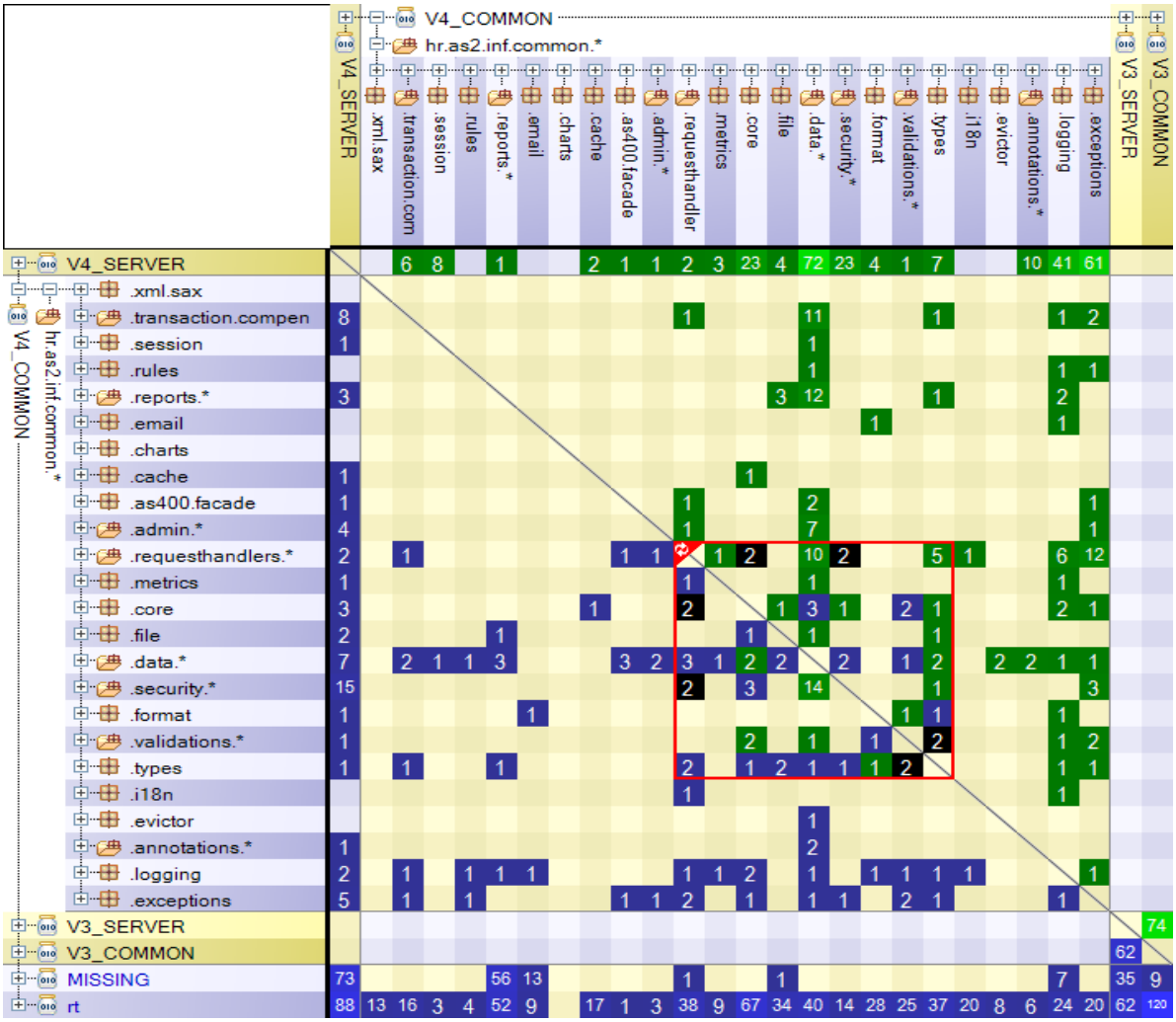
Slika 67 prikazuje ukupni broj poveznica podsustava *Common* i *Server* između sebe i prema vanjskim („*missing*“) komponentama te prema Java okružju („*rt*“). Broj poveznica se značajno povećao u inačici V4 u odnosu na inačicu V3, tako se primjerice broj poveznica od *Common* podsustava prema vanjskim komponentama od inačice V3 do inačice V4 povećao od 9 na 77. Povećanje broja poveznica rezultat je povećanja funkcionalnosti nove inačice okvira, uvođenja novih vanjskih komponenata, „preuzimanja“ interakcija prema vanjskim komponentama od poslovnih komponenata, što je i bio cilj prepravljanja postojeće inačice i razvoja novih funkcionalnosti u inačici V4.



Slika 67 DSM podsustavi *Common* i *Server* (V3 i V4)

[11] Izvorni programski kod okvira referentne arhitekture (RABA) nalazi se na slijedećim lokacijama:
https://github.com/zrosko/AS2-PLATFORM-SERVER.git
https://github.com/zrosko/AS2-PLATFORM-COMMON.git
https://github.com/zrosko/AS2-PLATFORM-CLIENT-GWT.git
https://github.com/zrosko/AS2-REFERENCE-APP.git

Slika 68 prikazuje rezultat statičke analize povezanosti komponenata podsustava (*Common* V4) okvira referentne arhitekture. U odnosu na izgled DSM strukture inačice V3 (slika 42) primjetno je povećane ukupnog broja komponenata od 14 na 24.

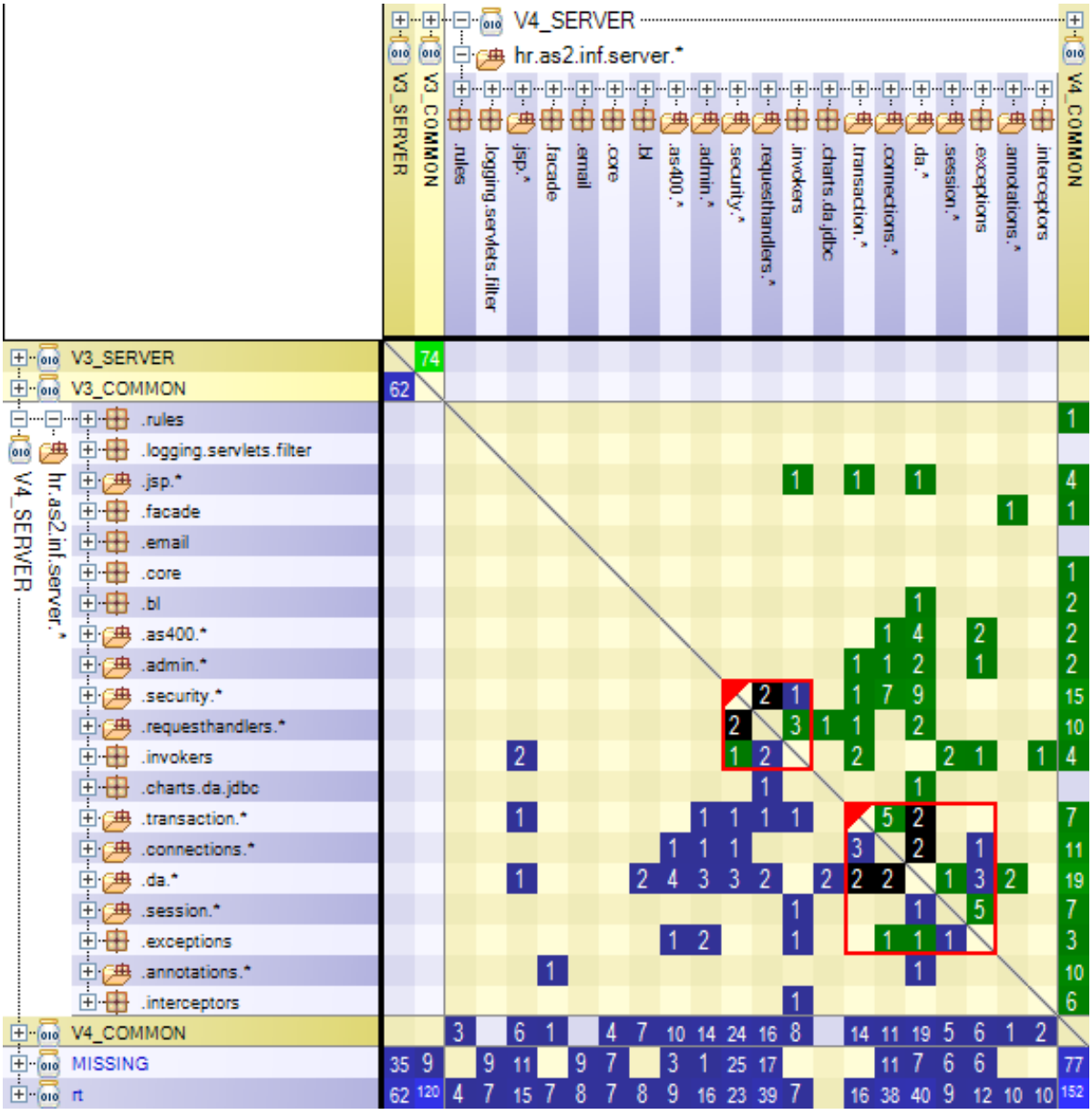


Slika 68 DSM podsustava *Common* (V4)

Također, broj međusobno povezanih komponenata je smanjen sa **9** na **3**. Osim apsolutnog broja, taj je broj zbog povećanja broja komponenata relativno još manji, što predstavlja značajno poboljšanje u odnosu na prethodnu inačicu V3.

Slika 69 prikazuje rezultat statičke analize povezanosti komponenata podsustava (*Server* V4) okvira referentne arhitekture. Broj kombinacija međusobno povezanih komponenata je 3, jednako kao i u inačici V3, što ukazuje na relativno poboljšanje budući da je broj komponenta porastao sa 16 na 20. Za razliku od inačice V3 koja je imala jednu kružnu (engl. *cycle*)

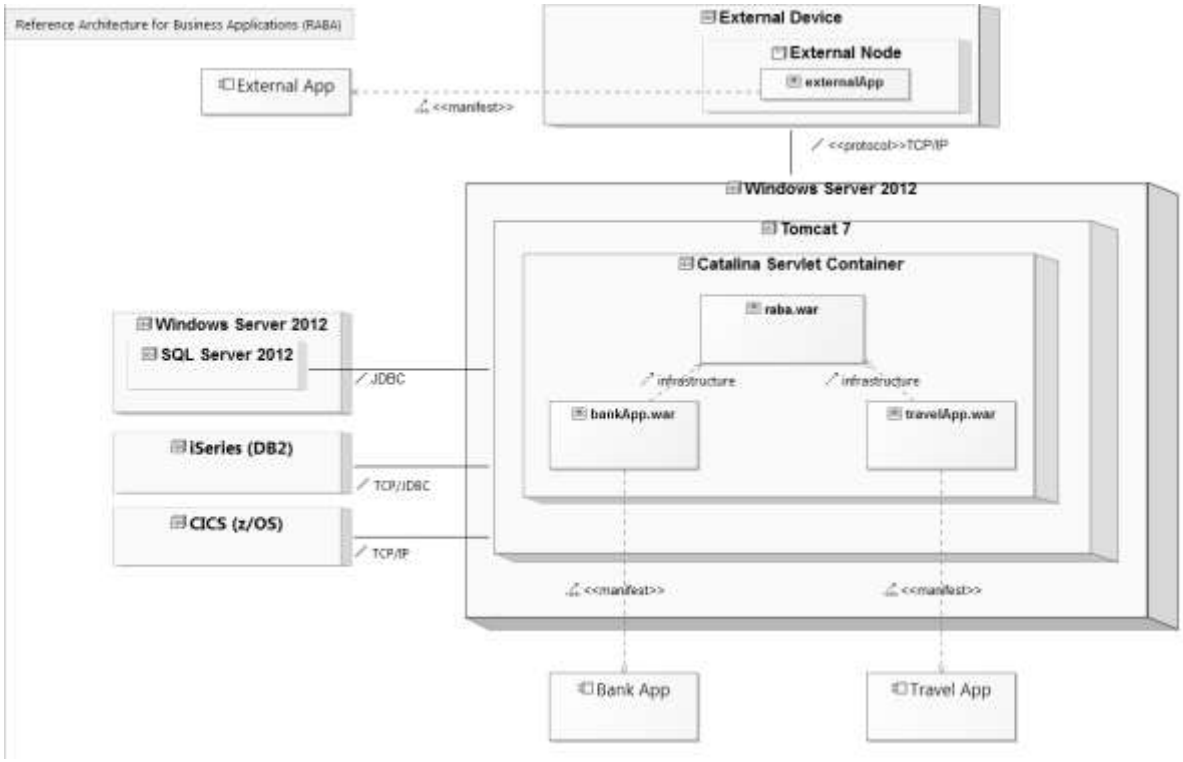
povezanost komponenata, inačica V4 ima dvije. Komponente koje su kružno povezane potrebno je u novoj inačici prepraviti kako bi se dobile još bolje karakteristike slojne arhitekture.



Slika 69 DSM podsustava *Server* (V4)

Poslovne aplikacije koje su razvijene primjenom predložene referentne arhitekture izvode se djelomično na poslužitelju, djelomično na računalu poslovnog korisnika. Aplikacije iz više poslovnih sektora moguće je instalirati na istom poslužitelju, primjerice aplikacije za bankarski sektor i aplikacije za sektor turizma ili prijevoza mogu se izvoditi na istom poslužitelju.

Slika 70 ilustrira primjer konteksta implementacije (engl. *deployment view*) predloženog sustava poslovnih aplikacija za dva poslovna sektora koji koriste referentnu arhitekturu (RABA): operativni sustav (Windows Server 2012), izvore podataka (DB2, CICS, SQL Server 2012), aplikacijski poslužitelj (Tomcat 7), odnos sa vanjskim komponentama koje se najčešće koriste u primjeni predložene referentne arhitekture. Dva poslovna sektora (*bankApp.war*, *travelApp.war*) koriste zajedničku referentnu arhitekturu (*raba.war*); sustav je otvoren za kolaboraciju sa vanjskim sustavima, omogućen je paralelan rad dvaju (po potrebni i više) poslovnih sektora.



Slika 70 Primjer primjene okvira referentne arhitekture (RABA)[12]

U primjeni je troslojna arhitektura poslovnih aplikacija koja nam omogućava kontakt sa neograničenim brojem korisnika, upravljanje sa lokacijama na kojima će se izvoditi poslovna logika (engl. *load balancing*), upravljanje sa iznimkama kod rada poslužitelja (engl. *failover*). Isto tako arhitektura u primjeni omogućava nam jednostavnu migraciju na različite platforme i zamjenu vanjskih komponenata koje se koriste u poslovnim aplikacijama.

[12] Izvorni dijagram – prema predlošku http://www.uml-diagrams.org

5.8. Eclipse *plug-in* za razvoj aplikacija

Produktivnost razvoja poslovnih aplikacija značajno se može ubrzati uz pomoć alata koji prema definiranim predlošcima stvaraju strukturu poslovnih aplikacija u razvojnom okruženju i generiraju izvorni programski kod na temelju raspoloživih poslovnih modela i njihovih meta podataka, prema principima i ograničenjima predložene referentne arhitekture. Razvili smo pomoćni alat (engl. *plug-in*) koji je sukladan razvojnom okruženju *Eclipse* pomoću kojega na temelju modela baze podataka ili pravila i ograničenja referentne arhitekture, generiramo klase i druge artefakte poslovnih aplikacija. Generirani artefakti odnose se na klase za pristup podacima, sučelja komponenata, klase za korisničko sučelje, klase za prijenos podataka (engl. *value objects*), konstante. Pomoćni alat koristi tehniku *Automatic Programming* (AP) i neke elemente tehnike *Generative Programming* (GP); u primjeni se koristi *Apache Velocity Engine* 1.7 kao generator programskog koda na temelju unaprijed definiranih predložaka. Slika 71 ilustrira primjer predloška za generiranje klasa za prijenos i čuvanje poslovnih podataka (engl. *value objects*) na temelju meta podataka poslovnog modela.

```
/**
 * Copyright: $copyright
 **/
#set($Vo = "Vo")
#set($Rs = "Rs")
#set($extendsVo = "AS2Record")
#set($extendsRs = "AS2RecordList")

#if ($type.equals("vo"))
import hr.as2.inf.common.data.AS2Record;

public class $class$Vo extends $extendsVo {
private static final long serialVersionUID = 1L;
#foreach( $column_name in $meta_data )
public static String ${table_name}__${column_name} = "$column_name";
#end
public $class$Vo() {
        super();
}
public $class$Vo($extendsVo value) {
        super(value);
}
//Getters
#foreach( $column_name in $meta_data )
public String get${column_name.substring(0,1)}${column_name.substring(1).replaceAll("_","")}() {
        return getAsString(${table_name}__${column_name});
}
#end
//Setters
#foreach( $column_name in $meta_data )
public void set${column_name.substring(0,1)}${column_name.substring(1).replaceAll("_","")}(String
value) {
        set(${table_name}__${column_name}, value);
}
#end
}
#end

#if ($type.equals("rs"))
```

```
import hr.as2.inf.common.data.AS2Record;
import hr.as2.inf.common.data.AS2RecordList;

public class $class$Rs extends $extendsRs {
private static final long serialVersionUID = 1L;

public $class$Rs() {
        super();
}
public $class$Rs(AS2RecordList set) {
        super();
        setColumnNames(set.getColumnNames());
        setColumnSizes(set.getColumnSizes());
        setMetaData(set.getMetaData());
        for (AS2Record row : set.getRows()) {
                $class$Vo vo = new $class$Vo(row);
                this.addRow(vo);
        }
}
}
#end
```

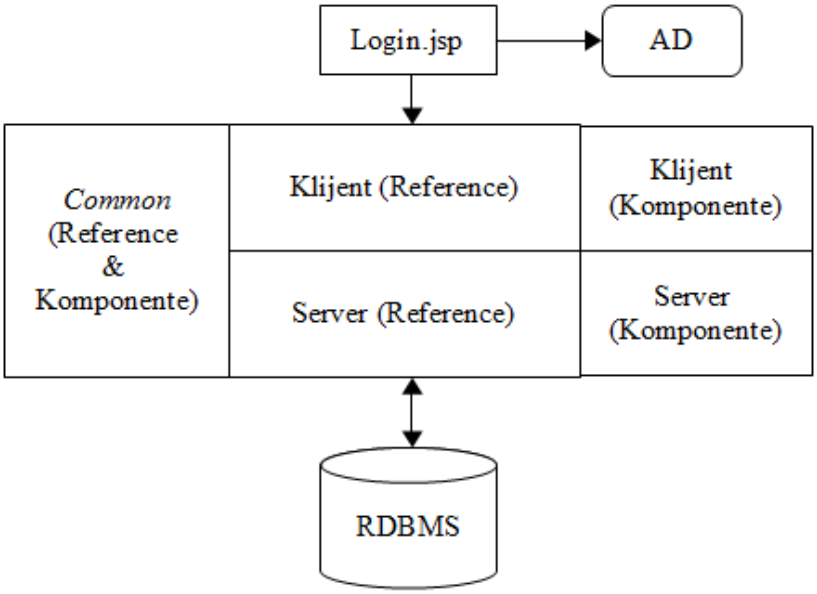
Slika 71 Predložak za generiranje programskog koda

Generiranje izvornog koda koristi meta podatke poslovnog modela kojeg je prethodno potrebno definirati u obliku tablica u relacijskoj bazi podataka. Pomoćni alat čita meta podatke koji su raspoloživi za tablice za koje se generira programski kod, koristi ih za generiranje nekoliko vrsta klasa koje na taj način postaju dio poslovnih komponenata ili poslovnih aplikacija. Pomoćni alat nije predmet validacije, nismo ga koristili kao artefakt kod provjere upotrebljivosti referentne arhitekture.

5.9. Referentna aplikacija

Predloženu referentnu arhitekturu opisali smo na apstraktan način, uglavnom pomoću uzoraka dizajna i detaljnog opisa njihove primjene u oblikovanju okvira referentne arhitekture. Razvoj poslovnih aplikacija samo na temelju takvog opisa arhitekture dosta je zahtjevan, stoga smo u cilju olakšavanja razvoja implementirali referentnu aplikaciju kao jedan od artefakata predložene referente arhitekture; početnu točku koja nam služi kao predložak za razvoj novih poslovnih aplikacija. Referentna aplikacija također nam služi za provjeru i testiranje novih funkcionalnosti i tehnoloških inovacija u okviru domenskog inženjerstva, te za edukaciju novih razvojnih programera. Referentna aplikacija može se lako pokrenuti u razvojnom okruženju, pomoću nje prikazujemo primjere najčešće korištenih funkcionalnosti, vrste podržanih korisničkih sučelja, raspoložive opcije za izgled poslovnih aplikacija, implementaciju poslovne logike i logike pristupa podacima. Slika 72 prikazuje strukturu referentne aplikacije koju koristimo kao model ili predložak za generiranje novih poslovnih aplikacija. Pri razvoju novih aplikacija, uz pomoć alata za generativno programiranje (Eclipse plug-in) artefakti referentne aplikacije se prepravljaju i zamjenjuju sa artefaktima nove aplikacije. Referentna aplikacija se

sastoji od standardnih dijelova, klijentskih komponenata, serverskih komponenata, zajedničkih (*common*) komponenata. Za pristup aplikaciji potrebno je napraviti prijavu u sigurnosni sustav Active Directory (AD), poslužiteljske komponente pristupaju podacima i spremaju ih u relacijsku bazu podataka (RDBMS).



Slika 72 Struktura referentne aplikacije

Slika 73 prikazuje primjer implementacije varijabilnosti i konfiguraciju poslužiteljskog dijela referentne aplikacije. Varijabilnost se odnosi na odabir strukturnih elemenata (komponenata) iz repozitorija poslužiteljskih komponenata od kojih se sastoji poslužitelj referentne aplikacije. Varijabilnost se implementira u vrijeme pakiranja (engl. *linking time*) poslužiteljskog dijela aplikacije. Sličan postupak se odnosi i na konfiguriranje klijentskog dijela referentne aplikacije.

```
<selectedElements exportClassFiles="true" exportOutputFolder="false">
  <javaElement handleIdentifier="=APP REFERENCE APPLICATION SERVER"/>
  <javaElement handleIdentifier="=AS2 PLATFORM SERVER"/>
  <javaElement handleIdentifier="=AS2 PLATFORM COMMON"/>
  <javaElement handleIdentifier="=hr.as2.app.common.ref"/>
  <javaElement handleIdentifier="=hr.as2.app.server.ref"/>
  <javaElement handleIdentifier="=hr.as2.app.server.services"/>
  <javaElement handleIdentifier="=hr.as2.app.server.resources"/>
</selectedElements>
```

Slika 73 Implementacija varijabilnosti za poslužitelj referentne aplikacije

6. VALIDACIJA

U ovom poglavlju sustavno iznosimo dobivene rezultate istraživanja, interpretiramo dobivene rezultate, naglašavamo na koji način ovo istraživanje pridonosi pomaku u području razvoja poslovnih aplikacija pristupom linija za proizvodnju softvera; opisuje se postupak i analiziraju rezultati: anketnog upitnika za provjeru upotrebljivosti artefakata referentne arhitekture, studije slučaja primjene referentne arhitekture u jednoj financijskoj instituciji, poboljšanja postojećih artefakata na temelju metrika izvornog programskog koda, provjere i potvrde valjanost novih metrika i predloženog modela za mjerenje utjecaja referentne arhitekture na održavljivost poslovnih komponenata, procjene zrelosti predložene referentne arhitekture (RABA) prema FEF metodi za zrelost linija za proizvodnju softvera.

Ovim poglavljem obuhvaćeni su slijedeći ciljevi ovog istraživanja: (C3) provjeriti razvijene artefakte referentne arhitekture i njihovu korisnost u praksi, (C4) provjeriti povezanost rezultata PR i MI modela za mjerenje održavljivosti aplikacijskih komponenata linije za proizvodnju softvera.

6.1. Odgovori na istraživačka pitanja i provjera hipoteza

Odgovore na istraživačka pitanja **P1** (*Koje funkcionalne zahtjeve treba zadovoljiti referentna arhitektura za poslovne aplikacije prema pristupu linija za proizvodnju softvera?)* i **P2** (*Može li se razviti referentna arhitektura koja zadovoljava te zahtjeve?*) dobili smo u prethodnom poglavlju (5). U ovom poglavlju opisujemo postupak i rezultate provjere valjanosti artefakata referentne arhitekture kako bi dobili odgovor na pitanje **P3** (*Da li je referentna arhitektura koja zadovoljava te zahtjeve korisna u praksi?*). Prikupljene kvantitativne podatke metrika izvornog programskog koda koristimo kako bi dobili odgovor na pitanje **P4** (*Postoji li povezanost između predloženog modela PR za mjerenje održavljivosti linije za proizvodnju softvera za poslovne aplikacije i standardnog modela MI, kada se oba modela koriste za mjerenje održavljivosti identičnih softverskih komponenata?*). Cilj nam je pokazati da je predložena referentna arhitektura korisna u praksi. Provjera korisnosti odnosi se na provjeru upotrebljivosti (engl. *usability*) i na provjeru primjenjivosti (engl. *utility*). Osim provjere korisnosti referentne arhitekture, drugi cilj nam je provjeriti povezanost rezultata predloženog PR modela sa standardnim MI modelom za mjerenje održavljivosti softverskih komponenata. Za validaciju istraživačkih pitanja P3 i P4 definirali smo dvije hipoteze:

(Hipoteza **H1**) *Nova referentna arhitektura prema pristupu linija za proizvodnju softvera je korisna za proizvodnju softvera za poslovne primjene.*

Za provjeru točnosti ove hipoteze, istražili smo **upotrebljivost** artefakata referentne arhitekture ispitivanjem njihovih korisnika, razvojnih programera. Korisnici su programeri koji su razvijali aplikacije na temelju referentne arhitekture, u tri različite organizacije (financijska institucija, telekomunikacijska tvrtka, softverska tvrtka za razvoj softvera iz područja geoinformatike). Ispitivanje je obavljeno u dvije organizacije koje koriste istu inačicu (V3) referentne arhitekture, te u jednoj organizaciji koja koristi poboljšanu (V4) inačicu. Cilj ovog ispitivanja je bio da se pokaže kako se artefakti u principu mogu koristiti za razvoj poslovnih aplikacija.

Jedini način da se dokaže korisnost predloženog pristupa za proizvodnju softvera za poslovne primjene jeste da se artefakti referentne arhitekture **primjenjuju** u praksi, da se podaci njihove primjene prikupe i prezentiraju. Stoga smo proveli longitudinalnu studiju slučaja upotrebe naše referentne arhitekture za poslovne aplikacije u jednoj financijskoj instituciji.

(Hipoteza **H2**) *Rezultati određivanja održavljivosti poslovnih komponenata (engl. Maintainability) informacijskog sustava metrikom „odgovornosti platforme" (engl. Platform Responsibility) podudaraju se s rezultatima dobivenim uz pomoć alternativne metrike.*

Operacionalizacija i provjera ove hipoteze temelji se na podacima koje smo prikupili za vrijeme provođenja navedene longitudinalne studije slučaja u financijskoj instituciji. Podaci koje smo koristili za provjeru točnosti ove hipoteze odnose se na aplikacijske poslovne komponente i metrike njihovog izvornog programskog koda.

6.2. Provjera upotrebljivosti referentne arhitekture

Prema pravilima za izvođenja laboratorijskog eksperimenta (Wohlin i ostali, 2003), izveli smo mjerenje upotrebljivosti referentne arhitekture. Cilj mjerenja upotrebljivosti (engl. *usability*) je dokazivanje da je referentna arhitektura u dovoljnoj mjeri upotrebljiva (H1). U kontekstu ovog mjerenja "dovoljno" znači da upotrebljivost referentne arhitekture ne sprječava namjere njene upotrebe u praksi. Mjerenje smo proveli ispitivanjem **5** razvojnih programera koji su koristili referentnu arhitekturu za razvoj poslovnih aplikacija u različitim poslovnim sektorima. Programeri su dobili upitnik sa pitanjima koja se odnose na upotrebljivost artefakata referentne arhitekture. Svi programeri imaju visoku stručnu spremu iz područja razvoja softvera, koristili su referentnu arhitekturu više od godinu dana i razvili najmanje jedan proizvod za krajnje korisnike. Za traženje odgovora na općenita pitanja o upotrebljivosti referentne arhitekture

koristili smo upitnik kojeg smo pripremili prema dobro poznatim metodama: *Nielsen's Attributes of Usability* (Nielsen, 1994), *Perceived Usefulness, Ease of Use* (F. D. Davis, 1989), te prema *Computer Systems Usability Questionnaire* (Lewis, 1995).

6.2.1. Anketni upitnik

Najprije smo definirali glavna pitanja u upitniku za provjeru upotrebljivosti artefakata referentne arhitekture; prema predlošcima postojećih upitnika za upotrebljivost[13]. Upitnik za provjeru upotrebljivosti se sastoji od četiri glavne grupe istraživačkih pitanja: prva grupa sa 5 pitanja iz područja upotrebljivosti, dvije grupe od po 6 pitanja za korisnost i lakoću korištenja, te grupu od 19 pitanja za sveobuhvatnu upotrebljivost. Prema (Nielsen, 1994), "*upotrebljivost je atribut kvalitete kojim se procjenjuje lakoća upotrebe korisničkog sučelja koji se dijeli u pet područja kakvoće: lakoća učenja, učinkovitost, memorabilnost, pogreške (točnost), i subjektivno zadovoljstvo*". Ovih pet područja kakvoće nazivamo *Nielsen's Attributes of Usability* (NAU). U upitniku za razvojne programere, korisnike artefakata, definirali smo slijedećih pet pitanja na temelju atributa kakvoće koje je definirao (Nielsen, 1994):

- *S kojom lakoćom korisnik može naučiti koristiti referentnu arhitekturu*? Ovo pitanje se odnosi na lakoću sa kojom korisnik može napraviti jednostavan programski zadatak ako koristi artefakte referentne arhitekture po prvi put.
- *Koliko efikasno korisnik može koristiti referentnu arhitekturu*? Efikasnost označava brzinu kojom korisnik (programer) može izvršiti programske zadatke korištenjem referentne arhitekture jednom kada je savladao njene osnovne funkcionalnosti. Procjena efikasnosti temelji se na procjeni "brzine" koja se može različito tumačiti, stoga se smatra da je najbolje pitati korisnika da li je korištenje artefakata „dovoljno brzo" kako bi se izvršio programski zadatak. Pri tome je važno da korisnik ima na umu relativnu brzinu u odnosu na uvjete kada se ne koristi referentna arhitektura ili kada se koristi alternativna referentna arhitektura.
- *Je li lako zapamtiti način korištenja referentne arhitekture*? Lakoća pamćenja odnosi se na lakoću sa kojom korisnik može izvršiti zadatak nakon jednog vremenskog razdoblja u kojem nije koristio referentnu arhitekturu.

[13] http://hcibib.org/perlman/question.html

- *Da li referentna arhitektura uzrokuje samo mali broj i ne previše značajnih grešaka*? Područje kakvoće koje se odnosi na pogreške (točnost) mjerimo brojem pogrešaka za vrijeme korištenja artefakata i lakoću sa kojom se greške mogu popraviti. Mjerenje nije jednostavno jer uzrok grešaka nije uvijek referentna arhitektura, već greške mogu biti rezultat pogrešnog načina korištenja referentne arhitekture od strane korisnika. Mjerenje "malog broja" ne previše značajnih grešaka je u velikoj mjeri subjektivno, stoga je za smanjivanje subjektivnosti potrebno pitati programere da li su zadovoljni sa učestalošću grešaka i načinom upravljanja s greškama od strane artefakata referentne arhitekture.
- *Da li su korisnici subjektivno zadovoljni korištenjem referentne arhitekture*? Subjektivno zadovoljstvo opisuje osjećaj zadovoljstva pri korištenju artefakata. Jedini način mjerenja ovog atributa referentne arhitekture je postavljanjem pitanja korisnicima na način da kažu svoje mišljenje.

Za mjerenje upotrebljivosti korištenjem NAU atributa definirali smo pet pitanja: Lakoća učenja? Efikasnost? Memoriranje? Greške? Subjektivno zadovoljstvo? Za svako pitanje pripremili smo objašnjenje značenja atributa kakvoće koje smo prezentirali i objasnili programerima koji su odgovarali na pitanja iz upitnika. Odgovore na pitanja smo mjerili intervalnom skalom: -2: vrlo loše, -1: loše, 0: u redu, 1: dobro, 2: vrlo dobro.

Osim pitanja koja smo definirali prema (Nielsen, 1994), koristili smo i drugu grupu pitanja koja se odnose na dvije varijable za koje se tvrdi (F. D. Davis, 1989) da predstavljaju osnove za korisničko iskustvo: korisnost prema percepciji korisnika (engl. p*erceived usefulness*), lakoća upotrebe prema percepciji korisnika (engl. *perceived ease of use*) (PUEU). Korisnost prema percepciji korisnika se odnosi na procjenu da li referentna arhitektura omogućava razvojnom programeru da radi brže, povećava produktivnost, povećava učinkovitost, općenito, da li olakšava rad razvojnih programera. Lakoća upotrebe prema percepciji korisnika se odnosi na procjenu da li je referentna arhitektura jednostavna za shvatiti, kontrolirati, jasna i razumljiva, fleksibilna, općenito - jednostavna za korištenje.

Na temelju PUEU definirali smo dva nova pitanja.

- *Kakva je korisnost referentne arhitekture prema percepciji korisnika*? U upitniku smo definirali šest potpitanja.
- *Kakva je lakoća korištenja referentne arhitekture prema percepciji korisnika*? Za odgovor na ovo pitanje definirali smo također šest potpitanja.

Odgovore na sva pitanja PUEU mjerili smo intervalnom skalom:

-2: izrazito se ne slažem, -1: ne slažem se, 0: niti se slažem niti se ne slažem, 1: slažem se, 2: izrazito se slažem.

Treću grupu pitanja u upitniku definirali smo prema *Computer System Usability Questionnaire* (CSUQ) kojeg je predložio (Lewis, 1995). Pitanja se odnose na objektivnu i subjektivnu procjenu upotrebljivosti artefakta referentne arhitekture. Neka pitanja u standardnom upitniku izričito se odnose na korisničko sučelje kompjuterskog sustava. U pojašnjenju pitanja koja smo prethodno dostavili programerima, korisničko sučelje referentne arhitekture smo definirali kao API (engl. *application programing interface*) za povezivanje referentne arhitekture i aplikacija ili poslovnih komponenata koje programeri izrađuju. Na temelju predloška CSUQ definirali smo 19 pitanja u upitniku. Odgovore na sva pitanja CSUQ mjerili smo intervalnom skalom:

-2: izrazito se ne slažem, -1: ne slažem se, 0: niti se slažem niti se ne slažem, 1: slažem se, 2: izrazito se slažem.

Grupa pitanja CSUQ nam je pomogla da dobijemo odgovor na pitanje:

- *Kakva je sveukupna upotrebljivost referentne arhitekture?*

Tablica 17 Upitnik za testiranje upotrebljivosti referentne arhitekture (RA)

Metoda	**Istraživačko pitanje**	**Pitanje u upitniku** (prema http://hcibib.org/perlman/question.html)
NAU	*S kojom lakoćom korisnik može naučiti koristiti referentnu arhitekturu*?	Lakoća učenja
	Koliko efikasno korisnik može koristiti referentnu arhitekturu?	Efikasnost?
	Je li lako zapamtiti način korištenja referentne arhitekture?	Memoriranje?
	Da li referentna arhitektura uzrokuje samo mali broj i ne previše značajnih grešaka?	Greške?
	Da li su korisnici subjektivno zadovoljni korištenjem referentne arhitekture?	Subjektivno zadovoljstvo?
PUEU	*Kakva je korisnost referentne arhitekture prema percepciji korisnika*?	Korištenje RA omogućava mi da brže završavam programske zadatke
		Korištenje RA omogućava mi da povećam svoj učinak u poslu
		Korištenje RA omogućava mi veću produktivnost
		Korištenje RA omogućava mi veću efektivnost u poslu
		Korištenje RA mi omogućava lakše obavljanje programskih zadataka
		RA je korisna u obavljanju moga posla
		Učenje kako koristiti RA je jednostavno

	Kakva je lakoća korištenja referentne arhitekture prema percepciji korisnika?	Nije teško pronaći način kako iskoristiti RA
		Korištenje RA je jasno i razumljivo
		RA je fleksibilna za korištenje
		Lako je postati vješt u korištenju RA
		RA jednostavna za korištenje
CSUQ	*Kakva je sveukupna upotrebljivost referentne arhitekture*?	Sve u svemu, zadovoljan sam s lakoćom korištenja RA
		RA je jednostavna za korištenje
		Mogu učinkovito završiti svoj rad pomoću RA
		Brže završavam zadatke ukoliko koristim RA
		Efikasnije završavam zadatke ukoliko koristim RA
		Osjećam se komotno koristiti RA
		Nije teško naučiti koristiti RA
		Brzo sam postavo produktivan korištenjem RA
		Greške koje javlja RA jasno ukazuju kako riješiti problem
		Kada pogriješim kod korištenja RA, brzo i lako to popravim
		Dokumentacija RA je jasna
		Jednostavno je pronaći informacije koje zatrebam
		Dokumentacija RA je razumljiva
		Dokumentacija mi efektivno pomaže završiti zadatke
		Organizacija dokumentacije RA je jasna
		Sučelje RA je zgodno
		Sviđa mi korištenje sučelja RA
		RA ima sve funkcionalnosti koje su potrebne
		Sve u svemu, zadovoljan sam sa RA

6.2.2. Postupak

Pripremljeni upitnik smo proslijedili ispitanicima uz pomoć javno dostupnog servisa (*docs.google.com*). Osnovne pojmove iz područja upotrebljivosti i pitanja iz upitnika su programerima pismeno i djelomično usmeno, razjašnjena od strane autora ove disertacije. Ispitivanje je bilo potpuno anonimno. Programeri su imali mogućnost postavljanja pitanja o značenju pojedinih pitanja iz upitnika, kako bi se umanjila vjerojatnost pogrešne interpretacije i osiguralo da dobijemo odgovore na sva pitanja. Osim odgovora na pitanja iz upitnika, prikupili smo i pozitivne i negativne primjedbe programera na upotrebljivost artefakata referentne arhitekture, podatke o broju razvijenih aplikacija i o alatima koje su koristili za razvoj.

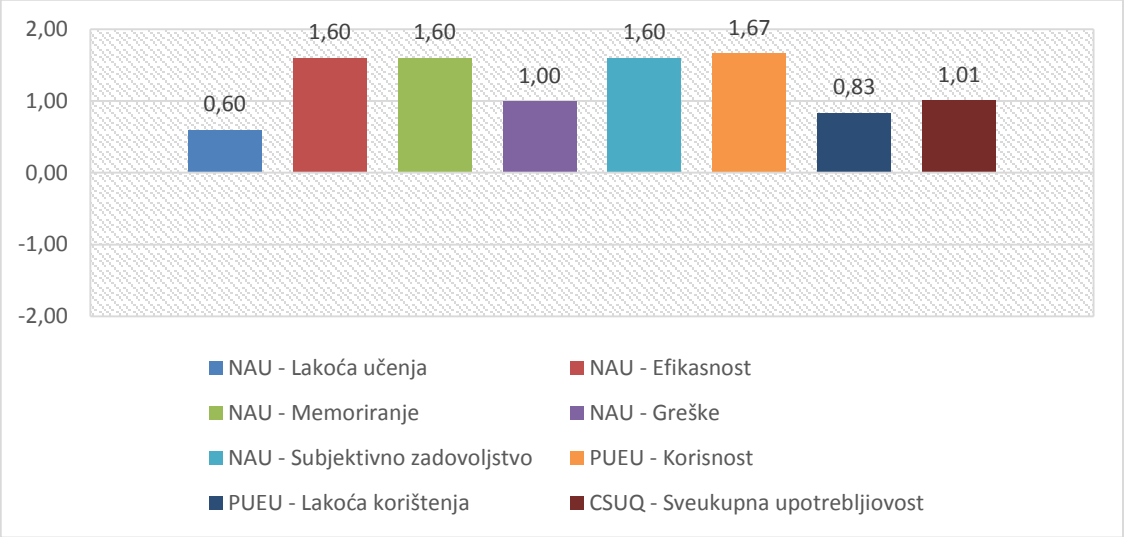
6.2.3. Rezultati ispitivanja

Rezultati dobiveni provedenim mjerenjem prikazani su u tablici 18. Općenito, prosječna vrijednost veća od nule ukazuje da promatrani aspekt referentne arhitekture (npr. lakoća učenja) ne sprječava korištenje referentne arhitekture u stvarnoj praksi. Prosječna vrijednost manja od nule ukazuje da konkretan aspekt upotrebljivosti referentne arhitekture potencijalno sprječava njeno korištenje u praksi. Niti jedan aspekt referentne arhitekture iz upitnika nije ocijenjen prosječnom vrijednošću manjom od nule. Sve u svemu, možemo tvrditi da je upotrebljivost referentne arhitekture dovoljno dobra da se može koristiti u praksi. Tablica 18 prikazuje pregled ocjena pojedinih aspekata referentne arhitekture koje smo dobili sintezom podataka iz upitnika.

Tablica 18 Rezultati ispitivanja upotrebljivosti referentne arhitekture

Metoda	**Istraživačko pitanje**	**Pitanja u upitniku**	**Prosječna ocjena**
NAU	S kojom lakoćom korisnik može naučiti koristiti referentnu arhitekturu?	Lakoća učenja	0,60
	Koliko efikasno korisnik može koristiti referentnu arhitekturu?	Efikasnost	1,60
	Je li lako zapamtiti način korištenja referentne arhitekture?	Memoriranje	1,60
	Da li referentna arhitektura uzrokuje samo mali broj i ne previše značajnih grešaka?	Greške	1,00
	Da li su korisnici subjektivno zadovoljni korištenjem referentne arhitekture?	Subjektivno zadovoljstvo	1,60
PUEU	Kakva je korisnost referentne arhitekture prema percepciji korisnika?	6 pitanja o percepciji korisnosti (tablica 23).	1,67
	Kakva je lakoća korištenja referentne arhitekture prema percepciji korisnika?	6 pitanja o percepciji lakoće korištenja (tablica 23).	0,83
CSUQ	Kakva je sveukupna upotrebljivost referentne arhitekture?	19 pitanja o sveukupnoj upotrebljivosti (tablica 23).	1,01

Slika 74 grafički prikazuje prosječne vrijednosti rezultata ispitivanja upotrebljivosti referentne arhitekture za svih 5 programera koji su ispunili upitnik.



Slika 74 Rezultati ispitivanja upotrebljivosti referentne arhitekture

S kojom lakoćom korisnik može naučiti koristiti referentnu arhitekturu?

Prosječna ocjena iznad nule potvrđuje da se referentna arhitektura može sa lakoćom naučiti koristiti. Ocjena lakoće učenja je relativno najniža u odnosu na druge karakteristike, vjerojatno zbog nedostatka sustavne edukacije prije početka rada na razvoju aplikacija. Artefakti referentne arhitekture, osim pomoćnog alata (engl. *plugin*) koji nije predmet ovog ispitivanja, nemaju vidljivo korisničko sučelje. Stoga je potrebno upoznati programsko sučelje (engl. *application programming interface*) glavnog programskog okvira referentne arhitekture kako bi ga se moglo koristiti u praksi.

Koliko efikasno korisnik može koristiti referentnu arhitekturu?

Efikasnost artefakata referentne arhitekture je ocjenjena vrlo dobrom. Vjerojatni razlog za povoljnu ocjenu su pozitivni rezultati primjene artefakata u praksi. Odzivno vrijeme i broj sistemskih grešaka poslovnih aplikacija koje koriste referentnu arhitekturu nije nikada bilo predmet razmatranja ili nužnog poboljšanja uslijed problema u praksi. Poboljšanje referentne arhitekture je uglavnom rezultat preventivnog ili održavanja u cilju poboljšanja karakteristika i proširivanja funkcionalnosti referentne arhitekture. Povećanje produktivnosti pri razvoju

poslovnih aplikacija korištenjem artefakata referentne arhitekture nije jednostavno izmjeriti, stoga je subjektivna ocjena razvojnih programera trenutno najprihvatljiviji oblik mjerenja.

Je li lako zapamtiti način korištenja referentne arhitekture?

Programeri su ocijenili da je lako zapamtiti način kako se koriste artefakti referentne arhitekture. Većina programera ima veliko iskustvo u korištenju Eclipse alata, što može dovesti u pitanje internu valjanost. Jedan od petero programera koji nisu koristili Eclipse u razvoju aplikacija ocijenili su skoro istim ocjenama pamćenje načina korištenja, te na osnovu toga možemo tvrditi da interna valjanost nije upitna s obzirom na to koji alat programeri koriste u razvoju aplikacija.

Da li referentna arhitektura uzrokuje samo mali broj i ne previše značajnih grešaka?

Analiza ocjena pokazuje da se greške rijetko događaju, ali se ipak događaju. Programeri su u razvoju poslovnih aplikacija koristili najveći broj funkcionalnosti referentne arhitekture, što uz činjenicu da su otkrivene greške redovito na vrijeme bile otklonjene, potvrđuje pouzdanost ovog testa.

Da li su korisnici subjektivno zadovoljni korištenjem referentne arhitekture?

Ako artefakte promatramo iz subjektivne perspektive njihove upotrebljivosti, možemo utvrditi da su programeri zadovoljni korištenjem artefakta (prosječna subjektivna ocjena je **1,6**).

Kakva je korisnost referentne arhitekture prema percepciji korisnika?

Percepcija korisnosti je ocjenjena najvećom ocjenom (**1,67**). Korisnost je vjerojatno teško izmjeriti s obzirom da programeri nemaju iskustvo u razvoju aplikacija korištenjem više raznih referentnih arhitektura kao što ga imaju arhitekti, što može biti i prijetnja valjanosti samog testa. Programeri koji su razvili više aplikacija ili modula korištenjem artefakata ocijenili su većom ocjenom korisnost referentne arhitekture.

Kakva je lakoća korištenja referentne arhitekture prema percepciji korisnika?

Ocjena lakoće korištenja je **0,83** (dobra). Korisnici koji su razvijali više aplikacija korištenjem artefakata ocijenili su sa većom ocjenom lakoću korištenja.

Kakva je sveukupna upotrebljivost referentne arhitekture?

Sveobuhvatna upotrebljivosti referentne arhitekture je ocjenjena sa **1,01** prema metodi CSUQ. Svi korisnici bez obzira na iskustvo i intenzitet korištenja slično su je ocijenili.

6.2.4. Zaključak provjere upotrebljivosti

Rezultati testiranja upotrebljivosti referentne arhitekture koje smo proveli potvrđuju da je referentna arhitektura „dovoljno upotrebljiva“. **Ovo ispitivanje ukazuje na dovoljnu upotrebljivost artefakta referentne arhitekture u razvoju poslovnih aplikacija** (hipoteza **H1** u poglavlju 1.5). Na temelju rezultata i sugestija ispitanika možemo tvrditi da upotrebljivost referentne arhitekture ne sprječava njenu upotrebu u praksi. Slijedi detaljni opis upotrebe referentne arhitekture u praksi razvoja poslovnih aplikacija u jednoj financijskoj instituciji.

6.3. Studija slučaja primjene referentne arhitekture

Tablica 19 Sažetak studije slučaja

Veličina:	3 - 4 razvojna programera.
Način uvođenja:	Strateška odluka organizacije, temelji se na iskorištavanju postojećih artefakata.
Rezultati:	- Produktivniji razvoj - 6 novih aplikacija u zadnjih 12 mjeseci. - Stabiliziranje postojećih aplikacija u produkciji. - Povećanje broja korisničkih transakcija - godišnji prosjek 334%. - Poboljšana ponovna upotreba raspoloživih komponenata. - Povećanje broja aktivnih korisnika od 101 na 169.
Poslovanje:	Dugoročna strategija uvođenja pristupa linija za proizvodnju softvera pri razvoju, održavanju i integraciji sa ostalim dobavljačima softvera; zbog potrebe povećanja produktivnosti, efikasnosti i fleksibilnosti kod konfiguriranja poslovnih aplikacija, efikasnije organizacije resursa za razvoj i održavanja, rada sa udaljene lokacije.
Arhitektura:	U instituciji je postojao razvijeni okvir referentne arhitekture koji se koristio nekoliko godina za razvoj poslovnih aplikacija. Tijekom ove studije postojeći okvir je prepravljen tako da zadovoljava postavljene kriterije za stabilnost i primjenu pristupa linija za proizvodnju softvera. U isto vrijeme, okvir je proširen na način da omogućava razvoj novih *web* aplikacija, migraciju postojećih aplikacija na *web* platformu. Nova inačica uključuje interno razvijene i *open-source* komponente.
Proces:	Proces razvoja se odvija prema agilnom procesnom modelu *Feature Driven Development* (FDD). Razvoj i održavanje artefakata referentne arhitekture je poseban proces (domensko inženjerstvo) koji jednom godišnje (početkom godine) započinje analizom stanja postojeće inačice, procjenom rizika, analizom okruženja, te na kraju pokretanjem projekta unaprjeđenja. Rezultati ovog procesa i projekata unaprjeđenja u prethodne tri godine su inačice V2, V3 i najnovija inačica V4 za razvoj *web* i mobilnih aplikacija. Razvoj poslovnih aplikacija rezultat je korisničkih zahtjeva na temelju kojih se pokreću projekti razvoja (aplikacijsko inženjerstvo). U ovom procesu aplikacije linije za proizvodnju se razvijaju korištenjem osnovnih artefakata i konfiguriranjem postojećih točaka varijabilnosti.
Organizacija:	Odjel za razvoj informacijskog sustava je organiziran prema jednom od modela za organizaciju pristupa linija za proizvodnju softvera „*razvojni odjel*“ (engl. *development department*). Razvoj poslovnih aplikacija i artefakata referentne arhitekture odvija se u jednoj organizacijskoj jedinici. Svaki član organizacijske jedinice može raditi na razvoju osnovnih artefakata i na razvoju aplikacija koje koriste osnovne artefakte. Razvoj se organizira po projektima koji se mogu svrstati u kategorije: infrastrukturni - za razvoj osnovnih artefakata infrastrukture ili njihovih novih inačica, te aplikativni projekti - za razvoj poslovnih komponenata i aplikacija, korištenjem osnovnih artefakata referentne arhitekture.

6.3.1. Uvod (engl. *problem statement*)

Predstavljamo longitudinalnu studiju jednog slučaja upotrebe naše referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera koju smo proveli u jednoj financijskoj instituciji s ciljem provjere primjenjivosti (engl. *utility*) njenih artefakata (C3.2, poglavlje 1.5) u praksi. Predmet proučavanja u ovoj studiji je referentna arhitektura za poslovne aplikacije općenito, njena primjena u razvoju većeg broja poslovnih aplikacija za podršku poslovnim procesima u financijskoj instituciji. Aplikacije zahtijevaju prilagodbu brzim i čestim poslovnim, zakonskim i tehnološkim promjenama. Intenzivna dinamika promjena u poslovnim aplikacijama zahtjeva prikladnu referentnu arhitekturu pomoću koje se te promjene mogu realizirati.

Referentnu arhitekturu (njene starije inačice) koja je predmet proučavanja u ovoj studiji prethodno smo koristili za razvoj poslovnih aplikacija u nekoliko različitih organizacija koje pripadaju raznim poslovnim sektorima kao što su: zdravstveno osiguranje, telekomunikacije, komunalne djelatnosti, bankarstvo, i prostorna geoinformatika. Na temelju iskustva u primjeni referentne arhitekture u različitim poslovnim sektorima, definirali smo funkcionalne zahtjeve i implementirali artefakte referentne arhitekture za razvoj poslovnih aplikacija općenito s ciljem obuhvata više poslovnih sektora.

Poboljšanje referentne arhitekture kroz nekoliko inačica (V1,V2,V3 i V4), njena prilagodba pristupu linija za proizvodnju softvera i ispitivanje primjenjivosti artefakata referentne arhitekture u praksi jedne financijske institucije, predmet su istraživanja u ovoj studiji slučaja. Detaljno ćemo opisati motivaciju i ciljeve naše suradnje sa financijskom institucijom, te dati pregled rezultata primjene referentne arhitekture u praksi.

6.3.2. Financijska institucija

Institucija. Financijska institucija sa sjedištem u Republici Hrvatskoj pruža usluge primanja depozita ili drugih povratnih sredstava od javnosti, odobravanje kredita i zajmova, uključujući potrošačke kredite i zajmove, otkup potraživanja s regresom, izdavanje garancija ili drugih jamstava te usluge vezane uz poslove kreditiranja, kao što su: prikupljanje podataka, izrada analiza i davanje informacija o kreditnoj sposobnosti pravnih i fizičkih osoba koje samostalno obavljaju djelatnost.

Financijska institucija ima preko 160 aktivnih korisnika poslovnih aplikacija koji svoje poslovne zadatke obavljaju na nekoliko umreženih fizičkih lokacija unutar institucije, te preko

nekoliko tisuća korisnika poslovnih aplikacija kao što su internet i mobilne aplikacije izvan institucije. Većina poslovnih aplikacija se izvodi na opremi institucije, dok se jedan dio opreme nalazi na lokacijama partnerskih tvrtki ili drugih institucija. Glavnina poslovnih aplikacija je oblikovana na način da velikom broju korisnika omogućava istovremeni pristup i njihovo korištenje. Podaci se uglavnom pohranjuju u relacijskim bazama podataka, istovremeno su dostupni raznim korisnicima putem poslovnih aplikacija, ovisno o ovlastima koje su im dodijeljene. Razvoj i održavanje glavnih poslovnih aplikacija (engl. *core*) obavlja se na lokaciji vanjskog dobavljača softvera, dok se većina ostalih, pomoćnih aplikacija, razvija i održava unutar financijske institucije.

Fokus studije slučaja. Ova studija slučaja se odnosi na razvoj, održavanje i korištenje nekoliko pomoćnih poslovnih aplikacija za podršku poslovnim procesima financijske institucije, a uključuje slijedeće aplikacije: obrada kreditnih zahtjeva, sustav za sprječavanje pranja novca, obrada zahtjeva za kreditne kartice, upravljanje kreditnim rizikom, upravljanje rizicima informacijskog sustava i operativnim rizicima, podrška procesu naplate potraživanja, poslovno izvješćivanje, *help desk*, podrška mandatnim poslovima, koje su implementirane kao Java *desktop* aplikacije; financijski kalkulatori, pisarnica, dionička knjiga, evidencija prisutnosti, Balanced Scorecard, e-obrasci, sjednice uprave, koje su implementirane kao *web* aplikacije. Poslovne aplikacije razvijaju se i održavaju pristupom linija za proizvodnju softvera. Ovaj pristup podrazumijeva odvajanje procesa razvoja referentne arhitekture i ponovno upotrebljivih komponenata (domensko inženjerstvo) od razvoja poslovnih aplikacija (aplikativno inženjerstvo). Prilagodba ovom pristupu zahtijevala je izmjene organizacijske strukture timova za razvoj i održavanje, prilagodbu procesa koji se koriste tijekom razvoja i održavanja, te promjenu arhitekture poslovnih aplikacija. Poslovni korisnici svoje zahtjeve za izmjenama aplikacija šalju timu za razvoj i održavanje putem internog *help desk* sustava.

6.3.3. Motivacija i ciljevi (engl. *research objectives*)

Glavni motivi financijske institucije za korištenje predložene referentne arhitekture i njenih artefakata, i uvođenje pristupa linija za proizvodnju softvera u kreditnu instituciju bili su: poboljšavanje prakse ponovne upotrebe (engl. *reuse*) postojećih komponenata u novim aplikacijama, povećanje produktivnosti u razvoju novih aplikacija, učinkovitija podjela aktivnosti razvoja i održavanja prema kvalifikacijama i specijalizaciji osoblja, smanjenje troškova razvoja i održavanja poslovnih aplikacija, smanjenje kašnjenja u odnosu na zahtjeve poslovnih korisnika za izmjenama, definiranje zajedničke arhitekture poslovnih aplikacija,

bolja dokumentiranost varijabilnosti sustava, te ublažavanje operativnog rizika u području razvoja i održavanja softvera uslijed nedovoljnog prenošenja znanja na zamjenske osobe unutar organizacije. Izmjene tehnoloških karakteristika poslovnih aplikacija zahtijevale su povećani napor i učestalo ponavljanje istih programerskih radnji na nekoliko različitih mjesta u više različitih aplikacija istovremeno. Ovisnost institucije o nestrukturiranom i nedokumentiranom znanju nekolicine pojedinaca, predstavljala je značajan operativni rizik za kreditnu instituciju.

Glavni cilj ove studije slučaja je bio pokazati da se predložena referentna arhitektura i njeni artefakti mogu uspješno primjenjivati u poslovnim aplikacijama za podršku poslovnim procesima u financijskoj instituciji. Predloženu referentnu arhitekturu financijska institucija koristi za razvoj poslovnih aplikacija pristupom linija za proizvodnju softvera. Glavna svrha primjene tog pristupa je jeftiniji, brži i kvalitetniji razvoj i održavanje poslovnih aplikacija kroz uvođenje promjena u arhitekturi poslovnih aplikacija, procesima i organizaciji.

Drugi ***cilj*** nam je prikupiti podatke metrika izvornog programskog koda artefakata referentne arhitekture i poslovnih aplikacija te pomoću njih provjeriti povezanost rezultata predloženog (PR) modela sa standardnim (MI) modelom za mjerenje održavljivosti softverskih komponenata.

6.3.4. Proces studije slučaja

Tablica 20 prikazuje proces koji smo slijedili u ovom istraživanju, odvija se u pet glavnih procesnih koraka. Proces je sličan kao i u drugim vrstama istraživanja, međutim, studija slučaja zbog svoje fleksibilnosti se najčešće odvija u ponavljajućim iteracijama, posebno kod prikupljanja podataka u longitudinalnoj vrsti studija slučaja kakva je ova.

Tablica 20 Proces studije slučaja

1.	*Dizajn studije slučaja* – definiraju se ciljevi i motivacija, definira se plan.
2.	*Priprema za prikupljanje podataka* – definiraju se procedure i protocol za prikupljanje podataka.
3.	*Prikupljanje podataka* – definiranje procedure za prikupljanje se provodi u praksi.
4.	*Obrada i analiza prikupljenih podataka* – podaci se obrađuju pomoću metoda za obradu podataka.
5.	*Prikazivanje rezultata istraživanja* – studija slučaja i rezultati istraživanja se na primjeren način prikazuju.

Prije samog početka ovog istraživanja analizirali smo raspoložive izvore podataka, repozitorij izvornog programskog koda, bazu podataka koja sadrži korisničke interakcije sa sustavom, "*help desk*" sustav korisničkih zahtjeva za izmjenama sustava. Analiza se odnosi na razdoblje od nekoliko prethodnih godina (2011 - 2014). Na temelju rezultata analize, motivacije i ciljeva, pristupili smo planiranju studije slučaja. U drugom koraku, u okviru priprema za prikupljanje podataka, definirali smo protocol za prikupljanje podataka. Protokol obuhvaća detaljnu proceduru za prikupljanje i analizu podataka, definiciju osnovnih pojmova, kriterija za prikupljanje podataka, izvore podataka, način i vrijeme njihovog prikupljanja. U prvoj iteraciji prikupili smo postojeće podatke za razdoblje od tri godine, obradili i analizirali prikupljene podatke te ih na primjeren način prikazali (Roško & Strahonja, 2014) i sačuvali za analizu u okviru ovog istraživanja. Nakon provjere atributa kvalitete postojeće inačice (V3) artefakata referentne arhitekture, pristupili smo prepravljanju (engl. *refactoring*) artefakata referentne arhitekture (poglavlje 5) kako bi postigli poboljšanje njihovih karakteristika. Nakon završenog prepravljanja, prema protokolu studije slučaja, ponovili smo prikupljanje podataka koji se odnose na najnoviju inačicu (V4), analizirali ih i prikazali u ovoj disertaciji.

6.3.5. Sustav poslovnih aplikacija koje istražujemo (engl. *context*)

Poslovna perspektiva: Glavni motiv razvoja poslovnih aplikacija koje istražujemo su povećanje produktivnosti, čuvanje podataka, automatizacija poslovnih procesa, povećanje kvalitete usluga i proizvoda koje institucija nudi svojim klijentima. Poslovne aplikacije koriste se u velikom broju poslovnih procesa, u svim organizacijskim jedinicama unutar financijske institucije. Aplikacije su integrirane sa centralnim „*core*" sustavom i drugim aplikacijama ostalih dobavljača unutar institucije. Korisnici poslovnih aplikacija su isključivo djelatnici institucije. Svakom korisniku poslovne aplikacije dodjeljuje se jedna ili više korisničkih uloga (engl. *roles*) pomoću kojih se definiraju ovlasti koje ti korisnici imaju u tijeku izvršavanja poslovnih procesa.

Arhitektura: Poslovne aplikacije koriste relacijske baze za pohranjivanje i dohvat podataka, internet preglednike kao klijentsku okolinu za izvođenje aplikacija. Poslovni podaci pohranjeni su u više različitih vrsta baza podataka, raznih dobavljača, što povećava kompleksnost upravljanja konfiguracijama i promjenama sustava poslovnih aplikacija. Aplikacije su razvijene u programskim jezicima *Java* i SQL, temelje se na vlastitoj referentnoj arhitekturi, koriste nekoliko standardnih vanjskih komponenata otvorenog izvornog programskog koda (engl. *open source*) kao što su Spring, Hibernate, Jasepersoft, iText, GWT.

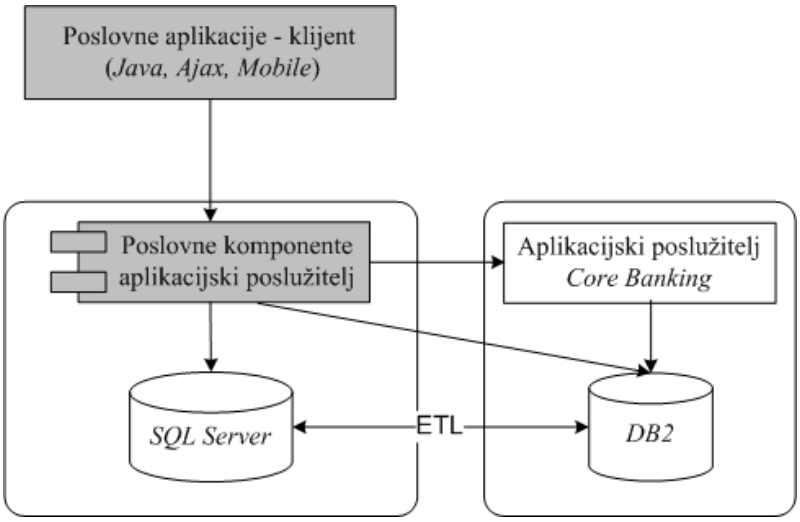
Tablica 21 prikazuje popis poslovnih aplikacija koje koriste referentnu arhitekturu koja je predmet ovog istraživanja.

Tablica 21 Poslovne aplikacije koje istražujemo (*desktop* i *web*)

Oznaka	**Naziv aplikacije**	**Broj korisnika**
Desktop A1	Poslovna izvješća	153
Desktop A2	*Help desk*	247
Desktop A3	Rizici informacijskog sustava	242
Desktop A4	Naplata	27
Desktop A5	Kreditni rizici	13
Desktop A6	Sprječavanje pranja novca	158
Desktop A7	Obrada kreditnih zahtjeva	59
Desktop A8	Zahtjevi za kreditne kartice	14
Desktop A9	Mandatni poslovi	5
Web A10	Financijski kalkulatori	150
Web A11	Pisarnica	6
Web A12	Dionička knjiga	3
Web A13	Evidencija prisutnosti	39
Web A14	*Balanced Scorecard*	12
Web A15	Obrasci	1
Web A16	Mobilne sjednice	5

Vlastito programiranje komponenata referentne arhitekture unutar financijske institucije važno je sa gledišta efikasnijeg povezivanja standardnih vanjskih komponenata, kontrole konteksta njihovog izvođenja, njihove eventualne zamjene, razvoja specifičnih komponenata za koje ne postoje standardna gotova rješenja otvorenog izvornog koda. Aplikacije su strukturirane u tri logička sloja: prezentacijski sloj (engl. *presentation layer*), sloj poslovne logike (engl. *business logic layer*) i sloj pristupa podacima (engl. *data access layer*). Prezentacijski sloj se u vrijeme korištenja izvodi na računalu korisnika aplikacije dok se ostali dio programske logike izvodi na aplikacijskom poslužitelju ili na poslužitelju baza podataka. Aplikacije se sastoje od komponenata koje se dinamički dodaju pojedinoj aplikaciji pri čemu se dinamički određuju i

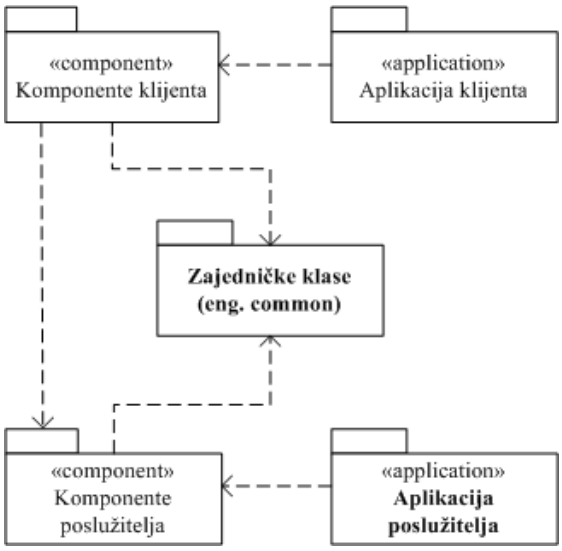
ovlasti korisnika poslovnih aplikacija. Slika 75 ilustrira kontekst korištenja referentne arhitekture za poslovne aplikacije (sivo obojano).



Slika 75 Kontekst izvođenja poslovnih aplikacija

Poslovne aplikacije koje istražujemo razvijale su se kroz duže vrijeme u financijskoj instituciji, prvotno se nisu razvile prema pristupu linija za proizvodnju softvera. Za provjeru našeg pristupa najprije smo pristupili rastavljanju postojećih monolitnih poslovnih aplikacija u manje programske cjeline (komponente). Podijelili smo komponente poslovnih aplikacija na komponente od kojih se sastoji serverski dio aplikacija i komponente od kojih se sastoje aplikacije na klijentu. Osim te dvije grupe komponenata, potrebno je bilo odvojiti klase koje se koriste u isto vrijeme i na serveru i na klijentu (engl. *common*) od komponenata koje se koriste samo na klijentu ili samo na serveru. Kao rezultat ovakve podjele bio je nužno uvesti novi model varijabilnosti (engl. *variability model*) aplikacija u odnosu na serverske i klijentske poslovne komponente od kojih se aplikacije sastoje - prema pristupu linija za proizvodnju softvera.

Slika 76 ilustrira predloženi model organizacije izvornog programskog koda za poslovne komponente i aplikacije koje koriste referentnu arhitekturu.



Slika 76 Organizacija programskih dijelova poslovnih aplikacija

Manji broj programskih rutina poslovnih aplikacija koje nisu poslovnog već tehničkog karaktera, te postojeći okvir (engl. *framework*) kojeg su aplikacije koristile, pretvorili smo u komponente referentne arhitekture koje se također dijele na grupe: server, klijent i zajednički (engl. *common*) dio. Kod pretvaranja monolitnih aplikacija u komponente poslužili smo se tkz. korisničkom perspektivom, tako da su nove samostalne komponente na klijentu vidljivi dijelovi aplikacije koji čine ponovno upotrebljivu cjelinu (npr. profil klijenta, zahtjev za kredit, pretraživanje klijenata) koju je moguće upotrijebiti u više aplikacija po potrebi. Svaka komponenta se može samostalno ugraditi u bilo koju aplikaciju korištenjem *plug-in* tehnike te prilagoditi ovlastima koje su definirane za pojedinog korisnika ili njegovu zadanu ulogu. Kriteriji za podjelu komponenata na serveru su skoro podjednaki kriterijima za podjelu komponenata na klijentu, koji osim poslovnog karaktera uključuju i organizacijsku prilagodbu te prilagodbu za ponovnu upotrebu.

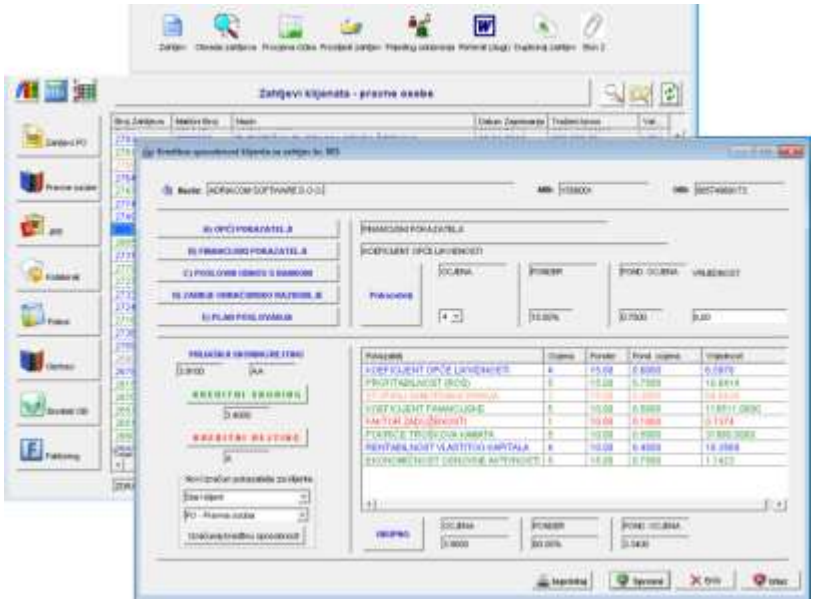
Tablica 22 prikazuje inačice sustava aplikacija i referentne arhitekture (liniju za proizvodnju softvera) koju istražujemo, broj poslovnih aplikacija i broj linija izvornog programskog koda (LOC) za svaku inačicu. Inačice V2, V3 i V4 su rezultat prilagodbe pristupu linija za proizvodnju softvera, pri čemu je inačica V3 rezultat pretvaranja postojećih aplikacija u dvije nove grupe komponenata: komponente referentne arhitekture (domensko inženjerstvo) i poslovne komponente (aplikativno inženjerstvo). Inačica V4 je rezultat poboljšanja kvalitete i uvođenja nekoliko značajnih novina kao što su: *Aspect Oriented* programiranje, *Java 7*, Google GWT, Google GWT Mobile. Sustav broji 16 aplikacija, ima preko 180.000 linija izvornog

programskog koda u Java programskom jeziku. Veliki dio poslovne logike je smješten u bazi podataka u obliku SQL programskih rutina (engl. *stored procedures, functions, views*) te u ostalim oblicima (engl. *javascript, html, css, xml, Jasper.jrxml, Pentaho.etl,* itd.). Programski jezik *Java* koristi se za razvoj poslovne logike na aplikacijskom serveru te za razvoj korisničkih sučelja u tri različita izdanja: *Mobile, Ajax, Java*.

Tablica 22 Inačice aplikacija i referentne arhitekture koje istražujemo

Predmeti (jedinice)	**Inačice**	**Okvir referentne arhitekture (LOC)**			**Aplikacije (LOC)**			**Broj aplikacija**
		Client	***Common***	***Server***	***Client***	***Common***	***Server***	
	V1	9843	13396	10140	45723	28223	26734	7
	V2	10250	13299	10307	52195	30673	29048	9
	V3	10323	13381	10426	57422	35599	33061	9
	V4	21551	15342	11845	73974	37160	28141	16

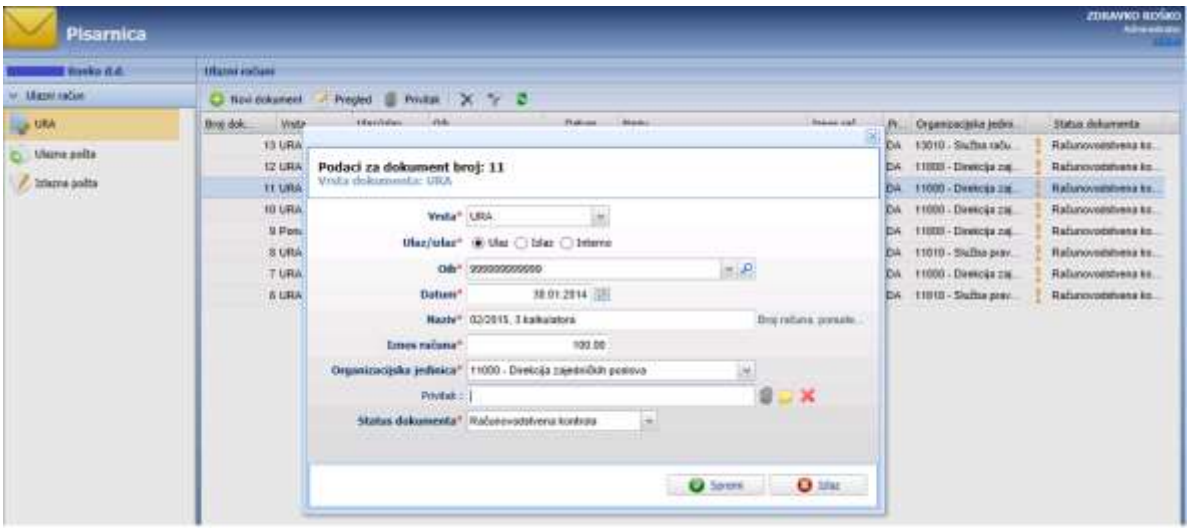
Poslovne aplikacije koje koriste V3 i starije inačice referentne arhitekture su klasične *desktop Java* aplikacije troslojne arhitekture, pristupaju aplikacijskom poslužitelju koji podatke smješta u relacijske baze podataka. Slika 77 ilustrira tipičan izgled korisničkog sučelja poslovnih aplikacija koje se baziraju na V3 inačici referentne arhitekture.



Slika 77 Korisničko sučelje aplikacije - Kreditni zahtjevi (Java SWT)

Glavne komponente korisničkog sučelja takvih aplikacija nazivamo perspektive koje se putem konfiguracije pojedinačno mogu ugraditi u bilo koju poslovnu aplikaciju. Inačica V3 broji ukupno 60 komponenata (perspektiva), aplikacije prosječno koriste 9,33 perspektive, dok se svaka perspektiva koristi u prosječno 1,40 poslovnih aplikacija.

Najnovija inačica V4 referentne arhitekture (prepravljena inačica V3) i poslovnih aplikacija predstavlja novi način razvoja aplikacija i prepravljanja postojećih aplikacija koji se temelji na *web* tehnologijama. Slika 78 ilustrira jednu takvu *web* aplikaciju, koja se također sastoji od perspektiva (lijevo – *URA, Ulazna pošta, Izlazna pošta*), centrale tablice i više formi za unos i izmjenu podataka.



Slika 78 Korisničko sučelje aplikacije - Pisarnica (Ajax GWT)

Razvoj mobilnih aplikacija je na samom početku u instituciji u kojoj provodimo istraživanje, trenutno postoji samo jedna aplikacija koja se koristi za praćenje sjednica Uprave financijske institucije. Referentna arhitektura prezentacijskog sloja još uvijek nije potvrđena u dovoljnom broju primjera u praksi, vanjske komponente koje koristimo nisu postale standard u softverskoj industriji. Očekujemo više korisničkih zahtjeva i veću potporu poslovnog dijela institucije kako bi ovo područje razvoja poslovnih aplikacija poboljšali do razine koju smo postigli u razvoju *web* aplikacija. Slika 79 prikazuje dio aplikacije za praćenje sjednica Uprave financijske institucije. Aplikacija se sastoji od perspektiva, kao i *web* aplikacije, dok su ostali elementi prezentacijskog sloja specifični za mobilne aplikacije, predstavljaju potpuno drugačiji način navigacije u odnosu prema ostale dvije vrste aplikacija.

Slika 79 Korisničko sučelje aplikacije - Sjednice uprave (GWT Mobile)

U pozadini, *web* (Ajax GWT), mobilne (GWT Mobile) i *desktop* (Java SWT) aplikacije koriste iste poslužiteljske poslovne komponente i okvir referentne arhitekture koji se primjenjuje na poslužitelju. Razlika između navedene tri vrste aplikacija odnosi se samo na dio okvira referentne arhitekture koji se primjenjuje za razvoj prezentacijskog sloja i na poslovne komponente koje ga primjenjuju.

6.3.6. Postojeća istraživanja (engl. *theory*)

Referentna arhitektura koju istražujemo je jedinstveni primjerak, stoga smo umjesto analize prethodnih istraživanja, analizirali slična istraživanja i teorijski okvir za provedbu ove studije slučaja. Proces kojeg koristimo u ovoj studiji slučaja temelji se na preporukama provedbe postupka istraživanja (Runeson i ostali, 2012; Wohlin i ostali, 2003, str. 55) i teorijskom okviru iz područja kvalitete softverskih proizvoda, ograničenjima, definicijama, modelima i kritičkoj analizi koju temeljimo na radovima (Folmer & Bosch, 2004, 2005; John & Bass, 2001).

Studije slučaja istraživanja primjenjivosti i upotrebljivosti programskih alata za konfiguraciju aplikacija linija za proizvodnju u nekoliko organizacija (Dhungana i ostali, 2006; Rabiser, 2009) su slična istraživanja ovom istraživanju a koja ujedno potvrđuju primjerenost postupka kojeg koristimo u ovoj studiji slučaja.

6.3.7. Istraživačka pitanja (engl. *research questions*)

U ovoj studiji slučaja opisujemo postupak i rezultate provjere valjanosti artefakata referentne arhitekture kako bi dobili odgovor na glavno istraživačko pitanje (**P3**) „*Da li je referentna arhitektura koja zadovoljava potrebne funkcionalne zahtjeve korisna u praksi?*“. Studija slučaja odnosi se na postavljeni istraživački cilj (C3.2) „*Primijeniti i koristiti razvijene artefakte referentne arhitekture u praksi putem longitudinalne studije slučaja s ciljem provjere njihove primjenjivosti* (engl. *utility*)“, kako bi se provjerila hipoteza **H1**: „Nova referentna arhitektura prema pristupu linija za proizvodnju softvera je korisna za proizvodnju softvera za poslovne primjene“.

Osim glavnog istraživačkog pitanja **(P3)**, u ovoj studiji slučaja analiziramo upotrebu artefakata referentne arhitekture kako bi dobili odgovor na istraživačko pitanje (**P4**) „*Postoji li povezanost između predloženog modela PR za mjerenje održavljivosti linije za proizvodnju softvera za poslovne aplikacije i standardnog modela MI, kada se oba modela koriste za mjerenje održavljivosti identičnih softverskih komponenata?*“. Prikupljene podatke koristimo kako bi ostvarili istraživački cilj (C4) „*Provjeriti povezanost rezultata PR i MI modela za mjerenje održavljivosti aplikacijskih komponenata linije za proizvodnju softvera*“ za provjeru hipoteze **H2**: „Rezultati određivanja održavljivosti poslovnih komponenata (engl. *maintainability*) informacijskog sustava metrikom „odgovornosti platforme“ (engl. *Platform Responsibility*) podudaraju se s rezultatima dobivenim uz pomoć alternativne metrike“.

Glavni cilj istraživanja ove studije slučaja je praktično ispitivanje utjecaja predložene referentne arhitekture na promjenjivost (engl. *changeability*) poslovnih aplikacija koje ju koriste, prema pristupu linija za proizvodnju softvera. Znanje prikupljeno praktičnim ispitivanjem korištenja referentne arhitekture u praksi razvoja poslovnih aplikacija dio je ponavljajućeg ciklusa poboljšanja predložene referentne arhitekture. Drugi cilj ove studije slučaja je omogućavanje korištenja predložene referentne arhitekture kao sredstva za razvoj poslovnih aplikacija poboljšane održavljivosti (promjenjivosti) u financijskoj instituciji.

6.3.8. Izbor predmeta istraživanja (engl. *case and subject selection*)

Izbor kreditne institucije za ovu studiju slučaja rezultat je postojećeg odnosa i obostrane zainteresiranosti istraživača i institucije za poboljšanje procesa razvoja i održavanja poslovnih aplikacija. Kreditna institucija postepeno uvodi pristup linija za proizvodnju softvera pri čemu je bitno već u ranom periodu, negdje u vrijeme razvoja treće ili četvrte aplikacije, ispitati

kvalitetu referentne arhitekture i njen utjecaj na promjenjivost aplikacija. Ispitivanje promjenjivosti aplikacija zahtjeva praćenje nekoliko inačica (longitudinalno) aplikacija zajedno sa referentnom arhitekturom koju aplikacije koriste. Predmet analize (engl. *unit of analysis*) u ovoj longitudinalnoj studiji su četiri inačice poslovnih aplikacija i predložena referentna arhitektura koju aplikacije koriste (engl. *embedded case study*). Pod jedinice (engl. *subunits*) ove studije su programske komponente (oko 80 klijentskih i oko 60 serverskih) koje se koriste za razvoj poslovnih aplikacija.

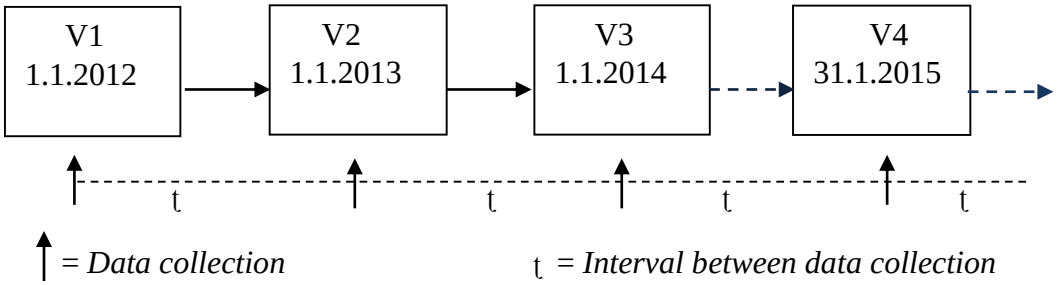
6.3.9. Prikupljanje podataka (engl. *methods of data collection*)

U ovoj studiji slučaja prikupljamo slijedeće podatke za analizu: (1) četiri inačice izvornog programskog koda referentne arhitekture i aplikacija koje ju koriste, (2) broj i vrste izmjena (*add, delete, update*) izvornog programskog koda između dviju uzastopnih inačica, (3) interakcije poslovnih korisnika sa poslovnim aplikacijama (transakcije), (4) korisničke zahtjeve za izmjenama poslovnih aplikacija. Prikupljanje podataka odvija se longitudinalno u razdoblju od 1.1.2012. do 31.1.2015. U tu svrhu formirali smo direktorij (bazu podataka studije slučaja) koji sadrži sve dokaze naših konačnih zaključaka kojima potkrepljujemo pouzdanost ovog istraživanja. U bazi podataka studije slučaja čuvamo izvorni programski kod u obliku Eclipse *Workspace*, kvantitativne podatke u obliku Excel tablica, obrađene podatke u obliku IBM SPSS datoteka. Tablica 23 prikazuje raspoložive izvore podataka koje smo koristili prilikom prikupljanja podataka u analizi studije slučaja.

Tablica 23 Izvori podataka koji su se koristili u studiji slučaja

Izvor podataka	***Opis izvora***	***Razdoblje***
Baza zahtjeva za izmjenama	*Help Desk* financijske institucije	2012 - 2015
Repozitorij izvornog koda	CollabNet Subversion Edge 1.1.0	2012 - 2015
Baza transakcijskih zapisa	*Log* baza aktivnosti korisnika	2013 - 2015
Zapis konfiguracija	Procedura za upravljanje konfiguracijama	2014
Razvojna dokumentacija	*Use Case* dokumenti, opis arhitekture	2012 -2015
Modeli baza podataka	*Embarcadero ER Studio* datoteke	2014
Promatranje (opažanje)		2012 - 2015

Glavni izvor podataka ove longitudinalne studije slučaja je repozitorij izvornog programskog koda (CollabNet Subversion Edge 1.1.0). Longitudinalno oblikovanje ove studije odabrali smo kako bi utvrdili eventualno poboljšanje u metrikama referentne arhitekture i njihovu povezanost sa ostalim metrikama izvornog programskog koda, posebno metrikama komponenata poslovnih aplikacija koje koriste referentnu arhitekturu. Promatrano razdoblje podjelili smo na četiri inačice referentne arhitekture i pripadajućih joj aplikacija. Prve tri inačice se odnose, svaka pojedinačno, na razdoblje od godine dana između inačica, i to u vrijeme dok se postepeno uvodio pristup linija za proizvodnju softvera u instituciju; zadnja inačica odnosi na razdoblje od 12 mjeseci u kojem se referenta arhitektura značajno izmijenila u odnosu na prethodno stanje. Slika 80 prikazuje vremenske intervale u kojima smo ponavljali prikupljanje podataka.



Slika 80 Oblikovanje longitudinalnog prikupljanja podatka (Kumar, 2005, str. 98)

Podaci metrika izvornog koda prikupljeni su korištenjem alata za prikupljanje metrika izvornog koda: MyEclipse Enterprise Workbench Version: 11.0.1, © 2013, Genuitec, L.L.C., CodePro Analytix™, Version 7.1.0.r36, © 2011, Google, Inc., za objektno-orijentirane i metrike međuovisnosti programskih elemenata (klasa i komponenata), te JHawk Release 5 Version 1.01, © 2010, Virtual Machinery. Podaci metrika za sve inačice izvornog koda referentne arhitekture i proizvoda linije prikupljeni su istim alatima, a u cilju smanjivanja vjerojatnosti krive interpretacije pojedinih metrika. Izvorni programski kod za istraživanje odnosi se na referentnu arhitekturu i na 16 poslovnih aplikacija koje tu arhitekturu koriste. Kriteriji za izbor izvornog koda za ovo istraživanje bili su:

a) korištenje *Java* programskog jezika (poslovne aplikacije, artefakti referentne arhitekture)

b) zajednička referentna arhitektura (poslovne aplikacije)

c) svakodnevno korištenje aplikacija u radu financijske institucije.

Tablica 24 prikazuje metrike koje smo prikupljali korištenjem alata za prikupljanje metrika, za svaku inačicu posebno. Jedinice mjerenja su komponente, klijentske ili poslužiteljske, sastoje se od više klasa, tako da većina metrika u biti predstavljaju prosjeke između klasa koje pripadaju promatranoj komponenti.

Tablica 24 Metrike izvornog programskog koda koje prikupljamo

Metrika	***Opis***
PR	*Platform responsibility*
MIC	*Maintainability index - sa komentarima*
MI	*Maintainability index - bez komentara*
ABD	*Average block depth*
ACC	*Average cyclomatic complexity*
ADIT	*Average depth of inheritance hierarchy*
AMC	*Average number of methods per class*
ALCM	*Average lines of code per method*
AWMC	*Average weighted methods per class*
NMETH	*Number of methods*
WM	*Weighted methods*
NCLASS	*Number of classes*
LOC	Broj linija programskog koda
D3	Broj poveznica poslovnih komponenata prema okviru referentne arhitekture
D4	Broj poveznica poslovnih komponenata prema Java platformi
D5	Broj poveznica poslovnih komponenata prema vanjskim komponentama

Izmjene u poslovnim aplikacijama i artefaktima referentne arhitekture prikupili smo u obliku broja i vrste izmjena (*add, delete, update*) izvornog programskog koda između dviju uzastopnih inačica (V1-V2, V2-V3), tablica 25. Za zadnju inačicu nismo prikupljali podatke o izmjenama jer su one značajne u odnosu na prethodnu inačicu. Prepravljanje (engl. *refactoring*) inačice V3 u značajno drugačiju inačicu V4 temelji se na istraživanju metrika prethodnih inačica, te poslovnoj odluci financijske institucije da se *Java* desktop aplikacije zamjene sa novim *web* aplikacijama, te da se u isto vrijeme uvede pristup linija za proizvodnju softvera u odjel za razvoj. Osnova za istraživanje izvornog programskog koda za *Java desktop* klijentske aplikacije su komponente koje čine jednu korisničku cjelinu, sastoje se od tri vrste klasa: *model, view, controller* prema *Model View Controller* (MVC) uzorku za dizajn korisničkih sučelja. Primjeri komponenata na klijentu su: *Profil pravne osobe, Kreditni zahtjev obrtnika, Naplata tekućih računa, Imovina informacijskog rizika, Depoziti pravnih osoba*. Klijentske komponente se mogu koristiti u više poslovnih aplikacija, bez izmjene, uz potrebnu konfiguraciju. *Web*

klijentske aplikacije koje koriste V4 inačicu referentne arhitekture, rezultat su najnovijih projekata razvoja, nisu bile predmet detaljnog istraživanja u ovoj studiji slučaja.

Tablica 25 Metrike izmjena u aplikacijama i artefaktima referentne arhitekture

Metrika	***Opis***
C-ADD	Broj novih klasa
C-UPDATE	Broj ažuriranih klasa
C-DELETE	Broj izbrisanih klasa
M-ADD	Broj novih metoda
M-UPDATE	Broj ažuriranih metoda
M-DELETE	Broj izbrisanih metoda
L-ADD	Broj novih linija
L-UPDATE	Broj ažuriranih linija
L-DELETE	Broj izbrisanih linija

Osnova za istraživanje na poslužitelju su komponente koje čine jednu poslovnu cjelinu a sastoje se od klasa vrste: *facade, business object, data access object*, prema J2EE uzorcima za oblikovanje aplikacijskih komponenata. Primjer poslužiteljskih komponenata su: *Profil pravne osobe, Kreditni zahtjev obrtnika, Naplata tekućih računa, Imovina informacijskog rizika, Depoziti pravnih osoba*, itd., slično kao i kod klijentskih komponenata. Komponente se mogu koristiti u više poslovnih aplikacija, prema pristupu linija za proizvodnju softvera.

6.3.10. Validacija podataka (engl. *quality control and assurance*)

Glavni kriteriji kod izbora metrika programskog koda koje smo koristili u ovoj studiji su njihova učestala upotreba u sličnim istraživanjima (Abreu & Carapuça, 1994; Belsley, Kuh, & Welsch, 2005; Zhou & Xu, 2008), te utvrđena provjera njihove valjanosti. Kako bismo osigurali objektivnost, valjanost i pouzdanost prikupljenih rezultata metrika i drugih prikupljenih podataka, koristili smo nekoliko tehnika za njihovu provjeru:

Kriterijska valjanost metrika pomoću koje smo provjeravali u kojoj su mjeri rezultati dobiveni jednim instrumentom povezani s rezultatima dobivenim s drugim instrumentom koji nam je služio kao vanjski kriterij valjanosti.

- Prikupljanje podataka metrika programskog koda napravili smo na dva usporediva načina: pojedinačno za svaku komponentu, te skupno za sve komponente (za svaku

inačicu) odjednom, kako bi mogli *usporedbom* rezultata metrika iz dva različita mjerenja utvrditi njihovu povezanost te zaključiti da je kriterijska valjanost dobra.

- Korištenjem dva različita alata (JHawk i CodePro Analytix™) ya prikupljanje većine podataka metrika izvornog programskog koda kako bi usporedbom utvrdili da je kriterijska valjanost dobra.

Konstruktna valjanost metrika pomoću koje smo uspoređivali rezultate mjerenja metrika sa dva ili više instrumenata kako bi utvrdili stupanj povezanosti među metrikama za koje se pretpostavlja da imaju isti predmet mjerenja (*konvertentna valjanost*).

- Rezultate mjerenja nekoliko metrika uspoređivali smo sa metrikama koje pripadaju istoj skupini, kako bi utvrdili njihov stupanj povezanosti, te smo tako utvrdili slijedeće:
    - WM = NMETH*CC
    - Postoji pozitivna korelacija između metrika NCLASS,LOC,WM.

Pouzdanost mjerenja metrika postupkom *vanjske konzistencije* kako bi dobili konzistentne rezultate u ponovljenom mjerenju pod istim uvjetima.

- Ponavljanjem mjerenja svake, slučajno odabrane, pete komponente, od ukupno preko stotinu komponenata, po potrebi u nekoliko ciklusa, sve dok nismo dobili istovjetne podatke za svaku komponentu u odnosu na prethodno mjerenje. Rezultate mjerenja i rezultate prethodnog mjerenja stavljali smo u odnos te ponavljali mjerenje u novom ciklusu sve dok taj odnos nije bio jednak **1** što ukazuje na 100% pouzdano mjerenje.

6.3.11. Demonstriranje korisnosti razvijenih funkcionalnosti

U ovom poglavlju opisati ćemo korisnost funkcionalnosti okvira referentne arhitekture i njihovu relevantnost kod primjene u financijskoj instituciji koju smo prezentirali kao studiju slučaja. Glavnina funkcionalnosti se intenzivno koristi u aplikacijama financijske institucije, detaljnije ćemo opisati funkcionalnosti koje se koriste za razvoj serverskih komponenata poslovnih aplikacija.

Ograničenja studije slučaja. U ovoj studiji slučaja primjenjuje se većina funkcionalnosti referentne arhitekture, međutim, prema načinu oblikovanja i arhitekturi funkcionalnosti implicitno je obuhvaćena i većina ostalih funkcionalnosti koje nisu direktno korištene u ovoj studiji slučaja. Primjerice, način pristupa podacima nekih specifičnih izvora podataka (npr.

CICS, JMS) koji se ne koristi u financijskoj instituciji, možemo potvrditi kao primjenjivu funkcionalnost na temelju iskustva primjene na nekim prethodnim projektima za koje smo koristili predloženu arhitekturu i oblikovanje.

Problem generalizacije. Alternativna primjena referentne arhitekture za poslovne aplikacije općenito, uključujući i financijski poslovni sektor, kroz nekoliko studija slučaja u više poslovnih sektora, bila bi vjerodostojnija u pogledu generalizacije. Međutim, predloženi način podjele slojeva poslovnih aplikacija prema hijerarhijskom principu koji na jedan način razdvaja komponente okvira referentne arhitekture od poslovnih komponenata pojedinih poslovnih sektora, implicitno omogućava da kroz jednu studiju slučaja možemo sa pouzdanjem tvrditi da je vjerodostojnost potvrde na jednoj studiji slučaja dovoljna. Isto tako, kao potvrdu primjenjivosti referentne arhitekture možemo uzeti u obzir i primjenu prethodnih inačica (V1-V3) artefakata referentne arhitekture u drugim poslovnim sektorima. Iako artefakti prethodnih inačica nisu bili prilagođeni pristupu linija za proizvodnju softvera, kao što je slučaj sa inačicom (V4) koju koristimo u studiji slučaja, artefakti su oblikovani prema istoj arhitekturi.

Tablica 26 prikazuje relevantnost funkcionalnosti okvira referentne arhitekture za server koje se koriste u studiji slučaja.

Prijenos i čuvanje podataka u jedinstvenom obliku - *Data Value Objects*: poslovni podaci koji se prenose i čuvaju u svim slojevima aplikacija uglavnom se čuvaju u objektima ovoga tipa. Sučelja poslovnih komponenata ih koriste kao ulazne i izlazne parametre, klase za pristup podacima ih koriste za prihvat podataka, korisnička sučelja kao *model* u kontekstu *Model View Controller* predloška, dok ih ostali servisi koriste kao standardni format za razmjenu i transformaciju podataka u i iz više različitih oblika (npr. JSON, XML).

Smještanje glavnine poslovne logike u klase istoga tipa - *Business Objects*: aplikacije u pravilu rijetko koriste ovu funkcionalnost, najviše stoga što se većina poslovne logike smješta u baze podataka u obliku procedura (engl. *stored procedures*), dok se ostatak poslovne logike nalazi u glavnim klasama koje predstavljaju poslovne komponente, njihovim sučeljima (engl. *facade*).

Smještanje logike pristupa podacima u klase istoga tipa - *Data Access Objects*: za pristup podacima najčešće se koriste funkcije za pristup relacijskim bazama podataka, značajno manje se koristi pristup sustavu *iSeries*, dok se ostale funkcionalnosti ne koriste u primjeni u financijskoj instituciji.

Jedinstveno upravljanje zahtjevima sa klijenta - *Server Request Handlers*: aplikacije na klijentu koriste tri različite tehnologije (Java SWT, GWT, GWT Mobile), njihova komunikacija sa serverom odvija se korištenjem nekoliko protokola: RMI, HTTP, REST (JSON).

Tablica 26 Funkcionalnosti referentne arhitekture za **server**

Broj	Funkcionalnosti okvira referentne arhitekture	Relevantnost
S1	Prijenos i čuvanje podataka u jedinstvenom obliku	++
S2	Smještanje glavnine poslovne logike u klase istoga tipa	+
S3	Smještanje logike pristupa podacima u klase istoga tipa	++
S4	Jedinstveno upravljanje zahtjevima sa klijenta	++
S5	Jedinstveno upravljanje poslovnim transakcijama	++
S6	Uniformna reprezentacija svih izvora podataka	+
S7	Uniformno upravljanje sa spajanjem na izvore podataka	+
S8	Priprema i prikazivanje izvještaja u više različitih formata	++
S9	Konfiguracija sigurnosnih postavki	+
S10	Upravljanje transakcijama	+
S11	Upravljanje sesijama	+
S12	Pregledavanje podataka na strani klijenta u više koraka	+

Jedinstveno upravljanje poslovnim transakcijama - *Remote Object Invokers*: prijenos podataka sa klijenta prema poslovnim komponentama na serveru, kako bi se uspostavila poslovna transakcija, odvija na jedinstven način, korištenjem samo jedne od raspoloživih vrsta ove funkcije.

Uniformna reprezentacija svih izvora podataka - *Data Sources*: poslovni podaci u financijskoj instituciji čuvaju se u relacijskim bazama podataka ili na sustavu *iSeries*, za pristup podacima koristimo samo dvije raspoložive mogućnosti (JDBC, *iSeries*). Ovu funkciju koristimo i za privremeno čuvanje podataka u memoriji poslužitelja (engl. *cache*).

Uniformno upravljanje sa spajanjem na izvore podataka - *Connection Pool*: upravljanje komunikacijom prema izvorima podatka podržano je na nekoliko načina, korištenjem standardnih i vlastitih rješenja koja za razliku od standardnih omogućavaju povezivanju više različitih izvora podataka u jednu jedinstvenu transakciju. Koristimo samo dvije raspoložive mogućnosti (JDBC, *iSeries*).

Priprema i prikazivanje izvještaja u više različitih formata - *Reporting Service*: u primjeni se koriste sve raspoložive funkcionalnosti (*Jasper, Excel, Word*, PDF), međutim najčešće se koristi *Jasper* format za koji postoje i koriste se ostali raspoloživi alati za pripremu izvješća (*iReport*) te za njihovo čuvanje i posluživanje (*JasperServer*).

Konfiguracija sigurnosnih postavki - *Security Authorization and Authentication*: za autorizaciju se koristi centralni sustav za prijavu (AD), dok se za autorizaciju ovlasti koristi vlastiti sustav za upravljanje ovlastima u poslovnim aplikacijama (RBAC).

Upravljanje transakcijama – *Transaction*: poslovne zadatke (engl. *logical unit of work*) koji se istovremeno odnose na različite vrste izvora podataka (npr. relacijske baze, CICS) potrebno je bilo obuhvatiti u jedinstvenu poslovnu transakciju. U primjeni koristimo samo relacijske baze podataka, stoga je bilo moguće izabrati između dvije opcije: JTA i vlastitog rješenja. Koristimo vlastito rješenje zbog više raspoloživih rezultata testiranja i iskustva primjene sličnih rješenja u drugim poslovnim organizacijama.

Upravljanje sesijama – *Session*: za privremeno čuvanje osnovnih korisničkih podataka koristimo spremište u memoriji klijentskog i/ili serverskog dijela aplikacije, ukoliko se radi o Java SWT vrsti klijenta, za ostale vrste klijentskih aplikacija koristimo *Cookie*.

Pregledavanje podataka na strani klijenta u više koraka - *Value List Handler*: koristi se samo za prikaz nekih nizova poslovnih podataka na klijentu za koje postoji veliki broj podataka u relacijskim bazama podataka, a koji se zbog ograničenja klijentskih uređaja ne mogu prikazati odjednom.

Na temelju rezultata primjene referentne arhitekture u praksi financijske institucije, možemo tvrditi da je **„nova referentna arhitektura prema pristupu linija za proizvodnju softvera korisna za proizvodnju softvera za poslovne primjene“** (hipoteza **H1** u poglavlju 1.5).

6.4. Ocjenjivanje evolucije okvira referentne arhitekture

Glavni artefakt referentne arhitekture za poslovne aplikacije koju predlažemo je njezin programski okvir (engl. *framework*). Ukupni napor i trajanje njegova razvoja te veliki utjecaj na održavljivost linije za proizvodnju softvera u usporedbi sa ostalim artefaktima, čine ga ključnim artefaktom. Pravilno je da se razvoj programskog okvira referentne arhitekture odvija polako ali konstantno. Promjene koje se događaju tijekom razvoja nužno dovode do jedne „kritične točke“ kod koje se značajno smanjuje njegova fleksibilnost i povećava njegova kompleksnost. To naglo poskupljuje i otežava njegov daljnji razvoj. Kako bi se izbjegla navedena situacija, važno je identificirati komponente koje je potrebno preoblikovati (Gall, Jazayeri, Klosch, & Trausmuth, 1997). Budući da promjene programskog okvira imaju veliki utjecaj na poslovne aplikacije koje ga primjenjuju, smatra se da je važno pravovremeno provjeravati stabilnost okvira i održavljivost komponenata koje ga primjenjuju. Stabilnost okvira predložene referentne arhitekture u ovom istraživanju provjeravamo pomoću pristupa za provjeru stabilnosti okvira (Mattson & Bosch, 1999) koji je u biti proširen *Evaluating application framework architecture structural and functional stability* (J. Bansiya, 1998) pristup, dok za održavljivost koja se standardno provjerava pomoću indeksa održavljivosti (engl. *Maintainability Index*) predlažemo novi model provjere održavljivosti (poglavlje 6.5).

Predloženi okvir referentne arhitekture je zahtjevan i kompleksan sustav koji sadrži **strukturu** i **funkcionalnosti** za razvoj poslovnih aplikacija ponovnom upotrebom njegovog oblikovanja (engl. *design*) i programskog koda, za razliku od knjižnica klasa (engl. *class libraries*) koje sadrže samo programski kod za ponovnu upotrebu. Okvir referentne arhitekture omogućava razvoj poslovnih komponenata kroz primjenu mehanizma (engl. *inversion-of-control*) pri čemu razvojni programer za aplikacije isključivo razvija samo programsku poslovnu logiku za specifičnu poslovnu komponentu. Kako bi se povećala produktivnost kod razvoja poslovnih komponenata, okvir referentne arhitekture treba biti lako razumljiv, stabilan i što manje kompleksan. Razvoj okvira i drugih artefakata referentne arhitekture zahtjeva: iskustvo u softverskom inženjerstvu, poznavanje poslovnog sektora za koji se aplikacije razvijaju, efektivno predviđanje budućih načina na koje će poslovne aplikacije koristiti artefakte referentne arhitekture. Uobičajeno je da se kvalitetan okvir stabilizira nakon postojanja nekoliko inačica (generacija) koje redovito sadrže popravke grešaka iz prethodnih inačica i ugrađene nove ili izmijenjene postojeće funkcionalnosti. Prve tri inačice okvira predložene referentne arhitekture su sadržavale osnovne funkcionalnosti, dok je zadnja inačica (V4) rezultat ugradnje novih funkcionalnosti na temelju provedenog istraživanja i provjere atributa

kvalitete prethodnih inačica, kako bi okvir postao kompletan, fleksibilan, jednostavniji za korištenje, te „preuzeo“ na sebe više odgovornosti (funkcionalnosti).

Karakteristike arhitekture okvira koje se mogu poboljšavati između inačica identificiraju se i provjeravaju korištenjem metrika izvornog programskog koda. Za provjeru intenziteta izmjene okvira između inačica koristimo (engl. *extent-of-change***)** metriku. Ista metrika nam služi kao indikator (indeks) stabilnosti oblikovanja okvira (strukture i funkcionalnosti). Za procjenu evolucije okvira referentne arhitekture koristili smo dvije metode:

- Identifikacija evolucije korištenjem povijesnih podataka (engl. *Evolution Identification Using Historical Information*)
- Procjena stabilnosti (engl. *Stability Assesment*)

Temeljem rezultata ovih metoda identificirali smo komponente koje se često mijenjaju, analizirali utjecaj promjena na druge komponente, odredili područja izmjena i dogradnje okvira referentne arhitekture (V3) kako bi dobili njegovu poboljšanju inačicu (V4).

6.4.1. Identifikacija evolucije korištenjem povijesnih podataka[14]

Identifikacija evolucije korištenjem povijesnih podataka (engl. *Evolution Identification Using Historical Information*) (Mattson & Bosch, 1999) je metoda koja se koristi za pronalaženje i karakterizaciju svojstava razvojnih okvira (njegovih podsustava i komponenata) koja su podložna promjenama, promatrajući povijesne podatke o promjenama koje su nastale tijekom razvoja i održavanja. Na temelju povijesnih podataka različitih inačica razvojnih okvira, moguće je donijeti zaključke o evoluciji okvira u pogledu veličine, stope rasta i stope promjene (Mattsson & Bosch, 1999). Za razliku od izvorne metode (Gall i ostali, 1997) na kojoj se temelji ova metoda, a koja koristi pojmove *sustav, podsustav, modul* i *program* kao elemente promatranja, ova metoda je prikladna za razvojne okvire koji imaju od 200 do 600 klasa, kakav je i razvojni okvir kojeg promatramo; metoda koristi pojmove *sustav, podsustav, komponenta* i *klasa* kao elemente promatranja koji su prikladniji za objektno orijentirane sustave.

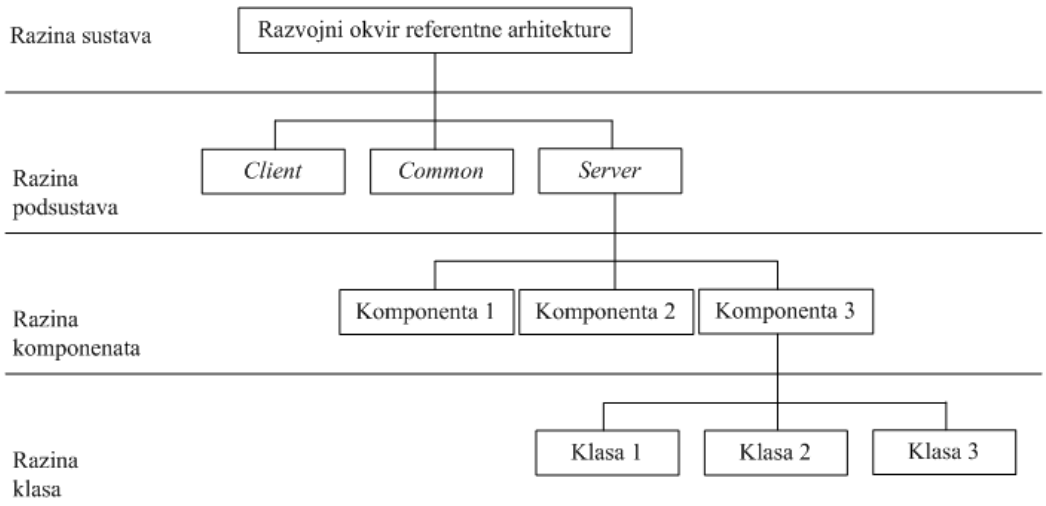
Prema metodi *Identifikacija evolucije korištenjem povijesnih podataka* (Mattson & Bosch, 1999), postupak pronalaženja i karakterizacije svojstava razvojnih okvira, njegovih podsustava i komponenata podložnih promjenama sastoji se od slijedećih koraka:

1. Računanje stope promjene i stope rasta razvojnog okvira, za svaku inačicu.

[14] Evolution Identification Using Historical Information (Mattsson, 2000)

2. Računanje stope promjene i stope rasta svih podsustava, za svaku inačicu.
3. Računanje stope promjene i stope rasta svih komponenata; za svaku inačicu onih podsustava koji imaju relativno više stope rasta i stope promjene.
4. Komponente koje imaju relativno više stope rasta i stope promjene, smatraju se potencijalnim kandidatima za restrukturiranje.

Metodu *Evolution Identification Using Historical Information* u ovom radu koristimo za praćenje promjena strukture razvojnog okvira referentne arhitekture u cilju otkrivanja potencijalnih nedostataka same strukture i pronalaženja komponenata koje je potrebno restrukturirati ili ponovno oblikovati (V3) kako bi dobili poboljšanju inačicu (V4). Struktura promatranog razvojnog okvira referentne arhitekture sastoji se od tri razine: razina podsustava, razina komponente, i razina klase. Svaka razina sastoji se od jednog ili više elemenata. Slika 81 prikazuje strukturu okvira referentne arhitekture koja predstavlja i organizacijsku i strukturu implementacije. Elementi podsustava "*common*" se koriste u ostalim podsustavima, dok se elementi podsustava "*client*" i "*server*" ne mogu koristiti u drugim podsustavima.



Slika 81 Struktura razvojnog okvira referentne arhitekture

Na temelju podataka o strukturi i promjenama razvojnog okvira referentne arhitekture koje su se događale u svakoj od promatranih inačica, prikupili smo slijedeće vrijednosti:

- Veličinu razvojnog okvira, svakog podsustava i svih komponenata u obliku broja klasa, broja metoda i broja linija izvornog koda (V1-V4).

- Stopu promjene, kao postotak izmjena klasa u odnosu na prethodnu inačicu. Za izračun postotka promjene potrebno je promatrati dvije najbliže inačice promatranog elementa (V1-V3).
- Stopu rasta kao postotak klasa koje su dodane (ili uklonjene) u novoj inačici u odnosu na prethodnu inačicu. Relativni broj novih klasa (razlika dodanih i uklonjenih) predstavlja stopu rasta promatranog elementa (V1-V3).

Mjerenje stope rasta i stope promjene radili smo za inačice (V1-V3). Rezultati mjerenja služe nam za pronalaženje komponenata koje je potrebno posebno prepravljati kod razvoja najnovije inačice (V4). Prepravljanje se odnosi na inačicu (V3) nakon čega smo dobili inačicu (V4) koja predstavlja značajno izmijenjeni okvir. Nije bilo smisla računati stope promjena između inačica (V3-V4) jer su one vrlo značajne i nemaju posebno analitičko značenje, već smo pažnju usmjerili na druge metrike koje pokazuju stabilnost iz druge (statične) perspektive.

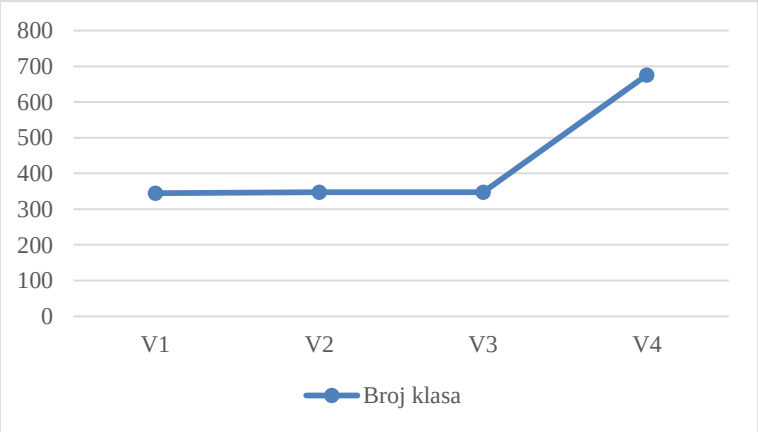
6.4.1.1. Rezultati i interpretacija podataka

Tablica 27 prikazuje broj klasa koje pripadaju svakom od podsustava razvojnog okvira. Razvojni okvir se sastojao od **3** podsustava (A,B,C) u prvoj inačici (V1); u četvrtoj inačici (V4) koja je rezultat naknadnih poslovnih i tehnoloških zahtjeva, te prepravljanja (engl. *refactoring*) razvojnog okvira na temelju rezultata mjerenja za vrijeme longitudinalnog istraživanja (prema metodi znanost o oblikovanju koju smo koristili kako bi poboljšali artefakte), broj podsustava se povećao na **5** (A,B,C,D,E).

Tablica 27 Broj klasa razvojnog okvira

Podsustav	***Inačica okvira referentne arhitekture***			
	V1	***V2***	***V3***	***V4***
A (*Common*)	138	135	135	167
B (*Server*)	84	87	87	134
C (*Java Client*)	123	125	125	125
D (*Web Client*)	0	0	0	231
E (*Mobile Client*)	0	0	0	18
Ukupno	**345**	**347**	**347**	**675**

Slika 82 prikazuje kretanje broja klasa za vrijeme evolucije razvojnog okvira referentne arhitekture u razdoblju od prve inačice (V1) do zadnje inačice (V4) koja je značajno narasla u odnosu na sve prethodne inačice. Stopa rasta broja klasa razvojnog okvira je niska, od prve (V1) do treće (V3) inačice; broj klasa u prvoj inačici je bio 345, broj metoda 2984, broj linija koda 33379, dok je u trećoj inačici taj broj neznatno porastao; broj klasa je bio 347, broj metoda 3055, broj linija koda 34130. Međutim, zadnja inačica (V4) je značajno izmijenjena, korištene su nove mogućnosti i tehnike programiranja, broj klasa je narastao na 675, broj metoda 4907, broj linija koda 48748. Ukupni broj klasa povećao se u za 194% od treće do četvrte inačice razvojnog okvira.



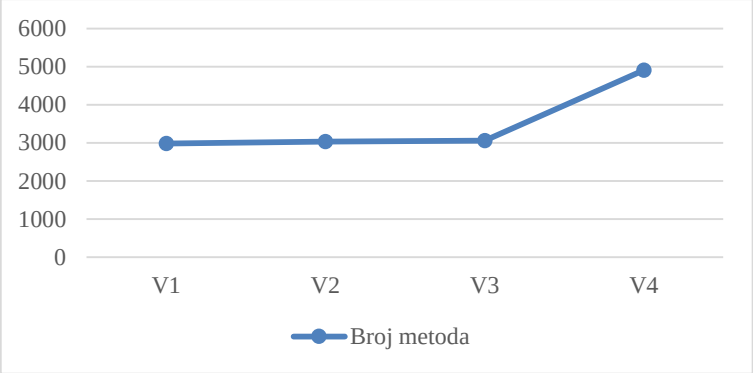
Slika 82. Evolucija veličine razvojnog okvira prema broju klasa

Tablica 28 prikazuje broj metoda koje pripadaju svakom od podsustava razvojnog okvira. Prosječan broj metoda po klasi je 8,6 u prvoj inačici, 8,7 u drugoj inačici, 8,8 u trećoj i 7,2 u četvrtoj inačici. Od prve do treće inačice primjetan je neznatan rast broja metoda po jednoj klasi.

Tablica 28 Broj metoda razvojnog okvira

Podsustav	***Inačica razvojnog okvira***			
	V1	***V2***	***V3***	***V4***
A (*Common*)	1476	1467	1477	1664
B (*Server*)	578	595	602	750
C (*Java Client*)	930	967	976	976
D (*Web Client*)	0	0	0	1446
E (*Mobile Client*)	0	0	0	71
Ukupno	**2984**	**3029**	**3055**	**4907**

Slika 83 prikazuje rast broja metoda za vrijeme evolucije razvojnog okvira, od inačice V1 do inačice V4 koja je značajno narasla u odnosu na sve prethodne inačice. Ukupni broj metoda povećao se u za 160% od treće do četvrte inačice razvojnog okvira.



Slika 83 Evolucija razvojnog okvira prema broju metoda

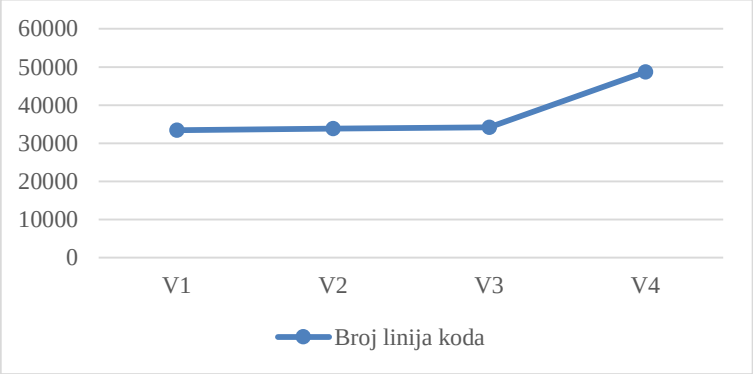
Tablica 29 prikazuje broj linija izvornog programskog koda koji pripada svakom od podsustava razvojnog okvira. Prosječan broj linija koda po klasi je 96 u prvoj inačici, 97 u drugoj inačici, 98 u trećoj i 72 linije koda u četvrtoj inačici. Od prve do treće inačice primjetan je neznatan rast broja linija izvornog programskog koda po klasi, dok je kod zadnje inačice (V4) koja je znatno veća od prethodnih po broju klasa, taj broj značajno pao (72).

Tablica 29 Broj linija izvornog koda razvojnog okvira

Podsustav	***Inačica razvojnog okvira***			
	V1	***V2***	***V3***	***V4***
A (*Common*)	13396	13299	13381	15342
B (*Server*)	10140	10307	10426	11845
C (*Java Client*)	9843	10250	10323	10323
D (*Web Client*)	0	0	0	10612
E (*Mobile Client*)	0	0	0	616
Ukupno	**33379**	**33856**	**34130**	**48122**

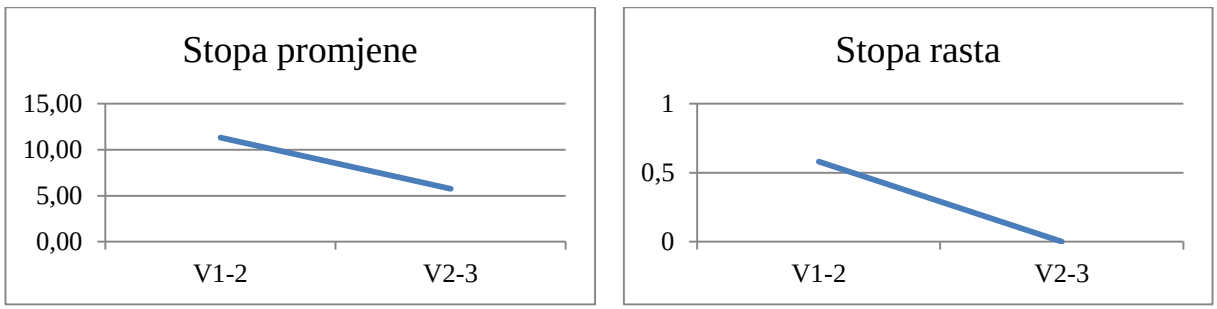
Slika 84 prikazuje rast broja linija izvornog koda za vrijeme evolucije razvojnog okvira, od inačice V1 do inačice V4 koja je značajno narasla u odnosu na sve prethodne inačice. Ukupni broj linija izvornog koda povećao se u za 142% od treće do četvrte inačice razvojnog okvira.

Broj klasa u prve tri inačice nije se značajno mijenjao, što se tumači kao pokazatelj povećanja zrelosti razvojnog okvira. Veća zrelost ujedno znači i povećanje strukturne stabilnosti. Međutim, broj klasa se značajno povećao u četvrtoj inačici, što nije rezultat smanjena stabilnosti, već je taj broj porastao zbog prelaska na nove paradigme (*Aspects, Java 1.7, Annotations, Ajax*). Za očekivati je da će se u slijedećoj inačici (V5) ponovno povećati zrelost i stabilnost razvojnog okvira.



Slika 84 Evolucija razvojnog okvira prema broju linija koda

Slika 85 prikazuje stopu promjene i stopu rasta razvojnog okvira; stopa promjene je veća od stope rasta za prve tri inačice. Stopa promjene kreće se od 11,30% između prve i druge inačice, 5,76% između druge i treće. Zadnja inačica (V4) nije prikazana na dijagramu, ne prati postojeći trend, znatno je rasla i bila izmijenjena. To je posljedica značajnih promjena u arhitekturi razvojnog okvira zbog uvođenja nove vrste klijenata (*web, mobile*), te promjene u arhitekturi razvojnog okvira referentne arhitekture koje su povezane sa uvođenjem ostalih novih tehnologija i funkcionalnosti.



Slika 85 Stope promjene i rasta razvojnog okvira (V1-V3)

Smanjivanje stope rasta od 0,58% do 0% u prve tri inačice, pokazatelj je povećanja strukturne stabilnosti. Značajno povećanje u četvrtoj inačici (V4) koje nije prikazano na dijagramu, rezultat je prelaska na nove paradigme i ne ukazuje na automatsko smanjivanje stabilnosti razvojnog okvira.

Kao zaključak promatranja razvoja aplikacijskog razvojnog okvira od prve do zadnje inačice navodimo slijedeće činjenice:

- Veličina razvojnog okvira se konstantno povećava.
- Stope promjena su se smanjivale osim u slučaju značajne izmjene arhitekture kod prelaska na inačicu (V4).
- Stope rasta koje se smanjuju u novim inačicama ukazuju na povećanje stabilnosti razvojnog okvira.

Smanjivanje stope rasta i stope promjene ukazuje na povećavanje strukturne stabilnosti i evoluciju razvojnog okvira na pravilan način. Stopa promjene koja je manja od 50% smatra se prihvatljivom. Međutim, ova zapažanja odnose se samo na razinu cjelokupnog razvojnog okvira, moguće je da se njegovi podsustavi od kojih se sastoji ne ponašaju na isti način kao i razvojni okvir.

6.4.1.2. Evolucija podsustava okvira referentne arhitekture

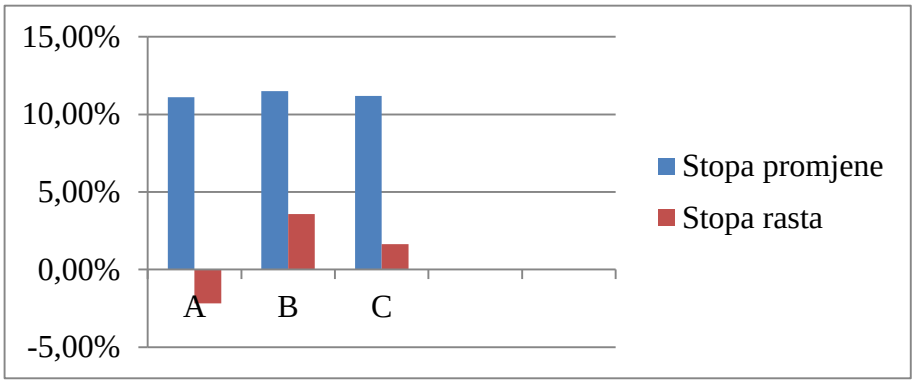
Promatranjem okvira referentne arhitekture kao cjeline, prikupljanjem podataka o njegovom rastu i promjenama koje su se događale između nekoliko njegovih inačica, možemo dobiti dojam općenito o njegovom razvoju. Međutim, evolucija razvojnog okvira kao cjeline, njegove stope rasta i stope promjene, vjerojatno se razlikuju od stopa rasta i stopa promjene njegovih podsustava od kojih se on sastoji. Kako bi istražili činjenice o promjenama koje su se događale na svakom od podsustava između njihovih inačica, promatrali smo svaki podsustav pojedinačno. Tablica 30 prikazuje stope promjene, stope rasta i relativne veličine svakog od podsustava okvira referentne arhitekture između inačica V1 i V2.

Tablica 30 Stope promjene i rasta podsustava V1-V2

Podsustav	***Stopa promjene***	***Stopa rasta***	***Relativna veličina***
A (*Common*)	11,11%	-2,17%	38,90%
B (*Server*)	11,49%	3,57%	25,07%
C (*Java Client*)	11,20%	1,63%	36,02%

Podsustav B ima najveću stopu rasta od 3,57%, podsustav C ima stopu rasta 1,63%. Relativna veličina podsustava B je 25,7% u odnosu na cijeli razvojni okvir, dok je relativna veličina podsustava C preko 36%.

Ako promatramo stope promjene, vidi se da je podsustav B značajno izmijenjen od inačice V1 do inačice V2, budući je njegova stopa promjene 11,49%, što je viša stopa od stope promjene cijelog razvojnog okvira, 11,30%. Podsustav C ima stopu promjene 11,20%, što je niža stopa od stope promjene razvojnog okvira.



Slika 86 Stope promjene i rasta podsustava V1-V2

Tablica 31 prikazuje podatke promjena i rasta između druge i treće inačice razvojnog okvira (V2-V3). Tada nije bilo rasta, dok su izmjene bile vrlo male. Takvo stanje ukazuje na veliku stabilnost sustava, u našem slučaju radi se o vremenskom periodu od godinu dana. Upravo takva situacija predstavlja povoljni moment za značajnije restrukturiranje i poboljšanje sustava.

Tablica 31 Stope promjene i rasta podsustava V2-V3

Podsustav	***Stopa promjene***	***Stopa rasta***	***Relativna veličina***
A (*Common*)	4,44%	0,00%	38,90%
B (*Server*)	4,60%	0,00%	25,07%
C (*Java Client*)	8,00%	0,00%	36,02%

Prema stopama promjena i stopama rasta, te njihovoj relativnoj veličini, može se zaključiti da su podsustav C i podsustav B prioritetni kandidati za daljnje istraživanje. Podsustav C predstavlja razvojni okvir za korisnička sučelja, koji se zbog čestih tehnoloških promjena po

prirodi učestalo mijenja, ne koristi se za razvoj svih već samo nekih proizvoda (aplikacija); možemo zaključiti da on nije tipičan predstavnik podsustava, te da se njegova evolucija može protumačiti kao utjecaj okoline. Podsustav B tipičan je predstavnik podsustava, koristi se kao razvojni okvir za programiranje poslovne logike svih proizvoda od kojih se sastoji promatrana linija za proizvodnju poslovnog softvera, ima relativno veće stope promjena i rasta u odnosu na ostale podsustave, stoga ćemo detaljnije istražiti njegova svojstva i karakteristike.

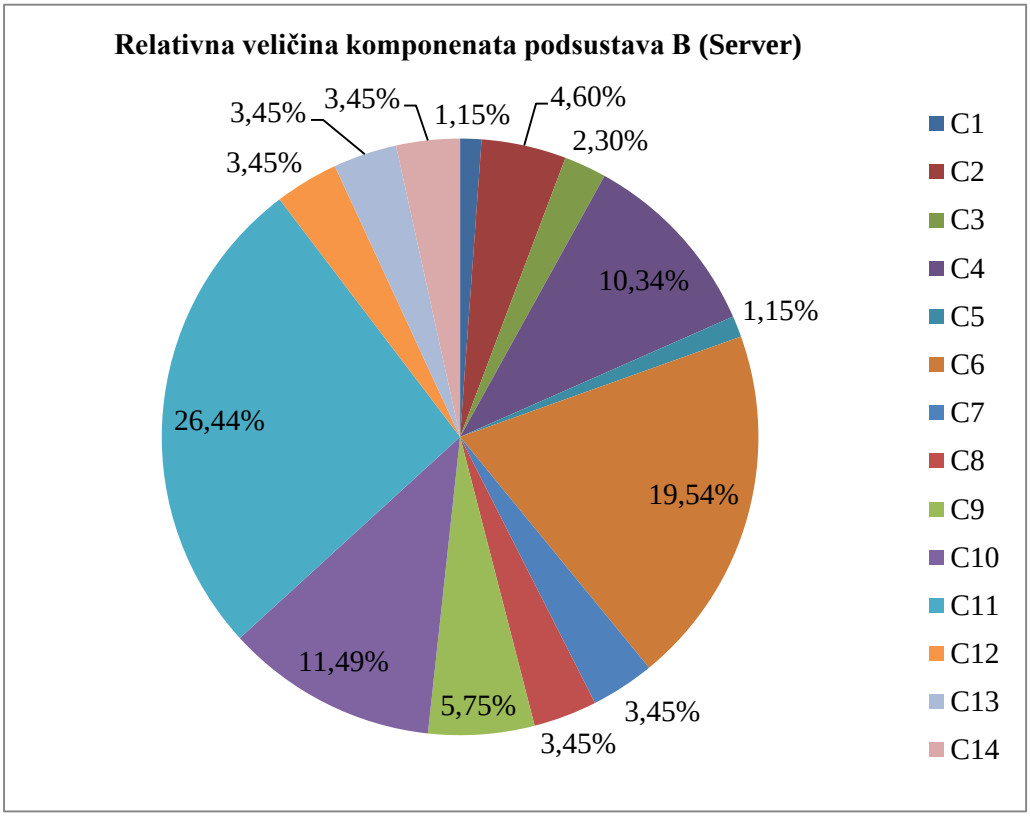
6.4.1.3. Evolucija podsustava B

Podsustav B pokazuje najveću stopu promjene i stopu rasta od svih podsustava, što upućuje na to da je taj podsustav vjerojatni kandidat za restrukturiranje i/ili reinženjering. Stoga je potrebno detaljnije istražiti pojedinačne komponente samog podsustava, njihove stope rasta i stope promjena u odnosu na stope rasta i stope promjena podsustava kao cjeline kojem komponente pripadaju. Podsustav B se sastoji od 14 komponenata koje označavamo sa C1 do C14. Prije početka rada na poboljšanju inačice (V3) kako bi dobili novu poboljšanu inačicu (V4), analizirali smo podsustav B. Tablica 32 prikazuje broj klasa, metoda i linija izvornog programskog koda svih komponenata od kojih se podsustav B sastoji. U razdoblju između inačice V1 do inačice V3 najviše su se mijenjale komponente C6, C10 i C11.

Tablica 32 Komponente podsustava B

Oznaka komponente	***Broj klasa***			***Broj metoda***			***Broj linija***		
	V1	**V2**	**V3**	**V1**	**V2**	**V3**	**V1**	**V2**	**V3**
C1	1	1	1	17	17	17	224	224	224
C2	4	4	4	14	14	14	230	230	230
C3	2	2	2	17	17	17	238	238	238
C4	9	9	9	109	109	109	2108	2108	2108
C5	0	1	1	0	4	4	0	12	12
C6	17	17	17	**142**	**143**	**146**	2740	2772	2828
C7	3	3	3	15	15	15	174	174	174
C8	3	3	3	7	7	7	44	44	44
C9	3	5	5	13	24	24	286	406	406
C10	10	10	10	**46**	**46**	**50**	761	761	829
C11	23	23	23	**150**	**151**	**151**	2635	2638	2633
C12	3	3	3	15	15	15	181	181	181
C13	3	3	3	16	16	16	220	220	220
C14	3	3	3	17	17	17	299	299	299
	84	**87**	**87**	**578**	**595**	**602**	**10140**	**10307**	**10426**

Slika 87 prikazuje relativnu veličinu komponenata podsustava B. Komponente C6, C10 i C11 imaju relativnu veličinu 19, 11 i 26 posto u odnosu na cijeli podsustav, pripadaju grupi od 3 najveće komponente koje zajedno čine preko 57% (V3).



Slika 87 Relativna veličina komponenata podsustava B (V3)

Tablica 33 prikazuje evoluciju komponenata C6, C9, C10 i C11. Njihove stope promjena i stope rasta su relativno najviše u odnosu na druge komponente, za razdoblje između inačice V1 i V3. Budući se broj klasa za sve tri (C6, C10 i C11) komponente nije mijenjao između inačica V1 i V3, kod izračuna stope rasta uzeli smo broj metoda umjesto broja klasa. Stope rasta su relativno najveće kod ove tri komponente. Komponenta C9 ima relativno najveću stopu rasta, ali pošto se radi o maloj komponenti od svega 3 klase koja nije tipični predstavnik komponenata podsustava B, nju nismo uzeli u daljnje razmatranje.

Tablica 33 Evolucija komponenata C6, C9, C10 i C11 (V1-V3)

Komponenta	***Stopa promjene***	***Stopa rasta***
C6	17,65%	2,82%
C10	20,00%	8,70%
C11	17,39%	0,67%
C9	100,00%	84,62%

Komponente C6, C10 i C11 imaju relativno najveću stopu rasta. Uzroci promjena u ovim komponentama odnose se na činjenicu da se radi o promjenama u dizajnu komponenata za pristup poslovnim podacima uslijed čestih izmjena inačica baza podataka i drugih izvora podataka raznih dobavljača.

6.4.1.4. Zaključak analize evolucije okvira referentne arhitekture

Rezultati promatranja evolucije razvojnog okvira referentne arhitekture slični su rezultatima istraživanja (Mattson & Bosch, 1999); potvrđuju da postoji razlika u ponašanju cijelog sustava u odnosu na ponašanje njegovih podsustava. Razvojni okvir je relativno stabilan u svom razvoju, dok se stope rasta i stope promjene njegovih podsustava B i C razlikuju od sustava u cjelini. Razlike sustava u cjelini i podsustava B i C nisu lako uočljive osim ako se podsustavi ne istražuju odvojeno od istraživanja sustava. Podsustav B kao tipičan i važan podsustav razvojnog okvira za poslovne aplikacije odabran je kao kandidat za restrukturiranje na temelju podataka istraživanja koje smo analizirali od inačice V1 do inačice V3, prema metodi znanost o oblikovanju koja se koristi u ovom radu. Podsustav B smo značajno restrukturirali za vrijeme prepravljanja i razvoja inačice V4.

6.4.2. Procjena stabilnosti okvira referentne arhitekture[15]

U drugom koraku ocjenjivanja evolucije okvira referentne arhitekture za procjenu stabilnosti razvojnog okvira koristimo *Stability Assessment* (Mattson & Bosch, 1999) metodu koja se bazira na upotrebi metrika izvornog programskog koda. Navedena metoda je u biti proširena metoda za kvantitativnu procjenu stabilnosti objektno-orijentiranog okvira (J. Bansiya, 1998) koja određuje pravila za stabilnost (engl. *stability rules*) te definira novu metriku; relativni opseg promjena (engl. *relative extent of change*) kao pokazatelj stabilnosti arhitekture okvira. Procjena stabilnosti važan je pokazatelj prije nego se krene u prepravljanje postojeće inačice (V3). Procjenu smo radili kako bi utvrdili trenutnu zrelost okvira (V3) za njegovo prepravljanje. Nije za očekivati da će inačica (V4) imati karakteristike stabilnog okvira, već je za očekivati da će inačica (V3) biti dovoljno stabilna da bi se bez velikog rizika moglo pristupiti njenoj preradi i prilagodbi novim tehnologijama.

[15] Stability Assesment (Mattsson, 2000)

Rezultate procjene stabilnosti promatranog razvojnog okvira, koji ukazuju na njegovu stabilnost, uspoređujemo sa navedenim empirijski definiranim pravilima za stabilnost. Metodu za procjenu stabilnosti razvojnog okvira referentne arhitekture primijenili smo kroz nekoliko koraka (Mattson & Bosch, 1999):

1. ***Identifikacija karakteristika evolucije razvojnog okvira***: za procjenu stabilnosti okvira potrebno je odabrati prikladne pokazatelje. Bansiya razlikuje dvije kategorije karakteristika evolucije nekog okvira, jedna se odnosi na arhitekturu a druga na individualne klase. Promjene karakteristika arhitekture odnose se na promjene u interakciji između klasa te na promjene strukture klasa u hijerarhiji nasljeđivanja. Druga kategorija, promjene karakteristika individualnih klasa odnose se na promjenu strukture (podaci), funkcionalnosti (metode) i povezanosti (reference) između klasa. Za provjeru stabilnosti razvojnog okvira, koristili smo samo metrike koje se odnose na prvu kategoriju karakteristika evolucije okvira, dok smo za analizu evolucije karakteristika individualnih klasa, umjesto metrika koristili promjene izvornog koda na razini klasa, metoda i linija programskog koda.
2. ***Izbor metrika za procjenu svake pojedine kategorije evolucije***: kod izbora metrika razmatrali smo nekoliko izvora najčešće korištenih objektno orijentiranih, strukturnih i metrika za okvire (Jagdish Bansiya, 1997; Chidamber & Kemerer, 1994a; Erni & Lewerentz, 1996; Li & Henry, 1993; Martin, 1994b; McCabe & Butler, 1989), prema metodi koju koristimo odabrali smo i dodali slijedeće metrike: EFC i ABS, tablica 34. U analizi metrika ovaj put nismo razmatrali podsustav „*C Java Client*“ budući ga ne planiramo značajnije koristiti u razvoju poslovnih aplikacija u budućnosti te budući da ga nismo mijenjali od inačice V3.

Tablica 34 Odabrane metrike arhitekture

Metrika	***Naziv***
DSC	Ukupan broj klasa
NSI	Broj klasa koje nasljeđuju samo od jedne klase prethodnika
NMI	Broj klasa koje nasljeđuju od više klasa prethodnika
ADI	Prosječna dubina stabla nasljeđivanja
AWI	Prosječna širina stabla nasljeđivanja (prosječan broj klasa nasljednica)
EFC	Odvodno spajanje (engl. *efferent coupling*)
ACIS	Prosječni broj javnih (engl. *public)* metoda u svim klasama
ABS	Postotak apstraktnih klasa

Ako bi ga uzeli u obzir dobili bismo nešto bolje karakteristike sustava, međutim naš dugoročni cilj je poboljšanje artefakata referentne arhitekture za razvoj *web* i mobilnih aplikacija.

3. ***Prikupljanje podataka***: metrike za sve inačice razvojnog okvira, prikupljene su korištenjem istih alata, CodePro Analytix^{Tm} i JHawk.

4. ***Analiziranje promijenjenih karakteristika arhitekture te izračun relativnog opsega promjene***: podaci metrika za svaku od inačica razvojnog okvira su analizirani, normalizirani, te je za njih izračunat relativni opseg promjena (engl. *relative extent of change*) .

5. ***Odlučivanje o zadovoljavanju pravila stabilnosti***: usporedba rezultata dobivenih mjerenjem svake od inačica referentne arhitekture sa skupom pravila za stabilnost okvira, od kojih smo koristili 5 od mogućih 6 pravila, tablica 35. Jedino pravilo, pravilo broj 6 u tablici, za koje nismo radili usporedbu, zbog nemogućnosti mjerenja metrike, odnosi se na analizu prosječnog broja drugih klasa koje promatrana klasa može koristiti (engl. *average number of distinct classes that a class may collaborate with*). Za sva ostala pravila stabilnosti, napravili smo usporedbu rezultata promatranog razvojnog okvira sa referentnim vrijednostima pravila za stabilnost.

Tablica 35 Pravila stabilnosti okvira (Mattson & Bosch, 1999)

Pravilo	***Opis pravila***
1	Stabilne okvire karakterizira povećavanje prosječne dubine stabla nasljeđivanja (ADI > **2,1**) te smanjivanje prosječne širine stabla nasljeđivanja (AWI < **0,85**) tijekom razvoja novih inačica.
2	Omjer NSI/DSC kod stabilnih aplikacijskih okvira je iznad **0,8** ako se u pravilu, rijetko koristi višestruko nasljeđivanje. Broj klasa slijednika kod stabilnih aplikacijskih okvira prelazi **80** posto.
3	Stabilni aplikacijski okvir ima vrijednost metrike normaliziranog opsega promjena (engl. *normalized-extent-of-change*) manji od **0,4**.
4	Stabilni aplikacijski okvir ima vrijednost metrike opsega promjena (engl. *relative-extent-of-change*) manju od **25** posto.
5	Aplikacijski okviri postaju stabilni tek u vrijeme nakon razvoja i korištenja njegove **treće** do **pete** inačice.
6	Normalizirani prosječan broj klasa koje neka klasa može koristiti, kreće se od **1.0** i niže.

6.4.2.1. Rezultati usporedbe sa pravilima stabilnosti

Prvo pravilo stabilnosti: promatranjem vrijednosti metrika u tablici 36 vidljivo je da je prosječna dubina nasljeđivanja (ADI) stabilna od prve do treće inačice (oko 3,31), u zadnjoj inačici je nešto smanjena (3,03), dok se prosječna širina stabla nasljeđivanja (AWI) od **0,66** nije mijenjala sve do zadnje inačice kada je značajno povećana (od 0,66 na 1,36). Ako usporedimo te vrijednosti sa referentnim vrijednostima prvog pravila, možemo zaključiti da su vrijednosti metrike aplikacijskog okvira za prosječnu dubinu nasljeđivanja (ADI) iznad 2,1 (**3,03**) što možemo protumačiti kao zadovoljavanje prvog pravila stabilnosti. Vrijednost metrike za prosječnu širinu nasljeđivanja (AWI) su bile ispod 0,85 (**0,66**) prije inačice (V4), međutim zbog uvođenja potpuno novih klasa za *web* aplikacije za koje još ne postoji dobro razrađena hijerarhija nasljeđivanja, vrijednost od **1,36** prelazi referentnu vrijednost. Ovaj dio prvog pravila nije zadovoljen u zadnjoj inačici; njegovo poboljšanje je predmet novog ciklusa poboljšanja koji je u tijeku.

Tablica 36 Vrijednosti metrika okvira referentne arhitekture

Metrika	***Inačica***			
	V1	*V2*	*V3*	*V4*
DSC	345	347	347	550
NSI	200	207	207	356
NMI	0	0	0	0
ADI	3,30	3,31	3,31	3,03
AWI	0,66	0,66	0,66	1,36
EFC	319	321	321	358
ACIS	8,09	8,14	8,21	5,26
ABS	11,80	11,50	11,50	17,00

Drugo pravilo stabilnosti: promatranjem vrijednosti odnosa NSI/DSC u tablici 37, vidimo da se vrijednost omjera povećava od 0,58 do 0,60 u prve tri inačice, i naglo pada do **0,35** u zadnjoj inačici. Usporedbom tih vrijednosti sa referentnom vrijednošću (iznad 0,8), na prvi pogled možemo zaključiti da aplikacijski okvir ne zadovoljava drugo pravilo stabilnosti. Međutim, činjenica je da promatrani razvojni okvir intenzivno koristi „višestruko“ nasljeđivanje (više od 11% *Java interface*), te da je broj klasa koje nasljeđuju samo od jedne klase prethodnika ako isključimo *Java interface* zbog toga implicitno manji, što upućuje na to da se ovo pravilo može smatrati djelomično zadovoljenim.

Tablica 37 Odnos NSI/DSC

Metrika	***Inačica***			
	V1	*V2*	*V3*	*V4*
DSC	345	347	347	550
NSI	200	207	207	194
NSI/DSC	*0,58*	*0,60*	*0,60*	*0,35*

Treće pravilo stabilnosti: tablica 38 prikazuje vrijednosti metrika normaliziranog opsega promjena (engl. *normalized extent of change*) koje se računaju tako da se vrijednosti metrika prve inačice uzimaju kao osnovica, dok se ostale vrijednosti dobiju dijeljenjem vrijednosti promatrane inačice sa vrijednostima prethodne inačice. Normalizirani opseg promjene jedne inačice, rezultat je zbrajanja svih normaliziranih vrijednosti za tu inačicu. Za promatrani aplikacijski okvir, normalizirani opseg promjena kreće se od 0,03 u prvoj inačici do **1,72** u zadnjoj inačici. Usporedimo li vrijednosti metrike iz tablice sa referentnom vrijednošću (<0,4), možemo zaključiti da razvojni okvir do inačice (V3) zadovoljava treće pravilo stabilnosti. Nova inačica (V4) koja predstavlja potpuno novu generaciju referentne arhitekture nastala je kao rezultat poboljšanja stabilne inačice (V3); za očekivati je da će se i ona stabilizirati tijekom njenog korištenja za razvoj poslovnih *web* aplikacija te tijekom budućih prepravljanja samog aplikacijskog okvira.

Tablica 38 Normalizirane vrijednosti metrika arhitekture

Metrika	***Inačica***			
	V1	***V2***	***V3***	***V4***
DSC	1	1,01	1,00	1,59
NSI	1	1,04	1,00	0,94
NMI	1	1,00	1,00	1,00
ADI	1	1,00	1,00	0,92
AWI	1	1,00	1,00	2,06
EFC	1	1,01	1,00	1,12
ACIS	1	1,01	1,01	0,64
ABS	1	0,97	1,00	1,48
Average change(Vi)	**8**	**8,03**	**8,01**	**9,70**
Normalized-extent-of-change	**0**	**0,03**	**0,01**	**1,72**

Četvrto pravilo stabilnosti: Ovo pravilo je slično prethodnom pravilu jer nastoji izmjeriti stabilnost razvojnog okvira, ali sa različite razine apstrakcije, budući se odnosi na metrike klasa

razvojnog okvira umjesto na metrike na razini apstrakcije referentne arhitekture. Prema (Mattson & Bosch, 1999) ova metrika se mjeri na način da se uspoređuje svaka klasa u odnosu na njenu prethodnu inačicu uzimajući u obzir eventualne promjene svih metrika pojedinačno koje su karakteristične za apstrakciju na razini klase. U ovom radu predstavljamo drugačiji pristup mjerenja relativnog opsega promjena između inačica aplikacijskog okvira, i to na tri razine: broj izmijenjenih klasa, broj izmijenjenih metoda, te broj izmijenjenih linija programskog koda. Izmjene se odnose na dodavanje novih, uklanjanje postojećih ili izmjene postojećih klasa, metoda ili linija programskog koda. Tablica 39 prikazuje rezultate relativno malih promjena u odnosu na prethodne inačice, pri čemu se najveća relativna promjena odnosi na promjene V1/2 od 11,30% pri čemu je samo 39 klasa u odnosu na ukupno 345 bilo promijenjeno, što je manje od referentne vrijednosti od 25%. Ako bi uzeli u obzir svih 11 metrika prema (Mattson & Bosch, 1999), pod pretpostavkom da se u obzir ne uzimaju naslijeđena svojstva, te broj promijenjenih klasa (39) relativno usporedili sa maksimalnim brojem klasa 345, tada bi dobili isti rezultat kao i kod predloženog načina mjerenja. Stoga iako na drugačiji način možemo sa sigurnošću tvrditi da aplikacijski okvir (V3) zadovoljava četvrto pravilo stabilnosti, te da je okvir zreo za prepravljanje.

Tablica 39 Relativni opseg promjena

	Promjene					
	Klase		***Metode***		***Linije koda***	
	V1/2	V2/3	V1/2	V2/3	V1/2	V2/3
Broj jedinica promjene	39	20	116	31	1468	290
Maksimalni broj jedinica	345	347	2984	3029	33379	33856
Relative extent of change	11,30%	5,76%	3,89%	1,02%	4,40%	0,86%

Peto pravilo stabilnosti: zadovoljavanje ovog pravila može se provjeriti na način da se svako od prethodnih pravila stabilnosti ispita za sve inačice aplikacijskog okvira. U tablici 40 navedena su pravila stabilnosti, Pravilo 1 do Pravila 4, te njihova usporedba sa sve četiri inačice razvojnog okvira. Oznaka „DA“ ukazuje na to da je pravilo stabilnosti zadovoljeno, dok oznaka „NE“ ukazuje na to da pravilo stabilnosti nije zadovoljeno. Rezultati ovog istraživanja, koji se odnose na sve inačice promatranog razvojnog okvira, ukazuju na to da sve inačice uglavnom zadovoljavanju sva pravila stabilnosti okvira.

Tablica 40 Pravila stabilnosti od 1 do 4

Inačica okvira	*Pravilo 1*	*Pravilo 2*	*Pravilo 3*	*Pravilo 4*
V1	DA	„DA“	DA	DA
V2	DA	„DA“	DA	DA
V3	DA	„DA“	DA	DA
V4	Djelomično	„DA“	NE	NE

Možemo zaključiti da aplikacijski okvir predložene referentne arhitekture zadovoljava pravila stabilnosti prema (Mattson & Bosch, 1999) i to u svim svojim inačicama od V1 do V3, dok je zadnja inačica (V4) zbog značajnih izmjena predmet budućih aktivnosti s ciljem njegovog stabiliziranja u inačicama većim od inačice V4.

Razlike između metode za procjenu stabilnosti koju smo koristili i originalne metode su:

- Izmjenu grupe metrika koje koristimo za procjenu stabilnosti aplikacijskog okvira, a koje su prema (Mattson & Bosch, 1999) dozvoljene, ukoliko se radi o specifičnoj okolini, kao što je u našem slučaju - linija za proizvodnju softvera.
- Izmjenu četvrtog pravila stabilnosti, relativnog opsega promjene, kod kojeg koristimo klase, metode ili linije programskog koda umjesto metrika.

6.4.3. Indeks zrelosti referentne arhitekture[16]

Metrika pod imenom *Software maturity indeks* (SMI) ili indeks zrelosti softvera, koristiti se i za mjerenje stabilnosti softverskih proizvoda, primjerice za mjerenje stabilnosti okvira referentne arhitekture, poslovnih komponenata, knjižnice klasa, i slično. Indeks zrelosti se izračunava temeljem izraza:

$$SMI = (M - (A + C + D))/M \quad (2)$$

M = ukupni broj klasa u trenutnoj inačici softvera

A = broj novih klasa u trenutnoj inačici softvera

C = broj klasa u trenutnoj inačici koje su izmijenjene u odnosu na prethodnu inačicu

D = broj uklonjenih klasa iz prethodne inačice u trenutnoj inačici

[16] Software Maturity Index (SMI) - IEEE 982.1-1988: M-number of modules in current version, A = number of added modules in current version, C = number of changed modules in current version, D = number of deleted modules in current version compared to the previous version. *SMI = 1 - N/M, where M is the total number of modules in the current version of the system and N is the number of modules added, changed or deleted between the previous version and this one.*

SMI stavlja u odnos ukupni broj modula i broj izmjena u modulima između dvije inačice softverskog proizvoda. Indeks zrelosti ukazuje na stabilnost softverskog proizvoda koja je to veća što se indeks zrelosti od nule više približava vrijednosti **1**. Korelacija vremena koje je potrebno za razvoj nove inačice softvera i indeksa zrelosti, može poslužiti kao indikator za predviđanje potrebnog napora za održavanje trenutne inačice softverskog proizvoda.

Tablica 41 prikazuje indeks zrelosti za inačice okvira referentne arhitekture kojeg istražujemo. Stabilnost okvira se s vremenom mijenjala, inačica V2 se može smatrati stabilnom, indeks zrelosti je 0,89, broj novih klasa u odnosu na prethodnu inačicu je povećan za 6 klasa, izmijenjeno je 29 klasa a uklonjene su 4 klase. Inačica V3 je još stabilnija, indeks zrelosti od 0,94 ukazuje na veliku stabilnost aplikacijskog okvira.

U trenutku kada imamo vrlo stabilne inačice artefakata referentne arhitekture, kao što je slučaj sa inačicama koje prethode inačici V4, povoljno je provoditi značajnije pothvate održavanja, stoga smo u to vrijeme velike stabilnosti planirali i proveli prepravljanje (engl. *refactoring*) postojeće inačice V3 kako bi uveli nove tehnološke mogućnosti i odgovorili na nove zahtjeve poslovnih korisnika. Inačica V4 predstavlja značajne izmjene, tehnološki potpuno novu generaciju okvira, indeks zrelosti se vjerojatno značajno smanjio, međutim, za očekivati je da će se tek nova inačica V5 stabilizirati i postići potrebnu zrelost, usporedivu sa zrelošću inačica V2 i V3.

Tablica 41 Indeks zrelosti primijenjen na okvir referentne arhitekture

Metrika	***Inačica***	
	V1/V2	*V2/V3*
Broj novih klasa (A)	6	0
Broj izmijenjenih klasa (C)	29	20
Broj uklonjenih klasa (D)	4	0
Ukupni broj klasa (M)	347	347
SMI = (M - (A + C + D)) / M	**0,89**	**0,94**

6.5. Model za predviđanje promjenjivosti poslovnih komponenata

Napredak organizacija koje razvijaju softver usko je povezan sa prihvaćanjem novih tehnologija. Ukoliko su nove tehnologije značajno drugačije od postojećih, potrebno je ublažiti eventualne rizike koji proizlaze iz procesa njihovog uvođenja u organizaciju; najčešće, provođenjem evaluacije novih tehnologija prije njihove šire primjene unutar organizacije. Evaluacija se može provesti pomoću kontroliranog eksperimenta ili pomoću studije slučaja. U oba slučaja moguće je koristiti metodu *Goal Question Metric* (GQM) (V. R. Basili, Caldiera, & Rombach, 1994) kao okvir za organizaciju evaluacije.

Glavni cilj prepravljanja postojeće inačice (V3) i razvoja najnovije inačice referentne arhitekture (V4) za poslovne aplikacije koja značajno koristi nove tehnologije, njeno je korištenje na projektima u različitim poslovnim organizacijama i sektorima u budućnosti. Prije donošenja odluke o korištenju u širem kontekstu, za veći broj poslovnih organizacija i sektora, radimo evaluaciju programskog okvira referentne arhitekture pomoću studije slučaja (poglavlje 6.3) i kontroliranog tehnološki orijentiranog eksperimenta kojeg detaljno opisujemo u ovom poglavlju. Predmet istraživanja kontroliranog eksperimenta su poslužiteljske poslovne komponente koje primjenjuju okvir referentne arhitekture. Svrha eksperimenta je ispitivanje ***efekta*** primjene okvira referentne arhitekture na razinu održavljivosti (promjenjivosti) poslužiteljskih poslovnih komponenata. Eksperiment provodimo i kako bi potvrdili mogućnost primjene predloženog modela (PR) za procjenu promjenjivosti (engl. *changeability*) poslovnih komponenata u praksi.

ISO / IEC 9126 standard (zamijenjen s ISO / IEC 25010: 2011) održavljivost promatra kao jednu od 6 glavnih značajki kvalitete softverskog proizvoda. IEEE (1990) definira održavljivost kao "lakoću kojom se softverski sustav ili komponenta može mijenjati kako bi se ispravile greške, poboljšale performanse ili drugi atributi, ili kako bi se softver prilagodio izmijenjenom okruženju" (Geraci i ostali, 1991). Održavljivost se dodatno razlaže na pod karakteristike: mogućnost analize (engl. *analyzability*), promjenjivost (engl. *changeability*), stabilnost (engl. *stability*) i provjerljivost (engl. *testability*) (International Organization for Standardization, 2001). Promjenjivost se odnosi na količinu napora koji je potreban da se promijeni softverski sustav (ISO / IEC 9126). U kontekstu linija za proizvodnju softvera, gdje se mnogi programi oslanjaju na zajedničku platformu, tehnička kvaliteta izvornog programskog koda ima svoje specifičnosti ako ga uspoređujemo s običnim sustavom, jer je tehnička kvaliteta važna odrednica održavljivosti (promjenjivosti) softverskog sustava. Model za procjenu održavljivosti

kojeg istražujemo u ovom radu, odnosi se na procjenu pod-karakteristike održavljivosti softverskog sustava, promjenjivosti (engl. *changeability*), budući da se metrike koje model koristi odnose na povezanost sustava sa ostalim sustavima. U prethodnom istraživanju (Rosko, 2013a) predložili smo metriku „odgovornost platforme“ (PR) za predviđanje održavljivosti (promjenjivosti) poslovnih komponenata koje koriste okvir referentne arhitekture linija za proizvodnju softvera.

U ovom istraživanju, evaluaciju okvira referentne arhitekture provodimo pomoću modela „odgovornosti platforme“ (PR) za predviđanje održavljivosti (promjenjivosti) poslovnih komponenata koje taj okvir koriste, i to u organizaciji koja koristi pristup linija za proizvodnju softvera pri razvoju i održavanju poslovnih aplikacija. Broj i kompleksnost izmjena neke komponente u promatranom razdoblju možemo definirati kao mjerilo njene promjenjivosti. Što je veći broj nužnih izmjena koje su se dogodile na nekoj komponenata to je manja njena promjenjivost i obrnuto. Aplikativne komponente tijekom vremena se mijenjaju ukoliko to zahtijevaju korisnici aplikacija ili ukoliko je to uvjetovano promjenama u tehnologiji koja se koristi za njihov razvoj. U ovom radu definicija promjenjivosti poslovne komponente se odnosi na učestalost izmjena koje su uzrokovane promjenama u tehnologiji. Promjenjivost komponenata koja se odnosi na učestalost promjena uzrokovanih korisničkim zahtjevima, također može biti uzrokovana vrstom tehnologije i načinom njene primjene, međutim, taj aspekt promjenjivosti nije predmet ovog eksperimenta.

6.5.1. Definiranje metrika modela (PR)

Model (PR) za predviđanje promjenjivosti sustava poslovnih aplikacija temelji se na kvantitativnim metrikama izvornog programskog koda. U procesu definiranja modela za predviđanje promjenjivosti komponenata koristili smo metodu GQM koja započinje sa definiranjem ciljeva mjerenja, nastavlja se postavljanjem određenih pitanja koji karakteriziraju postavljene ciljeve, završava predlaganjem metrika koje odgovaraju na postavljena pitanja. Ciljevi mjerenja definiraju svrhu, objekt i problem mjerenja, te glavne dionike koji koriste rezultate mjerenja. Svaki cilj se postepeno razgrađuje u obliku pitanja pomoću kojih se najčešće problem mjerenja dijeli na glavne komponente. Svako pitanje se nadalje prevodi u prikladne metrike. U nekim slučajevima jedna metrika može odgovarati na nekoliko postavljenih pitanja.

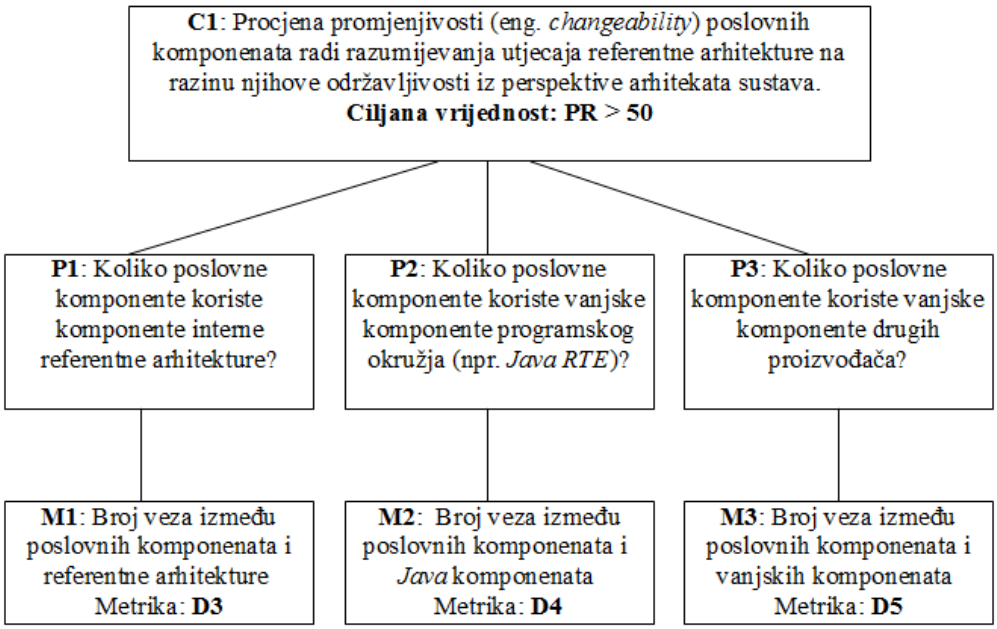
Rezultati primjene GQM metode su upotrebljivi samo ukoliko organizacija koja ju primjenjuje koristi standardne metode za razvoj, prikuplja podatke vezane uz projekte razvoja, te ukoliko poduzima potrebne radnje nakon analize prikupljenih podataka. Organizacija u kojoj

provodimo ovo istraživanje zadovoljava navedene kriterije, stoga očekujemo da će rezultati evaluacije biti upotrebljivi. Slika 88 prikazuje GQM model mjerenja promjenjivosti poslovnih komponenata koji se sastoji se od tri razine:

1. *Konceptualna razina* (C1). **Cilj** se definira za objekt mjerenja. Objekt mjerenja može biti proizvod, proces ili neki drugi resursi. U našem slučaju objekti mjerenja su poslovne komponente. Ciljana vrijednost promjenjivosti poslovnih komponenata prema modelu (PR) je iznad **50**, za koju tvrdimo da predstavlja granicu između lakše održavljive i teže održavljive poslovne komponente.
2. *Operativna razina* (P1, P2, P3). **Pitanja** se koriste za opis načina mjerenja definiranog cilja koji je postavljen na konceptualnoj razini.
3. *Kvantitativna razina* (M1, M2, M3). **Metrike** predstavljaju odgovore na pitanja koji se sastoje od kvantitativnih podataka, metrika izvornog programskog koda.

Definirali smo glavni cilj kao primarnu svrhu definiranja metrika predloženog modela (PR):

Cilj: Procjena promjenjivosti (engl. *changeability*) poslovnih komponenata radi razumijevanja utjecaja referentne arhitekture na razinu njihove promjenjivosti iz perspektive arhitekata sustava.



Slika 88 Cilj, pitanja i metrike prema metodi (GQM)

Cilj mjerenja definira *svrhu* (procjenu), *objekt* (poslovne komponente) i *problem* (promjenjivost) mjerenja, *glavne dionike* (arhitekte sustava) koji koriste rezultate mjerenja. Na

operativnoj razini definiramo niz pitanja povezanih sa postavljenim ciljem. Osim pitanja na slici 88 koje koristimo u modelu za procjenu promjenjivosti, definirali smo i jedno dodatno pitanje koje se odnosi na povezanost okvira referentne arhitekture sa vanjskim komponentama. Zadnji korak GQM metode je definiranje metrika, kao odgovora na postavljena pitanja na operativnoj razini. S obzirom na činjenicu da postoji samo nekoliko istraživanja koja se odnose na definiranje sličnih metrika (Dincel, Medvidovic, & van der Hoek, 2002; Rahman, 2004; Zhang, Deng, Wu, Zhou, & Ma, 2008) definirali smo metrike (D3, D4, D5) koje se odnose na postavljena pitanja na slici 88.

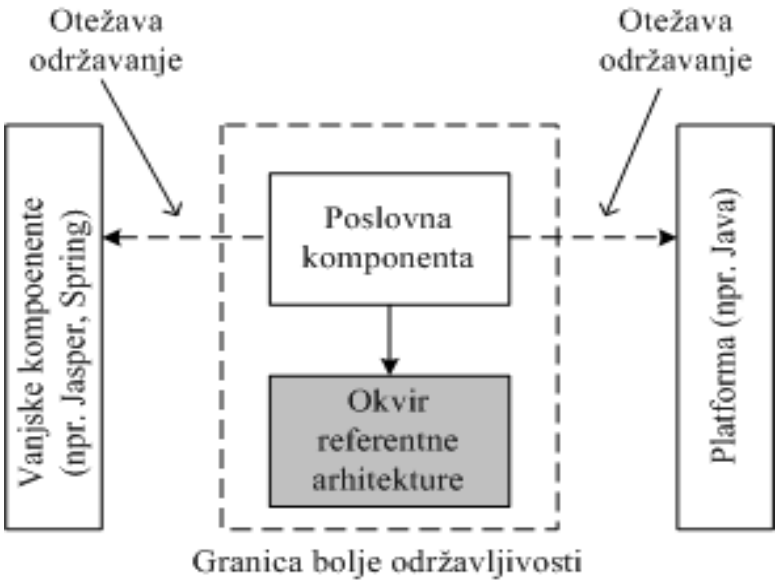
Tablica 42 prikazuje metrike predloženog modela (PR) za predviđanje promjenjivosti sustava poslovnih aplikacija koje koriste referentnu arhitekturu prema pristupu linija za proizvodnju softvera (Rosko, 2013a).

Tablica 42 Predložene metrike promjenjivosti sustava poslovnih aplikacija

Pitanje	***Metrika***	***Definicija***
Koliko okvir referentne arhitekture (platforma) koristi vanjske klase koje nisu dio okvira?	*D1. Platform Efferent Coupling*	Broj jedinstvenih referenci prema klasama programskog okružja (npr. *Java RTE*) od strane klasa okvira referentne arhitekture.
	D2. Platform Efferent Coupling	Broj jedinstvenih referenci prema klasama vanjskih komponenata drugih proizvođača (npr. *Spring*, *Google*, *Apache*, *Jasper*) od strane klasa okvira referentne arhitekture.
Koliko proizvod (aplikacija) i/ili komponente od kojih se proizvod sastoji, koriste vanjske klase koje nisu dio proizvoda?	*D3. Platform Afferent Coupling*	Broj jedinstvenih referenci prema klasama okvira referentne arhitekture od strane klasa proizvoda i/ili komponenata od kojih se proizvod sastoji.
	D4. Product Efferent Coupling	Broj jedinstvenih referenci prema klasama programskog okružja (npr. *Java RTE*) od strane klasa proizvoda i/ili komponenata od kojih se proizvod sastoji.
	D5. Product Efferent Coupling	Broj jedinstvenih referenci prema klasama vanjskih komponenata drugih proizvođača (npr. *Spring*, *Google*, *Apache*, *Jasper*) od strane klasa proizvoda i/ili komponenata od kojih se proizvod sastoji.
Koliko proizvod i/ili komponente od kojih se proizvod sastoji koriste okvir referentne arhitekture u odnosu na ostale klase programskog okružja i vanjske komponente drugih proizvođača.	*PR. Platform Responsibility* Omjer broja referenci prema ostalim klasama u odnosu na klase okvira referentne arhitekture.	Broj jedinstvenih referenci prema klasama vanjskih komponenata i programskog okružja od strane proizvoda i/ili komponenata od kojih se proizvod sastoji, u odnosu prema broju referenci prema klasama okvira referentne arhitekture od strane klasa proizvoda i/ili komponenata od kojih se proizvod sastoji.

Dvije metrike (D1, D2) koje ne koristimo direktno u modelu (PR) za procjenu promjenjivosti, predstavljaju odgovore na prethodno navedeno dodatno pitanje (Rosko, 2013a). Glavna ideja kod definiranja predloženih metrika bila je njihova jednostavnost i sveobuhvatnost u smislu procjene promjenjivosti objekata mjerenja (poslovnih komponenata), te njihovog korištenja u procesu evaluacije okvira referentne arhitekture. Broj referenci prema drugim klasama odnosi se na „jedinstvene“ veze, što znači da maksimalni broj referenci od jedne klase prema drugoj ne može biti veći od jedan. Primjerice, ukoliko neka klasa referencira drugu klasu nekoliko puta, metrike koje predlažemo u obzir uzimaju samo prvu referencu. Metrike predloženog modela su prethodno (Rosko, 2013a) analizirane pomoću okvira za teoriju mjerenja (Poels & Dedene, 1999) i okvira za procjenu poželjnih karakteristika predloženih metrika (Briand, Morasca, & Basili, 1996). Predložene metrike zadovoljavaju sve obavezne postavljene kriterije, što pretpostavlja njihovu moguću upotrebu u praksi.

Slika 89 ilustrira pretpostavke predloženog modela za predviđanje promjenjivosti poslovnih komponenata; komponenta je bolje održavljiva ukoliko je broj jedinstvenih interakcija sa vanjskim sustavima drugih dobavljača manji a veći broj interakcija sa komponentama vlastitog okvira referentne arhitekture. Organizacija sustava na manje elemente (module i komponente) smatra se kvalitetnijom ukoliko većina elemenata ovisi o elementima koji su stabilniji i nalaze se na nižoj razini (Sarkar, Rama, & Kak, 2007).

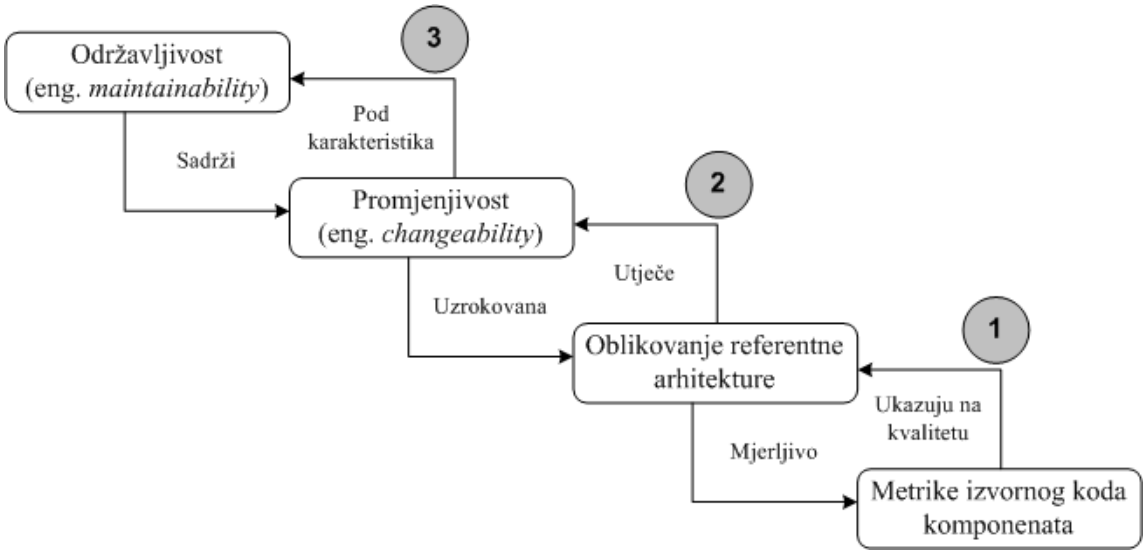


Slika 89 Pretpostavke promjenjivosti poslovnih komponenata

Slika 90, prema (Heitlager, Kuipers, & Visser, 2007) prilagođena modelu kojeg istražujemo, prikazuje hijerarhijsku strukturu PR modela, mapiranja između razina modela označena su sa brojevnim oznakama: 1, 2 i 3.

Oznaka 1 odnosi se na mapiranja metrika izvornog programskog koda i kvalitete oblikovanja okvira referentne arhitekture, kao što prikazuje oznaka **1** na slici 90, mjerenjem novo predloženih metrika (D3, D4, D5) koje su u linearnom odnosu prema kvaliteti oblikovanja. Na održavljivost poslovnih komponenata utječu i ostale metrike izvorno programskog koda (npr. *Cyclomatic Complexity*) koje nisu u linearnom odnosu prema kvaliteti oblikovanja referentne arhitekture, njihovu težinsku vrijednost kojom utječu na održavljivost odrediti ćemo kasnije, stavljanjem u odnos više kvantitativnih nezavisnih varijabli, metrika izvornog programskog koda i PR.

Oznaka 2 odnosi se na mapiranja kvalitete oblikovanja okvira referentne arhitekture i promjenjivosti softverskih komponenta, kao što prikazuje oznaka **2** na slici 90. Računa se kao ukupna relativna vrijednost PR metrike svih komponenata koje koriste okvir referentne arhitekture, što možemo promatrati kao razinu „odgovornosti“ okvira referentne arhitekture za preuzimanje kompleksnosti interakcija sa vanjskim sustavima na sebe, u cilju poboljšanja održavljivosti komponenata.



Slika 90 PR model održavljivosti (promjenjivosti) poslovnih komponenata

Oznaka 3 odnosi se na mapiranja (jedan prema jedan) promjenjivosti i održavljivosti. Promjenjivost komponenata je samo jedna od pod karakteristika održavljivosti za koju

predlažemo PR kao prikladan model mjerenja u kontekstu primjene pristupa linija za proizvodnju softvera za poslovne aplikacije.

6.5.2. Faze primjene modela (PR)

Tablica 43 prikazuje faze primjene metode (PR) prema prethodno opisanom modelu.

Tablica 43 Faze primjene metode PR

1.	Definiranje opsega mjerenja
2.	Raščlana sustava na poslovne komponente
3.	Prikupljanje izvornog koda poslovnih komponenata i okvira referentne arhitekture
4.	Izrada popisa poslovnih komponenata
5.	Brojanje poveznica (D3, D4, D5)
6.	Dopunjavanje popisa poslovnih komponenata s podacima brojanja poveznica
7.	Izračun metrike (PR)
8.	Obrada podataka

Prvi korak u primjeni metode (PR) je definiranje opsega koji obuhvaća poslovne aplikacije i komponente za koje provodimo mjerenje. Poslovne komponente mogu biti klijentske (koriste se u prezentacijskom sloju poslovnih aplikacija) ili poslužiteljske koje se koriste za poslovnu logiku i pristup podacima na poslužitelju. Nakon definiranja opsega, sustav je potrebno raščlaniti na poslovne komponente za koje će se raditi procjena promjenjivosti pomoću ove metode. Za svaku poslovnu komponentu prikuplja se izvorni programski kod. Alat za mjerenje metrika najčešće zahtjeva pristup do okvira referentne arhitekture i do vanjskih komponenata, stoga ih je potrebno ugraditi u okolinu alata za mjerenje prije početka mjerenja. Izrađuje se popis poslovnih komponenata u tablici u kojoj su predviđene kolone za unos rezultata mjerenja navedenih metrika. Broje se reference (metrike) poslovnih komponenata prema drugim komponentama (D3, D4, D5) i zapisuju se u za to predviđene kolone u tablici. Nakon zapisa vrijednosti svake od izmjerenih metrika, računa se metrika PR prema formuli (3).

$$\mathrm{PR} = \left(1 - \frac{D4 + D5}{D3 + D4 + D5}\right) * 100 \tag{3}$$

Na kraju se radi obrada dobivenih podataka, tj. procjena promjenjivosti poslovnih komponenata pojedinačno. Sumirajući vrijednosti pojedinačnih komponenata koje koriste okvir referentne

arhitekture, može se utvrditi promjenjivost sustava u cjelini (definirano opsegom ovog mjerenja) prema formuli (4), pri čemu je ***n*** broj poslovnih komponenata. Vrijednost PR metrike za cijeli sustav možemo promatrati kao razinu „odgovornosti“ okvira referentne arhitekture za preuzimanje kompleksnosti interakcija poslovnih komponenta prema vanjskim komponentama.

$$\mathrm{PR} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \left(1 - \frac{\mathrm{D4_i} + \mathrm{D5_i}}{\mathrm{D3_i} + \mathrm{D4_i} + \mathrm{D5_i}}\right) * 100 \tag{4}$$

Pretpostavljamo da vrijednost ***PR > 50*** cijelog sustava indicira da je okvir referentne arhitekture i način na koji se koristi u promatranom slučaju „preuzeo na sebe“ dovoljno kompleksnosti koje proizlaze iz potencijalnih interakcija poslovnih komponenata sa vanjskim komponentama, te da se okvir referentne arhitekture može u budućnosti koristiti za razvoj poslovnih aplikacija na takav isti ili sličan način. Ukoliko je vrijednost ***PR <= 50***, pretpostavljamo da okvir referentne arhitekture nije „preuzeo na sebe“ dovoljno kompleksnosti, te da se ne preporučuje za razvoj poslovnih aplikacija na takav način u budućnosti.

6.5.3. Mjerenje promjenjivosti pomoću modela (PR)

Za mjerenje promjenjivosti poslužiteljskih komponenata poslovnih aplikacija koje primjenjuju okvir referentne arhitekture koristimo kontrolirani tehnološki orijentirani eksperiment. Klijentske poslovne komponente, koje se koriste za izgradnju prezentacijskog sloja poslovnih aplikacija, nisu predmet ovog eksperimenta. Poslovne komponente se putem konfiguracije pridružuju poslovnim aplikacijama, tako da se ista komponenta može koristiti u nekoliko poslovnih aplikacija, prema pristupu linija za proizvodnju softvera. Mjerenje karakteristika poslovnih komponenata proveli smo četiri puta, za svaku inačicu (V1-V4) posebno.

Tablica 44 prikazuje podatke mjerenja metrika poslovnih komponenata za dvije zadnje inačice (V3, V4) referentne arhitekture i poslužiteljskih komponenata. Neke poslovne komponente su razvijene i koriste se u obje inačice sustava, dok se neke (K49 do K57) novo razvijene komponente odnose samo na zadnju inačicu (V4) sustava. Inačica sustava (V3) bila je predmet prepravljanja (engl. *refactoring*) kako bi se dobile bolje karakteristike okvira referentne arhitekture i poslovnih komponenata, uvele nove tehnologije, poboljšala održavljivost poslovnih komponenata. Pregledom vrijednosti metrika u tablici može se vidjeti poboljšanje vrijednosti PR (veće vrijednosti ukazuju na lakšu održavljivost). Za usporedbu, prikazane su i vrijednosti metrike MI za koju držimo da predstavlja alternativnu mjeru. PR predlažemo kao alternativu za MI koja pobliže procjenjuje promjenjivost (engl. *changeability*) poslovnih

komponenata u kontekstu razvoja i održavanja poslovnih aplikacija prema pristupu linija za proizvodnju softvera.

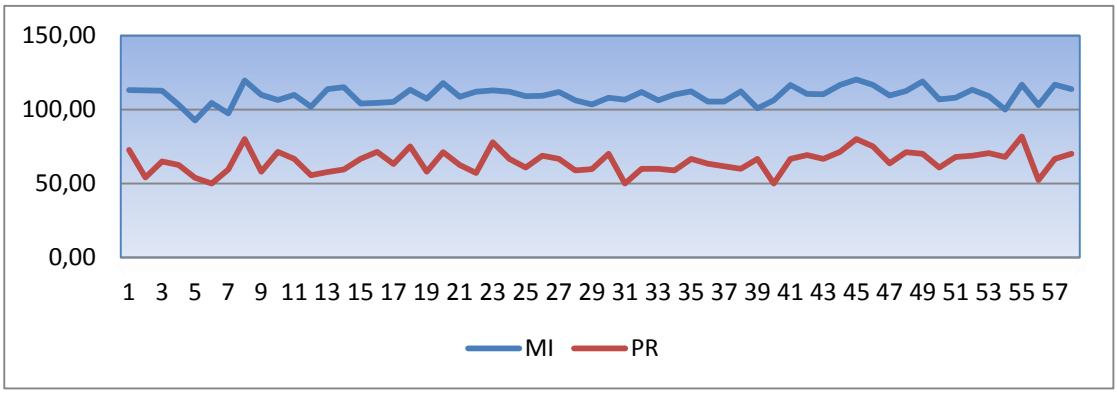
Tablica 44 Rezultati mjerenja poslužiteljskih poslovnih komponenata (V3 i V4)

Kom.	**Metrike odgovornosti platforme** (engl. *platform responsibility*)								**MI**	
	D3		**D4**		**D5**		**PR**			
	V3	V4	V3	V4	V3	V4	V3	V4	V3	V4
K 1	13	8	6	3	0	0	68,42	72,73	106,66	113,18
K 2	104	53	79	45	0	0	56,83	54,08	104,22	112,93
K 3	8	24	5	13	0	0	61,54	64,86	104,90	112,80
K 4	25	15	17	9	0	0	59,52	62,50	95,22	103,15
K 5	42	28	38	24	0	0	52,50	53,85	88,42	92,50
K 6	30	15	34	15	0	0	46,88	50,00	96,25	104,44
K 7	26	19	16	13	0	0	61,90	59,38	94,63	97,36
K 8	10	8	4	2	0	0	71,43	80,00	112,42	119,72
K 9	17	11	15	8	0	0	53,13	57,89	101,15	109,85
K 10	16	10	8	4	0	0	66,67	71,43	97,52	106,51
K 11	6	4	5	2	0	0	54,55	66,67	100,87	109,86
K 12	60	35	47	28	0	0	56,07	55,56	96,77	101,86
K 13	81	48	60	35	0	0	57,45	57,83	107,64	113,79
K 14	29	22	19	13	0	2	60,42	59,46	109,40	115,08
K 15	8	6	5	3	0	0	61,54	66,67	99,00	104,02
K 16	30	20	18	8	0	0	62,50	71,43	97,50	104,52
K 17	144	101	93	59	0	0	60,76	63,13	98,58	105,12
K 18	19	6	12	2	0	0	61,29	75,00	95,69	113,48
K 19	18	11	14	8	0	0	56,25	57,89	100,13	107,23
K 20	38	27	29	11	0	0	56,72	71,05	113,76	117,97
K 21	16	10	13	6	0	0	55,17	62,50	98,37	108,67
K 22	8	4	5	3	0	0	61,54	57,14	101,97	112,03
K 23	11	7	4	2	0	0	73,33	77,78	104,54	112,91
K 24	10	6	5	3	0	0	66,67	66,67	103,80	112,05
K 25	26	17	19	11	0	0	57,78	60,71	100,28	109,13
K 26	31	22	21	10	0	0	59,62	68,75	103,43	109,33
K 27	31	22	24	11	0	0	56,36	66,67	103,61	111,82
K 28	15	10	13	7	0	0	53,57	58,82	98,06	106,21
K 29	78	46	58	31	0	0	57,35	59,74	94,89	103,38
K 30	20	14	10	6	0	0	66,67	70,00	97,50	107,93
K 31	13	7	13	7	0	0	50,00	50,00	96,84	106,68
K 32	29	15	20	10	0	0	59,18	60,00	105,10	111,88
K 33	33	21	26	14	0	0	55,93	60,00	98,37	106,22
K 34	17	10	14	7	0	0	54,84	58,82	102,36	110,04
K 35	9	6	6	3	0	0	60,00	66,67	102,13	112,26
K 36	142	95	115	55	0	0	55,25	63,33	97,76	105,33
K 37	10	8	9	5	0	0	52,63	61,54	98,56	105,36
K 38	15	9	11	6	0	0	57,69	60,00	103,87	112,31
K 39	6	4	4	2	0	0	60,00	66,67	92,73	100,88
K 40	27	15	28	15	0	0	49,09	50,00	97,84	106,21
K 41	6	4	4	2	0	0	60,00	66,67	106,80	116,57
K 42	56	70	26	31	0	0	68,29	69,31	101,56	110,53
K 43	12	8	8	4	0	0	60,00	66,67	100,51	110,31
K 44	8	5	4	2	0	0	66,67	71,43	109,09	116,66
K 45	11	6	5	2	0	0	68,75	75,00	108,78	116,68

K 46	32	21	26	12	0	0	55,17	63,64	104,53	109,58
K 47	51	37	26	15	0	0	66,23	71,15	107,14	112,56
K 48	15	21	11	9	0	0	57,69	70,00	114,85	119,07
K 49	-	31	-	20	-	0	-	60,78	-	106,90
K 50	-	19	-	9	-	0	-	67,86	-	108,04
K 51	-	11	-	5	-	0	-	68,75	-	113,46
K 52	-	24	-	10	-	0	-	70,59	-	109,09
K 53	-	17	-	8	-	0	-	68,00	-	99,92
K 54	-	9	-	2	-	0	-	81,82	-	116,85
K 55	-	22	-	12	-	8	-	52,38	-	103,09
K 56	-	10	-	4	-	1	-	66,67	-	116,76
K 57	-	7	-	3	-	0	-	70,00	-	113,89
∑	1462	**1141**	1052	**669**	0	**11**	2851,84	**3677,91**	4876,00	**6235,96**

Prema podacima iz tablice 44 moguće je izračunati ukupnu održavljivost (promjenjivost) za svaku pojedinu inačicu. Pa tako korištenjem zbirnih vrijednosti možemo izračunati da je **PR** (V3) = (1 - (1052+0) / (1462+1052+0)) * 100 = **58,15** što ukazuje na zadovoljavajuću (> 50) održavljivost (promjenjivost) poslovnih komponenata koje primjenjuju okvir referentne arhitekture inačice (V3). Nakon prepravljanja okvira referentne arhitekture i postojećih poslužiteljskih komponenata (V3), te razvoja njegove nove inačice (V4) i novih poslužiteljskih komponenata (od K49 do K57), poboljšala se i njihova održavljivost; izračunata vrijednost **PR** (V4) = (1 - (669+11) / (1141+669+11)) * 100 = **62,66** ukazuje na značajno poboljšanje održavljivosti inačice (V4) u odnosu na prethodnu inačicu (V3). Ako u obzir uzmemo samo komponente koje su postojale u inačici (V3), 48 komponenata, vrijednost **PR** (V4a) = (1 - (596+2) / (991+596+2)) * 100 = **62,37**; vrijednost tog manjeg broja komponenata inačice (V4) također pokazuje značajno poboljšanje u odnosu na vrijednost u inačici (V3), što se direktno može povezati sa utjecajem okvira referentne arhitekture koji je prepravljen kako bi se postigla lakša održavljivost komponenata koje ga koriste.

Vrijednost alternativne metrike **MI** (V3) = 4876,00 / 48 = **101,58** ukazuje na izvrsnu održavljivost (> 85) komponenata inačice (V3). Vrijednost MI poslovnih komponenata za novu inačicu (V4) daje još bolji rezultat ukupne održavljivosti **MI** (V4) = **6235,96 / 57 = 109,40**. Slično kako i kod vrijednosti PR, vrijednost MI samo za komponente koje su postojale u inačici (V3), 48 komponenata, blago je niža u odnosu na ukupnu održavljivost svih komponenata (V4), **MI** (V4a) = **5247,96 / 48 = 109,33**. Slika 91 ukazuje na približan linearni odnos vrijednosti PR i MI metrika poslužiteljskih komponenta za inačicu (V4), prema podacima iz tablice 50. Budući da model PR mjeri promjenjivost (pod karakteristiku održavljivosti), koriste se potpuno druge metrike, nije bilo za očekivati potpunu korelaciju sa standardnim alternativnim modelom MI za procjenu održavljivosti.



Slika 91 Odnos PR i MI vrijednosti za inačicu (V4)

6.5.4. Statistička obrada dobivenih rezultata

Rezultate koje smo dobili mjerenjem i djelomično prikazane u tablici 44 (V3, V4) i u dodatku A (V2, V3, V4) statistički smo obradili pomoću programa IBM SPSS ® *for Windows* v.21. Podatke za inačicu V1 nismo uzimali u obzir prilikom statističke obrade iz jednostavnog razloga što u to vrijeme poslovne komponente nisu u potpunosti koristile okvir referentne arhitekture prema pristupu linija za proizvodnju softvera, kao što je slučaj kasnije sa inačicama V2-V4. Iako su podaci mjerenja za inačicu V1 vrlo slični podacima za inačice V2-V4, kod testiranja hipoteze **H2** naglasak smo postavili na metrike „odgovornosti platforme“ što uvjetuje kontekst u kojem se okvir referentne arhitekture koristi i način na koji ga poslovne komponente koriste.

Cilj statističke analize je bio utvrditi osnovne deskriptivne karakteristike podataka dobivenih mjerenjem izvornog programskog koda s naglaskom na normalnost distribucije i utvrđivanje preduvjeta za korelacijsku analizu; na svim poslužiteljskim komponentama (kumulativno) provjeriti stupanj povezanosti podataka dobivenih MI i PR metodom i utvrditi njihov odnos; definirati novi model za predviđanje promjenjivosti poslovnih komponenata i aplikacija (PR).

Tablica 45 prikazuje deskriptivnu statistiku za tri inačice (V2, V3, V4) poslužiteljskih komponenata i njihovih metrika s posebnim naglaskom na PR metrike (D3, D4, D5). Izbor ostalih metrika se temelji na činjenici da one mjere različite karakteristike komponenata: veličinu, povezanost, kompleksnost i nasljeđivanje.

Tablica 45 Deskriptivna statistika metrika izvornog programskog koda

Metrika	N	Min.	Maks.	Aritm. sredina	Stand. dev.	Koef. asim. (*skewness*)	Stand. pogr. asim.	Koef. zaob. (*kurtosis*)	Stand. pogr. zaob.
PR	150	46,88	81,82	61,3502	6,86702	,379	,198	-,133	,394
MI	150	88,42	119,72	104,5575	6,59755	,118	,198	-,565	,394
D3	150	4	144	25,91	26,791	2,508	,198	6,945	,394
D4	150	2	116	17,73	20,083	2,637	,198	8,301	,394
D5	150	0	8	,07	,677	11,115	,198	129,113	,394
ABD	150	1,08	1,77	1,4105	,13413	-,107	,198	-,161	,394
ACC	150	1,17	2,49	1,6094	,28303	,905	,198	,680	,394
ADIT	150	3,00	6,41	4,4263	,77759	,402	,198	-,326	,394
ALCM	150	5,50	20,44	11,1294	3,29943	,384	,198	-,346	,394
NMETH	150	6	255	45,35	48,895	2,468	,198	7,020	,394
AWMC	150	3,40	44,20	12,3780	7,54138	1,981	,198	4,923	,394
AMC	150	2,40	23,00	7,5698	4,13309	1,718	,198	3,397	,394
LOC	150	53	4049	601,67	682,022	2,876	,198	10,188	,394
NCLASS	150	2	26	6,12	5,731	1,914	,198	3,120	,394
NOS	150	31	3335	461,22	537,911	3,007	,198	11,562	,394
WM	150	8	488	74,81	84,612	2,844	,198	10,008	,394

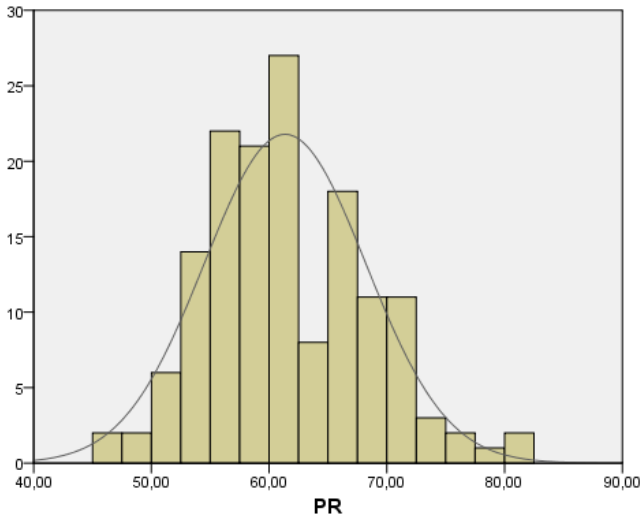
Osim toga, izabrane metrike su dio klasičnih metrika za objektno orijentirane sustave (Chidamber & Kemerer, 1994a), znanstveno su potvrđene i često korištene u kontekstu predviđanja promjenjivosti sustava (Ayalew & Mguni, 2013; Burrows, Garcia, & Taïani, 2010; Rosko, 2013a). Za subjektivnu provjeru da li distribucije podataka metrika PR i MI odgovaraju normalnoj distribuciji koristimo histograme (slika 92 i slika 93) iz kojih vidimo da su vrijednosti približno jednako raspoređene oko sredine, uz blagu nagnutost na desno.

Osim subjektivne provjere pripadnosti distribucije podataka metrika PR i MI normalnoj distribuciji, analiziramo i koeficijent asimetrije[17] (engl. *skewness*) i zaobljenosti[18] (engl. *kurtosis*), tablica 45. Koeficijenti asimetrije ukazuju na to da podaci ne odstupaju statistički značajno (PR=0,379, MI=0,118) od normalne distribucije; ako je vrijednost veća od jedan, onda se ona može tumačiti kao asimetrična. Pozitivne vrijednosti koeficijenta asimetrije ukazuju da je distribucija obje metrike blago nagnuta u desno. Koeficijenti zaobljenosti koji opisuju

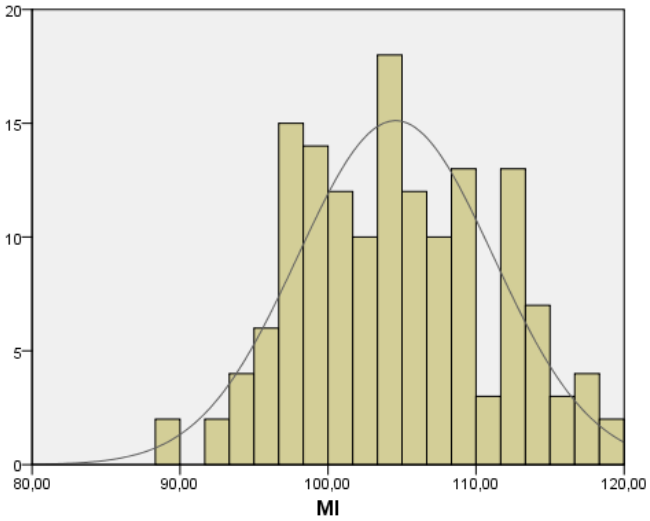
[17] Koeficijent asimetrije (engl. *skewness*, što znači nagnutost) ima negativnu vrijednost ako je zvonolika distribucija podataka nagnuta ulijevo; jednaka je nuli ako je simetrična; pozitivna je ako je distribucija nagnuta udesno (Dumičić i ostali, 2011).

[18] Ako pokazatelj zaobljenosti vrha distribucije (engl. *kurtosis*) ima negativni predznak, znači da je vrh distribucije plosnatiji od vrha normalne distribucije; ako je nula tada je zaobljenost „normalna“; pozitivna vrijednost pokazuje da je vrh distribucije zašiljeniji nego vrh normalne distribucije (Petz i ostali, 2012).

zaobljenost vrha distribucija frekvencija imaju negativni predznak (PR = -,133, MI = -,565) i ukazuju na blago plosnatiji vrh od vrha normalne distribucije.



Slika 92 Histogram metrike PR



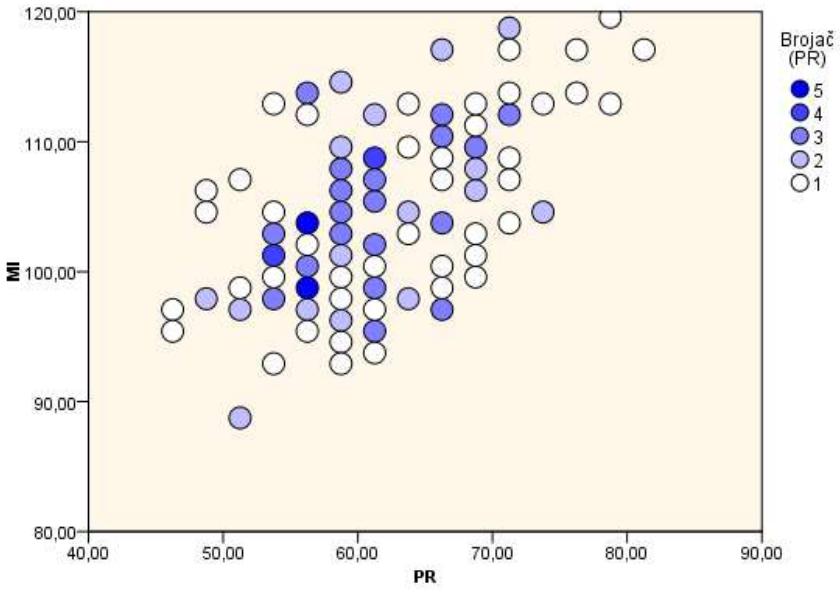
Slika 93 Histogram metrike MI

Raspon varijacije PR vrijednosti (81,82 – 46,88 = **34,94**) koji predstavlja razliku između najveće i najmanje vrijednosti podataka je apsolutno veći od raspona varijacije MI vrijednosti (119,72 – 88,42 = **31,30**), iako je bilo za očekivati obrnuto. Radi se o različitim jedinicama mjere, vrijednosti PR kreću se oko značajno manje aritmetičke sredine (oko 61) od vrijednosti

MI koje se kreću oko značajno veće aritmetičke sredine (oko 104). Međutim, u našem slučaju veći raspon varijacije PR vrijednosti možemo tumačiti kao nešto precizniju mjeru promjenjivosti od MI mjere ukoliko se potvrdi značajna povezanost između njih.

Koeficijent varijacije[19] vrijednosti PR ($\boldsymbol{V = \frac{61{,}3502}{6{,}86702}}$) jednak je **11,19**% dok je koeficijent varijacije vrijednosti MI ($\boldsymbol{V = \frac{104{,}5575}{6{,}59755}}$) jednak je **6,31**%. Koeficijenti varijacije poprimaju vrijednost manju od 25%, te možemo govoriti o dobroj reprezentativnosti prosjeka.

Na slici 94 prikazan je dijagram rasipanja (engl. *scatter diagram*) koji ukazuje da postoji pozitivna linearna statistička veza između MI i PR za poslužiteljske poslovne komponente koje koriste okvir referentne arhitekture; odnosno prema rasporedu točaka dijagrama rasipanja.



Slika 94 Odnos PR i MI metrika

Ako neka poslužiteljska poslovna komponenta ima veći MI, može se očekivati i njena veća PR vrijednost. Porast vrijednosti MI prati porast vrijednosti PR i obrnuto. Subjektivnom analizom izgleda rasporeda vrijednosti uočavamo da je uvjet homoscedasciteta[20] zadovoljen. Iz

[19] Koeficijent varijacije je postotak standardne devijacije od aritmetičke sredine. Poželjno je da je njegova vrijednost blizu nule, no u praksi se toleriraju i koeficijenti do 25% kao zadovoljavajući. (Dumičić i ostali, 2011, str. 143)

[20] Homoscedascitet je prisutan ako je varijabilitet rezultata Y oko pravca regresije podjedank duž cijeloga pravca (Petz i ostali, 2012)

navedenih podataka deskriptivne statistike proizlazi da se na opisanim podacima mogu provjeriti parametrijski, regresijski modeli te donositi inferencijalni sudovi[21].

Postojeća empirijska istraživanja održavljivosti softvera ukazuju na potencijalne višestruke koristi od praćenja i kontrole kvalitete softvera, posebno ako se taj proces započne već u ranoj fazi njegovog razvoja (Misra, 2005). Značajan utjecaj metrika izvornog programskog koda na održavljivost softvera empirijski je potvrđen (Misra, 2005; Zhou & Xu, 2008). U navedenim postojećim istraživanjima, zavisna varijabla je MI (engl. *maintainability index*), dok su nezavisne varijable metrike izvornog programskog koda.

Cilj ovog istraživanja je provjeriti povezanost rezultata PR (engl. *platform responsibility*) i MI modela za mjerenje održavljivosti softverskih komponenata u kontekstu linije za proizvodnju softvera poslovnih aplikacija, te pronalaženje adekvatnog modela (linearne kombinacije) koji se sastoji od skupa nezavisnih varijabli (metrika izvornog programskog koda) koje procjenjuju nepoznatu regresijsku funkciju zavisne varijable (PR). Na izbor metrika izvornog programskog koda za ovo istraživanje utjecala su postojeća istraživanja i njihova popularnost (Abreu & Carapuça, 1994; Belsley i ostali, 2005; Chidamber & Kemerer, 1994a), mogućnosti alata za automatsko prikupljanje vrijednosti metrika, prethodna empirijska istraživanja (Roško, 2014b).

Korelacija metrika PR i MI

Tablica 46 prikazuje rezultate izračunavanja stupnja povezanosti (korelacije) metrika izvornog programskog koda triju inačica poslužiteljskih poslovnih komponenata (longitudinalno).

Tablica 46 Rezultati testa korelacije metrika izvornog koda, PR i MI

		PR	MI	ABD	ACC	ADIT	AMC	ALCM	AWMC	NMETH
PR	*Pearson*	1	**,544****	-,281**	-,410**	-,216**	-,294**	-,517**	-,372**	-,240**
	Značajnost		,000	,000	,000	,008	,000	,000	,000	,003
	N	150	150	150	150	150	150	150	150	150
MI	*Pearson*	,544**	1	-,671**	-,722**	,135	-,056	-,948**	-,236**	-,066
	Značajnost	,000		,000	,000	,100	,496	,000	,004	,425
	N	150	150	150	150	150	150	150	150	150

**. Korelacija je statistički značajna na razini od 0.01 (dvosmjerni test).
*. Korelacija je statistički značajna na razini od 0.05 (dvosmjerni test).

Budući je naš cilj pronalaženje adekvatnog modela za procjenu održavljivosti u kojem je PR zavisna varijabla, prethodno je potrebno utvrditi da postoji značajna povezanost MI metrike

[21] Inferencijalna (analitička, matematička, induktivna) statistika odnosi se na postupke kojima se pomoću dijela informacija (uzoraka) donose sudovi o karakteristikama cjeline (populacije) na temelju teorije vjerojatnosti, a u svrhu definiranja odnosa između varijabli (Verčić, Sinčić Ćorić, & Pološki Vokić, 2010).

koja se koristi kao zavisna varijabla u postojećim modelima koji se najčešće koriste u softverskoj industriji i PR metrike koju predlažemo kao zavisnu varijablu novog modela. Koeficijent korelacije MI i PR metrika na razini značajnosti od 99% ($p \leq 0{,}01$) i 95% ($p \leq 0.05$) je 0,544**. Nije bilo za očekivati sigurno slaganje između ove dvije varijable budući da su izvedene iz potpuno drugačijih metrika. Njihovu povezanost možemo protumačiti kao „stvarnu značajnu povezanost – od 0,40 do 0,70“ prema (Petz i ostali, 2012).

Testiranje značajnosti koeficijenta korelacije r

Koeficijent korelacije (0,544) znači stvarnu značajnu povezanost. Da bi smo utvrdili da je dobiveni koeficijent korelacije *r* značajan (tj. razlikuje li se od nule bez obzira na predznak), izračunali smo *t*, i to prema formuli (5):

$$t = r\,\frac{\sqrt{(N-2)}}{\sqrt{1-r^2}} \tag{5}$$

Testiranjem značajnosti veze između PR i MI metrike poslužiteljskih komponenata poslovnih aplikacija koji su dio linije za proizvodnju softvera koju istražujemo dobili smo:

$$t = 0{,}544\,\frac{\sqrt{148}}{\sqrt{1-0{,}296}} = 7{,}887 \tag{6}$$

Na tablici *t*-vrijednosti očitana granična vrijednost *t* na razini značajnosti od 5% uz 148 (125 do 150) stupnjeva slobode iznosi 1,98. Budući je dobiveni *t* znatno veći, zaključujemo da je korelacija između PR i MI statistički značajna.

Također, očitavanjem s „dvosmjerne“ tablice graničnih vrijednosti koeficijenta korelacije (razina značajnosti za dvosmjerno testiranje), koeficijent korelacije mora iznositi najmanje 0,159 na razini značajnosti od 5%, a na razini značajnosti od 1% najmanje 0,208. Prema tome naš *r* je čak značajan na razini koja je manja od 1% ($P<0{,}01$).

Postojanje korelacije između MI i PR metrika ukazuje nam da je moguće koristiti PR za istu svrhu kao i MI (za predviđanje održavljivosti), te nam daje povod za razmatranje utjecaja, trećeg faktora (ostalih metrika - nezavisnih varijabli) na PR kako bi pronašli model za jednostavnije predviđanje održavljivosti komponenta softverskih sustava. Rezultati dobiveni primjenom analize koji su prikazani u tablici 46, u kojoj oznake ** i * ukazuju na statistički značajnu povezanost, ukazuju da svih 7 metrika koje se koriste u ovom istraživanju negativno i značajno utječu na PR (zavisnu varijablu). Porastom vrijednosti ovih metrika, smanjuje se

održavljivost komponenta, posljedično i cijelog sustava ako se temelji na primjeni pristupa linija za proizvodnju softvera poslovnih aplikacija.

Rezultati usporedbe MI standardnog modela za procjenu održavljvosti i predloženog modela PR pokazali su ispravnost druge hipoteze ovoga rada (**H2**) da se **rezultati određivanja održavljivosti poslovnih komponenata (engl. *maintainability*) informacijskog sustava metrikom „odgovornosti platforme“ (engl. *platform responsibility*) podudaraju s rezultatima dobivenim uz pomoć alternativne metrike**. Rezultati mjerenja modela PR odnose se na pod karakteristiku održavljivosti „promjenjivost“, stoga nije bilo za očekivati da postoji potpuna korelacija između dva modela. MI model se najčešće primjenjuje u kontekstu koji se odnosi na korištenje proceduralnih programskih jezika i pojedinačnih aplikacija, za razliku od konteksta ovog istraživanja koje se odnosi na liniju za proizvodnju softvera i objektno orijentirani programski jezik.

6.5.5. Analiza linearne povezanosti između PR i MI

Kada znamo da postoji statistički značajna korelacija između MI i PR metrika izvornog programskog koda, želimo utvrditi statistički model koji bi nam omogućio da temeljem poznatih PR vrijednosti predvidimo MI vrijednosti, sa što manjom pogreškom prognoze. U analizi koristimo univarijantnu regresiju koja se u statistici definira kao postupak modeliranja veze između jedne ili više varijabli označenih s Y i jednom ili više varijabli označenih s X, a rezultat takvog modela je da se Y za dani X izražava kao linearna funkcija od X. Slučajna komponenta *e* upućuje da veze između pojava u praksi nisu funkcionalne, nego su statističke ili stohastičke, te da postoje pozitivna i/ili negativna odstupanja originalnih vrijednosti. Model se računa uz pomoć regresije najmanjih kvadrata, a opisuje se jednadžbom (7):

$$y_i = b_0 + b_1 x_i + e_i \qquad i = 1,2,\dots,n. \qquad (7)$$

U izrazu (7) b_n su konstantni koeficijenti modela, a e_i iznos slučajne pogreške u predviđanju. Za svaku opaženu vrijednost, x_i opažena vrijednost y_i generirana je populacijskim modelom.

Tablica 47 prikazuje rezultate regresijske analize najmanjih kvadrata međusobne povezanosti PR i MI modela koji nam omogućava da pomoću poznatih PR vrijednosti (prediktor) predvidimo MI vrijednosti (kriterij) sa što manjom pogreškom prognoze.

Tablica 47 Rezultati izračuna linearne regresije za PR / MI

Model (sažetak)

Model	R	R^2	Korigirani R^2	Stand. pogr. proc.
1	,544[a]	,296	,291	5,55406

a. Prediktori: (Konstanta), PR

ANOVA

Model		SS	df	MS	F	Sig.
1	Regresija	1920,188	1	1920,188	62,248	,000[b]
	Reziduala	4565,439	148	30,848		
	Ukupno	6485,627	149			

a. Zavisna varijabla: MI

b. Prediktori: (konstanta), PR

Koeficijenti

Model		Nestandardizirani koeficijenti		Standardizirani koeficijenti	t	Sig.	Interval pouzdanosti 95,0% za B	
		B	Stand. Pogr.	Beta			Donja granica	Gornja granica
1	(konstanta)	72,485	4,090		17,721	,000	64,403	80,568
	PR	,523	,066	,544	7,890	,000	,392	,654

a. Zavisna varijabla: MI

Model linearnog odnosa se može opisati jednadžbom:

$$MI = 72{,}485 + (0{,}523 * PR)$$

Analizom tablice (*koeficijenti*) vidljivo je da postoji statistička značajna, pozitivna linearna povezanost između rezultata dobivenih PR i MI metodama. Svaka jedinica povećanja PR varijable uzrokuje povećanje od 0,523 jedinica MI varijable. Primjerice, za PR vrijednost od 61,54 predviđena vrijednost MI bi bila:

$$MI = 72{,}485 + (0{,}523 * 61{,}54) = 106{,}67$$

Standardna greška procjene je 5,55406 što znači da se na razini značajnosti od 95%, predviđena vrijednost MI od 104,67 nalazi između rezultata od **2,66** (104,67- (1,96 * 5,55406)) i **3,95** (104,67- (1,96 * 5,55406)).

Evaluacija snage linearnog modela

Snagu tumačenja linearnog regresijskog modela (koliko učinkovito varijable X tumače ponašanje varijable Y) provjeravamo pomoću *koeficijenta determinacije* R^2 koji se interpretira kao postotak (varira od 0 do 1) varijabilnosti od Y (MI) koji je protumačen regresijskom jednadžbom. Veće vrijednosti R^2 ukazuju na bolju regresiju. Kod jednostavne regresije kao što

je naša, gdje imamo samo jedan prediktor, *R* je jednak koeficijentu korelacije. Standardizirani koeficijent povezanosti (Beta[22]) iznosi 0,544 što ukazuje da se temeljem rezultata dobivenih PR metodom može objasniti 29,6% varijance rezultata dobivenih MI metodom. Iskusni analitičari utvrdili su da modeli koji se zasnivaju na podacima dobivenim od pojedinačnih slučajeva često imaju R^2 u rasponu od 0.10 do 0.20 (Newbold & Carlson, 2007, str. 423). Naše istraživanje odnosi se na pojedinačnu studiju slučaja, pa relativno nisku vrijednost R^2 od 0,296 možemo promatrati u tome kontekstu.

ANOVA tablica odnosi se na test nulte hipoteze da je R^2 jednak nula. R^2 koji ima vrijednost nula ukazuje na nepostojanje linearne veze između prediktora (PR) i zavisne varijable (MI). ANOVA tablica pokazuje da je izračunata F[23] vrijednost jednaka 1920,188 / 30,848 = 62,248 na razini značajnosti koja je manja od 0,001. Iz tablice točaka reza F-distribucije vidljivo je da je vrijednost omjera F (62,248) znatno veća od kritične vrijednosti za $\alpha = 0{,}01$ s jednim stupnjem slobode u brojniku i 148 stupnjeva slobode u nazivniku (F = 6,63), što ukazuje da razlike među varijancama nisu slučajne te da se nulta hipoteza može odbaciti. Rezultati prikazani u tablici 47 za slučaj regresije procjene MI, kazuju da je *p*-vrijednost za taj izračunati F jednaka 0.000, što daje dodatne argumente u korist odbacivanja hipoteze da ne postoji linearna veza između PR i MI.

6.5.6. Analiza linearne povezanosti između PR i ostalih metrika

Na temelju rezultata testa korelacije metrika izvornog programskog koda (tablica 46) pretpostavljamo da metrike kojima raspolažemo utječu na održavljivost (PR), pa to koristimo za pronalaženje novog modela održavljivosti koji te metrike uzima u obzir kod definiranja regresijskog modela. Analizu radimo pomoću multivarijantne regresijske analize, tehnike koja se najčešće koristi za modeliranje odnosa zavisne i više nezavisnih varijabli. Cilj regresijske analize je utvrđivanje povezanosti između skupa metrika (nezavisnih varijabli) i PR metrike (zavisne varijable), odnosno utvrđivanja regresijskog modela koji služi u analitičke i prognostičke svrhe. Multivarijantnu regresijsku analizu koristimo za ispitivanje skupnog utjecaja (većeg broja nezavisnih varijabli) svih metrika na PR (zavisnu varijablu), razine značajnosti od 95% ($p \leq 0.05$).

[22] IBM SPSS statistički program standardizirani koeficijent povezanosti označava sa *Beta*, za razliku od uobičajene oznake za koeficijent korelacije *r*.
[23] F-omjer je vrijednost omjera varijance među skupinama i varijance unutar skupina. Ako je F-omjer veći od 1 tada potoji dovoljno argumenata da se nulta hipoteza odbaci.

Procjenjujući model multiple regresijske analize računa se uz pomoć regresije najmanjih kvadrata, opisuje se jednadžbom (8):

$$y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + \cdots + b_k x_{ki} + e_i \qquad i = 1,2,\ldots,n. \qquad (8)$$

Koeficijenti višestruke regresije računaju se korištenjem procjenitelja dobivenih procedurom najmanjih kvadrata. Ta procedura je slična proceduri univarijantne regresije koja se koristi u prethodnom poglavlju. Izračunati koeficijenti zakomplicirani su odnosima između nezavisnih varijabli (metrika) koji postoje istodobno s odnosima između nezavisnih i zavisne varijable. Primjerice, ako jedna nezavisna varijabla linearno raste ili pada s obzirom na drugu, što daje pozitivnu ili negativnu korelaciju između njih, dok u isto vrijeme postoji povećanje ili smanjenje zavisne varijable, mi ne možemo utvrditi koja nezavisna varijabla je stvarno povezana s promjenom zavisne varijable. Na temelju toga možemo zaključiti da će procijenjeni regresijski koeficijenti biti pouzdani tim manje što veća bude korelacija između dviju ili više nezavisnih varijabli.

Tablica 48 prikazuje rezultate testa korelacije svih metrika koje koristimo za izračun modela. Statistički značajna korelacija postoji između svih sedam metrika i PR metrike.

Tablica 48 Rezultati testa korelacije metrika izvornog koda

Korelacija									
		PR	**ABD**	**ACC**	**ADIT**	**ALCM**	**NMETH**	**AMC**	**AWMC**
PR	Pearson Correlation	1	-,281**	-,410**	-,216**	-,517**	-,240**	-,294**	-,372**
	Sig. (2-tailed)		,000	,000	,008	,000	,003	,000	,000
	N	150	150	150	150	150	150	150	150
ABD	Pearson Correlation	-,281**	1	,802**	-,237**	,730**	-,056	,126	,338**
	Sig. (2-tailed)	,000		,000	,004	,000	,498	,125	,000
	N	150	150	150	150	150	150	150	150
ACC	Pearson Correlation	-,410**	,802**	1	-,037	,796**	,070	,071	,334**
	Sig. (2-tailed)	,000	,000		,649	,000	,394	,387	,000
	N	150	150	150	150	150	150	150	150
ADIT	Pearson Correlation	-,216**	-,237**	-,037	1	-,180*	,482**	-,152	-,144
	Sig. (2-tailed)	,008	,004	,649		,028	,000	,063	,078
	N	150	150	150	150	150	150	150	150
ALCM	Pearson Correlation	-,517**	,730**	,796**	-,180*	1	,080	,201*	,392**
	Sig. (2-tailed)	,000	,000	,000	,028		,331	,013	,000
	N	150	150	150	150	150	150	150	150
NMETH	Pearson Correlation	-,240**	-,056	,070	,482**	,080	1	,364**	,358**
	Sig. (2-tailed)	,003	,498	,394	,000	,331		,000	,000
	N	150	150	150	150	150	150	150	150

AMC	Pearson Correlation	-,294**	,126	,071	-,152	,201*	,364**	1	,952**
	Sig. (2-tailed)	,000	,125	,387	,063	,013	,000		,000
	N	150	150	150	150	150	150	150	150
AWMC	Pearson Correlation	-,372**	,338**	,334**	-,144	,392**	,358**	,952**	1
	Sig. (2-tailed)	,000	,000	,000	,078	,000	,000	,000	
	N	150	150	150	150	150	150	150	150
**. Korelacija je statistički značajna na razini od 0.01 (dvosmjerni test). *. Korelacija je statistički značajna na razini od 0.05 (dvosmjerni test).									

Priprema podatka za izračun regresijskog modela

Za izračun modela koristimo postupni (engl. *stepwise*) regresijski model za koji vrijedi pravilo da poredak unosa nezavisnih varijabli ovisi jedino o statističkim kriterijima, za razliku od drugih regresijskih tehnika[24]. Varijable koje su visoko korelirane sa zavisnom varijablom unose se prve u regresijski model. Nedostatak ove metode u našem slučaju odnosi se na specifičnost uzorka koji se odnosi na samo jednu liniju za proizvodnju softvera, tako da se u nekom drugom testu redoslijed unosa nezavisnih varijabli može razlikovati, ovisno o specifičnostima tog drugog promatranog uzorka. Međutim, ovakav postupni regresijski model se preporuča za izviđajni (engl. *exploratory*) tip istraživanja kakav je naš, u kojem istraživač nije siguran u relativnu snagu procjene nezavisnih varijabli koje koristi u istraživanju (Ho, 2006). Kada smo izabrali model postupne linearne regresije, potrebno je odabrati metodu izračuna između tri raspoložive metode: *forward selection*[25], *backward deletion*[26], i *stepwise regression*. Odabrali smo metodu *stepwise regression* koja kod izbora varijabli koristi *forward selection* i *backward deletion* metode. Nezavisne varijable unose se jedna po jedna ako zadovoljavaju statističke kriterije, ali mogu biti izbrisane u bilo kojem koraku ako više ne zadovoljavaju statističke kriterije regresijskog modela.

Preduvjeti za multiplu linearnu regresiju

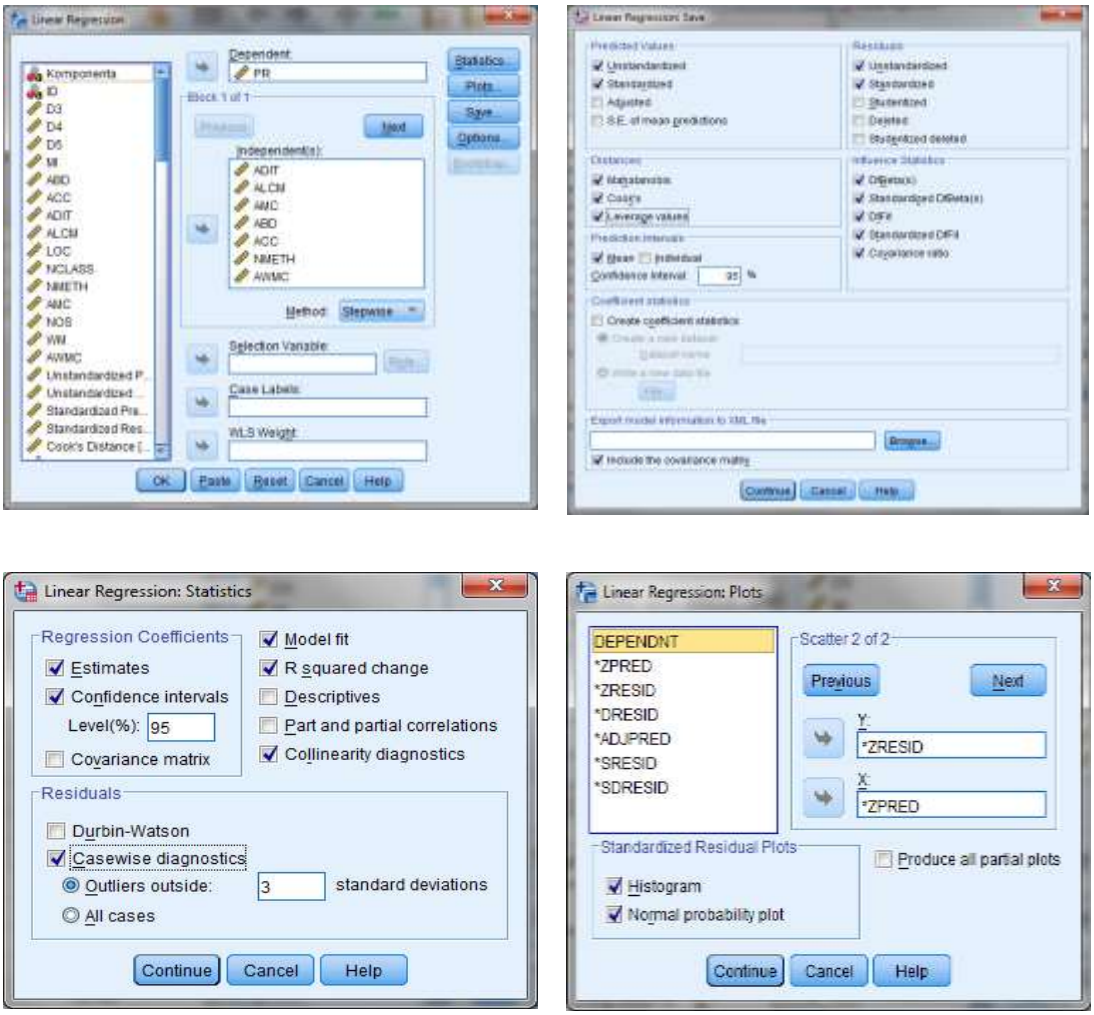
Uzorak koji koristimo sadrži N =150 elemenata. Pravilo da broj elemenata u uzorku mora biti najmanje 20 puta veći od broja nezavisnih varijabli (n=7) zadovoljeno je jer je 150 > (7*20).

[24] *Standardna multipla regresija* – sve nezavisne varijable unose se odjednom. Nedostatak ovog modela odnosi se na mogućnost da je neka nezavisna varijabla jako povezana sa zavisnom varijablom, pa da se svejedno smatra kao nevažan prediktor, ukoliko je njen koeficijent malen.
Hijerarhijska multipla regresija – predstavlja najfleksibilniji model, dozvoljava se da istraživač subjektivno određuje poredak unosa nezavisnih varijabli u regresijski model.
[25] *Foreword selection* – varijable se procjenjuju u odnosu na statističke kriterije, ako ih zadovoljavaju, daje im se prednost ulaska u model, ovisno o visini relativne korelacije sa zavisnom varijablom.
[26] *Backward deletion* – sve nezavisne varijable unose se odjednom, svaka varijabla se posebno procjenjuje, jedna po jedna, razmjerno njihovom doprinosu regresijskoj jednadžbi. Varijable koje ne doprinose značajno se brišu.

Izračun modela promjenjivosti PR

Programskom potporom IBM SPSS 21® procijenjen je model višestruke linearne regresije metodom najmanjih kvadrata.

Slika 95 prikazuje postavke i ulazne parametre koje smo koristili za izračun modela. Kod izračuna modela koristimo i jednu od metoda za procjenu utjecaja pojedinačnih vrijednosti na model (Cook's D) i njihovo uklanjanje kod izračuna ukoliko značajno odstupaju od ostalih vrijednosti (engl. *outliers*) kako bi se dobio što precizniji model. Cook's D izuzima sve vrijednosti koje su veće od 4/*n* pri čemu je *n* broj elemenata u skupu (N=150).



Slika 95 Parametri za izračun modela (IBM SPSS 21)

Tablica 49 prikazuje kriterije za odabir varijabli u model; tri su varijable (ALCM, ADIT, AMC) koje su zadovoljile postavljene kriterije. Ulazni parametar za izračun modela postupnom (engl. *stepwise regression*) metodom je *p*-vrijednost manja od 0,05.

Tablica 49 Kriteriji za odabir varijabli modela

Variables Entered/Removed[a]			
Model	*Variables Entered*	*Variables Removed*	*Method*
1	ALCM		Stepwise (Criteria: Probability-of-F-to-enter <= ,050, Probability-of-F-to-remove >= ,100).
2	ADIT		Stepwise (Criteria: Probability-of-F-to-enter <= ,050, Probability-of-F-to-remove >= ,100).
3	AMC		Stepwise (Criteria: Probability-of-F-to-enter <= ,050, Probability-of-F-to-remove >= ,100).
a. Zavisna varijabla: PR			

Tablica 50 prikazuje izračun koji se sastoji od tri modela multiple regresije. Model 3 uključuje tri nezavisne varijable; posebno ćemo ga analizirati. Model uključuje samo metrike koje su u prvom koraku izračuna (univarijantna regresijska analiza) bile statistički značajne (*p*-vrijednost <= 0,05). Model multivarijantne linearne regresije prikazan je u dijelu tablice (koeficijenti).

Tablica 50 Rezultati izračuna multiple linearne regresije za PR i 7 metrika

Sažetak modela[d]									
					Change Statistics				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	,517[a]	,267	,262	5,89745	,267	54,020	1	148	,000
2	,605[b]	,366	,357	5,50620	,098	22,780	1	147	,000
3	,648[c]	,420	,408	5,28231	,055	13,725	1	146	,000
a. Prediktori: (konstanta), ALCM									
b. Prediktori: (konstanta), ALCM, ADIT									
c. Prediktori: (konstanta), ALCM, ADIT, AMC									
d. Zavisna varijabla: PR									

ANOVA[a]						
Model		SS	df	MS	F	Sig.
1	Regresija	1878,817	1	1878,817	54,020	,000[b]
	Residual	5147,425	148	34,780		
	Total	7026,242	149			
2	Regresija	2569,466	2	1284,733	42,375	,000[c]
	Residual	4456,776	147	30,318		
	Total	7026,242	149			
3	Regression	2952,438	3	984,146	35,271	,000[d]
	Residual	4073,804	146	27,903		
	Total	7026,242	149			

a. Zavisna varijabla: PR

b. Prediktori: (konstanta), ALCM

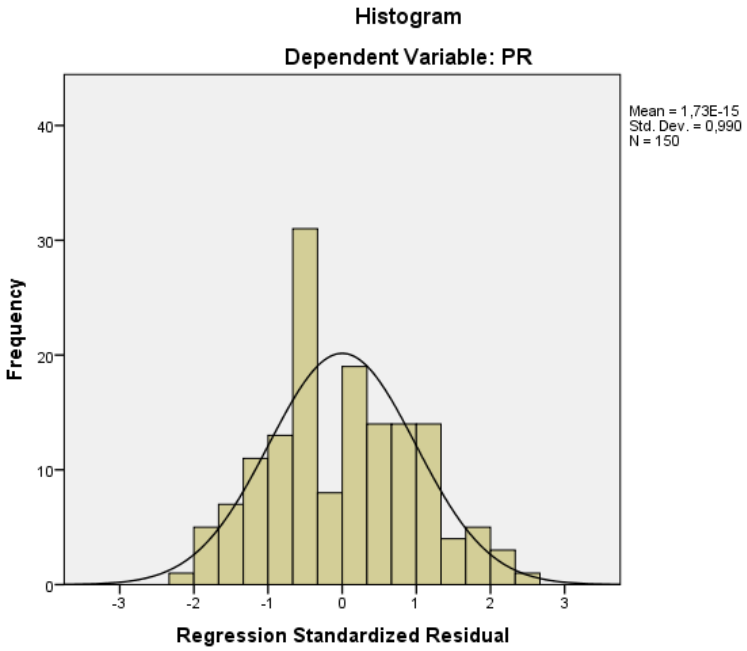
c. Prediktori: (konstanta), ALCM, ADIT

d. Prediktori: (konstanta), ALCM, ADIT, AMC

Koeficijenti[a]

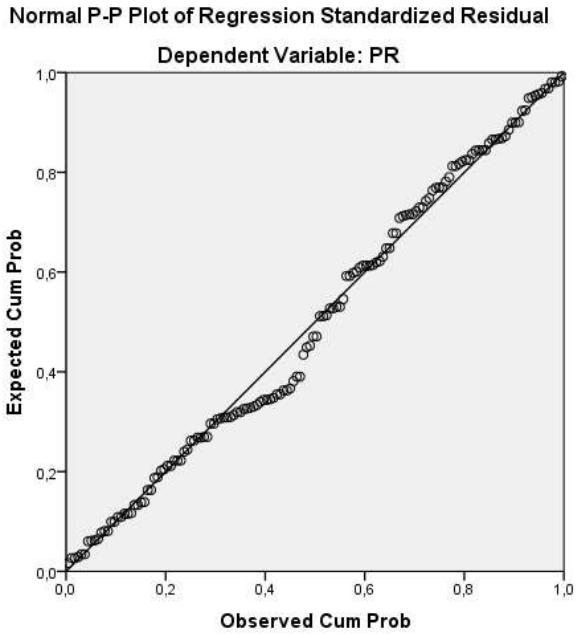
Model		Nestandar. koef.		Standar. koef.	t	Sig.	Interval pouzdanosti od 95,0% za B		Međusobna koreliranost	
		B	Stan. greška	Beta			Donja granica	Gornja granica	Toleran.	VIF
1	(konstanta)	73,328	1,699		43,151	,000	69,970	76,686		
	ALCM	-1,076	,146	-,517	-7,350	,000	-1,366	-,787	1,000	1,000
2	(konstanta)	87,112	3,295		26,436	,000	80,600	93,625		
	ALCM	-1,195	,139	-,574	-8,601	,000	-1,470	-,921	,968	1,033
	ADIT	-2,815	,590	-,319	-4,773	,000	-3,980	-1,649	,968	1,033
3	(konstanta)	90,255	3,273		27,576	,000	83,786	96,723		
	ALCM	-1,106	,136	-,531	-8,158	,000	-1,373	-,838	,937	1,068
	ADIT	-3,068	,570	-,347	-5,384	,000	-4,195	-1,942	,954	1,048
	AMC	-,399	,108	-,240	-3,705	,000	-,612	-,186	,946	1,058

Slika 96 prikazuje distribuciju rezidualnih vrijednosti koja treba biti normalna distribucija (aritmetička sredina = 0 i standardna devijacija = 1); izgled histograma je konzistentan sa navedenim zahtjevima.



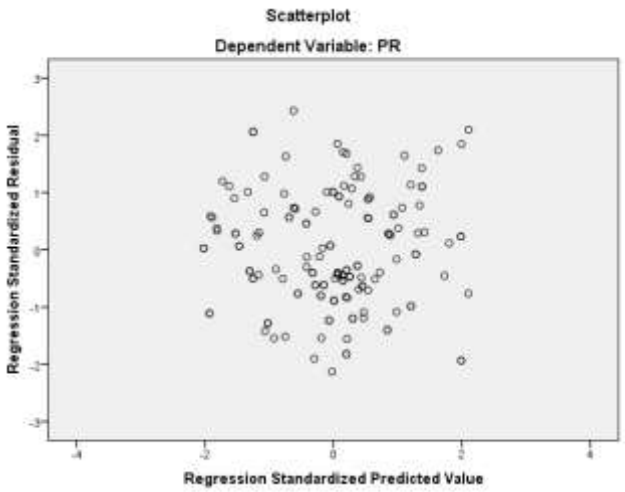
Slika 96 Histogram standardiziranih reziduala

Slika 97 prikazuje grafikon normalnih vrijednosti. Točke leže u blizini ravne linije, to je nužno za prihvaćanje modela, što ujedno potvrđuje i dojam koji se subjektivno stječe pregledom histograma o tome da su rezidualne vrijednosti normalno distribuirane.



Slika 97 Grafikon normalnih vrijednosti

Slika 98 prikazuje dijagram rasipanja reziduala u odnosu sa modelom standardiziranim predviđenim vrijednostima zavisne varijable.



Slika 98 Dijagram rasipanja „studentovih reziduala“ i predviđenih vrijednosti

Važno je da se ne zamjećuje nikakav trend među podacima, dijagram ne prikazuje sustavan odnos između pogrešaka i varijable PR, što ukazuje na to da je model linearne regresije primjenjiv na podatke skupa kojeg istražujemo jer ne postoji dokaz za neuniformiranost (heteroskedaktičnost) varijance.

Evaluacija snage linearnog modela

Snagu tumačenja linearnog regresijskog modela provjeravamo pomoću *koeficijenta determinacije* R^2 koji se interpretira kao postotak (varira od 0 do 1) varijabilnosti od Y (PR) koji je protumačen regresijskom jednadžbom. Veće vrijednosti R^2 ukazuju na bolju regresiju. Standardizirani koeficijent povezanosti (Beta[27]) iznosi 0,648 što ukazuje da se na temelju vrijednosti nezavisnih varijabli (metrika AMC, LCM, ADIT) može objasniti 42,0% varijance rezultata zavisne varijable PR. Iskusni analitičari utvrdili su da modeli koji se zasnivaju na podacima dobivenim od pojedinačnih slučajeva često imaju R^2 u rasponu od 0.10 do 0.20 (Newbold & Carlson, 2007, str. 423). Naše istraživanje odnosi se na pojedinačnu studiju slučaja, pa relativno nisku vrijednost R^2 od 0,420 možemo promatrati u tome kontekstu.

ANOVA tablica odnosi se na test nulte hipoteze da je R^2 jednak nula. R^2 koji ima vrijednost nula ukazuje na nepostojanje linearne veze između prediktora (AMC, ALCM, ADIT) i zavisne varijable (PR). ANOVA tablica pokazuje da je izračunata F[28] vrijednost jednaka 984,146 / 27,903 = 35,271 na razini značajnosti koja je manja od 0,001. Iz tablice točaka reza F-distribucije vidljivo je da je vrijednost omjera F (35,271) znatno veća od kritične vrijednosti za $\alpha = 0{,}01$ s jednim stupnjem slobode u brojniku i 148 stupnjeva slobode u nazivniku (F = 3,78), što ukazuje da razlike među varijancama nisu slučajne te da se nulta hipoteza može odbaciti. Rezultati prikazani u tablici 50 za slučaj regresije procjene PR, kazuju da je *p*-vrijednost za taj izračunati F jednaka 0.000, što daje dodatne argumente u korist odbacivanja hipoteze da ne postoji linearna veza između metrika (AMC, ALCM, ADIT) i PR.

Ako su varijable prediktori međusobno korelirane, njihov pojedinačni utjecaj je vrlo teško izračunati. Provjera međusobne koreliranosti metrika pri čemu su razine tolerancije (vrijednost veća od 0,10) za sve nezavisne varijable modela prihvatljive daje vrijednosti za ALCM=0,937, ADIT=0,954, AMC=0,946, tablica 50 (*koeficijenti*) . Isto tako „VIF“ vrijednosti (manje od 10)

[27] IBM SPSS statistički program standardizirani koeficijent povezanosti označava sa *Beta*, za razliku od uobičajene oznake za koeficijent korelacije *r*.
[28] F-omjer je vrijednost omjera varijance među skupinama i varijance unutar skupina. Ako je F-omjer veći od 1 tada potoji dovoljno argumenata da se nulta hipoteza odbaci.

za sve nezavisne varijable su također prihvatljive, tablica 50 (*koeficijenti*) vrijednosti za ALCM=1,068, ADIT=1,048, AMC=1,058.

PR model za predviđanje održavljivosti (promjenjivosti)

Procijenjena vrijednost konstante b_0 u modelu regresije iznosi ***90,255***, dok procjenjiva vrijednost koeficijenta nagiba b_1 (ALCM - prosječan broj linija koda po metodi) iznosi ***-1,106***, što ukazuje na to da povećanje prosječnog broja linija koda po metodi neke poslovne komponente umanjuje njenu održavljivost (promjenjivost), za svaku liniju približno jednu jedincu PR vrijednosti. Podsjećamo da komponentu koja ima održavljivost PR>50 smatramo održavljivom. Koeficijent nagiba b_2 (ADIT - prosječna dubina nasljeđivanja) iznosi ***-3,068***, što ukazuje da povećanje od jedne razine dubine nasljeđivanja klasa koje pripadaju komponenata umanjuje njenu održavljivost (promjenjivost) za otprilike tri jedinice PR vrijednosti. Koeficijent b_3 (AMC - prosječan broj metoda po klasi) iznosi ***-0,399***, što ukazuje da povećanje prosječnog broja metoda po klasi umanjuje održavljivost (promjenjivost) za približno polovicu jedinice PR vrijednosti. Važno je za napomenuti da vrijednosti nestandardiziranih regresijskih koeficijenata nisu relativnog karaktera, ne smijemo ih tumačiti kao relativne težinske faktore, oni se odnose na jedince mjerenja nezavisnih varijabli. Primjerice, koeficijent ALCM = -1,106 predstavlja prosječni broj linija programskog koda, jedinica mjerenja nije usporediva sa ostalim koeficijentima koji se odnose na neku drugu jedinicu mjere. Uspoređivati se mogu samo nezavisne varijable koje imaju istu jedinicu mjerenja, primjerice ako se dvije varijable mjere brojem linija koda. Za relativnu usporedbu mogu se koristiti vrijednosti standardiziranih koeficijenta iz susjedne kolone iste tablice (ALCM = -0,531, ADIT = -0,347, AMC = -0,240).

Procjeniteljska jednadžba modela je:

$$PR = 90{,}255 + (-1{,}106 * \text{ALCM}) + (-3{,}068 * ADIT) + (-0{,}399 * AMC)$$

Održavljivost (promjenjivost) poslovnih komponenata linije za proizvodnju softvera moguće je procijeniti na temelju vrijednosti tri lako dostupne metrike, ALCM metrika ima najveći utjecaj na promjenjivost komponente, dok druge dvije metrike ADIT i ACM imaju nešto manji utjecaj na promjenjivost komponenata.

Rezultati regresije sumiraju informacije sadržane u prikupljenim podacima i „ne dokazuju“ da povećanje ili smanjene vrijednosti pojedinih metrika „uzrokuje“ smanjenu ili povećanu održavljivost (promjenjivost) komponenata. Teorija iz područja softverskog inženjerstva sugerira postojanje uzročno-posljedične veze i naši rezultati tu teoriju podupiru. Deskriptivna statistika, dijagrami raspršenosti, korelacije i regresijske jednadžbe ne mogu dokazati uzročno-

posljedičnu vezu, ali mogu pružiti dokaze koji ju podupiru. Da bismo naše zaključke učinili valjanima, nužna je kombinacija teorije, iskustva u softverskom inženjerstvu i dobre statističke analize (Newbold & Carlson, 2007).

6.6. Procjena zrelosti referentne arhitekture prema FEF metodi

Jedan od istraživačkih ciljeva ovoga rada (C3.3) odnosi se na postizanje zrelosti referentne arhitekture razine „4“ (engl. *Variants Products*) prema modelu za zrelost linija za proizvodnju softvera (engl. *Family Evaluation Framework*) (F. Van Der Linden i ostali, 2004). U ovom poglavlju predstavljamo rezultate procjene zrelosti naše referentne arhitekture za poslovne aplikacije (RABA) prema pristupu linija za proizvodnju softvera.

Područje arhitekture u FEF modelu promatramo kao jednu od dimenzija modela zrelosti organizacije (ostale dimenzije uključuju poslovanje, organizaciju i proces), a odnosi se na tehnološka sredstva za razvoj aplikacija; pored uobičajenog pojma arhitekture, uključuje i zahtjeve, modeliranje domene, detaljni dizajn i testiranje. Arhitektura se nalazi jednim dijelom u aplikacijama (aplikacijska arhitektura), drugim dijelom u osnovnim komponentama (domenska arhitektura) a povezuje ih model varijabilnosti linije za proizvodnju softvera. Procjena zrelosti se uglavnom svodi na odnos tih dviju arhitektura (aplikacijske i domenske) te na način upravljanja sa varijabilnošću arhitekture.

Procjena zrelosti arhitekture prema ovom modelu radi se po slijedećim područjima:

- ***Asset reuse level***: razina korištenja postojećih komponenata u proizvodima.
- ***Reference architecture***: razina do koje referentna arhitektura utječe na arhitekturu aplikacija.
- ***Variability management***: eksplicitna upotreba točaka varijabilnosti i načina realizacije.

Rezultati procjene zrelosti arhitekture linije za proizvodnju softvera (RABA) koju smo proveli u financijskoj instituciji prikazujemo prema navedenim područjima.

Razina korištenja komponenata u proizvodima (engl. *asset reuse level*)	*Zrelost*
Najvažniji elementi ponovne upotrebe su referentna arhitektura koja se koristi u svim poslovnim aplikacijama, okvir referentne arhitekture i komponente koje su namijenjene za korištenje u više poslovnih sektora. Poslovne komponente koje pripadaju jednom	**4**

poslovnom sektoru (npr. bankarstvo) ponovno se koriste u jednoj ili više poslovnih aplikacija, također pripadaju domenskom inženjerstvu. Rezultati testiranja i korisnički zahtjevi koji se odnose na poslovne komponente isti su za sve aplikacije koje koriste te komponente. Budući da nisu sve poslovne komponente detaljno opisane i ne nalaze se u za to predviđenom repozitoriju, razina 5 nije postignuta, već je postignuta razina 4: „*postoji sistematski način ponovne upotrebe koja se temelji na upotrebi repozitorija komponenata, pri čemu komponente podržavaju eksplicitnu varijabilnost*“.	

Razina do koje referentna arhitektura utječe na arhitekturu aplikacija (engl. *reference architecture*)	*Zrelost*
Postoji jedinstvena referentna arhitektura za cijelu liniju za proizvodnju softvera. Referentna arhitektura sastoji se od tri sloja (engl. *layers*) i određuje strukturu proizvoda koji ju primjenjuju. Svaki sloj referentne arhitekture sastoji se od okvira i komponenata koje moraju biti prisutne u svakom proizvodu. Pojedinačne arhitekture aplikacija sastoje se od fiksnog broja osnovnih komponenta i varijabilnog broja ostalih poslovnih komponenta. Varijabilne komponente dodaju se u aplikacije putem konfiguracije prema definiranim točkama varijabilnosti. Budući da ne postoji automatski način konfiguriranja potrebnih komponenata za pojedinu aplikaciju, što je uvjet za ocjenjivanje ocjenom 5, razina zrelosti je ocjenjena ocjenom 4: „*postoji jedinstvena referentna arhitektura kojom se određuju točke varijabilnosti arhitektura pojedinačnih aplikacija, također, mnoga kvalitetna rješenja ugrađena su u arhitekturu linije za proizvodnju softvera“.*	**4**

Eksplicitna upotreba točaka varijabilnosti i tehnika načina realizacije (engl. *variability management*)	*Zrelost*
Varijabilnost pojedinih aplikacija određuje se na razini korisničkih zahtjeva, čime određujemo koje od komponenata se uključuju ili ne uključuju u poslovnu aplikaciju. Ponovno upotrebljive komponente imaju standardno sučelje, obilježene su sa anotacijama (engl. *annotation*) i drugim meta podacima, kako bi ih za vrijeme konfiguriranja mogli prepoznati. Navedeni mehanizmi realizacije varijabilnosti (meta podaci) su definirani na razini referentne arhitekture koja određuje pozicije gdje su varijacije moguće ili nisu. Ocjena varijabilnosti nije na razini 5 koja zahtjeva definiranje posebnog „jezika“ ili modela i potpuno automatizirano provođenje varijabilnosti. Razinu zrelosti ocjenjujemo sa ocjenom 4: „*arhitektura linije za proizvodnju softvera određuje konfiguracijske parametre arhitektura aplikacija; referentna arhitektura određuje varijabilne točke i*	**4**

ograničenja za većinu njih, postavlja pravila kojih se trebaju držati arhitekture pojedinačnih aplikacija“.	

Ukupna razina zrelosti referentne arhitekture linije za proizvodnju softvera u financijskoj instituciji za koju smo procjenjivali zrelost je na razini „**4“,** što je bio i jedan od istraživačkih ciljeva ovoga rada (C3.3). Na toj zrelosti potrebno je imati referentnu arhitekturu koja se koristi u svim poslovnim aplikacijama, zajedničke komponente i varijabilnost koje su upravljanje na sustavan način. U promatranoj liniji za proizvodnju softvera domenske komponente uključuju ne samo okvir referentne arhitekture i njegove komponente, već i poslovne komponente koje se mogu koristiti u više poslovnih aplikacija. Također, važno je da je upravljanje varijabilnošću eksplicitno definirano isključivo na razini referentne arhitekture linije za proizvodnju softvera.

7. ZAKLJUČAK

U ovom radu istražena je mogućnost primjene jednog od najučinkovitijih pristupa razvoju i održavanju softvera, koji se zove „linije za proizvodnju softvera“, za područje razvoja i održavanja poslovnih aplikacija. Jedna od najvažnijih aktivnosti kod uvođenja ovog pristupa u praksu neke organizacije je definiranje, razvoj i održavanje referentne arhitekture, njenog programskog okvira i osnovnih komponenata, što je ujedno i glavni fokus ovog istraživanja. Pojam poslovnih aplikacija obuhvaća standardne poslovne sustave za podršku temeljnim poslovnim procesima kao što su ERP, CRM SCM i ostale slične i/ili pomoćne aplikacije koje se integriraju u jedinstveni informacijski sustav.

Dosadašnja praksa najčešće korištenih pristupa pri razvoju poslovnih aplikacija pokazala je brojne nedostatke: kašnjenje na projektima razvoja, otežano održavanje, neusklađenost sa poslovnom strategijom organizacija za koje se poslovne aplikacije razvijaju, relativno velike i nerazmjerne troškove, nekonzistentnost u arhitekturi, neučinkovitu podjelu aktivnosti razvoja, nedovoljnu ponovnu upotrebu postojećih artefakata. Kao odgovor na ove izazove postavljena su četiri cilja i dvije hipoteze (poglavlje 1.5).

Na temelju analize postojećih studija slučaja primjene pristupa „linije za proizvodnju softvera“ u praksi, pregleda relevantne literature, iskustva autora u primjeni ovog pristupa, mišljenja iskusnih stručnjaka iz područja ovog istraživanja uz pomoć kojeg smo redefinirali inicijalni skup funkcionalnih zahtjeva, definirani su i potvrđeni funkcionalni i nefunkcionalni zahtjevi i strukturni aspekti novo razvijene referentne arhitekture za razvoj poslovnih aplikacija (RABA) prema pristupu linija za proizvodnju softvera čiji opseg obuhvaća više poslovnih sektora (poglavlje 5.2, 5.3).

Paralelno sa analizom postojećeg stanja i definiranja zahtjeva, kroz nekoliko iteracija smo oblikovali i razvijali artefakte novo razvijene referentne arhitekture (RABA) provođenjem studije slučaja u jednoj financijskoj instituciji. Prezentirali smo način oblikovanja komponenta okvira referentne arhitekture za čiji razvoj smo koristili nekoliko efektivnih tehnika kao što su uzorci dizajna, aplikativni okviri, generatori izvornog koda, aspekti (poglavlje 5.4.4).

Procjenu dovoljne upotrebljivosti referentne arhitekture ispitali smo putem kontroliranog eksperimenta u kojem su sudjelovali razvojni programeri koji su novo razvijenu referentnu arhitekturu koristili u praksi (poglavlje 6.2). Rezultati testiranja upotrebljivosti referentne arhitekture su potvrdili da je referentna arhitektura „dovoljno upotrebljiva“. Na temelju rezultata i sugestija ispitanika potvrđeno je da upotrebljivost referentne arhitekture ne sprječava

njenu upotrebu u praksi. Za provjeru primjene referentne arhitekture u praksi koristili smo longitudinalnu studiju pojedinačnog slučaja tipa „poboljšanja“ u jednoj financijskog instituciji (poglavlje 6.3), te na temelju rezultata primjene u praksi financijske institucije potvrdili njenu primjenjivost za poslovne namjene.

Razvojem referentne arhitekture (RABA) koja obuhvaća više poslovnih sektora horizontalno i koja se može koristiti za pojedini poslovni sektor vertikalno, te njenom primjenom i validacijom na studiji slučaja, postignuti su slijedeći ciljevi istraživanja: (C1) „*definirati i provjeriti funkcionalnosti strukturne aspekte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera*“ (poglavlje 5.3), (C2) „*odrediti i razviti artefakte referentne arhitekture za poslovne aplikacije prema pristupu linija za proizvodnju softvera koji zadovoljavaju definirane funkcionalne zahtjeve“* (poglavlje 5.4), (C3) „*provjeriti razvijene artefakte referentne arhitekture i njihovu korisnost u praksi*“ i potvrđena hipoteza (**H1**) „***nova referentna arhitektura prema pristupu linija za proizvodnju softvera je korisna za proizvodnju softvera za poslovne primjene***“.

U nedostatku prikladnih metrika i modela za mjerenje održavljivosti poslovnih komponenata koje primjenjuju programski okvir referentne arhitekture u kontekstu linija za proizvodnju softvera, definirali smo i potvrdili nove metrike i model (PR) za procjenu održavljivosti (promjenjivosti) poslovnih komponenata i aplikacija koje ih koriste. Standardni model (MI) za procjenu održavljivosti softvera, kao najraširenij i i znanstveno potvrđeni model, može se koristiti i u kontekstu linija za proizvodnju softvera, ali je prvenstveno namijenjen mjerenju održavljivosti pojedinačnih programskih proizvoda; model ne uzima u obzir specifičnosti uloge okvira referentne arhitekture, stoga smo smatrali da je potrebno definirati novi model koji je prilagođen navedenom kontekstu linija za proizvodnju softvera i ulozi koju okvir referentne arhitekture ima u tom kontekstu (poglavlje 6.5).

Model (PR) koji je definiran u ovom radu namijenjen je procjeni održavljivosti poslovnih komponenata ili cijele linije za proizvodnju softvera mjerenjem razine „preuzimanja odgovornosti“ za kompleksne interakcije sa vanjskim sustavima od strane okvira referentne arhitekture. Osnovna pretpostavka modela PR jeste da je održavljivost linije za proizvodnju softvera ugrađena u strukturu i funkcionalnost artefakata referentne arhitekture. Pretpostavili smo i pokazalo se da broj interakcija poslovnih komponenata prema vanjskim komponentama i broj interakcija sa okvirom referentne arhitekture određuje njihovu održavljivost (promjenjivost). U većini slučajeva poslovne komponente s relativno većim brojem interakcija

s okvirom referentne arhitekture u odnosu prema broju interakcija s vanjskim komponentama lakše je, jeftinije i jednostavnije održavati (mijenjati).

Rezultati koje smo dobili mjerenjem održavljivosti na stvarnim poslovnim komponentama, pokazali su da postoji korelacija između modela MI i PR, što možemo protumačiti da poznavanjem rezultata mjerenja jednog modela možemo procijeniti vrijednosti mjerenja drugim modelom. Kada smo utvrdili da postoji statistički značajna korelacija između MI i PR metrika izvornog programskog koda, utvrditi smo statistički model koji bi nam omogućio da temeljem poznatih PR vrijednosti predvidimo MI vrijednosti, sa što manjom pogreškom prognoze. Test linearne povezanosti između mjerenja modelima MI i PR daje koeficijent povezanosti od 0,52. Koeficijent povezanosti upućuje na značajnu povezanost dvaju modela, međutim, smatramo da dva modela ne mjere potpuno istu stvar - održavljivost poslovnih komponenata koje koriste istu referentnu arhitekturu prema pristupu linija za proizvodnju softvera; stoga je potrebno održavljivost promatrati iz šire perspektive nego što su to samo metrike izvornog programskoga koda koje se isključivo koriste u modelu MI. Predloženi model (PR) promatra održavljivost (promjenjivost) iz perspektive koja u obzir uzima isključivo intenzitet povezanosti poslovnih komponenata sa ostalim komponentama koje ne uključuju druge poslovne komponente, već se odnose isključivo na aplikacijski okvir referentne arhitekture i ostale vanjske komponente drugih proizvođača.

Na temelju rezultata testa korelacije metrika izvornog programskog koda pretpostavili smo da metrike kojima raspolažemo utječu na održavljivost (PR), pa smo to koristili za pronalaženje novog modela održavljivosti koji te metrike uzima u obzir kod definiranja regresijskog modela. Cilj regresijske analize je bio utvrđivanje povezanosti između skupa metrika (nezavisnih varijabli) i PR metrike (zavisne varijable), odnosno utvrđivanja regresijskog modela koji služi u analitičke i prognostičke svrhe. Regresijska analiza je pokazala da je održavljivost (promjenjivost) poslovnih komponenata linije za proizvodnju softvera moguće procijeniti na temelju vrijednosti tri lako dostupne metrike, među kojima ALCM metrika ima najveći utjecaj na promjenjivost komponenata, dok druge dvije metrike ADIT i ACM imaju nešto manji utjecaj.

U okviru empirijskog istraživanja u studiji slučaja (poglavlje 6.3) prikupljeni su podaci izvornog programskog koda za validaciju novo razvijenog modela za procjenu održavljivosti aplikacijskih komponenata koje primjenjuju artefakte referentne arhitekture kako bi postigli cilj (C4) „*provjeriti povezanost rezultata PR i MI modela za mjerenje održavljivosti aplikacijskih komponenata linije za proizvodnju softvera*“. Rezultati usporedbe MI standardnog modela za

procjenu održavljvosti i predloženog modela PR pokazali su ispravnost druge hipoteze ovoga rada (**H2**) „***rezultati određivanja održavljivosti poslovnih komponenata informacijskog sustava metrikom odgovornosti platforme podudaraju s rezultatima dobivenim uz pomoć alternativne metrike“***.

Temeljem postavljenih ciljeva koji su definirani u poglavlju 1.5 ostvarili smo sljedeće:

- Pregled literature, istraživanja i stanja u području referentnih arhitektura za poslovne aplikacije i u pristupima razvoju uz pomoć ponovno iskoristivih komponenata.
- Definirali funkcionalne zahtjeve referentne arhitekture za poslovne aplikacije i potvrdili pouzdanost u njihovu relevantnost putem anketnog upitnika.
- Implementirali artefakte referentne arhitekture za razvoj poslovnih aplikacija pristupom linija za proizvodnju softvera.
- Provjerili korisnost referentne arhitekture u praksi i njen utjecaj na održavljivost linije za proizvodnju softvera.
- Poboljšali ukupnu razinu zrelosti arhitekture linije za proizvodnju softvera u financijskoj instituciji do razine **4** prema modelu za zrelost (FEF).
- Definirali i provjerili novi model (PR) za provjeru održavljivosti poslovnih komponenata koje koriste referentnu arhitekturu.

U ovom istraživanju, osim općih istraživačkih metoda, kao krovnu istraživačku metodu koristili smo metodu znanost o oblikovanju (engl. *design science*). Proces istraživanja prema toj metodi odvijao se u pet ponavljajućih procesnih koraka (faza): definiranje problema, prijedlog rješavanja, razvoj artefakata, provjera artefakata, zaključak.

Ovo istraživanje daje teoretski i praktični doprinos u svakoj od pet faza istraživanja. U fazi definiranja problema analizirali smo stanje i probleme koji se odnose na referentnu arhitekturu poslovnih aplikacija. Posebno smo istražili pristup poznat kao „linije za proizvodnju softvera“ koje nisu dovoljno istražene u području razvoja poslovnih aplikacija. U fazi prijedloga rješenja došli smo do podataka o važnim karakteristikama koje referentna arhitektura treba posjedovati.

U prva dva koraka, pregledom literature, analizom postojećih referentnih arhitektura za poslovne aplikacije, te temeljem iskustva autora ovog istraživanja, dobili smo dovoljno informacija o tomu koje sve karakteristike (funkcionalne zahtjeve) referentne arhitekture za poslovne aplikacije po pristupu linija za proizvodnju softvera treba implementirati. Pouzdanost

u relevantnost definiranih funkcionalnih zahtjeva referentne arhitekture potvrdili smo ispitivanjem stručnjaka koji imaju iskustvo u istraživanju i praksi putem anketnog upitnika.

Polazeći od definiranih karakteristika referentne arhitekture, u trećem koraku oblikovali smo i razvili vlastitu instancu referentne arhitekture i njene artefakte korištenjem suvremenih ali i u praksi dokazanih tehnika i alata kao što su: *Eclipse 4, Java 7, Aspect Oriented, Patterns, Model Driven, Component Based, Object Oriented, Generative Programming*, itd.

U četvrtom koraku potvrdili smo da referentna arhitektura u primjeni daje dobre rezultate na temelju standardnih mjerenja, te prema metrikama koje smo predložili u okviru ovog istraživanja. Artefakte referentne arhitekture smo provjerili empirijskim metodama kako bi utvrdili „kako dobro artefakti rade“. Za provjeru da se referentna arhitektura može primjenjivati u praksi koristili smo longitudinalnu studiju pojedinačnog slučaja u jednoj financijskog instituciji. Za procjenu dovoljne upotrebljivosti artefakata koristili smo kontrolirani eksperiment u kojem su sudjelovali razvojni programeri koji artefakte referentne arhitekture koriste ili su ih koristili u praksi. U organizaciji u kojoj smo provodili studiju slučaja prikupili smo kvantitativne i kvalitativne podatke, statistički ih analizirali kako bi pored ostalog utvrdili faktore održavljivosti svake pojedine komponente koja koristi referentnu arhitekturu. Prikupljene podatke smo također koristili za potvrdu valjanosti novog predloženog modela (PR) za provjeru održavljivosti linije za proizvodnju softvera i provjeru njene povezanosti sa standardnim modelom (MI). Osim provjere korisnosti artefakta, mjerenjem i analizom metrika napravili smo i usporedbu aplikacijskog okvira sa poznatim referentnim vrijednostima.

Na temelju provedenog istraživanja možemo tvrditi da je predložena referentna arhitektura (RABA) primjenjiva i upotrebljiva u praksi za razvoj poslovnih aplikacija. Također, podaci koje smo dobili u fazi provjere mogu nam poslužiti kao temelj za daljnje istraživanje u području razvoja poslovnih aplikacija prema pristupu linija za proizvodnju softvera, za poboljšanje postojeće inačice artefakata referentne arhitekture, njenu primjenu u razvoju poslovnih aplikacija u kontekstu različitih poslovnih sektora.

Ograničenja ovog rada odnose se na broj studija slučaja, broj programskih jezika i specifično tehnološko okruženje (*Java*). Ograničenje koje se odnosi na samo jednu studiju slučaja donekle smo ublažili provođenjem longitudinalne studije slučaja, međutim, sve veća raspoloživost programskih proizvoda (ne nužno i poslovnih aplikacija) otvorenog izvornog programskoga koda (engl. *open source*) koji koriste jedinstvenu referentnu arhitekturu prema pristupu linija za proizvodnju softvera, može se iskoristiti s ciljem povećanja broja studija slučaja, kao smjer budućih istraživanja predloženog modela za mjerenje održavljivosti (PR).

Programski jezik *Java* i tehnološko okruženje koje se koristi u ovom istraživanju, već dugo je jedno od najznačajnijih za razvoj poslovnih aplikacija; istraživanje alternativnih programskih jezika i tehnoloških okruženja, primjerice .NET, te povećavanje broja funkcionalnosti okvira referentne arhitekture koje bi uključivalo komponente za integraciju sa novim paradigmama, (primjerice *cloud computing*, *data science*, *mobile services*) može biti još jedan od mogućih smjerova budućih istraživanja.

LITERATURA

Abbott, B. (2000). Software failure can lead to financial catastrophe. *InfoWorld*, *22*(40), 54-55.

Abowd, G., Bass, L., Clements, P., Kazman, R., & Northrop, L. (1997). *Recommended Best Industrial Practice for Software Architecture Evaluation.* DTIC Document.

Abran, A., Moore, J., Bourque, P., Dupuis, R. L., & Tripp, L. (2001). *Guide to the Software Engineering Body of Knowledge–SWEBOK, trial version*. IEEE-Computer Society Press.

Abreu, F. B., & Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. U *Proceedings of the 4th international conference on software quality* (Tom 186).

Ahmed, F., & Capretz, L. F. (2007). Managing the business of software product line: An empirical investigation of key business factors. *Information and Software Technology*, *49*(2), 194–208.

Aldekoa, G., Trujillo, S., Mendieta, G. S., & Díaz, O. (2006). Experience Measuring Maintainability in Software Product Lines. U *JISBD* (str. 173–182). Citeseer.

Alur, D., Malks, D., Crupi, J., Booch, G., & Fowler, M. (2003). *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc.

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., & Lucena, C. (2006). Refactoring product lines. U *Proceedings of the 5th international conference on Generative programming and component engineering* (str. 201–210). ACM.

Angelov, S., Grefen, P., & Greefhorst, D. (2012). A framework for analysis and design of software reference architectures. *Information and Software Technology*, *54*(4), 417–431.

Atkinson, C. (2002). *Component-based product line engineering with UML*. Pearson Education.

Au, Y. A. (2001). Design science I: The role of design science in electronic commerce research. *Communications of the Association for Information Systems*, *7*(1), 1.

Ayalew, Y., & Mguni, K. (2013). An Assessment of Changeability of Open Source Software. *Computer and Information Science*, *6*(3), p68.

Bagheri, E., & Gasevic, D. (2011). Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, *19*(3), 579–612.

Bansiya, J. (1997). *A hierarchical model for quality assessment of object-oriented designs*. The University of Alabama in Huntsville.

Bansiya, J. (1998). Evaluating application framework architecture structural and functional stability. *in Object-Oriented Application Frameworks: Problems & Perspectives, ME Fayad, DC Schmidt, RE Johnson (eds.), Wiley & Sons (Forthcoming), http://indus. cs. uah. edu/research-papers. htm*.

Basili, V., Briand, L., Condon, S., Kim, Y.-M., Melo, W. L., & Valett, J. D. (1996). Understanding and predicting the process of software maintenance release. U *Proceedings of the 18th international conference on Software engineering* (str. 464–474). IEEE Computer Society.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Experience factory. *Encyclopedia of software engineering*.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley Professional.

Batory, D., Cardone, R., & Smaragdakis, Y. (2000). Object-oriented frameworks and product lines. U *Software Product Lines* (str. 227–247). Springer.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. (2001).

Belsley, D. A., Kuh, E., & Welsch, R. E. (2005). *Regression diagnostics: Identifying influential data and sources of collinearity* (Tom 571). John Wiley & Sons.

Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education.

Bosch, J. (2001). Software product lines: organizational alternatives. U *Proceedings of the 23rd International Conference on Software Engineering* (str. 91–100).

Bosch, J. (2002). Maturity and evolution in software product lines: Approaches, artefacts and organization. U *Software Product Lines* (str. 257–271). Springer.

Bosch, J. (2009a). From software product lines to software ecosystems. U *Proceedings of the 13th International Software Product Line Conference* (str. 111–119). Carnegie Mellon University.

Bosch, J. (2009b). Software product lines and ecosystem.

Bosilj Vukšić, V., & Kovačić, A. (2004). *UPRAVLJANJE POSLOVNIM PROCESIMA*.

Briand, L. C., Bunse, C., & Daly, J. W. (2001). A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *Software Engineering, IEEE Transactions on*, *27*(6), 513–530.

Briand, L. C., Morasca, S., & Basili, V. R. (1996). Property-based software engineering measurement. *Software Engineering, IEEE Transactions on*, *22*(1), 68–86.

Brunner, T., & Zimmermann, A. (2012). Pattern-oriented Enterprise Architecture Management. U *PATTERNS 2012, The Fourth International Conferences on Pervasive Patterns and Applications* (str. 51–56).

Bryant, B. R., Gray, J., & Mernik, M. (2010). Domain-specific software engineering. U *Proceedings of the FSE/SDP workshop on Future of software engineering research* (str. 65–68). ACM.

Burrows, R., Garcia, A., & Taïani, F. (2010). Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies. U *Evaluation of Novel Approaches to Software Engineering* (str. 277–290). Springer.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing* (Volume 4 edition). Chichester England; New York: Wiley.

Casanave, C. (1997). Business-object architectures and standards. U *Business Object Design and Implementation* (str. 7–28). Springer.

Chen, E., & Ho, K. K.-L. (2002). Demystifying Innovation. *Perspectives on Business Innovation (8)*, 46–52.

Chidamber, S. R., & Kemerer, C. F. (1994a). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, *20*(6), 476–493.

Clements, P., & Northrop, L. (2002a). *Software product lines*. Addison-Wesley Boston.

Clements, P., & Northrop, L. (2002b). *Software product lines: Practices and Patterns*. Addison-Wesley Boston.

Clements, P., & Northrop, L. (2002c). Software product lines: practices and patterns.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., & Bone, M. (2010). The concept of reference architectures. *Systems Engineering*, *13*(1), 14–27.

Cohen, S. G. (2001). Predicting when product line investment pays. Predstavljeno na Second ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications.

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer*, *27*(8), 44–49.

Crnkovic, I., & Larsson, M. P. H. (2002). *Building reliable component-based software systems*. Artech House.

Czarnecki, K., & Eisenecker, U. W. (2000). Generative programming.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., & Wąsowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. U *Proceedings of the sixth international workshop on variability modeling of software-intensive systems* (str. 173–182). ACM.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, 319–340.

Davis, G. B., & Olson, M. H. (1984). *Management information systems: conceptual foundations, structure, and development*. McGraw-Hill, Inc.

DeBaud, J.-M., Flege, O., & Knauber, P. (1998). PuLSE-DSSA—a method for the development of software reference architectures. U *Proceedings of the third international workshop on Software architecture* (str. 25–28).

Deelstra, S., Sinnema, M., & Bosch, J. (2005). Product derivation in software product families: a case study. *Journal of Systems and Software*, *74*(2), 173–194.

Dhungana, D., Rabiser, R., Grunbacher, P., Prahofer, H., Federspiel, C., & Lehner, K. (2006). Architectural Knowledge in Product Line Engineering: An Industrial Case Stu. U *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on* (str. 186–197). IEEE.

Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grunbacher, P., Benavides, D., & Galindo, J. A. (2011). Configuration of multi product lines by bridging heterogeneous variability modeling approaches. U *Software Product Line Conference (SPLC), 2011 15th International* (str. 120–129). IEEE.

Dijkstra, E. W., Dijkstra, E. W., & Dijkstra, E. W. (1970). *Notes on structured programming*. Technological University Eindhoven Netherlands.

Dincel, E., Medvidovic, N., & van der Hoek, A. (2002). Measuring product line architectures. U *Software Product-Family Engineering* (str. 346–352). Springer.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221.

Dobrica, L., & Niemelä, E. (2000). *A strategy for analyzing product line software architectures*. Technical Research Centre of Finland.

Dumičić, K., Bahovec, V., Čižmešija, M., Kurnoga Živadinović, N., Čeh Časni, A., Jakšić, S., … Žmuk, B. (2011). *Poslovna statistika*.

Erni, K., & Lewerentz, C. (1996). Applying design-metrics to object-oriented frameworks. U *Software Metrics Symposium, 1996., Proceedings of the 3rd International* (str. 64–74). IEEE.

Fayad, M. E., Schmidt, D. C., & Johnson, R. E. (1999). *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc.

Folmer, E., & Bosch, J. (2004). Architecting for usability: a survey. *Journal of systems and software*, *70*(1), 61–78.

Folmer, E., & Bosch, J. (2005). Analyzing Software Architectures for Usability. U *Proceedings of the EuroMicro Conference on Software Engineering, Porto August*.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Fritsch, C., & Hahn, R. (2004). Product line potential analysis. U *Software Product Lines* (str. 228–237). Springer.

Fuller, R. B., & Kuromiya, K. (1992). *Cosmography: A posthumous scenario for the future of humanity*. MacMillan Publishing Company.

Fuller, R. B., & McHale, J. (1967). *World Design Science Decade, 1965-1975*. World Resources Inventory.

Gall, H., Jazayeri, M., Klosch, R. R., & Trausmuth, G. (1997). Software evolution observations based on product release history. U *Software Maintenance, 1997. Proceedings., International Conference on* (str. 160–166). IEEE.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. *Reading: Addison Wesley Publishing Company*, 30–31.

Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., … Springsteel, F. (1991). *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press.

Gomaa, H. (2005). Designing software product lines with UML. U *2012 35th Annual IEEE Software Engineering Workshop* (str. 160–216). IEEE Computer Society.

Hamza, H. S., Martinez, J., & Alonso, C. (2010). Introducing Product Line Architectures in the ERP Industry: Challenges and Lessons Learned. U *SPLC Workshops* (str. 263–266).

Heitlager, I., Kuipers, T., & Visser, J. (2007). A practical model for measuring maintainability. U Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the (str. 30–39). IEEE.

Herzum, P., & Sims, O. (2000). *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley & Sons, Inc.

Hevner, A. R., & Chatterjee, S. (2010). *Design research in information systems theory and practice*. New York; London: Springer.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004a). Design science in information systems research. *MIS quarterly*, *28*(1), 75–105.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004b). Design science in information systems research. *MIS quarterly*, *28*(1), 75–105.

Hirschheim, R., & Heinz, K. (2010). A short and glorious history of the information systems field. *Journal of the Association of Information Systems*.

Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied software architecture*. Addison-Wesley Professional.

Holmes, R., & Walker, R. J. (2012). Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *21*(4), 20.

Ho, R. (2006). *Handbook of univariate and multivariate data analysis and interpretation with SPSS*. CRC Press.

IEEE 610.12-1990. (2005). Standard Glossary of Software Engineering Terminology. Institute of Electrical and Electronics Engineers.

IEEE, A. W. G. (2000). IEEE Recommended practice for architectural description of software-intensive systems, IEEE Standard P1471.

International Organization for Standardization. (2001). ISO/IEC 9126-1: Software engineering - product quality - part 1: Quality model.

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). *The unified software development process* (Tom 1). Addison-Wesley Reading.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software reuse: architecture process and organization for business success* (Tom 285). acm Press New York.

Jalote, P. (2008). *A Concise Introduction to Software Engineering*. Springer.

Jansen, S., & Cusumano, M. A. (2013). Defining software ecosystems: a survey of software platforms and business network governance. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, 13.

John, B. E., & Bass, L. (2001). Usability and software architecture. *Behaviour & Information Technology*, *20*(5), 329–338.

Johnson, R. E., & Foote, B. (1988). Designing reusable classes. *Journal of object-oriented programming*, *1*(2), 22–35.

Junior, E. A. O., Gimenes, I. M., & Maldonado, J. C. (2013). Empirical Validation of Variability-based Complexity Metrics for Software Product Line Architecture.

Käköla, T., & Duenas, J. C. (2006). *Software Product Lines*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study*. DTIC Document.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-oriented reuse method with domain; specific reference architectures. *Annals of Software Engineering*, *5*(1), 143–168.

Kang, K. C., Sugumaran, V., & Park, S. (2010). *Applied software product line engineering*. Auerbach Publications.

Kapser, C. J., & Godfrey, M. W. (2008). "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, *13*(6), 645–692.

Kazman, R., Bass, L., Abowd, G., & Webb, M. (1993). *Analyzing the properties of user interface software*. DTIC Document.

Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: A method for analyzing the properties of software architectures. U *Proceedings of the 16th international conference on Software engineering* (str. 81–90). IEEE Computer Society Press.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. U *Engineering of Complex*

Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on (str. 68–78). IEEE.

Kircher, M., & Jain, P. (2004). Pattern-oriented software architecture volume 3: patterns for resource management.

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *Software, IEEE*, *12*(6), 42–50.

Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, *24*(2), 131–183.

Krueger, C. W. (2006). Introduction to the emerging practice of Software Product Line Development. *Methods and Tools*, *14*(3), 3–15.

Krueger, C. W. (2010). New Methods behind a New Generation of Software Product Line Successes.

Kumar, R. (2005). *Research Methodology: A Step-By-Step Guide for Beginners [RESEARCH METHODOLOGY 2/E]*. Sage Publications.

Leach, R. J. (2012). *Software Reuse: Methods, Models, Costs*. AfterMath.

Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, *7*(1), 57–78.

Linden, F. J., Schmid, K., & Rommes, E. (2010). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Li, W., & Henry, S. (1993). Object-oriented metrics that predict maintainability. *Journal of systems and software*, *23*(2), 111–122.

Losavio, F., & Matteo, A. (2013). Reference Architecture Design Using Domain Quality View. *Journal of Software Engineering & Methodology*, *3*(1), 47–61.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, *15*(4), 251–266.

Mari, M., & Eila, N. (2003). The impact of maintainability on component-based software systems. U *Euromicro Conference, 2003. Proceedings. 29th* (str. 25–32). IEEE.

Martin, R. (1994a). OO design quality metrics. *An analysis of dependencies*.

Martin, R. (1994b). OO design quality metrics. *An analysis of dependencies*.

Matsumoto, Y. (1987). *A software factory: An overall approach to software production*. IEEE CS Press.

Mattson, M., & Bosch, J. (1999). Three evaluation methods for object-oriented frameworks evolution-application, assessment and comparison. *Department of Software Engineering and Computer Karlskrona/Ronneby*.

Mattsson, M. (2000). Evolution and composition of object-oriented frameworks. *University of Karlskrona/Ronneby. Sweden*.

Mattsson, M., & Bosch, J. (1999). Observations on the Evolution of an Industrial OO Framework. U *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on* (str. 139–145). IEEE.

McCabe, T. J., & Butler, C. W. (1989). Design complexity measurement and testing. *Communications of the ACM*, *32*(12), 1415–1425.

McGregor, J. (2001). Testing a software product line.

McIlroy, M. D. (1968). Mass Produced Software Components. Predstavljeno na NATO Software Engineering Conference, Garmisch, Germany.

Minoli, D. (2008). *Enterprise architecture A to Z: frameworks, business process modeling, SOA, and infrastructure technology*. CRC Press.

Misra, S. C. (2005). Modeling design/coding factors that drive maintainability of software systems. *Software Quality Journal*, *13*(3), 297–320.

Nakagawa, E. Y., Oquendo, F., & Becker, M. (2012). RAModel: A Reference Model for Reference Architectures. U *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)* (str. 297–301).

Nash, K. S. (2000). Companies don't learn from previous IT snafus. *Computerworld*, *16*(21), 32–33.

Nasir, M. H. N., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific research and essays*, *6*(10), 2174–2186.

Newbold, P., & Carlson, W. (2007). Statistics for Business and Economics With CD.

Nielsen, J. (1994). *Usability engineering*. Elsevier.

Nöbauer, M., Seyff, N., Dhungana, D., & Stoiber, R. (2012). Managing variability of ERP ecosystems: research issues and solution ideas from microsoft dynamics AX. U *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems* (str. 21–26). ACM.

Nosek, J. T., & Palvia, P. (1990). Software maintenance management: changes in the last decade. *Journal of Software Maintenance: Research and Practice*, *2*(3), 157–174.

Olumofin, F. G., & Mišić, V. B. (2007). A holistic architecture assessment method for software product lines. *Information and Software Technology*, *49*(4), 309–323.

Oman, P., & Hagemeister, J. (1992). Metrics for assessing a software system's maintainability. U *Software Maintenance, 1992. Proceerdings., Conference on* (str. 337–344). IEEE.

OMG. (1996). Common Facilities RFP-4, Common Business Objects and Business Object Facility. Request for Proposal. OMG TC Document CF/96-01-04. Object Management Group.

Opdyke, W. F. (1992). *Refactoring object-oriented frameworks*. University of Illinois.

Oppedijk, F. R. (2008). *Comparison of the SIG Maintainability Model and the Maintainability Index*. Master's thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands.

Parnas, D. L. (1976). On the design and development of program families. *Software Engineering, IEEE Transactions on*, (1), 1–9.

Parra, C. (2011). *Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations*. Université des Sciences et Technologie de Lille-Lille I.

Petz, B., Kolesarić, V., Ivanec, D., Milas, G., Podlesek, A., & Galić, Z. (2012). *Petzova statistika: osnovne statističke metode za nematematičare*. Naklada Slap.

Poels, G., & Dedene, G. (1999). DISTANCE: A framework for software measure construction. *DTEW Research Report 9937*, 1–47.

Pohl, K., Böckle, G., & Linden, F. J. van der. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer.

Pohl, K., Böckle, G., & van der Linden, F. J. (2005). *Software product line engineering: foundations, principles, and techniques*. Springer.

Purhonen, A., Niemelä, E., & Matinlassi, M. (2004). Viewpoints of DSP software and service architectures. *Journal of Systems and Software*, *69*(1), 57–73.

Rabiser, M. R. (2009). *A user-centered approach to product configuration in software product line engineering*. Johannes Kepler Universität Linz.

Rahman, A. (2004). Metrics for the structural assessment of product line architecture. *Master's thesis, School of Engineering, Blekinge Institute of Technology*.

Reinhartz-Berger, I., Dori, D., & Katz, S. (2009). Reusing semi-specified behavior models in systems analysis and design. *Software & Systems Modeling*, *8*(2), 221–234.

Rischbeck, T., & Erl, T. (2009a). *SOA Design Patterns* (1 edition). Upper Saddle River, NJ: Prentice Hall PTR.

Rosko, Z. (2010a). Backward-Forward Transaction Service Design Pattern. U *CECIIS-2010*.

Rosko, Z. (2011). *Software Product Lines: Source Code Organization for 3-tier OLTP Architecture Systems*. CECIIS.

Rosko, Z. (2012). Strategy Pattern as a Variability Enabling Mechanism in Product Line Architecture.

Rosko, Z. (2013a). Assesing the Responsibility of Software Product Line Platform Framework for Business Applications. Predstavljeno na CECIIS-2013.

Rosko, Z. (2013b). Business applications architecture model based on software product line approach. U *International Doctoral Seminar*.

Rosko, Z., & Konecki, M. (2008a). Dynamic Data Access Object Design Pattern. U *CECIIS-2008*.

Roško, Z. (2007). Sovereign Value Object Design Pattern. U *Central European Conference on Information and Intelligent Systems*.

Roško, Z. (2014a). Case Study: Refactoring of Software Product Line Architecture-Feature Smells Analysis (str. 326–344). Predstavljeno na 25th Central European Conference on Information and Intelligent Systems, Varaždin.

Roško, Z. (2014b). Predicting the Changeability of Software Product Lines for Business Application. Predstavljeno na 23rd International Conference on Information Systems Development, Varaždin.

Roško, Z., & Strahonja, V. (2014). A Case Study of Software Product Line for Business Applications Changeability Prediction. *Journal of Information and Organizational Sciences*, *38*(2), 145–160.

Rozanski, N., & Woods, E. (2011). *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley Professional.

Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

Sarkar, S., Rama, G. M., & Kak, A. C. (2007). API-based and information-theoretic metrics for measuring the quality of software modularization. *Software Engineering, IEEE Transactions on*, *33*(1), 14–32.

Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., & Offutt, A. J. (2002). Maintainability of the Linux kernel. *IEE Proceedings-Software*, *149*(1), 18–23.

Schach, S. R., Jin, B., Yu, L., Heller, G. Z., & Offutt, J. (2003). Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering*, *8*(4), 351–365.

Schmid, K. (2004). A quantitative model of the value of architecture in product line adoption. U *Software Product-Family Engineering* (str. 32–43). Springer.

Seacord, R. C., Plakosh, D., & Lewis, G. A. (2003). *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional.

SEI, S. E. I. (2006). Ultra-Large-Scale Systems: Software Challenge of the Future.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., & Saake, G. (2012). SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, *20*(3-4), 487–517.

Simon, H. A. (1969). *The sciences of the artificial* (Tom 136). MIT press.

Svahnberg, M., Van Gurp, J., & Bosch, J. (2005). A taxonomy of variability realization techniques. *Software: Practice and Experience*, *35*(8), 705–754.

Szyperski, C. (2002). *Component software: beyond object-oriented programming*. Pearson Education.

Taylor, R. N., & Hoek, A. van der. (2007). Software Design and Architecture - The once and future focus of software engineering. *ICSE - Future of Software Engineering (FOSE'07)*, *IEEE*, 226–243.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2009). *Software architecture: foundations, theory, and practice*. Wiley Publishing.

Thiel, S., & Hein, A. (2002). Systematic integration of variability into product line architecture design. U *Software Product Lines* (str. 130–153). Springer.

Tirila, A. (2002). Variability enabling techniques for software product lines. *Department of Information Technology, Tampere University of Technology*, *59*.

Tizzei, L. P., Dias, M., Rubira, C. M., Garcia, A., & Lee, J. (2011). Components meet aspects: Assessing design stability of a software product line. *Information and Software Technology*, *53*(2), 121–136.

Tsai, W.-H., Shaw, M. J., Fan, Y.-W., Liu, J.-Y., Lee, K.-C., & Chen, H.-C. (2011). An empirical investigation of the impacts of internal/external facilitators on the project success of ERP: A structural equation model. *Decision Support Systems*, *50*(2), 480–490.

Ulrich, W. M., & Newcomb, P. (2010). *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Morgan Kaufmann.

Vaishnavi, V. K., & Kuechler Jr, W. (2007). *Design science research methods and patterns: innovating information and communication technology*. Auerbach Publications.

Vaishnavi, V., & Kuechler, W. (2004). Design research in information systems.

Van der Linden, F. (2002). Software product families in Europe: the Esaps & Cafe projects. *Software, IEEE*, *19*(4), 41–49.

Van Der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., & Obbink, H. (2004). Software product family evaluation. U *Software Product Lines* (str. 110–129). Springer.

Van Der Linden, F. J., & Muller, J. K. (1995). Creating architectures with building blocks. *Software, IEEE*, *12*(6), 51–60.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *ACM Sigplan Notices*, *35*(6), 26–36.

Van Ommering, R. (2002). Building product populations with software components. U *Proceedings of the 24th international conference on Software engineering* (str. 255–265). ACM.

Van Vliet, H. (2008). *Software engineering: principles and practice* (Tom 2). Wiley.

Verčić, A. T., Sinčić Ćorić, D., & Pološki Vokić, N. (2010). Priručnik za metodologiju istraživačkog rada–kako osmisliti, provesti i opisati znanstveno i stručno istraživanje. *Zagreb: MEP*.

Vladimír, M. (2014). *Handbook of Research on Design and Management of Lean Production Systems*. IGI Global.

Völter, M., Kircher, M., & Zdun, U. (2005). *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. John Wiley & Sons.

Weiss, D. M. (2008). The Product Line Hall of Fame. U *SPLC* (Tom 8, str. 39).

Weiss, D. M., & others. (1999). Software product-line engineering: a family-based software development process.

Welker, K. D., & Oman, P. W. (1995). Software maintainability metrics models in practice. *Crosstalk, Journal of Defense Software Engineering*, *8*(11), 19–23.

Whetten, D. A. (1989). What Constitutes a Theoretical Contribution? *The Academy of Management Review*, *14*(4), 490–495. http://doi.org/10.2307/258554

Wijnstra, J. G. (2002). Critical factors for a successful platform-based product family approach. *Software Product Lines* (str. 68–89). Springer.

Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical research methods in software engineering. U *Empirical Methods and Studies in Software Engineering* (str. 7–23). Springer.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM systems journal*, *26*(3), 276–292.

Zelenika, R. (2011). *Metodologija i tehnologija izrade znanstvenog i stručnog djela: Pisana djela na stručnim i sveučilišnim studijama*. Ekonomski fakultet u Rijeci.

Zhang, T., Deng, L., Wu, J., Zhou, Q., & Ma, C. (2008). Some metrics for accessing quality of product line architecture. U *Computer Science and Software Engineering, 2008 International Conference on* (Tom 2, str. 500–503). IEEE.

Zhou, Y., & Xu, B. (2008). Predicting the maintainability of open source software using design metrics. *Wuhan University Journal of Natural Sciences*, *13*(1), 14–20.

Zweben, S. H., Edwards, S. H., Weide, B. W., & Hollingsworth, J. E. (1995). The effects of layering and encapsulation on software development cost and quality. *Software Engineering, IEEE Transactions on*, *21*(3), 200–208.

Žugaj, M., Dumičić, K., & Dušak, V. (2006). *Temelji znanstvenoistraživačkog rada: Metodologija i metodika (2. dopunjeno i izmijenjeno izd.)*. Varaždin.

DODATAK A: METRIKE KOMPONENATA (*V2, V3, V4*)

ID	*D3*	*D4*	*D5*	***PR***	***MI***	*ABD*	*ACC*	*ADIT*	*ALCM*	*LOC*	*NCL*	*NMETH*	*AMC*	*NOS*	*WM*	*AWMC*
SC1	13	6	0	68,42	106,66	1,45	1,73	4,00	10,90	140	2	11	5,50	108	20	10,00
SC2	104	79	0	56,83	104,22	1,21	1,30	5,77	8,88	1352	18	119	6,61	1108	156	8,67
SC3	8	5	0	61,54	104,90	1,38	1,46	3,00	10,61	166	2	13	6,50	139	19	9,50
SC4	25	17	0	59,52	96,39	1,53	1,58	4,20	13,30	214	5	12	2,40	186	19	3,80
SC5	42	38	0	52,50	88,42	1,62	2,47	4,61	20,44	939	13	38	2,92	818	99	7,62
SC6	30	34	0	46,88	97,44	1,60	1,79	4,20	16,64	1384	5	78	15,60	1162	140	28,00
SC7	26	16	0	61,90	94,63	1,27	1,73	3,60	16,27	431	5	22	4,40	361	38	7,60
SC8	10	4	0	71,43	112,42	1,18	1,19	3,00	6,93	143	2	16	8,00	117	19	9,50
SC9	17	15	0	53,13	101,15	1,47	1,59	3,66	13,94	881	3	59	19,67	737	94	31,33
SC10	16	8	0	66,67	97,52	1,51	1,59	4,33	14,72	597	3	37	12,33	518	60	20,00
SC11	6	5	0	54,55	100,87	1,50	1,50	3,50	13,09	292	2	20	10,00	250	30	15,00
SC12	60	47	0	56,07	96,84	1,41	2,06	5,63	14,53	1103	11	65	5,91	853	138	12,55
SC13	81	60	0	57,45	107,64	1,23	1,32	5,25	8,99	1531	16	141	8,81	1264	187	11,69
SC14	29	19	0	60,42	109,40	1,25	1,33	3,75	9,22	658	4	63	15,75	535	71	17,75
SC15	8	5	0	61,54	99,00	1,42	1,71	4,00	12,28	110	2	7	3,50	92	12	6,00
SC16	30	18	0	62,50	97,50	1,55	1,76	4,20	14,23	636	5	38	7,60	536	67	13,40
SC17	64	44	0	59,26	99,71	1,43	1,65	5,59	13,56	1416	10	91	9,10	1163	150	15,00
SC18	19	12	0	61,29	96,35	1,57	1,86	4,00	16,14	273	3	14	4,67	211	26	8,67
SC19	18	14	0	56,25	100,79	1,45	1,75	5,00	11,25	284	4	20	5,00	241	36	9,00
SC20	38	29	0	56,72	114,35	1,12	1,30	6,29	6,51	832	17	100	5,88	631	132	7,76
SC21	16	13	0	55,17	99,03	1,53	2,19	4,66	16,00	461	3	26	8,67	344	59	19,67
SC22	8	5	0	61,54	102,63	1,46	1,53	4,00	12,53	214	2	15	7,50	175	23	11,50
SC23	11	4	0	73,33	105,20	1,30	1,31	3,00	9,92	158	2	13	6,50	131	17	8,50
SC24	10	5	0	66,67	104,45	1,39	1,47	4,66	11,06	199	3	15	5,00	163	22	7,33
SC25	26	19	0	57,78	100,93	1,47	1,63	4,50	12,68	567	6	38	6,33	472	63	10,50
SC26	31	21	0	59,62	104,11	1,34	1,51	4,20	10,95	538	5	41	8,20	445	63	12,60
SC27	31	24	0	56,36	104,26	1,33	1,43	4,00	10,80	541	5	42	8,40	448	61	12,20
SC28	15	13	0	53,57	98,72	1,53	1,91	4,00	15,13	701	3	43	14,33	568	86	28,67
SC29	78	57	0	57,78	95,56	1,52	1,86	4,53	15,73	1645	13	91	7,00	1371	169	13,00
SC30	20	10	0	66,67	98,15	1,52	1,52	3,83	14,28	355	6	21	3,50	305	32	5,33
SC31	13	13	0	50,00	97,50	1,54	1,92	4,33	15,37	423	3	24	8,00	348	46	15,33
SC32	29	20	0	59,18	105,75	1,22	1,40	5,33	8,65	397	6	35	5,83	336	49	8,17
SC33	33	26	0	55,93	99,11	1,54	1,89	3,60	15,53	1887	5	114	22,80	1544	219	43,80
SC34	17	14	0	54,84	103,02	1,43	1,43	4,33	11,91	501	3	37	12,33	427	53	17,67
SC35	9	6	0	60,00	102,78	1,52	1,84	3,00	12,57	265	2	19	9,50	213	35	17,50
SC36	142	116	0	55,04	98,42	1,49	1,87	5,65	14,23	4043	26	255	9,81	3330	487	18,73
SC37	10	9	0	52,63	99,21	1,45	1,64	3,50	13,63	180	2	11	5,50	153	18	9,00
SC38	15	11	0	57,69	104,52	1,33	1,44	3,66	10,44	231	3	18	6,00	192	26	8,67
SC39	6	4	0	60,00	93,54	1,64	2,29	4,00	18,52	343	2	17	8,50	297	39	19,50
SC40	27	28	0	49,09	98,56	1,43	1,68	5,42	12,91	578	7	37	5,29	488	62	8,86

SC41	6	4	0	60,00	107,45	1,33	1,50	3,50	7,33	67	2	6	3,00	55	9	4,50
SC42	55	24	0	69,62	102,23	1,56	1,97	3,85	12,43	966	14	65	4,64	742	132	9,43
SC43	12	8	0	60,00	101,16	1,50	1,50	4,33	11,66	177	3	12	4,00	149	18	6,00
SC44	8	4	0	66,67	109,75	1,27	1,27	4,00	7,90	114	2	11	5,50	94	14	7,00
SC45	11	5	0	68,75	109,43	1,36	1,55	4,50	7,90	115	2	11	5,50	91	18	9,00
SC1	13	6	0	68,42	106,66	1,45	1,73	4,00	10,90	140	2	11	5,50	108	20	10,00
SC2	104	79	0	56,83	104,22	1,21	1,30	5,77	8,88	1352	18	119	6,61	1108	156	8,67
SC3	8	5	0	61,54	104,90	1,38	1,46	3,00	10,61	166	2	13	6,50	139	19	9,50
SC4	25	17	0	59,52	95,22	1,53	1,62	4,20	13,30	240	5	13	2,60	209	21	4,20
SC5	42	38	0	52,50	88,42	1,62	2,47	4,61	20,44	939	13	38	2,92	818	99	7,62
SC6	30	34	0	46,88	96,25	1,60	1,82	4,20	16,64	1396	5	78	15,60	1173	142	28,40
SC7	26	16	0	61,90	94,63	1,27	1,73	3,60	16,27	431	5	22	4,40	361	38	7,60
SC8	10	4	0	71,43	112,42	1,18	1,19	3,00	6,93	143	2	16	8,00	117	19	9,50
SC9	17	15	0	53,13	101,15	1,47	1,59	3,66	13,94	881	3	59	19,67	737	94	31,33
SC10	16	8	0	66,67	97,52	1,51	1,59	4,33	14,72	597	3	37	12,33	518	60	20,00
SC11	6	5	0	54,55	100,87	1,50	1,50	3,50	13,09	292	2	20	10,00	250	30	15,00
SC12	60	47	0	56,07	96,77	1,41	2,08	5,63	14,53	1121	11	65	5,91	870	139	12,64
SC13	81	60	0	57,45	107,64	1,23	1,32	5,25	8,99	1531	16	141	8,81	1264	187	11,69
SC14	29	19	0	60,42	109,40	1,25	1,33	3,75	9,22	658	4	63	15,75	535	84	21,00
SC15	8	5	0	61,54	99,00	1,42	1,71	4,00	12,28	110	2	7	3,50	92	12	6,00
SC16	30	18	0	62,50	97,50	1,55	1,76	4,20	14,23	636	5	38	7,60	536	67	13,40
SC17	144	93	0	60,76	98,58	1,43	1,59	5,30	13,58	3199	23	206	8,96	2617	330	14,35
SC18	19	12	0	61,29	95,69	1,57	1,86	4,00	16,14	273	3	14	4,67	211	26	8,67
SC19	18	14	0	56,25	100,13	1,45	1,75	5,00	11,25	284	4	20	5,00	241	36	9,00
SC20	38	29	0	56,72	113,76	1,11	1,29	6,29	6,50	843	17	102	6,00	638	134	7,88
SC21	16	13	0	55,17	98,37	1,53	2,19	4,66	16,00	461	3	26	8,67	344	59	19,67
SC22	8	5	0	61,54	101,97	1,46	1,53	4,00	12,53	214	2	15	7,50	175	23	11,50
SC23	11	4	0	73,33	104,54	1,30	1,31	3,00	9,92	158	2	13	6,50	131	17	8,50
SC24	10	5	0	66,67	103,80	1,39	1,47	4,66	11,06	199	3	15	5,00	163	22	7,33
SC25	26	19	0	57,78	100,28	1,47	1,63	4,50	12,68	567	6	38	6,33	472	63	10,50
SC26	31	21	0	59,62	103,43	1,34	1,51	4,20	10,97	539	5	41	8,20	446	63	12,60
SC27	31	24	0	56,36	103,61	1,33	1,43	4,00	10,80	541	5	42	8,40	448	61	12,20
SC28	15	13	0	53,57	98,06	1,53	1,91	4,00	15,13	701	3	43	14,33	568	86	28,67
SC29	78	58	0	57,35	94,89	1,52	1,86	4,53	15,85	1656	13	91	7,00	1373	169	13,00
SC30	20	10	0	66,67	97,50	1,52	1,52	3,83	14,28	355	6	21	3,50	305	32	5,33
SC31	13	13	0	50,00	96,84	1,54	1,92	4,33	15,37	423	3	24	8,00	348	46	15,33
SC32	29	20	0	59,18	105,10	1,22	1,40	5,33	8,65	397	6	35	5,83	336	49	8,17
SC33	33	26	0	55,93	98,37	1,54	1,90	3,60	15,64	1899	5	114	22,80	1555	221	44,20
SC34	17	14	0	54,84	102,36	1,43	1,43	4,33	11,91	501	3	37	12,33	427	53	17,67
SC35	9	6	0	60,00	102,13	1,52	1,84	3,00	12,57	265	2	19	9,50	213	35	17,50
SC36	142	115	0	55,25	97,76	1,49	1,87	5,65	14,25	4049	26	255	9,81	3335	488	18,77
SC37	10	9	0	52,63	98,56	1,45	1,64	3,50	13,63	180	2	11	5,50	153	18	9,00
SC38	15	11	0	57,69	103,87	1,33	1,44	3,66	10,44	231	3	18	6,00	192	26	8,67
SC39	6	4	0	60,00	92,73	1,70	2,35	4,00	18,94	351	2	17	8,50	304	40	20,00
SC40	27	28	0	49,09	97,84	1,47	1,71	5,42	13,44	611	7	38	5,43	517	65	9,29

SC41	6	4	0	60,00	106,80	1,33	1,50	3,50	7,33	67	2	6	3,00	55	9	4,50
SC42	56	26	0	68,29	101,56	1,57	1,97	3,93	12,45	1014	15	68	4,53	775	138	9,20
SC43	12	8	0	60,00	100,51	1,50	1,50	4,33	11,66	177	3	12	4,00	149	18	6,00
SC44	8	4	0	66,67	109,09	1,27	1,27	4,00	7,90	114	2	11	5,50	94	14	7,00
SC45	11	5	0	68,75	108,78	1,36	1,55	4,50	7,90	115	2	11	5,50	91	18	9,00
SC46	32	26	0	55,17	104,53	1,35	1,43	5,33	11,16	717	6	56	9,33	599	80	13,33
SC47	51	26	0	66,23	107,14	1,29	1,32	5,00	9,67	1126	12	100	8,33	933	132	11,00
SC48	15	11	0	57,69	114,85	1,20	1,42	5,79	6,45	201	5	24	4,80	153	37	7,40
SC1	8	3	0	72,73	113,18	1,30	1,50	4,50	7,00	93	2	10	5,00	56	15	7,50
SC2	53	45	0	54,08	112,93	1,19	1,29	5,83	6,73	1012	18	119	6,61	638	154	8,56
SC3	24	13	0	64,86	112,80	1,35	1,35	4,66	7,38	293	6	31	5,17	177	42	7,00
SC4	15	9	0	62,50	103,15	1,41	1,58	4,40	10,16	173	5	12	2,40	120	19	3,80
SC5	28	24	0	53,85	92,50	1,71	2,49	4,71	16,76	793	14	39	2,79	562	102	7,29
SC6	15	15	0	50,00	104,44	1,54	1,77	4,40	11,62	934	5	74	14,80	625	131	26,20
SC7	19	13	0	59,38	97,36	1,18	1,64	3,79	13,77	364	5	22	4,40	278	36	7,20
SC8	8	2	0	80,00	119,72	1,18	1,19	3,50	5,50	116	2	16	8,00	71	19	9,50
SC9	11	8	0	57,89	109,85	1,41	1,59	4,00	9,75	614	3	58	19,33	382	92	30,67
SC10	10	4	0	71,43	106,51	1,45	1,57	4,66	11,27	459	3	37	12,33	296	59	19,67
SC11	4	2	0	66,67	109,86	1,45	1,50	4,00	9,25	211	2	20	10,00	135	30	15,00
SC12	35	28	0	55,56	101,86	1,38	2,03	5,72	11,75	886	11	65	5,91	566	136	12,36
SC13	48	35	0	57,83	113,79	1,20	1,30	5,43	7,14	1217	16	141	8,81	788	183	11,44
SC14	22	13	2	59,46	115,08	1,31	1,42	4,00	7,67	555	4	64	16,00	345	89	22,25
SC15	6	3	0	66,67	104,02	1,45	1,45	4,50	10,72	138	2	11	5,50	97	16	8,00
SC16	20	8	0	71,43	104,52	1,55	1,76	4,40	10,94	494	5	38	7,60	329	67	13,40
SC17	101	59	0	63,13	105,12	1,42	1,63	5,40	10,87	2888	25	234	9,36	1819	383	15,32
SC18	6	2	0	75,00	113,48	1,44	1,44	4,00	7,88	92	2	9	4,50	54	13	6,50
SC19	11	8	0	57,89	107,23	1,39	1,70	5,25	8,90	225	4	20	5,00	147	35	8,75
SC20	27	11	0	71,05	117,97	1,08	1,22	6,41	5,63	724	17	100	5,88	423	122	7,18
SC21	10	6	0	62,50	108,67	1,50	2,04	5,00	11,53	335	3	26	8,67	191	55	18,33
SC22	4	3	0	57,14	112,03	1,39	1,47	4,50	9,06	156	2	15	7,50	94	22	11,00
SC23	7	2	0	77,78	112,91	1,30	1,31	3,50	7,53	121	2	13	6,50	78	17	8,50
SC24	6	3	0	66,67	112,05	1,33	1,40	5,00	8,33	152	3	15	5,00	93	21	7,00
SC25	17	11	0	60,71	109,13	1,42	1,53	4,66	9,18	417	6	38	6,33	263	59	9,83
SC26	22	10	0	68,75	109,33	1,35	1,49	4,40	8,40	453	5	45	9,00	293	68	13,60
SC27	22	11	0	66,67	111,82	1,33	1,40	4,20	7,97	405	5	42	8,40	261	60	12,00
SC28	10	7	0	58,82	106,21	1,45	1,86	4,50	11,36	543	4	44	11,00	334	86	21,50
SC29	46	31	0	59,74	103,38	1,46	1,86	4,61	11,71	1226	13	91	7,00	817	170	13,08
SC30	14	6	0	70,00	107,93	1,52	1,52	4,00	9,47	244	6	21	3,50	157	32	5,33
SC31	7	7	0	50,00	106,68	1,45	1,83	4,66	10,41	294	3	24	8,00	194	44	14,67
SC32	15	10	0	60,00	111,88	1,19	1,23	5,50	6,62	302	6	35	5,83	197	43	7,17
SC33	21	14	0	60,00	106,22	1,50	1,83	4,00	11,08	1374	5	115	23,00	887	215	43,00
SC34	10	7	0	58,82	110,04	1,40	1,43	4,66	8,67	370	3	37	12,33	241	53	17,67
SC35	6	3	0	66,67	112,26	1,52	1,79	3,50	9,42	201	2	19	9,50	123	34	17,00
SC36	95	55	0	63,33	105,33	1,48	1,76	5,73	11,04	3133	26	254	9,77	2089	458	17,62
SC37	8	5	0	61,54	105,36	1,36	1,45	4,00	10,45	142	2	11	5,50	100	16	8,00

SC38	9	6	0	60,00	112,31	1,27	1,33	4,33	8,00	179	3	18	6,00	116	24	8,00
SC39	4	2	0	66,67	100,88	1,64	2,00	4,50	13,58	257	2	17	8,50	179	35	17,50
SC40	15	15	0	50,00	106,21	1,44	1,63	5,57	9,92	457	7	38	5,43	308	62	8,86
SC41	4	2	0	66,67	116,57	1,33	1,33	4,00	5,50	53	2	6	3,00	31	8	4,00
SC42	70	31	0	69,31	110,53	1,47	1,84	3,91	8,15	988	24	95	3,96	533	179	7,46
SC43	8	4	0	66,67	110,31	1,50	1,50	4,66	8,33	130	3	12	4,00	82	18	6,00
SC44	5	2	0	71,43	116,66	1,27	1,27	4,50	6,27	91	2	11	5,50	55	14	7,00
SC45	6	2	0	75,00	116,68	1,19	1,20	5,00	6,00	82	2	10	5,00	47	12	6,00
SC46	21	12	0	63,64	109,58	1,34	1,45	5,50	8,70	557	6	55	9,17	366	80	13,33
SC47	37	15	0	71,15	112,56	1,29	1,33	5,25	7,71	914	12	101	8,42	576	134	11,17
SC48	21	9	0	70,00	119,07	1,17	1,17	6,00	5,68	220	5	29	5,80	129	34	6,80
SC49	31	20	0	60,78	106,90	1,38	1,40	6,09	8,54	471	11	42	3,82	318	59	5,36
SC50	19	9	0	67,86	108,04	1,41	1,42	5,00	7,33	134	5	12	2,40	86	17	3,40
SC51	11	5	0	68,75	113,46	1,37	1,38	5,00	6,00	76	3	8	2,67	46	11	3,67
SC52	24	10	0	70,59	109,09	1,39	1,42	4,83	8,84	411	6	40	6,67	269	57	9,50
SC53	17	8	0	68,00	99,92	1,75	2,19	4,00	12,93	246	4	16	4,00	167	35	8,75
SC54	9	2	0	81,82	116,85	1,33	1,33	3,50	6,33	75	2	9	4,50	45	12	6,00
SC55	22	12	8	52,38	103,09	1,77	2,19	4,83	11,33	479	6	36	6,00	302	81	13,50
SC56	10	4	1	66,67	116,76	1,18	1,18	4,00	6,09	88	2	11	5,50	54	13	6,50
SC57	7	3	0	70,00	113,89	1,50	1,50	3,66	7,75	84	3	8	2,67	47	12	4,00

ŽIVOTOPIS

<table>
<tr><td>Zdravko Roško</td><td>Datum rođenja:</td><td>12. studenoga 1962.</td></tr>
<tr><td rowspan="3">Stablinac 4/58
22211 Vodice
Hrvatska</td><td>Mjesto rođenja:</td><td>Sonković, Šibenik</td></tr>
<tr><td>Ime i prezime majke i/ili oca:</td><td>Anica Roško i Jerko Roško</td></tr>
<tr><td>Mob:</td><td>+385.99.700.7822</td></tr>
<tr><td>e-mail: zrosko@gmail.com</td><td>Tel:</td><td>+385.22.441.423</td></tr>
<tr><td colspan="3">Linkedin: ca.linkedin.com/in/zdravkorosko</td></tr>
<tr><td rowspan="3">OBRAZOVANJE:</td><td colspan="2">1977.-1981. Srednja ekonomska škola (programer), Šibenik.</td></tr>
<tr><td colspan="2">1982.-1988. Diplomski studij poslovne ekonomije (informacijski sustavi) – Ekonomski Fakultet Zagreb, diplomirao, 10.11.1988.</td></tr>
<tr><td colspan="2">2006.-2013. Poslijediplomski doktorski studij informacijskih znanosti – Fakultet Organizacije i Informatike Varaždin, položen kvalifikacijski doktorski ispit, 11.6.2013.</td></tr>
<tr><td rowspan="11">RADNO ISKUSTVO:</td><td colspan="2">2015. McKinsey & Company, New York – vanjski konzultant.</td></tr>
<tr><td colspan="2">2007.-2015. Jadranska banka d.d., Šibenik – arhitektura i razvoj bankarskih aplikacija, procjena rizika, definiranje standarda.</td></tr>
<tr><td colspan="2">2005.-2007. Adriacom software d.o.o., Vodice – razvoj aplikacija za komunalna poduzeća.</td></tr>
<tr><td colspan="2">2004.-2004. Banksoft d.o.o., Zagreb – arhitektura i razvoj osnovnih bankarskih komponenata.</td></tr>
<tr><td colspan="2">2003.-2003. Hrvatski telekom d.d., Zagreb – arhitektura i razvoj sustava za inventuru telekomunikacijske opreme.</td></tr>
<tr><td colspan="2">1999.-2002. Merck, New Jersey – glavni arhitekt na projektu razvoja informacijskog sustava za zdravstveno osiguranje koji uključuje veliki broj korisnika sustava i preko 10 milijuna osiguranika.</td></tr>
<tr><td colspan="2">1999.-1999. First Marathon Securities, Toronto – glavni arhitekt na projektu razvoja financijskih aplikacija.</td></tr>
<tr><td colspan="2">1997.-1999. IBM Canada, Toronto – konzultant za arhitekturu i razvoj poslovnih komponenata za bankarstvo.</td></tr>
<tr><td colspan="2">1996.-1997. Nortel Canada, Toronto – arhitektura i razvoj sustava za inventuru telekomunikacijske opreme.</td></tr>
<tr><td colspan="2">1990.-1995. Air Canada, Toronto – programer/analitičar na sustavu rezervacija, osiguranja i prodaje.</td></tr>
<tr><td colspan="2">1988.-1989. APIS (CAOP), Zagreb – programer/analitičar na projektima poslovnih aplikacija.</td></tr>
</table>

POPIS RADOVA

1. Roško, Z., Strahonja, V. (2014): A Case Study of Software Product Line for Business Applications Changeability Prediction, Journal of Information and Organizational Sciences, 38 (2), str. 145-160, Varaždin.
2. Roško, Z. (2014): Predicting the Changeability of Software Product Lines for Business Application, Proceedings of 23rd International Conference on Information Systems Development (ISD2014 Croatia), Varaždin.
3. Roško, Z. (2014): Case Study: Refactoring of Software Product Line Architecture-Feature Smells Analysis, Proceedings of 25th Central European Conference on Information and Intelligent Systems (CECIIS), str. 326-344, Varaždin.
4. Roško, Z. (2013): Assessing the Responsibility of Software Product Line Platform Framework for Business Applications, Proceedings of 24th Central European Conference on Information and Intelligent Systems (CECIIS), str. 276-282, Varaždin.
5. Roško, Z. (2013): Business applications architecture model based on software product line approach, Research Papers Faculty of Materials Science and Technology Slovak University of Technology, 21 (Special Issue), str. 90-97, Dubrovnik.
6. Roško, Z. (2012): Strategy Pattern as a Variability Enabling Mechanism in Product Line Architecture, Proceedings of 23rd Central European Conference on Information and Intelligent Systems (CECIIS), str. 463-470, Varaždin.
7. Roško, Z. (2011): Software Product Lines: Source Code Organization for 3-tier OLTP Architecture Systems, Proceedings of 22nd Central European Conference on Information and Intelligent Systems (CECIIS), Varaždin.
8. Roško, Z. (2010): Backward-Forward Transaction Service Design Pattern, Proceedings of 21st Central European Conference on Information and Intelligent Systems (CECIIS), str. 509-516, Varaždin.
9. Roško, Z., Konecki, M. (2008): Dynamic Data Access Object Design Pattern, Proceedings of 19th Central European Conference on Information and Intelligent Systems (CECIIS), Varaždin.
10. Roško, Z. (2007): Sovereign Value Object Design Pattern, Proceedings of 18th Central European Conference on Information and Intelligent Systems (CECIIS), str. 420-423, Varaždin.
11. Kuzic, J., Roško, Z. (2006): Navigating customer satisfaction web site issues, Information Technology Interfaces, 28th International Conference, Cavtat, str. 407-412.