Univerzitet u Novom Sadu
Ekonomski fakultet u Subotici
Studijski program: Poslovna informatika

Metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima

Doktorska disertacija

Mentor: **Prof. dr Pere Tumbas**

Kandidat: **Mirjana Marić**

Subotica, 2015. godine

Univerzitet u Novom Sadu
Ekonomski fakultet u Subotici

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, RBR:	
Identifikacioni broj, IBR:	
Tip dokumentacije, TD:	Monografska dokumentacija
Tip zapisa, TZ:	Tekstualni štampani materijal
Vrsta rada, VR:	Doktorska disertacija
Ime i prezime autora, AU:	Mirjana Marić
Mentor, MN:	Dr Pere Tumbas, redovni profesor
Naslov rada, NR:	Metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima
Jezik publikacije, JP:	Srpski
Jezik izvoda, JI:	Srpski / engleski
Zemlja publikovanja, ZP:	Srbija
Uže geografsko područje, UGP:	Vojvodina
Godina, GO:	2015.
Izdavač, IZ:	Autorski reprint
Mesto i adresa, MA:	24000 Subotica, Segedinski put 9-11
Fizički opis rada, FO: (broj poglavlja / stranica / slika / grafikona / tabela / referenci / priloga)	6/290/35/6/51/200/17
Naučna oblast, NO:	Ekonomske nauke – poslovna informatika
Naučna disciplina, ND:	Analiza i dizajn informacionih sistema
Predmetna odrednica, ključne reči, PO:	Softverska arhitektura, agilni procesi razvoja, framework
UDK	
Čuva se, ČU:	Biblioteka Ekonomskog fakulteta u Subotici, Segedinski put 9-11, 24000 Subotica
Važna napomena, VN:	Nema
Izvod, IZ:	Softverski sistemi pokazuju tendenciju sve veće decentralizovanosti, heterogenosti, kao i sve veću potrebu za povezivanjem sa drugim sistemima, kako bi efikasno podržali poslovanje savremenih preduzeća. Njih karakteriše visoka frekventnost promena poslovnih zahteva, ponderisana konstantno rastućom kompleksnošću njihovog konteksta, koju dodatno komplikuje nepodudaranje fizičkog i logičkog područja poslovanja. Iz tog razloga se, u novije vreme, prepoznaje značaj upotrebe agilnih procesa u domenu razvoja

	kompleksnih sistema, iako inicijalno namenjenih za razvoj manje kompleksnih softverskih proizvoda. U skladu sa opisanim tendencijama, u industriji softvera postoji gap između stalnog rasta kompleksnosti softvera i manjkavosti agilnih procesa da podrže njegov razvoj i održavanje. Novija istraživanja bave se mogućnošću proširenja agilnih procesa, sa komplementarnim elementima tradicionalnog razvoja, dokazujući na taj način da je njihova koegzistencija i integracija moguća. Najosetljivija pitanja proširenja agilnih procesa razvoja jesu arhitekturalna pitanja. Motiv za sprovođenje istraživanja u doktorskoj disertaciji je aktuelan problem iznalaženja ravnoteže između agilnog i tradicionalnog načina razvoja softverske arhitekture, tačnije, između primene eksplicitnih arhitekturalnih praksi i agilnosti razvojnog procesa. Empirijsko istraživanje sprovedeno je klasičnom varijantom Delfi metode, kroz tri iteracije istraživanja, na svrhovitom uzorku od 20 eksperata iz Srbije. Dobijeni rezultati empirijskog istraživanja obezbedili su odgovore na postavljena istraživačka pitanja. Glavni doprinos istraživanja je predloženi metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima, kojim se empirijski identifikovane značajne ekspicitne arhitekturalne prakse inkorporiraju na empirijski identifikovane kritične tačke agilnog procesa razvoja.
Datum prihvatanja teme od strane Senata, DP:	22.01.2015. godine
Datum odbrane, DO:	
Članovi komisije, KO: (ime i prezime / titula / zvanje / naziv organizacije / status)	predsednik: _____ član: _____ član: _____ član: _____ član: _____

University of Novi Sad
Faculty of Economics Subotica

KEY WORD DOCUMENTATION

Accession number, ANO:	
Identification number, INO	
Document type, DT:	Monograph documentation
Type of record, TR:	Textual printed material
Contents code, CC:	PhD thesis
Author, AU:	Mirjana Marić
Mentor, MN:	Pere Tumbas, PhD Full Professor
Title, TI:	A Methodological Framework for Developing the Software Architecture of Business Software in Agile Processes
Language of text, LT:	Serbian
Language of abstract, LA:	Serbian/English
Country of publication, CP:	Serbia
Locality of publication, LP:	Vojvodina
Publication year, PY:	2015
Publisher, PU:	Author's reprint
Publication place, PP:	9-11 Segedinski put Street, 24000 Subotica
Physical description, PD:	6/290/35/6/51/200/17
Scientific field, SF:	Economics – Business informatics
Scientific discipline, SD:	Analysis and design of information systems
Subject, Key words, SKW:	Software architecture, agile processes, framework
UC	
Holding data, HD:	Faculty Library, 9-11 Segedinski put Street, 24000 Subotica
Note, N :	None
Abstract, AB:	Software systems display a tendency of increasing decentralisation and heterogeneity, as well as an increasing need for connecting with other systems, in order to support the business processes of contemporary companies efficiently. They are characterised by a high frequency of changes in the business demands, weighted by the constantly growing complexity of their context, which is additionally complicated by the non-concurrence of the physical and logical areas of business operations. For this reason, the importance of the use of agile processes is recently recognised in the domain of developing complex systems,

	albeit initially intended for developing less complex software products. In accordance with the described tendencies, there is a gap in the software industry between a constant growth in the complexity of software and inadequacy of agile processes to support their development and maintenance. Recent research deals with the possibility of extending agile processes with complementary elements of traditional development, thus proving that their coexistence and integration are possible. The most sensitive issues of extending agile processes are the architectural issues. The motive for conducting the research in the doctoral dissertation is the relevant problem of finding a balance between the agile and traditional method of developing software architecture; more precisely, between the application of explicit architectural practices and the agility of the development process. The empirical research was conducted by the classical variant of the Delphi method, through three research iterations, on a convenience sample of 20 experts from Serbia. The obtained results of the empirical research have provided answers to the posed research questions. The main contribution of the research is the proposed methodological framework for developing the software architecture of business software in agile processes, whereby the empirically identified significant explicit architectural practices are incorporated onto the empirically identified critical points of the agile development process.
Accepted on Senate on, AS:	22.01.2015.
Defended, DE:	
Thesis Defend Board, DB:	president: _____ member: _____ member: _____ member: _____ member: _____

Abstrakt:

Softverski sistemi pokazuju tendenciju sve veće decentralizovanosti, heterogenosti, kao i sve veću potrebu za povezivanjem sa drugim sistemima, kako bi efikasno podržali poslovanje savremenih preduzeća. Njih karakteriše visoka frekventnost promena poslovnih zahteva, ponderisana konstantno rastućom kompleksnošću njihovog konteksta, koju dodatno komplikuje nepodudaranje fizičkog i logičkog područja poslovanja. Iz tog razloga se, u novije vreme, prepoznaje značaj upotrebe agilnih procesa u domenu razvoja kompleksnih sistema, iako inicijalno namenjenih za razvoj manje kompleksnih softverskih proizvoda. U skladu sa opisanim tendencijama, u industriji softvera postoji gap između stalnog rasta kompleksnosti softvera i manjkavosti agilnih procesa da podrže njegov razvoj i održavanje. Novija istraživanja bave se mogućnošću proširenja agilnih procesa, sa komplementarnim elementima tradicionalnog razvoja, dokazujući na taj način da je njihova koegzistencija i integracija moguća. Najosetljivija pitanja proširenja agilnih procesa razvoja jesu arhitekturalna pitanja. Motiv za sprovođenje istraživanja u doktorskoj disertaciji je aktuelan problem iznalaženja ravnoteže između agilnog i tradicionalnog načina razvoja softverske arhitekture, tačnije, između primene eksplicitnih arhitekturalnih praksi i agilnosti razvojnog procesa.

Empirijsko istraživanje sprovedeno je klasičnom varijantom Delfi metode, kroz tri iteracije istraživanja, na svrhovitom uzorku od 20 eksperata iz Srbije. Dobijeni rezultati empirijskog istraživanja obezbedili su odgovore na postavljena istraživačka pitanja. Glavni doprinos istraživanja je predloženi metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima, kojim se empirijski identifikovane značajne ekspicitne arhitekturalne prakse inkorporiraju na empirijski identifikovane kritične tačke agilnog procesa razvoja.

Abstract:

Software systems display a tendency of increasing decentralisation and heterogeneity, as well as an increasing need for connecting with other systems, in order to support the business processes of contemporary companies efficiently. They are characterised by a high frequency of changes in the business demands, weighted by the constantly growing complexity of their context, which is additionally complicated by the non-concurrence of the physical and logical areas of business operations. For this reason, the importance of the use of agile processes is recently recognised in the domain of developing complex systems, albeit initially intended for developing less complex software products. In accordance with the described tendencies, there is a gap in the software industry between a constant growth in the complexity of software and inadequacy of agile processes to support their development and maintenance. Recent research deals with the possibility of extending agile processes with complementary elements of traditional development, thus proving that their coexistence and integration are possible. The most sensitive issues of extending agile processes are the architectural issues.

The motive for conducting the research in the doctoral dissertation is the relevant problem of finding a balance between the agile and traditional method of developing software architecture; more precisely, between the application of explicit architectural practices and the agility of the development process.

The empirical research was conducted by the classical variant of the Delphi method, through three research iterations, on a convenience sample of 20 experts from Serbia. The obtained results of the empirical research have provided answers to the posed research questions. The main contribution of the research is the proposed methodological framework for developing the software architecture of business software in agile processes, whereby the empirically identified significant explicit architectural practices are incorporated onto the empirically identified critical points of the agile development process.

Doktorsku disertaciju posvećujem ćerki Mioni...

Sadržaj

1.

Uvod

Razvoj poslovnih softverskih rešenja agilnim procesima razvoja, primenjuje se od strane velikog broja preduzeća, u nastojanju da se smanje troškovi i poveća sposobnost prilagođavanja promenama, koje su rezultat dinamičkih uslova na tržištu (Ambler & Lines, 2013). Pojava agilnih procesa razvoja je značajno uticala na razvoj industrije softvera, ali je otvorila i brojna pitanja koja su u fokusu naučnih krugova. Jedno od aktuelnih pitanja je uloga softverske arhitekture i njen značaj u agilnim procesima (Abrahamsson, Babar, Kruchten, 2010; Babar, Brown, Mistrik, 2014), što je i predmet istraživanja doktorske disertacije.

Uvodnim delom rada detaljno će biti opisan predmet istraživanja, problem istraživanja, istraživačka pitanja i ciljevi, koji moraju biti ispunjeni nakon sprovođenja teorijskog i empirijskog istraživanja u radu. Takođe, ukratko će biti opisan i sadržaj predstojećih poglavlja disertacije.

1.1 Definisanje predmeta i problema istraživanja

Softverska arhitektura počinje da dobija na značaju tek tokom 90-ih godina iako se ideja, o njenom pozitivnom uticaju na proizvodnju softvera visokog kvaliteta, pojavila već 70-tih godina (Gacek, Abd-Allah, Clark, Boehm, 1995).

Parnas (1972) je prvi u razvoj softvera uveo pojam modularizacije, što znači dekomponovanje sistema na manje delove - module, koji se razvijaju nezavisno i mogu biti ponovo korišćeni ili zamenjeni drugim postojećim modulima. Pokazao je da je modularizacija sredstvo za povećanje fleksibilnosti i razumljivosti sistema, kao i njegovog lakšeg proširivanja i održavanja. Cilj dekomponovanja sistema je stvaranje labavih veza između njegovih visoko

kohezivnih delova, kako bi se efekat promena u pojedinim komponentama lokalizovao (Parnas, 1974).

Prve definicije softverske arhitekture dali su Perry i Wolf (1992), dok je danas u upotrebi veliki broj različitih formulacija softverske arhitekture i ne postoji opšteprihvaćena definicija (Rozanski & Woods, 2012; Bass, Clements, Kazman, 2012; Booch, Maksimchuk, Engle, Young, 2007; Gorton, 2006; Bosch, 2000; Shaw & Garlan, 1996; Gacek, Abd-Allah, Clark, Boehm, 1995; Perry & Wolf, 1992). Bass, Clements i Kazman (2012) softversku arhitekturu definišu kao skup struktura koje su sačinjene od softverskih komponenti, karakteristika komponenti, međusobnih odnosa komponenti i osobina tih odnosa.

U tradicionalnim metodologijama, razvoj softverske arhitekture je proces sa dobro definisanim životnim ciklusom. Danas su u upotrebi brojne metode za razvoj softverske arhitekture, a dve će biti detaljno predstavljene ovim istraživanjem:

1. Attribute Driven Design (u nastavku ADD), razvijena od strane grupe naučnika sa Instituta za softverski inžinjering, Carnegie Mellon univerziteta (Bass, Clements, Kazman, 2013)
2. The Rational Unified Process (u nastavku RUP) razvijen i komercijalizovan od strane Rational Software, sada IBM kompanije. RUP sadrži 4+1 View (u nastavku pogled) na kojem je zasnovan i formalni proces razvoja softverske arhitekture (Kruchten, 1995).

Odabrane metode će u radu biti opisane i analizirane, sa ciljem da se identifikuju korisne tradicionalne arhitekturalne prakse koje je potrebno inkorporirati u agilan proces razvoja u slučaju razvoja kompleksnog poslovnog softverskog rešenja. Tvorci ovih metoda saglasni su po pitanju tri opšte aktivnosti (Hofmeister et al., 2007):

1. Arhitekturalna analiza: ima za cilj definisanje i razumevanje problema koji se rešava, kao i identifikovanje arhitekturalno značajnih zahteva.
2. Arhitekturalna sinteza: podrazumeva dizajn arhitekture, uz prethodnu analizu i ocenu potencijialnih arhitekturalnih rešenja.
3. Arhitekturalna evaluacija: aktivnost koja se bavi ocenom arhitekturalnog rešenja.

Zagovornici stava o vitalnoj ulozi arhitekture u razvoju kvalitetnih kompleksnih sistema sumnjaju u skalabilnost agilnih procesa razvoja, iz razloga što ne obraćaju dovoljno pažnje na arhitekturu (Parsons, 2008; Nord & Tomayko, 2006; Ihme & Abrahamsson, 2005). Skloni su da agilne prakse ocene i kao amaterske i ograničene na domen veoma malih internet poslovnih rešenja.

Nasuprot tome, pristalice agilnog procesa razvoja unapred planirani razvoj arhitekture poistovećuju sa tradicionalnom strategijom Big Design Up Front (u nastavku, BDUF), što je u suprotnosti sa osnovnim agilnim principima i vrednostima, jer vodi masovnoj dokumentaciji i razvoju funkcija koje u implementaciji nisu potrebne (Babar, 2014). Zastupaju filozofiju zasnovanu na brzom razvoju, direktnoj i kontinuiranoj komunikaciji sa

stejkholderima, samoorganizaciji tima i kreativnosti svih učesnika, izbegavanjem balasta tradicionalnog razvoja: preterano planiranje, modelovanje, dokumentovanje (Abrahamsson et al., 2010). Agilni procesi nemaju tipične aktivnosti za razvoj softverske arhitekture (kao što su analiza, sinteza i evaluacija), jer one iziskuju dodatne troškove, a ne proizvode adekvatnu poslovnu vrednost za korisnika (Babar, 2014).

Agilnim procesima dominira razvoj funkcionalnosti softvera, naspram razvoja njegove arhitekture. Agilni procesi koncept metafore i tehniku izmene programskog koda (u nastavku refaktorisanje) smatraju adekvatnom zamenom za tradicionalan proces razvoja arhitekture. Arhitektura po njima nastaje postepeno, nakon svake iteracije, kao rezultat kontinuiranih izmena programskog koda, a ne iz neke unapred izgrađene strukture (Thapparambil, 2005; Beck, 2004). Ovakav stav u skladu je sa agilnim principom da se što ranije isporuči poslovna vrednost za korisnike.

Savremeni softversko-intenzivni sistemi pokazuju tendenciju sve veće decentralizovanosti, heterogenosti kao i sve veću potrebu za povezivanjem sa drugim sistemima, a klimu u kojoj se oni izvršavaju odlikuje dinamičnost. U skladu sa takvim tendencijama, može se reći da danas u industriji softvera postoji gap, između stalnog rasta kompleksnosti poslovnog softvera i manjkavosti agilnih procesa da podrže njegov razvoj i održavanje.

Poslednjih godina naučna zajednica prepoznaje da suprotstavljeni procesi razvoja (tradicionalni vs. agilni) imaju komplementarne uloge u razvoju softvera i bavi se proučavanjem mogućnosti njihovog koegzistiranja i prednostima njihove integracije (Ambler & Lines, 2013; Nord & Tomayko, 2006). Agilni procesi omogućuju preduzećima efikasnost, kvalitet i fleksibilnost u prihvatanju promena, ali je za razvoj kompleksnih softvera važno da se koriste i tradicionalne arhitekturalne prakse (Babar & Abrahamsson, 2008; Parsons, 2008; Nord & Tomayko, 2006; Ihme & Abrahamsson, 2005). Tehnika refaktorisanja, kao dominantna arhitekturalna praksa u agilnim procesima, može biti dovoljno uspešna, samo dok je dobar konceptualni dizajn softverske arhitekture. Na taj način se izbegava visok stepen refaktorisanja, koji uzrokuje visoke troškove razvoja sistema u kasnijim fazama, a ujedno se smanjuje i rizik nastanka erozije arhitekture (Stal, 2014; Kruchten, 2008; Ihme & Abrahamsson, 2005).

U skladu sa opisanim predmetom istraživanja, postavljen je problem istraživanja doktorske disertacije**:** *pronaći adekvatan balans između planiranog razvoja arhitekture i rane implementacije funkcionalnosti prilikom razvoja kompleksnog poslovnog softvera agilnim procesima.*

Cilj rešavanja postvaljenog istraživačkog problema je da se izbegne zamka preobimnog inicijalnog planiranja arhitekture na početku projekta (BDUF) i da se u što većoj meri uvaži jedan od principa agilnosti: „izbegavanje suvišnog razvoja funkcija koje neće biti implementirane“ (You Ain’t Gonna Need It, u nastavku YAGNI).

Istraživačka pitanja koja su proizašla iz definisanog problema su sledeća**:**

IP1. Da li je moguće ostvariti pozitivan uticaj na kvalitet poslovnog softverskog rešenja inkorporacijom tradicionalnih arhitekturalnih praksi u agilan proces razvoja?

Odgovor na postavljeno istraživačko pitanje dobiće se teorijskim izučavanjem metoda za razvoj softverske arhitekture i empirijskim utvrđivanjem značajnih tradicionalnih arhitekturalnih praksi koje je potrebno inkorporirati u agilne procese prilikom razvoja kompleksnih poslovnih softverskih rešenja.

IP2. Koja su kritična mesta u procesu agilnog razvoja koja je potrebno proširiti tradicionalnim arhitekturalnim praksama?

Odgovor na istraživačko pitanje dobiće se teorijskim izučavanjem tri agilna procesa razvoja: Extreme Programming (u nastavku XP) (Beck, 2004), Scrum (Schwaber, 2004) i Lean (Poppendieck & Poppendieck, 2003; Womack, Jones, Roos, 1990) i empirijskim utvrđivanjem kritičnih mesta na koja je potrebno inkorporirati značajne tradicionalne arhitekturalne prakse.

IP3. Kako izabrati i integrisati tradicionalne arhitekturalne prakse u agilni proces razvoja poslovnog softvera?

Projekat razvoja kompleksnog poslovnog softvera ima jedinstvene arhitekturalne prioritete, koji utiču na nivo značajnosti pojedinih tradicionalnih arhitekturalnih praksi koje treba inkorporirati u agilni proces razvoja. Stoga će karakteristike ovog tipa projekta predstavljati težinske faktore u izboru tradicionalnih arhitekturalnih praksi. Drugi, jednako važan izazov ovog segmenta istraživanja jeste prilagođavanje i harmonizacija seta identifikovanih tradicionalnih arhitekturalnih praksi, pre njihove inkorporacije u agilni proces razvoja, kako se ne bi narušili osnovni principi agilnosti. Odgovor na ovo istraživačko pitanje biće metodološko-radni okvir za inkorporiranje značajnih tradicionalnih arhitekturalnih praksi na kritične tačke u agilnom procesu razvoja.

1.2 Ciljevi istraživanja

C1. Identifikovane tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih kompleksnih softverskih rešenja.
C2. Identifikovana kritična mesta u agilnom procesu razvoja pogodna za primenu tradicionalnih arhitekturalnih praksi.
C3. Razvijen metodološko-radni okvir za inkorporaciju tradicionalnih arhitekturalnih praksi u agilni proces razvoja.
C4. Identifikovani budući pravci istraživanja.

Hipoteza: *Postoji značajna međuzavisnost inkorporiranja tradicionalnih arhitekturalnih praksi i razvoja visoko kvalitetnog poslovnog softvera u agilnom procesu razvoja.*

1.3 Struktura rada

Istraživanje doktorske disertacije organizovano je u sledećih šest logičkih celina (poglavlja): Uvod; Metodologija istraživanja; Teorijska istraživanja agilnih procesa razvoja i metoda za razvoj softverske arhitekture; Empirijsko istraživanje razvoja softverske arhitekture i agilnih procesa; Rezultati empirijskih istraživanja, interpretacija i evaluacija razvijenog metodološko-radnog okvira; Zaključna razmatranja i budući pravci istraživanja.

Prvo poglavlje rada predstavlja uvodni deo u kojem je sadržan opis predmeta i problema istraživanja doktorske disertacije. Aktuelnost i značaj predmeta istraživanja sistematski je pokazana kroz navode stavova i empirijske rezultate vodećih naučnika u oblasti iz preko četrdeset bibliografskih izvora. Pored toga, opisana su postavljena istraživačka pitanja, definisani ciljevi i hipoteza istraživanja.

U drugom poglavlju rada predstavljen je dizajn istraživanja, uz detaljan opis svih istraživačkih aktivnosti, kao i metoda i tehnika putem koji su iste realizovane. Fokus poglavlja je na opisu načina sprovođenja i dobijenim rezultatima sistematskog pregleda literature. Takođe, detaljno je opisan i način realizacije empirijskog istraživanja putem Delfi metode.

Treće poglavlje počinje predstavljanjem osnovnih koncepata i procesa razvoja softverske arhitekture. Nakon kratkog pregleda brojnih metoda za razvoj softverske arhitekture, sprovodi se analiza odabranih metoda: RUP i ADD. Fokus je na opisu i analizi njihovih arhitekturalnih aktivnosti i artifakata, koji bi potencijalno mogli biti inkorporirani u agilan proces razvoja kompleksnih poslovnih softverskih rešenja. Sprovedene analize doprinos su dale u ostvarivanju prvog definisanog istraživačkog cilja.

Poglavlje sadrži i analizu tri agilna procesa razvoja: XP, Scrum i Lean. Svrha njihovog detaljnog razmatranja bila je identifikovanje kritičnih mesta u agilnom procesu razvoja, koja su pogodna za primenu tradicionalnih arhitekturalnih praksi, što je u skladu sa drugim istraživačkim ciljem rada. Poglavlje sadrži i opis rezultata sprovedenog sistematskog pregleda literature.

Četvrto poglavlje opisuje detaljno sve realizovane istraživačke aktivnosti, po krugovima istraživanja klasične varijante Delfi metode. To podrazumeva opis uzorkovanja učesnika istraživanja, razvoj instrumenata istraživanja, način prikupljanja, pripreme i analize podataka i dobijene rezultate. Prikaz i interpretacija rezultata vrši se po krugovima istraživanja, kako bi se

sagledao celokupan tok procesa istraživanja. U fokusu je i diskusija rezultata na osnovu kojih je donešena odluka da ne postoji potreba za pokretanjem nove iteracije istraživanja.

Peto poglavlje sadrži prikaz i interpretaciju rezultata sprovedenog istraživanja, prema postavljenim pitanjima i ciljevima istraživanja. U kontekstu trećeg istraživačkog cilja prikazan je metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima, kao i njegova procena putem evaluacione studije. Poglavlje završava elaboracijom ograničenja istraživanja.

Zaključna razmatranja i opis budućih pravaca istraživanja dati su u **šestom poglavlju rada**.

Pored navedenih poglavlja i detaljnog spiska **literature**, rad sadrži i sledeće **priloge:**

Prilog 1: Pozivno pismo ekspertima za učešće u istraživanju.
Prilog 2: Indeksi sadržajne valjanosti pitanja, upitnika prvog kruga istraživanja.
Prilog 3: Dopis ekspertima za ocenu značajnosti pitanja polustrukturiranog intervjua.
Prilog 4: Konačan set pitanja upitnika za sprovođenje prvog kruga istraživanja, polustrukturiranim intervjuom.
Prilog 5: Instrukcije za ocenjivanje sadržajne valjanosti pitanja upitnika, instrumenta drugog kruga istraživanja.
Prilog 6: Indeksi sadržajne valjanosti pitanja upitnika za drugi krug istraživanja.
Prilog 7: Pozivno pismo ekspertima za drugi krug istraživanja.
Prilog 8: Instrukcije za popunjavanje upitnika drugog kruga istraživanja.
Prilog 9: Deo rezultata kvantitativne analize podataka drugog kruga istraživanja (primer dobijenih rezultata za 3 stavke 29. pitanja iz upitnika).
Prilog 10: Pozivno pismo ekspertima za učešće u završnom krugu istraživanja.
Prilog 11: Instrukcije za popunjavanje upitnika u završnom krugu istraživanja.
Prilog 12: Obrasci po kojima su se sprovodile procedure za kvantitativnu analizu podataka.
Prilog 13: Deo rezultata kvantitativne analize podataka drugog kruga istraživanja (Cohen-ov kappa koeficijent).
Prilog 14: Deo rezultata kvantitativne analize podataka trećeg kruga istraživanja.
Prilog 15: Deo rezultata kvantitativne analize podataka u trećem krugu istraživanja - vrednosti McNemar-Bowker-ovog testa za ispitanika 20.
Prilog 16: Artifakti metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima.
Prilog 17: Dopis ekspertima za ocenu postavljenog metodološko-radnog okvira.

Napomene:

- Imena stranih autora su pisana u originalu.
- U radu su pojedine reči napisane na engleskom jeziku, zbog nemogućnosti njihovog adekvatnog prevoda na srpski jezik.

2.

Metodologija istraživanja

Dizajn istraživanja doktorske disertacije rezultat je modifikacija okvira postavljenog od strane grupe autora Hevner i dr. (2004). Redosled istraživačkih aktivnosti i korišćene metode i tehnike prikazane su na slici 2.1 i ukratko će biti objašnjeni narednim tekstom.

Prezentovani problem i pitanja istraživanja u uvodnom delu disertacije, rezultat su faze *identifikovanje predmeta istraživanja,* u kojoj je sprovedena analiza literature i empirijski uvid u razvoj kompleksnih sistema agilnim procesima, tehnikom workshop-a.

Utvrđeni vladajući stavovi u literaturi rezultat su faze *teoretsko istraživanje*, realizovane pomoću metode sistematskog pregleda literature, prema preporukama datim od strane Kitchenham (2004).

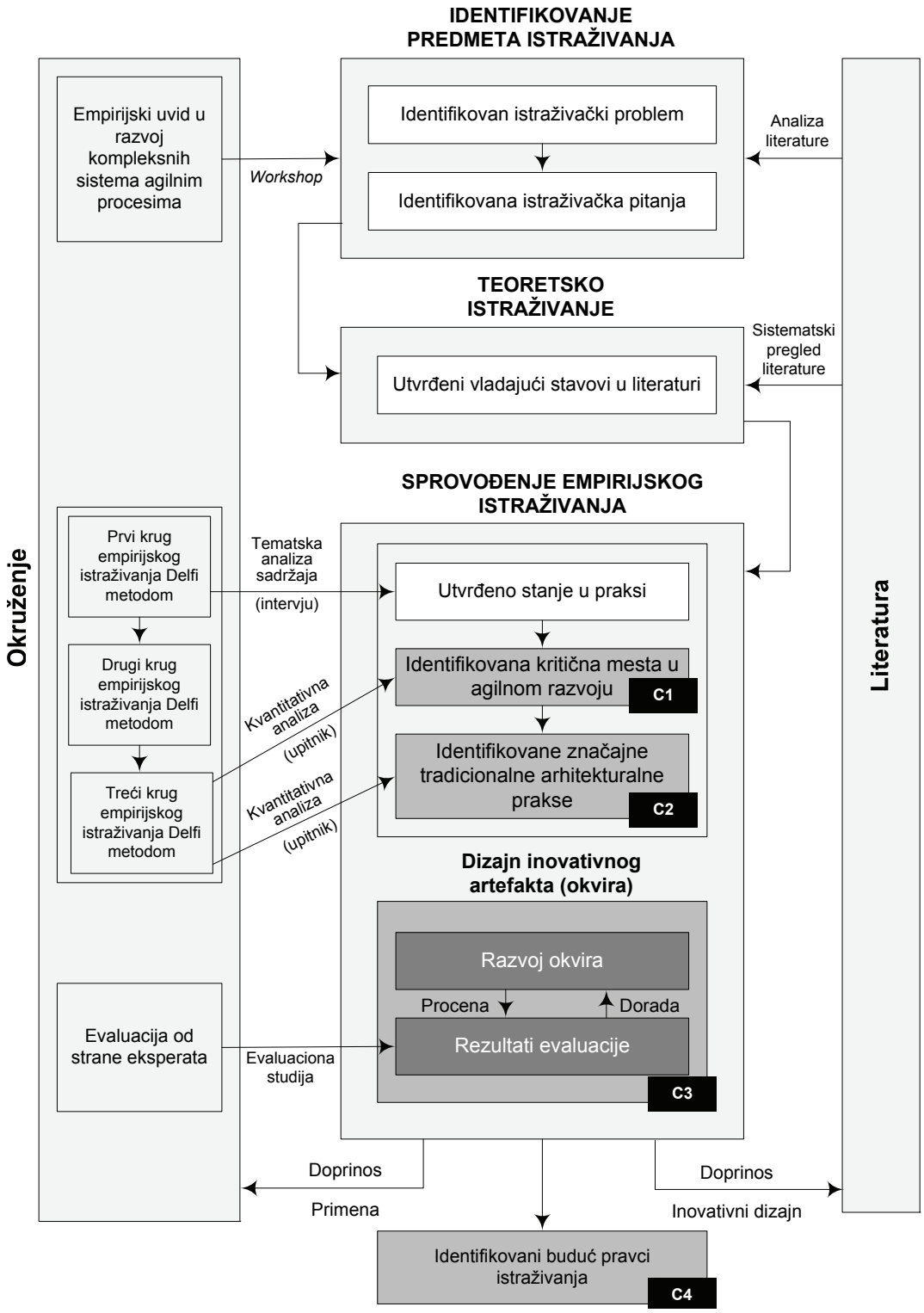
Poslednja faza istraživanja obuhvatala je set istraživačkih aktivnosti, grupisanih u okviru logičke celine *sprovođenje empirijskog istraživanja*. Empirijsko istraživanje sprovedeno je putem klasične varijante Delfi metode, koja je realizovana kroz tri kruga istraživanja.

Utvrđeno stanje u praksi rezultat je prvog kruga istraživanja Delfi metodom. Korišćeni instrument istraživanja bio je intervju, a metoda za analizu prikupljenih podataka - tematska analiza sadržaja. Kodiranje podataka intervjua i tematska analiza sadržaja realizovani su prema preporukama autora Miles i Huberman (1994).

Drugi krug istraživanja Delfi metodom realizovan je putem upitnika, dok je nad prikupljenim podacima sprovedena kvantitativna analiza, u softveru Statistical Package for the Social Scientists (u nastavku SPSS). Deo dobijenih rezultata direktno je doprineo ostvarivanju prvog i drugog istraživačkog cilja (C1 i C2). Treći krug istraživanja Delfi metodom realizovan je takođe putem upitnika, a nad prikupljenim podacima sprovedena je kvantitativna analiza u softveru SPSS. Dobijeni rezultati omogućili su ostvarivanje prvog i drugog istraživačkog cilja (C1 i C2).

Aktivnost *dizajn metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima* podrazumevala je postavku metodološko-radnog okvira i njegovu evaluaciju, putem evaluacione studije. Rezultat ove aktivnosti omogućio je realizaciju trećeg istraživačkog cilja (C3).

Rezultati faze *sprovođenje empirijskog istraživanja* omogućili su identifikovanje budućih pravaca istraživanja, što ujedno predstavlja četvrti istraživački cilj (C4).



Slika 2.1 Dizajn istraživanja
Izvor: Prilagođeno prema Hevner i dr. (2004)

2.1 Teoretsko istraživanje

Teoretsko istraživanje sprovedeno je primenom metode sistematske analize literature, sa ciljem da se utvrdi pregled vladajućih stavova u postojećoj literaturi koja je iz domena predmeta istraživanja doktorske disertacije. Proces sistematske analize literature sprovodio se prema preporukama Kitchenham (2004) i podrazumevao je tri glavne faze: planiranje analize, sprovođenje analize i izveštavanje o nalazima analize. Svaka od navedenih faza ukratko je opisana narednim tekstom:

- Faza planiranja analize imala je u fokusu razvoj okvira za sprovođenje analize. Cilj definisanog okvira bio je da smanji rizik od sprovođenja analize literature prema slobodnom nahođenju i očekivanjima istraživača. Okvirom su definisana sledeća pitanja:
  - Cilj analize i istraživačka pitanja
  - Strategija pretrage
  - Kriterijumi za uključivanje/isključivanje naučnog materijala
  - Kriterijumi za evaluaciju naučnog materijala
  - Strategija ekstrahovanja podataka i strategija sintetizovanja ekstrahovanih podataka

- Faza sprovođenja analize, prema definisanom okviru iz prethodne faze, podrazumevala je realizaciju sledećih koraka:
  - Pretragu primarnog naučnog materijala
  - Selekciju primarnog naučnog materijala koji zadovoljava definisane
  - Kriterijume uključivanja/isključivanja
  - Procenu kvaliteta naučnog materijala
  - Ekstrahovanje podataka
  - Sintezu podataka

- Faza izveštavanja o nalazima sprovedene analize, podrazumevala je davanje odgovora na postavljeno istraživačko pitanje, opisom dobijenih rezultata.

2.1.1 Postavka okvira za sprovođenje analize literature

Faza planiranja analize otpočela je definisanjem cilja teoretskog istraživanja. *Cilj* sistematskog pregleda literature bio je izučavanje postojećih empirijskih nalaza koji se odnose na pitanja razvoja arhitekture kompleksnih sistema u agilnim procesima, kao i identifikovanje problema i izazova u integraciji agilnih i arhitekturalnih praksi.

Realizacija definisanog cilja istraživanja moguća je davanjem odgovora na sledeće istraživačko pitanje:

IP1. Koji su rezultati dosadašnjih empirijskih istraživanja po pitanju integracije agilnih i arhitekturalnih praksi?

Drugi korak u postavci okvira za analizu literature podrazumevao je definisanje formalne strategije pretrage, kako bi se obuhvatio samo onaj naučni materijal koji je od značaja za dobijanje odgovora na postavljeno istraživačko pitanje. Strategija je podrazumevala definisanje izvora pretrage tj. elektronskih baza podataka (tabela 2.1) i preuzimanje samo onog naučnog materijala, koji je predstavljao primarne studije. Paralelno sa sprovođenjem evaluacije primarnog naučnog materijala, sprovodila se i sekundarna pretraga. Sekundarna pretraga podrazumevala je pretragu naučnog materijala, po referencama relevantnih primarnih studija.

Za sistematski pregled literature korištišćen je servis Konzorcijuma biblioteka Srbije za objedinjenu nabavku (Kobson), gde je slobodan pristup velikom broju naučnog materijala.

Tabela 2.1 Odabrane elektronske baze podataka

Izvori podataka
IEEE Xplore
ScienceDirect
ACM Digital library

Strategija pretrage uključivala je i sledeće definisane termine pretrage:

Agile Software Architecture/ Agile methods (methodologies) and Architecture/ Agility and Architecture.

Treći korak u postavci okvira za analizu literature bio je definisanje kriterijuma za uključivanje/isključivanje naučnog materijala:

- Kriterijumi uključivanja
  - Vremenska dimenzija: pretraga naučnog materijala objavljenog od 2000. godine do 2014. godine
  - Tip studije: naučni materijal napisan na engleskom jeziku, koji zadovoljava definisane ključne reči pretrage ili po naslovu ili u apstraktu. Naučni materijal uključivao je recenzirane naučne i stručne radove objavljene u časopisima i zbornicima radova sa workshop-ova i konferencija.

- Kriterijumi isključivanja: radovi koji za termin agilnosti ne vezuju agilne metodologije; radovi koji su fokusirani samo na softversku arhitekturu bez konteksta agilnog okruženja; radovi koji su isključivo fokusirani na određen tip softverske arhitekture (npr. SOA); radovi koji nemaju empirijsko istraživanje ili predložen pristup/metod; radovi koji su zasnovani isključivo na mišljenju autora.

Četvrti korak podrazumevao je definisanje kriterijuma za sprovođenje evaluacije kvaliteta naučnog materijala, koji je zadovoljio definisane kriterijume uključivanja. U radu su se primenjivali kriterijumi kvaliteta prikazani u tabeli 2.2, a definisani su na osnovu opštih kriterijuma datih od strane autora Dyba i Dingsoyr (2008). Definisani kriterijumi odnose se na tri ključna aspekta: strogost, kredibilitet i relevantnost.

Tabela 2.2 Kriterijumi kvaliteta za evaluaciju naučnog materijala

1.	Da li postoji jasan cilj istraživanja?	Da/Ne
2.	Da li postoji adekvatan opis konteksta istraživanja?	Da/Ne
3.	Da li postoji adekvatan opis doprinosa metode ili pristupa?	Da/Ne
4.	Da li je predloženi pristup/metod evaluiran?	Da/Ne
5.	Da li postoje jasni podaci o rezultatitama istraživanja ?	Da/Ne
6.	Da li studija izveštava o empirijskom istraživanju?	Da/Ne
7.	Da li je studija značajna za praksu ili istraživanje?	Da/Ne

Izvor: Prilagođeno prema (Dyba & Dingsoyr, 2008)

Poslednji korak u postavci okvira za analizu postojeće literature, bio je definisanje strategije ekstrahovanja podataka i strategije sinteze ekstrahovanih podataka. Proces ekstrahovanja podrazumevao je višestruko iščitavanja relevantnih radova i identifikovanje podataka koji su bitni za davanje odgovora na postavljeno istraživačko pitanje.

Strategija ekstrahovanja podrazumevala je sprovođenje procesa kodiranja teksta, tj. identifikovanje tema, sublimaciju sličnih tema u koncepte i definisanje hijerarhije kategorija. Upravljanje ključnim konceptima, nalazima i zaključcima iz radova, podrazumevalo je korišćenje softvera NVivo. Strategija sinteze podataka odnosila se na povezivanje i komparaciju identifikovanih tema, koncepata i kategorija.

2.1.2 Sprovođenje analize literature

Faza sprovođenja analize literature, prema postavljenom okviru, otpočela je pretragom naučnog materijala u okviru definisanih elektronskih baza i prema utvrđenim ključnim rečima. Prikaz ukupnog broja pogodaka za svaku elektronsku bazu ponaosob, dat je u tabeli 2.3.

Prvi korak u analizi dobijenih rezultata bio je fokusiran na analizu naslova i apstrakata. Radovi koji su po naslovu i apstraktu bili povezani sa definisanim istraživačkim pitanjem, sačuvani su u excel tabeli, uz opis njihovih osnovnih karakteristika (naslov, autor, izvor, apstrakt).

U drugom koraku, analizirana je napravljena excel tabela, sa ciljem da se utvrde duplikati radova. Nakon eliminisanja 59 radova duplikata, nad preostalih 69 radova sprovedena je detaljna analiza. Proces detaljne analize ogledao se u njihovom višekratnom iščitavanju u

celosti, uz primenu definisanih kriterijuma za evaluaciju njihovog kvaliteta. Sprovedenom evaluacijom kvaliteta eliminisano je 43 rada, te je broj relevantnih radova bio 26.
Parelelno sa evaluacijom kvaliteta ovih radova, sprovodila se i sekundarna pretraga, po referencama koje se nalaze u njima. Proces je rezultirao sa 33 dodatna rada, od kojih je nakon procene njihovog kvaliteta, svega 8 radova identifikovano kao relevantno za istraživanje.

Sprovedeno istraživanje rezultiralo je konačnim brojem od 34 relevantna rada, od kojih je 26 rezultat primarne pretrage, a 8 rezultat sekundarne.

Elektronske verzije konačnog skupa relevantnih radova (34), arhivirane su u softver za upravljanje referencama - Mendeley, radi njihovog lakšeg navođenja u fazi opisa rezultata sprovedenog istraživanja. Takođe, sačuvane su i u softveru NVivo, kako bi se lakše upravljalo ključnim konceptima, nalazima i zaključcima koje su radovi sadržavali.

Strategija ekstrahovanja podataka podrazumevala je sprovođenje procesa kodiranja teksta, tj. identifikovanje tema, sublimaciju sličnih tema u koncepte i identifikovanje ključnih kategorija. Ovim je bio završen proces sistematizacije ključnih nalaza, čime su stvoreni uslovi za realizaciju procesa sinteze podataka. Sinteza podataka realizovana je povezivanjem i komparacijom identifikovanih tema, koncepata i kategorija.

Tabela 2.3 Rezultati pretrage elektronskih baza podataka

Izvori podataka	Broj pogodaka po ključnim rečima	Broj radova uključenih u dalju analizu	Broj isključenih radova
IEEE Xplore	701	45	656
ScienceDirect	46	12	34
ACM Digital library	237	12	225
Ukupno	984	69	915

Faza izveštavanja o nalazima sprovedene analize, podrazumevala je davanje odgovora na postavljeno istraživačko pitanje, opisom dobijenih rezultata. Opis je dat u delu rada *3.3.5 Softverska arhitektura i agilni procesi razvoja*, koji je sastavni deo trećeg poglavlja rada: *Teorijska istraživanja agilnih procesa razvoja i metoda za razvoj softverske arhitekture.*

2.2 Sprovođenje empirijskog istraživanja

Empirijska istraživanja klasifikovana su u dve osnovne grupe: kvalitativna i kvantitativna. Njihova suštinska razlika vazana je za podatke, koji kod kvalitativnih istraživanja potiču iz reči, dok kod kvantitativnih dolaze iz brojeva i podrazumevaju kvantifikaciju i merenje (Punch, 2005).

Rešavanje identifikovanog problema i ostvarivanje postavljenih ciljeva doktorske disertacije, pripada mešovitom tipu istraživanja, što je podrazumevalo primenu i kvantitativnih i kvalitativnih metoda prikupljanja i analiziranja podataka. Creswell (2009) navodi: "Istraživanje pomešanim metodama je dizajn istraživanja u kojem istraživač prikuplja, analizira i meša (integriše ili povezuje) kvantitativne i kvalitativne podatke u okviru iste studije ili u višefaznom programu istraživanja". Pored navedenog, istraživanje doktorske disertacije ima svojstva i metodološko razvojnih istraživanja, koja imaju za cilj razvoj/unapređivanje procesa, praksi, proizvoda i sl.

Istraživanje u radu sprovedeno je primenom klasične varijante Delfi metode, koja uključuje i kvalitativne i kvantitativne procedure prikupljanja i analize podataka. Postupak Delfi metode je relativno složen i dug, jer se definisani koraci tehnike ponavljaju ili prema unapred utvrđenom broju iteracija (najčešće tri) ili do ostvarivanja potrebnog nivoa saglasnosti eksperata (koji je u literaturi najčešće 70%) ili dok se ne dokaže da ni jedan ekspert više ne menja svoja rešenja, već ostaje pri onima koje je već dao (Keeney, Hasson, McKenna, 2011; Linstone & Turoff, 1975; Helmer & Rescher, 1959).

Priroda problema istraživanja nalagala je da se vrši nameran odabir jedinica uzorka ($n \geq 20$). Stoga uzorak sačinjavaju samo eksperti koji mogu ponuditi najbolje informacije za ostvarivanje postavljenih ciljeva istraživanja, tj. osobe koje imaju iskustva u agilnom razvoju i razvoju softverske arhitekture. Sva tri kruga istraživanja sprovodila su se nad istim panelom eksperata, u okviru referentnih preduzeća iz ICT sektora u Srbiji. Pored empirijskog uzorkovanja realizovano je i automatsko, softversko bootstrap uzorkovanje na 1000 replikacija.

Na slici 2.1, prikazana faza *sprovođenje empirijskog istraživanja*, obuhvata set istraživačkih aktivnosti realizovanih kroz tri kruga istraživanja Delfi metodom.

Prvi krug empirijskog istraživanja Delfi metodom predstavljao je kvalitativnu komponentu tj. prikupljanje i analizu kvalitativnih podataka, radi eksploracije fenomena koji se proučava. Prikupljanje empirijskih podataka vršilo se metodom polustrukturiranog intervjua. Polustrukturirani intervju je bio najprikladniji metod, jer su se iz seta otvorenih pitanja dobile značajne informacija o mišljenjima, uverenjima i stavovima ispitanika. Skup unapred pripremljenih inicijalnih pitanja bio je zasnovan na analizi referentne literature. Instrument istraživanja bio je podvrgnut i evaluaciji od strane grupe eksperata. Intervjui su se sprovodili "licem u lice" i bili su snimani. Snimljeni intervjui su potom bili transkriptovani i potvrđeni od strane ispitanika.

Rezultat prvog kruga istraživanja (prikazan na slici 2.1) nazvan je *utvrđeno stanje u praksi,* a podrazumevao je primenu kvalitativne metode za analizu intervjua - tematsku analizu sadržaja. Kodiranje podataka intervjua i tematska analiza sadržaja realizovani su u softveru NVivo, prema preporukama autora Miles i Huberman (1994).

Drugi krug empirijskog istraživanja Delfi metodom predstavljao je kvantitativnu komponentu istraživanja, te je iz tog razloga realizovan putem upitnika. Instrument istraživanja drugog kruga razvijen je na osnovu literature i rezultata empirijskog istraživanja iz prvog kruga. Upitnik je sadržavao skale procenjivanja, po uzoru na Likertovu, „čekliste“ i „čektabele“. Instrument je bio podvrgnut i evaluaciji od strane grupe eksperata. U cilju njegove lakše distribucije bio je napravljen u elektronskoj formi (e-upitnik). Nad prikupljenim podacima iz drugog kruga, sprovodila se kvantitativna analiza primenom sledećih tehnika: izračunavanje mera centralne tendencije/lokacije (aritmetička sredina/medijana), izračunavanje mera varijabilnosti (standardna devijacija/interkvartilni raspon), Efron-ovo ponavljano uzorkovanje, Cohen-ov kappa koeficijent, Cronbach-ov koeficijent α. Deo dobijenih rezultata (slika 2.1: identifikovana kritična mesta u agilnom razvoju i identifikovane značajne arhitekturalne prakse) ovog kruga istraživanja doprineo je realizaciji prvog i drugog istraživačkog cilja (C1 i C2).

Treći krug empirijskog istraživanja Delfi metodom realizovan je takođe putem upitnika. Instrument istraživanja razvijen je na osnovu rezultata iz drugog kruga i uključivao je samo ona pitanja iz drugog kruga, za koja nije postignut nivo saglasnosti eksperata od 70%. Nad prikupljenim podacima iz trećeg kruga sprovodila se kvantitativna analiza primenom sledećih tehnika: izračunavanje mera centralne tendencije/lokacije (aritmetička sredina/medijana), izračunavanje mera varijabilnosti (standardna devijacija/interkvartilni raspon), Efron-ovo ponavljano uzorkovanje, Cohen-ov kappa koeficijent, pokazatelj individualne stabilnosti između dve uzastopne Delfi iteracije (Chaffin–Talley-ev indeks individualne stabilnosti), McNemar-ov test značajnosti promene i McNemar-Bowker-ov test. Dobijeni rezultati, označeni na slici 2.1 kao *identifikovana kritična mesta u agilnom razvoju i identifikovane značajne arhitekturalne prakse,* omogućili su ostvarivanje prvog i drugog istraživačkog cilja (C1 i C2).

Na slici 2.1 vidi se da je poslednja istraživačka aktivnost, u okviru faze *sprovođenje empirijskog istraživanja,* bila aktivnost *dizajn metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima.* Ova istraživačka aktivnost podrazumevala je postavku metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima, kao i njegovu evaluaciju putem evaluacione studije. Za sprovođenje evaluacione studije odabran je „svrhoviti“ tip uzorka, odgovarajućih eksperata iz Srbije i inostranstva (zemlje Evropske Unije; $n \geq 8$). Završetkom ove aktivnosti realizovan je treći istraživački cilj (slika 2.1, C3).

Rezultati faze *sprovođenje empirijskog istraživanja* omogućili su identifikovanje budućih pravaca istraživanja, čime je ujedno realizovan četvrti istraživački cilj (slika 2.1, C4).

3.

Teorijska istraživanja agilnih procesa razvoja i metoda za razvoj softverske arhitekture

Prvi deo poglavlja sadrži prikaz osnovnih koncepata i metoda za razvoj softverske arhitekture, kao i detaljnu analizu procesa razvoja arhitekture upotrebom RUP-a i ADD metode za razvoj softverske arhitekture. Fokus je na opisu i analizi arhitekturalnih aktivnosti i artifakata, koji bi mogli biti inkorporirani u agilan proces razvoja kompleksnih poslovnih softverskih rešenja. Rezultati sprovedene analize biće korišćeni za izradu instrumenta empirijskog istraživanja, kao i za davanje odgovora na prvo istraživačko pitanje.

Drugi deo ovog poglavlja sadrži prikaz i komparaciju najzastupljenijih agilnih procesa razvoja u praksi: XP, Scrum i Lean. Cilj analize ovih agilnih procesa je identifikovanje kritičnih mesta u agilnom razvoju, koja su pogodna za primenu značajnih tradicionalnih arhitekturalnih praksi. Rezultati sprovedene analize koristiće se za izradu instrumenta empirijskog istraživanja, kao i za davanje odgovora na drugo istraživačko pitanje.

Poglavlje sadrži i opis rezultata sprovedenog sistematskog pregleda literature, koji su usko vezani za definisani istraživački problem: razvoj softverske arhitekture kompleksnih softverskih rešenja u agilnim procesima.

3.1 Koncepti i proces razvoja softverske arhitekture

Softverska arhitektura, kao koncept, egzistira od ranih sedamdesetih godina u radovima Edsger Dijkstre i Davida Parnasa, koji su još tada prepoznali značaj strukture sisitema za njegov kvalitet. Parnas (1972) je prvi u tumačenju procesa razvoja softvera uveo pojam modularizacije, što je značilo dekomponovanje budućeg sistema na manje delove, module, koji se razvijaju nezavisno, mogu biti ponovo korišćeni kao i zamenjeni drugim postojećim modulima. Pokazao je da je modularizacija sredstvo za povećanje fleksibilnosti i razumljivosti sistema i ujedno njegovog lakšeg proširivanja i održavanja (Parnas, 1974). Prve definicije softverske arhitekture dali su Perry i Wolf (1992). Danas su u upotrebi brojne definicije, ali ni jedna od njih nije opšte prihvaćena (Rozanski & Woods, 2012; Bass, Clements, Kazman, 2012; Booch, Maksimchuk, Engle, Young, 2007; Gorton, 2006; Bosch, 2000; Shaw & Garlan, 1996; Gacek, Abd-Allah, Clark, Boehm, 1995; Perry & Wolf, 1992).

Perry i Wolf (1992) softversku arhitekturu definišu kao model, koji je sačinjen od elemenata, forme i obrazloženja. Elementi predstavljalju proces obrade, podatke ili vezivne elemente sa određenom formom (osobinama i odnosima), dok obrazloženje predstavlja razloge za izbor datih elemenata i forme. Bass, Clements i Kazman (2012) softversku arhitekturu definišu kao skup struktura koje su sačinjene od softverskih komponenti, međusobnih odnosa komponenti i osobina tih odnosa. Cilj dekomponovanja sistema je stvaranje labavih veza između njegovih visoko kohezivnih delova, kako bi se efekat promena u pojedinim komponentama lokalizovao.

Arhitektura predstavlja samo jedan aspekt dizajna, jer je fokusirana na njegove određene specifičnosti. Garlan i Shaw (1994) sugerišu da je softverska arhitektura nivo dizajna koji se bavi pitanjima celokupne strukture sistema, podrazumevajući sledeća strukturalna pitanja: grubu organizaciju i globalnu kontrolnu strukturu; protokole za komunikaciju, sinhronizaciju i pristup podacima; dodeljivanje funkcionalnosti elementima dizajna; fizičku distribuciju; kompoziciju elemenata dizajna; skaliranje i performanse; izbor između alternativnih rešenja dizajna.

Arhitektura, međutim, predstavlja mnogo širi koncept od samog definisanja strukture sistema, jer obuhvata odnos sistema i njegovog okruženja - korisničkog i razvojnog, uz uvažavanje ekonomskih i sociooloških aspekata (ISO/IEC, 2011).

I pored različitih definicija softverske arhitekture, naučna zajednica je saglasna u pogledu njena tri ključna elementa: komponente, konektori i konfiguracija. Komponente se smatraju crnim kutijama sa visokim nivoom enkapsulacije i abstrakcije i definisanim interfejsom, putem kojeg međusobna komuniciraju. Konektori predstavljaju model interakcije između komponenti i set pravila koji regulišu ove interakcije (Medvidovic & Taylor, 2000). Svaki konektor ima specificiran protokol, koji definiše njegova svojstva. Svojstva uključuju pravila o vrstama interfejsa za koje mogu da posreduju, osobinama interakcija, pravila o redosledu

kojim se stvari odvijaju i preuzete obaveze date interakcije (npr. naručivanje i sl.) (Shaw & Garlan, 1996). Arhitekturalne konfiguracije ili topologije predstavljaju skupove povezanih komponenti i konektora koji izgrađuju arhitekturalnu strukturu. Konfiguracije omogućavaju odgovarajuću komunikaciju komponenti putem konektora koji ih povezuju. Tačke putem kojih komponente mogu međusobno da komuniciraju zovu se portovi. Portovi "objavljuju" ponašanja, koja date komponente pružaju ili zahtevaju, što znači da oni "objavljuju" interfejs ili njegov deo. Interfejs predviđa usluge, koje data komponenta zahteva ili pruža drugim komponentama (Medvidovic & Taylor, 2000).

Devedesetih godina prošloga veka učinjeni su prvi koraci definisanja ove discipline, a tek 2007. godine prihvaćen je prvi standard: Practice for Architecture Description of Software-Intensive Systems std. 1471 (ISO/IEC, 2007), a potom i ISO/IEC/IEEE std. 42010 (ISO/IEC, 2011). Ovi standardi obezbeđuju osnove za efikasniji razvoj, analizu i održavanje softverske arhitekture. Pružaju definicije osnovnih termina i meta model za opis arhitekture; uputstva za razložni izbor arhitekturalnog rešenja; uputstva za identifikovanje arhitekturalno značajnih zahteva stejkholdera; uputstva za razvoj arhitekturalnog rešenja putem seta pogleda, u kojima se opisuju različiti aspekti arhitekture. Najveći doprinos razvoju ove discipline pružaju Istraživački institut Carnegie Mellon univerziteta i Istraživački institut Irvine, California univerziteta , kao i brojne međunarodne konferencije.

Razvoj softverske arhitekture podrazumeva širok spektar zadataka: analizu i opis svojstava sistema na visokom nivou apstrakcije, validaciju softverskih zahteva, procenu troškova procesa razvoja i održavanja, ponovnu upotrebu softvera, uspostavljanje osnova i smernica za detaljan dizajn sistema. Iz tog razloga brojni autori smatraju da arhitektura ima vitalnu ulogu u razvoju velikih/kompleksnih softverskih sistema (Rozanski & Woods, 2012; Bass, Clements, Kazman, 2012; Booch, Maksimchuk, Engle, Young, 2007; Gorton, 2006; Kruchten, Obbink, Staord, 2006; Bosch, 2000; Shaw & Garlan, 1996; Gacek, Abd-Allah, Clark, Boehm, 1995; Perry & Wolf, 1992).

Danas se upotrebljava veliki broj različitih metoda za razvoj softverske arhitekture, koji koriste različitu terminologiju i predviđaju različite faze u razvojnom procesu. Međutim, osnovni arhitekturalni koncepti su zajednički za većinu metoda.

Prvi koncept razvoja softverske arhitekture predstavljaju statička i dinamička struktura sistema. Rozanski i Woods (2012) smatraju da elementi i veze između njih grade strukturu nekog sistema na dva načina: statički i dinamički. Statička struktura sistema definiše njegove unutrašnje elemente i njihovo uređenje. To mogu biti programi, objektno orijentisane klase ili paketi, procedure baza podataka, servisi ili neke druge samostalne jedinice koda. Unutrašnji elementi podataka uključuju klase, entitete/tabele baze podataka i datoteke podataka. Unutrašnji hardverski elementi podrazumevaju računare ili njihove komponente kao što su disk ili centralni processor, i mrežne elemente kao što su kablovi, ruteri ili habovi. Statička uređenost sistema predstavlja skup asocijacija, specifičnih veza i odnosa između ovih

elemenata. Za softverske module može postojati na primer statička povezanost, kao što je hijerarhija elemenata (modul A izgrađen od modula B i C) ili zavisnost između elemenata (modul A se oslanja na servise modula B). Za klase, relacione entitete ili druge elemente podataka, veze definišu na koji način je jedna jedinica podatka povezana sa drugom. U slučaju hardvera, veze definišu neophodne fizičke interkonekcije između različitih hardverskih komponenata sistema.

Dinamička struktura sistema pokazuje kako sistem radi, šta se dešava prilikom njegovog izvršavanja i kako odgovara na spoljne/unutrašnje stimulanse. Dakle, ona definiše izvršne elemente sistema i njihove interakcije, koje mogu predstavljati tok informacija između elemenata (elemenat A šalje poruke elementu B); paralelno ili sekvencijalno izvršavanje unutrašnjih zadataka (elemenat Z poziva rutinu elementa W); ili mogu biti izraženi u terminima njihovog uticaja na podatke (jedinica podatka D je kreirana, ažurirana i konačno uništena). Statička i dinamička struktura sistema su međusobno tesno povezane, stim da bez statičke strukture ne postoji dinamička struktura (Rozanski & Woods, 2012).

Drugi važan koncept u razvoju softverske arhitekture predstavljaju osobine budućeg sistema, koje se manifestuju kroz spoljašnje, vidljivo ponašanje sistema (šta sistem radi) ili kroz kvalitativne osobine sistema, koje opisuju kako sistem radi. Spoljašnje ponašanje sistema predstavlja izvršavanje određenih funkcionalnosti usled interakcije sistema i njegovog okruženja. Ono podrazumeva tok informacija u/iz sistema, odgovarajuće reagovanje sistema na stimulanse i sl. Kvalitativne osobine sistema predstavljaju nefunkcionalne zahteve sistema, koji opisuju kako sistem radi u terminima performantnosti, skalabilnosti, sigurnosti i sl. Uređenost statičke i dinamičke stukture sistema čini njegovu arhitekturu. Ona mora biti takva da može da zadovolji ključne funkcionalne i nefunkcionalne zahteve. Svaki sistem ima arhitekturu bez obzira da li je ona dokumentovana, razumljiva, efektivna ili efikasna za određeni set zahteva stejkoholdera. Stoga je uloga softver arhitekte da pronađe najadekvatnije arhitekturalno rešenje za sistem koji se razvija i da ga dokumentuje na način koji je u skladu sa projektnim i organizacionim kontekstom (Rozanski & Woods, 2012).

Rozanski i Woods kao treći bitan koncept ističu arhitekturalne elemente. Arhitekturalni elementi, ili samo elementi, predstavljaju osnovne delove budućeg sistema. Priroda ovih elemenata zavisi pre svega od tipa sistema koji se gradi, kao i od konteksta u kojem se oni razmatraju. Elemente karakteriše sledeći set osobina: jasno definisane odgovornosti, jasno definisane granice i jasno definisan interfejs putem kojeg se opisuju usluge, koje dati elemenat obezbeđuju drugim elementima (Rozanski & Woods, 2012).

Četvrti koncept u razvoju softverske arhitekture sistema predstavljaju njeni stejkholderi, koji mogu biti pojedinci, timovi ili organizacija. Stejkholderi ispoljavaju određeni interes vezan za arhitekturu budućeg sistema, koji može biti izražen kroz njihove zahteve, ciljeve, ograničenja, namere ili težnje. Mnogi od ovih interesa mogu biti zajednički za sve stejkholdere, ali veliki deo njih može biti i različit, pa čak vrlo često i konfliktan. Rešavanje ovih konflikata, na način

kojim će svi stejkholderi biti zadovoljni u odnosu na sopstvene zahteve, predstavlja pravi izazov u razvoju softverske arhitekture. Uloga stejkholdera u razvoju softverske arhitekture je od izuzetnog značaja, jer njihovi zahtevi opredeljuju pravac razvoja i sam "oblik" arhitekture. Adekvatna arhitektura je ona koja odgovara isključivo zahtevima njenih stejkholdera (Rozanski & Woods, 2012).

Peti koncept predstavlja opis ili dokumentovanje softverske arhitekture putem seta različitih artifakata, čiji izbor zavisi od konkretne metode koja se upotrebljava za razvoj softverske arhitekture i konteksta u kojem se razvija budući sistem. Artifakti imaju za cilj da na što jasniji i razumljiviji način uvere stejkholdere da predložena arhitektura ispunjava njihove zahteve. Opis i dokumentovanje softverske arhitekture vrši se različitim tehnikama: dokumentima, tabelama, kodom, wiki, UML alatima za modelovanje i dr. Modeli poseduju značajnu ulogu u procesu razvoja softverske arhitekture, jer predstavljaju apstraktnu, pojednostavljenu ili parcijalnu reprezentaciju nekih aspekata arhitekture, putem kojih se može ostvarivati efikasnija komunikacija sa stejkholderima.

Softverska arhitektura kompleksnih sistema ne može se predstaviti jedinstvenim modelom, već setom međusobno povezanih modela, koji zajedno čine celinu. Setovi modela organizuju se u poglede, pri čemu se svaki pogled bavi opisom specifičnog aspekta arhitekture. Svaki arhitekturalni pogled sastoji se od bar jednog modela. Arhitekturalni modeli se mogu strukturirati na formalne: kvalitativne ili kvantitativne modele i neformalne modele – skice. Kvalitativnim modelima predstavljaju se ključni strukturni ili bihejvioralni elementi, karakteristike ili atributi arhitekture koja se modeluje. Kvantitativni arhitekturalni modeli predstavljaju merljive osobine arhitekture, kao što su performantnost, skalabilnost, kapacitet i dr. Skice predstavljaju neformalne grafičke modele ili slike, namenjene uglavnom za komunikaciju sa netehničkim osobljem u cilju njihovog što bolje razumevanja suštine arhitekture (Rozanski & Woods, 2012).

Arhitekturalni modeli predstavljaju tehniku kojom se predstavljaju arhitekturalno značajni aspekti sistema, putem arhitekturalno značajnih elemenata. Arhitekturalno značajni elementi su oni koji imaju širok uticaj na strukturu sistema i njegovu performantnost, skalabilnost i evolutivnost.

Arhitekturalno značajne elemente čine (Kruchten, 2004):

- glavne klase, uglavnom one koje koje predstavljaju glavne poslovne entitete,
- arhitekturalni mehanizmi, koji glavnim klasama obezbeđuju ponašanje, kao što su mehanizmi komunikacije i sl,
- obrasci (paterni) i radni okviri,
- slojevi i podsistemi,
- interfejsi,
- glavni procesi, ili niti kontrole.

Prilikom izrade arhitekturalnih modela koriste se različiti jezici za modelovanje, tj. jezici za opis arhitekture (Architecture Description Languages, u nastavku ADL): Acme - sa Carnegie Mellon univerziteta; Achitecture Analysis and Design Language (u nastavku AADL) - standardizovan od strane Society of Automotive Engineers (SAE); Rapide - sa Stanford univerziteta; ArchiMate - standardizovan od strane Open Group i dr. Alternativu za korišćenje ovih ADL jezika, predstavlja The Unified Modeling Language (u nastavku UML) - opšti jezik modelovanja (Rozanski & Woods, 2012).

Pored opisanih koncepata, u literaturi se pominje i kategorija ostali arhitekturalni koncepti, koju čine: arhitekturalni stilovi, arhitekturalni obrasci, arhitekturalni mehanizmi i taktike (perspektive). Arhitekturalni stilovi predstavljaju postojeća, u praksi proverena i potvrđena rešenja dizajna za određene kompleksne arhitekturalne probleme. Predstavljaju vrstu dokumentovanog arhitekturalnog znanja i standardizovano rešenjem za određene arhitekturalne probleme, koje može biti ponovno upotrebljeno u rešavanju sličnih problema. Arhitekturalni stil izražava osnovnu strukturalnu organizacionu šemu softverskog sistema. Upotreba arhitekturalnih stilova smanjuje izbor mogućih formi arhitekturalnog rešenja i nameće izvestan stepen uniformnosti arhitekture. Odabrani stil obezbeđuje set predefinisanih tipova elemenata, njihovih odgovornosti, pravila i uputstva za ostvarivanje njihovih veza. Stil može biti definisan setom obrazaca (paterna) ili setom specifičnih komponenti i konektora, kao osnovnih gradivnih elemenata arhitekture. Rešenje koje predstavlja arhitekturalni stil odnosi se na sistem kao celinu, a ne na njegov poseban deo. Rešenje se opisuje u terminima arhitekturalnih elemenata, njihovih interfejsa, tipova konektora i načina kombinovanja elemenata i konektora. Primeri često korišćenih stilova su: cevi i filteri (pipe and filter) i klijent-server (Rozanski & Woods, 2012; Rational Team, 2010; Kruchten, 2004).

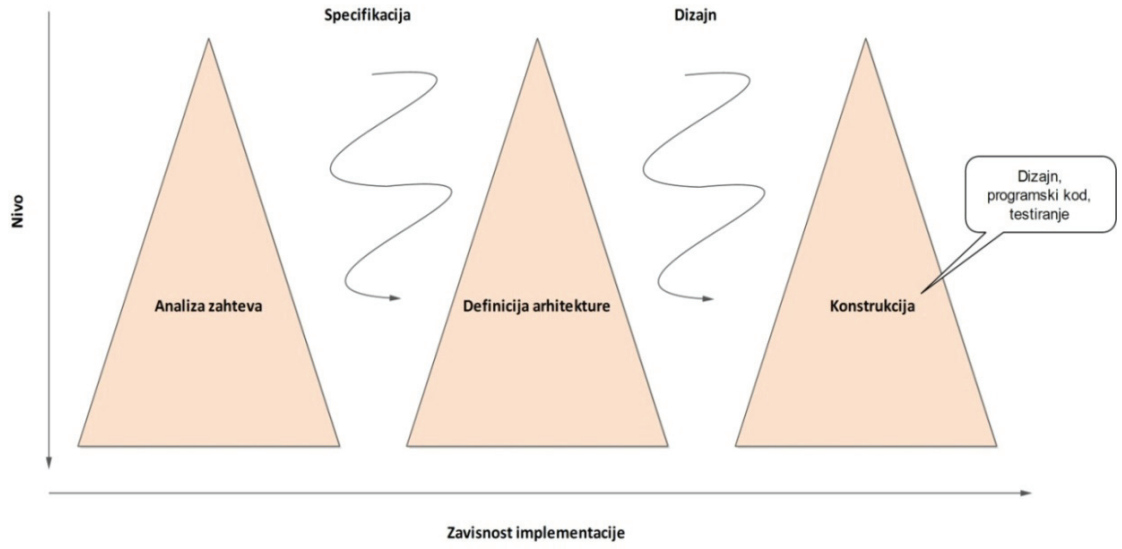
Arhitekturalni softverski obrasci (paterni) dizajna predstavljaju, takođe, postojeća i u praksi verifikovana rešenja dizajna (bar tri puta uspešno primenjena), ali za mnogo specifičnije probleme koji se odnose na strukturu jednog ili više specifičnih delova sistema. Obrasce dizajna predstavljaju sledeći termini: procedure, klase, strukture podataka i strukture koje se formiraju njihovim kombinovanjem. Obrasci, kao vrsta standardizovanog arhitekturalnog znanja, razlikuju se prema svojoj nameni u odnosu na tip i domen problema. Primere obrazaca dizajna predstavljaju: Model-View-Controller (MVC) i Object Request Broker (ORB) (Rozanski & Woods, 2012; Rational Team, 2010; Kruchten, 2004).

Arhitekturalni mehanizam predstavlja klasa, grupa klasa ili čak obrazac koji obezbeđuje zajedničko rešenje za zajednički problem. Koriste se uglavnom na srednjim i nižim slojevima arhitekture, sa ciljem obezbeđivanja određenog ponašanja klasa (Kruchten, 2004).

Softverska arhitektura predstavlja most između procesa analize zahteva i procesa izgradnje softvera, koji uključuje: dizajn, kodiranje i testiranje. Rozanski i Woods (2012) su definisali Three Peaks model, koji je nastao proširivanjem Twean Peaks modela, postavljenog od strane

Nuseibeh (2001). Model je predstavljen na slici 3.1 i jasno opisuje vezu i interakciju koja postoji između analize zahteva, arhitekture i izgradnje softvera:

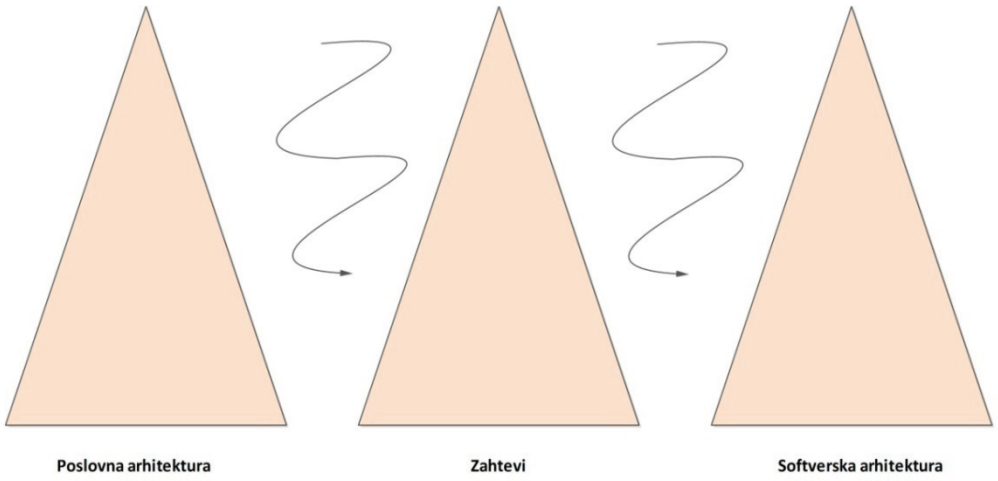
- Rezultati analize zahteva (identifikovani obim, sistemski funkcionalni i nefunkcionalni zahtevi) pružaju kontekst za dizajn softverske arhitekture.
- Prilikom definisanja softverske arhitekture često se otkrivaju nedoslednosti u zahtevima, nedostatak zahteva, kao i informacije vezane za visoke troškove i kompleksnost ispunjavanja pojedinih zahteva stejkholdera. Ove informacije predstavljaju povratnu spregu procesu analize zahteva, na osnovu koje se vrši ponovno utvrđivanje prioriteta zahteva i kompromisno donosi odluka, shodno vremenskim i budžetskim ograničenjima.
- Kada se utvrdi da definisano arhitekturalno rešenje može da ispuni set identifikovanih zahteva stejkholdera, otpočinje aktivnost planiranja izgradnje sistema.
- Izgradnja predstavlja set inkrementalnih isporuka izvršnog softverskog rešenja, pri čemu svaka isporuka obezbeđuje povratne informacije arhitekturalnom nivou. Povratne informacije ili potvrđuju adekvatnost arhitekturalnog rešenja ili identifikuju probleme, koji trebaju biti rešeni na nivou arhitekture. Ovim se započinje novi ciklus (Rozanski & Woods, 2012).



Slika 3.1 Interakcija Analize zahteva-Arhitekture-Konstrukcije
Izvor: (Rozanski & Woods, 2012)

Drugi model koji nosi isti naziv, Three Peaks, takođe je nastao kao rezultat proširenja Twean Peaks modela (Nuseibeh, 2001), a predložili su ga Satish Chandra i Satyendra Bhattaram 2003. godine. Ovaj model polazi od koncepta da se i funkcionalni i nefunkcionalni zahtevi, kao inputi za razvoj softverske arhitekture, izvode iz poslovne arhitekture. Uticaj softverske arhitekture na zahteve ima kaskadni efekat i na poslovnu arhitekturu i obrnuto. Uticaj softverske arhitekture na modifikaciju zahteva, te njihov dalji uticaj na poslovnu arhitekturu, često rezultira reinženjeringom poslovnog procesa. Kreiranjem poslovnog modela, na osnovu

kojeg se identifikuju arhitekturalno značajni zahtevi, povećava se kvalitet razvijene softverske arhitekture sistema (Chandra & Bhattaram, 2003). Model je predstavljen na slici 3.2.



Slika 3.2 Model istovremenog razvoja poslovne arhitekture, zahteva i softverske arhitekture.
Izvor: (Chandra & Bhattaram, 2003)

3.2 Metode za razvoj softverske arhitekture

Danas je u upotrebi veliki broj različitih metoda, tehnika, uputstava, procesa, kao i najboljih praksi za razvoj softverske arhitekture (Rozanski & Woods, 2012; Bass, Clements, Kazman, 2012; Booch, Maksimchuk, Engle, Young, 2007; Gorton, 2006; Kruchten, Obbink, Staord, 2006; Bosch, 2000; Shaw & Garlan, 1996; Gacek, Abd-Allah, Clark, Boehm, 1995; Perry & Wolf, 1992). Raznolikost metoda za razvoj softverske arhitekture upućuje na njihovo međusobno razlikovanje, kako u pogledu korišćene terminologije, tako i u pogledu samog razvojnog procesa. Adekvatnost primene neke metode zavisi od domena za koji je ona predviđena, od veličine organizacije, veličine razvojnog tima, orijentisanosti na posebnu vrstu sistema i sl. Sa druge strane, sve ove metode za razvoj softverske arhitekture imaju i mnogo toga zajedničkog, jer se u osnovi bave istim problemom. U nastavku sledi pregled najzastupljenijih metoda za razvoj softverske arhitekture:

1. ADD metoda (Bass, Clements, Kazman, 2013) - specifična je jer prilikom dekomponovanja sistema ili njegovih elemenata stavlja naglasak na nefunkcionalne zahteve sistema. Na dekomponovane elemente sistema primenjuju se one arhitekturalne taktike i obrasci, za koje se smatra da mogu najadekvatnije doprineti ispunjavanju identifikovanih nefunkcionalnih zahteva sistema.

2. Siemens 4 Views (u nastavku S4V) (Hofmeister, Nord, Soni, 2000) - zasniva se na četiri arhitekturalna pogleda: konceptualni pogled, pogled modula, pogled izvršavanja i pogled koda. Metoda je efikasna isključivo u razvoju arhitekture ugrađenih sistema i sistema koji funkcionišu u realnom vremenu.

3. RUP 4+1 Views (Kruchten, 2004) – sadrži 4+1 pogled, čiji opisi predstavljaju sastavni deo arhitekturalne dokumentacije: pogled slučajeva upotrebe, logički pogled, procesni pogled, pogled raspoređivanja i pogled implementacije.

4. Business Architecture Process and Organization (u nastavku BAPO) (America, Obbink, Rommes, 2003) - razvijena od kompanije Philips, sa ciljem da razvoj softverske arhitekture (A) softversko intenzivnih sistema bude usklađen sa poslovnim kontekstom (B), razvojnim procesom (P) i organizacijom (O).

5. Views and Beyond – razvijena od strane Istraživačkog instituta Carnegie Mellon univerziteta (Clements et al., 2010). Fokus metode je pre svega na dokumentovanju arhitekture, zbog čega je potrebno koristiti u kombinaciji sa nekom drugom metodom za razvoj arhitekture. Nema tipične poglede već nudi set arhitekturalnih stilova koji su grupisani u kategorije: komponenta i konektor, modul i alokacija.

6. Garlan i Anthony metod (Garland & Anthony, 2003) - predstavlja set pogleda koji pogoduju razvoju arhitekture informacionih sistema: analiza, analiza interakcija, uopštena analiza, komponente, interakcija komponenti, stanje komponenti, kontekst, raspoređivanje, slojevi podsistema, logički podaci, fizički podaci, proces, stanje procesa, zavisnosti interfejsa podsistema.

7. The Integrated Architecture Framework (u nastavku IAF) (Wout et al., 2010) - je arhitekturalni okvir razvijen u okviru francuske IT kompanije Capgemini početkom 90-tih godina. Prvobitno je fokus okvira bio na razvoju arhitekture sistema, da bi kasnije bio dorađen i proširen i za razvoj arhitekture na nivou preduzeća (enterprise architecture). Zasnovan je na četiri pogleda: kontekstualni, konceptualni, logički i fizički pogled. Uz navedene poglede predviđa i četiri aspekta: poslovna arhitektura, informaciona arhitektura, arhitektura informacionih sistema i tehnološka infrastruktura.

8. Reference Model for Open Distributed Processing, RM-ODP - je nastao kao rezultat zajedničkih napora ISO i IEC institucija. Radni okvir je namenjen za razvoj arhitekture distribuiranih sistema i sadrži pet pogleda putem kojih se ujedno vrši i razvoj i opis arhitekture: pogled preduzeća, pogled informacija, pogled tehnologije, razvojni pogled i računarski pogled (Putman, 2000).

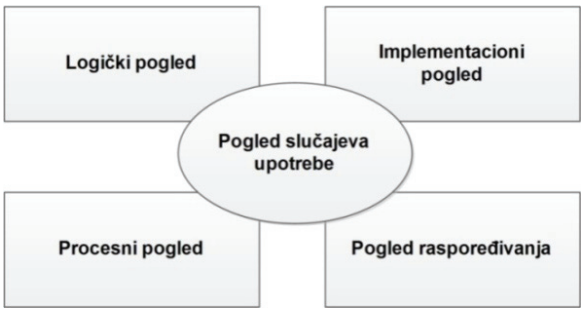
U narednom delu rada biće predstavljen Kruchten 4+1 proces razvoja softverske arhitekture, iz razloga što je inkorporiran u RUP, čija je detaljna analiza, sa aspekta arhitekturalnih aktivnosti jedan od ciljeva teorijskog dela istraživanja doktorske disertacije.

3.2.1 Kruchten 4+1 proces razvoja softverske arhitekture

U slučaju razvoja kompleksnih sistema, nije moguće jedinstvenim modelom predstaviti ključne funkcionalnalne i nefunkcionalne zahteve sistema, tako da on bude razumljivim svim stejkholderima. Široko prihvaćeno stanovište je da razvoj arhitekture kompleksnih sistema treba da bude zasnovan na brojnim odvojenim, ali međusobno povezanim, pogledima, pri čemu svaki od njih opisuje poseban aspekt arhitekture. Ideja potiče još od Parnasa 1970. godine, ali je u širokoj upotrebi tek pošto je Kruchten objavio svoj model pogleda na razvoj softverske arhitekture, poznat pod nazivom "Architectural Blueprints 4+1". Model je postao i sastavni deo RUP radnog okvira. Standardi IEEE 1471 i IEEE 42010 formalizovali su koncept pogleda, obezbeđujući detaljna uputstva za njihov opis (Rozanski & Woods, 2012).

Proces razvoja arhitekture, prema Kruchten-u, prikazan je na slici 3.3, a realizuje se kroz opis svih ili nekih od sledećih pogleda u okviru artifakta - dokument softverske arhitekture (Kruchten, 1995):

- Logički pogled, opisuje funkcionalne zahteve sistema. Model dizajna je njegov sastavni deo i sadrži glavne pakete dizajna, podsisteme i klase.
- Procesni pogled, oslikava aspekte dizajna koji se odnose na konkurentnost i sinhronizaciju, u terminima procesa i niti.
- Pogled raspoređivanja, opisuje mapiranje softvera i hardvera i odražava aspekt distribuiranosti.
- Implementacioni pogled, opisuje organizaciju statičkih modula softvera u radnom okruženju, u terminima paketa i slojeva.
- Pogled slučaja upotrebe, opisuje ključne scenarije ili slučajeve upotrebe, koji su putokaz za dizajn arhitekture, ali i sredstvo validacije drugih pogleda.



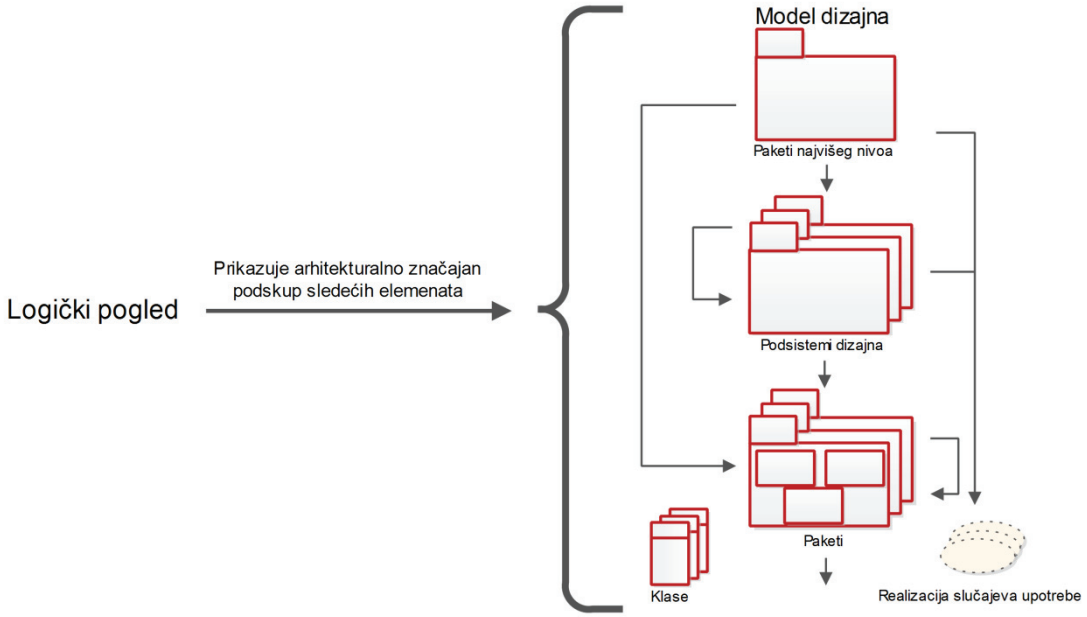
Slika 3.3 Model4+1
Izvor: (Rational Team, 2010)

Logički pogled, prikazan na slici 3.4, fokusira se na analizu funkcionalnih zahteva i izbor onih koji predstavljaju ključne funkcionalnosti budućeg sistema. Sadrži podskup artifakta, kojima se realizuje dekomponovanje budućeg sistema na skup apstrakcija (objekata i klasa, njihova organizacije u pakete i podsisteme i organizacija paketa i podsistema u slojeve). Na

identifikovane apstrakcije primenjuju se principi apstrakcije, enkapsulacije i nasleđivanja (Rational Team, 2010; Kruchten, 2004).

Opis logičke arhitekture kompleksnih sistema polazi od predstavljanja paketa najvišeg nivoa, njihovih međuzavisnosti i raspoređenosti u slojeve, potom predstavljanja najznačajnih paketa unutar njih i tako sve do paketa najnižeg nivoa hijerarhije. Arhitekturalno značajni paketi dizajna se opisuju putem sledećih informacija: naziv, kratak opis, i dijagram značajnih klasa i paketa koje sadržava. Svaku značajnu klasu paketa opisuju sledeće komponente: naziv, kratak opis i opciono neke njene odgovornosti, operacije, atributi i bitni odnosi.

Logički pogled najčešće sadrži i opis realizacija najznačajnijih slučajeva upotrebe, koji odražavaju centralne funkcionalnosti budućeg sistema ili delikatne tačke arhitekture. Takođe, cilj ovog pogleda je i da objasni kako različiti elementi modela dizajna doprinose datim funkcionalnostima. Za svaki značajan slučaj upotrebe, za koji se izrađuje realizacija, potrebno je dati opis koji sadrži sledeće informacije: naziv realizovanog slučaja upotrebe, kratak opis, opis toka događaja, navođenje najznačajnijih interakcija ili dijagrama klasa povezanih sa realizacijom slučaja upotrebe, i opis zahteva izvedenih iz realizacije slučaja upotrebe (Rational Team, 2010; Kruchten, 2004).

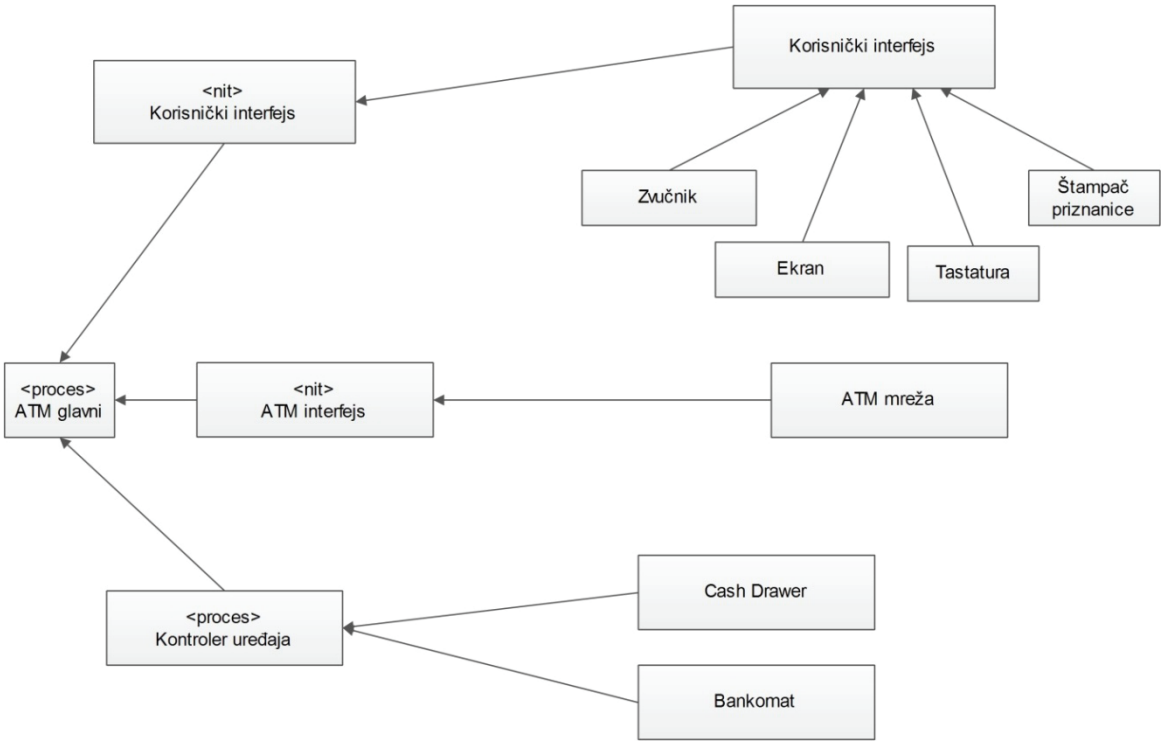


Slika 3.4 Logički pogled
Izvor: (Rational Team, 2010)

Procesni pogled, prikazan na slici 3.5, fokusiran je na procesnu dekompoziciju sistema, pitanja konkurentnog pristupa i distribucije budućeg sistema, kao i na pitanja integriteta sistema i njegove rezistentnosti na greške. Procesna arhitektura se može prikazati na različitim nivoima apstrakcije, sa ciljem rešavanja različitih arhitekturalnih pitanja. Na najvišem nivou, procesna arhitektura predstavlja set međusobno nezavisnih izvršnih logičkih mreža programa,

koji međusobno komuniciraju (tzv. „procesa“), raspoređenih na hardverske resurse koji grade lokalnu (LAN) ili WAN mrežu. Proces predstavlja set aktivnosti koje čine izvršivu celinu, dok je aktivnost pojedinačna nit nekog procesa (Rational Team, 2010; Kruchten, 2004).

Aktivnosti se grupišu na glavne aktivnost i sporedne aktivnosti. Glavne aktivnosti predstavljaju elemente arhitekture kojima pojedinačno treba posvetiti pažnju, dok sporedne aktivnosti predstavljaju dodatne aktivnosti koje se uvode na lokalnom nivou tokom implementacije. Glavne aktivnosti međusobno komuniciraju putem dobro definisanih mehanizama za komunikaciju: sinhronih i asinhronih servisa za komunikaciju zasnovanih na porukama, daljinskim pozivanjem porcedura, itd. Sporedne aktivnosti komuniciraju sastajanjem ili deljenom memorijom. Svaku mrežu procesa potrebno je opisati sledećim informacijama: naziv, procesi od kojih se sastoji, interakcije između procesa (u formi dijagrama komunikacije u kojima objekti predstavljaju procese koji sadrže sopstvene niti kontole). Procesi se ukratko opisuju navođenjem njihovog ponašanja, veka trajanja i komunikacionih karakteristika. Vizuelna sredstva kojima se opisuje ovaj pogled su dijagrami klasa i dijagrami objekata (Rational Team, 2010; Kruchten, 2004).



Slika 3.5 Primer procesnog pogleda
Izvor: (Rational Team, 2010)

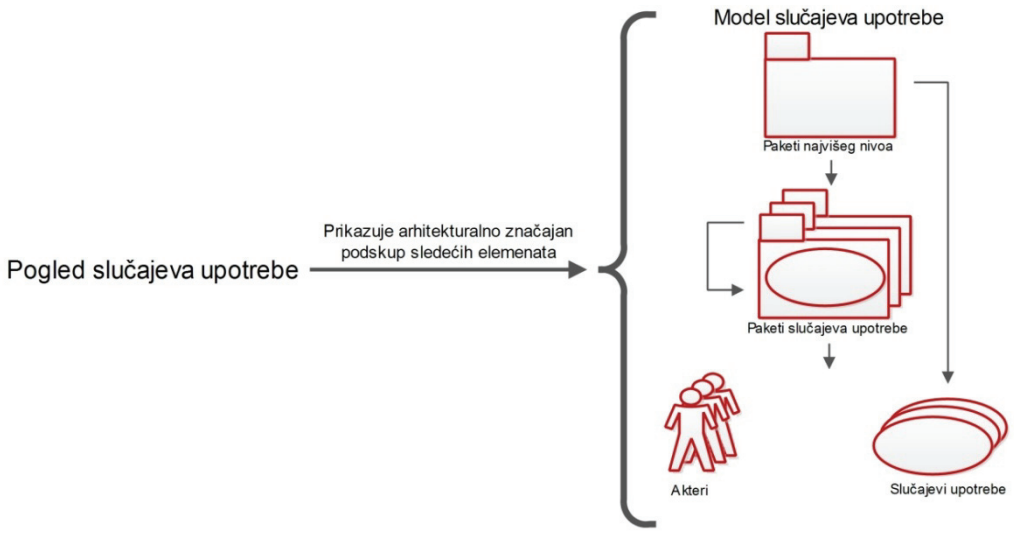
Implementacioni pogled obuhvata izradu modela implementacije putem kojeg se popisuju svi podsistemi; dijagrama komponenti koji ilustruje kako su podsistemi organizovani u slojeve i hijerarhije; i uvoznih zavisnosti podsistema. Struktura modela implementacije razvija se rano, paralelno sa razvojem drugih aspekata arhitekture. Model treba da eliminiše probleme

vezane za upravljanje konfiguracijom i da omogući nesmetanu implementaciju, integraciju i proces izgradnje. Pored ove primarne uloge i odgovornosti, softver arhitekta bi trebao imati znanja i u domenu integracija na nivou sistema, kao i iskustva u programskom jeziku kojim će komponente biti pisane (Rational Team, 2010; Kruchten, 2004).

Pogled slučajeva upotrebe, prikazan na slici 3.6, podrazumeva opis slučajeva upotrebe koji su značajni sa aspekta arhitekture budućeg sistema. Identifikovanjem ovih slučajeva upotrebe spoznaju se ključne funkcionalnosti koje sistem treba da pruži korisnicima. Ovaj pogled predstavlja podskup sledećih artifakata: modela slučajeva upotrebe, dijagrama slučajeva upotrebe, aktera i klasa dizajna, dijagrama sekvenci.

Svaki arhitekturalno značajan slučaj upotrebe treba da bude eksplicitno opisan. Atributi njegovog opisa su (Rational Team, 2010; Kruchten, 2004):

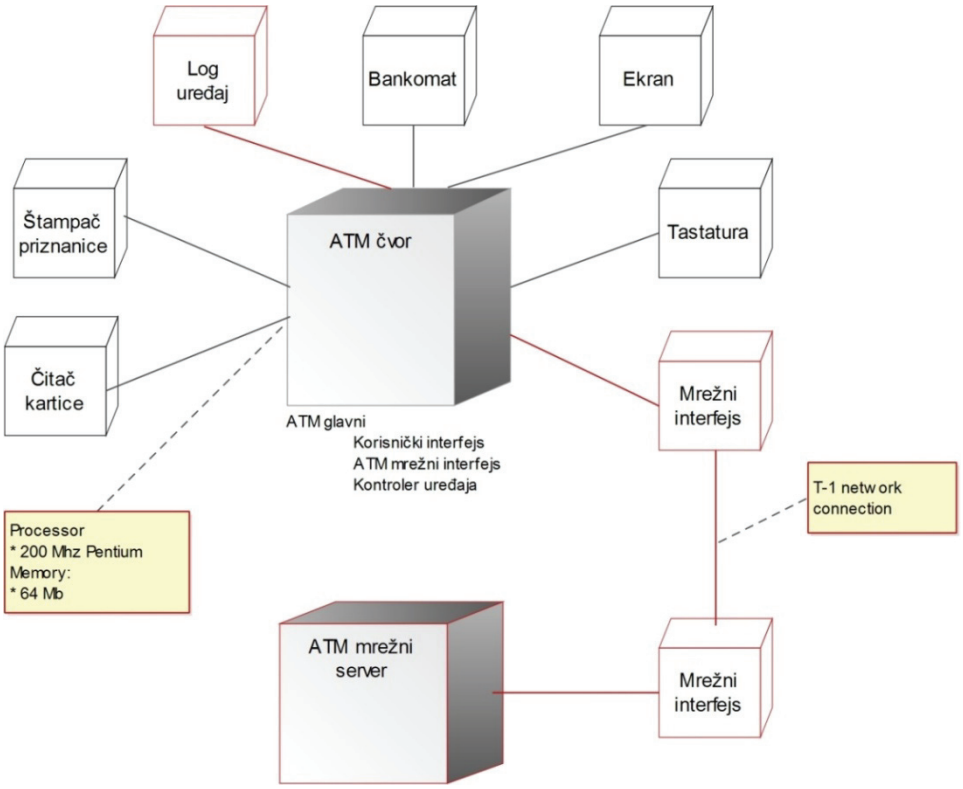
- naziv,
- kratak tekstualni opis,
- opis toka događaja datog slučaja upotrebe,
- opis odnosa koje uključuje dati slučaj upotrebe,
- značajni slučajevi upotrebe sa kojima je povezan,
- opis posebnih zahteva datog slučaja upotrebe,
- prikaz korisničkog interfejsa radi pojašnjavanja slučaja upotrebe i
- izrada realizacije datog slučaja upotrebe.



Slika 3.6 Pogled slučajeva upotrebe
Izvor: (Rational Team, 2010)

Pogled raspoređivanja, prikazan na slici 3.7, fokusira se na razmatranje nefunkcionalnih zahteva sistema (npr. performantnost, skalabilnost, sigurnost i dr.) i na raspoređivanje

softverskih elemenata na odgovarajuću mrežu fizičkih, hardverskih elemenata – čvorova na kojima će se izvršavati. Za svaku konfiguraciju fizičke mreže ovaj pogled predviđa sledeće informacije: ime, dijagram raspoređivanja, opis opštih pravila mapiranja softvera i hardvera, opis mrežnih konfiguracija za testiranje softvera i simulacije. Opisan pogled se dokumentuje putem artifakta model raspoređivanja, koji ilustruje distribuciju obrade preko niza čvorova sistema, uključujući fizičku distribuciju procesa i niti (Rational Team, 2010; Kruchten, 2004).



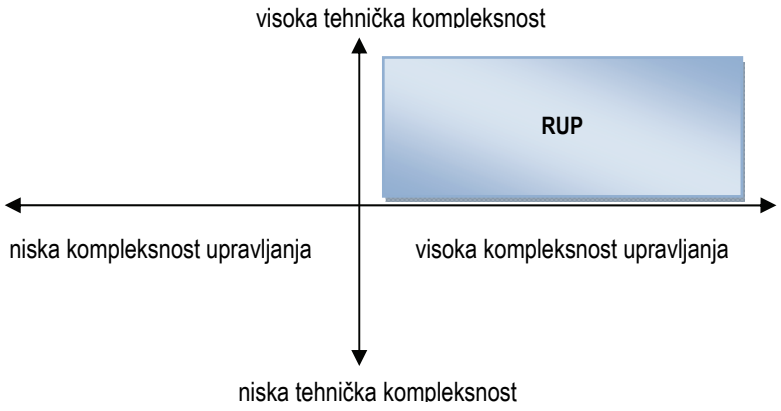
Slika 3.7 Primer pogleda raspoređivanja
Izvor: (Rational Team, 2010)

Razvijena softverska arhitektura dokumentuje se artifaktom - dokument softverske arhitekture, koji sadrži opis predstavljenih arhitekturalnih pogleda i artifaktom arhitekturalni prototip. Arhitekturalni prototip implementira najvažnije komponente ili rešenja dizajna u cilju njihove validacije, testiranja i ocene. Zajedno sa proof-of concept-om (u nastavku POC) predstavlja jednu od tehnika evaluacije predloženog arhitekturalnog rešenja. Primena koncepta evaluacije počinje rano, čim se donese set arhitekturalnih odluka i traje sve dok stejkholderi ne potvrde validnost arhitekture. Evaluacija arhitekturalnog rešenja, putem navedenih tehnika, obezbeđuje rano dobijanje povratnih informacija o validnosti rešenja u odnosu na arhitekturalno značajne zahteve, s jedne strane i sticanje poverenja stejkholdera u ljude, tehnologiju i proces. Izgrađeni prototip, tokom faze elaboracije, evoluira tokom faze konstrukcije u konačan sistem (Kruchten, 2004).

3.2.2 RUP

RUP razvojni model, postavila je i komercijalizovala Rational Software kompanija, koja je danas u vlasništvu IBM-a. Predstavlja objektno orijentisani razvojni proces, baziran na generativnom razvojnom procesu Unified Software Development Process, koji se skraćeno naziva Unified Process (UP). Svoju popularnost i široku zastupljenost u praksi otpočinje nakon što su Ivar Jacobson, Grady Booch i James Rumbaugh objavili knjigu The Unified Software Development Process (1999).

Po robusnosti i nivou ceromonijalnosti predstavnik je "teških" metodologija i smatra se tradicionalnim procesom razvoja softvera. Predstavlja dobro definisan i struktuiran proces, u kojem je jasno definisano ko je odgovoran za šta i kada. Predviđa 635 aktivnosti i zadataka i 35 različitih uloga, čije se odgovornosti delegiraju u okviru devet disciplina. RUP se prevashodno smatra procesnim radnim okvirom koji sadrži skup dobro opisanih preporuka i detaljnih uputstava za razvoj softverskih rešenja, na osnovu kojih je svaka organizacija u mogućnosti da kreira sopstveni metodološki okvir. Zahvaljujući tome RUP je, sve do pojave zahteva za razvoj malih internet poslovnih softverskih rešenja, jednako uspešno nalazio svoju primenu u razvoju velikih i kompleksnih projekata s jedne strane, ali i projekata male i srednje veličine (slika 3.8.). Uspostavljanje samog razvojnog procesa postao je, međutim, zahtevniji i vremenski duži posao od samog internet rešenja koje se trebalo razviti. Tradicionalni proces razvoja iziskuje velika ulaganja resursa, što preduzeća suočena sa brzim razvojem tehnologije i čestim promenama u poslovnom okruženju, nisu u mogućnosti da obezbede. Stoga je prva faza u modifikaciji razvojnih procesa predstavljala agilizaciju tradicionalnih procesa razvoja, a druga, još uvek aktuelna, jeste tradicionalizacija agilnih procesa razvoja softvera (Matković, 2011; Matković, Tumbas, Sakal, 2011).



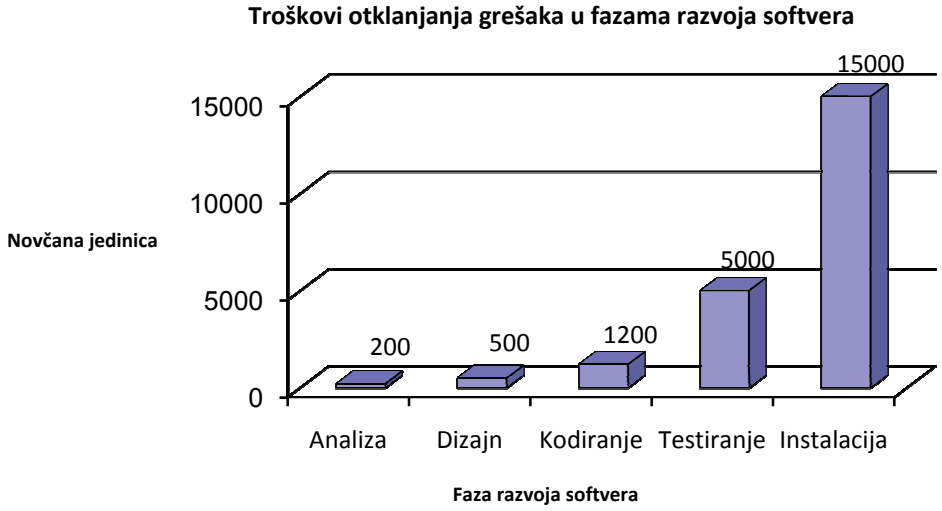
Slika 3.8 RUP radni okvir u realizaciji kompleksnih projekata
Izvor: (Rational Team, 2010)

Principi na kojima je zasnovan RUP, obezbeđuju razvoj kvalitetnih kompleksnih softverskih rešenja, zbog čega je i odabran za analizu u radu. Istaknuti RUP principi su: iterativnost i inkrementalnost, upravljanje zahtevima, orijentisanost na arhitekturu, razvoj vođen

slučajevima upotrebe, vizuelno modelovanje, kontinuirana potvrda kvaliteta i kontrola promena softvera.

Iterativnost i inkrementalnost- predstavlja jedan od principa koji označava razvoj softvera kroz veliki broj manjih iteracija, pri čemu se svaka iteracija sastoji od više sekvencijalnih procesa klasičnog modela vodopada. Ovaj razvojni princip podrazumeva ranu identifikaciju i razvoj najvažnijih funkcionalnosti softvera, njihovo rano testiranje, rano dobijanje povratnih informacija od korisnika i ranu isporuku poslovne vrednosti korisnicima. Kontinuiranim i iterativnim testiranjem razvijenih softverskih funkcionalnosti obezbeđuje se manji jaz između momenta nastanka greške i momenta njenog otkrivanja (Kruchten, 2004).

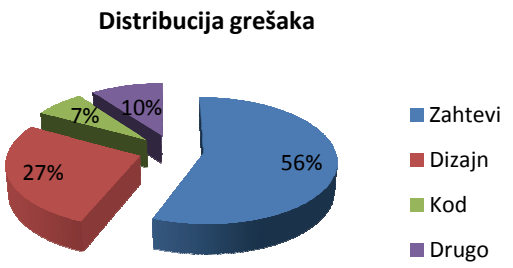
Na ovaj način se ublažava rizik otkrivanja grešaka u kasnijim razvojnim fazama u kojima su, prema istraživanjima, troškovi njihovog otklanjanja višestruko skuplji. Slika 3.9 prikazuje rezultate istraživanja koje su objavili Boehm i Basili (2001), iz kojih se vidi da je za grešku koja se otkrije u fazi analize potrebno 200 NJ (novčanih jedinica) za njeno otklanjanje, a u fazi instalacije 15000 NJ.



Slika 3.9 Troškovi otklanjanja grešaka u fazama razvoja softvera
Izvor: (Boehm & Basili, 2001)

Upravljanje zahtevima – je važan princip što potvrđuju empirijski podaci, prikazani na slici 3.10, koje je James Martin još 1984. godine objavio u knjizi An Information Systems Manifesto. Prema istraživanju koje je sproveo, 56% softverskih grešaka potiče od zahteva, bilo usled njihovog lošeg identifikovanja, analize, organizovanja ili nedostatka upravljanja (Martin, 1984).

Zbog ovakve značajne uloge zahteva u procesu razvoja softvera, RUP čitavu jednu disciplinu fokusira na njih, a cilj discipline je da opiše šta sistem treba da radi. Taj opis treba da bude prihvaćen i od strane korisnika i od strane razvojnog tima. Informacioni zahtevi predstavljaju inpute za dizajn sistema, testiranje sistema, kao i izradu korisničke dokumentacije.



Slika 3.10 Distribucija grešaka prema fazama životnog ciklusa razvoja softvera
Izvor: (Martin, 1984)

Orijentisanost na arhitekturu je princip koji ukazuje na značaj procesa razvoja softverske arhitekture u RUP-u. Dobra arhitektura obezbeđuje ispunjavanje bitnih nefunkcionalnih zahteva, što dalje implicira mogućnost sistema da obuhvati te efikasno i efektivno izvršava sve funkcionalne zahteve. Kompleksan sistem nije moguće efikasno dekomponovati i opisati samo jednim modelom, već je neophodan set međusobno povezanih modela koji zajedno opisuju ključne funkcionalne karakteristike i kvalitativne osobine sistema (nefunkcionalne zahteve). Dizajn arhitekture se u RUP-u odvija putem četiri odvojena, ali povezana pogleda pri čemu svaki od njih opisuje poseban aspekt arhitekture. Na ovakav način opisan kompleksan sistem postaje razumljiv širokom spektru stejkholdera, od poslovnih do tehničkih, koji sistem posmatraju na različite načine i stoga imaju i različite zahteve u odnosu na to šta sistem treba da radi i na koji način.

RUP razvoj softverske arhitekture bazira na Kruchten-ovom modelu 4+1 pogleda: logički pogled, procesni pogledi, fizički pogled, pogled rasporeda i pogled slučajeva upotrebe. Razvoj softverske arhitekture u RUP-u olakšava i princip razvoj baziran na komponentama (Component Based Development, u nastavku CBD), čime se naglašava ponovna upotreba softverskih komponenti, kao potvrđenih arhitekturalnih znanja. Ovakvim razvojem se kroz svaku iteraciju proizvodi izvršna arhitektura sistema koja je merljiva, pogodna za testerinje i ocenjivanje u odnosu na identifikovane arhitekturalne zahteve koje sistem treba da ispuni (Shuja & Krebs, 2008).

Razvoj vođen slučajevima upotrebe su tehnika kojom se identifikuju ključni poslovni procesi date organizacije, na osnovu kojih se potom definišu funkcionalnosti budućeg sistema. Celokupan proces razvoja softvera u RUP-u baziran je na aktivnostima vezanim za slučajeve upotrebe: od njihove identifikacije, opisa, analize, dizajna do same implementacije. Slučajevi upotrebe su osnova i za definisanje slučajeva testiranja softvera (Matković, 2011).

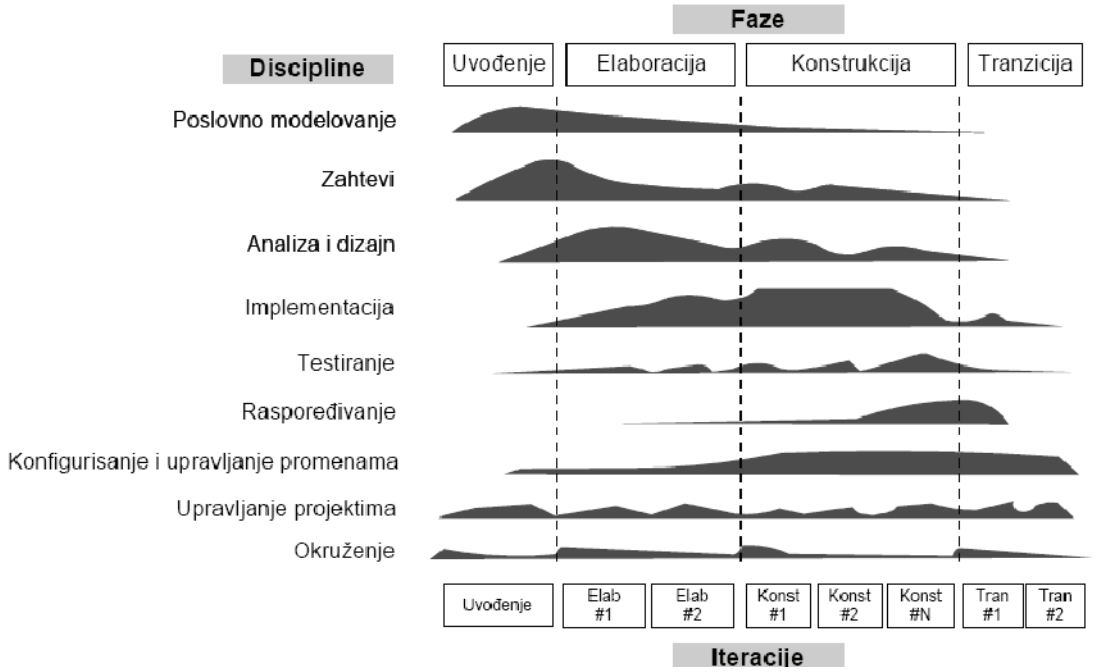
Vizuelno modelovanje – ima za cilj da pojednostavljeno predstavi realnost, odnosno budući sistem iz različitih perspektiva. Modelima se apstrahuju detalji i obuhvataju samo najvažniji elementi, zbog čega su veoma korisna tehnika u razumevanju sistema. Predstavljaju koristan medij za ostvarivanje efikasne komunikacije razvojnog tima i stejkholdera kao i komunikacije

između članova razvojnog tima. Vizuelno modelovanje u RUP-u zasnovano je na standardizovanom jeziku, Unified Modeling Language (u nastavku UML), koji je opšte prihvaćen u softverskoj industriji. UML se koristi kroz ceo životni ciklus razvoja softvera u svrhu: vizualizacije, specifikacije, konstrukcije i dokumentacije softvera (Matković, 2011).

Kontinuirana potvrda kvaliteta – obezbeđuje se kroz kontinuirano testiranje funkcionalnosti sistema i njegovog ponašanja, sa aspekta ispunjenosti nefunkcionalnih zahteva. Prema FURPS (Functionality, Usability, Reliability, Performance, Supportability) modelu za ocenu kvaliteta (Grady, 1992) softverskih sistema izvršena je klasifikacija svojstava koje sistem treba da ispuni, što predstavlja osnovu za testiranje softvera u RUP-u. Procedure testiranje trebaju biti implementirane pre, u toku samog razvoja i nakon uvođenja softvera u radno okruženje. U skladu sa iterativnim razvojem neophodno je testirati svaku iteraciju procesa razvoja. Ovim principom omogućeno je da se sve vrste grešaka otkriju rano, gotovo u momentu njihovog nastanka, čime je obezbeđena mitigacija rizika i njihovo ispravljanje u kasnijim fazama razvoja kada su troškovi daleko veći (Matković, 2011).

Kontrola promena softvera predstavlja veliki izazov naročito na kompleksnim softverskim projektima u kojima učestvuje veći broj timova sa velikim brojem članova, kroz čiji sinergentski rad na brojnim iteracijama se proizvodi softversko rešenje. Razvojne promene su prirodan tok u stvaranju krajnjeg softverskog proizvoda, ali ih je nužno kontrolisati, kako bi se efikasno pratio napredak i razvoj projekta. RUP pruža standardizaciju toka procesa sastavljenog od aktivnosti, artifakata i uloga, koji omogućuje praćenje razvojnih promena, identifikovanje nastalih problema i njihovo brzo otklanjanja (Matković, 2011).

RUP radni okvir razvoj softvera realizuje kroz dve dimenzije (slika 3.11): vremensku (dinamičku) i sadržajnu (statičku). Ovakvim pristupom bitno je umanjen ranije identifikovani problem, koji je karakterisao prethodne razvojne modele: jaz momenta nastanka greške i njenog kasnog otkrivanja. Razdvajanjem vremenske i sadržajne dimenzije, razvoj se zasniva na velikom broju iteracija koje se realizuju setom malih sekvencijalnih koraka. RUP je zasnovao i koncept razlaganja problema na delove umesto dotadašnjeg razlaganja rešenja na delove (Matković, 2011).



Slika 3.11 Prikaz dve dimenzije RUP radnog okvira
Izvor: (Rational Team, 2010)

Vremensku dimenziju predstavljaju četiri faze životnog ciklusa razvoja softvera.

Faza uvođenja je prva razvojna faza koja ima za cilj da utvrdi da li je projekat izvodljiv i finansijski isplativ. Ovi ciljevi se ostvaruju putem realizacije seta aktivnosti, koje kao razultat daju brojne artifakte (tabela 3.1) na osnovu kojih se vrše analize i donosi odluka da li će projekat biti implementiran (Kroll & Kruchten, 2003).

Aktivnosti koje se realizuju u ovoj fazi su sledeće (Rational Team, 2010):

- razumevanje novog projekta,
- priprema projektnog okruženja,
- procena poslovnog statusa,
- priprema okruženja za iteraciju,
- definisanje projektnih planova,
- nadgledanje i kontrola projekta,
- razvoj početne vizije,
- razvoj problemskog modela,
- upravljanje obimom sistema,
- definisanje sistema,
- rad na arhitekturalnoj sintezi,
- definisanje misije ocenjivanja,
- upravljanje iteracijom i
- planiranje naredne iteracije.

Tabela 3.1 Pregled najznačajnijih artifakata faze uvođenja

KLJUČNI ARTIFAKTI FAZE UVOĐENJA	STANJE ARTIFAKTA U KLJUČNOJ TAČCI FAZE UVOĐENJA
Dokument vizije	Identifikovani osnovni zahtevi, ključne karakteristike budućeg sistema i najznačajnija ograničenja koja se nameću budućem sistemu(politička, ekonomska, pravna, tehnička, funkcionalna i sl.). Identifikovani i definisani problemi projekta; identifikovani ključni stejkholderi; utvrđene granica sistema u cilju razumevanja šta se datim projektom razvija;
Dokument poslovnog slučaja	Definisan i odobren. Sadrži neophodne poslovne informacije i izračunat faktor povrata investicije na osnovu kojih se donosi odluka o opravdanosti/neopravdanosti ulaganja u dati projekat.
Lista rizika	Formirana Inicijalna lista rizika. Izvršena njihova procena i rangiranje prema prioritetu. Na osnovu ovog dokumenta donosi se odluku o tome da li je nivo rizika prihvatljiv za dati projekat.
Razvojni plan projekta	Izrađuje se na osnovu artifakt dokument poslovnog slučaja. Definiše se okvirni plan projekta i kriterijumi za merenje uspešnosti projekta. Definisani su ciljevi i metrike za njihovo merenje. Definisana strategija za upravljanje identifikovanim rizicima projekta. Definisan plan za prevazilaženje problema ukoliko se pojave tokom projekta. Definisan plan kojim se vrši praćenje kvaliteta izlaznih rezultata projekta. Grubi proračuni utrošaka resursa (vreme, osoblje, troškovi vezani za okruženje i sl.) koji moraju biti konzistentni sa dokumentom poslovnog slučaja. Troškovi mogu biti proračunati za čitav projekat, pri čemu se ovi proračuni revidiraju u narednim fazama ili samo za fazu elaboracije, kao narednu fazu u životnom ciklusu razvoja proizvoda.
Plan iteracija	Plan iteracija u ovoj tački sadrži u potpunosti razrađenu i odobrenu prvu iteraciju za fazu elaboracije.
Razvojni proces	Prilagođavanje ili konfiguracija razvojnog procesa za konkretan projekat vrši se u cilju uspešnog zadovoljavanja specifičnih potreba projekta. Stoga se prvo sprovodi analiza datog projekta, a potom se definiše šta je potrebno koristiti iz RUP-a. Podrazumeva i aktivnost izbora alata koji će biti korišćeni u razvojnom procesu datog projekta.
Razvojna infrastruktura	Alati za rad na projektu su izabrani, a alati potrebni za rad u fazi uvođenja i početnoj iteraciji faze elaboracije su instalirani.
Rečnik	Definisani značajni pojmovi. Rečnik pregledan i odobren.
Model slučajeva upotrebe	Većina aktera i slučajeva upotrebe je identifikovana, a dijagrami aktivnosti su razrađeni za najznačajnije slučajeve upotrebe.

Izvor: (Rational Team, 2010)

Faza elaboracije – je druga faza realizacije koja treba da obezbedi uslove za efektivnu i efikasnu realizaciju aktivnosti u narednim razvojnim fazama: konstrukciji i tranziciji. Faza je fokusirana na identifikovanje arhitekturalno značajnih rizika, ocenu predloženih arhitekturalnih rešenja (koji su outputi prethodne razvojne faze), uspostavljanje stabilne arhitekture sistema, njenu verifikaciju i odobravanje od strane stejkholdera (implementacijom nekog modela za razvoj prototipa) (Kroll & Kruchten, 2003). U tabeli 3.2 dat je prikaz najznačajnijih artifakata faze eleboracije.

Tabela 3.2 Pregled najznačajnijih artifakata faze elaboracije

KLJUČNI ARTIFAKTI FAZE ELABORACIJE	STANJE ARTIFAKTA U KLJUČNOJ TAČCI FAZE ELABORACIJE
Razvojni prototip	Kreiran jedan/više izvršnih prototipova s ciljem da se identifikuju kritične funkcionalnosti i arhitekturalno značajni scenariji. Izgradnjom prototipova smanjuje se tehnički rizik.
Lista rizika	Proširena identifikovanim rizicima vezanim za arhitekturu sistema, tačnije za nefunkcionalne zahteve koje sistem treba da ispuni.
Razvojni proces	Izmenjen na osnovu iskustva u ranijim fazama projekta i pripremljen za fazu konstrukcije.
Razvojna infrastruktura	Pripremljeno razvojno okruženje za fazu konstrukcije, uključujući sve alate i drugu automatizovanu podršku razvojnog procesa.
Dokument softverske arhitekture	Predstavlja ključni dokument arhitekture sistema, koji sadrži opis arhitekture sa različitih aspekata, pogleda (4+1 View). Sadrži opis arhitekturalno značajnih slučajeva upotrebe, identifikovane ključne mehanizme i elemente dizajna, opisan procesni pogled i pogled raspoređivanja.
Dizajn model	Utvrđen dizajn realizacija onih slučajeva upotrebe koji imaju arhitekturalni značaj. Utvrđeno i ponašanje pojedinačnih elemenata dizajna. Identifikovane softverske komponente i doneta odluka o njihovoj izgradnji, kupovini ili ponovnoj upotrebi, te je na osnovu toga utvrđen raspored poslova u fazi konstrukcije.
Model podataka	Definisani i verifikovani ključni elementi modela podataka (značajni entiteti, njihovi atributi i relacije).
Implementacioni model	Inicijalna struktura je kreirana i izrađen je prototip ključnih komponenti.
Dokument vizije	Prepravljen i proširen informacijama vezanim za arhitekturalno značajne slučajeve upotrebe, koje utiču na buduće odluke vezane za arhitekturu i definisanje planova.
Plan razvoja softvera	Dopunjen i proširen radi pokrivanja faza konstrukcije i tranzicijie.
Plan iteracija	Sadrži u potpunosti razrađenu i definisanu prvu iteraciju faze Konstrukcije.
Model slučajeva upotrebe	Identifikovani svi slučajevi upotrebe i akteri; oko 80% slučajeva upotrebe je u potpunosti razrađeno.
Dodatne specifikacije	Dokumentovane i verifikovane. Sadrže specifikaciju i opis nefunkcionalnih zahteva sistema.
Test paket	Testovi su izgrađeni i sprovedeni za sve izvršne elemente koji su izgrađeni u fazi elaboracije (elementi koji grade arhitekturalno rešenje).
Arhitektura automatizivanih testova	Identifikovani ključni softverski elementi koji će biti testirani.

Izvor: (Rational Team, 2010)

Aktivnosti koje se realizuju kroz fazu elaboracije su (Rational Team, 2010):

- Priprema okruženja za iteraciju.
- Tekuće upravljanje i podrška.
- Izmena i zaključivanje projektnih planova.

- Dorađivanje definicije sistema.
- Definisanje predloga arhitekture.
- Dorađivanje arhitekture.
- Izrada komponenti.
- Integracija i testiranje.
- Izrada materijala za podršku i
- Planiranje naredne iteracije.

Faza konstrukcije – predstavlja treću razvojnu fazu koja ima za cilj završetak izgradnje sistema u skladu sa definisanom i postavljenom arhitekturom. Fokus ove faze je na upravljanje resursima i kontrolu operacija radi optimizacije troškova, planova i kvaliteta kao i proizvodnju upotrebljive i testirane verzije softverskog rešenja koje se isporučuje korisnicima. Pregled najznačajnih artifakata faze konstrukcije dat je u tabeli 3.3 (Kroll & Kruchten, 2003):

Tabela 3.3 Pregled najznačajnijih artifakata faze konstrukcije

KLJUČNI ARTIFAKTI FAZE KONSTRUKCIJE	STANJE ARTIFAKTA U KLJUČNOJ TAČCI
Sistem	Izgrađen izvršiv sistem, spreman za „beta“ testiranje.
Plan raspoređivanja	Izgrađena inicijalna verzija plana raspoređivanja
Implementacioni model	Kreirani svi implementacioni elementi
Test paket	Izgrađeni i izvršeni kako bi se potvrdilo stabilnost svake isporučene verzije (release) softverskog rešenja.
Korisnički materijali za podršku	Izrađeni na osnovu slučajeva upotrebe. Imaju značaj u slučaju kada korišćenje sistema zavisi od upotrebe korisničkog interfejsa.
Plan iteracija	Plan iteracija za fazu tranzicije je izgrađen i odobren.
Dizajn model	Dopunjen sa novim elementima nastalim usled novonastalih zahteva.
Razvojni proces	Redefinisan na osnovu projektnog iskustva i prilagođen za naredu fazu razvoja.
Razvojna infrastruktura	Pripremljena za narednu fazu. Svi alati i automatizovani procesi prilagođeni i spremni za narednu fazu projekta.
Model podataka	Dopunjen sa svim elementima neophodnim za podršku implementaciji.

Izvor: (Rational Team, 2010)

Aktivnosti faze konstrukcije (Rational Team, 2010):

- Priprema okruženja za iteraciju
- Tekuće upravljanje i podrška
- Izrada komponenti
- Integracija i testiranje

- Izrada materijala za podršku
- Planiranje naredne iteracije
- Generisanje paketa za raspoređivanje.

Faza tranzicije - ima za cilj da nakon izvršenog testiranja i korekcije grešaka razvijenog softverskog rešenja, isti isporuči korisnicima. U slučaju da po uvođenju softvera u radno okruženje korisnici zahtevaju veće korekcije, potrebno je pokrenuti novi životn ciklus razvoja. Fokus ove faze je i na obuci korisnika i inženjera koji će održavati isporučeno softversko rešenje i samostalno rešavati određene probleme. Važan elemenat jeste i evaluacija softverskih karakteristika, nakon uvođenja u radno okruženje, u odnosu na dokument vizije i kriterijume prihvatljivosti softverskog rešanja sa početka projekta (Kroll & Kruchten, 2003). Pregled najznačajnijih artifakata faze tranzicije dat je u tabeli 3.4.

Tabela 3.4 Pregled najznačajnijih artifakata faze tranzicije

KLJUČNI ARTIFAKTI FAZE KONSTRUKCIJE	STANJE ARTIFAKTA U KLJUČNOJ TAČCI
Verzija proizvoda	U potpunosti završeno re[enje, u skladu sa zahtevima korisnika. Finalni proizvod bi trebao biti u takvom stanju da je u potpunosti upotrebljiv od strane korisnika.
Materijali za podršku	Materijali koji treba da pomažu krajnjim korisnicima u korišćenju proizvoda treba da su u ovoj tački u potpunosti završeni.
Elementi implementacije	Elementi koji su se dorađivali u ovoj fazi bi trebali biti u potpunosti završeni i inkorporirani u finalni proizvod.
Test paket („smoke test")	Test paket ,koji je razvijen da utvrdi primenjivost, treba da je u potpunosti razvijen i sproveden nad finalnim proizvodom.
Upakovan proizvod za dalju prodaju	U slučaju kada se radi o proizvodu namenjenom nepoznatom kupcu, tj. širokoj prodaji, potrebno je proizvod upakovati i pripremiti ga za isporuku. U ovoj tačci proizvod mora da je u potpunosti spreman za konačnu isporuku.

Izvor: (Rational Team, 2010)

Aktivnosti faze tranzicije (Rational Team, 2010):

- priprema okruženja za iteraciju,
- otklanjanje grešaka u komponentama,
- planiranje raspoređivanja,
- izrada preostalih komponenti,
- integracija i testiranje,
- tekuće održavanje i podrška,
- izrada preostalih materijala za podršku,
- upravljanje testom za utvrđivanje prihvatljivosti,
- beta testiranje,
- završna priprema proizvoda za isporuku,

- planiranje naredne iteracije,
- zaključivanje projekta.

Sadržajnu dimenziju RUP-a čini šest osnovnih i tri pomoćne discipline, koje su sačinjene od seta aktivnosti, rezultata u formi artifakata i uloga koje su odgovorne za njihovu realizaciju (Kruchten, 2004):

1. disciplina poslovno modelovanje,
2. disciplina zahteva,
3. disciplina analize i dizajna,
4. disciplina implementacije,
5. disciplina testiranja,
6. disciplina raspoređivanja,
7. disciplina konfigurisanja i upravljanja promenama,
8. disciplina upravljanje projektom,
9. disciplina okruženja.

3.2.3 Analiza arhitekturalno značajnih aktivnosti i artifakata prema disciplinama RUP-a

Analiza RUP-a vršiće se u skladu sa predstavljenim modelima na slici 3.1 i slici 3.2, na osnovu kojih se može zaključiti, da se razvoj efektivnog i efikasnog arhitekturalnog rešenja zasniva na tesnoj vezi sledećih RUP disciplina: poslovno modelovanje - zahtevi - analiza i dizajn - implementacija.

3.2.3.1 Disciplina poslovno modelovanje

Disciplina poslovnog modelovanja obezbeđuje informacije o kontekstu i radnom okruženju organizacije, u kojoj će budući sistem biti implementiran. Informacije o načinu poslovanja, poslovnim ciljevima, pravilima i poslovnoj arhitekturi ciljne organizacije su od značaja za razvoj arhitekture sistema koji se razvija.

Tesna sprega između poslovne i softverske arhitekture najbolje je predstavljena i opisana Three Peaks modelom koji je predstavljen u prethodnom delu rada (Chandra & Bhattaram, 2003).

Fokus ove discipline je na proceni statusa ciljne organizacije i identifikovanju područja u njenom poslovanju na kojima je moguće izvršiti određena poboljšanja. Na osnovu ovih razultata potrebno je odabrati najadekvatniji scenario (od šest mogućih) prema kojem će se sprovesti dalji koraci poslovnog modelovanja.

U slučaju da se na osnovu izvršene procene ciljne organizacije utvrdi da ni jedan poslovni model nije neophodan u svojoj punoj razmeri, već da je dovoljan samo model domena, onda se poslovno modelovanje sprovodi prema koracima aktivnosti "Razvoj modela domena" - scenario #2. Model domena predstavlja podskup modela poslovne analize, koji obuhvata samo poslovne entitete tog modela. Pogodan je u slučaju razvoja softverskih rešenja koja za cilj imaju upravljanje i prikaz informacija (npr. sistem upravljanje narudžbinama ili bankarski sistem) (Rational Team, 2010).

Ako se utvdi da razvoj budućeg sistema neće implicirati velike promene u poslovnim procesima ciljne organizacije, dovoljno je predstaviti iste i na bazi njih identifikovati softverske zahteve. Koraci poslovnog modelovanja u tom slučaju odvijaju se prema scenariju #1. Dati scenario podrazumeva izradu organizacionog dijagrama, kojim se predstavlja postojeće poslovanje organizacije, kako bi se bolje razumeli poslovni zahtevi. Opis postojećih poslovnih procesa ciljne organizacije neophodan je i u slučaju implementacije potpuno novog sistema, da bi se sagladele mogućnosti njegove implementacije i adaptacije u organizaciju. Razvoj takvog sistema podrazumeva izradu seta modela kojima se opisuje trenutno poslovanje organizacije, ali podrazumeva i modifikaciju tih modela, u skladu sa ulogom koju će novi sistem imati u njemu (Rational Team, 2010).

Ukoliko implementacija novog sistema u organizaciji treba da dovede do poboljšanja ili reinženjeringa postojećih poslovnih procesa, neophodan je set modela koji predstavlja tekuće poslovanje, ali i set modela koji oslikava željeni način poslovanja organizacije. U ovom slučaju ključni artifakt je dokument poslovne arhitekture. Scenariji poslovnog modelovanja koje je moguće primenjivati u ovoj sistuaciji su #3, #4 i #6. Scenario #3 je poznat pod nazivom "Jedan posao mnogo sistema" i najviše se koristi u razvoju velikih, kompleksnih sistema. Podrazumeva izradu seta modela, pomoću kojih se identifikuju arhitekturalno značajni zahtevi, na čijim osnovama se dizajnira arhitektura sistema. Scenario #4 nosi naziv "Opšti poslovni model" i primeren je za situacije kada budući sistem treba da bude korišćen u nekoliko organizacija (npr. aplikacija podrške prodaje). Ovaj scenario poslovnog modelovanja usmeren je na analizi poslovanja ciljnih organizacija, a potom i na njihovo međusobno usklađivanje, kako bi se izbegli previše kompleksni sistemski zahtevi. Drugi način primene ovog scenaria podrazumeva takođe analizu poslovanja svake organizacije ponaosob, nakon čega se upravlja njihovim razlikama koje budući sistem treba da podrži. Scenario #6 – "Prepravka" predstavlja adekvatan izbor u slučaju reinženjeringa poslovanja ciljne organizacije. Scenario omogućuje identifikovanje i analizu postojećih poslovnih procesa ali i iznalaženje načina za njihovu rekonstrukciju u cilju ostvarivanja radikalnog poboljšanja poslovanja (Rational Team, 2010).

Kada budući sistem treba da podrži otpočinjanje potpuno novog poslovnog projekta, modeli koji opisuju trenutno poslovanje ne postoje. U tom slučaju preporučuje se primena scenaria #5 – "Novo poslovanje". Scenario se implementira prilikom razvoja nove linije poslovanja ciljne organizacije, novog poslovnog procesa ili nove organizacije. Poslovno modelovanje u ovim slučajevima ima za cilj da utvrdi izvodljivost nove poslovne ideje (Rational Team, 2010).

Aktivnosti koje se realizuju u okviru disciple poslovno modelovanje su sledeće (Rational Team, 2010):

1. Procena poslovnog statusa - ima za cilj da proceni status ciljne organizacije, identifikuje područja mogućeg poboljšanja i dokumentuje rezultate u okviru artifakata: procena ciljne organizacije i dokument poslovne arhitekture. Artifakt procena ciljne organizacije opisuje ljude, poslovne procese i alate koji se koriste u ciljnoj organizaciji. Nakon procene organizacije vrši se kategorizacija razvojnog projekta, radi odabira odgovarajućeg scenarija poslovnog modelovanja i izbora ključnih artifakata. Korak koji sledi nakon toga je izrada artifakta dokument poslovne vizije, u kojem se opisuje problem koji se rešava, sa jasno definisanim realnim ciljevima koje treba ostvariti poslovnim modelovanjem. U definisanju ciljeva poslovnog modelovanja potrebno je učešće i saglasnost svih stejkholdera, što se ostvaruje različitim tehnikama (npr. brainstorming i sl.). Na osnovu definisane poslovne vizije, pristupa se identifikovanju i dokumentovanju poslovnih pravila organizacije kao i poslovnih ciljeva, koji će u kasnijim iteracijama predstavljati osnovu za evaluaciju dokumentovanih poslovnih slučajeva upotrebe. Radi lakšeg rada na poslovnom modelovanju organizacije poželjno je formirati i tokom razvoja održavati, artifakt poslovni rečnik, u kojim se definiše jedinstvena terminologija u cilju lakše komunikacije stejkholdera i članova tima.

Zadatak analiza poslovne arhitekture, realizuje se samo u slučaju kada se poslovno modelovanje vrši s ciljem poboljšanja tekućeg poslovanja ili reinženjeringa poslovnih procesa ciljne organizacije. Artifakti koji dokumentuju rezultate ovog zadatka su: dokument poslovne arhitekture, model poslovne analize, model poslovnog raspoređivanja, model poslovnog dizajna, poslovni entiteti, poslovni radnici i poslovni sistem. Artifakt poslovna arhitektura sadrži opis tri različite, ali međusobno povezane arhitekture: poslovne arhitekture, aplikativne arhitekture i tehničke arhitekture. Poslovna arhitektura predstavlja opis značajnih aspekata organizacije: proizvoda i usluga, procesa, organizacije i lokacija. Aplikativna arhitektura fokusira se na opis softverskih aplikacija koje omogućavaju odvijanje poslovanja, uključujući opis načina korišćenja datih aplikacija i njihovu međusobnu povezanost. Tehnička arhitektura opisuje hardversku infrastrukturu koja omogućuje izvršavanje softverskih aplikacija.

Poslovna arhitektura upravlja aplikativnom arhitekturom, a aplikativna tehničkom, bez definisanog hijerarhijskog odnosa među njima. Naime ciljevi i ograničenja usmeravaju njihovu komunikaciju u jednom smeru: poslovna→aplikativna→tehnička, dok donete arhitekturalne odluke u obrnutom smeru: tehnička→aplikativna→poslovna (npr. neka poslovna ograničenja mogu biti namerno ignorisana, jer iziskuju ulaganja koja su veća od eventualnih negativnih efekata).

Projektovanje ovih arhitektura zahteva balansiranost i pravljenje kompromisa, kako bi se proizvelo rešenje koje optimalno zadovoljava njihove suprotstavljene zahteve. Sve tri arhitekture se moraju imati u vidu prilikom komunikacije sa stejkholderima, kako bi se sprečilo da odluke o jednoj arhitekturi, ne dovedu u pitanje ispravnost drugih arhitektura.

Poslovna arhitektura je kompleksna i teška za merenje, jer je stejkholderi sagledavaju iz različitih perspektiva. Stoga se njen opis vrši putem različitih pogleda, koji sačinjavaju artifakt dokument poslovne arhitekture. Svaki pogled opisuje jedan aspekt poslovanja, u obimu u kojem je on značajan sa aspekta poslovne arhitekture organizacije. Tipični pogledi poslovne arhitekture su:

- Tržišni pogled - opisuje ciljno tržište datog poslovanja kroz opis profila kupaca i proizvoda/usluga koji se nude na tržištu.
- Pogled poslovnih procesa - opisuje značajne poslovne ciljeve i ključne poslovne slučajeve upotrebe koji podržavaju opisane ciljeve.
- Organizacioni pogled - opisuje uloge i odgovornosti koje učestvuju u realizaciji poslovnih slučajeva upotrebe.
- Pogled domena - opisuje glavne poslovne koncepte i strukture informacija koje se koriste u poslovanju.
- Geografski pogled - opisuje distribuiranu organizacionu strukturu, funkcije i resurse putem fizičkih lokacija kao što su gradovi i države.
- Komunikacioni pogled - opisuje komunikacionu putanju unutar poslovanja.
- Pogled ljudskih resursa - opisuje profile zarada, podsticajne mehanizme, ključne karakteristike organizacione kulture, profile stručnosti i mehanizme edukacija i obuka.

2. Opis trenutnog poslovanja - ima za cilj razumevanje trenutnih ("as is") organizacijskih procesa i strukture. Aktivnost počinje procenom trenutnog stanja organizacije (npr. ljudi, procesa, alata, tržišta i sl.), kako bi se identifikovala područja problema i uvidele mogućnosti njihovog poboljšavanja. Nakon toga, u saradnji sa stejkholderima, postavljaju se ciljevi poslovnog modelovanja. Naredni korak podrazumeva identifikovanje poslovnih ciljeva i ključnih indikatori performansi, kao i uspostavljanje veze između strategijskih i operativnih ciljeva organizacije. Ova aktivnost podrazumeva definisanje ko će i na koji način biti u interakciji sa sistemom kao i izradu artifakta model poslovnih slučajeva upotrebe. Artifaktom poslovna pravila identifikuju se ključna poslovna pravila, a artifaktom rečnik definišu se poslovni termini. Analiza poslovne arhitekture vrši se samo u slučaju razvoja poslovanja organizacije.

3. Definisanje poslovanja - predstavlja opis budućeg ("to be") poslovanja i sastoji se od sledećih koraka:

- Identifikovanje poslovnih procesa - podrazumeva utvrđivanje granica organizacije, utvrđivanje terminologije i formiranje rečnika poslovnih termina, identifikovanje poslovnih ciljeva, izradu skice modela poslovnih slučajeva upotrebe, odabir poslovnih slučajeva upotrebe koji će biti detaljno opisani, opis poslovnih aktera, izradu dodatnih poslovnih specifikacija (koje sadrže opis zahteva koji nisu pokriveni slučajevima upotrebe), dokumentovanje poslovnih pravila i ažuriranje artifakta dokument poslovne arhitekture u cilju identifikovanja arhitekturalno značajnih poslovnih slučajeva upotrebe.

- Detaljano definisanje poslovnih procesa - podrazumeva detaljan opis poslovnih slučajeva upotrebe i načina na koji poslovni slučajevi upotrebe podržavaju poslovne ciljeve. Poželjno je identifikovati i veze između poslovnih slučajeva upotrebe kao i izvršiti njihovo grupisanje u pakete i kategorisanje prema njihovom prioritetu. Aktivnost se završava proverom i potvrdom artifakta model poslovnih slučajeva upotrebe.
- Dizajn realizacije poslovnih procesa - podrazumeva prvi korak u definisanju artifakta realizacija poslovnih slučajeva upotrebe budućeg poslovanja, koje se u zavisnosti od tipa projekta, može razvijati na osnovu izrađenih modela trenutnog poslovanja ili pak iz početka. Poslovna pravila koja se identifikuju pri izvršavanju ovog zadatka, dodaju se u već definisani artifakt poslovna pravila, kao i novi poslovni termini u izgrađeni artifakt rečnik. Takođe, neophodno je ažurirati i artifakt dokument poslovne arhitekture, sa identifikovanim arhitekturalnim pitanjima.
- Definisanje poslovnih operacija - počinje prikazom sistema, njegovog interfejsa i veza sa njegovim akterima, uključujući i spoljne izlazno/ulazne poslovne entitete koji se javljaju između sistema i poslovnih aktera. Potom se sprovodi korak analiza poslovne arhitekture (koja je gore opisana) kao i definisanje spolja vidljivih osobina sistema u terminima operacija.
- Definisanje uloga i odgovornosti - podrazumeva ažuriranje sledećih artifakata: poslovni akteri i poslovni entiteti, realizacija poslovnih slučajeva upotrebe, poslovna pravila.

4. Automatizacija procesa - ima za cilj identifikovanje delova poslovnih procesa koji trebaju biti automatizovani, potom zahteva budućeg sistema i načina adaptacije budućeg sistema u organizaciji.

5. Razvoj modela domena - je alternativna aktivnosti koja generiše artifakt model domena. Model domena opisuje proizvode, isporuke, poslovne entitete i događaje bitne za domen poslovanja.

U tabeli 3.5 je dat pregled ključnih artifakata discipline poslovno modelovanje.

Tabela 3.5 Ključni artifakti discipline poslovno modelovanje

KLJUČNI ARTIFAKTI DISCIPLINE POSLOVNO MODELOVANJE	SVRHA
Poslovni radnici	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena.
Dokument poslovne arhitekture	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena.
Poslovni entiteti	Obavezan u slučaju poslovnog modelovanja
Poslovni događaji	Obavezan u slučaju automatizacije određenih poslovnih procesa ili u slučaju modelovanja domena

Poslovni ciljevi	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena.
Rečnik	Poželjan artifakt
Model poslovne analize	Obavezan u slučaju poslovnog modelovanja
Poslovna pravila	Može ali ne mora postojati
Poslovni sistem	Može ali samo u slučaju velikih ili kompleksnih poslovnih modela
Poslovni slučajevi upotrebe	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena
Model poslovnih slučajeva upotrebe	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena.
Realizacija poslovnih slučajeva upotrebe	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena.
Poslovna vizija	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena
Poslovni radnici	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena
Dodatne poslovne specifikaciije	Trebao bi uvek postojati
Procena ciljne organizacije	Obavezan artifakt u slućaju reinženjeringa ili poboljšavanja poslovanja. Opcioni u slučaju modelovanja domena

Izvor: (Rational Team, 2010):

3.2.3.2 Disciplina zahtevi

Osnovna svrha discipline zahteva jeste prikupljanje istih od stejkholdera budućeg sistema i njihovo transformisanje u set softverskih zahteva. Ova disciplina je izuzetno važna sa aspekta razvoja arhitekture, jer obezbeđuje inpute na kojima se zasnivaju buduće arhitekturalne odluke. Međusobna povezanost zahteva i softverske arhitekture predstavljena je Tween Peaks modelom (Nuseibeh, 2001), koji je objašnjen u prethodnom delu rada.

Ciljevi koje je potrebno ispuniti kroz ovu disciplinu su sledeći (Shuja & Krebs, 2008):

- Uspostaviti saglasnost sa stejkholderima šta sistem treba da radi.
- Obezbediti razumevanje sistemskih zahteva od strane razvojnog tima.
- Definisati granice sistema.
- Obezbiditi osnove za planiranje tehničkih sadržaja iteracija.
- Obezbediti osnove za procenu troškova i vremena razvoja sistema.
- Definisati korisnički interfejs sa fokusom na potrebe i ciljeve korisnika.

Disciplinu zahtevi čini set aktivnosti koje se realizuju, kroz brojne zadatke, od strane uloga koje su odgovorne za njihovo izvršavanje (Rational Team, 2010):

1. Analiza problema - podrazumeva identifikovanje ključnih stejkholdera, granica i ograničenja budućeg sistema. Cilj je da svi učesnici projekta u potpunosti razumeju problem koji budući sistem treba da reši. U dostizanju ovog cilja važno je uspostaviti jedinstvenu terminologiju na projektu, koju treba kontinuirano dorađivati tokom trajanja projekta (artifakt rečnik). Najznačajniji atifakt ove aktivnosti je dokument vizije, gde se opisuje korisnički pogled na sistem koji se gradi.

Na osnovu utvrđenih korisničkih potreba, identifikuju se ključne karakteristike sistema, u čijem definisanju učestvuju i ključni stejkholderi. Identifikovanim karakteristikama se dodeljuju atributi priotiteta, na osnovu kojih će se definisati plan rada. Izradom modela slučajeva upotrebe, utvrđuju se granice budućeg sistema i identifikuju korisnici koji su u interakciji sa njim. Artifakt slučajevi upotrebe predstavlja funkcionalne zahteve sistema, koji uz dokumentovane nefunkcionalne zahteve i ograničenja (artifakt dodatne specifikacije) čine artifakt specifikacija softverskih zahteva.

Kroz ovu aktivnost generiše se i artifakt plan za upravljanje zahtevima, u cilju opisivanja načina njihovog organizovanja, dokumentovanja, merenja i kontrole promena. Pomenuti artifakt opisuje i način na koji se utvrđuju grupe sličnih zahteva, način dodeljivanja atributa zahtevima i način utvrđivanja sledljivosti zahteva.

2. Razumevanje korisničkih potreba - predstavljaju osnovu za definisanje karakteristika sistema, dokumentovanih artifaktom specifikacija softverskih zahteva.

3. Definisanje sistema - obuhvata detaljnu razradu inicijalnog seta zahteva, identifikovanih u prethodnom koraku, kao i ažuriranje sledećih dokumenata: vizija, slučajevu upotrebe, akteri, rečnika, plana upravljanja zahtevima i dodatne specifikacije.

4. Upravljanje obimom sistema - aktivnost se realizuju kroz ulogu sistem analitičara i softver arhitekte. Sistem analitičar realizuje zadatke: razvoj vizije i upravljanje zavisnostima, a softver arhitekta: utvrđivanje redosleda razvoja slučajeva upotrebe. Cilj aktivnosti je da se što jasnije preciziraju zahtevi i utvrdi plan njihovog razvoja po iteracijama.

Realizacija zadatka *razvoj vizije*, podrazumeva sledeće korake:

1. Sporazum o problemima koji će biti rešavani - podrazumeva kreiranje opšte prihvaćenih formulacija za sve identifikovane probleme. Tehnike koje se koriste za identifikaciju problema su: “brainstorming”, “fishbone” dijagrami i “pareto” dijagrami.

2. Identifikovanje stejkholdera sistema - realizuje se putem intervjua, sa fokusom na pitanja ko će koristiti sistem, ko kupuje sistem, ko odobrava njegovo korišćenje i sl.

3. Definisanje granica sistema - koje predstavljaju interfejs preko kojeg korisnici iz realnog okruženja komuniciraju sa datim sistemom.

4. Identifikovanje ograničenja koja će biti nametnuta budućem sistemu - potiču iz realnog okruženja i mogu biti različitog porekla: politička, ekonomska, pravna, tehnička, funkcionalna, itd.

5. Formulisanje izjave o problemima - podrazumeva opis uticaja svakog identifikovanog problema na stejkholdere i sam sistem, kao i generisanje liste koristi usled uspešnog rešavanja datog problema.

6. Definisanje karakteristika sistema - vrši se na osnovu prethodno sačinjene liste koristi.

7. Provera dokumenta vizije - podrazumeva korišćenje ček liste koja sadrži neophodne elemente dokumenta.

Zadatak *upravljanje zavisnostima* - podrazumeva sagledavanje zavisnosti između zahteva u cilju upravljanja obimom projekta i promenama u zahtevima. Realizuje se kroz sledeće korake:

1. Dodeljivanje atributa - podrazumeva dodeljivanje vrednosti atributa za pojedine klase zahteva. Klasu zahteva sačinjava grupa sličnih identifikovanih zahteva. Atributi koji se dodeljuju datoj klasi mogu oslikavati rizik, uticaj na arhitekturu i slično.

2. Uspostavljanje i potvrđivanje sledljivosti - podrazumeva uspostavljanje veza između zahteva i rezultata koji se ostvaruju tokom trajanja projekta. Primer ovakve veze je odnos između zahteva i funkcionalnosti budućeg sistema.

3. Upravljanje promenama zahteva – podrazumeva mogućnost promenljivosti zahteva u skladu sa artifaktom plan upravljanja zahtevima.

Zadatak *utvrđivanje redosleda razvoja slučajeva upotrebe* realizuje softver arhitekta. Da bi se sa stanovišta sistema kao celine utvrdio adekvatan redosled razvoja slučajeva upotrebe, neophodno je izvršiti procenu značaja svakog pojedinačnog slučaja upotrebe, sa aspekta arhitekture. Koraci koji se realizuju pri izvršavanju ovog zadatka su sledeći:

1. Utvrđivanje redosleda razvoja slučajeva upotrebe i scenarija - Softver arhitekta predlaže tehničke sadržaje i redosled iteracija, izborom određenog broja scenarija i slučajeva upotrebe koji će se analizirati i dizajnirati. Tehnički predlog se kompletira i sa aspekta članova razvojnog tima: dostupnost kadrova, dostupnost alata i sl. Izbor scenarija i

slučajeva upotrebe vrši se s aspekta njihove značajnosti za razvoj arhitekture. Izbor je zasnovan na sledećim ključnim faktorima:

- Korisnošću koju stejkholderi imaju od scenarija: kritičan/važan/koristan.
- Uticajem scenarija na arhitekturu: nema/proteže se/menja. Mogu postojati slučajevi upotrebe ocenjeni kao kritični, a da pri tom nemaju uticaj na arhitekturu, kao i slučajevi upotrebe niske korisnosti ali velikog uticaja na arhitekturu.
- Rizicima koje treba ublažiti (performansa, dostupnost proizvoda, pogodnost komponenti).
- Drugim taktičkim ciljevima ili ograničenjima.

2. Dokumentovanje pogleda slučajevi upotrebe - Dokumentovanje se vrši u okviru artifakta dokument softverske arhitekture (sekcija dokumenta: pogled slučajeva upotrebe).

3. Evaluacija rezultata - Putem ček liste proverava se da li su svi identifikovani slučajevi upotrebe na liniji cilja, bez ulaženja u njihovu detaljnu analizu. Ulazni artifkti za realizaciju ovog zadatka su: dokument softverske arhitekture, lista rizika i plan iteracija. Rezultati ove aktivnosti utiču na ažuriranje sledećih artifakata: dokument softverske arhitekture, softverski zahtevi i atributi zahteva.

5. Doradivanje definicije sistema - je aktivnost koju sprovode stručnjak za specifikaciju zahteva i sistem analitičar. Cilj aktivnosti je detaljna razrada funkcionalnih i nefunkcionalnih zahteva sistema, na osnovu kojih definiše budžet i vreme trajanja projekta. Razrada zahteva podrazumeva da zahtevi u potpunosti budu jasni dizajnerima, testerima i stručnjacima za dokumentovanje**.**

6. Upravljenje promenama zahteva - sprovodi sistem analitičar u cilju analize uticaja promenjenih korisničkih zahteva na specifikovane softverske zahteve. Ova aktivnost implicira promene na sledećim dokumentima: model slučajeva upotrebe, atributi zahteva i sledljivost zahteva.

Ključni artifakti koji nastaju prilikom realizacije opisanih aktivnosti u okviru discipline zahtevi, predstavljeni su u tabeli 3.6.

Tabela 3.6 Artifakti discipline zahtevi

KLJUČNI ARTIFAKTI DISCIPLINE ZAHTEVI	SVRHA
Model slučaja upotrebe (artifakt akteri, artifakt slučaj upotrebe, artifakt paketi slučaja upotrebe)	Korististi se za definisanje funkcionalnih zahteva. Metod koji se preporučuje za gotovo sve projekte
Skice ekrana budućeg softvera (storyboard)	Korististi se za identifikovanje funkcionalnih zahteva koji nisu razumljivi, kao i za njihov detaljniji opis. Ova tehnika je opciona.

Rečnik podataka	Osigurava da svi učesnici projekta koriste zajedničku terminologiju. Preporučuje se za sve projekte.
Plan upravljanja zahtevima	Opisuje mehanizme za merenje, izveštavanje i kontrolu promena softverskih zahteva. Preporučuje se u slučaju projekata gde je upravljanje zahtevima kompleksno
Atributi zahteva	Osigurava da su svi zahtevi adekvatno kategorisini prema važnosti (prioritetu), kao i njihovo praćenje, sledljivost. Preporučuje se za projekte sa većim obimom zahteva.
Specifikacija softverskih zahteva	Kolektira sve softverske zahteve u cilju njihove ocene i potvrde od strane korisnika. Opcioni dokument
Zahtevi stejkholdera	Obuhvata sve korisničke zahteve kao i način na koji će biti ispunjeni. Preporučuje se na svim projektima.
Dodatne specifikacije	Opisuje nefunkcionalne zahteve sistema. Preporučuje se za sve projekte.
Vizija	Identifikuje zahteve visokog nivoa i ograničenja dizajna. Preporučuje se za sve projekte

Izvor: (Rational Team, 2010)

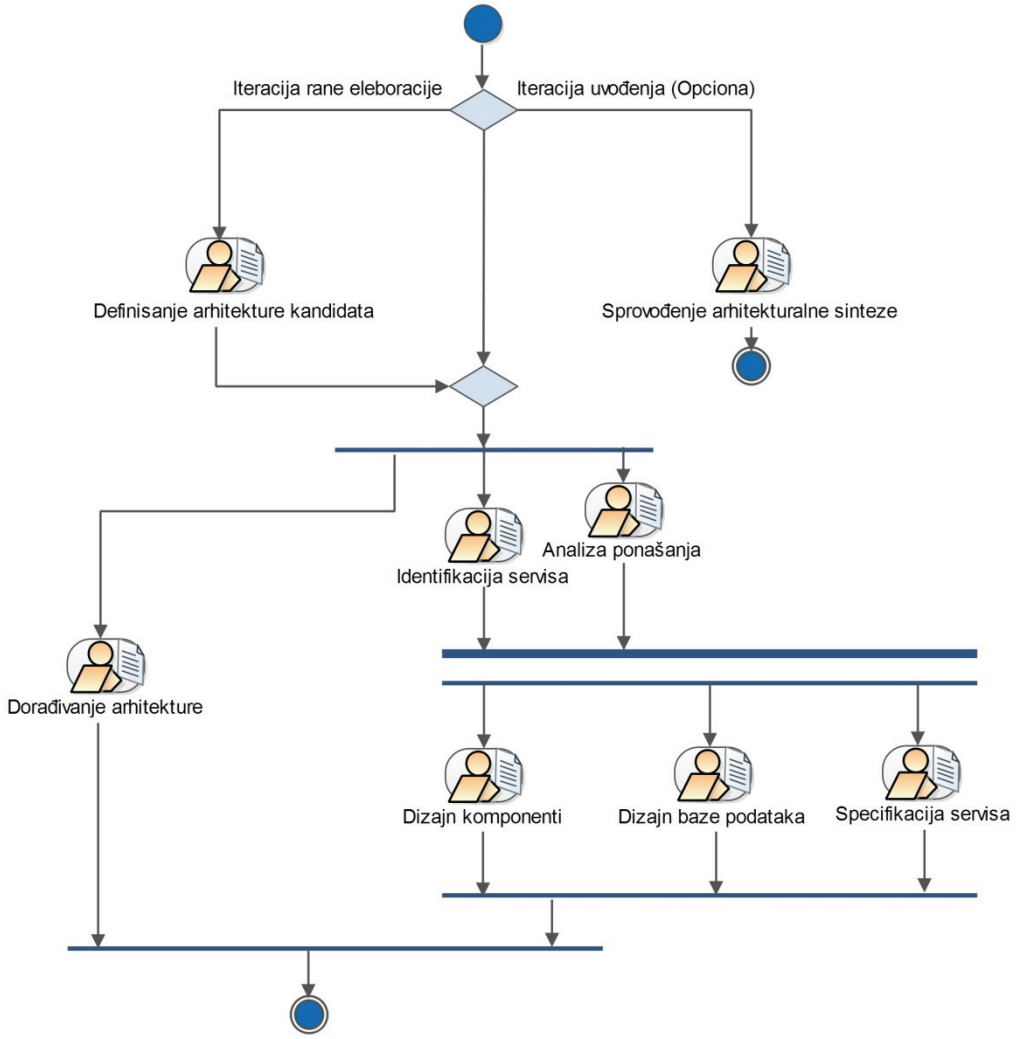
3.2.3.3 Disciplina analiza i dizajn

Disciplina analiza i dizajn, u fazi uvođenja, bavi se utvrđivanjem izvodljivosti razvojnog projekta i procenom potencijalnih tehnologija za buduće softversko rešenje. U fazi rane elaboracije, fokus ove discipline je na kreiranju inicijalnog arhitekturalnog rešenja, a ukoliko arhitektura već postoji, fokus je na doradi postojeće. Dizajn baze podataka odvija se paralelno sa razvojem komponenti sistema.

Disciplina analiza sistema bavi se prikupljanjem sistemskih zahteva i njihovom transformacijom u klase i podsisteme. Disciplina dizajn sistema vši prilagođavanje klasa i podsistema (identifikovanih u okviru discipline analiza sistema) ograničenjima koja nameću nefunkcionalni zahtevi i izvršno okruženje (Kroll & Kruchten, 2003).

Na slici 3.12 prikazane su sve aktivnosti koje se realizuju u okviru discipline anaaliza i dizajn sistema (Shuja & Krebs, 2008):

1. sprovođenje arhitekturalne sinteze,
2. definisanje arhitekture kandidata,
3. identifikacija servisa,
4. dorađivanje arhitekture,
5. analiza ponašanja,
6. dizajn komponenti,
7. dizajn baze podataka,
8. specifikacija servisa.



Slika 3.12 Tok aktivnosti u okviru discipline analiza i dizajn
Izvor: (Rational Team, 2010)

1. Sprovođenje arhitekturalne sinteze

Uloge koje su zadužene za sprovođenje ove aktivnosti su: sistem analitičar i softver arhitekta. Sistem analitičar definiše kontekst budućeg sistema, odnosno kolaboraciju sistema i njegovih aktera, kao i spoljne ulazno/izlazne entitete koji teku između njih. Ključni artifakt je dijagram konteksta koji se izrađuje na osnovu modela slučajeva upotrebe. Softver arhitekta na ovoj aktivnosti ostvaruje sledeće zadatke (Rational Team, 2010):

1. Sprovođenje arhitekturalne analize - podrazumeva identifikovanje arhitekturalnih tehnika i potencijalnih arhitekturalnih rešenja, oslanjanjem na iskustva prethodnih projekta. Rezultati ove aktivnosti dovode do izmena sledećih artifakata: model rasporeda, opis softverske arhitekture, analiza klasa, model analize.

Koraci koji se realizuju tokom ovog zadatka su sledeći:

- Razvoj arhitekturalnog pregleda - izvršava se rano u fazi uvođenja i fokusira se na donošenje odluka koje se tiču fizičke i logičke arhitekture i nefunkcionalnih zahteva sistema. Nastaje saradnjom softver arhitekte i sponzora projekta. Konceptualno, ovaj pregled ilustruje suštinu predloženog arhitekturalnog rešenja i glavne gradivne blokove (elemente) od kojih će biti izgrađena arhitektura sistema. Predloženo arhitekturalno rešenje vrlo često je zasnovano na referentnoj arhitekturi, arhitekturalnim obrascima i drugim već postojećim arhitekturalnim rešenjima. Uobičajena je izrada i artifakta model rasporedjivanja, iz razloga što pruža informacije o produkcionom okruženju sistema (postojeći dizajn logičke i fizičke mreže, postojeći standardi, postojeći podaci i dizajn baze podataka i sl.). Ove informacije umnogome utiču na/ograničavaju izbor arhitekturalnog rešenja.
- Analiza raspoloživih sredstava - podrazumeva identifikovanje postojeće softverske imovine (postojećih softverskih komponenti, radnih okvira i sl.), vrednovanje njene upotrebljivosti u konkretnom projektu i odabir odgovarajuće.
- Definisanje organizacije inicijalnih podsistema - podrazumeva kreiranje artifakta model dizajna, u kojem je fokus na dva sloja: aplikativni i poslovni.
- Identifikovanje ključnih apstrakcija - podrazumeva definisanje/ažuriranje sledećih artifakata: klase analize, rečnik, model domena i model poslovne analize.
- Identifikovanje stereotipnih interakcija - ima za cilj prikaz ključnih aktivnosti sistema. Dokumentuju se u okviru artifakta realizacija slučajeva upotrebe.
- Razvoj modela raspoređivanja - podrazumeva njegovu inicijalnu postavku, bez detalja. Model pretstavlja mapiranje komponenti i podataka sistema sa fizičkim elementima, kao što su čvorovi. Značajan iz više razloga: omogućava sagledavanje održivosti implementacije budućeg sistema; olakšava razumevanje geografske distribuiranosti i kompleksnosti sistema; predstavlja osnovu za ranu procenu troškova i napora.
- Identifikovanje mehanizama analize - podrazumeva odabir obrazaca koji konstituišu zajedničko rešenje za zajednički problem. Primeri mehanizama analize su: perzistentnost klasa, komunikacija procesa, usmeravanje poruka, kontrola procesa i sinhronizacija, upravljanje transakcijama, razmena informacija, sigurnost, redudancija, izveštavanje o greškama, konverzija formata.
- Sagledavanje rezultata.

2. Konstruisanje arhitekturalnog POC-a

Aktivnost se fokusira na opis izgrađenog arhitekturalnog POC-a u treminima identifikovanih arhitekturalno značajnih zahteva, rizika i softverske imovine koju poseduje ciljna organizacija. Rezultati ove aktivnosti se dokumentuju artifaktom arhitekturalni POC.

3. Procena upotrebljivosti arhitekturalnog POC-a

Ima za cilj da opiše način na koji će se sprovoditi evaluacija arhitekturalnog POC-a, kao i da definiše tačne kriterijume evaluacije. Dobijeni rezultati se dokumentuju artifaktom referenta arhitektura.

4. Definisanje konteksta sistema

Osnovnu ulogu u ovom zadatku ima sistem analitičar, a dodatnu softver arhitekta. Kroz ovu aktivnost se vrši izrada artifakta konteksni dijagram, koji na visokom nivou apstrakcije prikazuje odnos između sistema i aktera koji intereaguju sa sistemom. Artifakti koji se izvršavanjem ove aktivnosti moraju ažuriratisu: model analize i model slučajeva upotrebe.

2. Definisanje arhitekture kandidata

Svrha ove arhitekturalne aktivnosti svodi se na sledeće razultate (Kruchten, 2004):

- Kreiranje inicijalnog nacrta arhitekture sistema, definisanjem sledećih segmenata: inicijalnog seta arhitekturalno značajnih elemenata; inicijalnog seta mehanizama analize; inicajlnih slojeva i organizacije sistema; realizacija slučajeva upotrebe za aktuelnu iteraciju.
- Identifikovanje klasa analize, iz arhitekturalno značajnih slučajeva upotrebe.
- Ažuriranje artifakta realizacija slučajeva upotrebe.

Aktivnost sprovode softver arhitekta, sistem analitičar i dizajner, sa ciljem da se postavi inicijalna skica softverske arhitekture budućeg sistema. Aktivnost se realizuje kroz izvršavanje sledećih zadataka (Rational Team, 2010):

1. Arhitekturalna analiza

Ima za cilj da definiše predlog softverske arhitekture i identifikuje arhitekturalne tehnike koje će biti korišćene u sistemu. Sprovodi je softver arhitekta i to kroz realizaciju sledećih podaktivnosti:

- Razvoj opšteg pregleda arhitekture - realizuje se u u fazi uvođenja. Nastaje na osnovu odluka, dokumentovanih artifaktom vizija sistema, i odluka vezanih za fizičku i logičku arhitekturu sistema. Opšti pregled arhitekture razvija softver arhitekta u saradnji sa sponzorom projekta, najčešće u obliku neformalnih dijagrama.
- Identifikovanje postojeće softverske imovine organizacije - podrazumeva donošenje odluke da li postojeća softverska rešenja, komponente i dr. mogu biti korišćena na razvoju budućeg sistema.
- Definisanje organizacije podsistema - predstavljati nivo dizajna sistema na najvišem nivou apstrakcije.
- Identifikovanje ključnih apstrakcija dizajna - podrazumeva definisanje inicijalnih elemenata dizajna, koji će se tokom konstrukcije sistema modifikovati.
- Identifikovanje interakcija između ključnih apstrakcija - podrazumeva definisanje načina komunikacije između elemenata dizajna.
- Dorađivanje inicijalno postavljenog modela raspoređivanja.
- Provera dobijenih rezultata - s ciljem identifikovanja eventualnih nekonzistentnosti.

Artifakti koji dokumentuju rezultate ovog koraka su: klase analize, model analize, model raspoređivanja, dokument softverske arhitekture.

2. Definisanje konteksta sistema

Primarnu ulogu u realizaciji ovog zadatka ima sistem analitičar, a dodatnu, softver arhitekta. Fokus zadatka je na izradi artifakta dijagram konteksta sistema, koji treba da prikaže odnos sistema i aktera koji sarađuju sa sistemom.

3. Detaljna analiza slučajeva upotrebe

Sprovodi se od strane dizajnera, sa ciljem utvrđivanja klasa sistema,kao i operacija i atributa klasa. Rezultati aktivnosti ažuriraju sledeće artifakte: dokument softverske arhitekture, klase analize, model analize, realizacija slučajeva upotrebe.

4. Analiza operacija

Aktivnost je fokusirana na transformaciju opisa ponašanja sistema u grubu strukturu sistema. Rezultati se uključuju u sledeće artifakte: model analize, operacije, realizacija operacija, model slučajeva upotrebe.

5. Identifikovanje obrazaca za sigurnost

Arhitekta za sigurnost zadužen je da identifikuje i odabere ključne obrasce, koji će osigurati neophodni nivo sigurnosti sistema. Odabrani obrasci se dorađuju u kasnijim razvojnim fazama, shodno odabranoj tehnologiji i platformi budućeg sistema. Rezultati ove aktivnosti postaju sastavni deo artifakta dokument softverske arhitekture.

3. Identifikacija servisa

Ovaj korak podrazumeva realizaciju brojnih aktivnosti:

1. Dekompozicija domena
   - Analiza funkcionalnog područja
   - Dorađivanje poslovnih slučajeva upotrebe
   - Analiza poslovnih procesa
   - Analiza poslovnih slučajeva upotrebe (servisno orijentisane arhitekture, u nastavku SOA)
2. Modelovanje cilj-servis
   - Identifikovanje poslovnih ciljeva i ključnih indikatora performanse (u nastavku KPIs)
   - Identifikovanje i pridruživanje servisa ciljevima

3. Analiza postojeće imovine
   - Analiza postojeće imovine
   - Analiza modela podataka
   - Analiza poslovnih pravila
   - Izrada arhitekturalnog POC-a (SOA)

ali će u nastavku biti opisane samo one koje su značajne sa aspekta razvoja softverske arhitekture (Rational Team, 2010):

1. Analiza poslovnih procesa

Realizacija ove aktivnosti vrši se na osnovu artifakta model poslovnih procesa, a kao rezultat proizilazi set potencijalnih servisa i njihovih međusobnih zavisnosti i načina komunikacije. Cilj aktivnosti je definisanje servisa i strukture servisno orijentisanih rešenja. Primarnu ulogu u ovoj aktivnosti ima dizajner, a softver arhitekta vrši dodatno ažuriranje artifakta specifikacija servisa. Ovaj artifakt obezbeđuje i strukturalnu i bihejvioralnu specifikaciju određenog servisa. Može sadržavati i set propisa kojima se reguliše pristup nekom servisu ili opis načina upotrebe servisa (u vidu skupa operacija koje pruža dati servis putem svog interfejsa). Ova specifikacija predstavlja vrstu ugovora između klijenta servisa i implementatora servisa; klijent razume kako da komunicira sa servisom, a implementator razume ponašanje koje se očekuje u implementaciji. Ključni artifakt ove aktivnosti je model servisa, koji sadrži ključne elemente SOA arhitekture i detalje vezane za servise.

2. Analiza poslovnih slučajeva upotrebe

Input za realizaciju ovog zadatka predstavlja artifakt realizacija poslovnih slučajeva upotrebe, na osnovu kojeg se identifikuje seta potencijalnih servisa (kandidati). Servisi mogu naknadno biti dorađivani. Izlazni artifakt ovog zadatka je model servisa.

3. Analiza modela podataka

Zadatak takođe ima za cilj identifikovanje seta potencijalnih servisa, koji mogu biti dorađivani tokom razvoja. Rezultati ovog zadatka dokumentuju se u okviru artifakta model servisa.

4. Analiza poslovnih pravila

Zadatak takođe ima za cilj identifikovanje seta potencijalnih servisa, koji mogu biti dorađivani tokom razvoja. Rezultati ovog zadatka dokumentuju se u okviru artifakta model servisa.

5. Izrada arhitekturalnog POC- a

Cilj zadatka je definisanje arhitekturalnog POC-a za SOA rešenje, na osnovu postojećih arhitekturalnih zahteva i profila rizika. Izlazni artifakt je arhitekturalni POC.

4. Dorađivanje arhitekture

Predstavlja tačku prelaza iz discipline analize u disciplinu dizajn sistema, što podrazumeva identifikovanje odgovarajućih elemenata i mehanizama dizajna, na osnovu odgovarajućih elemenata i mehanizama analize (Shuja & Krebs, 2008).

U sprovođenju ovoga koraka učestvuju softver arhitekta, dizajner i recenzent. Najviše odgovornosti u ovome koraku ima softver arhitekta, a njegove obaveze su vezane za sprovođenje sledećih aktivnosti (Rational Team, 2010):

1. Identifikacija mehanizama dizajna

Pored identifikacije mehanizama dizajna, softver arhitekta, je odgovoran i za njihovu proveru, u smislu da li ispunjavaju svoj cilj. Dokumentovanje i detaljno obrazlaganje svrhe odabranih mehanizama, vrši se u dokumentu dizajn softverske arhitekture. Ulazni artifakti za realizaciju ove aktivnosti su: klase analize i model servisa (ukoliko je u pitanju SOA arhitektura).

2. Identifikacija elemenata dizajna

Identifikovane klase analize transformišu se u različite elemente dizajna. Česta je situacija da jednu klasu analize implementira čak više elemenata dizajna:

- Klase dizajna - set detaljno struktuiranih odgovornosti, operacija i atributa.
- Podsistemi - najčešće nastaju kada klase analize imaju kompleksno ponašanje koje ne može biti odgovornost samo jedne klase dizajna. U tom slučaju se odgovornost raspodeljuje na više klasa, koje se grupišu u podsistem. Podsistem vrši enkapsulaciju kolaboracije ovih klasa dizajna. Podsistemi mogu nastati i usled grupisanja klasa prema njihovoj sličnosti izmena (čime se lokalizuje uticaj budućih promena); usled sprovođenja određenih pravila vidljivosti (npr.pravilo o definisanju granica između više slojeva u klijent-server aplikaciji); i sl. Podsisistemi su složeni gradivni elementi, koji pored klasa mogu sadržavati i druge podsisteme. Podsistemi se koriste za predstavljanje izvršnog proizvoda razvojnog tima, kao integralne jedinice funkcionalnosti.
- Aktivne klase - omogućavaju konkurentnost u radu sistema. Najčešće se koriste u kompoziciji sa klasama, koje se nazivaju pasivne. Aktivne klase koordiniraju i pokreću ponašanje pasivnih klasa i ovakva kompozicija je preporučljiva za modelovanje kompleksnog ponašanja sistema. Aktivne klase obično predstavljaju neki eksterni objekat, koji se koristiti konkurentno. Mogu se koristititi i za predstavljanje eksternih fizičkih uređaja, povezanih sa računarom, jer su ovi fizički entiteti sami po sebi konkurentni. Takođe, aktivnim klasa moguće je predstaviti i logički konkurentne aktivnosti (npr. finansijske transakcije, telefonski poziv) kao i resurse, koje koristi više istovremenih aktivnosti.
- Interfejs - predstavlja apstraktnu deklaraciju odgovornosti, koje se dodeljuju klasi ili podsistemu. Interfejs specifikuju set ponašanja i omogućava interoperabilnost sastavnih delova sistema.

Pored opisanih osnovnih elemenata dizajna, u slučaju sistema koji se izvršavaju u realnom vremenu, potrebno je identifikovati i sledeće elemente:

- Događaje - predstavljaju dešavanja u prostoru i vremenu i zahtevaju neki odgovor, akciju sistema. Događaji i njihove karakteristike koriste se u cilju identifikovanje elemenata dizajna koji će upravljati njima (npr. aktivne klase). Inicijalna lista spoljnjih događaja može se sačiniti na osnovu artifakta model slučajeva upotrebe tj. na osnovu interakcije aktera sa slučajevima upotrebe. Interni događaji mogu se specifikovati na osnovu opisa tokova

slučajeva upotrebe. Specifikacija karakteristika nekog događaja naročito je važna za modelovanje konkurentnosti i/ili asinhronog slanja poruka.

- Signale - predstavljaju asinhronu komunikaciju događaja. Izražavaju podatke koje oni nose ili veze između signala (npr. generalizacija).
- Kapsule - koriste se umesto aktivnih klasa, jer nude jaču semantiku za razvoj konkurentnih aplikacija. Poseduju neke osobine klase, a neke podsistema tj. predstavljaju enkapsulirane kolaboracije klasa koje zajedno predstavljaju nit kontrole u sistemu. Od podsistema se razlikuju jer su rezultat rada jednog dizajnera dok je podsistem odgovornost razvojnog tima programera. Takođe, podsistem može sadržavati kapsule.
- Protokole - za precizno definisanje poruka koje mogu biti poslate i primljene na portu kapsule. Predstavljaju vstu interfejsa, koje određuje ponašanje kapsule. Koriste se uglavnom za definisanje asinhrone komunikacije zasnovane na porukama signala.

Ulazni artifakti za realizaciju ove aktivnosti su: klase analize i model servisa. Izlazni artifakti koji sumiraju rezultate (dizajn podsistema, dizajn paketa, dizajn klasa, signale, događaje, interfejs, kapsule, protokol) su: model servisa i servisna komponenta.

3. Inkorporiranje postojećih elemenata dizajna

Aktivnost se sprovodi prema nekim od sledećih koraka:

- Traženje postojećih podsistema ili komponenti koji nude sličan interfejs. Svodi se na upoređivanje svakog identifikovanog interfejsa sa interfejsima postojećih podsistema ili komponenti. Prvo se pronalaze slična ponašanja i vraćene vrednosti, a onda se razmatraju parametri.
- Izmena novo identifikovanih interfejsa u cilju njihovog unapređenja. Ukoliko postoji mogućnost vrši se izmena na interfejsu kandidatu (novo identifikovanom), kako bi se poboljšala njegova usklađenost sa postojećim interfejsom. Promene podrazumevaju preuređenje ili dodavanje parametara interfejsu kandidatu, a potom deljenje datog interfejsa na nekoliko interfejsa, od kojih jedan ili više odgovara onima iz postojeće komponente, a ostali obezbeđuju nova ponašanja sistema.
- Zamena interfejsa kandidata sa postojećim u slučaju identifikovanih poklapanja. Nakon pojednostavljivanja interfejsa i njegovog faktoringa, ukoliko se identifikuje interfejs koji se poklapa sa već postojećim interfejsom postojećih komponenti, potrebno je eliminisati interfejs kandidata i koristiti postojeći.
- Povezivanje postojećih komponenti sa podsistemima kandidatima. Podrazumeva faktorizaciju podsistema na komponente koje su korišćene, kako bi se identifikovale one koje mogu da zadovolje određeno ponašanje sistema koji se gradi. Tamo gde podsistem kandidat može biti realizovan postojećim komponentama, potrebno je ispratiti dizajn datog podsistema i komponente u Modelu Implementacije.
- Izgradnja podsistema ponovnom upotrebom postojećih komponenti, podrazumeva i razmatranje mehanizama dizajna koji su povezani sa datim podsistemom, jer je moguće da

zahtevi performanse ili sigurnosti diskvalifikuju komponentu iz ponovne upotrebe i pored njene savršene povezanosti sa operativnim potpisima.

Artifakti koji se koriste za realizaciju ove aktivnosti su: dizajn softverske arhitekture i model servisa. Rezultate aktivnosti dokumentuju sledeći izlazni artifakti: dokument softverske arhitekture (podsistemi dizajna, paketi dizajna, klase dizajna, interfejs), model servisa i servisna komponenta.

4. Uspostavljanje strukture implementacionog modela

Ova aktivnost predstavlja tačku prelaska iz dizajna ka implementaciji, te je potrebno uvideti da li paketi dizajna imaju odgovarajuće podsisteme implementacije. Ovi podsistemi će sadžavati jedan ili više direktorijuma i fajlova (artifakt elementi implementacije) potrebnih za implementaciju odgovarajućih elemenata dizajna. Transformisanje modela dizajna u model implementacije se može menjati, jer je svaki podsistem implementacije alociran na određeni sloj u arhitekturi.Informacije koje su neophodne za realizaciju ove aktivnosti sadržane su u artifaktu. Izlazni artifakti su: dokument softverske arhitekture, model implementacije i podsistemi implementacije.

5. Opis radne arhitekture

U slučaju razvoja distribuiranog sistema, ključni koncepti koje je potrebno elaborirati u ovom koraku su: koncept procesa, zadataka, niti i brojna druga pitanja karakteristična za paralelizam u radu sistema. Svi ovi aspekti opisuju se u artifaktu dokument softverske arhitekture (sekcija procesni pogled i pogled raspoređivanja). Pogled raspoređivanja definiše mrežnu konfiguraciju i alokaciju procesa na čvorove u mreži. Procesna arhitektura definiše set aktivnih klasa i objekata kao i njihovu povezanost sa nitima operativnog sistema i procesima. Aktivni objekti predstavljaju niti koje se istovremeno izvršavaju u sistemu, dok niti predstavljaju zadatke koji se istovremeno izvršavaju u okviru jednog procesa. Instrukcije koje sačinjavaju jednu nit se izvršavaju istovremeno sa instrukcijama druge niti, bilo da je u pitanju više procesora ili samo jedan, pa je paralelizam u radu prividan. Opis radne arhitekture svodi se na sledeće korake:

- analiza konkurentnih zahteva,
- identifikovanje procesa i niti,
- identifikovanje mehanizama komunikacije između procesa,
- alokacija resursa koordinacije između procesa,
- identifikovanje životnog ciklusa procesa,
- mapiranje procesa ka implementacionom okruženju,
- raspodela elemenata modela dizajna između identifikovanih procesa.

U sistemima koji funkcionišu u realnom vremenu, modelovanje istovremenosti u radu, vrši se u okviru artifakta kapsule. Kapsule predstavljaju enkapsulirane kolaboracije klasa, koje

zajedno predstavljaju nit kontrole u sistemu. Karakteriše ih bogatija semantika, jer poseduju neke od osobina i aktivnih klasa i podsistemima.

Ulazni artifakti za realizaciju ove aktivnosti je dokument softverske arhitekture, koji nakon modifikovanja, predstavljaju ujedno i izlazni artifakt.

6. Opis distribucije
Ovaj zadatak definiše arhitekturu u terminima raspoređivanja elemenata sistema na fizičke čvorove i njihove interkonekcije. Inicijalni model raspoređivanja definisan je već tokom faze arhitekturalne analize i u ovom koraku se samo dorađuje. Ažurirani model je sastavni deo artifakta dokument softverske arhitekture tj. njegove sekcije pogled raspoređivanja. Koraci koji se sprovode prilikom realizacije ove aktivnosti su sledeći: analiza distribuiranih zahteva, definisanje mrežne konfiguracije, alokacija sistemskih elemenata na čvorove.

5. Analiza ponašanja

Svrha realizacije ove aktivnosti je transformisanje opisanog ponašanja sistema (u artifaktu slučajevi upotrebe) u set elemenata dizajna. Zadaci koji se realizuju u okviru ove akrivnosti su (Rational Team, 2010):

1. Analiza slučajeva upotrebe
Aktivnost opisuje kako na osnovu artifakta slučaj upotrebe izraditi artifakt realizacija slučaja upotrebe. Artifakt realizacija slučaja upotrebe sadrži objekte i njihove kolaboracije. Proces otpočinje identifikovanjem objekata analize, dodeljivanjem odgovornosti objektima (da bi se sagladalo kako zajedno ostvaruju funkcionalnost datog slučaja upotrebe), detaljnom analizom klasa i njihovih veza (da bi se utvrdilo da li dolazi do preklapanja funkcionalnosti klasa i da li njihove veze imaju smisla), definisanjem redosleda izvršavanja klasa i načina njihove komunikacije.

2. Identifikovanje elemenata dizajna
Podrazumeva definisanje elemenata dizajna: klase dizajna, aktivne klase dizajna, interfejse, događaje, signale, protokole, podsisteme (detaljno opisano u okviru aktivnosti dorađivanje arhitekture).

3. Analiza operacija
Aktivnost se fokusira na transformisanje opisanog ponašanja sistema u strukturu sistema.

4. Dizajn korisničkog interfejsa
Aktivnost se bavi dizajnom grafički orijentisanog interfejsa, sa akcentom na njegovu upotrebljivost. Podrazumeva identifikovanje primarnih prozora korisničkog interfejsa na osnovu analize zahteva (slučajeva upotrebe).

5. Prototip korisničkog interfejsa
Podrazumeva izradu prototipa korisničkog interfejsa, sa namerom da se utvrdi da li je korisnički interfejs u skladu sa funcionalnim zahtevima i zahtevima upotrebljivosti.

6. Pregled dizajna
Definiše kako izvršiti pregled dizajna i sprovesti analizu dobijenih rezultata.

6. Dizajn komponenti

Realizacijom aktivnosti dizajn komponenti, potrebno je ostvariti sledeće ciljeve: doraditi definisane elemente dizajna; ažurirati artifakt realizacije slučajeva upotrebe (novoidentifikovanim elementima dizajna) i izvršiti pregled dizajna. Uglavnom sve zadatke realizuje dizajner i stoga oni neće biti detaljno opisivani (Rational Team, 2010):

1. Dizajn slučaja upotrebe
Fokus zadatka je na opisu ponašanja sistema tj. na kolaboraciji i interakciji sistema.

2. Dizajn podsistema
Fokus je na opisu kolaboracije elemenata dizajna sistema sa spoljnjim podsistemima/ interfejsima i na utvrđivanju zavisnosti od drugih podsistema.

3. Dizajn klasa
Fokus je na inkorporiranju potrebnih mehanizama dizajna u okviru klasa i na upravljanju nefunkcionalnim zahtevima koji se odnose na datu klasu.

4. Definisanje testnih elemenata
Podrazumeva identifikovanje elemenata koji su neophodni za izradu testova; definisanje uslova neophodnih za fizičko testiranje softvera; definisanje fizičkih elemenata neophodnih za implementaciju testa.

5. Dizajn kapsula
Zadatak podrazumeva detaljan opis karakteristika kapsula.

6. Pregled dizajna
Zadatak ima za cilj da proveri da li je razvijeni dizajn u skladu sa definisanim uputsvima, kao i da li sama uputstva ispunjavaju definisane ciljeve.

Zadatak na kojem pored dizajnera učestvuje i softver arhitekta je:

7. Dizajn operacija
Softver arhitekta dorađuje rezultate dobijene nakon izvršavanja zadatka analiza operacija i dokumentuje ih u okviru artifakta operacije sistema. Rezultat ovog koraka su: definisane

operacije, definisan interfejs, način saradnje podsistema sa drugim podsistemima, definisan kontekst podsistema (akteri i ulazno/izlazni entiteti).

7. Dizajn baze podataka

Dizajn baze podataka je značajna arhitekturalna aktivnost ukoliko je na projektu potrebno upravljati velikom količinom podataka. Ulogu u ovoj aktivnosti ima dizajner i realizuje je kroz sledeće korake: dizajn baze podataka, dizajn klasa i pregled dizajna. Rezultati ove aktivnosti mogu se svrstati u tri kategorije: identifikovanje persistent klasa dizajna; dizajn strukture baze podataka u cilju čuvanja persistent klasa u njoj; definisanje mehanizama i strategija čuvanja i preuzimanja persistent podataka. Dizajn baze podataka je aktivnost koja se odvija paralelno sa dizajnom komponenti sistema, u cilju identifikovanja klasa koje se odnose na bazu podataka (Kroll & Kruchten, 2003).

Persistent klase su one čije je stanje potrebno čuvati na nekom trajnom mediju. Čuvanje njihovog stanja potrebno je za trajno čuvanje podataka date klase, backup u slučaju pada sistema ili za razmenu informacija u sistemu. Tokom faze analize sistema identifikuju se svi potrebni mehanizmi dizajna, pa time i persistent mehanizmi neophodni za rad sa bazom podataka. Persistent objekti ne moraju biti izvedeni samo iz entitetskih klasa, oni mogu imati i funkciju upravljanja nefunkcionalnim zahtevima generalno (npr. čuvaju stanje informacija između transakcija). Identifikovanje persistent klasa budućeg sistema sprovodi softver arhitekta, čime ostvaruje preduslove za dizajn baze podataka. Dizajn baze podataka podrazumeva razvoj prvo logičkog modela podataka, a potom i fizičkog, kao i odabir nekog relacionog sistema za upravljanje bazom podataka. Izrada logičkog modela podataka podrazumeva identifikovanje entiteta za svaku persistent klasu, dokumentovanu u artifaktima model analize i model dizajna. Fizički nivo dizajna baze podataka podrazumeva definisanje tabela (i njihovih karakteristika kao što su primarni ključ, referencijalni integriteti i druga ograničenja tabela), pogleda, procedura i sl. Oba modela sastavni su deo artifakta model podataka (Rational Team, 2010).

8. Specifikacija servisa

Aktivnost specifikacija servisa sprovodi se u slučaju razvoja SOA arhitekture i podrazumeva generisanje sledećih rezultata: definisani servisi i struktura SOA rešenja (u kontekstu kolaboracije elemenata dizajna i spoljnjih podsistema/interfejsa); utvrđene zavisnosti i komunikacije između servisa; dokumentovane servisne specifikacije. Navedeni rezultati se ostvaruju realizacijom sledećih zadataka: izrada servisnih specifikacija, rad na analizi podsistema i izrada specifikacija komponenti (Shuja & Krebs, 2008).

Narednim tekstom će detaljno biti opisan samo zadatak *izrada servisnih specifikacija*, jer u njegoj realizaciji pored dizajnera učestvuje i softver arhitekta:

Uloga ovog zadatka je da odabere konačan set servisa, iz grupe potencijalnih servisa, identifikovanih u prethodnim razvojnim fazama projekta. Odluka je arhitekturalno značajna, jer svaki servis iziskuje određene troškove, te je potrebno za svaki servis ponaosob analizirati da li ugrožava ekonomski aspekt projekta ili pak nefunkcionalne zahteve sistema. Anliza se sprovodi prema unapred definisanim kriterijumima, radi sagledavanja njihove finansijske opravdanosti i veličine poslovne vrednosti koju obezbeđuju. Najčešće korišćena tehnika su lakmus testovi. Servisi koji ne prođu testove, a bitni su za ispunjavanje određenih poslovnih potreba, mogu biti implementirani kao metode određenih servisnih komponenti. Primena ovih testova sprovodi se iterativno. Postoji mogućnost da se, zbog neke projektno specifične odluke (poslovne ili arhitekturalne), razvijaju i servisi koji nisu prošli testiranje (Rational Team, 2010). Odluke se dokumentuju u artifaktima : model servisa i model cilj-servis. Artifakt model cilj-servis prikazuje direktnu vezu između postavljenih poslovnih ciljeva i odabranih servisa, kao IT rešenja za ostvarivanje poslovnih ciljeva. (Rational Team, 2010).

Tabela 3.7 daje pregled artifakata discipline analiza i dizajn i kratak opis njihove svrhe i značajnosti u odnosu na različite tipove projekata.

Tabela 3.7 Artifakti discipline analiza i dizajn

ARTIFAKT	SVRHA
Model analize (klase analize)	Opcioni artifakt. Koristi se za analizu i bolje razumevanje zahteva, u cilju donošenja adekvatnih odluka dizajna
Prototip korisničkog interfejsa	Preporučuje se za sve vrste projekata. Koristi se u cilju predstavljanja i testiranja funkcionalnosti i korisnosti pre otpočinjanja implementacije. Značajna tehnika za validaciju dizajna.
Model dizajna	Preporučuje se za sve vrste projekata. Značajan jer je eliminiše dodatne troškove vezane za ispravljanje grešaka dizajna nakon implementacije.
Klase dizajna, paketi dizajna	Klase i paketi predstavljaju osnovu objektno orijentisanog pristupa dizajna, koji se najčešće koristi. Ovi artifakti se preporučuju za sve vrste projekata.
Realizacija slučajeva upotrebe	Preporučuje se za sve vrste projekata. Predstavlja most prelaska iz faye analize u dizajn.
Interfejs	Preporučuje se za sve vrste projekata.Koristi se za definisanje ponašanja koje će realizovati softverske komponente.
Dizajn podsistema	Preporučuje se za sve vrste projekata. Koristi se za enkapsulaciju ponašanja unutar komponenti koje obezbeđuju interfejs.Koristi se enkapsulaciju interakcija klasa i/ili drugih podsistema.
Događaj	Preporučuje se u slučaju razvoja real-time sistema. Koristan za sistema koji odgovaraju na puno spoljašnjih događaja.
Protokol	Preporučuje se u slučaju razvoja real-time sistema.
Signal	Preporučuje se u slučaju razvoja konkurentnog izvršavanja sistema.
Kapsule	Preporučuje se u slučaju razvoja real-time sistema, ali i sistema koji imaju izraženu konkurentnost u radu.
Model podataka	Preporučuje se za projekte koji koriste bazu podataka.

Model raspoređivanja	Opcioni artifakt je, jer raspoređivanje je sastavni deo artifakta dokument softverske arhitekture. Svrha ovog modela je da se predstavi raspoređivanje softverskih elemenata na fizičke (čvorove).
Arhitekturalni POC	Preporučuje se za većinu projekata. Svrha ovog artifakta jeste da proveri da li arhitekturalno rešenje zadovoljava arhitekturalno značajne zahteve. Može imati različite forme: spisak poznatih tehnologija koje su adekvatne za rešenje, skica konceptualnog modela rešenja, simulacija rešenja, izvršni prototip.
Dokument softverske arhitekture	Preporučuje se za sve vrste projekata. Svrha dokumenta je da pruži sveobuhvatan arhitekturalni pregled sistema, u cilju boljeg razumevanja sistema i opisa ključnih arhitekturalnih odluka.
Referentna arhitektura	Preporučuje se za većinu projekata. Svrha ovog artifakta jeste da upotrebom potvrđenih, postojećih rešenja ubrza razvoj i smanji rizik.

Izvor: (Rational Team, 2010)

3.2.3.4 Disciplina implementacija

Disciplina implementacija bavi se pitanjima razvoja i organizacije softverskog koda, testiranjem i integracijom razvijenih komponenti u izvršni sistem (Shuja & Krebs, 2008).

Aktivnost struktuiranje modela implementacije, koju realizuje softver arhitekta, predstavlja vezu između razvoja softverske arhitekture i discipline implementacije. Rezultat ove aktivnosti je set podsistema implementacije, koji se dokumentuju u artifaktu struktura modela implementacije. Inicijalni model se razvija rano, paralelno sa razvojem drugih aspekata arhitekture. Cilj modela je da eliminiše probleme vezane za upravljanje konfiguracijom i da omogući nesmetanu implementaciju, integraciju i proces izgradnje sistema (Kruchten, 2004).

Ključni koraci u realizaciji ***aktivnosti struktuiranje modela implementacije*** su: (Rational Team, 2010):

1. Uspostavljanje strukture modela implementacije
Fokus je na transformisanju modela dizajna u model implementacije. U tom procesu paketi dizajna postaju podsistemi implementacije, koji sadrže jedan ili više direktorijuma/fajlova (artifakt elementi implementacije) za implementaciju odgovarajućih elemenata dizajna. Ovim procesom može doći do promene u alokaciji podsistema na drugi sloj arhitekture.

2. Podešavanje podsistema implementacije
Odnosi se na izmene u organizaciji podsistema, ukoliko okruženje u kojem će se implementacija sprovoditi, to zahteva. Identifikovane promene dokumentuju se u artifaktu model dizajna. U ovom koraku vrši se i raspodela zadataka i odgovornosti između pojedinaca ili timova, za razvoj određenih podsistema implementacije. Takođe, identifikuje se i potreba za "uvozom" izlaznih rezultata drugih podsistema.

3. Definisanje uvoza svakog podsistema

Podrazumeva donošenje odluke o tome koje podsisteme treba da koristiti dati podsistem. Moguća je situacija kada se dozvoljava da svi podsistemi, jednog sloja, koriste (uvoze) sve podsisteme nižeg sloja.

4. Ažuriranje pogleda implementacije

Podrazumeva ažuriranje artifakta dokument softverske arhitekture, čiji je sastavni deo pogled implementacije. Pogled implementacije sadrži dijagram komponenti, koji prikazuje identifikovane slojeve i alokaciju podsistema implementacije na slojeve, kao i uvozne zavisnosti između podsistema.

3.2.4 ADD metoda

ADD metoda za razvoj softverske arhitekture razvijena je od strane Instituta za softverski inžinjering i Carnegie Mellon univerziteta. Specifičnost ove metode ogleda se u njenoj usredsređenosti na nefunkcionalne zahteve sistema, prilikom dekomponovanja sistema ili elemenata sistema. Metoda naglašava značaj arhitekturalnih taktika i obrazaca, čijom upotrebom se lakše ispunjavaju identifikovani nefunkcionalni zahtevi sistema.

Rekurzivni proces razvoja arhitekture može otpočeti tek kada su identifikovani funkcionalni i nefunkcionalni zahtevi sistema i ograničenja dizajna. Nefunkcionalni zahtevi oslikavaju osobine koje dati sistem treba da poseduje prilikom izvršavanja (npr. dostupnost, portabilnost, performase, bezbednost, upotrebljivost i dr). Nefunkcionalni zahtevi moraju biti vrednovani i rangirani po prioritetu , u odnosu na poslovne ciljeve organizacije. Funkcionalni zahtevi predstavljaju ključne funkcionalnosti budućeg sistema, a ograničenja dizajna su faktori okruženja koje treba uzeti u obzir prilikom izbora arhitekturalnog rešenja (primeri ograničenja: za skladištenje podataka mora se koristiti SQL Server, sistem mora biti implementiran upotrebom C# programskog jezika, sistem treba da funkcioniše i na Windows i na Unix platformi i sl.) (Bass, Clements, Kazman, 2013).

Razvoj i dokumentovanje softverske arhitekture, upotrebom ADD metode, vrši se putem sledećih arhitekturalnih pogleda: modul, komponenta i konektor,raspoređivanje. Proces razvoja softverske arhitekture može se opisati kroz sledeće korake:

Korak 1: analiza zahteva

Početni korak podrazumeva procenu identifikovanih zahteva sistema i utvrđivanje njihovih prioriteta u odnosu na poslovne ciljeve organizacije. Arhitekturalne odluke se ne donose u ovom koraku, već se samo provera da li inputi (funkcionalni i nefunkcionalni zahtevi, ograničenja dizajna) za razvoj arhitekture imaju adekvatan kvalitet. Inputi predstavljaju smernice u odabiru elemenata arhitekture sistema.

Svaki nefunkcionalni zahtev potrebno je predstaviti u formi "stimulans-odgovor", slično scenarijima kvaliteta atributa (Bass, Klein, Bachmann, 2001):

- izvor stimulansa,
- stimulans,
- artifakt,
- okruženje,
- odgovor,
- mera odgovora.

Cilj ovakvog opisa nefunkcionalnih zahteva je izbor adekvatnih taktika i obrazaca za njihovo ispunjavanje. Izrađeni scenariji nefunkcionalnih zahteva, predstavljaju ujedno i njihovo dokumentovanje.

Korak 2: izbor elementa sistema za dekomponovanje

Drugi korak karakteriše odabir ključnih elemenata sistema, koji će se razrađivati narednim razvojnim koracima. Odabir elemenata vrši se na dva načina:

1. U slučaju razvoja potpuno novog sistema - budući sistem predstavlja jedini element koji se može dekomponovati.
2. U slučaju dorađivanja već postojećeg sistema, potrebno je izvršiti izbor jednog od elemenata kojim će otpočeti rekurzivni proces razvoja arhitekture. Odabir elementa postojećeg sistema, može se izvršiti na osnovu nekih od sledećih kriterijuma (Wojcik et al., 2006):

- Trenutno stanje arhitekture - izbor opredeljuje broj zavisnih odnosa koje neki elemenat ima sa drugim elementima sistema.
- Rizik i težina - podrazumeva analizu arhitekturalno značajnih zahteva nekog elementa, sa aspekta rizika i mogućnosti njihovog realizovanja.
- Poslovni kriterijumi - podrazumevaju sagladavanje uloge koju elemenat ima u inkrementalnom razvoju sistema, kao i da li će dati elemenat biti izgrađen, kupljen, iznajmljen ili korišćen kao "open source". Bitno je sagledati i uticaj koji dati elemenat ima na rok izlaska sistema na tržište.
- Organizacioni kriterijum - se odnose na sagledavanje uticaja koji dati elemenat ima na angažovanje resursa (na primer ljudskih i računarskih), veština potrebnih za njegov razvoj i sl.

Korak 3: identifikacija arhitekturalno značajnih zahteva

Fokus ovog koraka je na rangiranje identifikovanih zahteva u odnosu na njihov arhitekturalni značaj. Naime, svi identifikovani zahtevi nekog elementa dvostruko se rangiraju: sa aspekta

ostvarivanja poslovnih ciljeva organizacije i sa aspekta njihovog potencijalnog uticaja na arhitekturu. Nakon izvršene prioretizacije, odabira se nekoliko visokoprioritetnih zahteva, koji usmeravaju arhitekturalni razvoj datog elementa (Wojcik et al., 2006).

Korak 4: odabir koncepata dizajna

Podrazumeva izbor glavnih tipova elemenata, koji će sačinjavati arhitekturu sistema, kao i definisanje tipova odnosa između njih. Uloga softver arhitekte u ovom koraku ogleda se u realizaciji sledećih zadataka (Bass, Clements, Kazman, 2013):

1. Identifikovanje potencijalnih zahteva dizajna
Potencijalni zahtevi dizajna se utvrđuju na osnovu izgenerisane liste potencijalnih zahteva.

2. Formiranje liste alternativnih obrazaca dizajna
Primena obrazaca dizajna treba da predupredi potencijalne probleme dizajna. Neophodno je formirati listu potencijalnih obrazaca, te za svaki obrazac ponaosob identifikovati diskrimintorne parametre. Za svaki utvrđeni diskriminatorni parametar neophodno je definisati procentualnu vrednosti njegove značajnosti.Cilj je, na osnovu vrednosti diskriminatornih parametara doneti odluka koji će se od obrazaca koristiti za potencijalni problem dizajna.

3. Odabir najpogodnijih obrazaca za ispunjavanje identifikovanih arhitekturalno značajnih zahteva.
Idealna situacija je da jedan obrazac ispunjava sve identifikovane zahteve. U slučaju da takav obrazac ne postoji, bira se onaj koji je najbliži idealnoj situaciji, te se vrše odgovarajuća prilagođavanja, upotrebom arhitekturalnih taktika. Ukoliko se ne može iznaći ni jedan obrazac koji je dovoljno optimalan, proces dekomponovanja može otpočeti arhitekturalnom taktikom. Odabir obrsca/taktike svodi se na kreiranje matrice putem koje se analiziraju prednosti/nedostaci primene datog obrasca, u odnosu na svaki identifikovani arhitekturalno značajan zahtev. Takođe je bitno razmotriti koji su očekivani kompromisi pri korišćenju obrazaca, koliko se dobro pojedini obrasci kombinuju jedni sa drugima i da li se neki od obrazaca međusobno isključuju.

4. Analiza identitfikovanih obrazaca sa aspekta njihove međusobne povezanosti
Aktivnost se sprovodi realizacijom sledećih zadataka:

- Sagledavanje načina povezanosti pojedinih tipova elmenata iz različitih obrazaca.
- Identifikovanje tipova elemenata različitih obrazaca koji nisu u međusobnoj vezi.
- Identifikovanje preklapajućih funkcionalnosti obrazaca.
- Identifikovanje novih tipova elemenata koji se pojavljuju kao rezultat kombinovanja obrazaca.

5. Opis odabranih obrazaca

Opis usvojenih obrazaca dizajna realizuje se putem različitih arhitekturalnih pogleda: modul, komponenta i konektor, raspoređivanje.

6. Evaluacija i rešavanje nekonzistentnosti

Svodi se na evaluaciju realizovanog dizajna, u odnosu na identifikovane arhitekturalno značajne zahteve. U tu svrhu koriste se različite metoda za evaluaciju arhitekture. Ukoliko se kao rezultat evaluacije utvrde arhitekturalni zahtevi koji nisu adekvatno ispunjeni, potrebno je ponovo sprovesti vrednovanje i izbor nekog drugog obrasca ili pak primeniti odgovarajuću arhitekturalnu taktiku. Nekonzistentnosti dizajna se može utvrditi i poređenjem dizajna datog elementa, sa dizajnom drugih sličnih elemenata arhitekture (dizajn datog elementa ima osobine kao neki drugi elementi arhitekture).

Korak 5: instanciranje arhitekturalnih elemenata

Podrazumeva instanciranje softverskih elemenata, odabranih u prethodnom koraku. Instanciranim elementima se dodeljuju odgovarajuće odgovornosti. Odgovornosti prizilaze iz funkcionalnih zahteva datog elementa i funkcionalnih zahteva vezanih za nadređeni element - roditelj. Na kraju ovog koraka, svaki funkcionalni zahtev povezan sa elementom roditeljem mora biti predstavljen sekvencom odgovornosti unutar elementa potomka. Ovaj razvojni korak može se razložiti na sledećih šest podkoraka (Wojcik et al., 2006):

1. Instanciranje svakog tipa elementa koji je odabran u koraku 4

Instance se nazivaju "decom" (ili "elementima potomcima") elemenata "roditelja" - koji su trenutno u procesu dekomponovanja.

2. Raspoređivanje odgovornosti koje poseduje elemenat roditelj na njegove potomke

Svrhu koraka najbolje opisuje sledeći primer: ukoliko je elemet "roditelj" u bankarskom sistemu odgovoran za upravljanje distribucijom gotovine, prikupljanje gotovine, i beleženje transkacija, ove odgovornosti se raspodeljuju između njegove "dece" tj. elemenata potomaka.

3. Kreiranje dodatnih instanci tipova elemenata

Vrši se u cilju ispunjavanja određenih nefunkcionalnih zahteva, za koje je dati elemenat odgovoran (npr. element "potomak" može biti odgovoran za očitavanje i prikupljanje podataka putem klijentske kartice na bankomatu, kao i za prikaz istih. Tokom vremena može se uočiti nedostatak po pitanjnu performansi vezanih za prikupljanje podataka. To implicira instanciranje novog elementa, čija bi odgovornost bila prikupljanje podataka, dok bi originalni elemenat zadržao samo njihov prikaz).

4. Analiza i dokumentovanje odluka donešenih tokom petog koraka

Zadatak se realizuje upotrebom različitih pogleda:

- Pogled modul u fokusu ima dokumentovanje svojstava sistema koji nisu vezani za tok njegovog rada.
- Pogled komponenta i konektor koristi se za dokumentovanje svojstava sistema i njegovog ponašanja tokom izvršavanja (npr. interakcija elemenata tokom rada kojom se ispunjavaju nefunkcionalni zahtevi i neophodne performanse elemenata sistema.
- Pogled raspoređivanje koristi se za dokumentovanje odnosa između softverskih i hardverskih elemenata.

U ovoj razvojnoj fazi donosi se set bitnih arhitekturalnih odluka (Wojcik et al., 2006):

- Odluka o broju instanci koje će biti napravljane za svaki tip elementa; odluka o individualnim svojstvima i strukturalnim odnosima instanci elementa.
- Identifikovanje elemenata koji podržavaju glavne modove operacija.
- Definisanje strukture elementa, kojom će biti ispunjeni nefunkcionalni zahtevi.
- Odluka o načinu raspodele funkcionalnosti na elemente.
- Odluka o načinu povezanosti softverskih elemenata jedne strukture i elemenata drugih arhitekturalnih struktura.
- Odluka o načinu komunikacija različitih softverskih elemenata, kao i elemenata i eksternih entiteta. Podrazumeva definisanje tipa mehanizama i protokola, koji će se koristiti u njihovoj komunikaciji; identifikovanje nefunkcionalnih zahteva vezanih za date mehanizme komunikacije; definisanje modela podataka na kojima počiva komunikacija i sl.
- Donošenje odluke o potrebnim resursima za softverske elemente.
- Identifikovanje zavisnosti između internih softverskih elemenata.
- Odluka o mehanizmima apstrakcije koji će se koristiti.
- Odluka o modelima procesa/niti koji će biti upotrebljeni.

Korak 6: definisanje interfejsa za instancirane elemente

U koraku 6 definišu se "usluge" koje softverski elementi zahtevaju i/ili pružaju jedni drugima. Set arhitekturalnih odluka koje je potrebno doneti odnose se na: eksterne interfejse sistema, interfejse između delova sistema visokoga nivoa, interfejse između aplikacija unutar delova sistema visokoga nivoa, interfejse infrastrukture (Bass, Clements, Kazman, 2013).

Korak 7: verifikovanje i dorađivanje zahteva i načinite ih ograničenjima za instancirane elemente

Cilj koraka je da utvrdi da li sprovedeno dekomponovanje elemenata zadovoljava funkcionalne i nefunkcionalne zahteve i ograničenja dizajna koji su identifikovani u prvom koraku. Takođe, potrebno je pripremiti elemente potomke za njihovo dalje dekomponovanje (Bass, Clements, Kazman, 2013).

Korak 8: izbor novog elementa sistema za dekomponovanje

Realizacijom razvojnih koraka od 2 do 7, završava se dekompozija odabranih elemenata roditelja u elemente potomke. Svaki elemenat potomak poseduje: kolekciju odgovornosti, interfejs putem kojeg komunicira sa ostalim elementima, funkcionalne zahteve koje implementira, nefunkcionalne zahteve koje treba da ispuni i ograničenja dizajna koja mora da uvaži. Ovo je ujedno i momenat kada može da otpočne nova iteracija procesa dekompozicije, vraćanjem na korak 2, kako bi se odabrao sledeći elemenat za dekomponovanje (Bass, Clements, Kazman, 2013).

***

Istraživanjem grupe autora (Hofmeister et al., 2007), analizirano je sledećih pet metoda za razvoj softverske arhitekture, s ciljem da se utvrde njihove sličnosti i razlike: Architectural Separation of Concerns (ASC), RUP 4+1, S4V, BAPO i ADD. Kao rezultat istraživanja definisan je radni okvir koji sve njihove različite aktivnosti grupiše u tri zajedničke razvojne faze:

1. Arhitekturalna analiza, ima za cilj da definiše problem koji se rešava i identifikaciju arhitekturalno značajnih zahteva.
2. Arhitekturalna sinteza, podrazumeva postavku inicijalne arhitekture, uz prethodno razmatranje alternativnih arhitekturalnih rešenja.
3. Arhitekturalna evaluacija, podrazumeva verifikovanje postavljene arhitekture u odnosu na arhitekturalno značajne zahteve, nekom od tehnika: POC koncept, prototip, izvršni kod ili sl.

S obzirom na predmet istraživanja ove doktorske disertacije, u tabeli 3.8 prikazana je komparacija aktivnosti RUP radnog okvira i ADD metode za razvoj softverske arhitekture, dok su u tabeli 3.9 prikazani njihovi arhitekturalni artifakti.

Tabela 3.8 Aktivnosti RUP radnog okvira vs. aktivnosti ADD metode za razvoj softverske arhitekture

	ADD	RUP
Arhitekturalna analiza	Izbor zahteva značajnih za arhitekturu, upotrebom scenaria za opis nefunkcionalnih zahteva.	Identifikacija ključnih, arhitekturalno značajnih slučajeva upotrebe.
Arhitekturalna sinteza	Izbor arhitekturalnih obrazaca koji zadovoljavaju arhitekturalno značajne zahteve; instanciranje modula i alokacija funkcionalnosti; definisanje interfejsa instanciranih modula.	Postepena izgradnja arhitekture tokom faze elaboracije, opisom 4 različita pogleda, uz paralelnu implementaciju arhitekturalnog prototipa.
Arhitekturalna evaluacija	Provera i dorađivanje funkcionalnih zahteva i scenarija nefunkcionalnih zahteva. Dodeljivanje ovih ograničenja modulima deci.	Izgradnja izvršnog arhitekturalnog prototipa u cilju procene da li su ključni arhitekturalnni ciljevi ostvareni i rizici eliminisani.

Izvor: (Hofmeister et al., 2007)

ADD metod za razvoj softverske arhitekture je komplementaran sa RUP radnim okvirom, iz sledećih razloga. ADD metoda, za razliku od RUP-a, ne sadrži uputstva za razvoj softverske arhitekture do nivoa detalja, niti se bavi pitanjima njene implementacije. S druge strane, ima detaljno razrađen pristup za definisanje arhitekture kandidata, što RUP nema. Stoga je korisno aktivnosti ADD metode uključiti u okviru RUP-ove discipline analiza i dizajn i to u fazi rane elaboracije. Koraci ADD metode komplementarni su i sa RUP-ovom aktivnošću analiza ponašanja. RUP ovom aktivnošću u fokus stavlja transformisanje identifikovanih zahteva, koji opisuju ponašanje sistema, u set softverskih elemenata, dok ADD koraci vrši raspodelu funkcionalnosti, tek na osnovu prethodno identifikovane strukture arhitekture kandidata. Ostale RUP arhitekturalne aktivnosti (dorađivanje arhitekture, dizajn komponenti i dizajn baze podataka) nisu predmet razmatranja ADD metode i trebaju se realizovati nezavisno (Kazman, Kruchten, Nord, Tomayko, 2004).

Takođe, primetna razlika je i da ADD metoda razvoj arhitekture bazira isključivo na nefunkcionalnim zahtevima sistema, na osnovu kojih se oblikuje struktura arhitekture, nakon čega se vrši dodeljivanje funkcionalnosti elementima. Za adekvatno ispunjavanje nefunkcionalnih zahteva, ADD predviđa upotrebu arhitekturalnih obrazaca i taktika (Kazman, Kruchten, Nord, Tomayko, 2004).

Tabela 3.9 Artifakti RUP radnog okvira vs. ADD metode za razvoj softverske arhitekture

ARTIFAKTI	ADD	RUP
Identifikovanje i analiza arhitekturalnih zahteva	Funkcionalni zahtevi, nefunkcionalni zahtevi i ograničenja dizajna.	Dokument vizije, dopunska specifikacija sa nefunkcionalnim zahtevima, lista rizika i drugih tehničkih pitanja.
Kontekst	Poslovni kvalitativni ciljevi (pr. vreme izlaska na tržište, troškovi) i nefunkcionalni zahtevi sistema (atributi kvaliteta).	Poslovni slučaj i dokument vizije.
Arhitekturalno značajni zahtevi	AZZ rezultat su kombinacije funkcionalnih, nefunkcionalnih i poslovnih zahteva koji utiču na oblik arhitekture. Da bi se identifikovali potrebno je definisati scenarije nefunkcionalnih zahteva koji odražavaju naprioritetnije poslovne ciljeve i imaju najviše uticaja na arhitekturu.	AZZ se identifikuju iz sledećih artifakata: dokument vizije, model slučaja upotrebe, dodatna specifikacija i liste rizika.
Arhitektura kandidat	Kolekcija pogleda, obrazaca i arhitekturalnih taktika. Arhitektura je takođe povezana i sa doradom scenarija koji pokazuju preslikavanje odluka iz zahteva i pomažu pri planiranju naredne iteracije dizajna.	Odluke vezane za dizajn se donose inkrementalno i dokumentuju opisom 4 pogleda (logički, procesni, implementacioni i raspoređivanja) i izradom arhitekturalnog prototipa.
Validacija arhitekture	Arhitektura opisuje sistem kao kontejnere određenih funkcionalnosti i interakcije između kontentejnera, predstavljeno putem tri pogleda: modul dekompozicije, paralelan rad i raspoređivanje. Arhitektura se potvrđuje ukoliko su zadovoljeni AZZ/ograničenja u odnosu na dekompoziciju.	Polazna osnova celovita, izvršni arhitekturalni prototip na kraju faze elaboracije. Prototip je dovoljno potpun da može biti testiran, kako bi se potvrdilo da su glavni arhitekturalni ciljevi (funkcionalni i nefunkcionalni) ispunjeni a glavni tehnički rizici ublaženi.
Lista zadataka	Informacije koje treba obraditi u narednim koracima: zahteve koje treba analizirati, odluke koje treba integrisati, zahteve koje treba verifikovati.	Lista pitanja/problema se kontinuirano održava i sadrži elemente nezavršenih zadataka. Arhitekturalni ciljevi se dodeljuju predstojećim iteracijama. Specifikuju se u okviru plana Iteracije u obliku ciljeva date iteraciije.

Izvor: (Kazman, Kruchten, Nord, Tomayko, 2004).

Arhitekturalne aktivnosti RUP radnog okvira, koje se realizuju u okviru discipline analiza i dizajn, mogu se proširiti koracima ADD metode na sledeći način (Kazman, Kruchten, Nord, Tomayko, 2004):

1. Arhitekturalna sinteza.
2. Definisanje arhitekture-kandidata:
   a. *izbor modula koji će u narednim koracima biti dekomponovan,*
   b. *izbor arhitekturalno značajnih zahteva, isključivo iz seta nefunkcionalnih zahteva sistema,*
   c. *izbor obrazaca i arhitekturalnih taktika koji mogu na adekvatan način da doprinesu ispunjavanju identifikovanih arhitekturalno značajnih zahteva,*
   d. *instanciranje modula i dodeljivanje funkcionalnosti pojedinačnim instancama,*
   e. *definisanje interfejsa instanciranih modula (dece),*
   f. *provera i dorađivanje funkcionalnih zahteva i scenarija (putem kojih se vrši opis i dokumentovanje nefunkcionalnih zahteva), koji će predstavljati ograničenja instance.*
3. Dorađivanje arhitekture.
4. Analiza ponašanja.
5. Dizajn komponenti.
6. Dizajn baze podataka.

Inpute za razvoj aritekture kandidata, primenom ADD koraka, obezbeđuju sledeći artifakti u RUP-u: dokument vizije, arhitekturalni POC, model slučajeva upotrebe i dokument dodatne specifikacije (Kazman, Kruchten, Nord, Tomayko, 2004).

3.3 Pregled i analiza odabranih agilnih procesa razvoja

Poglavljem se ukratko predočavaju istorijski uslovi koji su doveli do nastanka agilnih procesa razvoja, a predmet detaljne analize su tri najzastupljenija agilna procesa razvoja u praksi: XP, Scrum i Lean. Analiza ovih agilnih procesa razvoja sprovodi se u cilju obezbeđivanja osnova za realizaciju empirijskog dela istraživanja i davanja odgovora na drugo istraživačko pitanje.

3.3.1 Istorijski pogled na agilne procese razvoja

Razvoj informaciono tehnologija i interneta uticao je na pojavu globalnog poslovanja i 24 časovne ekonomije. Preduzeća su bila prinuđena da se prilagode revolucionarnim promenama koje su nastale u poslovnoj klimi, kako bi obezbedile svoj opstanak na tržištu. Prilagođavanje novim tržišnim prilikama i ekonomskim uslovima nije zaobišlo ni preduzeća za razvoj softverskih proizvoda, jer je softver postao deo gotovo svih poslovnih operacija.

Postizanje kompromisa između brzog razvoja i isporuke, s jedne strane, i proizvodnje visoko kvalitetnog poslovnog softvera, predstavljao je izazov u softverskoj industriji. Rapidne promene u poslovnom okruženju onemogućile su inicijalno definisanje stabilnih i fiksnih

softverskih zahteva. Promenljivost zahteva tokom celog životnog ciklusa razvoja softvera, pa i nakon njegove implementacije, postao je imperativ u razvoju poslovnog softvera krajem dvadesetog veka. U ovakvim okolnostima tr adicionalni procesi razvoja, koji su do tada uspešno odgovarali na poslovne zahteve, nisu mogli adekvatno da odgovore novim poslovnim izazovima.

Dobro definisani procesi, sa velikim brojem uloga, aktivnosti i masovnom dokumentacijom, pogodovali su razvoju velikih skalabilnih softverskih rešenja koja se lako održavaju, ali su predstavljali balast u razvoju malih internet poslovnih rešenja. Primer takvog procesa razvoja je RUP. Uspostvaljanje samog razvojnog procesa po RUP-u predstavljao je zahtevniji i vremenski duži posao od samog rešenja koje se trebalo razviti. Tradicionalni proces razvoja iziskuje velika ulaganja resursa, što preduzeća suočena sa brzim razvojem tehnologije i čestim promenama u poslovnom okruženju nisu u mogućnosti da obezbede. Kreiranje i prihvatanje promena postao je imperativ za ostvarivanje njihove konkurentnosti na tržištu. To je istorijski moment pojave "lakih" procesa razvoja softvera, koji se svrstavaju u grupu agilnih metodologija (Matković, Tumbas, Sakal, 2011; Cohen, Lindvall, Costa, 2004; Highsmith & Cockburn, 2001).

Agilne metodologije karakteriše spremnost na kontinuirane promene (Highsmith & Cockburn, 2001), neformalnost (Cockburn, 2007), hitrost, spretnost, popustljivost, opreznost (Erickson, Lyytinen, Siau, 2005) i samosvesnost razvojnog tima (Moe, Dingsoyr, Dyba, 2009). Lindstrom i Jeffries (2004) u rezultatima istraživanja navode da je većina tradicionalnih projekata propadala iz sledećih razloga:

- Zahtevi nisu jasno saopšteni i rano su "zamrzavani".
- Zahtevi ne rešavaju realne poslovne probleme.
- Promena zahteva pre završetka projekta predstavlja realan problem.
- Kod nije testiran od strane korisnika koji će ga koristiti.
- Razvijeni softver teško je modifikovati.
- Softver se koristi za funkcije za koje nije namenjen.
- Projekti ne koriste resurse predviđene planom.
- Raspored i obim obaveza pravljeni su pre potpunog razumevanja zahteva i rizika.

Pojava prvih agilnih procesa razvoja beleži se kasnih 90-tih godina: DSDM (Stapleton, 1997), Scrum (Schwaber & Beedle, 2002), XP (Beck & Andres, 2004; Beck, 1999). Danas je teoriji i praksi poznat širok dijapazon agilnih procesa razvoja: Crystal skup metoda (2002. godine Cockburn), Feature Driven Development (2002. godine Palmer i Felsing), Adaptive Software Development (2000. godine Highsmith), Open Source Software Development (1999. godine O'Reilly), Lean Software Development (2003. godine Mary i Tom Poppendieck).

Svaki agilan proces razvoja je jedinstven po svojoj terminologiji, načinu implementacije i artifaktima. Agilne metodologije predstavljaju kolekcije različitih tehnika (praksi), ali dele iste

vrednosti i osnovne principe razvoja. Većina agilnih praksi nije potpuno nova za softversku zajednicu, ali jeste način na koji su one kombinovane i postavaljene u „teoretski i praktični okvir“ (Merisalo-Rantanen, Tuure, Matti, 2005). Novina koju nosi agilnost je drugačiji set prioriteta i vrednosti koje se favorizuju. Reč je o njihovoj usmerenost na ljude i korisnikovo zadovoljstvo kao glavne faktore uspeha projekta, kao i intenzivan fokus na efikasnost i manevrisanje (Cockburn, 2007;Glass, 2001; Highsmith & Cockburn, 2001).

Jim Highsmith (2001) je agilnost opisao na sledeći način: “Brza isporuka. Brze promene. Česte promene”. Ostvarivanje ovih imperativa agilni procesi zasnivaju na iterativno-inkrementalnom razvojnom modelu. Razvoj softverskog rešenja realizuje se kroz veliki broj malih iteracija, čije trajanje nije duže od četiri nedelje. Svaka iteracija predstavlja zaokruženi ciklus razvoja softvera, uključujući planiranje, analizu zahteva, dizajn, kodiranje, testiranje i prihvatanje testiranja. Rezultat svake iteracije je nova funkcionalnost softvera koja se isporučuje korisniku uz minimalnu dokumentaciju. Korisnici su uključeni u ceo razvojni proces, što obezbeđuje dobijanje brzog odgovora za razvijeni set funkcionalnosti, brzo definisanje neophodnih promena i seta novih softverskih zahteva, kao inputa za narednu iteraciju. Ovakav razvoj doprinosi fleksibilnosti projekta, omogućavajući brzo prilagođavanje promenama. Agilan razvoj naglašava interakciju i komunikaciju svih članova tima i svih ključnih korisnika (stejkholdera), a marginalizuje ceremonijalnost (Dingsoyr, Nerur, Moe, 2012; Leffingwell, 2007).

Temelj korišćenja mnoštva agilnih procesa razvoja sistematizovan je u dokumentu Agilni Manifest (Manifesto for Agile Software Development). Definisala ga je agilna Alijansa, neprofitna organizacija, osnovana na skupu u zimskom odmaralištu u državi Juti u Sjedinjenim američkim državama od 11. do 13. februara 2001. godine (Beck, et al., 2001).

Agilnim Manifestom proklamovane su četiri ključne vrednosti, zajedničke za sve agilne procese razvoja (Sommerwille, 2010; Beck et al., 2001):

1. Pojedinci i interakcije iznad procesa i alata.
2. Primenljiv softver iznad detaljne dokumentacije.
3. Saradnja sa klijentima iznad ugovora.
4. Reakcija na promenu iznad pridržavanja plana.

Tvorci agilnog manifesta: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, postavili su i dvanaest principa na kojima počivaju sve agilne metodologije (Beck et al., 2001):

1. Zadovoljan klijent je naš vrhunski prioritet, koji ostvarujemo blagovremenom i kontinuiranom isporukom vrhunskog softvera.

2. Spremno prihvatamo promene zahteva, čak i u kasnoj fazi razvoja. Agilni procesi omogućavaju uspešno prilagođavanje izmenjenim zahtevima, što za rezultat ima prednost naših klijenata u odnosu na konkurenciju.
3. Redovno isporučujemo primenljiv softver, u periodu od nekoliko nedelja do nekoliko meseci, dajući prednost kraćim intervalima.
4. Poslovni ljudi i razvojni tim svakodnevno sarađuju u tokom trajanja projekta.
5. Projekte ostvarujemo uz pomoć motivisanih pojedinaca. Obezbeđujemo im ambijent i podršku koja im je potrebna i prepuštamo im posao s poverenjem.
6. Za najproduktivniji i najefikasniji metod prenosa informacija, do i unutar razvojnog tima, smatramo kontakt licem u lice.
7. Primenljiv softver je osnovno merilo napretka.
8. Agilni procesi promovišu održivi razvoj. Sponzori, razvojni tim i korisnici moraju biti u stanju da kontinuirano rade ujednačenim tempom, nezavisno od vremena trajanja projekta.
9. Stalna posvećenost vrhunskom tehničkom kvalitetu i dobrom dizajnu pospešuju agilnost.
10. Jednostavnost je prioritet, kao veština eliminisanja nepotrebnog, suvišnog rada.
11. Najbolje arhitekture, zahtevi i dizajn rezultat su rada samoorganizovanih timova.
12. Tim u redovnim vremenskim intervalima razmatra načine kako da postane efikasniji, zatim se usaglašava, a potom prilagođava svoje ponašanje u skladu sa donetim odlukama.

Pojava agilnih procesa donela je značajne i korisne novine u razvoj softvera, te su se pre desatak godina pojavili i prvi pokušaji agilizacije tada dominantnog, tradicionalnog procesa razvoja: RUP-a. Ideja je proistekla iz činjenice da su obe metodologije zasnovane na iterativno inkrementalnom razvojnom modelu, što znači da dele iste osnovne principe. Ovi napori rezultovali su pojavom sledećih metodologija: Agile Unified Process - AUP (Ambler, 2001), Essential Unified Process - EssUP (Jacobson, Wei Ng, Spence, 2006) i Open Unified Process - OpenUP (Balduino, 2007). Sve tri metodologije zadržale su osnovnu strukturu RUP-a, ali su inkorporirale neke od agilnih tehnika kao što su: Test Driven Development (TDD), Agile Model Driven Development (AMDD), Agile Change Management (ACM) i Database Refactoring.

3.3.2 Scrum

Scrum proces razvili su Ken Schwaber i Jeff Sutherland, krajem devedesetih godina prošlog veka (Sutherland, J., Schwaber, 1995). Nastao je na osnovama teorije kompleksnih adaptivnih sistema i idejama japanskog menadžmenta. Pojam "scrum" prvi put se pojavljuje 1986. godine u radovima Hirotaka Takeuchi i Ikujiro Nonaka (1986).

Scrum predstavlja iterativno inkrementalni procesni okvir, koji se koristi za upravljanje razvojem i održavanjem softverskih proizvoda. Okvir sadrži set menadžerskih preporuka, ali

ne definiše aktivnosti samog razvojnog procesa. Iz tog razloga često se koristi u kombinaciji sa drugim procesima razvoja softvera, kao što su: XP i RUP. Zasnovan je na empirijskoj kontroli procesa, koju omogućavaju tri bazična koncepta: transparentnost (vidljivost značajnih aspekata procesa onima koji su odgovorni za rezultate), pregled (provera artifakata) i adaptacija (prilagođavanje procesa ili resursa ukoliko se utvrdi da rezultati nisu odgovarajući). U skladu sa tim, Scrum meri proizvedene izlaze budućeg sistema nakon svake iteracije. Scrum okvir izgrađuju četriri ključna elementa (Schwaber & Sutherland, 2013): uloge u timu, događaji, artifakati i pravila.

Pravila Scrum-a čine vezivno tkivo između događaja, uloga i artifakata, uređujući njihove odnose i interakcije (Schwaber & Sutherland, 2013).

Odgovornosti u realizaciji Scrum aktivnosti podeljene su na sledeće **uloge** koje imaju pojedinci/timovi (Matković, 2011; Schwaber & Beedle, 2002):

1. *Vlasnik proizvoda* ima zadatak da prikuplja inpute od kupaca, krajnjih korisnika i članova razvojnog tima. Prikupljene inpute transformiše u zahteve i vrši njihovo vrednovanje sa aspekta prioriteta u razvoju. Proces prioritizacije zahteva zasnovan je na konceptu maksimiziranja poslovne vrednosti isporukom date funkcionalnosti. Vlasnik proizvoda može biti tehničko lice, ali je najčešće iz redova menadžmenta organizacije.

2. *Razvojni tim* je krosfunkcionalan i samoorganizujuć, a čini ga od pet do deset članova sledećih profila eksperata: analitičari, programeri, dizajneri i testeri. Razvojni tim ima autonomiju u donošenju odluka, kao i slobodu da vlasniku proizvoda dostavlja ideje za unapređenje proizvoda. Razvojni tim je oubičajeno angažovan samo na jednom projektu, jer u suprotnom dolazi do pada njegove produktivnosti i kreativnosti tokom rada.

3. *Scrum vođa* je odgovoran za uspešan razvoj krajnjeg proizvoda. Scrum vođa nije menadžer, već lider koji može biti ujedno i član razvojnog tima. Treba da poseduje širok spektar znanja o samom procesu razvoja (od analize do upravljanja projektom) i da ima iskustva na razvojnim projektima. Zadužen je za uspešnu implementaciju Scrum-a na projektu, pružajući kontinuiranu pomoć i podršku članovima razvojnog tima.

4. *Menadžer* je odgovoran za konačno donošenje odluka, a učestvuje i u procesu postavljanja ciljeva i definisanju zahteva.

5. *Kupac* je osoba koja sa ostalim članovima tima učestvuje u procesu generisanja zahteva i definisanja funkcionalnosti koje budući sistem treba da ima. Takođe, učestvuje i u procesu provere proizvedenih rezultata (funkcionalnosti), pružajući povratne informacije razvojnom timu.

Propisani događaji u Scrum-u obezbeđuju pravilnosti u radu i omogućavaju implementaciju tri osnovna koncepata na kojima je zasnovan Scrum: transparentnost, provera i adaptacija. Događaji su vremenski ograničeni, sa definisanim maksimalnim vremenom trajanja. Glavni događaj, koji uključuje sve druge događaje jeste sprint.

Sprint predstavlja iteraciju koja može da traje najduže mesec dana, a za rezultat daje izvršni proizvod. Nakon pokretanja sprinta, nije moguće vršiti izmene postavljenih ciljeva sprinta, poslovnih zahteva, kao ni promenu učesnika sprinta. Potrebne izmene, usled povećanja kompleksnosti, rizika ili poslovnih potreba, moguće je implementirati pokretanjem narednog sprinta. U periodu trajanja sprinta vrši se kontinuirana provera i adaptacijia rezultata, u odnosu na postavljene ciljeve sprinta. Izvršavanje sprinta može biti prekinuto samo u ekstremnim slučajevima, kada nastupe promene tehnoloških ili tržišnih uslova, pa definisani ciljevi sprinta bivaju zastareli. Priviligija otkazivanja sprinta pripada samo vlasniku proizvoda, iako može biti razultat zajedničke odluke celog tima (Schwaber & Sutherland, 2013).

Ključni događaj za pokretanje sprinta jeste *planiranje sprinta*. Na događaju planiranja sprinta precizira se šta u okviru sprinta treba biti urađeno tj. šta će biti razultat inkrementa i kako će se taj rezultat ostvariti. Događaj planiranje sprinta je, kao i sam sprint, vremenski ograničen i može trajati najduže osam sati, a uključuje vlasnika proizvoda i razvojni tim. Rezultat planiranja sprinta je lista zadataka sprinta, kroz čije izvršavanje se realizuju postavljeni ciljevi sprinta.

Vreme trajanja sprinta se ne može produžavati, pa je važna njegova realna procena tokom planiranja sprinta (Schwaber & Sutherland, 2013). Plan realizacije postavljenih ciljeva sprinta podrazumeva definisanje funkcionalnosti koje razvijeni inkrement treba da ima da bi ciljevi bili ispunjeni (Schwaber & Sutherland, 2013; Rubin, 2012).

Dnevni sastanci predstavljaju drugi bitan dogadjaj u Scrum-u. Njihovo trajanje ne sme biti duže od petnaest minuta, te se iz tog razloga održavaju u stojećem položaju prisutnih. Dnevni sastanci obezbeđuju transparentnost u radu, jer je svaki član razvojnog tima u obavezi da podnese izveštaj o problemima koje je identifikovao, o urađenom poslu prethodnog dana i o rezultatima koje planira da ostvariti do sutrašnjeg dnevnog sastanka. Scrum vođa ima zadatak da beleži iznesene probleme članova razvojnog tima, kao i da im nakon sastanka pomogne u njihovom rešavanju. Vlasnik proizvoda i menadžer mogu biti samo pasivni posmatrači na ovom događaju, bez prava na diskusiju (Rubin, 2012).

Ocena sprinta je događaj koji se odvija na kraju sprinta i ima za cilj demonstriranje rezultata ostvarenih kroz sprint. Članovi razvojnog tima, scrum vođa, vlasnik proizvoda, korisnici, menadžer i ključni stejkholderi u mogućnosti su da po prvi put upotrebljavaju izrađene funkcionalnosti softverskog rešenja. Ovaj događaj omogućava proveru izgrađenog rešenja, dobijanje povratnih informacija i generisanje polaznih inputa za planiranje narednog sprinta.

Sastanak podrazumeva aktivnu diskusiju svih prisutnih, u cilju unapređivanja samog razvojnog proces i učenja iz implementiranog rada (Rubin, 2012).

Retrospektiva sprinta je događaj na kojem učestvuje ceo tim, a ima za cilj da identifikuje ispravnosti/defekte u radu izgrađenog softverskog rešenja. Na ovom događaju se putem table, ili neke druge vizuelne tehnike, u jednu kolonu evidentiraju ispravne funkcionalnosti rešenja, dok se u drugu vrši popis neispravnih funkcionalnosti. Nakon tabelarne analize rešenja, pristupa se diskusiji i razmatranju razloga identifikovanih nedostataka (Rubin, 2012).

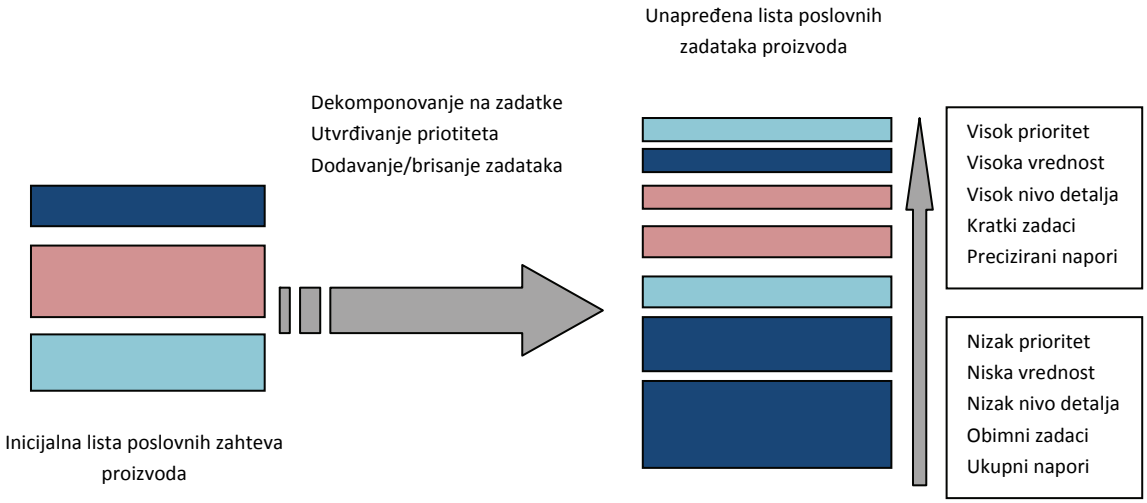
U Scrum-u se generišu četiri ključna **artifakta**, koja doprinose transparentnosti u radu i efikasnijoj komunikaciju između članova razvojnog tima, kao i njihovoj komunikaciji sa stejkholderima:

1. Artifakt lista poslovnih zadataka proizvoda (Product Backlog) je lista zahteva koju generiše i održava vlasnik proizvoda, a koja je rezultat specifikovanih poslovnih potreba i očekivanja korisnika. Zahtevi se najčešće zapisuju u formatu korisničke priče, mada sve češće i u formatu epova. Epovi predstavljaju korisničke priče za čiji razvoj i testiriranje je potrebno više od dva sprinta (Cohn, 2009).

Svi zahtevi sa liste imaju utvrđene prioritete, koji im se dodeljuju u odnosu na važnost koju imaju u ostvarivanju vrednosti za kupca (slika 3.13). Zahtevi mogu sadržavati i opise kao što su rizici i zavisnosti koje ih karakterišu. Artifakt lista poslovnih zadataka proizvoda je promenljiva tokom celog procesa razvoja, a ažurira se zbog promena tržišnih uslova i identifikovanja novih poslovnih potreba. Iz tog razloga na početku projekta samo zahtevi visokog prioriteta sadrže detaljan opis korisničkih priča, jer se na osnovu njih vrši planiranje prvih sprintova.

Razvojni tim pomaže vlasniku proizvoda da definiše listu poslovnih zahteva proizvoda, naročito u delu utvrđivanja njihovih prioriteta i očekivanih napora za njihovo izvršenje. Očekivani napori se vrednuju sa bodovima, a veličina boda je rezulatat subjektivnog stava članova i jedinstvena je za svaki projekat. Na osnovu prioritizovane liste poslovnih zahteva proizvoda, razvojni tim definiše artifakt lista zadataka sprinta (Sprint Backlog). Jedan poslovni zahtev može biti razložen na više radnih zadataka sprinta, ali je moguće i da više poslovnih zahteva bude grupisano u jedan radni zadatak sprinta. Neposredni izvršioci zadataka određuju koliko je potrebno rada (sati po radnom danu) da bi se isti mogao završiti u planiranom vremenu trajanja sprinta (Schwaber & Sutherland, 2013; Rubin & 2012).

Artifakt lista zadataka verzije softverskog rešenja (Release Backlog) predstavlja podskup artifakta lista poslovnih zadataka, a generiše ga vlasnik proizvoda (Deemer, Benefield, Larman, Vodde, 2010).



Slika 3.13 Scrum, Lista poslovnih zahteva proizvoda
Izvor: (Rubin, 2012)

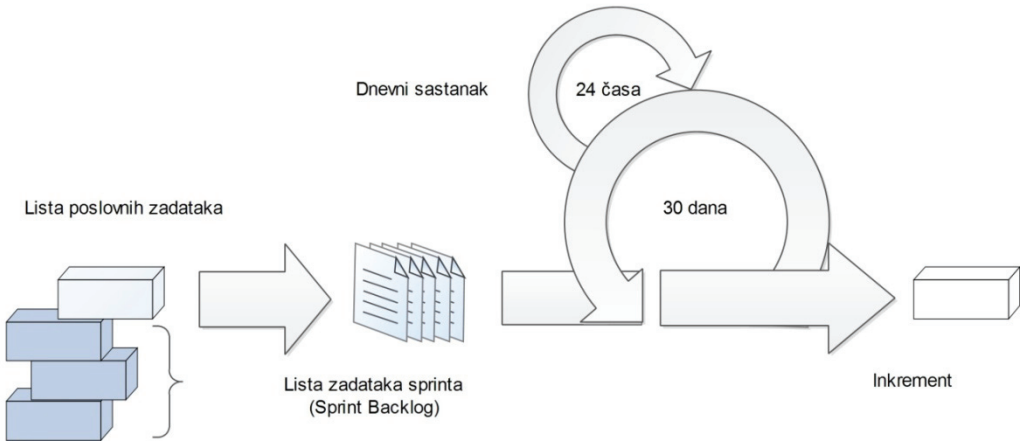
2. Artifakt lista zadataka sprinta (*Sprint Backlog)* generiše se izborom stavki iz artifakta lista poslovnih zadataka proizvoda. Odabir stavki vrši razvojni tim, na osnovu svojih radnih sposobnosti. Artifakt listu zadataka sprinta ažurira jedino tim i to tokom celog perioda trajanja sprinta. Lista može biti ažurirana dodavanjem novih zadataka ili brisanjem zastarelih, kao i promenama statusa zadataka i procenama preostalog rada. Status zadataka sprinta se prikazuju na sprint tabli u jednu od tri predviđene kolone, shodno njihovom trenutnom statusu: uraditi/u toku/završen (tabela 3.10).

Tabela 3.10 Lista zadataka sprinta

				Procena preostalih napora (časovi rada)								
Poslovni zadaci	**Radni zadaci**	**Izvršioci**	**Status**	**Scrum dan**								
				1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**...**
Poslovni zadatak 1: Omogućiti studentu da sačini listu predmeta koje će slušati u semestru	Dizajn poslovne logike	Peđa	UT		6							
	Dizajn baze podataka	Mirjana	UT		4							
	Dizajn korisničkog interfejsa	Vuk	UT		5							
	Back - end kod	Lazar	U		4							
	Front-end kod	Olivera	U		7							
	Testiranje koda	Jana	U		6							
	Dokumentovanje	Miloš	U		5							
Poslovni zadatak 2: Omogućiti studentu ga generiše sopstveni raspored časova	Radni zadatak 1	Igor	U		8							
	Radni zadatak2	Rada	UT		6							
	Radni zadatak n	Jela	UT		4							
	Procena ukupno preostalih napora (časova rada)				**57**							

Izvor: (Prilagođeno prema Deemer et al., 2010)

3. Inkrement predstavlja izvršno softversko rešenje, koje je razvio tim nakon završetka sprinta i koje je spremno za isporuku u radno okruženje (slika 3.14).



Slika 3.14 Scrum okvir na konceptualnom nivou
Izvor: (Rubin, 2012)

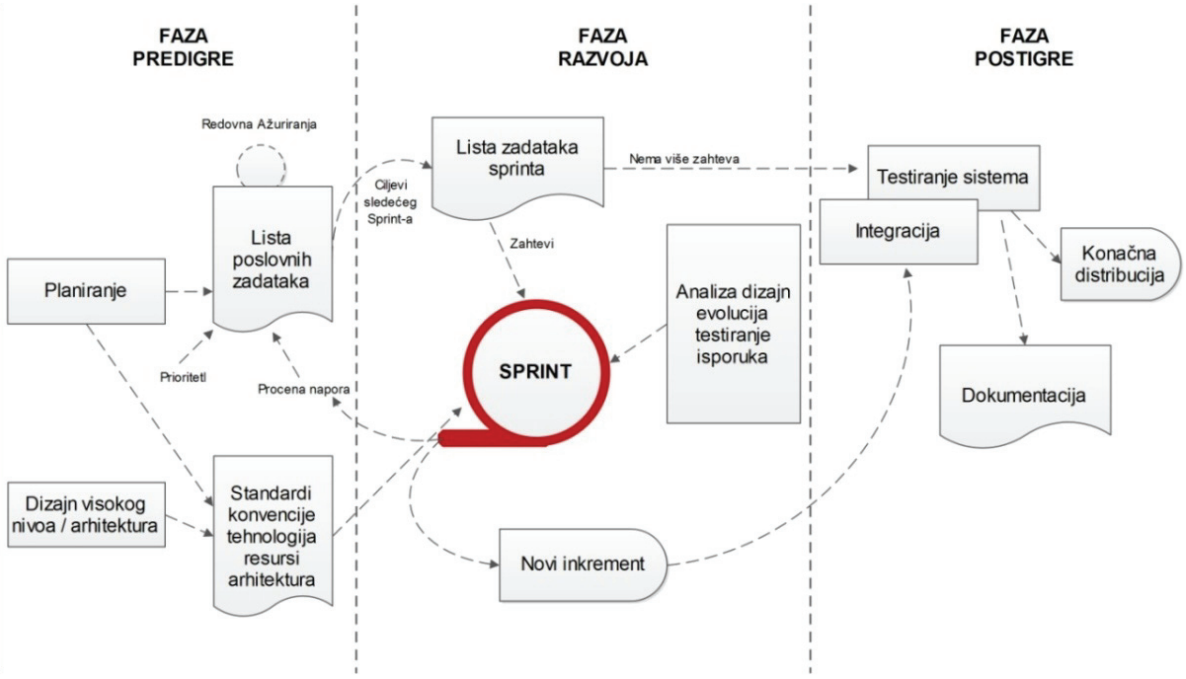
4. Artifakt dijagram toka realizacije sprinta ima za cilj praćenje napretka samoorganizujućeg Scrum tima, shodno definisanim ciljevima. Izrađuje se u terminima količine preostalog posla i potrebnih časova rada tima. Članovi razvojnog tima u obavezi su da svakodnevno ažuriraju preostale sate potrebne za obavljanje posla neophodnog za kompletiranje tekućeg zadatka. Ukupni časovi preostalog rada u sprintu, zbir su časova rada svakog učesnika ponaosob. Suma ukupno preostalog rada svih učesnika predstavlja se na dijagramu, kako bi svi stejkholderi mogli svakodnevno da imaju uvid u realnu sliku toka realizacije sprinta u odnosu na planiranu. Ukoliko rezultati na dijagramu nisu optimistični, tim mora izvršiti prilagođavanja, u smislu smanjenja obima posla ili pronalaženja efikasnijih načina rada (Deemer, Benefield, Larman, Vodde, 2010).

5. Pored artifakta dijagram toka realizacije sprinta u upotrebi je i artifakt d*ijagram toka realizacije isporuke verzije softverskog rešenja (Release Backlog)*, koji pokazuje progres u realizaciji planiranog datuma njegove proizvodnje i isporuke u produkciju (Deemer, Benefield, Larman, Vodde, 2010).

3.3.2.1 Scrum proces razvoja

Izgradnja softverskog rešenja Scrum procesom realizuje se kroz tri razvojne faze, prikazane na slici 3.15 (Schwaber & Beedle, 2002; Abrahamsson, Salo, Ronkainen, Warsta, 2002):

1. Predigra (planiranje i definisanje zahteva, dizajn/arhitektura visokog nivoa apstrakcije).
2. Igra - razvojna faza (razvoj, sprintovi - iterativni ciklusi, poboljšanja, nove verzije).
3. Postigra (nema novih zahteva, testiranje, integracija).



Slika 3.15 Scrum proces
Izvor: (Abrahamsson, Salo, Ronkainen, Warsta, 2002)

1. Faza predigre

Faza predigra uključuje dve podfaze: fazu planiranja i fazu dizajna arhitekture visokog nivoa. Planiranje primarno podrazumeva definisanje sistema koji će biti razvijan, a odvija se kroz izradu dva ključna artifakta: lista poslovnih zadataka proizvoda i lista radnih zadataka sprinta. Planiranje podrazumeva i definisanje projektnog tima, alata i dr. resursa koji će se koristiti na projektu, kao i procenu rizika i pitanja kontrolinga, identifikovanje potereba za obukama. Faza planiranja traje nekoliko sati, ali ne duže od osam sati, za sprint koji će trajati četiri nedelje (Abrahamsson, Salo, Ronkainen, Warsta, 2002).

Artifaktom lista poslovnih zadataka proizvoda potrebno je identifikovati sve zahteve koji su poznati u datom momentu razvoja, te je vremenom ažurirati usled realnih promena potreba kupaca, poslovanja, tržišnih okolnosti i sl. Generisanje inicijalne liste poslovnih zadataka proizvoda vrši vlasnik proizvoda, na osnovu prikupljenih zahteva od stejkholdera, koji mogu biti u različitim formama artifakata: karakteristike sistema, problemi korisnika ili softveski zahtevi (Matković, 2011). Vlasnik proizvoda uz pomoć razvojnog tima identifikovane poslovne zadatke proizvoda rangira prema njihovoj važnosti koju imaju u proizvodnji poslovne vrednosti za korisnika. Poslovni zadatak koji obezbeđuje karakteristiku sistema visokog prioriteta i vrednosti za korisnika, biće višlje rangiran na listi. Pored utvrđivanja prioriteta inicijalne liste poslovnih zahteva proizvoda, potrebno je u ovoj fazi razvoja izvršiti i procenu očekivanog napora za njihovu realizaciju, sistemom dodele bodova svakom zahtevu ponaosob (Deemer, Benefield, Larman, Vodde, 2010).

Artifakt radni zadaci sprinta generiše se u drugoj podfazi planiranja, na sastanku za planiranje sprinta. Svrha sastanka je da se, kroz diskusiju svih članova razvojnog tima i vlasnika proizvoda, utvrde ciljevi i način implementacije odabranih zahteva. Poslovni zadaci se razbijaju na radne zadatke, za koje se procenjuju potrebni sati rada izvršioca da bi bili realizovani (Schwaber & Sutherland, 2013).

Poslednja aktivnost faze planiranja podrazumeva raspodelu zaduženja i definisanje radnog vremena svakog učesnika sprinta, vodeći računa o propisanoj normi koja iznosi od četiri do šest sati čistog rada na realizaciji sprinta. Nakon ovog momenta vlasnik proizvoda nije više u mogućnosti da menja zahteve do planiranja narednog sprinta (Deemer, Benefield, Larman, Vodde, 2010).

Faza dizajna arhitekture visokog nivoa podrazumeva planiranje arhitekture na osnovu postojećih stavki iz artifakta lista poslovnih zadataka proizvoda. Prilikom rasta i razvoja sistema, promene koje je potrebno izvršiti radi implementacije novih stavki sa liste definišu se zajedno sa problemima do kojih te promene mogu da dovedu. U ovoj podfazi održava se sastanak radi razmatranja dizajna sa ciljem da se daju predlozi načina njegove implementacije. Dizajn arhitekture je evoluirajuć, a njegove promene i poboljšanja ostvaruju se tehnikom refaktorisanja tj. izmenom programskog koda. Rezultat ove podfaze jesu i preliminarni planovi sadržaja distribucije (Schwaber & Beedle, 2002; Abrahamsson, Salo, Ronkainen, Warsta, 2002).

2. Faza Igre

Razvojna faza u Scrum-u naziva se faza igre. Otpočinje pokretanjem sprinta, a traje od dve do četiri nedelje u zavisnosti od planiranog obima sprinta i procenjenog napora za njegovu realizaciju. Svaki sprint uključuje tradicionalne faze razvoja softvera: zahteve, analizu, dizajn, evolucija i isporuku. Arhitektura i dizajn sistema evoluiraju tokom razvoja sprinta. Najčešće je potrebno od tri do osam sprintova da bi sistem bio spreman za distribuciju.

Kroz razvojnu fazu vrši se kontinuirana provera i adaptacija svakodnevnih rezultata, kao i njihovo prilagođavanje promenama. Ovakav vid kontole je moguć zahvaljujući održavanju dnevnih sastanaka, na kojima se vrši pregled i ocena urađenog posla kao i identifikovanje potencijalnih problema u razvoju. Ključni artifakt koji proizilazi iz ovog događaja je dijagram toka realizacije sprinta. Artifakt se izrađuje na osnovu svakodnevnog ažuriranja očekivanog vremena završetka sprinta (od strane svih učesnika ponaosob), na osnovu čega scrum vođa evidentira zbir očekivanih vremena i generiše dijagram realnog stanja realizacije sprinta (Abrahamsson, Salo, Ronkainen, Warsta, 2002).

Sprintevi se po završetku rada dostavljaju vlasniku proizvoda, koji razvojnom timu dostavlja zamerke na razvijene funkcionalnosti. Obično poslednje nedelje sprinta vrše se testeranja

koda i ispravke funkcionalnosti, jer se time sprečava preveliki obim posla na kraju projekta (Matković, 2011).

3. Faza Postigre

Faza postigre podrazumeva da su svi zahtevi razvijeni i da je time završena i faza implementacije. Aktivnosti koje karakterišu fazu postigre su: integracija, testiranje sistema i izrada potrebne dokumentacije. Događaji karakteristični za ovu fazu su: ocena sprinta, retrospektiva sprinta, startovanje narednog sprinta, planiranje distribucije. Ključni artifakt je dijagram toka realizacije sprinta, kojim se vrši monitoring progresa sprinta u odnosu na postavljene ciljeve.

Na osnovu dobijenih informacija sa ocene i retrospektive tekućeg sprinta, vlasnik proizvoda ažurira artifakt lista poslovnih zadataka proizvoda, čime može da otpočne događaj startovanje narednog sprinta. Ažuriranje liste poslovnih zadataka proizvoda podrazumeva izmenu prioriteta postojećih zadataka, dodavanje novih poslovnih zadataka i brisanje neadekvatnih (Schwaber & Sutherland, 2013; Rubin, 2012).

Inicijalno planiranje distribucije vrši se na samom početku projekta, izradom plana distribucije, kojim se definiše datum distribucije, a neretko i neophodan set funkcionalnosti. Plan distribucije je moguće menjati tokom razvoja, shodno okolnostima i toku realizacije projekta. Distribucija se obično vrši nakon tri do osam realizovanih sprintova, čijom integracijom funkcionalnosti se proizvodi softversko rešenje koje može stvarati vrednost za stejkholdere (Schwaber & Sutherland, 2013; Rubin, 2012).

3.3.3 XP

Kent Beck je u drugoj polovini devedesetih godina postavio temelje XP razvojnog procesa, dok je njegova konačna forma nastala kao rezultat zajedničkog rada Beck-a i Cunningham-a. Prvu knjigu, pod nazivom Extreme Programming - Explained, Beck je objavio 1999. godine. Autor XP definiše kao disciplinu razvoja softvera, koja organizuje ljude da proizvode softver visokog kvaliteta na produktivniji način (Beck, 1999).

Osnovne proklamovane vrednosti XP razvojnog procesa su (Beck & Andres, 2004):

- **Jednostavnost** - podrazumeva što jednostavniju izradu funkcionalnosti softverskog rešenja, iz razloga što je jednostavniji kod uvek lakše prepraviti i održavati.
- **Komunikacija** - između članova tima u neformalnoj i usmenoj formi, iznad je komunikacije putem dokumenata.

- **Povratna sprega** - podrazumeva da svaka izgrađena vrednost u razvojnom procesu bude ocenjena od strane stejkholdera, čime se razvojnom timu obezbeđuju povratne informacije za sprovođenje kontinuirane potvrde kvaliteta i kontrole projekta.
- **Hrabrost** - podrazumeva podsticanje članova tima da samostalno donose odluke i rešavaju probleme.

Osnovni proklamovani principi XP razvojnog procesa su (Beck & Andres, 2004):

- **Brz povrat informacija** - podrazumeva kratke linije komunikacije u cilju razvoja proizvoda koji zadovoljava potrebe korisnika.
- **Usvajanje jednostavnosti** - predstavlja pravilo dizajniranja samo trenutne iteracije, bez konceptualnog pogleda na celokupno softversko rešenje. To je u skladu sa akronimom YAGNI od „*You aren't going to need it*“, koji govori o razvoju isključivo onih softverskih funkcionalnosti koje direktno doprinose ostvarivanju vrednosti za korisnika. Razvoj suvišnih funkcionalnosti utiče na rast troškova razvojnog projekta putem rasta kompleksnosti koda, a time i grešaka.
- **Inkrementalne promene** - su u skladu sa inkrementalnim razvojnim konceptom, koji je zajednički za sve agilne procese. Odnosi se na isporuku softverskih rešenja kroz male inkremente.
- **Prihvatanje promena** - je princip koji se primenjuje u celom periodu trajanja projekta. Promene se smatraju poželjnim ukoliko doprinose proizvodnji softverskog rešenja koje zadovoljava potrebe stejkholdera.
- **Kvalitetan rad** - ima za cilj isporuku kvalitetnog proizvoda. Kvalitetan proizvod je faktor uspeha razvoja koji se ne može kompenzovati, jer nema alternativu.

Pored navedenih vrednosti i principa XP se zasniva i na primeni dvanaest praksi, grupisanih u četiri kategorije (Jeffries, 2014):

1. Povratne informacije

- *Programiranje u paru* predstavlja tehniku kod koje dva programera dele jedan računar, s tim da dok jedan programira, drugi razmišlja o konceptualnim postavkama samoga procesa kodiranja i kontinuirano kontroliše napisan kod. Cilj je da se broj načinjenih grešaka u što većoj meri smanji. Programeri naizmenično menjaju svoje uloge u paru (Beck & Andres, 2004).
- *Planiranje igre* predstavlja osnovu procesa planiranja u XP-u. Pod igrom se misli na sastanak koji se održava po jednom za svaku iteraciju. Proces planiranja podeljen je na dva dela: planiranje isporuke (release-a) i planiranje iteracije. Planiranje isporuke počinje procesom identifikovanja zahteva sistema koji imaju visoku vrednost (za korisnika) i njihovom transformacijom u korisničke priče. Plan isporuke sadrži definisane funkcionalnosti koje će biti isporučene, kao i datum sledeće isporuke. Plan je rezultat dogovora programera i poslovnih stejkholdera, a determinisane funkcionalnosti u njemu mogu biti ažurirane

tokom vremena. Planiranje iteracije podrazumeva planiranje aktivnosti i zadataka programera za korisničke priče koje su predmet realizacije date iteracije (Melnik & Maurer, 2004).

- *Razvoj vođen testovima (Test Driven Development, u nastavku TDD)* podrazumeva izradu pojedinačnih testova i testova prihvatljivosti pre procesa izrade programskog koda. Programerima se na ovakav način olakšava rad jer imaju definisan pravac razvoja programskog koda, koji je ispravan jedino ako zadovoljava postavljene testove (Beck, 2003). Testove prihvatljivosti definiše korisnik, a cilj im je da potvrde ispravnost razvijenih funkcionalnosti u odnosu na specificirane zahteve kupca. Pojedinačne testove izrađuje razvojni tim, kako bi obezbedio provera ispravnosti napisanog koda. Obe vrste testova sprovode se automatizovano, s tim što se prva vrsta testova implementira po principu "otvorene kutije", dok se pojedinačni testovi realizuju po principima "crne kutije". Situacije u kojima su uspešno realizovani svi pojedinačni testovi, ali ne i test prihvatljivosti, implicira ponavljanje iteracije. Ponovljena iteracija ima za cilj otklanjanje svih utvrđenih korisnikovih zamerki. Ponavljena iteracija zahteva i ponovno sprovođenje pojedinačnih testova i testa prihvatljivosti. Postupak se ponavlja sve do momenta dok kupac ne potvrdi ispravnost razvijenih funkcionalnosti (Beck, 2003).
- *Ceo tim* je princip koji naglašava aktivnu ulogu stejkholdera, tokom čitavog razvojnog procesa, kroz njegovu spremnost da u svakom momentu pomogne razvojnom timu.

2. Kontinuirani proces

- *Kontinuirana integracija koda* može se sprovoditi na neki od sledećih načina (Fowler, 2006; Matković, 2011):

  - Zajednički direktorijum je mesto čuvanja programskog koda, pri čemu svaki član tima ima pravo da mu pristupa i menja ga. Predstavlja najkompleksniji način integracije, jer zahteva izuzetne napore u koordiniranju samog procesa, kako se kontinuiranom integracijom ne bi uništavao prethodno napisan i integrisani kod. Proces je olakšan primenom tehnike zamrzavanja koda u momentu njegove integracije, kao i tehnikom transakcija koje trebaju da osiguraju povratak na početno stanje, u slučaju nastanka greške tokom integracije.
  - Zajednički direktorijum sa zaključavanjem se razlikuje od prethodno opisane metode, jer za vreme trajanja integracije niko ne može da koristi osnovni kod. Ovakvim konceptom sprečava se mogućnost „preklapanja" u integraciji, ali se s druge strane, usporava proces rada jer je onemogućen paralelan rad više timova.
  - Osnovni kod i kod u zaštićenom okruženju je metod koji omogućava paralelan rad timova. Osnovni kod se po zaključavanju kopira u zaštićeno okruženje u kojem radi razvojni tim. Takođe, zaključavanje koda vrši se i prilikom kopiranja koda iz zaštićenog okruženja u osnovni kod. Metoda dozvoljava da više timova može napraviti kopiju osnovnog koda u sopstveno zaštićeno okruženje, pa u momentu integracije moguća je pojavi sledećih situacija: osnovni kod i kod iz zaštićenog okruženja su identični; osnovni

kod ima nove fajlove koji ne postoje u zaštićenom okruženju; kod u zaštićenom okruženju ima fajlove koji ne postoje u osnovnom kodu; svi fajlovi su isti, ali ima fajlova čija je sadržina različita. Iz ovih razloga integracija spada u najkompleksnije aktivnosti u procesu razvoja, pa je vrlo često sprovodi tim ljudi.

- Sinhronizacija koda iz zaštićenog okruženja i osnovnog koda je metoda kojom se pre svakog zaključavanja osnovnog koda vrši sinhronizacija, čime se eliminiše rizik od razlika u trenutku vraćanja koda iz zaštićenog okruženja u osnovni kod. Razlike koje mogu da se pojave tiču se izmena načinjenih u konkretnom zaštićenom okruženju i stanja poslednje sinhronizacije. Njihova eliminacija postiže se sinhronizacijom pre integracije. Paralelan rad timova i sinhronizacija koda znatno može biti olakšana upotrebom softverskih alata za tu namenu (npr. Microsoft Visual Source Safe, Rational Clear Case, Concurrent Versions System i sl.).

- *Refaktorisanje* je tesno povezano sa principom jednostavnosti, a ogleda se u kontinuiranim izmenama programskog koda, u cilju njegovog lakšeg razumevanja i izrade jednostavnijeg rešenja dizajna. Refaktorisanje podrazumeva izmenu interne strukture softverskog rešenja, kreiranjem novih metoda i klasa, premeštanjem metoda iz klase u klasu, premeštanjem metoda iz podklasa u nadklase i sl. Pojam kompozitnog refaktorisanja odnosi se na potrebu sprovođenja više opisanih refaktorisanja nižeg nivoa (Stephens & Rosenberg, 2003; Fowler et al., 1999).

  XP nema tipične aktivnosti za razvoj softverske arhitekture, kao što su analiza, sinteza i evaluacija, jer one ne proizvode poslovnu vrednost za korisnika, a iziskuju dodatne troškove (Babar, 2014). XP razvojni proces tehniku metafore i refaktorisanja smatra dovoljno dobrim alternativama za tradicionalan proces razvoja softverske arhitekture. Arhitektura u XP procesu razvoja nastaje postepeno, nakon svake iteracije, kao rezultat kontinuiranih izmena programskog koda (nascentna arhitektura), a ne iz neke unapred izgrađene strukture (Thapparambil, 2005; Beck, 2004).

- *Mali release* softverskih rešenja su vrlo efikasan način da se u kratkim vremenskim intervalima dobiju povratne informacije od stejkoldera, na osnovu kojih se sprovodi kontinuirano unapređenje kvaliteta proizvoda i sticanje poverenja kod stejkholdera u progres projekta.

3. Zajedničko razumevanje

- *Standardi kodiranja* su neophodni zbog primene tehnike kodiranja u parovima i smene uloga programera. Praksa podrazumeva definisanje standarda u pisanju koda i njihovo dostavljanje svim programerima u pismenoj formi.

- *Kolektivno vlasništvo koda* podrazumeva da svi učesnici u projektu imaju pravo da menjaju kod kada procene da za to postoji realna potreba. Ova praksa nadomešćuje ne postojanje

uloge softver arhitekte u XP procesu, čija je tradicionalna uloga podrazumevala poznavanje globalne slike rešenja i upravljanje promenama u kodu. Na ovaj način se u XP procesu odgovornosti softver arhitekte delegiraju na sve programere (Beck & Andres, 2004).

- *Jednostavan dizajn* je praksa koja podrazumeva da kod mora biti što jednostavniji u cilju njegovog boljeg razumevanja, lakšeg održavanja i dodavanje novih funkcionalnosti. Takođe, podrazumeva izvršenje svih testova, eliminisanje duplog koda i izgradnju najmanjeg mogućeg broja klasa, kako bi se ispoštovao YAGNI princip (Fowler, 2004).

- *Metafore sistema* predstavljaju pojednostavljenu sliku budućeg sistema i definišu se uglavnom na početku projekta, u formi priče ili grafičkog prikaza. Metafore imaju za cilj da predoče timu funkcionisanje sistema ili nekog njegovog dela. Metaforama se uspostavlja zajednička terminologija na projektu i omogućava konzistentno davanje imena klasama i metodama u kodu. Metafore predstavljaju tehniku razvoja softverske arhitekture u agilnim procesima, jer zajedno sa tehnikom refaktorisanja izgrađuju nascentni, pojavni dizajn softverskog rešenja. Pojavni, evoluirajući dizajn podrazumeva da strukture arhitekture nisu unapred definisane, već da proizilaze iz procesa implementacije tj. kodiranja. Kontinuirani razvoj funkcionalnosti zahteva i kontinuirano sagledavanje dizajna, kako bi se obezbedili uslovi za njihovu efikasnu implementaciju. Svako identifikovanje „neprijatnih mirisa“ u dizajnu rešenja, zahteva sprovođenje refaktorisanja programskog koda (Fowler, 2004).

4. Zaštita programera

Održivi tempo rada podrazumeva da svi učesnici u XP procesu razvoja imaju zagarantovanu četrdesetčasovnu radnu nedelju. Cilj ove prakse je da se spreči premor zaposlenih, jer se time dovodi u pitanje njihova kreativnost, produktivnost, motivisanost i timski duh.

3.3.3.1 XP proces razvoja

Izgradnja softverskog rešenja XP procesom realizuje se kroz razvojne faze, prikazane na slici 3.16:

1. Istraživanje - predstavlja prvu razvojnu fazu koja je fokusirana na prikupljanje korisničkih zahteva i njihovo oblikovanje u korisničke priče. Korisničke priče predstavljaju artifakt koji u sažetoj tekstualnoj formi izrađuju korisnici, da bi razvojnom timu predočili potrebne funkcionalnosti koje budući sistem treba da podrži. U procesu definisanja korisničkih priča učestvuju i članovi razvojnog tima, koji kroz set pitanja i sugestija vrše neki vid kontrole i usmeravanja rada korisnika. Razvojni tim je odgovoran da, nakon korisnikovog definisanja redosleda razvoja korisničkih priča, izvrši procenu potrebnog vremena i resursa za realizaciju svake priče. Uz svaku definisanu priču korisnici su obavezni da izrade i test prihvatljivosti

priče, koji je osnovno sredstvo za testiranje razvijenih funkcionalnosti implementirane korisničke priče. Korisnici su u mogućnosti da tokom celog razvojnog procesa sprovode ažuriranje definisanih priča. Načinjene izmene realizuju se u okviru predstojećih iteracija projekta. Prosečno vreme razvoja jedne korisničke priče je do sedam dana.

2. Planiranje - podrazumeva izradu sledećih artifakata: plan release-a, plan iteracije i dnevni plan. Release plan integriše planove svih distribucija softverskog rešenja kao i planove iteracija. Planiranjem release-a, kupac i razvojni tim definišu korisničke priče koje će biti razvijane u prvih nekoliko iteracija. Kupac je dodatno odgovoran za utvrđivanje prioriteta korisničkih priča, prema značaju koji imaju za krajnjeg korisnika, a razvojni tim da proceni ukupno vreme i napote potrebno za razvoj softverskog rešenja. Zbog svoje vremenske (planirani datumi) i sadržajne dimenzije (planirane funkcionalnosti), plan releas-a je sredstvo za merenje progresa razvoja projekta (Cohn, 2005).

Artifakt plan iteracije uključuje artifakt plan implementacije korisničkih priča koje ulaze u datu iteraciju. Razvojni tim u okviru artifakta plan implementacije korisničke priče definiše vreme potrebno za njen razvoj, dok kupac utvrđuje prioritet realizacije korisničkih priča unutar svake iteracije. Planom iteracije definiše se set aktivnosti i zadataka razvojnog tima, kao i resursi neophodni za implementaciju korisničkih priča. Uobičajeno vreme trajanja jedne iteracije je od jedne do tri nedelje. Vreme utrošeno na realizaciju zadatka u okviru iteracije mora biti u opsegu od jednog do tri osmočasovna radna dana. Sastavni delovi plana iteracije su i dnevni planovi koji se definišu direktnom komunikacijom na svakodnevnim desetominutnim sastancima tima. Cilj sastanaka i dnevnih planova je pregled obavljenog/neobavljenog posla od prethodnog dana, planiranje zadataka za tekući dan, prikaz izveštaja o testovima prihvatljivosti razvijenih funkcionalnosti, iznošenje identifikovanih problema u razvoju (Cohn, 2005).

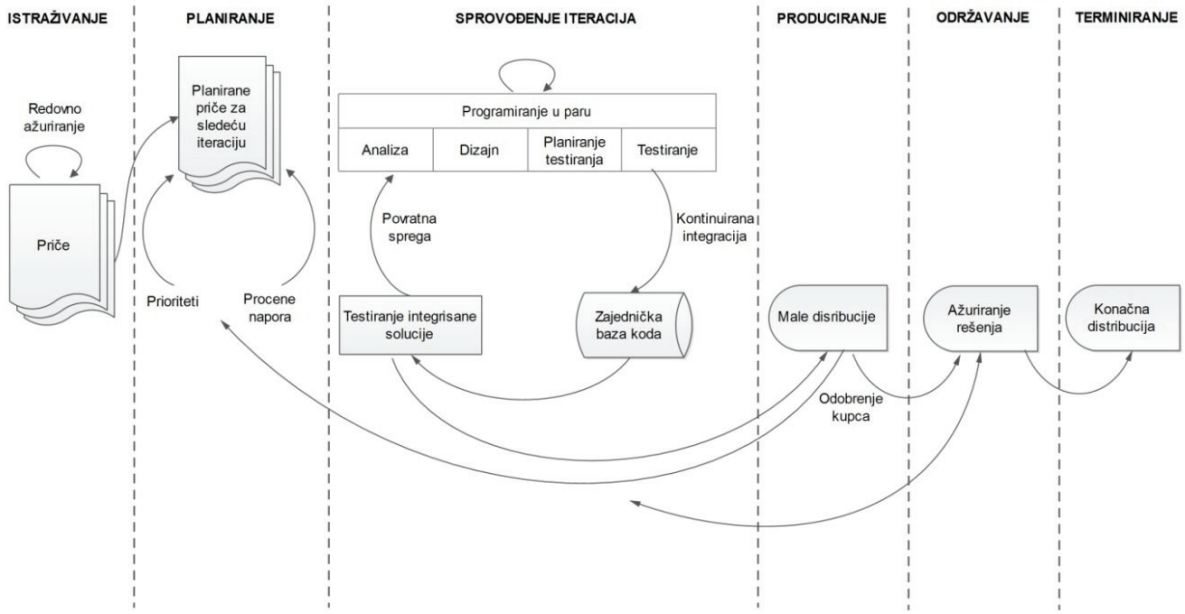
3. Sprovođenje iteracija - je faza koja podrazumeva primenu seta sekvencijalnih razvojnih koraka: analiza, dizajn, kodiranje i testiranje. Cilj analize je tumačenje korisničkih priča i funkcionalnosti koje je potrebno razviti. Dizajn podrazumeva definisanje ključnih metafora sistema, na osnovu kojih se pristupa identifikovanju ključnih objekata i njihovih međusobnih odnosa. Identifikovani objekti se grupišu u klase, čija dalja elaboracija se sprovodi tehnikom klasa - odgovornost - saradnja (Class - Responsibility - Collaboration, u nastavku CRC kartica). CRC kartice se izrađuju na papiru u obliku T strukture, pri čemu se u zaglavlju navodi naziv klase i ime roditeljske klase, ukoliko postoji. Sa leve strane T strukture beleže se operacije klase, koje će obezbediti izvršavanje potrebnih funkcionalnosti sistema, a sa desne strane, unosi se set klasa sa kojima je data klasa povezana. Nakon ovako izgrađenog modela dizajna, pristupa se implementaciji korisničkih priča, odnosno izradi programskog koda, primenom preporučenih XP praksi (opisane prethodnim poglavljem rada: programiranje u paru, angažovanje kupca u procesu programiranja, utvrđivanje standarda za pisanje koda, kodiranje vođeno testovima, refaktorisanje koda, uspostavljanje zajedničke baze koda, sprovođenje kontinuirane integracije koda i uspostavljanje četrdeset časovne radne nedelje).

Testiranje razvijenih funkcionalnosti vrši se putem izgrađenih testova prihvatljivosti i pojedinačnih testova. Pozitivni rezultati testerinja impliciraju kompletiranje korisničke priče i otpočinjanje naredne razvojne faze.

5. Produkcija - podrazumeva fazu u okviru koje se vrši povezivanje izgrađenih rešenja u novi release softvera. Isporuka release-a vrši se u opsegu od 30 do 180 dana.

6. Održavanje - nakon isporuke verzije softverskog rešenja, kontinuirano sprovodi se njegovo održavanje tj. ažuriranje.

7. Terminiranje - predstavlja fazu kojom se završava ažuriranje isporučenog softverskog rešenja, jer je uspostavljena njegova konačna verzija.



Slika 3.16 XP proces razvoja
Izvor: (Prilagođeno prema (Jeffries, 2014)

Navedene razvojne faze sastoje se od seta aktivnosti i artifakata, koje realizuju odgovarajuće XP uloge (Corporation IBM, 2006):

- Trener - je odgovoran za aktivnosti praćenja razvojnog procesa i pružanje pomoći razvojnom timu u realizaciji njihovih razvojnih aktivnosti.
- Kupac - aktivno učestvuje u toku celog razvojnog procesa, zbog čega uz programera predstavlja najznačajniju ulogu u XP procesu. Odgovoran je za definisanje zahteva, redosled razvoja funkcionalnosti sistema i proveru svih razvijenih funkcionalnosti.
- Programer - ima najvažnije mesto u razvojnom timu, a primarna uloga mu je implementacija korisničkih priča (definisanih zahteva) u kod i razvoj jednostavnog dizajna.
- Administrator - definiše i vrši podešavanje razvojnog okruženja, brine o fizičkom razmeštaju opreme i o standardizaciji alata i njihovom podešavanju.

- Kontroler - je zadužen za praćenje realizacije korisničkih priča, kroz realizaciju iteracija, release-a i testova prihvatljivosti.
- Tester - je odgovoran za konačan izgled testova koje je kupac definisao, da izvrši kodiranje testova, sprovede ih i pripremi okruženje u kojem će se testovi sprovoditi.

Pregled svih XP uloga, njihovih aktivnosti i artifakata za koje su zaduženi dat je u tabeli 3.11:

Tabela 3.11 Uloge, aktivnosti i artifakti XP procesa razvoja

ULOGA	AKTIVNOSTI	ARTIFAKTI
Trener	Prilagođavanje i unapređenje razvojnog procesa	Rezultati praćenja
	Objašnjavanje procesa	Izveštaji i beleške sa sastanaka
	Unapređenje veština razvojnog tima	
	Usmeravanje procesa	
	Rešavanje konflikata	
Kupac	Podešavanje obima (granica) iteracije	
	Definisanje korisničkih testova	Korisničke priče
	Definisanje iteracija	Tim kupaca (korisnika)
	Definisanje release-a	Plan iteracija
	Definisanje vizije	*Release Plan*
	Izveštavanje o rezultatima testiranja	Vizija
	Izveštavanje o statusu projekta	Testovi prihvatljivosti
Programer	Definisanje inženjerskih zadataka	
	Definisanje standarda kodiranja	Standardi kodiranja
	Procena zadataka	Metafore sistema
	Procena korisničkih priča	Produkcioni kod
	Implementacija spike-ova	Verzija softvera
	Integracija i izgradnja verzije softvera	Pojedinačni testovi
	Refaktorisanje	
	Pisanje koda	
Administrator	Podešavanje razvojnog okruženja	
Kontroler	Praćenje progresa iteracija	Izveštaj o napretku
	Praćenje progresa release-a	Podaci o merenju
Tester	Automatizacija korisničkih testova prihvatljivosti	Podaci testa
	Pokretanje korisničkih testova	Rezultati testa
	Podešavanje okruženja za testiranje	Alati za testerinje

Izvor: (Prilagođeno prema (Jeffries, 2014))

3.3.4 Lean

Prvi koncepti Lean menadžment filozofije potiču iz Japana i to još iz perioda posle II svetskog rata. Široj javnosti postaju poznati tek devedesetih godina prošlog veka, nakon objavljivanja Womack-ovih radova (Womack & Jones, 2009; Womack & Jones, 2003; Womack, Jones, Roos, 1990) i knjige "Mašina koja je promenila svet". U knjizi je opisano empirijsko istraživanje koje je sprovedeno nakon II svetskog rata, a obuhvatilo je najveće svetske kompanije na polju automobilske industrije. Istraživanje je predstavljalo zajednički

program univerziteta, kompanija i vlade (International Motor Vehicle Program, u nastavku IMVP).

Rezultati pomenutog istraživanja su pokazivali da se japanske kompanije, naročito Toyota, izdvajaju od drugih svetskih kompanija u svakom pogledu, pri čemu je i zaključeno da Toyota ima razvijen jedinstveni Lean metod - Toyota Production System (u nastavku TPS). Ideja ovog metoda potekla je od tri inženjera Toyote: Kiichiro Toyoda, Eiji Toyoda i Taiichi Ohno (Womack, Jones, Roos, 1990).
Osnovna ideja filozofije je da svaka organizacija proklamovane Lean principe implementira na sopstveni način, kroz kontinuirano učenje iz sopstvenog iskustva. Iz tog razloga ne postoje utemeljene instrukcije za njihovu efikasnu realizaciju.

Lean filozofija fokusirana je na smanjenje ukupnih troškova, na poboljšavanje sveukupnog kvaliteta i kvaliteta proizvoda koji se isporučuje korisnicima, skraćenje vremena isporuke proizvoda i na povećanje zadovoljstva korisnika. Ostvarivanje ovih ciljeva moguće je primenom sledećih ključnih Lean principa:

1. Princip vrednost i otpad (Value and Waste)

Prvi Lean princip podrazumeva definisanje vrednosti u okviru izvršavanja nekog procesa. Koncept vrednosti se definiše iz perspektive kupca, u terminima koji se vezuju za određeni proizvod i sa kojima se može meriti da li su kupčeve potrebe zadovoljene po datoj ceni i u datom vremenu. Lean filozofija naglašava ulogu kupca/korisnika za ostvarivanje uspeha ma kog procesa.

Otpad podrazumeva sve aktivnosti nekog procesa, koje ne doprinose stvaranju vrednosti za krajnjeg korisnika. Pošto se takve aktivnosti ne mogu u potpunosti isključiti, jer su sastavni deo i najboljih procesa, cilj je eliminisati ih u najvećoj mogućoj meri. Eliminisanje otpada iz procesa ostvaruje se ili kroz bolju iskorišćenost zaposlenih i resursa ili kroz poboljšavanje načina na koje se sprovode aktivnosti u okviru procesa. Lean razlikuje dva tipa otpada. Prvi tip uključuje aktivnosti koje ne proizvode vrednost za korisnika, ali se moraju realizovati usled korišćenja određene metode, tehnologije i sl. Ovaj tip otpada je teže elimisati, jer može zahtevati reinženjering procesa ili kupovinu novih alata i opreme. Drugi tip otpada, podrazumeva aktivnosti koje je moguće elimisati, bez većih dodatnih ulaganja (Womack & Jones, 2003).

U okviru Lean TPS metode identifikovano je sedam klasa otpada (Liker & Meier, 2006):

1. *Hiperprodukcija,* podrazumeva proizvodnju velike količine robe, koja nije zasnovana na realnoj tražnji kupaca. Ova vrsta otpada generiše i druge otpade: prevelik broj zaposlenih, skladištenje robe, dodatne troškove prevoza viška zaliha.

2. *Čekanje,* podrazumeva da radnici zbog manjkavosti procesa proizvodnje, često imaju prazan hod, čekajući naredni korak obrade, potrebne alate, snabdevanje, deo ili pak uopšte ne rade zbog nedostatka zaliha, zastoja opreme ili uskog grla kapaciteta.
3. *Transport,* predstavlja otpad u slučaju premeštanja materijala, delova i gotovih artikala u/ili iz skladišta ili između različitih dislociranih procesa.
4. *Višak obrade,* generiše otpad u slučajevima kada se u proizvodnji javljaju nepotrebni, suvišni koraci obrade. Neefikasna obrada, usled lošeg alata ili dizajna proizvoda, izaziva nepotrebno kretanje i proizvodnju defekata. Otpad se generiše i u slučaju proizvodnje većeg kvaliteta proizvoda nego što je to realno potrebno.
5. *Zalihe,* podrazumevaju da višak sirovina ili gotovih proizvoda implicira duže vremenske rokove, zastarelost, oštećenost robe, transport i troškove skladištenja. Takođe zalihe dovode do proizvodne neravnoteže, neblagovremene isporuke od dobavljača, nedostatka i zastoja opreme.
6. *Kretanje,* kao otpad podrazumeva sve akcije radnika kao što su: traženje, premeštanje, slaganje delova, alata i sl. i mogu se lako eliminisati.
7. *Nedostaci - defekti,* podrazumevaju proizvodnju neispravnih delova i proizvoda što zahteva njihove naknadne korekcije, koje troše dodatno vreme, trud i dr. resurse.

Opisane vrste otpada potiču iz proizvodnog okruženja, ali se smatraju opštim klasama otpada Lean filozofije (Liker, 2004; Womack & Jones, 2003).

2. Princip toka vrednosti (The Value Stream)

Vrednost nekog proizvoda treba da se gradi kontinuirano, od momenta identifikovanja kupčevih zahteva do isporuke gotovog proizvoda. Feigenbaum (2008) je ovaj tok opisao pojmom industrijski ciklus, a Porter (2008) pojmom lanac vrednosti. Lean princip izgradnja toka vrednosti podrazumeva vizuelno predstavljanje poslovnih aktivnosti, koje se sprovode u okviru nekog procesa, i to od faze naručivanja do faze isporuke gotovog proizvoda.

Vizuelno se predstavljaju, kako aktivnosti koje dodaju vrednost krajnjem proizvodu, tako i aktivnosti koje se smatraju otpadom, ali ih je nemoguće eliminisati. Nakon vizualizacije seta aktivnosti nekog procesa, neophodno je pristupiti njihovoj pojedinačnoj analizi, a potom i analizi njihove međusobne povezanosti u okviru predstavljenog toka vrednosti. Primena ovog principa ima za cilj da poveća razumevanje ključnih aktivnosti nekog procesa, što je osnova za njihovo kontinuirano poboljšavanje, kako bi se dostigao krajnji cilj: što veća isporučena vrednost kupcu (Womack & Jones, 2003).

Value stream mapping (u nastavku, VSM) je alat koji se koristi kao efikasna podrška u identifikovanju toka vrednosti nekog procesa (Rother & Shook, 1999). Alat ima standardizovan jezik za predstavljanje procesa (toka njegovih aktivnosti), čiji je rezultat dijagram trenutnog stanja nekog procesa ("as is" dijagram realnog stanja). Na osnovu ovog dijagrama razvija se plan akcija i razvojnih mogućnosti, u cilju kreiranja dijagrama idealnog

stanja ("should be" dijagram) procesa, kojem organizacija treba da teži u bućnosti (Rother & Shook, 1999; Womack & Jones, 2003; Hines & Taylor, 2000).

3. Princip protok (Flow)

Nakon definisanja vrednosti i dizajniranog toka izgradnje vrednosti nekog procesa, neophodno je obezbediti da aktivnosti toka koje dodaju vrednost teku kontinuirano (bez zastoja, preskoka i uskih grla), dodajući maksimalnu vrednost na svakoj tačci toka vrednosti. Da bi se postigao kontinuirani protok, neophodno je prethodno preispitati sve tradicionalne procedure rada i prakse, u cilju eliminisanja svih zastoja i ponavljajućih operacija. To zapravo znači, da kada se resursi jednom uključe u neki proces, treba što brže da prođu kroz njega i da izađu u formi završenog proizvoda. Završeni proizvod se predaje narednom korisniku, koji može biti unutar ili izvan organizacije (unutrašnji ili spoljnji korisnik). Opisan način rada jeste tzv. "proizvodnja u jednom komadu", kojim se eliminiše dodatni rad, zalihe i serijska proizvodnja i smanjuje ukupan rad (Womack & Jones, 2003).

Kontinuirani protok je jednako važan i prilikom transformacije informacija, podataka i znanja u novi proizvod ili uslugu, kao i u opisanom procesu prozvodnje (Hines et al., 1999).

4. Princip vući (Pull)

Princip pull ("vuci") podrazumeva planiranje i proizvodnju u skladu sa realnim potrebama i u momentu dobijanja naloga od strane kupaca/korisnika. Princip važi za sve procese i sve aktivnosti nekog procesa. Dakle, proces višeg nivoa u hijerarhiji ne može da otpočne realizaciju, dok ne dobije signal od procesa nižeg nivoa u hijerarhiji. Uspešna primena ovog principa zahteva efikasnu implementiranost prethodna tri principa, jer efikasan tok vrednosti i nesmetan protok, dovode do smanjivanja rokova u okviru procesa, što omogućuje da pull sistem proizvodnje radi kako je i predviđen, uz egzistiranje minimalnog nivoa zaliha (Womack & Jones, 2003).

5. Princip savršenstva (Perfect)

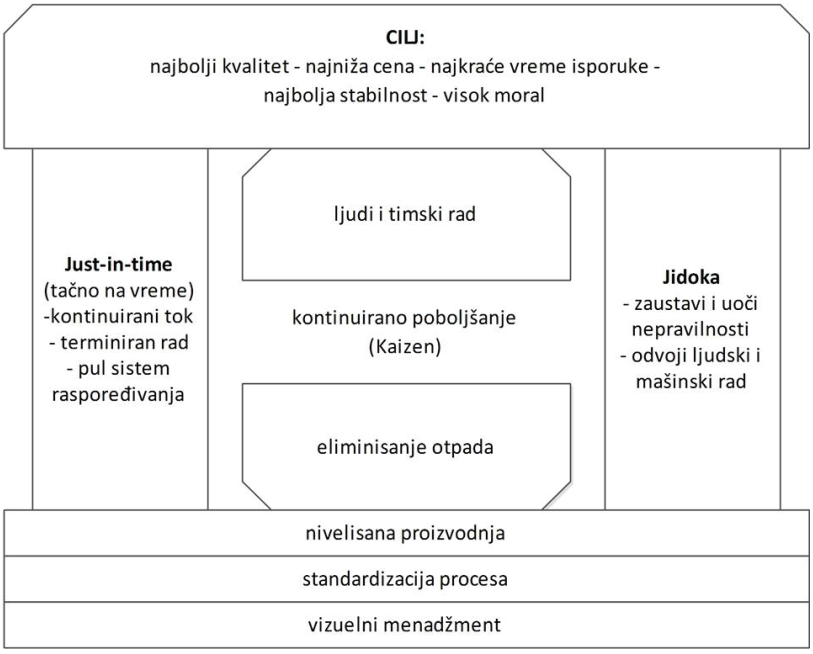
Stvaranje Lean organizacije je postupak dugotrajnog unapređenja procesa. Cilj je težiti savršenstvu, ujedno shvatajući da ga je nemoguće ostvariti. Kaizen ili kontinuirano poboljšanje je sličan pojmu kvaliteta, koji opisuje Total Quality Menagement koncept (u nastavku TQM). Kaizen znači kontinuirani razvoj svih praksi, u malim koracima, ka postepenom poboljšavanju realizacije celog procesa. Kaizen je prirodni nastavak radikalnih promena u procesu, u smislu eliminisanja otpada iz njega i promene načina njegovog realizovanja.

Kontinuirano poboljšanje obezbeđuje da se izvršene promene ne završavaju jednim radikalnim poboljšanjem, već da se nastave. Promene koje zahteva Lean ne mogu se izvršiti

bez podrške na svim nivoima organizacije, od menadžera do radnika. Kaizen aktivnosti moraju sprovoditi i obični radnici, jer oni znaju posao i proces, te su i najkompetentniji da identifikuju šta je dobro, a šta treba poboljšati u procesu. Radnici ili grupe radnika na taj način rešavaju probleme koji se javljaju u njihovom radu, edukuju sebe i druge o relevatnim temama i sarađuju sa drugim nivoima organizacije po pitanju novih ideja za poboljšavanje procesa.

Osnova ovog principa leži u "krugovima kvaliteta" koji su poznati i pod nazivom Demingov ciklus ili "Plan-Do-Check-Act" (u nastavku, PDCA) (Ishikawa, 1985). Drugi važan koncept principa savršenstva jeste standardizacija rada, kao logičan sled stvari kada se neko poboljšanje sprovede i verifikuje kao korisno za proces. Cilj standardizacije je da svi zaposleni znaju redosled i način sprovođenja aktivnosti u okviru nekog procesa (Liker, 2004; Womack & Jones, 2003).

Na slici 3.17 prikazana je organizacija principa i praksi Lean proizvodnog sistema:



Slika 3.17 Lean kuća
Izvor:(Liker, 2004)

3.3.4.1 Lean u razvoju softvera

Womack Jones (2003) su smatrali da opšti Lean principi mogu biti primenjeni u gotovo svim vrstama organizacijama, uz izvesna prilagođavanja specifičnom kontekstu primene.

Različiti autori bavili su se prilagođavanju opštih Lean principa domenu razvoja softvera (Poppendieck & Cusumano, 2012; Ebert, Abrahamsson, Oza, 2012; Holden, 2010; Larman & Vodde, 2009; Liker & Morgan, 2006; Middleton, Flaxel, Cookson, 2005; Womack &

Jones, 2003; Poppendieck & Poppendieck, 2003; Highsmith, 2002; Middleton, 2001; Ahlstrom, 1998; Ohno, 1988).

Principi ovih autora sistematizovani su i prikazani u tabeli 3.12. Bitno je napomenuti da oni ne predstavljaju alternativu za opisane opšte Lean principe, već su komplementarni sa njima.

Tabela 3.12 Lean principi po autorima

AUTORI	LEAN PRINCIPI
Ahlstrom i Karlsson (1996, 1998)	1. Eliminacija otpada; 2. Kontinuirano poboljšanje; 3. Proizvod bez nedostataka; 4.Planiranje u skladu sa potrebama (pull sistem); 5. Multifunkcionalan tim; 6. Decentralizovane odgovornosti; 7. Tim lider; 8. Vertikalni informacioni sistem.
Ebert i dr. (2012)	1. Fokus na korisnicima; 2. Smanjivanje otpada; 3. Osnaživanje tima; 4. Efikasnost zasnovana na toku vrednosti; 5. Kontinuirano poboljšanje.
Highsmith (2002)	1 .Zadovoljan korisnik je najviši prioritet; 2.Uvek pružiti korisniku najveću vrednost za određenu sumu novca; 3. Uspeh zavisi od aktivnog učešća korisnika; 4. Razvoj projekta je timski rad; 5. Sve je promenljivo; 7. Potpunost, ne izgradnja; 8. 80% rešenja danas je više i bolje nego 100% rešenja sutra; 9. Minimalizam je esencijalan; 10. Potrebe određuju tehnologiju; 11. Rast proizvoda je rast njegovih funkcija, a ne rast veličine; 12. Nikada ne gurajte lean razvoj izvan njegovih granica.
Holden (2010)	1. Eliminisanje otpada; 2. Kontinuirani tok, rad sa minimalnim kašnjenjem; 3. Osnaživanje radnika; 4. Automatizacija; 5. Rešavanje uzroka datog problema; 6. Kontinuirano poboljšanje.
Larman i Vodde (2009)	1. Upravljačke odluke na dug rok; 2. Protok; 3. Pull system – razvoj potrebnih funkcija; 4. Smanjti varijabilnost u odnosu na opterećenje; 5. Prilagoditi se zaustavljanju i rešavanju identifikovanog problema; 6. Master prakse; 7.Jednostavan vizuelni menadžment; 8. Dobro testirana tehnologija; 9. Lider u timu; 10. Razviti izuzetne ljude; 11. Pomoći partnerima da primene Lean; 12. Donosi odluke sporo i sa konsenzusom; 13. Idi da vidiš; 14. Kaizen
Liker i Morgan (2009)	1.Definisanje vrednosti iz perspektive korisnika; 2. Odlučivati što je kasnije moguće; 3. Kreiranje nivoa toka procesa; 4. Standardizacija; 5. Postojanje glavnog inženjera sistema; 6 Balans između funkcionalne stručnosti i unakrsne funkcionalne integracije; 7. Razvojem se podiže tehnička kompetentnost svih inženjera u timu; 8. Uključivanje dobavljača u razvoj proizvoda; 9.Izgradnja kulture koja podržava izvrsnost; 10. Izgradnja okruženja koje kontinuirano uči; 11. Prilagođavanje tehnologije zaposlenima; 12. Jednostavna i vizuelna komunikacija; 13. Upotreba alata u svrhu organizacije koja uči.

Middleton (2001)	1.Kontinuirano poboljšavanje kvaliteta; 2. Osnaživanje radnika u preuzimanju odgovornosti; 3. Prevencija grešaka, ne slučajna detekcija ; 4. Jednostavno, vizuelno merenje kvaliteta; 5. Automatsko merenje kvaliteta, uređajima.
Middleton i dr. (2005)	1.Kontinuirani protok obrade/razvoja; 2. Korisnički definisana vrednost; 3. Dizajniraj krosfunkcionalnu strukturu i tok; 4. Uobičajeni tempo rada; 5. Poveži procese; 6. Standardizuj procedure; 7. Eliminisati ponovljeni rad (prepravke); 8. Balansiraj opterećenje; 9. Isporučl rezultate; 10. Odluke zasnovane na podacima; 11. Minimiziraj zalihe
Ohno (1988)	1.Eliminisanje otpada; 2. Kvalitetan razvoj i gradnja; 3. Automatizacija; 4. Umeren rast proizvodnje, razvoja; 5. Tok procesa; 6. Smanjenje troškova; 7. Osnaživanje tima; 8. Članovi tima poseduju više veština; 9. Proizvodno nivelisanje; 10. Standardizovan rad; 11. Identifikovanje izvora problema; 12. Raspoređivanje u skladu sa potrebama.
Poppendieck i Poppendieck (2003)	1. Eliminisanje otpada; 2. Pojačati učenje; 3. Odluke donositi što je kasnije moguće; 4. Isporučiti što je brže moguće; 5. Izgraditi integritet; 6. Sagledaj celinu; 7. Osnaži tim.
Poppendieck i Cusumano (2012)	1.Optimizuj celinu; 2. Eliminiši otpad; 3. Kvalitetna izrada, razvoj; 4. Uči konstantno; 5. Isporučuj brzo; 6. Uključi svakog; 7. Kontinuirano poboljšavaj
Womack i Jones (2003)	1. Definisati vrednost za kupca; 2. Definisati tok vrednosti; 3. Omogući protok vrednosti; 4. Razvijaj i proizvodi u odnosu na kupčeve potrebe i naloge (pull sistem); 5. Teži izvrsnosti, savršenstvu.

Narednim tekstom biće opisani ključni Lean principi u razvoju softvera, oko kojih su saglasni svi navedeni autori:

1. Definisati vrednost za kupca eliminisanjem otpada

Highsmith (2002) definisanje vrednosti posmatra kao identifikovanje karakteristika proizvoda kojim se zadovoljavaju korisnikove potrebe u određenom vremenu i za određeni novac. Womack i Jones (2003) smatraju da vrednost mogu i treba da definišu samo korisnici, dok proizvođači treba da je kreiraraju. Svaki specifičan proizvod ima specifičnu vrednost, uključujući pritom i specifičnu cenu za specifičnog korisnika. Middleton i ostali (2005) smatraju da je uspešna analiza korisničkih zahteva bitan preduslov za razvoj proizvoda koji ostvaruje vrednost korisniku. Stoga smatraju da svi članovi tima, u saradnji sa ključnim stejkholderima, treba da učestvuju u analizi i prioritizaciji korisničkih zahteva.

Prvi Lean princip razvoja softvera pored koncepta vrednosti uključuje i koncept otpada. Poppendieck M. i Poppendieck T. (2003) sistematizovali su klase otpada u procesu razvoja softvera. Komparacija opštih klasa otpada sa klasama otpada u domenu razvoja softvera prikazana je u tabeli 3.13.

Tabela 3.13 Opšte klase otpada vs. klase otpada u razvoju softvera

OTPAD U PROIZVODNJI	OTPAD U RAZVOJU SOFTVERA
Zalihe u procesu	Delimično urađen posao
Više obrade	Dodatni procesi/ponovni rad/ponovno učenje
Hiperprodukcija	Dodatne (suvišne) funkcije
Transport	Prebacivanje zadataka
Kašnjenje	Kašnjenje
Kretanje	Kretanje
Greške	Greške

Izvor: (Poppendieck & Poppendieck, 2003)

Delimično urađen posao se može identifikovati tokom celog razvojnog procesa i smatra se esencijalnim otpadom. Ovim otpadom smatra se programski kod za koji razvojni tim nema informacije po pitanju kvaliteta razvijenih funkcionalnosti, niti povratne informacije o zadovoljstvu korisnika. Otpad je i programski kod koji je završen, ali nije testiran ili dokumentovan., kao i kod koji nije integrisan u celinu i testiran u realnom okruženju (Milunsky, 2009; Poppendieck & Poppendieck, 2003).

Dodatni proces je otpad koji se odnosi na prepravku završenih softverskih funkcija usled lošeg kvaliteta koda. Drugi primer takvog otpada je neblagovremeno dokumentovanje programskog koda, što implicira ponovno učenje razvojnog tima u slučaju njegovih naknadnih izmena, uzrokovanih nezadovoljstvom korisnika ili neblagovremenim identifikovanjem grešaka. Ponovno učenje realizovanog koda iziskuje ulaganje dodatnog vremena i novca. Otpad nastaje i u slučaju lošeg planiranja projekta, kada se programerima nasumično dodeljuju funkcije koje trebaju biti razvijane, menjane i sl. Takođe, visoki troškovi projekta indukuju se i otklanjanjem programskih grešaka nakon postavljanja aplikacije u realno okruženje, ali i usled loše komunikacije i nedeljenja znanja između članova tima. Ovakav otpad bi se mogao umanjiti adekvatnom primenom pojedinačnih testova, pre isporuke aplikacije u realno ukruženje, kao i upotrebom wiki alata (Naftanaila & Paul, 2009; Poppendieck & Poppendieck, 2003).

Izrada dodatnih funkcija softvera predstavlja otpad usled hiperprodukcije, odnosno razvoja funkcionalnosti softvera koje korisnici nisu identifikovali kao potrebne. Razvoj suvišnih funkcionalnosti troši resurse i implicira rast ukupnih troškova razvoja. Drugim rečima, zahteva dodatni rad na njihovom kompajliranju, testiranju i integraciji, utiče na rast

kompleksnosti programskog rešenja i veću mogućnost pojave grešaka. Indirektno utiče na porast troškova održavanja rešenja i na rizik od potencijalnog neuspeha projekta (Naftanaila & Paul, 2009; Poppendieck & Poppendieck, 2003).

Prebacivanje zadataka, je vrsta otpada koja utiče na usporavanje progresa procesa razvoja i na neispunjavanje principa: "isporuči što je brže moguće". Primer ove vrste otpada je kada programer ispostavlja svoj kod drugom pragrameru. Ako isti nije adekvatno dokumentovan, drugi programer će morati utrošiti vreme i energiju da bi ga razumeo, stim da u slučaju pogrešnih pretpostavki u tumačenju dobijenog koda mogu nastati ogromne greške u sistemu. Stoga je važno da sve razvijene funkcionalnosti budu dokumentovane i prilikom njihove isporuke testerima. Slična vrsta otpada nastaje i u slučaju isporuke neadekvatno dokumentovanog i testiranog koda klijentu, jer se indukuje veći broj poziva klijenata za dodatnom podrškom u radu softvera (Naftanaila & Paul, 2009; Poppendieck & Poppendieck, 2003).

Kašnjenja/čekanja u procesu razvoja softvera mogu nastati već na samom početku projekta, ali i kroz sve druge razvojne faze: od identifikovanja ključnih stejkholdera i njihovog aktivnog uključivanja u razvojni proces, preko preteranog dokumentovanja zahteva, predugog razmatranja i odobravanja, čekanja na povratne informacije od testera i sl. Smanjenjem otpada koji potiče od kašnjenja u procesu razvoja, direktno se povećava vrednost za korisnika i stepen njegovog zadovoljstva. Brzina kojom tim može da odgovori na promene i nove korisničke zahteve direktno je proporcionalna sistematskom kašnjenju u okviru razvojnog procesa (Naftanaila & Paul, 2009; Poppendieck & Poppendieck, 2003).

Kretanje kao vrsta otpada u procesu razvoja softvera može nastati iz više razloga: ukoliko stejkholderi sistema ne rade zajedno (analitičari zahteva, dizajneri sistema, programeri, testeri i korisnici); ukoliko se članovi razvojnog tima prebacuju sa jednog zadatka na drugi, pa troše dodatno vreme da bi ušli u normalan tok rada i podsetili se zadataka na kojima nisu radili izvesno vreme; ukoliko članovi razvojnog tima pripadaju istovremeno više od jednom razvojnom timu, ili pak rade na nekoliko razvojnih projekata istovremeno (Naftanaila & Paul, 2009; Poppendieck & Poppendieck, 2003).

Nedostaci proizvoda se takođe smatraju otpadom, a moguće ih je u velikoj meri eliminisati kontinuiranim procesom integracije i testiranjem. Eliminisanjem nedostataka proizvoda direktno se povećava njegova vrednost. Bitno je greške identifikovati što ranije u procesu razvoja, jer i najmanji problem koji nastane nakon isporuke proizvoda korisniku predstavlja ozbiljan otpad jer izaziva velike finansijske gubitke (Poppendieck & Poppendieck, 2003; Beck, 1999).

Lean filozofija predviđa i prakse za eliminisanje otpada iz procesa razvoja softvera. Najzastupljenija je Just in Time praksa (u nastavku JIT) i dizajn toka izgradnje vrednosti. JIT praksa podrazumeva da se razvijaju prave stvari (u smislu razvoja samo onih funkcija softvera

koje je korisnik zahtevao) u pravo vreme (date funkcije softvera razvijaju se u skladu sa korisničkim preferencijama i prioritetima) i u pravom obimu. Praksa dizajniranje toka izgradnje vrednosti podstiče menadžere da prate ceo razvojni ciklus projekta, evidentirajući svaki zadatak i vreme potrebno za njegovo izvršavanje. Na taj način menadžeri su u stanju da sagledaju strukturu ukupnog razvojnog projekta, identifikujući pri tome nepotrebne poslove koje treba kategorisati kao otpad.

2. Odlučiti što je kasnije moguće

Princip podrazumeva da se bilo koji posao u procesu razvoja ne sprovodi pre nego što je njegov rezultat potreban sledećem koraku u tom procesu.

Poppendieck M. i Poppendieck T. (2003) su opisali dva moguća pristupa u donošenju odluka: „breadth-first decision" i "depth-first decision". Depth-first pristup donošenja odluke pogodan je u situacijama kada odluku donosi ekspert koji je spreman da odlučuje rano i kada su zahtevi fiksni tj. ne očekuju se njihove izmene nakon donošenja odluke. Ovakav način donošenja odluka tipičan je za sekvencijalni razvoj softvera.

U slučajevima kada je potrebno rešavati nov problem i u kratkom vremenskom periodu, pogodniji pristup za donošenje odluka jeste breadth-first decision. Breadth first odluke su pogodne i u situacijama kada su zahtevi promenljivi i kada ne postoji način da se izgradi funkcionalnost softvera koja će zadovoljavati korisničke zahteve svo vreme trajanja procesa razvoja. Tada se predlaže izgradnja jednostavnog rešenja koje se može brzo menjati, u skladu sa izmenjenim zahtevima (Poppendieck & Poppendieck, 2003).

Iterativan razvoj softvera pogodan je za donošenja odluka po principu što je kasnije moguće. Kasnijim donošenjem odluke smanjuje se nivo neizvesnosti i ukupni troškovi razvoja. Predstavlja efikasan način za identifikovanje eventualnih problema i njihovo rano rešavanje. Ovakvo odlučivanje omogućava da se pre donošenja konačne odluke razmotre sva moguća rešenja, što je isplativije i utiče na skraćivanje vremena isporuke proizvoda i njegov kvalitet. Odluke koje su bitne na samom početku projekta (npr. izbor jezika koji će se koristiti, arhitekturalne odluke, izbor baze podataka i sl.) trebaju biti zasnovane na breth-first pristupu donošenja odluka, jer njihove izmene u kasnijim fazama razvoja mogu koštati i 100 puta više (Poppendieck & Poppendieck, 2003).

3. Isporučiti što je brže moguće

Brza isporuka proizvoda direktno utiče na povećanje zadovoljstva korisnika, što je jedan od primarnih ciljeva Lean-a. Brzom isporukom troši se manje resursa, tako da korisnici mogu menjati svoje zahteve uz manji rizik. Na primer, ako programeri pišu puno koda bez testerinja ili bez integracije u celovito rešenje, rizik je izvesniji. Njegovo smanjivanje može se obezbediti brzom isporukom proizvoda.

Za brzu isporuku proizvoda Middleton i dr. (2005) predlažu razvoj softvera u malim iteracijama, kroz ceo životni ciklus projekta, od analize zahteva, preko dizajniranja i kodiranja do testiranja rešenja. Opisanim pristupom umanjuje se jaz momenta nastanka greške/problema i momenta njegovog otklanjanja, što implicira manje razvojne troškove i proizvodnju veće vrednosti za korisnika.

Princip brze isporuke proizvoda tesno je povezan sa opštim Lean principom protoka. Naime, nakon definisanja vrednosti nekog procesa, iz perspektive korisnika, a potom i dizajniranja toka izgradnje vrednosti, potrebno je obezbediti protok takav da svaka aktivnost u okviru toka vrednosti teče kontinuirano, dodajući maksimalnu vrednost na svakoj tački toka i isporučujući proizvod (vrednost) bez kašnjenja (Womack & Jones, 2003). Da bi protok izgradnje vrednosti za korisnika težio savršenstvu Middleton i dr. (2005) predlažu da se vodi računa o raspodeli zadataka u timu, tako da su u skladu sa njihovim veštinama i stručnošću.

Poppendieck M. i Poppendieck T. (2003) izdvajaju tri alata koji pomažu ostvarivanje principa brze isporuke: pull sistem, teorija čekanja u redu i obračun troškova kašnjenja. To dalje znači da tim treba da izgrađuje samo one funkcije proizvoda koje su odobrene od strane korisnika i to u onom vremenskom trenutku koji je određen korisnički definisanim prioritetima i preferencijama (pull sistem razvoja), umesto da rukovodilac projekta samostalno odlučuje šta će se razvijati, u kom vremenu i kom trajanju (pushing sistem). Za efikasnu implementaciju pull sistema neophodan uslov je samousmeren i samomotivisan projektni tim, u koji spadaju i ključni stejkholderi projekta. Kanban je Lean praksa koja potpomaže uspešno funkcionisanje pull sistema. Kanban podrazumeva primenu vizuelnih tehnika u cilju praćenje realizacije svake aktivnosti procesa. Drugi efikasni alati za ostvarivanje brze isporuke proizvoda: brz obračun troškova kašnjenja i teorija čekanja u redu, biće detaljno opisani u narednom poglavlju rada.

4. Pojačati učenje

Lean podstiče učenje iz iskustva, deljenje znanja u okviru tima ali i sa stejkholderima. Ovaj princip može se primenjivati u različitim fazama razvoja softvera. Na primer u fazi dizajna, u slučajevima kada je problem dobro definisan, uglavnom se koristi top-down pristup razvoja rešenja. U slučajevima kada problem nije u potpunosti jasan, arhitekta i dizajneri koriste različite prakse kako bi iznašli odgovarajuće rešenje: višekratna analiza različitih scenarija rešenja problema, višekratna analiza zahteva značajnih za dizajn rešenja, segmentacija rešenja visokog nivoa i sl. Upotreba svih ovih praksi na putu iznalaženja rešenja problema, predstavlja zapravo ciklus učenja (Poppendieck & Poppendieck, 2003). U istoj situaciji se mogu naći i analitičari zahteva, programeri, testeri, pa i poslovni ljudi (stejkholderi) koji su deo projektnog tima.

Isporuka funkcionalnog softvera i njegovo testiranje od strane korisnika (stejkholdera) takođe doprinosi rastu znanja u okviru tima. Poppendieck, M. i Poppendieck T. (2003) smatraju da

pojačanom učenju u organizacijama značajno mogu doprineti sledeće četiri prakse: povratne informacije, iteracije, sinhronizacija i razvoj zasnovan na analizi više opcija i pojavi rešenja.

U tradicionalnom razvoju softvera povratne informacije su smatrane pretnjom, jer ugrožavaju unapred definisani plan. U iterativnom razvoju smatraju se neophodnim alatom koji obezbeđuje zajednički rad razvojnog tima i stejkholdera. U svakoj iteraciji razvijaju se funkcije koje je definisao korisnik, prema sopstvenim prioritetima, pri čemu se razvijeni funkcinalni softver mora sinhronizovati nakon svake promene od strane korisnika. Poppendieck M. i Poppendieck T. (2003) predlažu tri načina sinhronizacije funkcija: harmonizacija i stabilizacija, proširivanje aplikacije i matrica.

5. Osnaživanje tima

Tradicionalni razvoj softvera podrazumeva da je projekat pod kontrolom projektnog manadžera. drugim rečima, projektni menedžer je imao ulogu da usmerava ukupan rad na projektu, da donosi odluke o problemima identifikovanim u procesu razvoja, da predlaže rešenja identifikovanih problema, da zapošljava i otpušta zaposlene, da zna cilj projekta, identifikuje i kontroliše greške u radu. Zbog svih navedenih odgovornosti, uspeh projekta se pripisivao upravo projektnom menadžeru.

Lean projektni menadžeri imaju drugačiju ulogu na projektu, deleći odgovornost za uspeh projekta sa svim članovima tima. Lean projektni menadžer ne razvija rešenja za identifikovane probleme u procesu razvoja softvera, već na to podstiče tim. Lean projektni mednadžer obezbeđuje neophodne resurse na projektu, uvažavajući pritom mišljenje članova tima (Miller, 2005).

U cilju osnaživanje tima Lean predlaže sledeće tehnike: samoopredeljenje, motivacija, liderstvo i stručnost. Samoopredeljenje znači da je tim obučen kako da projektuje sopstvene radne procedure, te je uloga menadžmenta u tome izlišna. Tim pred menadžere može izaći i sa inovativnim idejama oko poboljšavanja procesa razvoja i izmena starih procedura rada, pri čemu je uloga menadžmenta da te ideje verifikuje i donose konačnu odluku.
Motivisanost članova tima je drugi neophodan uslov za osnaživanje tima. Motivisanost se može postići kroz obezbeđivanje osećaja pripadnosti, sigurnosti, kompetentnosti i ličnog napretka svakog člana tima. Prisustvo lidera u timu je takođe važan elemenat za osnaživanje tima. Uloga lidera je da usmerava razvoj, vrednuje članove tima na osnovu njihove stručnosti i obezbedi uslove za njihovu veću motivisanost na projektu (Larman & Vodde, 2009; Liker & Morgan, 2006). Efikasnost i snaga nekog tima u direktnoj je vezi sa nivoom stručnosti i znanja svih njegovih članova.

Osnaživanje tima je jedan od osnovnih agilnih principa i zasnovan je na pretpostvaci da neposredni izvršioci nekog posla imaju najveće znanje o tom poslu, kako u pogledu ograničenja tako i u pogledu njegovog unapređenja. Centralizovano odlučivanje u agilnim

procesima razvoja nije moguće sprovesti, jer bi ono onemogućilo princip brze isporuke i odlučivanje što je kasnije moguće (Poppendieck & Poppendieck, 2003). Pored osnaživanja tima bitna je i njegova organizaciona struktura, kako bi protok informacija, odluka i resursa tekao efikasno. Agilni timovi zahtevaju krosfunkcionalnu organizaciju, koja podrazumeva da su članovi na nivou eksperata. Krosfunkcionalna organizacija tima doprinosi povećanju povratnih informacija, brzini korektivnih akcija i smanjenje kašnjenja u procesu razvoja (Ohno, 1988; Karlsson & Ahlstrom, 1996).

6. Fokus na korisnika

Opstanak nekog poslovanja zasnovan je na lojalnosti njegovih korisnika, a poslovni uspeh na zadovoljstvu korisnika. Zadovoljan korisnik je onaj čije su potrebe ispunjene. U današnjim uslovima poslovanja korisnici usluga i proizvoda imaju veliku mogućnost izbora, stoga menadžeri i timovi treba da bude svesni toga i da obezbede rešenje kojim će biti zadovoljene korisničke potrebe i očekivanja. Oblast menadžmenta koja se bavi ovim pitanjima je Lean Six sigma. Six sigma koncept podrazumeva da se članovi tima bave sa nekoliko korisnika istovremeno, vodeći pritom računa da se svaki korisnik osećao posebnim.

Profesor Noriaki Kano 1980. godine je razvio tzv. Kano model za povećanje zadovoljstva korisnika. Kano model razmatra tri tipa korisničkih potreba: osnovne potrebe, performanse i neočekivane potrebe. U grupu osnovnih potreba spadaju one koje je korisnik očekivao od proizvoda, te njihov neuspeh ili nedostatak direktno utiče na nezadovoljstvo korisnika. Osnovne potrebe u razvoju softvera podrazumevaju isporuku proizvoda na vreme, proizvod bez grešaka i proizvod koji efikasno funkcioniše.

Potrebe performansi imaju proporcionalan uticaj na zadovoljstvo korisnika. Na primer ukoliko troškovi razvoja softverskog proizvoda rastu, proporcionalno će opadati zadovoljstvo korisnika proizvoda.

Neočekivane potrebe su one koje korisnici ni ne naslućuju, te se njihovo izostavljanje iz proizvoda ne odražava na zadovoljstvo korisnika, ali njihovo isporučivanje, s druge strane, ima veliki uticaj na rast zadovoljstva korisnika.

Na zadovoljstvo korisnika utiče i način na koji razvojni tim komunicira sa njima. Highsmith (2002) smatra da je najefikasniji oblik komunikacija licem u lice i da ona predstavlja temelj uspešnog razvoja softverskog rešenja.

7. Izvrsnost proizvoda

Princip izvrsnosti se odnosi na proizvodnju kvalitetnih softverskih rešenja, koja zadovoljavaju tržišnu tražnju. Različiti autori predlažu različite principe za ostvarivanje izvrsnosti proizvoda. Poppendieck M. i Poppendieck T. (2003) predlažu da se "gradi integritet proizvoda", pri čemu integritet opisuju kroz dva koncepta: uočljivi i konceptualni integritet.

Uočljivi integritet proizvoda se postiže uspostavljanjem balansa između funkcionalnosti, korisnosti, pouzdanosti i ekonomskih aspekata proizvoda, koji sinergetski doprinose zadovoljstvu korisnika. Konceptualni integritet znači da centralni koncepti sistema zajedno rade skladno kao kohezivna celina.

Relevantni stejkholderi važan su elemenat oba koncepta. Za realizaciju uočljivog integriteta softverskog proizvoda bitna je aktivna uloga eksperata domena, korisnika, kupaca i vlasnika proizvoda, a za realizaciju konceptualnog integriteta bitna je posvećenost programera, analitičara, dizajnera i testera. Uočljivi integritet obezbeđuje protok informacija od kupca/korisnika do tima, a konceptualni obezbeđuje protok tehničkih informacija između članova tima.

Izgradnja integriteta proizvoda znači u suštini graditi ono što ima vrednost za njegovog korisnika. Pored toga, integritet podrazumeva da proizvod tokom vremena održava svoju korisnost i kvalitet, u čemu značajnu ulogu ima koherentna arhitektura, koja omogućuje održavanje, prilagođavanje i iterativnost (Poppendieck & Poppendieck, 2003).

Middleton (2001) smatra da se ostvarivanje izvrsnosti kompleksnih softverskih sistema može olakšati izradom modela (npr. konceptualni model domena, dijagrami slučajeva upotrebe i dr.), tehnikom refaktorisanja, vizuelnim i automatskim merenjem kvaliteta i testiranjem softverskog rešenja od strane korisnika.

Ohno (1988) za vizualizaciju informacija preporučuje upotrebu Kanban table, čime se obezbeđuje transparentnost, lakša komunikaciju i aktivno učešće svih članova tima u ostvarivanju izvrsnosti softverskog proizvoda. Ohno podstiče i automatizaciju tokom razvoja proizvoda, smatrajući da ona može obezbediti kontrolu kvaliteta (smanjenjem otpada iz procesa) i sprečiti proizvodnju proizvoda sa nedostacima. Slikoviti primer svrsishodnosti automatizacije jeste otkrivanje grešaka tokom testiranja koda

Larman i Vodde (2009) ističu povezanost tehnologije i proizvodnju kvalitetnog softverskog proizvoda. Predlažu da izbor tehnologije bude u skladu sa potrebama kompanije i potrebama datog projekta, kao i da je zadatak menadžera projekta da pomogne timu da se prilagodi odabranoj tehnologiji.

8. Kontinuirano poboljšanje

Prema Larman-u dva su stuba Lean filozofije: poštovanje ljudi i kontinuirano poboljšanje. Autor smatra da je za ostvarivanje kontinuiranog poboljšanja važno "kreiranje atmosfere kontinuiranog učenja i okruženja koje rado prihvata promene".

Uspeh kontinuiranog poboljšanja Larman i Vodde (2009) procenjuju kroz četiri elementa:

- Vidi sam - u smislu da i stejkholderi i članovi razvojnog tima umesto da sede u svojim kancelarijama i primaju informacije o procesu, treba da idu na mesto dešavanja, da bi bolje razumeli proces.
- Kaizen - praksa koja podrazumeva stalnu težnju ka savršenstvu.
- Izazov savršenstva - imati visoka očekivanja i uključiti sve stejkholdere u rešavanju izazova.
- Raditi u pravcu protoka (definisanog toka vrednosti) - jer će se takvim radom otkrivati slabosti i otpad što je prilika za poboljšavanje procesa.

Liker i Morgan (2006) su listu dopunili sa još jednim elementom: standardizacijom. Autori smatraju da standardizacija pomaže u smanjivanju varijacija, eliminiše otpad i omogućava razvoj visoko kvalitetnog proizvoda. Kada različiti ljudi obavljaju isti posao u različitom vremenu, kvalitet proizvoda varira od osobe do osobe, ukoliko ne postoje standardizovane procedure.

Calderone (2010) predlaže različite načine povećanja standardizacije:

- Razjašnjavanje uloga u procesu, kako bi svi zaposleni znali šta je njihov domen rada.
- Utvrđivanje standarda performansi, kako bi se merio učinak zaposlenih.
- Vizualizacija procesa, kako bi zaposleni razumeli njegov tok realizacije i posledice u slučaju neadekvatnog izvršavanja njihovih zadataka.
- Izrada kontrolnih listi, kako bi se proveravala realizacija ključnih koraka procesa.

Kaizen praksa je u direktnoj vezi sa principom kontinuiranog poboljšanja, jer podrazumeva da tim stalno iznalazi bolje tehnike i prakse za svoj rad. Kaizen podrazumeva i zaustavljenje procesa razvoja u slučaju da član tima identifikuje problem, te da se pristupi njegovom kolektivnom rešavanju, iznalaženjem samog uzroka problema. Stalno poboljšavanje omogućuje ostvarivanje Lean principa - sustizanje savršenstva.

***

Opisani principi Lean filozofije zauzimaju strateški nivo u organizaciji koja ih implementira, te je ove principe strateškog nivoa potrebno podržati alatima i metodama na operativnom nivou (Hines, Holweg, Rich, 2004). U tom smislu danas je aktuelan trend kombinovanja Lean principa i drugih agilnih procesa razvoja softvera. Lean se prema nekim klasifikacijama procesa razvoja (Sommerwille, 2010) svrstava u grupu agilnih metodologija. Pored sličnosti agilnih i Lean principa nekolicina autora je istraživala njihove razlike. Razlike koje su identifikovali Coplien i Bjornvig (2010) prikazane su u tabeli 3.14.

Tabela 3.14 Razlike agilne i Lean filozofije

LEAN	AGILE
Razmisli i uradi	Uradi-proveri-prilagodi
Planiraj i reaguj na povratne informacije	Povratne informacije - reaguj na promene
Visok protok	Malo kašnjenje
Fokus na procese	Fokus na ljude
Tim radi kao jedan	Pojedinci i interakcije
Komplikovani sistemi	Kompleksni sistemi
Prihvatanje standarda	Pregledati i prilagoditi
Ponovni rad u fazi dizajna dodaje vrednost - nije otpad	Ponovni rad na kodu obezbeđuje kvalitet
Neke odluke se donose unapred	Odloži donošenje svih odluka do poslednjeg odgovornog momenta

Izvor: (Coplien, J.O., Bjornvig, 2010)

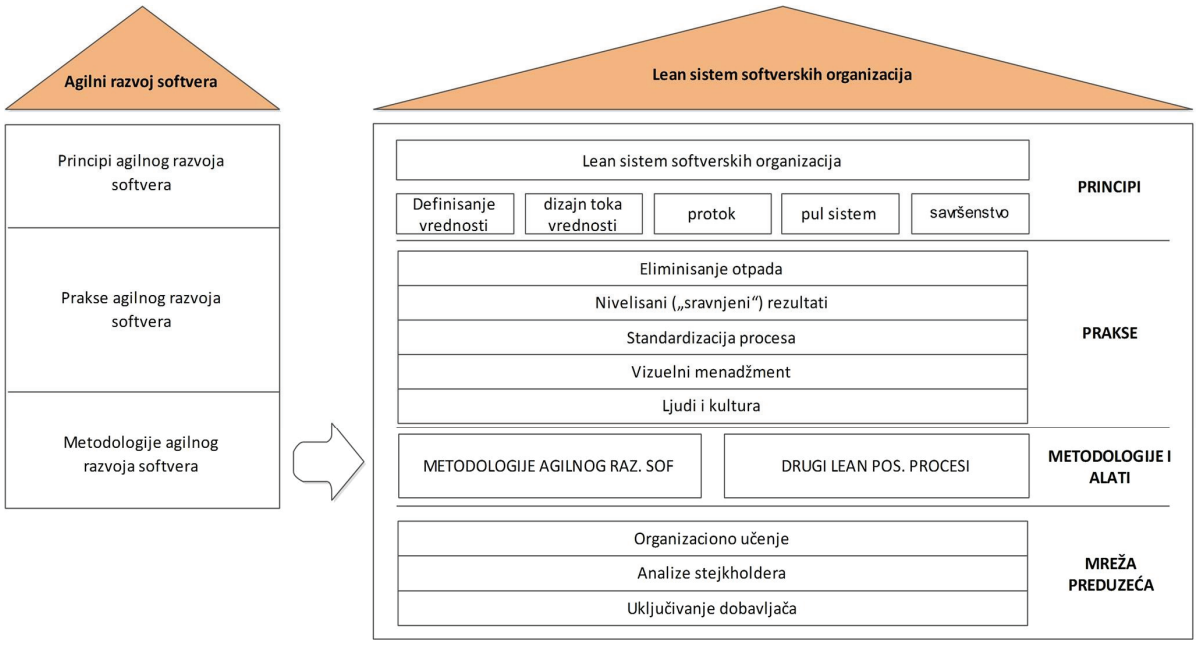
Leffingwell (2007) je pored opšte poznatih principa Agilnog Manifesta, utvrdio i šest paradigmi, zajedničkih za sve agilne procese. U tabeli 3.15 prikazan je komparativni pregled agilnih paradigmi i Lean vrednosti:

Tabela 3.15 Komparacija agilnih paradigmi i Lean vrednosti

PARADIGMA	AGILAN RAZVOJ	LEAN VREDNOSTI
merenje uspeha	odgovor na promene, promenljivost koda	vrednost za kupca
menadžment kultura	rukovodstvo/kolaborativno	osnaživanje tima/liderstvo
zahtevi i dizajn	kontinuirani/pojavni/na vreme	na vreme
kodiranje i implementacija	kod i jedinice testa, serijska isporuka	protok jednog komada
obezbeđivanje i testiranje kvaliteta	kontinuirano/konkurentno/rano testiranje	dizajniranje kvaliteta
planiranje i raspoređivanje	dva nivoa plana/fiksni datumi, procena obima	sravljen protok

Izvor: (Leffingwell, 2007)

Slikom 3.18 prikazan je konceptualni pogled na način i mesto integrisanja principa i praksi agilnih procesa razvoja u Lean sistem softverskih organizacija:



Slika 3.18 Lean sistem softverskih organizacija i agilan razvoj softvera
Izvor: (Kaikkonen, 2011)

3.3.4.2 Lean prakse u razvoju softvera

Lean karakterišu brojne prakse koje su korišćene u proizvodnim kompanijama. U nastavku će biti opisane prakse koje se najčešće primenjuju u domenu razvoja softvera (tabela 3.16):

1. Tačno na vreme (Just In Time)

Praksa JIT podrazumeva proizvodnju potrebnih funkcionalnosti softvera, u potrebnom obimu i tačno određenom vremenu (Yavuz, 2010). Koncept JIT podrazumeva i da se angažovanje određenih resursa na projektu vrši u momentu kada se ukaže realna potreba za tim. Naime, generišući tražnju za nekim karakteristikama proizvoda, korisnici ujedno generišu i tražnju za resursima neophodnim za njihovu implementaciju. Ovakav način planiranja resursa zahteva jaku i dugoročnu saradnju sa stejkholderima, a za rezultat ima minimiziranje svih vrsta zaliha. Praksa ima za cilj da optimizuje upotrebu resursa i pospeši efektivnost i efikasnost procesa razvoja.

Praksa JIT tesno je povezana sa drugim Lean praksama i tehnikama, koje će biti opisane narednim tekstom.

Kanban tehnika primenjivana je po prvi put u okviru Toyote, još 1953. godine, u cilju kontrole i veće efikasnosti procesa proizvodnje. Njena prvobitna primena u proizvodnji i transportu, ubrzo je proširena i na nivou celog lanca vrednosti. Kanban na japanskom znači ploča, koja nema unapred zadatu formu. Na takvoj praznoj ploči se crta, vizualizuje dati proizvodni proces, kako bi svi zaposleni mogli da prate sve događaje i aktivnosti u okviru njega.

Opisana kanban praksa je primenljiva i u domenu razvoja softvera. Programeri razvoj određenih funkcionalnosti softvera otpočinju kada dobiju "signal" da su one potrebne korisniku, čime se eliminiše jedan od otpada poznat kao hiperprodukcija. Hiperprodukcija je bila karakteristična za tradicionalan razvoj softvera, jer se proizvod razvijao u skladu sa raspoloživim ljudskim, materijalnim i tehnološkim resursima, bez obzira na realnu tražnju korisnika (Womack & Jones, 2003; Ohno, 1988).

Pored kanban prakse, u razvoju softvera primenljiva je i heijunka praksa. Praksa se koristi u svrhu planiranja i terminiranja razvojnih aktivnosti, u cilju skraćenja ciklusa od faze identifikovanja zahteva do isporuke proizvoda. Heijunka, za razliku od tradicionalnog razvoja, preferira proizvodnju softvera u malim serijama, iteracijama. Heijunka na ovaj način postiže da varijacije u proizvodima i procesima budu u što većoj meri kontrolisane i eliminisane (Liker & Meier, 2006; Womack & Jones, 2003).

Praksa vizuelnog menadžmenta, takođe, nalazi svoju primenu u domenu razvoja softvera. Ona podrazumeva "učiniti sve vidljivim" u toku procesa razvoja softvera. Alati koji se u tu svrhu mogu koristi su sledeći: kanban table, andon table, poka-yoke, 5S. Vizuelni menadžment ima za cilj da omogući lakšu komunikaciju, identifikovanje problema, upravljanje zalihama i JIT razvoj.

Andon table su kontrolni uređaji kojima se obezbeđuje brz način protoka informacija u celom proizvodnom, ali i razvojnom okruženju. Primeri ovih uređaja su „semafori", koji daju vizuelan znak da li je proces ili neki resurs u zastoju.

Poka-yoke je tehnika koja podrazumeva proveru grešaka uz pomoć pratećih audio ili video signala. Na primer, funkcionalnosti softvera mogu biti vizuelno označene karticama u boji, kako bi lakše mogle biti uočene nekompletnosti tokom implementacije. Poka yoke je sredstvo kojim se obezbeđuje izgradnja kvalitetnog proizvoda, jer se kvalitet ugrađuje u proizvod na vreme, a ne nakon sprovođenja kontrole.

Poka yoke tehnika može se primenjivati i u standardizaciji rada, ilustrovanjem radnog mesta na standardizovanom listu papira, koji će predstavljati uputstvo za realizaciju zadataka. Na ovaj način se obezbeđuje vizuelna komunikacija u timu. Standarde dizajniraju sami izvršioci zadataka, jer se polazi od pretpostavke da oni najbolje znaju posao, te i najlakše mogu definisati efikasne načine rada. Definisane procedure mogu se kontinuirano poboljšavati, ako se identifikuje potreba za tim (Liker & Meier, 2006).

2. Kaizen

Kaizen praksa je potekla iz Japana, a danas predstavlja svetski pokret, prepoznatljiv po sledećem cilju: eliminisanje svih oblika rasipanja i štedljivo korišćenje resursa u

proizvodnji/razvoju proizvoda i usluga. Tvorac ove prakse je Maasaki Imai, osnivač Kaizen instituta (1986. godine).

Kaizen se definiše kao poslovna i menadžment filozofija, koja se zasniva na inkrementalnom, neprekidnom i sveobuhvatnom unapređenju poslovnih procesa, načina i organizacije rada, kvaliteta proizvoda i usluga. Kaizen praksa polazi od stava da ni jedan proces nije dovoljno dobar, kao i da ni jedna inovativna ideja, u cilju njegovog unapređenja, nije previše mala.

Kaizen rešenja nisu finansijski zahtevna niti rizična, jer su fokusirana na male korake koji impliciraju velika poboljšanja na nivou kompanije (Barraza, Smith, Dahlgaard- Park, 2009; Wanga, Conboyb, Cawleyc, 2012).

Kaizen praksa zasniva se na sledeća tri ključna elementa (Imai, 2008):

1. Shojinka - zaposleni su glavni nosioci uvođenja promena i unapređenja procesa.
2. Soikufu - kreativno razmišljanje i plasiranje inovativnih ideja od strane zaposlenih mora biti podsticano u organizaciji.
3. Jidoka - podrazumeva stalnu kontrolu kvaliteta i identifikovanje grešaka u proizvodima, koje nastaju tokom realizacije aktivnosti u okviru procesa. Podržava samostalnost zaposlenih, u smislu da svako može da zaustavi proizvodni/razvojni proces ukoliko uoči defekte. Podstiče delegiranje kontrole na automatizovane uređaje, koje signaliziraju detektovane greške, a zaposleni ih potom samo saniraju. Ovim je omogućeno da zaposleni nesmetano rade, a kontrolu procesa vrše jedino u slučajevima kada dobiju signal od uređaja da je nastao problem.

Kaizen predviđa i pet principa za pronalaženje i otklanjanje uzroka identifikovanog problema u procesu. Prvi princip nalaže da se nakon identifikovanja problema, treba otići na mesto na kojem se proces odvija, jer je tamo nastao sam problem. U drugom koraku potrebno je proveriti opremu, alat i ceo inventar koji se koristi u datom procesu. Ako se tada ne pronađe uzrok nastalom problemu, potrebno je odgovoriti na pitanje "zašto" i to ne jednom, nego obavezno pet puta. Treći princip podrazumeva sprovođenje kontra mera, a četvrti otklanjanje uzroka problema. Peti princip predviđa definisanje standarda da bi se sprečilo pojavljivanje istog problema u budućnosti (Imai, 2008).

3. Teorija čekanja

Teorija čekanja predstavlja matematički metod, koji se koristi za izračunavanje vremena kašnjenja, odnosno čekanja u procesu razvoja softvera. Teorija doprinosi celishodnijem rešavanju problema u procesu razvoja i bržoj isporuci softverskog proizvoda. Fundamentalna merenja u teoriji čekanja vezuju se za pojam "ciklus vremena". Ciklus vremena predstavlja ukupno vreme koje je potrebno za prolazak kroz neki proces od tačke A do tačke B, npr. ciklus vremena od identifikovanja korisničkih zahteva do isporuke funkcionalnog softverskog

rešanja. U navedenom primeru ukupno vreme se izračunava sabiranjem vremena koje tim provede u radu sa korisnicima oko definisanja korisničkih priča, vremena za utvrđivanje prioriteta zahteva, vremena provedenog u kodiranju, testiranju i verifikaciji izrađenih funkcionalnosti. Skraćenjem ciklusa vremena isporuka softverskog proizvoda postaje brža (Poppendieck & Poppendieck, 2003).

4. Liderstvo

Liderstvo predstavlja jednu od ključnih Lean praksi (Larman & Vodde, 2009; Liker & Morgan, 2006). Uloge menadžera i lidera se suštinski razlikuju po njihovim zadacima, nadležnostima i odgovornosti. Zadaci menadžera su planiranje i budžetiranje, organizovanje procesa i ljudi, praćenje i kontrola, a lidera: postavljanje pravaca, projektovanje organizacije i kulture posvećenosti izvrsnosti (Poppendieck & Poppendieck, 2003).

Liker i Morgan (2006) ističu da je lider visoko stručna i harizmatična osoba iz redova tima, koja nema formalni autoritet, a odgovorna je za kretanje proizvoda od početka do kraja nekog procesa, prateći da li je sve urađeno na pravi način.

Lider je osoba koja svojim ponašanjem usmerava tim ka zajedničkom cilju, ne koristeći sredstva prinude, već se oslanja na: komunikaciju, motivaciju, organizacionu kulturu, ohrabrivanje saradnje, timskog duha i izgradnju poverenje među članovima tima. Lider je osoba koja ima viziju i sposobnost da ostale članove tima motiviše da je ostvare. Podstiče snažne međuljudske odnose u timu, kreira uslove dobre komunikacije u timu i teži održavanju uspostavljenih dobrih odnosa. Dok su menadžeri okrenuti ostvarivanju efikasnosti u radu, lideri u fokusu imaju efektivnost rada. Menadžeri zaposlenima određuju zadatke, a lideri način na koji da ih realizuju (Northouse, 2015).

5. Obračun troškova odlaganja

Praksa podrazumeva obračun troškova i dobiti pre donošenja neke odluke. Recimo ako programeri zahtevaju od menadžera kupovinu nekog alata, kako bi povećali svoju efikasnost rada, neophodno je da menadžer obračuna troškove i dobit od kupovine novog alata (Poppendieck & Poppendieck, 2003).

Ovakav obračun troškova primenljiv je i prilikom planiranja arhitekture, na uštrb razvoja funkcionalnosti. Svako odlaganje implementacije iziskuje troškove zbog odložene isporuke vrednosti stejkholderima, te ovi troškovi usled odlaganja trebaju biti manji od troškova koje bi indukovao redizajn arhitekture u kasnijim razvojnim fazama.

6. Identifikovanje toka vrednosti

Praksa identifikovanje toka vrednosti u okviru procesa razvoja softvera, suštinski je identična sa istoimenim opštim Lean principom. Podrazumeva identifikovanje "realnog" toka vrednosti

u procesu razvoja softvera, kao i izradu "idealnog" toka. Idealan tok vrednosti je mapa koja prikazuje samo one aktivnosti koje proizvode vrednost za korisnika, pri čemu je glavni cilj ostvarivanje maksimalne efektivnosti i efikasnosti u postizanju zadovoljstva korisnika. Kao alat za ovu praksu može biti korišten samo papir i olovka, pri čemu menadžeri crtaju dijagram koji pokazuje sve neophodne korake procesa, beležeći i njihovo vreme izvršavanja. Cilj vizualizacije i analize toka aktivnosti u procesu je identifikovanje nepotrebnog vremena čekanja u pojedinim aktivnostima, što je osnov za eliminisanje otpada iz toka vrednosti datog procesa (Poppendieck & Poppendieck, 2003).

Tabela 3.16 Lean prakse po autorima

LEAN PRAKSE	IZVORI
Kanban	Ohno (1988); Polk (2011)
Kaizen	Ohno (1988); Barraza, Smith i Dahlgaard-Park (2009); Holden (2010); Wanga, Conboyb i Cawleyc (2012)
Jidoka	Bruuna i Meffordb (2003)
Odloženo odlučivanje	Poppendieck i Poppendieck (2003)
Kano analiza	Miskelly (2009)
Učiniti sve vidljivim	Womack i Jones (2003); Middleton (2001)
Teorija čekanja	Propendieck i Propendieck (2003)
Obračun troškova kašnjenja	Poppendieck i Poppendieck (2003)
Identifikovanje toka vrednosti	Womack i Jones (2003); Poppendieck i Poppendieck (2003)
Heijunka	Middleton i dr. (2005); Wanga, Conboyb i Cawleyc (2012)

3.3.5 Softverska arhitektura i agilni procesi razvoja

Narednim poglavljem biće prikazani rezultati teoretskog istraživanja, dobijeni nakon sprovođenja sistematskog pregleda literature, prema metodologiji koju je razvila Kitchenham (2004). Okvir po kojem je sprovedeno dato straživanje, definisan je u drugom poglavlju doktorske disertacije, zajedno sa ostalim metodološkim aspektima rada.

U tabeli 3.17 dat je konceptualni prikaz rezultata istraživanja, kroz prizmu kategorisanih arhitekturalnih pitanja/problema identifikovanih u literaturi, prema autorima koji su ih izučavali. Data pitanja i problemi razmatraće se i interpretirati narednim tekstom, čime se ujedno daje odgovor na postavljeno istraživačko pitanje:

IP1. Šta pokazuju rezultati empirijskih istraživanja, u pogledu integracije agilnih i arhitekturalnih praksi?

Tvrdnja pristalica agilnog razvoja, da su eksplicitne arhitekturalne aktivnosti nepotrebne u agilnim procesima razvoja, predmet je istraživanja većine radova, do kojih se došlo sistematkom analizom literature.

Proučena literatura ukazuje da je nascentna arhitektura dobra alternativa konvencionalnim pristupima za razvoj softverske arhitekture, ali samo u nekim arhitekturalnim oblastima, dok druge uopšte ne pokriva. Friedrichsen (2014) navodi da je nascentna arhitektura dobra praksa za detaljan dizajn, ali ne pokriva set važnih arhitekturalnih aktivnosti, koje se odnose na pitanje da li rešenje radi pravu stvar. Ove aktivnosti se sastoje od diskusija arhitekti sa stejkholderima, u nameri da se razumeju njihove potrebe, identifikuju zahtevi i reše njihove kontradiktornosti i konflikti. Isti autor navodi i da implementacija ovih aktivnosti zahteva veštine, iskustvo i mnogo šire znanje od onog koje imaju programeri. Eliminisanje ove eksplicitne arhitekturalne aktivnosti ima za posledicu da pogrešne odluke dizajna mogu biti kasno vidljive, kao i da interesi programera budu prenaglašeni u odnosu na potrebe drugih stejkholdera sistema (Friedrichsen, 2014).

Nalazi u literaturi pokazuju i da se u praksi ne posvećuje u dovoljnoj meri pažnja nefunkcionalnim zahtevima sistema u samom procesu razvoja, uz obrazloženje da se njihovo ostvarivanje realizuje tokom faze održavanja sistema, jer ona ima duže trajanje i veći budžet. Realizacija nefunkcionalnih zahteva vrši se isključivo tehnikom refaktorisanja programskog koda (Babar, 2009).

Međutim, Bellomo, Nord, i Ozkaya (2013) smatraju da se konvencionalan agilan proces, naročito u slučaju razvoja velikih i kompleksnih sistema, treba proširiti sledećim eksplicitnim arhitekturalnim aktivnostima koje se bave nefunkcionalnim zahtevima: TDD sa fokusom na nefunkcionalne zahteve, izrada prototipova sa fokusom na nefunkcionalne zahteve i monitoring tehničkog duga sa fokusom na nefunkcionalne zahteve.

Huang, Czauderna i Mirakhorli (2014) uvode pojam arhitekturalno-pametne persone koja vrši funkciju identifikovanja i analiziranja nefunkcionalnih zahteva. Navedeni autori su imali cilj da upotrebom koncepta persona iz različitih domena, poboljšaju proces otkrivanja, analize i upravljanja arhitekturalno značajnim zahtevima.

Jeon, Han, Lee, E., Lee K. (2011) predstavljaju metod Acrum, koji Scrum proces proširuje sa tri eksplicitne arhitekturalne aktivnosti, koje u fokusu imaju nefunkcionalne zahteve: analiza i upravljanje nefunkcionalnim zahtevima sistema (neposredno nakon analize funkcionalnih zahteva); mapiranje funkcionalnih i nefunkcionalnih zahteva (putem matrice njihovih međusobnih odnosa), u cilju generisanja liste poslovnih zadataka proizvoda i liste zadataka sprinta; verifikacija ispunjenosti/neispunjenosti nefunkcionalnih zahteva nakon svakog sprinta. U slučaju da nakon procesa verifikacije, identifikovani nefunkcionalni zahtev nije ispunjen i pored implementacije svih funkcionalnosti sa kojima je u vezi, onda funkcije moraju biti implementirane iz početka ili se mora formulisati nova strategija za postizanje datog nefunkcionalnog zahteva.

Brown, Nord, Ozkaya (2010) takođe naglašavaju neophodnost eksplicitnog identifikovanja nefunkcionalnih zahteva sistema, kako bi se za svaku iteraciju mogle utvrditi zavisnosti funkcionalnih i nefunkcionalnih zahteva sistema. I funkcionalni i nefunkcionalni zahtevi

trebaju biti prioritizovani na jedinstvenoj listi, da bi se definisao adekvatan plan rada tokom release-a. A Faber (2010) dodaje da je odgovornost arhitekte da nefunkcionalne zahteve sistema korisnicima isporuči kao vrednosti i da ih implementira u tesnoj saradnji sa programerima.

Definisanje arhitekturalnih struktura za glavni deo sistema, je takođe eksplicitna arhitekturalna aktivnost, koja ne proizilazi iz nascentne arhitekture (Friedrichsen, 2014), kao ni aktivnost anticipiranja budućih potreba sistema. Anticipiranje budućih potreba značajno je kako bi se donela adekvatna odluka da li određenu arhitekturalnu aktivnost preduzeti danas, usled estimacije većih koristi i manjih troškova u budućnosti (Brown et al., 2010). Planiranje arhitekture podrazumeva njeno razmatranje izvan okvira tekuće iteracije (ili sprinta kada je u pitanju Scrum proces), kako bi se sagledali budući zahtevi, koje arhitekturalno rešenje treba da podrži (Friedrichsen, 2014; Isotta-Riches & Randell, 2014; Brown et al., 2010).

Smeštanje svih aktivnosti dizajna u okviru tekuće iteracije predstavlja strategiju ekstremno visokog rizika, a naročito na projektima razvoja softvera u velikim poslovnim organizacijama, koje karakteriše veliki broj različitih aplikacija (starih-nasleđenih i novih-modernih), različite tehnologije i veliki broj timova (Isotta-Riches & Randell, 2014). Planiranje isključivo u okviru jedne iteracije (ili sprinta) doprinosi da se dizajn degeneriše i da nema fleksibilnosti, što ugrožava agilnost celog projekta (Weitzel, Rost, & Scheffe, 2014). Ključan razlog za to je što se funkcionalni zahtevi ne mogu razmatrati i razvijati potpuno nezavisno, jer među njima postoje zavisnosti (Brown et al., 2010). Čest je slučaj da funkcionalni zahtevi koji za korisnika imaju visoku poslovnu vrednost, stoga i visok prioritet realizacije, zavise od zahteva sa nižom poslovnom vrednošću, koje je iz tog razloga neophodno ranije implementirati. Pored analize međusobne zavisnosti funkcionalnih zahteva, neophodno je sprovoditi i analizu zavisnosti funkcionalnosti u odnosu na nefunkcionalne zahteve i arhitekturalne elemente sistema. U suprotnom, povećava se rizik da su implementirane odluke dizajna sistema suboptimalne, što će implicirati rast tehničkog duga u budućnosti. Nagomilavanjem tehničkog duga, tokom vremena, dovodi do pojave problema koji ne mogu biti rešeni samo izmenama programskog koda, već zahtevaju radikalnije izmene arhitekture (Nord, Ozkaya, & Sangwan, 2012; Brown et al., 2010). Predloženi koncept proširuje planiranje release-a, tako što pored izbora funkcionalnosti koje će biti implementirane, razvojni tim identifikuje i arhitekturalne elemente koji takođe moraju biti razvijeni, da bi podržali funkcionalnosti (Bellomo, Ozkaya, & Nord, 2013; Boehm, Lane, Koolmanojwong, Turner, 2010).

Razmatranje zavisnosti između funkcionalnih zahteva i arhitekturalnih elemenata potrebno je sagledati ne samo za tekući release već i za buduće release-ove. Ovakvim pristupom identifikuju se arhitekturalni elementi koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera (npr. broj početnih aplikacija koji se raspoređuju se očekuje da bude mali, pa nefunkcionalni zahtev, kao što je sklabilnost, bi mogao biti odložen za naredni release, ali ulaganjem u

skalabilnost u toku tekućeg release-a može se smanjiti tehnički dug u budućnosti) (Brown et al., 2010).

Weitzel, Rost i Scheffe (2014) širenje horizonta planiranja arhitekture izvan jednog sprinta, u Scrum procesu, nazivaju epik arhitektura. Epik arhitektura predstavlja dizajn arhitekture za koherentnu grupu funkcionalnih zahteva (korisničkih priča u XP procesu). Cilj epik arhitekture je definisanje zajedničkih elemenata i postavlja se za oko 8 planiranih sprintova. Izdvajanjem i implementacijom njihovih sličnosti u tekućem sprintu, smanjuje se ukupan napor implementacije i povećava se uniformnost funkcionalnosti koje će biti implementirane narednim sprintovima. Implementacioni zadaci sprinta izvode se na osnovu definisanih arhitekturalnih zahteva koji čine arhitekturalne priče.

Isotta-Riches i Randell (2014) smatraju da danas, kada je cilj postići što veću agilnost organizacije, agilna arhitektura treba da bude fokusirana ne samo na nefunkcionalne zahteve, već i na zahteve promena tj. da ima mogućnost da se tokom životnog ciklusa projekta promeni, uz niske troškove i mali rizik. Da bi ovi zahtevi bili identifikovani, neophodno je razmatranje arhitekture izvan tekuće iteracije (ili sprinta). Pérez, Díaz, Garbajosa i Yagüe (2014) smatraju da je, u razvoju velikih softverskih sistema, neophodno implementirati kontinuirani proces praćenja promena zahteva (putem izmena ili dodavanja novih funkcionalnosti sistemu) i njihovih efekata na arhitekturu. Uvode koncept radne arhitekture, čiji su centralni elementi promenljive i nekompletne komponente, koje evoluiraju zajedno sa implementacijom proizvoda. Predloženi pristup teži uspostavljanju i održavanju sledljivosti između funkcionalnosti i njihove implementacije u radnoj arhitekturi. Cilj je da se promenama funkcija, automatski detektuju delovi radne arhitekture na koje se promene odnose. Iterativnim procesom, komponente postaju kompletne, izgrađujući tako arhitekturu finalnog proizvoda.

Pošto je teško predvideti i anticipirati buduće potrebe sistema, empirijski nalazi u literaturi pokazuju da je u praksi česta jedna od tri situacije: a) arhitekte se boje da donesu odluke da ne bi propustili nešto što će u budućnosti biti važno, pa s toga odlažu odluku; b) definišu ekstremno generičke i fleksibilne arhitekture kojima se struktura drži otvorenom i nespecifičnom, što je duže moguće, kako bi se mogla prilagoditi širokom spektru budućih promenljivih zahteva; c) sistem se dizajnira tako da podrži poznate uslove, a promenljivi zahtevi u budućnosti se ignorišu (Friedrichsen, 2014).

Svaka od ovih identifikovanih situacija u praksi je suboptimalna: prva odlaže implementaciju, druga povećava kompleksnost sistema i nerazumljivost sistema, a razumljivost je preduslov za njegovu promenljivost. Tako da previše fleksibilnosti dovodi u pitanje razumljivost sistema, a time i promenljivost. Zato je bitno naći ravnotežu između razumljivosti i fleksibilnosti. Ignorisanjem, gubi se uvid u to koje tačke arhitekture trebaju biti fleksibilnije u cilju budućih promena i odluka (Friedrichsen, 2014).

Najveći, i još uvek nerešen izazov u postojećoj literaturi predstavlja iznalaženje načina za uspostavljanje ravnoteže između obima planiranja arhitekturalnih elemenata i nascentne arhitekture, koja nastaje iz programskog koda. Reč je ravnoteži taktičkog nivoa razvoja softvera - koji daje brze vidljive vrednosti kroz razvoj funkcionalnosti i strategijskog nivoa razvoja softvera, koji proizvodi dugoročne vrednosti kroz razvoj softverske arhitekture. Analizom radova koji se bave ovim pitanjem, može se zaključiti da postoji jedinstveni stav, da up front dizajn mora biti dovoljan, što znači da se ne sme zapasti u zamku BDUF strategije, koja karakteriše tradicionalan razvoj. Narednim tekstom biće prikazani nalazi u literaturi koji se odnose na ovo pitanje.

Friedrichsen (2014) ističe da procena dovoljnog obima up front analize i dizajna zavisi od iskustva, veština, znanja i dobre komunikacije softver arhitekte sa stejkholderima, dok su Waterman, Noble i Allan (2012) ovim faktorima dodali i poznavanje odabranog arhitekturalnog rešenja i korišćenje predefinisane arhitekture (u smislu postojećih šablona, arhitekture preporučene od prodavca ili alata koji automatizuju proces). Isotta-Riches i Randell (2014) pored iskustva i znanja arhitekte, naglašavaju značaj i stepena neizvesnosti zahteva i rizika. Smatraju da je ključni cilj da oni budu na zadovoljavajućem nivou, ali i da troškovi usled odlaganja implementacije, budu manji od troškova redizajna arhitekture. U tom smislu, analiza koju predlažu fokusirana je sledeća tri elementa (Isotta-Riches & Randell, 2014):

- Sagledavanje kako neizvesnost zahteva utiče na arhitekturu. Na početku projekta mogu postojati značajne neizvesnosti u zahtevima, ali one same po sebi ne moraju biti značajne za arhitekturu. Cilj je utvrditi koje neizvesnosti treba razmotriti, da postavljena arhitektura na početku, ne bi doživela fundametalne izmene u kasnijim iteracijama. Ukoliko je ovakvih neizvesnosti malo, biće potrebno i malo up front analize i dizajna arhitekture.
- Utvrđivanje broja značajnih neizvesnosti. Ukoliko je ovaj broj velik, potrebno je više vremena izdvojiti na početku projekta, kako bi se razumeo njihov uticaj na arhitekturu i ublažio rizik.
- Identifikovanje nivoa napora i troškova za rešavanje neizvesnosti i nejasnoća u zahtevima. Drugim rečima, kompleksnost zahteva utiče na količinu up front analize i dizajna arhitekture. Neophodno je uporediti relativne troškove koji potiču od odlaganja implementacije (i kasnije isporuke vrednosti korisniku), sa troškovima usled eventualne dorade ili redizajna arhitekturalnog rešenja u fazi implementacije. Troškovi izmene arhitekturalnog rešenja u fazi implementacije trebaju biti viši od troškova odlaganja implementacije, da bi obimnija up front analiza arhitekture bila opravdana.

Brown, Nord i Ozkaya (2010) takođe smatraju da planiranje arhitekture ne sme biti preveliko, već "dovoljno", predlažući u tu svrhu pristup zasnovan na tri koncepta: analiza zavisnosti na nivou arhitekture, analiza prave opcije i upravljanje tehničkim dugom. Analiza zavisnosti podrazumeva sagledavanje i upravljanje, kako zavisnostima koje postoje između funkcionalnih zahteva, tako i zavisnostima između funkcionalnih i nefunkcionalnih zahteva,

odnosno arhitekturalnih elemenata. Cilj je da se na vreme razviju arhitekturalni elementi, koji mogu da podrže implementaciju potrebnih funkcionalnosti. To zahteva planiranje arhitekture preko jedne iteracije tj. anticipiranje i analizu budućih potreba.

Analiza obe vrste zavisnosti predstavlja se jednom tabelom, te se Dependency Structure Matrix analizom (u nastavku DSM) razmatra međusobna zavisnost svih konstitutivnih podsistema, koji predstavljaju određene funkcionalnosti (npr. način razmene informacija između podsistema prodaje i naručivanja), a Domain Mapping Matrices analizom (u nastavku DMMs) njihova zavisnost od pojedinih arhitekturalnih elemenata (npr. od komponenti korisničkog interfejsa, logike pristupa podacima, bezbednosti i sl.).

Značaj procesa analize softverske arhitekture u agilnim procesima istakli su i Buchgeher, i Weinreich (2014), zaključujući da je najefikasnija tehnika takođe, analiza zavisnosti, ali na nivou koda. Fokus predložene tehnike je da se izdvoje statičke zavisnosti iz izvornog koda, kako bi se izvršilo poređenje implementirane arhitekture sa nameravanom. Identifikovane zavisnosti se koriste kao indikatori odnosa između ova dva nivoa arhitekture, čime se pored standardnih agilnih praksi (kontinuirano testiranje, kontinuirana analiza koda, kontinuirana integracija koda, kontinuirano refaktorisanje i programiranje u paru), obezbeđuje dodatna kontinuirana kontrola kvaliteta.

Nakon završetka analize zavisnosti, Brown, Nord i Ozkaya (2010) smatraju da je ključan momenat optimalan odabir potrebnih arhitekturalnih elemenata, koji obezbeđuju skalabilnost release-a. Autori u tu svrhu koriste teoriju prave opcije - model finansijske analize, koji se koristi u korporativnim finansijama, u cilju donošenja odluke da li je isplativo investirati u odgovarajuću poslovnu odluku. Analiza prave opcije može biti uspešno korišćena u planiranju release-a tj. za alokaciju arhitekturalnih elemenata na release-ove. Analiza prave opcije i upravljanje tehničkim dugom mogu da optimizuju investicije arhitekturalnih odluka, na osnovu rezultata analize zavisnosti i rezultata cost/benefit analize arhitekturalne odluke koja se razmatra. Konačna odluka treba da bude opravdana i kroz prizmu mitigacije rizika od neizvesne budućnosti. Cilj je odabrati dovoljno dobro i isplativo rešenje danas, bez ugrožavanja potrebe za potpunijim rešenjem sutra.

Teoriju Prave opcije koriste i Blair, Watt i Cull (2010), ali za rešavanje problema iznalaženja najboljeg momenta donošenja arhitekturalnih odluka, polazeći od stanovišta da se one u agilnom procesu donose ili prerano ili prekasno. Predlažu okvir koji timu može pomoći da prepozna najprikaladniji momenat za donošenje pojedinih arhitekturalnih odluka. Okvir podrazumeva pravljanje Excel tabele, gde kolone čine razvojne faze, a redove arhitekturalna pitanja. Predloženi okvir ima cilj da up front dizajn drži u okvirima, kako ne bi upao u zamku BDUF.

Poboljšanjem procesa donošenja arhitekturalnih odluka u agilnim procesima bavili su se i van der Ven, Bosch, i Groningen (2014), polazeći od činjenice da se u agilnim projektima arhitekturalne odluke donose JIT i to od strane programera, dok arhitekta ima ulogu savetodavnog karaktera u ovom procesu. Autori predstavljaju okvir 3A (Agile, Architecture, Axes) zasnovan na tri ose koje se trebaju razmotriti na svakom projektu, kako bi se postavio

jedinstveni proces donošenja arhitekturalnih odluka koji odgovara datom projektu. Na prvoj osi (Who) predstavljene su uloge koje mogu imati odgovornost u procesu donošenja arhitekturalnih odluka (razvojni tim, arhitekta aplikacije, domen arhitekta, enterprise arhitekta, menadžment). Na drugoj osi (How) predstavljeni su načini dokumentovanja arhitekturalnih odluka u cilju komunikacije sa stejkholderima (direktna komunikacija, beleške sa sastanaka, ad hoc dokumentacije - wiki, neobavezna dokumentacija zasnovana na šablonima i obavezna dokumentacija zasnovana na šablonima). Treća osa (When) prikazuje period od donošenja arhitekturalnih odluka do dobijanja povratnih informacija o njenoj validnosti (više od 6 meseci, 1-6 meseci, manje od 1 meseca).

Kruchten (2010) predlaže set heuristika za rešavanje problema balansa između razvoja funkcija i arhitekture softvera. Implementacija predloženih heuristika zahteva workshop različitih uloga na projektu, sa ciljem da se razmotri osam predloženih dimenzija: semantika, obim, životni ciklus, uloge, dokumentacija, metod, vrednost i trošak. Davanjem odgovora na ključna pitanja iz ovih oblasti, učesnici usvajaju zajednički mentalni model oko primene pojednih arhitekturalnih praksi. Nakon toga mogu definisati proces upravljanja i sam tehnički proces koji usmerava arhitekturalne aktivnosti u agilnom procesu (tzv."zipper model"). Chen i Babar (2014) su istakli četiri kategorije kontekstualnih faktora (projekat, tim, praksa i organizacija), koji determinišu da li se efikasna arhitektura može razviti samo izmenama programskog koda. Predloženi frejmvork je komplementaran Kruchten-ovom (2013) kontekstualnom modelu razvijenom na osnovu iskustva. Babar i Chen su Kruchtenov model proširili sa empirijski identifikovanim faktorima, kao što su: iskustvo i veštine.

Hadar i Silberman (2008) predlažu metod C3A - koji pojam agilne arhitekture tumači kao sintezu tzv. referentne arhitekture, koja ilustruje viziju u tehničkom i funkcionalnom smislu (rezultat je planiranja) i tzv. arhitekture implementacije, koja predstavlja deo referentne arhitekture. Metod sadrži proces za evaluaciju referentne arhitekture i evoluciju arhitekture implementacije, kojim se analizira gap između ostvarenog i planiranog u release-u. Ključni koraci metode su: slušaj i posmatraj, gledaj, reflektuj, unapredi, prouči i počni.

Nekolicina autora bavila se pitanjem uloge softver arhitekte u ostvarivanju balansa up front i nascentne arhitekture u agilnim procesima razvoja.

Faber (2010) smatra da je promena konvencionalne uloge arhitekte ključna za balansirano ulaganje u funkcionalne/nefunkcionalne zahteve. Ističe da su arhitekte nosioci ukupnog kvalitata sistema i da od njihovog izbora adekvatnih odluka dizajna zavisi ravnoteža između implementacije funkcionalnih i nefunkcionalnh zahteva sistema. Arhitekte treba da budu provajderi usluga i programerima i klijentu, imajući različite uloge u interakciji sa njima. Klijentu treba da obezbede vrednost, kroz ispunjavanje nefunkcionalnih zahteva sistema, a programerima kontinuiranu podršku tokom implementacije. Stoga je neophodno da na početku projekta izgrade/nadograde svoje sistemsko, tehnološko i domensko znanje, koje im je neophodno za pružanje kontinuirane pomoći programerima i za balansirano ulaganje u

kvalitet sistema tokom implementacije. Uloga arhitekte, kao provajdera usluga, realizuje se kroz dve faze (Faber, 2010):

1. Fazu pripreme, koja podrazumeva sticanje tehnoloških i sistemskih znanja, naročito u pogledu nefunkcionalnih zahteva sistema, kako bi pružao adekvatnu podršku programerima tokom razvoja. Sastoji se od:
   - Elaboracije nefunkcionalnih zahteva sistema, koji će kasnije biti transformisani u implementacione zadatke razvojnog tima.
   - Opisa globalne arhitekture i delova koji su krucijalni za sistem.
   - Identifikovanja struktura (koje se odnose na glavni deo sistema i čija je modifikacija u fazi održavanja skupa) i pravila dizajna (npr. uputstva za stil korisničkog interfejsa, uputstva za programiranje, rukovanje greškama i porukama itd.) Uz tesnu saradnju sa programerima.
   - Razvoja prototipova u cilju provere delova sistema koji treba da ispune prethodno utvrđene nefunkcionalne zahteve.
   - Razvoja skeleta sistema putem izvršnog koda, koji ne sadrži funkcionalnosti, već samo sistemske konekcije i interfejs. Cilj je da se implementiraju prototipovi ponašanja sistema (nefunkcionalni zahtevi).

2. Fazu podrške implementaciji, koja podrazumeva da arhitekta zajedno sa programerima učestvuje u razvoju krucijalnih delova sistema i kritičnih elemenata kvaliteta sistema. Podrazumeva sledeće aktivnosti:
   - Pomaganje razvojnom timu u postavljanju arhitekture aplikacije, koja treba da bude konzistentna sa postavljenom osnovnom, krucijalnom arhitekturom sistema. Nakon definisanja arhitekture aplikacije, razvojni tim će moći samostalno da implementira funkcionalnosti.
   - Direktno učešće u implementaciji u cilju dobijanja povratnih informacija o arhitekturi tj. Ranog uočavanja i rešavanje arhitekturalnih problema, da bi adekvatno podržala implementaciju potrebnih funkcionalnosti.
   - "Gašenje požara", podrazumeva rešavanje komplikovanih arhitekturalnih problema koji nastaju u implementaciji (npr. optimizacija performansi tokom integracije sistema). Arhitekturalni nedostaci mogu biti detektovani i od strane programera, primenom statičke analize koda, ali praksa pokazuje da programeri često odlažu i zanemaruju rešavanje ove vrste problema. Razlog za to je pritisak na projektu, zbog kratkih vremenskih rokova, koji imaju po pitanju implementacije funkcionalnosti. Međutim, investiranje u arhitekturu aplikacije treba da se vrši što ranije, dok razvoj ne odmakne i za to je ključna uloga arhitekte. To ne znači da arhitekte trebaju da budu uključene do detalja u proces implementacije, već samo u onom delu razvoja sistema koji više timova iznova upotrebljava i zbog čega ti delovi imaju veći uticaj na kvalitet sistema.
   - Podsticanje kontinuirane komunikacija sa stejkholderima i svim projektnim ulogama u cilju upravljanja kvalitetom ukupnog sistema.

Hadar (2012) takođe ističe da je važno uključivanje softver arhitekte kroz ceo razvojni proces na agilnim projektima, a Blair, Watt i Cull (2010) njegovu tesnu kolaboraciju sa razvojnim timom uz kontinuirano deljenje ideja tokom celog perioda trajanja projekta.

Hopkins i Harcombe (2014) smatraju da je uloga softver arhitekta neizostavna u razvoju i isporuci velikih, kompleksnih sistema, koji obično zahtevaju razvoj i integraciju više sistema i koordinaciju stotine pojedinaca. Softver arhitekta, po njima, jedini može da odluči o vitalnim aspektima sistema, jer to često nisu u stanju ni senior programeri. Zadatak arhitekte je da na početku projekta sagleda problem koji se rešava, iz više različitih perspektiva, jer je svaki poslovni problem drugačiji i zahteva različite tačke gledišta, kako bi se identifikovali jedinstveni aspekti. Nakon konceptualnog opisa sistema, od strane softver arhitekte, agilan tim može da odluči kako će biti testiran (npr. dinamičko izvršavanje, statična testiranja ili simulacije). Ranom postavkom i proverom ovog procesa, arhitekta treba sopstvenu viziju sistema da pretvori u zajedničku (celog razvojnog tima) i da prati razvoj, kako bi reagovao u slučaju neočekivanih problema ili neophodnih promena.

Madison (2010) smatra da softver arhitekta treba da bude vezivno tkivo agilnog procesa razvoja i metoda za razvoj softverske arhitekture. On mora dobro da razume agilni proces, kao i da uspostvi balans između poslovnih i arhitekturalnih prioriteta, kako bi razvio agilnu arhitekturu i time iskoristio prednosti oba pristupa. Predstavio je hibridni model Scrum-a, XP-a i sekvencijalnog upravljanja projektima za razvoj agilne arhitekture. Zalaže se za korišćenje arhitekturalnih funkcija i veština (komunikacija, nefunkcionalni zahtevi, izbor odgovarajućeg softvera i hardvera, paterni dizajna) kroz četiri tačke procesa: up front planiranje, postavljanje priče, sprint i izvršni (working) softver.

Hopkins i Harcombe (2014) ističu da je važno balansirano primenjivati up front arhitekturalne aktivnosti, da se ne bi ugrozio koncept agilnosti na velikim i kompleksnim projektima. Preporučuju da svaki veliki projekat otpočne sa up front analizom rizika, kako bi se identifikovale i izolovale oblasti kompleksnosti; identifikacijom elemenata softvera; infrastrukture i arhitekture podataka. Početak projekta podrazumeva i opis problema na konceptualnom nivou, kao i predlaganje rešenja za datu oblast problema. Ponekad se problem/rešenje iznalazi izučavanjem postojećih sistema ili izborom komercijalnih rešenja za pojedine oblasti. Takođe, arhitekta što ranije treba da odluči i kako će najrizičnije aspekte svake identifikovane perspektive problema testirati, da identifikuje arhitekturalno značajne zahteve i pravi prototipove ukoliko se ukaže potreba.

Qureshi (2012) je za razvoj velikih i srednjih projekata predlažio modifikaciju i proširenje razvojnih faza XP procesa: planiranje projekta, analiza i upravljanje rizicima, dizajn i implementacija, i testiranje, dok su Nord i Tomayko (2006), za agilne projekte velikih i kompleksnih sistema, predložili hibridni model zasnovan na integraciji XP procesa sa metodama za razvoj softverske arhitekture, koje je razvio Institut za softverski inženjering Carnegie Mellon univerziteta (Architecture Tradeoff Analysis Method, Quality Attribute

Workshop, Attribute-Driven Design method, Cost Benefit Analysis Method, Active Reviews for Intermediate Design). Ove metode mogu dodati vrednost agilnim procesima, jer naglašavaju nefunkcionalne zahteve i njihovu ulogu u dizajnu arhitekture.

Decentralizovanost, heterogenost i interoperabilnost, usled integracija sistema sa drugim sistemima u organizaciji ali i van nje, izazov je koji karakteriše savremeno poslovanje. Razvoj ovakvih sistema obično uključuje više od 100 članova tima i često moraju biti skalirani do najviših nivoa performansi i bezbednosti. Često su i mission critical sa zahtevom da nikada "ne propadnu". Sve ovo zahteva jaku arhitekturalnu podršku sistema i dobru dokumentovanost, ali sa druge strane, korisnici očekuju da ova kompleksna poslovna softverskih rešenja budu i prilagodljiva promenama u poslovnom okruženju, što zahteva primenu agilnih principa razvoja. Boehm, Lane, Koolmanojwong i Turner (2010) identifikovali su tri faktora koja se trebaju razmatri prilikom uspostavljanje balansa između arhitekture i primene principa agilnosti na projektima razvoja kompleksnih sistema: veličina sistema, kritičnost sistema i promenljivost zahteva. Predlažu pristup koji je zasnovan na kvantitativnoj proceni troškova i rizika (primenom modela COCOMO II i koncepta Risk Resolution factor, u nastavku RESL) usled investicija u arhitekturu.

Boehm, Lane, Koolmanojwong i Turner su zaključili da troškovi koji potiču od dorade arhitekture predstavljaju eksponencijalnu funkciju u odnosu na veličinu projekta. Drugim rečima, na malim projektima (10.000 linija koda) prilagodjavanje arhitekture rapidnim promenama, tehnikom refaktorisanja, povećaće arhitekturalne investicije za svega 14%, dok na velikim projektima (više od 10.000 linija koda) dorada arhitekture putem refaktorisanja povećaće arhitekturalne investicije za čak 91%. Posledica ovako visokih dodatnih troškova jeste i negativan povrat investicije (Return of Investment, u nastavku ROI). Zbog toga je veličina projekta jedan od ključnih faktora koji utiče na veći obim up front razmatranja arhtekture. Na osnovu rezultata istraživanja autori su, za razvoj ovakvih kompleksnih sistema, predložili hibridni (agile/plan driven) procesni okvir - Incremental Commitment Model (u nastavku, ICM). Okvir predstavlja sintezu koncepata iz postojećih procesnih modela: koncept rane verifikacije i validacije iz V modela, konkurentni koncepti iz modela konkurentnog inženjeringa, koncept lakih procesa - iz Lean-a i agilnih modela, koncept rizika iz spiralnog modela, koncept faza i ključnih tačaka - iz RUP-a.

Nord, Ozkaya i Sangwan (2012) uspostavljanje ravnoteže up front i nascente arhitekture obezbeđuju kroz primenu Lean koncepta upravljanja protokom vrednosti kroz razvojni proces. Prema Lean konceptu, sav otpad koji potiče od arhitekturalnih aktivnosti, može se svrstati u tri međusobno tesno povezane kategorije: hiperprodukcija, kašnjenje i nedostaci. Cilj istraživanja je bio da se pokaže kako razvoj arhitekture kroz inkremente, utiče na troškove koji proizilaze iz pomenutih vrsta otpada, te identifikovanje načina poboljšanja procesa i efikasnog upravljanja vremenom i otpadom u razvoju (Waste In Production, u nastavku WIP). Autori smatraju da strategija projektovanja arhitekture kroz mnogo manjih inkrementa smanjuje troškove kašnjenja, koji se javljaju usled čekanja da se celokupan posao

projektovanja arhitekture završi na početku projekta. Međutim, dorada arhitekture je mnogo skuplja, jer može zahtevati značajne izmene. Primenjujući manje većih inkremenata smanjuju se troškovi dorade, ali se povečavaju troškovi kašnjenja zbog većeg obima arhitekturalnih aktivnosti. Troškovi kašnjenja, dakle, nastaju ili zbog čekanja ili zbog kašnjenja procesa implementacije, dok troškovi dorade nastaju zbog arhitekturalnih nedostataka ili otpada usled hiperprodukcije. Ovo su dve ekstremne strategije planiranja razvoja iteracije. Cilj je, međutim, da ukupni kumulativni troškovi projekta budu manji od isporučene kumulativne vrednosti jednog release-a. Da bi razvoj tekao po srednjem putu sa aspekta troškova, autori predlažu primenu koncepta vizualizacije ulaganja u arhitekturu u toku inkrementa, kako bi se demonstrirali efekti arhitekturalnog otpada (usled arhitekturalne hiperprodukcije, kašnjanja ili nedostataka) na celokupan projekat. U tu svrhu, predlažu identifikovanje arhitekturalno značajnih zahteva kao i testova prihvatljivosti, kako bi se obetbedila vidljivosti arhitekturalnih zadataka na listi zadataka iteracije ili Kanban tabli. Sprovođenjem koncepta WIP na testove prihvatljivosti obezbeđuje se poboljšanje protoka procesa razvoja, jer se na taj način upravlja otpadom od prekomernog projektovanja arhitekture (Nord et al., 2012). Da bi se predupredili troškovi koji potiču od otpada usled prepravki arhitekture u kasnijim razvojnim fazama, predlaže se monitoring promene kvaliteta sistema (putem arhitekture), koji se obično javlja kao posledica tehničkog duga (Bellomo, Nord, et al., 2013).

Hinsman, Sangal i Stafford (2009) takođe ističu da je vidljivost arhitekture ključna za uspostavaljanje ravnoteže između up front i nascentne arhitekture, naročito u uslovima brzih promena. Smatraju da tradicionalnom tehnikom refaktorisanja koda nije izvodljivo rešavati problem kompleksnosti koda i prevelike međuzavisnosti koji nastaje usled brzog razvoja (tehničkog duga), prilagođavanja poslovnim promenama i nadogradnje sistema. Predlažu globalniji pristup - refaktorisanje sa aspekta arhitekture, koji podrazumeva proces vođen arhitekturalnim planom, a koji se definiše kao rezultat identifikovanih zavisnosti na nivou struktura. Refaktorisanje se prvo sprovodi na prototipu, a tek potom na izvršnom kodu. Podrazumeva pet koraka: definisanje problema, vizualizacija trenutne arhitekture, model željene arhitekture u smislu aktuelnih elemenata, konsolidovanja i izmena baze koda i automatizacija upravljanja arhitekturom kroz kontinuirane integracije. Iskustveni rezultati pokazuju da analiza zasnovana na arhitekturi može poboljšati produktivnost u razvoju softvera i smanjiti troškove održavanja sistema (Hinsman et al., 2009).

Stal (2014) predlaže da refaktorisanje sa aspekta arhitekture treba da bude uključeno u agilan razvoj kao obavezan proces, jer se na taj način rano mogu detektovati i eliminisati pogrešne ili neodgovarajuće odluke dizajna, čime se obezbeđuje visok kvalitet sistema. Aktivnost treba da se sprovodi barem jednom po iteraciji. Stal predlaže pristup za sistematsko sprođenje procesa refaktorisanja arhitekture, koji se sastoji od sledećih ključnih koraka: procena arhitekture (identifikovanje problema dizajna); prioritizacija arhitekturalnih pitanja (utvrđivanje redosleda rešavanja identifikovanih problema u odnosu na njihovu važnost); selekcija odgovarajućih paterna (ukoliko postoje) za svaki identifikovani problem ponaosob ili konvencionalni redizajn; obezbeđivanje kvaliteta, putem procene ili testiranja, kako refaktorisanjem ne bi bila

ugrožena semantika. Arhitekturalni problemi koji se trebaju rešavati ovim procesom su: nejasne uloge entiteta, kompleksnost arhitekturalnog rešenja, velika centralizovanost, asimetrična struktura ili ponašanje i sl. Ukoliko se problemi nepravovremeno rešavaju, nastupiće erozija dizajna, a tada refaktorisanje arhitekture neće moći da reši date arhitekturalne probleme. U tim slučajevima jedini način za oporavak arhitekture je reinženjering ili ponovni razvoj arhitekture.

Keuler, Wagner, Winkler (2012) vidljivost arhitekturalnih struktura obezbeđuju primenom predloženog frejmvorka, koji omogućava povezivanje arhitekturalnih odluka sa izvršnim kodom. Proces počinje opisom arhitekture u XML fajlu, na osnovu kojeg se generiše kod koji implementira opisanu arhitekturu i predstavlja svojevrstan kostur aplikacije. Okvir doprinosi da više timova radi paralelno na istom kodu, sa minimalnim konfliktima, sve dok se efikasno upravlja zavisnostima izvornog koda. Na ovakav način programerima se omogućava da se bave funkcionalnošću komponente, dok implementirani frejmvork upravlja komponentama, interfejsima i njihovim zavisnostima.

Rezultati sprovedenog istraživanja literature pokazuju, da velika grupa problema potiče od još jednog suštinskog konflikta: zahteva minimalizma u agilnim procesima i potrebe za dobro dokumentovanom arhitekturom u kompleksnim sistemima (Hadar & Sherman, 2012). Potrebu za adekvatnim upravljanjem arhitekturalnim znanjem u agilnim procesima, pojačava i činjenica sve prisutnijeg globalnog razvoja softvera (Clerc, Lago, Vliet, 2011).

Istražujući ovaj problem, Babar (2009b) je došao do empirijskih nalaza koji pokazuju da je u praksi pristutna modifikovana tradicionalna praksa dokumentovanja u formi Software Architectural Overall Plan (u nastavku, SAOP), ali samo za konceptualni opis arhitekture. Ostale odluke dizajna opisuju se na wiki-ju. Odgovornost arhitekturalne analize i izbora arhitekturalnog rešenja prebačena je na klijente. Evaluacija arhitekture je po prilično neformalna i obavlja se u okviru Architecture Review Board (u nastavku, ARB), gde programeri ocenjuju predložena rešenja. Empirijski rezultati drugog istraživanja, međutim, pokazuju da agilni programeri imaju pozitivne stavove prema softverskoj arhitekturi i da arhitekturalne artifakte vide korisnim u komunikaciji između članova razvojnog tima, kasnijim odlukama dizajna, dokumentovanju i oceni rešenja (Falessi et al., 2010).

U praksi su prisutni uglavnom eksteremni slučajevi: ili preobimna dokumentacija ili odsustvo arhitekturalne dokumentacije (Hadar, Sherman, Hadar, Harrison, 2013). Slaba dokumentacija za posledicu ima "isparavanje" arhitekturalnih informacija i znanja (Hadar & Silberman, 2008), implicira nerazumevanje arhitekture i slabu komunikaciju što vodi u haos i neuspeh projekta (Pareto, Sandberg, Eriksson, Ehnebom, 2011). Previše dokumentacije, predstavlja otpad, u smislu gubitka vremena i resursa i udaljavanje od suštine (Pareto, Sandberg, Eriksson, Ehnebom, 2011).

Previše dokumentacije, nije efikasno za agilne procese i iz razloga što stejkholderi kao što su programeri, korisnici i kupci nemaju energije ni vremena da izučavaju tradicionalne arhitekturalne artifakte, kako bi razumeli arhitekturu. Programeri žele jasne i decizne smernice dizajna, kupci i korisnici žele jasno razumevanje promena radnog okruženja budućeg sistema, kako bi bili sigurni da arhitektura ispunjava njihove poslovne potrebe. Arhitekte koje nasleđuju nečiji rad žele jasno istaknute ključne aspekte arhitekture, uključujući i obrazloženja arhitekturalnih opcija koje je originalni arhitekta razmatrao i implementirao (Tyree & Akerman, 2005). Faber (2010) smatra da je upravo softver arhitekta, sa ulogom provajdera usluga, odgovoran za usvajanje središnje pozicije u odnosu na slabo/prekomerno davanje i dokumentovanje smernica u razvoju.

Važnost uspostavljanja ravnoteže između previše (tradicionalno) i premalo (agilno) arhitekturalne dokumentacije, u cilju očuvanja i deljenja arhitekturalnog znanja, prepoznala je nekolicina istraživača, predlažući različite načine za rešavanje ovog arhitekturalnog problema.

Rešenje ovog izazova Tyree i Akerman (2005) vide u dokumentovanju pre svega arhitekturalnih odluka i njihovih obrazloženja. Smatraju da adekvatno dokumentovanje arhitekturalnih odluka predstavlja efikasan alat za komunikaciju sa stejkholderima, kao i za njihovo brzo i jednostavno razumevanju arhitekture. Pored toga efikasno je sredstvo za usmeravanje implementacije sistema, ali i za praćenje sledljivosti između zahteva i njihove tehničke implementacije. Dokumentovane odluke mogu biti korišćene i kao sredstvo evaluacije arhitekturalnih rešenja.

Hadar, Sherman, Hadar i Harrison (2013) daju predlog šablona za dokumentovanje arhitekture, koji je usklađen sa agilnom filozofijom i lean dokumentacijom. Predloženi arhitekturalni dokumenat je rezultat sprovedenog emirijskog istraživanja i u fokusu ima samo najrelevantnije informacije o dizajniranom arhitekturalnom rešenju release-a. Sačinjen je od 4 glavna dela: pregled proizvoda, ciljevi proizvoda za predstojeći release, pregled arhitekture proizvoda i nefunkcionalni zahtevi. Opis arhitekture se vrši preko 4 konceptualna nivoa: sloj sistemskih komponenti, sloj zajedničkih komponenti, sloj komponenti za spoljnu integraciju i sloj funkcionalnih komponenti. Pristup je podržan alatom za automatsko generisanje dokumentacije. Alat za dokumentovanje je povezan sa alatom za modelovanje arhitekture i dele istu bazu podataka, tako da su promene na modelu ažurne i u dokumentima (Hadar, Sherman, Hadar, Harrison, 2013).

Pareto, Sandberg, Eriksson i Ehnebom (2011) predložili su metod za prioritizaciju arhitekturalne dokumentacije, za projekte velikih organizacija, u kojima se sprovode korektivne akcije nad postojećom arhitekturom. Arhitekturalna dokumentacija, po njima podrazumeva skup modela i pogleda koji razmatraju različite arhitekturalne aspekte iz perspektiva različitih stejkholdera. Utvrđivanje prioriteta arhitekturalne dokumentacije, iz perspektive grupe stejkholdera, podrazumeva da postojeća arhitekturalna dokumentacija možda ne pokriva arhitekturalna pitanja bitna za korisnike sistema. Predloženi metod

kombinuje kolaborativne i analitičke tehnike sa različitim grupama stejkholdera, u cilju identifikovanja aritekturalnih oblasti koje je potrebno poboljšati. Zahteva angažovanje i arhitekti, koji imaju znanja da razviju i održavaju potrebne arhitekturalne poglede, ali i menadžera koji treba da obezbede ekonomske resurse i organizaciju.

Eloranta i Koskimies (2012) slabo dokumentovanje u agilnim procesima posmatraju kao problem deljenja arhitekturalnog znanja, ističući da na projektima srednje veličine i kompleksnosti nije dovoljna samo direktna komunikacija članova razvojnog tima i stejkholdera. Autori predlažu primenu koncepta Architecture Knowledge Management (u nastavku, AKM), koji su uskladili i integrisali sa Scrum procesom. Pristup podrazumeva izgradnju arhitekturalne baze znanja i primenu tehnike evaluacije arhitekturalnih odluka. Predložena tehnika evaluacije (DCAR) analizira svaku arhitekturalnu odluku ponaosob, od dole ka vrhu, i to u momentu njenog donošenja. Nakon analize, arhitekturalne odluke i njihova obrazloženja se pohranjuju u arhitekturalnu bazu znanja, koja predstavlja svojevrstan informacioni sistem. Arhitekturalna baza znanja može automatski da generiše (on-line ili štampane) arhitekturalne izvještaje ili dokumente za specifične svrhe, koristeći informacije koje su sačuvane u njoj.

DCAR metod evaluacije je pogodan za agilne procese jer zahteva 20 čovek/sati, kao i zbog svoje inkrementalne prirode koja omogućava evaluaciju delova arhitekture tj. podskupa arhitekturalnih odluka u jednom momentu, umesto cele arhitekture, što je karakteristično za tradicionalne metode evaluacije softverske arhitekture (npr. ATAM metod). DCAR se bazira na istraživanju faktora koji utiču na odluke dizajna arhitekture. Moguća arhitekturalna rešenja rezultat su, procene arhitekturalnih odluka naspram faktora identifikovanih u procesu evaluacije. Procena se vrši istraživanjem koji faktori su uticali na odluku i u kom pravcu. Odluke se potom klasifikuju kao problematične, potencijalno problematične ili sigurne. Suština DCAR metode sastoji se, dakle, od tri faze: identifikovanje faktora, identifikovanje odluka i analiza i prioritizacija odluka (Eloranta & Koskimies, 2012).

3.3.5.1 Validnost sprovedenog teoretskog istraživanja

Predmet istraživanja predstavlja mlado i poprilično teško područje za sprovođenje empirijskih istraživanja, iz razloga što je teško utvrditi kauzalnosti usled prisustva mnogih drugih faktora koji mogu uticati na ishod. Sa druge strane, postoji dobro definisana i rigidna metodologija koja propisuje stroge korake i zahteve koje mora da zovolji dizajn empirijskog istraživanja. Stoga, ne iznenađuje činjenica da velika većina tvrdnji o odnosu arhitekturalnih i agilnih praksi i njihovoj integraciji ima ograničenu naučnu vrednost. Najveći deo radova koji postoji u literaturi je baziran isključivo na mišljenju eksperata, a daleko manji broj na empirijskim podacima i dokazima.

Radovi bazirani isključivo na ekspertskom mišljenju, bez predlaganja pristupa i nekog vida njegove validacije, nisu obuhvaćeni opisanim istraživanjem. Njihova eliminacija sprovedena je u skladu sa prethodno definisanim kriterijuma isključivanja.

Pretnju validnosti sprovedenog teoretskog istraživanja predstavlja i upotreba nejasno definisanih termina, kao što su „agile“ i „architecture“. U cilju prevazilaženja ove pretnje, definicije ovih termina usaglašene su bile sa još dva istraživača na Ekonomskom fakultetu u Subotici.

Takođe, validnosti sprovedenog istraživanja doprinosi i činjenica da je razvijeni okvir za sprovođenje istraživanja evaluiran je od strane dva nezavisna istraživača. Evaluacija je sprovedena i u delu isključivanja/uključivanja relevantnih radova u istraživanje, kao i delu analize njihovih sadržaja. Na ovaj način eliminisana je mogućnost analize naučnog materijala i interpretacije njegovih rezultata po slobodnom nahođenju istraživača.

3.3.5.2 Zaključci sprovedenog teoretskog istraživanja

Glavni zaključak autora je da postoji nedostatak naučnih argumenata za mnoge tvrdnje u postojećoj literaturi. Velika grupa radova sadrži tvrdnje koje su isključivo bazirane na stručnom mišljenju naučnika ili eksperata iz date oblasti. Takođe, brojni radovi svoje pristupe testiraju na studentskim projektima, a mali broj radova predložene pristupe testira na industrijskim slučajevima. Može se zaključiti i da je generalno nedovoljno empirijskih nalaza, koji se bave odnosom agilnih procesa razvoja i softverske arhitekture.

Konkretno, od ukupnog broja relevantnih radova ovog istraživanja (34), svega 12 je zasnovano na empirijskim nalazima. Vrlo je mali je i broj radova (oko 15) koji se na neki način bave pitanjem balansa arhitekture i razvoja funkcionalnosti softvera u agilnim procesima.

Kodiranjem koncepata, ciljeva i rezultata relevantnih radova identifikovan je veliki broj tema i koncepata koji se mogu agregirati u jednu glavnu kategoriju: *eksplicitne arhitekturalne aktivnosti,* koja sadrži tri podkategorije: *upravljanje arhitekturalnim znanjem u agilnim procesima, faktori, prakse i pristupi za balansiranje up front i nascentne arhitekture* i *uloga softver arhitekte.*

Na osnovu opisanih rezultata teorijskog istraživanja, može se zaključiti da u literaturi, u okviru navedenih kategorija, postoji više vrsta arhitekturalnih problema, kojima se bave istraživači. Pregled ključnih kategorija arhitekturalnih problema, prema autorima koji su ih izučavali, dat je u tabeli 3.18.

Tabela 3.17 Arhitekturalna pitanja/problemi identifikovani u literaturi

Aktuelna arhitekturalna pitanja	Izvori
nefunkcionalni zahtevi	Babar (2009b); Bellomo, Nord i Ozkaya (2013); Huang, Czauderna i Mirakhorli (2014); Jeon, Han, Lee i Lee (2011); Brown, Nord i Ozkaya (2010); Faber (2010);
anticipiranje budućih potreba sistema i postavka arhitekture	Brown, Nord i Ozkaya (2010); Isotta-Riches i Randell (2014); Weitzel, Rost i Scheffe (2014); Nord, Ozkaya i Sangwan (2012); Bellomo, Ozkaya i Nord (2013); Boehm, Lane, Koolmanojwong i Turner (2010); Pérez, Díaz, Garbajosa i Yagüe (2014); Friedrichsen (2014);
kontekstualni faktori u razvoju arhitekture	Chen i Babar (2014); Kruchten (2013);
balans up front i nascentne arhitekture	Friedrichsen (2014); Waterman, Noble i Allan (2012); Isotta-Riches i Randell (2014); Brown, Nord i Ozkaya (2010); Buchgeher i Weinreich (2014); Blair, Watt i Cull (2010); van der Ven, Bosch i Groningen (2014); Kruchten (2010); Hadar i Silberman (2008); Faber (2010); Hopkins i Harcombe (2014); Qureshi (2012); Nord i Tomayko (2006); Boehm, Lane, Koolmanojwong i Turner (2010); Nord, Ozkaya i Sangwan (2012); Hinsman, Sangal i Stafford (2009);
uloga softver arhitekte	Faber (2010); Hadar (2012); Blair, Watt i Cull (2010); Hopkins i Harcombe (2014); Madison (2010); Babar (2009b); Hadar i Sherman (2012);
vidljivost arhitekturalnih zadataka	Stal (2014); Hinsman, Sangal i Stafford (2009); Keuler, Wagner i Winkler (2012);
dokumentovanje arhitekture	Hadar (2012); Babar (2009b); Falessi i dr. (2010); Hadar, Sherman, Hadar i Harrison (2013); Hadar i Silberman (2008); Pareto, Sandberg, Eriksson i Ehnebom (2011); Tyree i Akerman (2005); Faber (2010); Eloranta i Koskimies (2012);

4.

Empirijsko istraživanje razvoja softverske arhitekture i agilnih procesa

4.1 Opis načina obavljenog istraživanja

Poglavljem se opisuju koraci klasične varijante Delfi metode, kojom je sprovedeno empirijsko istraživanje. Takođe, opisuje se i način uzorkovanja učesnika istraživanja, razvoj instrumenata istraživanja, načini prikupljanja, pripreme i analize podataka, kao i rezultati dobijeni po iteracijama istraživanja.

4.1.1 Opis primene Delfi metode

Primena klasične varijante Delfi metode podrazumevala je tri kruga ili iteracije istraživanja (Keeney, Hasson, McKenna, 2011; Helmer & Rescher, 1959):

I. Prvi krug

1. Prva faza prvog kruga: potvrđena je raspoloživost neophodnih resursa za početak primene klasične varijante Delfi tehnike.
2. Druga faza prvog kruga: određen potrebni nivo saglasnosti respondenata-eksperata.
3. Treća faza prvog kruga:
   - određen ciljni osnovni skup,
   - određen način izbora panela eksperata,

- određena veličina panela eksperata,
- prikupljene adrese mogućih članova panela eksperata,
- definisana strategija za uvećanje stope odgovora respondenata-eksperata (u svakom krugu),
- razvijene administrativne procedure:
  - početno pozivno pismo članovima panela,
  - oblikovan upitnik za polustrukturirani intervju,
  - razvijen sistema kodova za praćenje respondenata, članova panela,
  - kreirani fajlovi za odgovore respondenata.

4. Četvrta faza prvog kruga: utvrđeni datumi za sprovođenje intervjua.
5. Peta faza prvog kruga podrazumevala je izradu dokumenata za prvi krug:
   - pozivno pismo,
   - kratak opis projekta,
   - opis i razjašnjenje procesa primene delfi tehnike,
   - osiguravanje poverljivosti odgovora i anonimnosti respondenata,
   - utvrđeni načini komuniciranja istraživača i respondenata,
   - izrađen upitnik za prvi krug (sa setom otvorenih pitanja),
   - izračunat indeks sadržajne valjanosti upitnika,
   - izrađena potrebna uputstva za respondente.
6. Šesta faza prvog kruga:
   - izabrani članova panela,
   - izabrani respondenti za probni prvi krug,
   - realizovan probni prvi krug, proveren sistem administracije i izvršene potrebne ispravke.
7. Početak prvog kruga – sprovedeni intervjui.

II. Drugi krug

1. Prva faza drugog kruga:
   - sređeni odgovori respondenata,
   - sprovedena analiza sadržaja odgovora/tematska analiza,
   - izrađen upitnik za drugi krug istraživanja (na osnovu rezultata analize sadržaja/tematske analize),
   - izračunat indeks sadržajne valjanosti upitnika/skale za drugi krug,
   - pripremljene informacija za respondente o rezultatima iz prvog kruga,
   - napisana potrebna uputstava respondentima.
2. Druga faza drugog kruga: poslat upitnik/skala i odgovarajuća uputstava respondentima. Prikupljeni odgovori respondenata.
3. Treća faza drugog kruga: pripremljeni odgovori respondenata u drugom krugu za sprovođenja analiza.
4. Četvrta faza drugog kruga podrazumevala je analizu podataka dobijenih u drugom krugu.

III. Treći krug

U klasičnoj Delfi tehnici treći krug je tipično poslednji krug, osim u slučajevima kada za veliki broj stavki nije postignuta saglasnost respondenata.

1. Prva faza trećeg kruga:
   - svakom članu panela omogućeno je da vidi kako su drugi eksperti u panelu odgovorili u drugom krugu,
   - svakom članu panela omogućeno je da se podseti na odgovore koje je davao u drugom krugu.
2. Druga faza trećeg kruga:
   - od svakog člana panela se tražilo da ponovo razmotri stavku i eventualno promeni svoj originalni odgovor, u odnosu na odgovor koji je dala grupa eksperata,
   - prikupljeni i sređeni odgovori eksperata u trećem krugu.
3. Treća faza trećeg kruga podrazumevala je analizu podataka dobijenih u trećem krugu.

4.1.2 Uzorkovanje učesnika istraživanja

Priroda problema istraživanja nalagala je da se vrši nameran odabir jedinica uzorka. Namerno uzorkovanje je bilo neophodno, kako bi ispitanici bili samo oni pojedinci koji imaju potrebne vrste znanja, veštine, stručnost, iskustvo i informacije iz problemskog domena. Stoga su uzorak sačinjavali eksperti, koji mogu ponuditi najbolje informacije za ostvarivanje postavljenih ciljeva istraživanja.

Istraživanje je sprovedeno u referentnim preduzećima iz ICT sektora, na „svrhovitom" - homogenom uzorku od 20 eksperata iz Srbije, što je u skladu sa preporukama za Delfi tehniku (Skulmoski, Hartman, Krahn, 2007; Akins, Tolson, Cole, 2005; Jones & Twiss, 1978; Delbecq, Van de Ven, Gustafson, 1975).

Standardna procedura procesa uzorkovanja eksperata prema (Keeney, Hasson, McKenna, 2011) nalagala je da se eksplicitno definiše nivo saglasnosti eksperata, koji se mora postići za najveći deo stavki tokom istraživanja. Izučavanjem sličnih postojećih istraživanja i druge referentne literature, usvojen je nivo saglasnsti od 70%.

Drugi korak sastojao se u definisanju liste kriterijuma za uzorkovanje učesnika istraživanja tj. eksperata iz oblasti kojom se bavi doktorska disertacija (Keeney, Hasson, McKenna, 2011). Lista je formirana prilagođavanjem opštih kriterijuma, koje su definisali (Skulmoski, Hartman, Krahn, 2007; Ziglio, 1996):

- Znanje i praktična iskustva u domenu problema istraživanja: razvoj softverske arhitekture kompleksnih softverskih rešenja agilnim procesima razvoja.
- Kapacitet i spremnost učesnika da doprinese istraživanju.

- Potvrda da će imati vremena i biti dovoljno posvećen istraživanju.
- Dobre komunikacione veštine.
- Akademski nivo obrazovanja.
- Višegodišnje profesionalno iskustvo (više od 5 godina).

Treći korak procesa uzorkovanja podrazumevao je pravljanje liste najznačajnijih softverskih kompanija u Srbiji, koje se bave razvojem poslovnog softvera, a potom i liste najiskusnijih ljudi u njima. Generisanje liste potencijalnih učesnika istraživanja otpočelo je na osnovu preporuka eksperata, koje je istraživač lično poznavao, jer je to prema Gordon-u (1992) jedan od legitimnih načina uzorkovanja eksperata.

Nakon formiranja preliminarne liste učesnika, pojedinci su pojedinačno kontaktirani, kako bi im se predočio cilj istraživanja, način istraživanja i vremenski period u kojem se očekuje njihovo aktivno učešće. Ukoliko je potencijalni učesnik potvrdio da dobro poznaje problematiku i da je zainteresovan da učestvuje u predočenom istraživanju, bio je označen kao ekspert istraživanja. Takođe, zamoljen je bio i da predloži pojedince koje poznaje i za koje smatra da su stručnjaci iz oblasti problema istraživanja, te se tako inicijalna lista potencijalnih učesnika proširivala. U razgovoru su dobijene osnovne informacije o potencijalnim učesnicima, koje je ekspert predložio, kao i njihov kontakt telefon i elektronska adresa.

Ovakvim procesom definisan je panel od 20 eksperata, čije je prosečno iskustvo bilo 13 godina. Svim ekspertima mejlom je poslato pozivno pismo, uz kratak opis Delfi metode, problema i ciljeva istraživanja (prilog 1).

4.1.3 Instrumenti (metode) za prikupljanje podataka

Poglavlje opisuje način izrade instrumenata istraživanja, za svaki krug istraživanja ponaosob. Takođe, fokus je i na prikupljanju, pripremi i analizi empirijskih podataka.

4.1.3.1 Razvoj instrumenta za prvi krug istraživanja

Razvoj instrumenta za sprovođenje prvog kruga istraživanja podrazumevao je izradu liste okvirnih pitanja otvorenog tipa, na osnovu kojih se sprovodio polustrukturirani intervju. Proces izrade upitnika je otpočeo oktobra 2014. godine, na osnovu prethodno proučene referentne literature. Prva verzija upitnika sadržavala je 62 pitanja. Instrument je podvrgnut evaluaciji od strane pet eksperata, kako bi se ocenila njegova sadržajna valjanost (prilog 2). Prvih pet pitanja iz upitnika nisu bila predmet evaluacije, jer su se odnosila na demografske podatke o ispitaniku. Odabrane eksperte, kojima je poslato pozivno pismo sa instrukcijama o

načinu sprovođenja evaluacije (prilog 3), činili su 3 stručnjaka iz prakse i 2 doktora nauka sa užom naučnom oblašću poslovna informatika.

Za vrednovanje svakog pojedinačnog pitanja korišćena je skala od četiri nivoa, po uzoru na Likertovu:
1. beznačajno; 2. donekle značajno; 3. dovoljno značajno; 4. vrlo značajno.

Pored ocene značajnosti svakog potencijalnog pitanja, eksperti su imali mogućnost da daju i komentare i sugestije uz svako pitanje. Nakon sprovedene evaluacije upitnika, izračunat je indeks sadržajne valjanosti svakog pojedinačnog pitanja (ISVP), kao i upitnika u celini, prema (Polit & Beck, 2006). Indeks sadržajne valjanosti prve verzije upitnika (ISVU) iznosio je 0.76, što je ukazivalo da je neophodno sprovesti izmene u skladu sa mišljenjem eksperata. Modifikovanje upitnika sastojalo se u eliminisanju pojedinih pitanja, čiji je indeks sadržajne valjanosti bio < 0.8, kao i u izmenama formulacija pojedinih pitanja i spajanju grupe vezanih pitanja u jedno pitanje i sl.

Nova verzija upitnika sadržavala je 40 okvirnih pitanja, koja su bila grupisana u 5 tematskih celina: I - Podaci o ispitaniku i kontekstu, II - Podaci o modelima razvojnih procesa, III - Podaci o identifikovanim problemima i uzrocima, IV - Podaci o softverskoj arhitekturi, V - Kontekstualni faktori (prilog 4). Isti je poslat ekspertima na ponovnu ocenu, u cilju dodatnih izmena. Eksperti nisu imali novih zahteva za izmenom pitanja nove verzije upitnika. Indeks sadržajne valjanosti nove verzije upitnika iznosio je 0.98. Proces kreiranja instrumenta za prvi kruga istraživanja bio je završen u decembru 2014. godine.

4.1.3.2 Prikupljanje, priprema i analiza podataka prvog kruga istraživanja

Pregledom literature utvrđeno je da postoje različita mišljenja i značajne varijacije o prihvatljivosti postignute stope odgovora ispitanika u istraživanjima Delfi metodom. Ovaj procenat kreće se od 8% (Cooney et al., 1995) do 100% (McKenna, Keeney, Bradley, 2003; Owens, Ley, Aitken, 2008). Razlog za ovakve varijacije je dužina trajanja i broj iteracija istraživanja, te je čest slučaj da eksperti odustanu u toku procesa.

U prvom krugu istraživanja, koji je otpočeo početkom januara 2015. godine, stopa odgovora iznosila je 100%. Sprovođenju intervjua prethodio je dogovor sa svakim ekspertom ponaosob oko zakazivanja tačnog dana i vremena intervjuisanja. Proces je rezultirao generisanjem tačnog plana sprovođenja svih intervjua.

Intervjui su se sprovodili licem u lice i bili su snimani, a vreme trajanja intrevjua bilo je od 40 minuta do nekoliko časova. Snimanjem, te naknadnim transkriptom intervjua i njegovom potvrdom od strane ispitanika, obezbedila se veća tačnost i potpunost podataka. Svakom ekspertu je predočena anonimnost, u smislu neobjavljivanja informacija koje bi mogle da

ukazuju na njihov identitet, ili identitet kompanije u kojoj rade. Anonimnost je obezbedila veću otvorenost i slobodu eksperata tokom razgovora. Svi intervjui su se sprovodili u okviru njihovih kompanija. Takođe, na početku svakog razgovora ekspertima su ponovo na uvid data istraživačka pitanja i ciljevi.

Priprema podataka u cilju obavljanja kvalitativne analize istih, podrazumevala je proces transkriptovanja snimljenih intervjua. Pomenuti proces otpočeo je početkom februara 2015. godine i trajao je oko 30 dana. Transkriptovani podaci u Word-u, poslati su eksertima na proveru i potvrdu, a nakon toga je formiran kodni sistem (ISP1, ISP2….ISP20), koji je umesto prezimena i imena predstavljavao identifikaciju eksperata. Transkriptovani podaci su snimljeni u softver NVivo, u kojem se sprovodila tematska analiza njihovih sadržaja.

Tematska analiza sadržaja podrazumevala je proces kodiranja, koji je sproveden primenom deduktivno-induktivne metode i manuelnim kodiranjem u softveru NVivo. Naime, izvestan broj kategorija i podkategorija utvrđen je prethodnim izučavanjem referentne literature, dok je određeni broj kategorija nastao iz samih podataka.

Glavni cilj sprovedene tematske analize sadržaja bio je bolje razumevanje istraživanog fenomena - proces razvoja softverske arhitekture u agilnim procesima. Odnosno, cilj je bio identifikovati eksplicitne arhitekturalne aktivnosti koje koriste eksperti u praksi, kao i mesta u agilnim procesima razvoja gde je njihova primena korisna. Dobijeni rezultati predstavljali su input za pripremu druge iteracije istraživanja, koja je podrazumevala izradu e-upitnika i dobijanje kvantitativnih rezultata istraživanja, neophodnih za davanje odgovora na istraživačka pitanja.

Slika 4.1 prikazuje glavnu identifikovanu kategoriju u procesu kodiranja intervjua: *dizajn agilne arhitekture* i njene tri podkategorije: *faktori arhitekturalne strategije*, *eksplicitne arhitekturalne aktivnosti* i *uloge i odgovornost donošenja arhitekturalnih odluka*. Glavna kategorija opisuje proces razvoja agilne arhitekture, kao i kako se i koje odluke donose up front u odnosu na nascentnu arhitekturu.

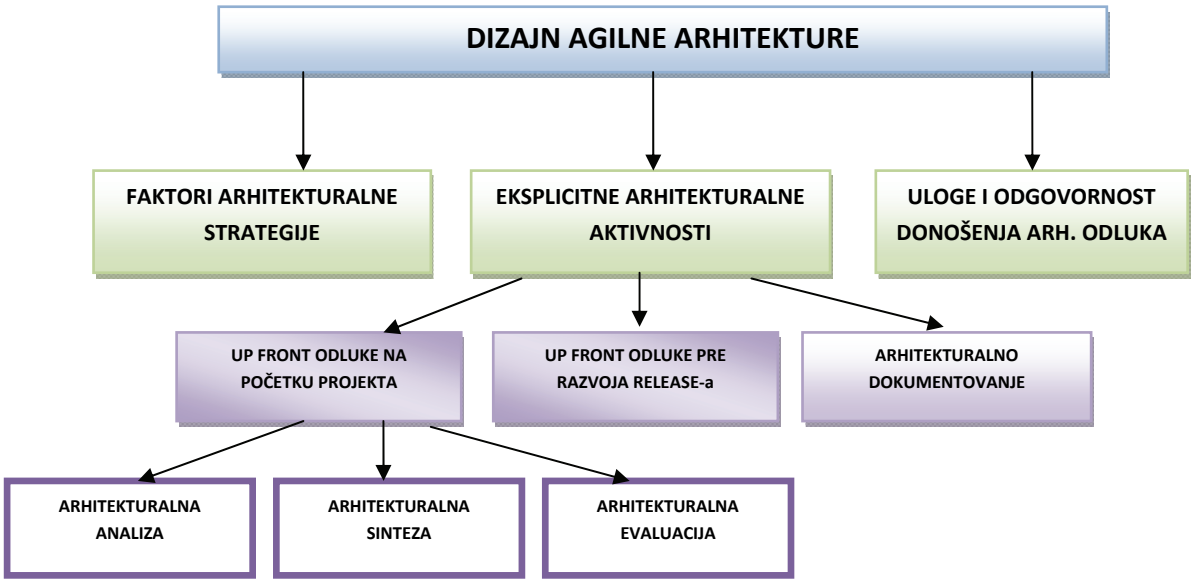
Podkategorija *faktori arhitekturalne strategije* predstavlja faktore koje agilni timovi razmatraju prilikom utvrđivanja koliko up front arhitekturalnih odluka i aktivnosti je potrebno na projektu.

Podkategorija *uloge i odgovornost donošenja arhitekturalnih odluka* predstavlja uloge koje su odgovorne za donošenje arhitekturalnih odluka, kao i njihove odgovornosti prilikom razvoja arhitekture u agilnim procesima.

Podkategorija *eksplicitne arhitekturalne aktivnosti* sadrži sve eksplicitne arhitekturalne aktivnosti koje se donose up front, na početku projekta i koje su kategorizovane prema tradicionalnim fazama razvoja softverske arhitekture: arhitekturalna analiza (razumevanje problema,

konteksta, poslovne arhitekture, rizika), arhitekturalna sinteza (izbor arhitekturalnog rešenja i postavka rešenja) i arhitekturalna evaluacija (validacija ključnih elemenata: POC, prototip, arhitekturalni spike i sl.).

Up front arhitekturalne odluke donose se i na nivou planiranja release-a., te je iz tog razloga uvedena i kategorija *up front odluke pre razvoja release-a.* Bitan aspekt u razvoju softvera predstavlja aktivnost dokumentovanja i stoga nju reprezentuje posebna podkategorija: *arhitekturalno dokumentovanje*.



Slika 4.1 Ključne identifikovane kategorije
Izvor: Autor

Tabela 4.1 prikazuje jedan deo kodiranih podataka i kodove, koji su proizašli iz procesa analize intervjua. Procesom analize, neki kodovi su postali koncepti na višem nivou, dok su neki grupisani u kategorije. Koncept je obrazac razmišljanja pojedinca ili grupe ispitanika. U navedenom primeru kodiranje je rezultovalo sa preko 40 identifikovanih tema (kodova), koje su potom grupisane u 18 koncepata. Svi koncepti su se odnosili na način odabira arhitekturalnog rešenja, ali su identifikovane razlike između pojedinih koncepata u smislu njihovog fokusa pri odabiru arhitekturalnog rešenja. To je impliciralo uvođenje 4 koncepta višeg nivoa: *arhitekta, postojeća rešenja, testiranje, klijent i biznis*. Ovi koncepti grupisani su u jednu kategoriju: *izbor arhitekturalnog rešenja.*

Kategorija *izbor arhitekturalnog rešenja* predstavljala je podkategoriju kategorije *arhitekturalna sinteza.* Istim postupkom identifikovana je i druga podkategorija *arhitekturalne sinteze*: *postavka arhitekturalnog rešenja.*

Tabela 4.1 Primer kodiranih podataka i kodova

ISPITANIK	PODATAK	KOD
ISP1	...Pa ja prvo krenem od onog u mojoj glavi što je idealizovano, to ne mora da bude tačan i pravi deo kako treba da se uradi, to je recimo po meni idealno da se uradi. Onda pokušam to da vidim kako će se to uglaviti u postojeću arhitekturu. Ako ne može da se uglavi onda moram naći najmanji sadržalac moje ideje i trenutne ideje koja postoji, i da vidim šta treba da promenim kod mog rešenja da bi se to uglavilo u postojeću arhitekturu.	VIZIJA ARHITEKTE
ISP2	...trudimo se da iskoristimo naše znanje, našu ekspertizu, tako da uvek kad rešavamo neki sistem i pravimo arhitekturu pravimo ga u nekoj viziji naših prethodnih projekata, komponenti koje možemo iskoristiti, znanja koja imamo...	ISKUSTVO I PRETHODNO ZNANJE
	...kad radimo za neke krajnje klijente kažemo ovo će rješenje koštati toliko i toliko a radićemo to, ovo će koštati toliko za to. Onda će klijenti izabrati šta im odgovara...	ODGOVORNOST KLIJENTA
ISP3	...lično moje iskustvo i znanje, ništa drugo ne može da utiče na to...	ISKUSTVO I PRETHODNO ZNANJE
ISP4	...nemate tu mnogo opcija, jer za svaku stvar imate tačno te i te opcije, gde sad morate da izmerite.... srećna okolnost je što imate trend, što imate opcije.... recimo mikroservisi - šta to znači, šta to znači za vasu kompaniju, šta to znači za klijenta, koji su benefiti...	TREND OPCIJA
	...identifikovati šta je relevantno za klijenta... pomažete se sa proof of concept-om koji je time boxed... morate da identifikujete ciljeve i morate da identifikujete šta ćete da probate, pa da onda napravite analizu...	PROOF OF CONCEPT
	...arhitektura je skup odgovora na requiremente...bitan je i profil klijenta, pa da li odgovara problemu koji rešavate, na koji način, kako se uklapa u celu priču, i na kraju da li su vam ljudi sposobni da koriste tu tehnologiju.	ZAHTEVI, PROFIL KLIJENTA, PROBLEM I TIM
ISP5	...u principu se sve svodi na neko iskustvo, na iskustvo sa nekim tipovima, nekad je zgodnije slojevito, nekad u distribuirane komponente, to sve zavisi od slučaja do slučaja... testiranje obično obavlja u nekim the risking fazama na samom početku, znači ako ne znamo da li ćemo levo ili desno, onda mi obavimo neki the risking	ISKUSTVO I TESTIRANJE
ISP6	...uvek imamo opcije. Razgovaramo, i na kraju onaj čovek koji će biti odgovoran da isporuči on presudi, šta on misli da je najbolje. I može i ne mora da testira. Onaj koji nosi projekat je odgovoran za to i to je njegova odgovornost.	ODGOVORNOST ARHITEKTE
ISP8	... koristite se sa onim što znate da radite. To je taj koncept koji ste radili u prošlosti, osloniće se i sad na to, na iskustvo.	ISKUSTVO, PRETHODNO ZNANJE
	...više je to kroz komunikacije sa drugim ljudima, jer nismo mi velika grupa ljudi i prosto smo svi vezani i dosta razgovaramo na tu temu. Dosta čitam, i to je de facto stanje. Literatura je dostupna i sve to negde već piše...ako te stvari predlaže neko ko je kredibilan i čije se mišljenje uvažava i neko na koga ste se možda oslanjali u prošlosti, ne testiram ja tu mnogo.	BRAINSTORMING, POSTOJEĆA REŠENJA
ISP9	...proof of concept je skupa stvar, najčešće se nema vremena za to niti resursa. Najčešće se radi više o misaonoj vežbi, research u smislu pročita se šta se pročita...	MISAONO TESTERINJE, LITERATURA
	... čovek ima tehnička iskustva iz nečega ili nema pa pita nekog ko ga ima, i takva rešenja prezentujemo nekoj široj grupi i vidimo šta ta grupa misli i te se stvari uvrste.	ISKUSTVO, BRAINSTORMING
ISP18	...prvo isprobamo na nekom manjem projektu kako će se ponašati...prototipom	PROTOTIP
	...koristimo iskustva koja imamo. Vidimo iskustva drugih firmi koja su radila na sličnim projektima, pogledamo malo na internetu kako se pokazala takva rešenja....	LIČNO ISKUSTVO, ISKUSTVO DRUGIH KOMPANIJA
ISP10	...trudimo se da idemo ka nekim fleksibilnijim rešenjima, koja su više vezana za servise ili mikro servise, kako bi u momentu kada vidimo da nešto nije kako smo planirali, tu bili fleksibilni da reagujemo. I naravno, koristimo slojevitu arhitekturu, da je sve podeljeno na nivoe: backend, frontend, servisi koji su koliko toliko individualni, kako bi se mogli tako i testirati i nezavisno razvijati.	FLEKSIBILNA REŠENJA
	...budućnost razvoja te aplikacije. To umnogome može da menja pristup. Znači dobro razumevanje poslovnog konteksta aplikacije, šta ta aplikacija treba da radi. Koliko korisnika će je koristiti, koje su dalje mogućnosti za razvoj i nadograđivanje i unapređivanje, i od te cele kombinacije zavisi šta će se odabrati, da li će se odabrati neko lako i jednostavno rešenje ili nešto što je kompleksnije i zahtevnije, pogotovo u startu, da bi imalu tu fleksibilnost u daljem razvoju...	BUDUĆNOST APLIKACIJE
	...sa konkretnim zahtevima dolazimo do toga kakva će biti organizacija softvera, da li će biti komponente, da li će biti neki plug-in-i, da li će biti standardna arhitektura kao web portala, web servira ili web sajtova, bukvalno zavisi od samih zahteva...	ZAHTEVI

ISP11	...brainstorming. Prvo rešenje se stvara na osnovu dosadašnjeg iskustva i do sada odrađenih postojećih rešenja u softveru. Na osnovu toga da kažem, postavimo prvi predlog rešenja. Onda u nekom brainstormingu sa ostalim arhitektama diskutujemo koliko je to dobro, sagledavamo sve aspekte na šta to rešenje utiče, šta utiče na to rešenje i onda kroz taj neki iterativni proces to polako menjamo, što nekad može da znači da totalno promenimo inicijalno rešenje, a nekada to bude samo nijansa od ovog prvog.	ISKUSTVO, BRAINSTORMING
	...pre izbora bi dobro bilo da se uradi testiranje ili prof of concept...	PROOF OF CONCEPT
ISP12	...čim znamo šta su zahtevi, sednemo i vidimo koji je najoptimalniji način iz našeg iskustva, koje paterne da koristimo iz arhitekture i kako da organizujemo kod, kako pohranjujemo podatke....	ISKUSTVO
	...krećemo sa jednim rešenjem za koje mislimo da je najbolje, ako nisu performanse dobre, onda ćemo ga menjati. Pa da, kroz taj proces da dobijemo proof of concept. Mi napravimo bukvalno skelet i ako taj skelet može da stoji, i ako to zadovoljava to što mi hoćemo na kraju, to je onda to...	PROOF OF CONCEPT
ISP19	...čim se vidi kakav je problem možemo ga svrstati u neki od već postojećih tipova arhitekture i onda samo detalji se razlikuju...	POSTOJEĆA REŠENJA
	...na iskustvo.	ISKUSTVO
	...većina projekata kada uđemo u branšu je poznata. Samo kad imamo nove, koje ulaze u neko potpuno novo tržište biznisa, onda tu imamo više mogućih arhitektonskih rešenja, pa onda okupimo konzorcijum i dogovorimo se oko rešenja...	BRAINSTORMING
ISP13	...na osnovu nekog predznanja, znači to je to neko iskustvo i znanje koje vam je potrebno da bi ste to mogli da identifikujete.	ISKUSTVO, PRETHODNO ZNANJE
	...na osnovu arhitekture gde možete pratiti koji su to trendovi, sa druge strane proučavanja tih nekih paterna kako se rešavaju neki problemi... oni vam definišu orijentacionu arhitekturu...	TREND OPCIJA
	...i naravno sa druge strane business case kuće...	BUSINESS CASE
ISP14	...ukoliko je nešto novo onda bih definitivno preporučio eksperiment, analizu i više rešenja ne samo jedno...unapred definisane metrike i kriterijume za uspeh.	EKSPERIMENT I ANALIZA NA OSNOVU METRIKA
ISP15	...rekao bih da se gomila aplikacija može podovesti pod isti kalup, pogotovu kad govorim o web aplikacijama. To je neki industrijski standard koji su diktirani od strane frameworka.	TREND OPCIJA
	...ukoliko ima pitanja da li je A ili B varijanta, obično imamo odluku predloga zašto neka varijanta ima prednost. Tipično radimo proof of koncept fazu, gde probamo jedno i drugo time-box rešenje. U odnosu na date parametre pre proof of koncepta odlučujemo kojim putem idemo. Ako nakon proof of koncepta ni jedna nije odgovarajuća, ide se na sledeću.	PROOF OF CONCEPT
ISP16	... postoje određeni kriterijumi po kojima se određuje da li je nešto najbolje. Za neke stvari mogu da se vide performanse, kompleksnost, rizik koji unosi u softver. Sprovodi se analiza u nekoj tabeli gde postoji neki cost-benefit odnos tog rešenja...i tako se vrši njegovo vrednovanje...	COST-BENEFIT ANALIZA
	...dešava se da nekada imam 3 rešenja i potrebna mi je asistencija nekoga, pošto je domen malo veći.	BRAINSTORMING
ISP17	...odredi se odgovorni arhitekta, a odredi se i tim review-era. Neki od njih su formalni, znači njihov input se mora sačekati, a neki su samo informativno uključeni i njihovi input je dobrodošao, ali se ne ne čeka. Kroz takav proces se načini nekoliko draft-ova dok se svi ne usaglase ionda se dođe do toga da je to taj dizajn. Ako ne mogu da se usaglase, onda se skalira do mene i imamo sastanak na kome se eskalacija razrešava, ali to se stvarno retko dešava.	ARCHITECTURAL BOARD/ ULTIMATIVE CHIEFS
ISP20	...radimo neku analizu prvo impacta koji možemo imati na postojeću arhitekturu. Na osnovu tog impacta na postojeću arhitekturu i nekih funkcionalnih zahteva koje imamo, imamo čoveka koji treba da predloži rešenje, naravno uz konsultaciju sa drugim ljudima.	ANALIZA UTICAJA NA POSTOJEĆE REŠENJE
	...mi možda imamo par rešenja ali pros and cons za neko rešenje, sa biznis tačke gledišta utiču na to, ili sam arhitekta često pravi kompromise zarad mogućnosti implementacije nečega...	BIZNIS
	..imamo mali tim ljudi koji radi na tome. Koji prezentuje bordu sistem arhitekti, znači gde ultimative chiefs sistem arhitect donosi approval svega toga.	ARCHITEkTURALNI ODBOR

	...pravilo kojim se ja vodim je sledeće: ukoliko postoje dva moguća rešenja koji imaju sličnu biznis value, treba ići sa onim rešenjem iz koga je najlakše moguće rekaverovati. Znači tom se logikom vodim... odabrati rešenje koje mogu lakše da se vratim na prethodno stanje, da bih mogao da krenem nekim drugim putem ako nije dobro.	REŠENJEM IZ KOGA JE NAJLAKŠE MOGUĆE VRATITI NA PRETHODNO STANJE
	... radimo neki prototip, pokušavamo da izvučemo šta je to što je najosnovnije što možemo da uradimo, čisto da vidimo da li će to arhitektonsko rešenje biti zadovoljavajuće ili ne... To je kod nekih novih rešenja.	PROTOTIP
	Kod nekih postojećih rešenja legacy, pokušavamo prvo da utvrdimo gde smo, znači gde se mi nalazimo, koje su naši indikatori koji pokazuju gde smo i šta smo, koliki je gap između toga i nekog rešenja koje treba. I onda koji su koraci da premostimo taj gap. Naravno to sa sobom vuče neke pare, neke investicije, to je ono što ne može da se radi van biznisa. Znači biznis ultimativno ima finalni say-ing, ali neophodno je da to bude timska odluka u neku ruku.	TIMSKA ODLUKA, KLIJENTOVO ODOBRAVANJE

Za ključne identifikovane kategorije u softveru NVivo kreirani su dokumenti - tzv. beleške, u kojima je opisana suština date kategorije i razlozi za njeno uvođenje. Generisana dokumenta imala su dvostruki cilj. Pre svega, olakšala su diskusiju krajnjih rezultata istraživanja, ali su ujedno stvorila osnovu su za ponovljivost istraživačkog procesa i njegovu proverljivost od strane trećeg lica. U nastavku sledi izvod jedne beleške o kategoriji *postojeća rešenja*:

" Danas za većinu rešenja postoji neki industrijski standard koji se diktira radnim okvirima. Tip problema koji se rešava upućuje na neki od već postojećih tipova arhitekture i arhitekturalni patern, pri čemu je potrebno razraditi samo detalje. Posao softver arhitekte, u kontekstu postavke arhitekturalnog rešenja, je stoga umnogome olakšan, jer treba da izabere opciju koja odgovara tipu problema koji se rešava. Ključno je da softver arhitekta bude upoznat sa mogućim opcijama i da napravi analizu istih spram problema, profila klijenta, razvojnog tima. Cilj analize postojećih opcija je da se utvrde njihovi benefiti i ograničenja. Fleksibilnija rešenja su generalno trend u razvoju arhitekture danas, naročito servisi i mikro servisi, iz razloga što omogućavaju bezbolnije i lakše korekcije ukoliko se uvidi da u fazi planiranja nije nešto razmotreno. Vodilja u izboru rešenja je, dakle, mogućnost lakog povratka na prethodno stanje. U ovom procesu veliki oslonac je i internet, gde se istražuju prednosti i nedostaci postojećih rešenja, kao i iskustva drugih kompanija sa njima".

4.1.3.3 Razvoj instrumenta za drugi krug istraživanja

Druga iteracija istraživanja predstavljala je kvantitativnu komponentu i sprovodila se putem e-ankete. Instrument istraživanja razvijen je na osnovu dobijenih rezultata iz prethodnog kruga istraživanja i uvidom u rezultate sistematskog pregleda literature. Uz većinu pitanja upitnika (26) bila je postavljena četvorostepena skala, po uzoru na Likertovu (1. "nije značajno"; 2. "delimično značajno"; 3. "značajno"; 4. "ekstremno značajno"). Cilj odabira skale sa četiri nivoa bio je da onemogući eksperte da daju neodređene odgovore tj. ocene tipa "nije ni značajno ni neznačajno", već da svoje mišljenje opredele isključivo na jednu stranu.

Inicijalna verzija upitnika napravljana je krajem aprila 2015. godine, nakon čega je poslata petorici eksperata na evaluaciju (prilog 5). Eksperti su ocenili značajnost svakog potencijalnog pitanja i dali propratne komentare i sugestije za nekolicinu pitanja. Pitanja koja su imala indeks sadržajne valjanosti <0.8, eliminisana su iz upitnika, a nekolicina pitanja je bila preformulisana, zbog sugestija evaluatora da nisu dovoljno jasna. Indeks sadržajne valjanosti (Polit & Beck, 2006) upitnika iznosio je 0.83 (prilog 6). Revidirani upitnik poslat je ekspertima na ponovnu procenu, međutim, zahteva za dodatnim izmena nije bilo.

Konačan broj pitanja upitnika iznosio je 30 (prilog), sa ukupno 115 stavki, tj. varijabli. Ukupan broj pitanja za koja se sprovodila ocena značajnosti (na skali od 1- 4) od strane eksperata je 94. Četiri pitanja bila su predstavljena čeklistama, uz mogućnost višestrukog izbora.

Instrument drugog kruga istraživanja bio je završen početkom maja 2015. godine., nakon čega je upitnik napravljen na Google Drive-u, pomoću alata Google Forms. Ekspertima je, uz link na upitnik, mejlom poslato i pozivno pismo za učešće u drugom krugu istraživanja (prilog 7), kao i propratno pismo u kojem su dobili instrukcije o načinu popunjavanja upitnika i svrsi drugog kruga istraživanja (prilog 8).

4.1.3.4 Prikupljanje, priprema i analiza podataka drugog kruga istraživanja

Podaci su prikupljeni sredinom maja 2015. godine, a stopa odgovora u drugom krugu istraživanja iznosila je 100%. Prikupljeni sirovi podaci kolektirani su u Microsoft Excel tabelu, nakon čega je usledila njihova priprema u obliku koji je pogodan za kvantitativnu analizu u softveru SPSS. Pored emirijskog uzorkovanja, sprovedeno je bootstrap uzorkavanje sa 1000 replikacija, u cilju obezbeđivanja stabilnosti dobijenih rezultata.

Prema gore opisanim koracima Delfi metode, otpočinjanje trećeg kruga istraživanja zahtevalo je obavljanje određenih kvantitativnih analiza, kako bi se dobili rezultati na osnovu kojih je moguće utvrditi set pitanja i stavki za koje nije postignut potrebni nivo saglasnosti eksperata od 70%.

U nastavku je, kroz konkretan primer, opisana procedura kojom je izvršena analiza dobijenih kvantitativnih rezultata (deo rezultata kvantititativnih analiza podataka dat je u prilogu 9). Analiza kvantitativnih rezultata imala je za cilj razvoj instrumenta istraživanja za treći krug. Primer se odnosi na 29. pitanje iz upitnika i njegovu 1. stavku:

29.1: Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa:

1. Identifikovanje ključnih stejkholdera sistema.

U tabeli 4.2 (koja je napravljena u softveru SPSS) prikazana je učestalost (vrednosno i procentno) u kolonama Frequency i Percent, s obzirom na ocene značajnosti za datu stavku 29. pitanja. Redovi u tabeli 4.2 predstavljaju varijable četvorostepene skale ocenjivanja iz upitnika (1. "nije značajno"; 2. "delimično značajno"; 3."značajno"; 4. "ekstremno značajno"). Dok tabela 4.3 prikazuje donje i gornje granice izračunatih procenata ocena, uz interval poverenja od 95%, koji je dobijen bootstrap uzorkovanjem sa 1000 replikacija. Tako je npr. u tabeli 4.3 procenat ocene 3 između 40 i 80 uz izračunati interval poverenja od 95%.

Tabela 4.2 Značajnost varijable identifikovanje ključnih stejkholdera sistema

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent	
						Bias	Std. Error
Valid	1,00	2	10,0	10,0	10,0	,0	6,6
	2,00	3	15,0	15,0	25,0	,1	8,4
	3,00	12	60,0	60,0	85,0	,2	11,1
	4,00	3	15,0	15,0	100,0	-,3	7,8
	Total	20	100,0	100,0		,0	,0

Tabela 4.3 Donje i gornje granice izračunatih procenata ocena, uz interval poverenja od 95%

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	1,00	,0	25,0
	2,00	,0	35,0
	3,00	40,0	80,0
	4,00	,0	30,0
	Total	100,0	100,0

Na osnovu prikazanih rezultata, napravljena je sumarna tabela za dihotomizovane podatke (za svih 115 stavki iz upitnika), koja prikazuje njihovu značajnost u agilnim procesima razvoja. Značajnost svake stavke prikazana je putem pokazatelja proporcija ocenjivača. Pokazatelj je dobijen primenom navedene formule, kojom se sabiraju procentualne vrednosti ocena za varijable 3 i 4, iz kolone Percent. Date ocene predstavljaju "značajnu" i "ekstremno značajnu" vrednost sa skale iz upitnika.

Na primeru odabrane stavke (tabela 4.2), proporcija ocenjivača dobijena je sabiranjem procentualne vrednosti za ocenu 3, koja je iznosila 60% i procentualne vrednosti za ocenu 4, koja je iznosila 15%. Tako dobijen procenat od 75%, nakon deljenja sa 100, predstavljao je proporciju ocenjivača koji datu stavku upitnika ocenjuju kao značajnu.

Formula:

Proporcija ocenjivača koji praksu ocenjuju kao značajnu = (kolona Percent [za red 3] + kolona Percent [za red 4]) / 100

Druga vrsta analize nad dobijenim kvantitativnim rezultatima, podrazumevala je utvrđivanje "grupnog" odgovora za svako pitanje (i sve njegove stavke, ako ih ima) koje su eksperti ocenjivali. U tu svrhu korišćena je izračunata vrednost medijane za svaku varijablu (pitanje/stavku) iz upitnika.

U tabeli 4.4 dat je primer sprovedene statističke analize (u SPSS softveru) za 29. pitanje iz upitnika, koje sadrži 31 stavku (varijablu). Vrednost medijane, u koloni Statistic, tumačena je na sledeći način: npr. ako je njena vrednost 3.000 za neku varijablu, to znači da je grupni odgovor o značajnosti neke izjave: "značajan". U slučaju kada je medijana 4.000 onda je grupni odgovor za neku izjavu iz upitnika: "ekstremno značajna". Ukoliko su vrednosti medijane predstavljale granične vrednosti npr. 3.5000, grupni odgovor je tada imao je dvostruku vrednost, npr. "značajna ili "ekstremno značajna".

Tabela 4.4 Medijane i intervali poverenja za medijane stavki skale za procenu značajnosti eksplicitnih arhitekturalnih aktivnosti

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Median	P29.01	3,0000	-,0115	,0929	3,0000	3,0000
	P29.02	3,0000	,3185	,4219	3,0000	4,0000
	P29.03	3,5000	,0065	,4466	3,0000	4,0000
	P29.04	3,5000	-,0025	,4544	3,0000	4,0000
	P29.05	3,0000	,1705	,3381	3,0000	4,0000
	P29.06	4,0000	-,3460	,4324	3,0000	4,0000
	P29.07	3,0000	,0855	,2569	3,0000	4,0000
	P29.08	3,0000	,0875	,2533	3,0000	4,0000
	P29.09	3,0000	,0075	,0721	3,0000	3,0000
	P29.10	4,00	-,09	,26	3,00	4,00
	P29.11	3,0000	-,0015	,0474	3,0000	3,0000
	P29.12	3,0000	-,3385	,4334	2,0000	3,0000
	P29.13	3,0000	-,1740	,3374	2,0000	3,0000
	P29.14	3,0000	,0000	,0000	3,0000	3,0000
	P29.15	3,0000	,0010	,0223	3,0000	3,0000
	P29.16	2,5000	-,0080	,4490	2,0000	3,0000
	P29.17	3,0000	-,0270	,1442	2,5000	3,0000
	P29.18	3,0000	-,3315	,4178	2,0000	3,0000
	P29.19	3,0000	,0265	,1535	3,0000	3,5000
	P29.20	3,00	-,01	,08	3,00	3,00
	P29.21	3,0000	-,0090	,0996	3,0000	3,0000
	P29.22	2,0000	,1820	,3486	2,0000	3,0000
	P29.23	2,0000	,1430	,4034	1,5000	3,0000
	P29.24	2,0000	,1545	,4136	1,5000	3,0000
	P29.25	3,0000	-,0275	,1747	2,5000	3,0000
	P29.26	3,0000	-,0375	,1654	2,5000	3,0000
	P29.27	3,0000	,0805	,2384	3,0000	4,0000
	P29.28	3,0000	-,0020	,0447	3,0000	3,0000
	P29.29	3,0000	,3260	,4236	3,0000	4,0000
	P29.30	4,00	-,20	,37	3,00	4,00
	P29.31	3,0000	,0095	,0785	3,0000	3,0000

Na ovaj način je za svaku izjavu za koju je prethodno utvrđeno da nema neophodni nivo saglasnosti ekperata (tabela 4.4) utvrđena medijana, odnosno vrednost grupnog odgovora. Ovim su stečeni uslovi za razvoj instrumenta istraživanja za treći krug.

4.1.3.5 Razvoj instrumenta za treći krug istraživanja

Upitnik u trećem krugu istraživanja činila su samo ona pitanja iz upitnika prethodnog kruga, za koja nije postignut neophodni nivo saglasnosti od 70%. Analizom dobijenih rezultata iz originalnog upitnika je od 30 pitanja sa ukupno 115 stavki, eliminisano 19 pitanja sa ukupno 87 stavki, jer su imali zahtevani nivo saglasnosti eksperata.

Treći krug istraživanja uključivao je svega 11 pitanja, odnosno 28 stavki (varijabli) od prvobitnih 115. U tabeli 4.5 prikazana su sva pitanja i njihove stavke koje su sačinjavale upitnik za treći krug istraživanja:

Tabela 4.5 Pitanja i stavke za III krug istraživanja

Rbr. pitanja	Rbr. stavke	Pitanje iz upitnika II kruga istraživanja	NIVO SAGLASNOSTI IZRAŽEN U PROCENTIMA	GRUPNI ODGOVOR
11.		Da li je značajno obezbediti vidljivost arhitekturalno značajnih zahteva i testova prihvatljivosti kroz postavku arhitekturalnih zadataka na listi zadataka iteracije ili Kanban tabli, kroz ceo proces razvoja?	65	3
16.		Da li je značajno sprovoditi estimaciju veličine tehničke korisničke priče i brzine njenog razvoja?	65	3
23.		Ocenite značaj strategije koja momenat donošenja i sprovođenja arhitekturalnih odluka zasniva na proceni relativnih troškova, sa jedne strane usled kašnjenja implementacije i troškova usled eventualne dorade ili redizajna rešenja u fazi implementacije.	65	3
26.	1.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju među organizacijama koje su uključene u razvoj, produkciju, operativne aktivnosti i održavanje sistema	65	3
26.	3.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za dokumentovanje osnovnih postavki sistema, njegove namene i okruženja	65	3
26.	16.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži pripremu dokumentacije za akviziciju	65	3
15.		Koliko je značajno pre implementacije release-a utvrditi zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti?	60	3

21.	3.	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Test case-ovi	60	3
26.	6.	Koliko je značajna svaka od sledećih namena softverske arhitekture u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži review, analizu i evaluaciju sistema	60	3
26.	15.	Koliko je značajna svaka od sledećih namena softverske arhitekture u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži planiranje sistema i budžetske aktivnosti	60	3
27.	4.	Ocenite sledeće benefite od inicijalne postavke arhitekture na početku agilnog projekta: Poboljšava svesnost razvojnog tima o vezi sistema i organizacije koja ga implementira (njenih ciljeva, infrastrukture, strategije i dr.)	60	3
27.	7.	Ocenite sledeće benefite od inicijalne postavke arhitekture na početku agilnog projekta: Poboljšava komunikaciju, prezentujući stejkholderima šta će se graditi i na koji način	60	3
29.	13.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema	60	3
27.	3.	Ocenite sledeće benefite od inicijalne postavke arhitekture na početku agilnog projekta: Izbegavanje tehničkog duga	55	3
28.		Koliko je značajno da ceo arhitekturalni proces tokom razvoja bude praćen nekim od alata (Jira ili sl.)?	55	3
29.	12.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Generisanje top level dokumentacije	55	3
29.	18.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem interreaguje tokom release-a	55	3
29.	16.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji	50	2.5
21.	6.	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Statička analiza koda u cilju analize struktura i pravila kodiranja	45	2
22.		Da li smatrate korisnom strategiju Test Driven Development sa fokusom na nefunkcionalne zahteve?	45	2
12.		Da li ceo projekat ugrožen usled razmatranja arhitekture na nivou samo jedne iteracije?	40	2
29.	22.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje detaljnog dizajna svakog modula	40	2
29.	23.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Dokumentovanje detaljnog dizajna	40	2

29.	24.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Razmatranje detaljnog dizajna i korekcije	40	2
26.	5.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju sa stejkholderima sistema (klijentima, korisnicima, integratorima i dr.)	35	2
26.	11.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Sredstvo razvoja i održavanja dokumentacije	35	2
26.	14.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju između klijenata, korisnika i programera, kao deo dogovorenog ugovora	35	2
26.	17.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Kao input za izbor generatora sistema i alata za analizu	35	2

Struktura upitnika za III krug istraživanja predstavljena je u tabeli 4.6. Prva kolona sadržavala je tekst pitanja/stavke (iz tabele 4.5). Druga kolona sadržavala je ekspertov odgovor iz prethodnog kruga istraživanja ("nije značajno", "delimično značajno", "značajno", "ekstremno značajno"), a treća je sadržavala grupni odgovor za svaku stavku ponaosob.

Način izračunavanja grupnog odgovora opisan je u prethodnom delu rada (deo rezultata dobijenih kvantitativnom analizom podataka dat je u prilogu 9). Četvrta kolona je bila prazna i u njoj je bio predviđen ekspertov odgovor u trećem krugu istraživanja. Peta kolona je omogućavala unos argumenata i komentara eksperata.

Tabela 4.6 Struktura upitnika za III iteraciju istraživanja

Pitanja iz II kruga bez konsenzusa od 70%	Vaš odgovor u prethodnom krugu istraživanja	Grupni odgovor (izračunata medijana na osnovu odgovora 20 eksperata)	Vaš odgovor u ovom krugu istraživanja	Obrazloženje odgovora

U Excel-u je napravljeno ukupno 20 upitnika (za svakog eksperta ponaosob) prikazane strukture (tabela 4.6). Isti su potom, sredinom juna 2015. godine, mejlom poslati ekspertima, uz pozivno (prilog 10) i propratno pismo, u kojem su im date detaljne instrukcije oko načina popunjavanja istog (prilog 11).

4.1.3.6 Kolektiranje i priprema podataka III kruga istraživanja

Svi odgovori eksperata iz trećeg kruga istraživanja prikupljeni su krajem juna 2015. godine, putem mejla. Isti su potom snimljeni u jedinstvenu Excel tabelu, nakon čega su pripremljeni za kvantitativnu obradu u SPSS-u.

4.1.4 Metode analize podataka

Poglavlje sadrži kratak opis i svrhu metoda korišćenih za analizu empirijskih podataka, po iteracijama istraživanja. Obrasci korišćenih procedura navedeni su u prilogu doktorske disertacije.

4.1.4.1 Metode analize podataka dobijenih u prvom krugu

Podaci prvog kruga istraživanja predstavljali su kvalitativnu komponentu, te je za njihovu analizu bio korišćen metod tematske analize sadržaja. Nevedeni metod zasnovan je na subjektivnoj interpretaciji tekstualnih podataka, primenom sistematskog procesa kodiranja i identifikovanja tema, koje ilustruju spektar značenja istraživanog fenomena, a ne statistički značajne pojave iz analiziranih tekstova (Zhang & Wildemuth, 2009; Hsieh & Shannon, 2005).

Primena ove metode podrazumevala je realizaciju sledećih koraka (Taylor-Powell & Renner, 2003; Zhang & Wildemuth, 2009):

1. Upoznavanje sa podacima - putem višekratnog iščitavanja tekstova i beleženja sopstvenih impresija tokom čitanja.

2. Odabir fokusa analize - podrazumevao je sprovođenje analize u smeru postavljenih istraživačkih ciljeva, korišćenjem dva pristupa. Jedan pristup značio je organizovanje prikupljenih podataka prema pitanjima iz intervjua, u cilju identifikovanja konzistentnosti i razlika u odgovorima ispitanika, kao i veza i odnosa između pojedinih pitanja. Drugi pristup podrazumevao je organizovanje i analizu podataka prema ispitanicima.

3. Kategorizacija informacija - podrazumevala je kodiranje prikupljenih podataka i to kroz dve faze. Prva faza se odnosila na identifikovanje tema koje oslikavaju ideje, koncepte, ponašanja, terminologiju ili korišćene fraze u odgovorima ispitanika. Druga faza podrazumevala je njihovo organizovanje u koherentne kategorije, koje predstavljaju metafore značenja pojedinih delova teksta tj. odgovora ispitanika. Deo kategorija je bio definisan unapred, na osnovu referentne literature (deduktivan pristup), a nekolicina

kategorija identifikovana je u toku samog procesa analize. Sve kategorije bile su u tekstualnom obliku.

4. Testiranje šeme kodiranja - sprovedeno je na uzorku podataka, u cilju testiranja jasnoće i konzistentnosti definisanih kategorija.

5. Kodiranje celog teksta - podrazumevalo je analizu svih odgovora, na sva pitanja, svakog ispitanika ponaosob, uz označavanje delova teksta (identifikovanih tema) i njihovo povezivanje sa jednom ili više definisanih kategorija. Tokom ovog procesa određene teme su, zbog njihove sličnosti i značenja, grupisane u odgovarajuće podkategorije.

6. Identifikovanje obrazaca/konekcija unutar pojedinih kategorija - podrazumevalo je identifikovanje sličnosti i razlika u odgovorima pojedinaca, koji su bili grupisani u okviru date kategorije. Analiza je rezultovala dokumentima u kojima je opisivana suština svake kategorije.

7. Međukategorijska analiza - imala je za cilj identifikovanje superkategorija, putem kombinovanja više različitih kategorija, ali međusobno povezanih na određen način. Analizom je sagledavana i značajnost pojedinih kategorija, prebrojavanjem koliko puta se neka tema pojavljuje, kao i prebrojavanje jedinstvenih odgovora za neku temu. I konačno, analiza je omogućila i identifikovanje kauzaliteta između kategorija. Odnosno, da li pojavljivanje neke teme uvek, ili gotovo uvek, podrazumeva pojavljivanje i neke druge teme.

8. Interpretacija podataka i izveštavanje o rezultatima - podrazumevalo je retrospektivu sprovedene analize tj. deskriptivan opis i tumačenja dobijenih nalaza, uz uključivanje navoda ispitanika.

Tematskoj analizi sadržaja, prethodile su neke početne analize u softveru NVivo. Jedna je podrazumevala izračunavanje učestalosti javljanja reči u transkriptima, pri čemu je izgenerisana lista od 200 najčešće korišćenih reči. Takođe, u početnim analizama podataka primenjena je i tehnika ključne reči u kontekstu (Key Word in Context, u nastavku KWIC), koja je olakšala višekratno iščitavanje, razumevanje i kodiranje tekstova, prikazujući automatski markirane delova teksta, u odnosu na zadate ključne reči (Manning & Schütze, 1999). Ove analize olakšale su proces definisanja kategorija i sam proces tematske analize sadržaja.

4.1.4.2 Metode analize podataka dobijenih u drugom krugu

U nastavku će biti navedene procedure koje su korišćene za kvantitativnu analizu podataka, dobijenih Delfi tehnikom u drugom krugu. Obrasci po kojima su se sledeće procedure sprovodile nalaze se u prilogu (prilog 12):

- Izračunavanje mera centralne tendencije/lokacije (aritmetička sredina/medijana) -korištene su u svrhu izračunavanja grupnog odgovora panela eksperata.
- Izračunavanje mera varijabilnosti (standardna devijacija/interkvartilni raspon) - korištene u svrhu izračunavanja grupnog odgovora panela eksperata.
- Efron-ovo ponavljano uzorkovanje (bootstrapping) - korišćeno je u cilju obezbeđivanja stabilnosti dobijenih rezultata empirijskog istraživanja. Tačnije, rezultati na emirijskom uzorku od 20 eksperata, imali su veću statističku značajnost u smislu donošenja zaključaka koji važe za celu populaciju, usled softverskog bootstrap uzorkavanja sa 1000 replikacija.
- Cohen-ov kappa koeficijent, korišćen je za ocenjivanje saglasnosti ocenjivača tj. za sagledavanje stepena njihovog međusobnog slaganja. Tumačenje dobijenih vrednosti kappa koeficijenta, sprovodilo se prema skali koju su razvili (Landis & Koch, 1977) i koja je prikazana u tabeli 4.7.

4.1.4.3 Metode analize podataka dobijenih u trećem krugu

U nastavku će biti navedene procedure korišćene za kvantitativnu analizu podataka, dobijenih Delfi tehnikom u trećem krugu. Obrasci po kojima su se sledeće procedure sprovodile nalaze se u prilogu (prilog 12):

- Izračunavanje mera centralne tendencije/lokacije (aritmetička sredina/medijana) - korištene su u svrhu izračunavanja grupnog odgovora panela eksperata.
- Izračunavanje mera varijabilnosti (standardna devijacija/interkvartilni raspon) - korištene su u svrhu izračunavanja grupnog odgovora panela eksperata.
- Efron-ovo ponavljano uzorkovanje (bootstrapping) - podrazumevalo je softversko, automatsko uzorkovanje sa 1000 replikacija, u cilju obezbeđivanja veće stabilnosti rezultata istraživanja i izvođenje zaključaka koji važe za celu populaciju.
- Cohen-ov kappa koeficjent, korišćen je za ocenjivanje saglasnosti ocenjivača tj. za sagledavanje stepena njihovog međusobnog slaganja. Tumačenje dobijenih vrednosti kappa koeficijenta, sprovodilo se prema skali koju su razvili (Landis & Koch, 1977) i koja je prikazana u narednoj tabeli 4.7:

Tabela 4.7 Mere saglasnosti za kategoričke podatke

Kappa Statistic	Nivo saglasnosti
< 0.00	slab
0.00 – 0.20	mali
0.21 – 0.40	korektan
0.41 – 0.60	umeren
0.61 – 0.80	znatan
0.81 – 1.00	gotovo savršen

- Pokazatelj individualne stabilnosti između dve uzastopne Delfi iteracije (Chaffin-Talley-ev indeks individualne stabilnosti) - korišćen je za identifikovanje ustaljenosti odgovora ispitanika od drugog do trećeg kruga istraživanja. Pored kriterijuma postignutog nivoa saglasnosti eksperata od 70%, predstavljao je dodatni kriterijum za donošenje odluke o obustavljanju iteracija istraživanja.
- McNemar-ov test značajnosti promene i McNemar-Bowker-ov test, korišćeni su sa svrhom identifikovanja ispitanika kod kojih je postojala statistički značajna promena u mišljenju, iz drugog u treći krug istraživanja.

4.2 Rezultati sprovedenog istraživanja po iteracijama istraživanja

Poglavljem se daje prikaz rezultata po iteracijama istraživanja, kako bi se sagledao celokupan tok procesa istraživanja i tumačili rezultati koji su predstavljali neophodan uslov za otpočinjanje narednog kruga istraživanja. Takođe, u fokusu su i rezultati na osnovu kojih je donešena odluka o obustavljanju iteracija istraživanja. Poglavljem se, dakle, opisuju samo oni rezultati koji neće biti interpretirani u 5. poglavlju doktorske diseratcije, u kontekstu davanja odgovora na postavljena istraživačka pitanja.

4.2.1 Rezultati prvog kruga istraživanja

Podaci o učesnicima istraživanja predstavljaju značajan elemenat u kontekstu diskusije dobijenih rezultata. Kvalitativnom analizom intervjua utvrđeno je da svi ispitanici imaju iskustva i u tradicionalnom i u agilnom razvoju softvera. Svi ispitanici koriste neke modifikovane oblike Scrum procesa razvoja. Nekolicina ispitanika je navela da uz Scrum koristi i Lean koncept, isključivo u kontekstu Kanban table (ISP6, ISP10, ISP13, ISP14, ISP15). Pet ispitanika, koji pripadaju istoj kompaniji, naveli su da trenutno izučavaju SAF-e agilni frejmvork, sa ciljem implementiranja nekih njegovih delova u kompaniji (ISP9, ISP11, ISP16, ISP17, ISP20). Samo jedan ispitanik naveo je da u kompaniji, pored Scrum-a, koriste i prakse XP agilnog procesa razvoja (ISP14).

Svi ispitanici imaju iskustva u razvoju poslovnih softverskih rešenja i dugo su bili, a većina i trenutno jeste, na poziciji softver arhitekte (tabela 4.8). Izuzetak predstavlja ispitanik ISP20,

jer nikada nije radio kao arhitekta, ali je njegova pozicija trenera za agilan razvoj u kompaniji tesno vezana za samu problematiku istraživanja, te je iz tog razloga i odabran za eksperta u istraživanju. Svi učesnici istraživanja su bili muškog pola i starosti u rasponu od 30-50 godina.

Tabela 4.8 Godine iskustva ispitanika i njihova trenutna uloga na projektima

Ekspert	Godine iskustva	Trenutna pozicija na projektima
ISP1	6	softver arhitekta
ISP2	18	projektni menadžer softver arhitekta
ISP3	15	projektni menadžer softver arhitekta
ISP4	15	softver arhitekta
ISP5	16	projektni menadžer softver arhitekta
ISP6	15	menadžer proizvoda
ISP7	11	softver arhitekta
ISP8	10	softver arhitekta
ISP9	10	menadžer proizvoda
ISP10	10	šef pružanja usluga
ISP 11	8	softver arhitekta
ISP 12	15	softver arhitekta
ISP 13	15	menadžer proizvoda
ISP 14	11	IT konsultant
ISP 15	15	menadžer isporuke
ISP 16	8	sistem arhitekta direktor arhitekture
ISP 17	15	sistem arhitekta
ISP 18	10	softver arhitekta
ISP 19	15	vlasnik proizvoda softver arhitekta
ISP 20	19	trener za agilan razvoj

Identifikovane kategorije podataka

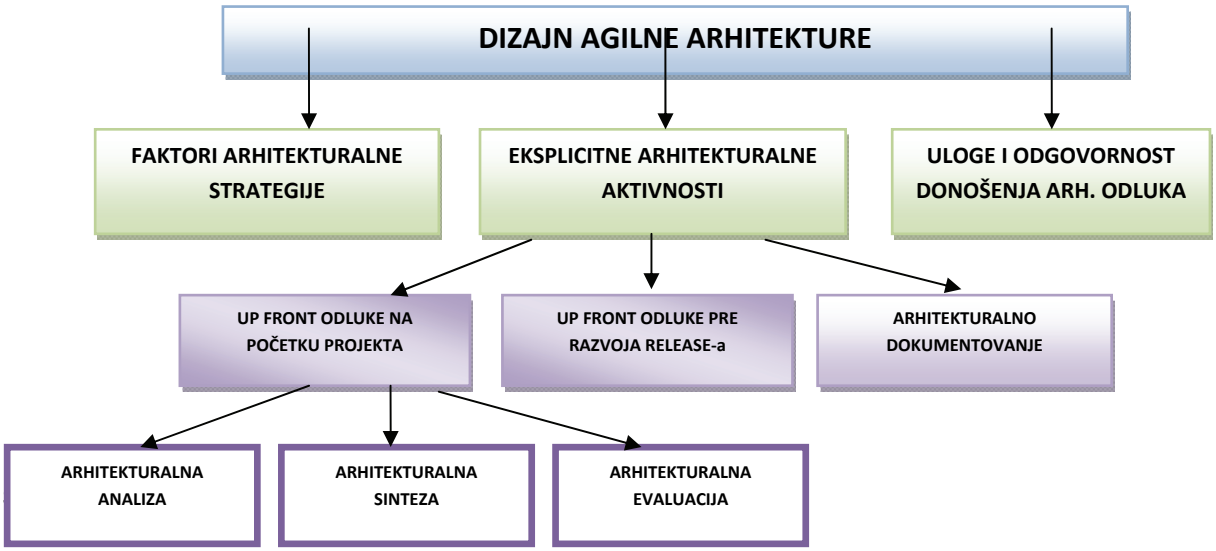
Slika 4.2 prikazuje hijerarhiju ključnih kategorija, korišćenih u kvalitativnoj analizi podataka. Glavna kategorija prezentuje proces razvoja agilne arhitekture od strane agilnih timova u praksi, polazeći od toga ko donosi arhitekturalne odluke, kao i na koji način utvrđuju koliko up front eksplicitnih arhitekturalnih odluka je potrebno doneti. Glavna identifikovana kategorija nosi naziv: *dizajn agilne arhitekture,* a njene tri podkategorije: *faktori arhitekturalne strategije, eksplicitne arhitekturalne aktivnosti i uloge i odgovornost donošenja arhitekturalnih odluka.*

Podkategorija *faktori arhitekturalne strategije* predstavlja empirijski identifikovane faktore, koje agilni timovi razmatraju prilikom utvrđivanja potrebnog obima up front arhitekturalnih odluka i aktivnosti na projektu.

Podkategorija *uloge i odgovornost donošenja arhitekturalnih odluka* predstavlja uloge koje su odgovorne za donošenje arhitekturalnih odluka, kao i njihove odgovornost prilikom razvoja arhitekture u agilnim procesima.

Podkategorija *eksplicitne arhitekturalne aktivnosti* sadrži sve eksplicitne arhitekturalne aktivnosti koje se donose up front, na početku projekta i koje su kategorizovane prema tradicionalnim fazama razvoja softverske arhitekture: *arhitekturalna analiza* (razumevanje problema, konteksta, poslovne arhitekture, rizika), *arhitekturalna sinteza* (izbor arhitekturalnog rešenja i postavka rešenja) i *arhitekturalna evaluacija* (validacija ključnih elemenata: POC, prototip, arhitekturalni spike i sl.).

Up front arhitekturalne odluke donose se i na nivou planiranja release-a, te je iz tog razloga uvedena i kategorija *up front odluke pre razvoja release-a.* Podkategorija *arhitekturalno dokumentovanje* predstavlja bitan aspekt razvoja softvera i važnu eksplicitnu arhitekturalnu aktivnost, koja mora sa merom biti inkorporirana u agilne procese.



Slika 4.2 Ključne identifikovane kategorije
Izvor: Autor

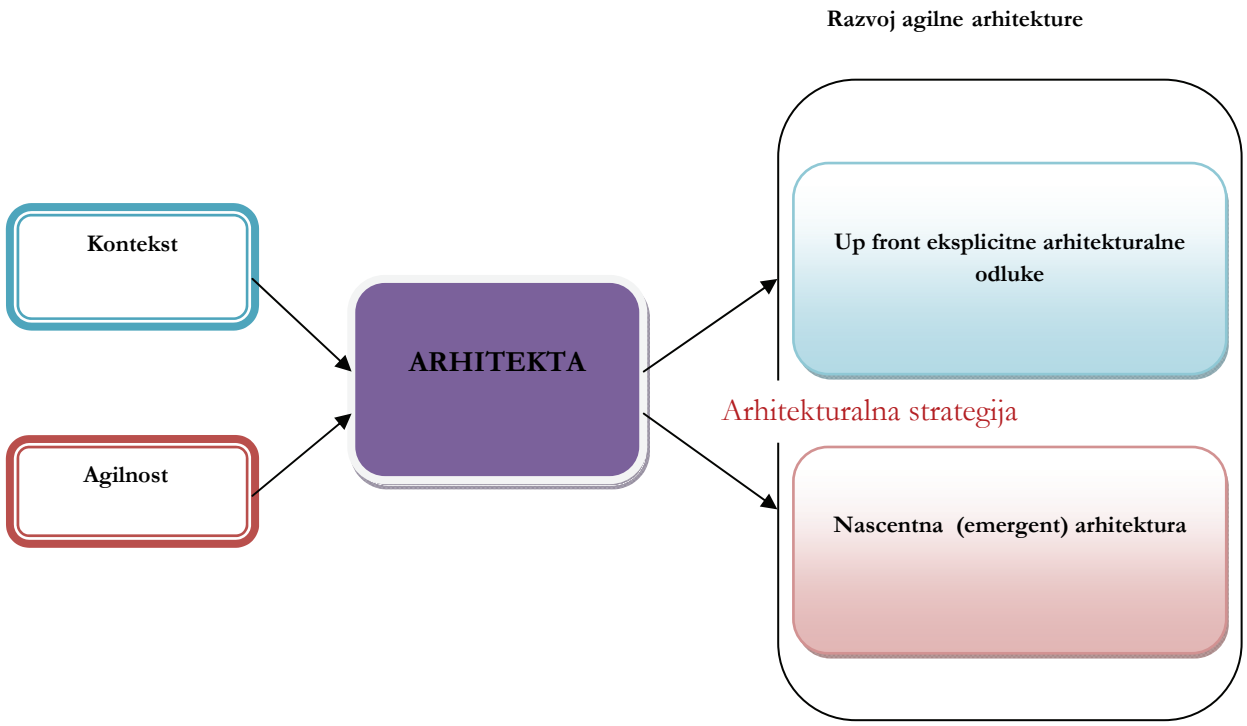
4.2.1.1 Kategorija faktori arhitekturalne strategije

Rezultati kvalitativne analize podataka pokazuju da agilni timovi u razvoju kompleksnih poslovnih softverskih rešenja, ne koriste ni jednu od dve ekstremne strategije razvoja softverske arhitekture: BDUF vs. nascentna arhitektura. Tačnije, agilni timovi primenjuju takvu strategiju razvoja softverske arhitekture kompleksnih sistema, koja je negde između ta

dva ekstrema. Agilni timovi donose up front eksplicitne arhitekturalne odluke, na svim kompleksnim razvojnim projektima, ali je obim tih odluka varirajući, u zavisnosti od konteksta sistema koji se razvija. Analizom intervjua, identifikovani su faktori koje agilni timovi razmatraju, da bi utvrdili koliko je up front eksplicitnih arhitekturalnih odluka neophodno doneti na nekom projektu. Agilna arhitektura, za razliku od tradicionalne, mora podržavati agilnost tima, što implicira da mora biti jednostavna za modifikovanje i prilagođavanje promenama:

ISP1: "...presudno je da se napravi takva arhitektura koja može da podrži veliku količinu promena zahteva, samim tim veliku količinu promena softvera, bez da se menja taj neki kostur arhitekture. To znači manje refaktorisanja, manji rizik, manje bagova, brži odziv, brži razvoj, brže će se napraviti neki user story i odmah će se pokazati klijentu i samim tim će klijent biti srećniji pošto će odmah videti taj rezultat promene posle prvog sprinta, a neće videti rezultat posle četvrtog sprinta, zato što smo svašta morali refaktorisati kako bi zadovoljili njegov zahtev..."

Slika 4.3 prikazuje ključne elemente za razvoj agilne arhitekture u praksi. Naime agilni timovi, na čelu sa softver arhitektom, imajući na umu kontekst i agilnost projekta, eksperimentišu učeći iz sopstvenih grešaka, kako bi iznašli adekvatnu strategiju razvoja agilne arhitekture.



Slika 4.3 Razvoj agilne arhitekture u praksi
Izvor: Autor

Induktivnim pristupom kodiranja identifikovana je kategorija *faktori arhitekturalne strategije.* Kategorija uključuje brojne koncepte, koji su sublimirani u četiri podkategorije:

A. Kategorija nivo kompleksnosti: problema, domena problema, zahteva i arhitekturalnog rešenja.
B. Kategorija zahtevi: nepotpuni / promenljivi.
C. Kategorija tim: iskustvo, komunikacija, znanje, brojnost i dr.
D. Kategorija stejkholderi**:** agilnost, kvalitet komunikacije, znanje i dr.

Podkategorije predstavljaju empirijski identifikovane faktore koje agilni timovi razmatraju u praksi, prilikom utvrđivanja koliko up front arhitekturalnih odluka je optimalno na projektu. Dobijeni rezultati će se narednim tekstom diskutovati i kroz prizmu postojećih nalaza u literaturi (Abrahamsson, Babar, Kruchten, 2010; Boehm, Turner, 2003; Cockburn, 2007).

A. Kategorija nivo kompleksnosti

U literaturi se u za pojam kompleksnosti vezuju tri atributa: "broj stvari u sistemu", „broj različitih stvari u sistemu" i „broj veza između tih stvari" (Kruchten, 2012). Agilni timovi u praksi, za kategoriju *nivo kompleksnosti* vezuju sledeće koncepte: kompleksnost problema koji se rešava, kompleksnost domena problema, zahteva i samog arhitekturalnog rešenja:

ISP16: "...kompleksnost u smislu konkretne oblasti, domena problema, pogotovo ako je taj domen ogroman i gde jedan čovek ne može da sagleda sve i gde je potreban određen brain storming i gde je potrebna kooperacija između ljudi koji su eksperti u određenoj oblasti. Mi imamo tim od 5 sistem arhitekti. Niko od nas nije ekspert za sve, tako da ako imamo neki kompleksniji problem moramo zajedno da radimo na njemu i to produžava fazu planiranja...".
ISP20: "...kompleksnost rešenja je takođe bitno sagledati. Ako je proizvod koji ima mission safety critical... to je proizvod koji može, ukoliko dolazi do problema sa performansama, da ima i fatalne posledice. Ako mi se na telefonu pojavi poruka error, pa neki broj, ja ću onda da se iznerviram i to je to. Ali ako se desi u softveru koji kontorliše prenos struje u nekom gradu, pa dođe do ispada sistema, gde 100 hiljada ljudi ostane bez struje i ne mogu brzo da reagujem, onda su tu konsekvence žestoke...".
ISP20: "...potrebno je sagledati prvo kompleksnost rešenja, kompleksnost zahteva i kompleksnost domena u kome se nalazimo. To definiše koliko zaista je potrebno uložiti eforta u toj nekoj up front analizi zahteva...".

Veličinu projekta ispitanici nisu doveli u direktnu vezi sa kompleksnošću, što je u suprotnosti sa nalazima u literaturi (Abrahamsson, Babar, Kruchten, 2010; Boehm & Turner, 2003; Cockburn, 2007):

ISP12: "...projekat može biti vremenski velik, ili može podrazumevati veći broj ljudi angažovanih na njemu. Ne znači ako je projekat veliki, da mora i toliko da se planira. Kompleksan problem mora dobro da se

isplanira, dok veliki projekat, može samo da zahteva dosta fizičkog rada.... u smislu kucanja koda i razvoja...".
IP10: "...arhitekturalna aplikacija može da bude relativno jednostavna, ali njena implementacija može da bude kompleksna, pa je onda više naglasak na samom planiranje realizacije, odnosno implementacije.Tako da nema direktne zavisnosti između veličine projekta i njegove kompleksnosti...".

Veličina projekta se dovodi u vezu sa izborom servisnih i mikro servisnih arhitekturalnih rešenja, čime se na taj način lakše upravlja njegovom veličinom:

ISP6: "...arhitektura koja je servisno orijantisana, zasnovana na mikro servisima ili je modularna omogućuje da softver bude menadžibilan. Radili smo na ogromnom softveru gde je development tim brojao 40-50 ljudi, koji je bio napravljen kao monolitna aplikacija i to je bio veliki promašaj...".

Kompleksnost zahteva je elemenat koji utiče na veći obim up front arhitekturalnih odluka što arhitekturalnu strategiju pomera bliže ekstremu BDUF. Zahtevi koji od sistema zahtevaju da radi na više platformi, da se integriše sa nekim nasleđenim sistemom i podacima, da interreaguje sa drugim sistemima, da ispoštuje zakonske obaveze, kao i zahtev tipa da se podrazumeva da sme nikada da "propadne", zahtevaju obimniju arhitekturalnu analizu:

ISP14: "...u situacijama kada se radi integracija sa third party servisima, kada nije sve u rukama tima koji radi razvoj i ako imaju uticaja zakon i zakonske obaveze. Definitivno veće planiranje je kada projekat ima veće constrainte, pogotovo što se tiče tih nekih zakonskih. Npr. ako je softver za medicinu ili ako utiče na ljudske živote, ako postoji velika zainteresovanost medija okoline, da tu bih išao sa pristupom koji je nešto strožiji i imao neki plan na početku, ali ne bih ginuo za taj plan, nego bih dozvoljavao da se taj plan razvija...".

Međutim sama priroda takvih projekata je drugačija i zahteva fokus na razumevanje nasleđenog sistema i identifikovanje načina za njegovu interkonekciju sa novim sistemom, kao i na analizi tehnologija kojim je razvijan:

ISP13: "...utiče samo način kako je omogućena interkonekcija između ta dva sistema. Vaša arhitektura može biti nezavisna od arhitekture tog drugog sistema, dok god taj vaš interfejs između njih je definisan nekim standardima (npr. web servisi). Ako odskače od standarda onda to može da utiče na vašu arhitekturu u nekoj meri...".
ISP19: "...u slučaju kada imate legacy sisteme sa kojima treba da interreaguje sistem, to svakako otežava arhitekturu...pogotovo ako je neka nova tehnologija tj. dovoljno stara da još nismo sa njom radili... to može direktno da utiče na arhitekturu i oteža i produži fazu planiranja arhitekture...".

Ispitanici naglašavaju da je najveći problem u praksi, kod ovakvog tipa zahteva, da se razmotre pitanja vezana za nasleđene podatke. Odnosno koji su nasleđeni podaci relevantni za izradu modela podataka novog sistema, gde će se čuvati i kako će se njima upravljati do momenta prelaska softvera u produkciju:

ISP15: "...treba da se zna da postoji momenat kada će stari sistem da prelazi na novi sistem. Treba da se ili očuvaju podaci ili da se konvertuju u novu strukturu ili da se izvrši čišćenje tih podataka. Mora da se zna šta se radi u ranoj fazi planiranja projekta, evaluirati i utvrditi koji su to podaci koji su relevantni i ulaze u sam model podataka. Gde ćemo mi to da sačuvamo, ili ćemo ga koristiti externo kao arhivu. Znači moramo da znamo kako ćemo hendlati same podatke do momenta prelaska iz naše aplikacije u produkciju. Ono što je ključno je da taj deo ne utiče direktno na arhitekturu. Arhitektura novog sistema bi trebalo da bude osokoljena novim metodologijama, novim mogućnostima, a tranzicija starih podataka unutar nove strukture je kao neki mini projekat koji se mora planirati u okviru samog projekta".

Takođe, jedan od ključnih problema postojanja nasleđenih sistema vezuje se za nedostatak dokumentacije o istom:

ISP10: "...povezivanje sa nekim legacy sistemimom je baš izazovno, s tim da je tu pre naglasak na upoznavanju i razumevanju legacy sistema...obično u tim legacy sistemima ili fali dokumentacija, ili je štura ili uopšte ne postoji, pa je time veći izazov razumeti nasleđeni sistem, pa samim tim i novo rešenje prilagoditi postojećem sistemu...".

Problem višeplatformskog rešanja ispitanici adresiraju kao problem poznavanja različitih tehnologija što zavisi od samog iskustva i znanja softver arhitekte:

ISP3: "...sve zavisi od iskustva, ja lično sam radio na veoma velikom broju platformi. Počev od Windows razvoja, Linux-a, C++, .NET-a, Jave, Rubija, Piton-a, i još nekih stvari, tako da, uključujući i razne baze podataka, Microsoft SQL, NoSQL baze, MDB. Ako je arhitekta iskusan, i ako ima iskustva sa svim tim rešenjima, onda sama kompleksnost svega toga i ne utiče toliko na produžetak faze planiranja, jer je to već sve prirodan okvir u kom se arhitekta kreće. Naravno, ukoliko ste arhitekta koji je specijalizovan za jednu platformu ili neki skup rešenja, onda uključivanje bilo kojeg spoljnjeg, nekog novog dela, zahteva prilično izučavanje. Barem je tako meni bilo kad sam se uključivao u sve ovo... svaki alat mora dobro da se ispita da li stvarno može da uđe u rešenje ili ne može i u kojoj meri i na koji način da se iskoristi u rešenju...".

Međutim, ističu i da zahtev višeplatformskog rešenja itekako ima implikacija na samu arhitekturu, što zahteva podrobnu analizu i izbor adekvatne tehnologije na početku projekta:

ISP20: "...glavni impact je na arhitekturu. Kako se komunicira, kako se prenose podaci, kako se event-i opaljuju, kako se prenose, da li Windows odreaguje na isti način, kako asinhrono daje feedback, itd. Znači ogroman impact. Sama multiplatforma Windowsa zahteva značajnije analize i veće up front razmišljanje o samoj arhitekturi i mogućem rešenju koje je jedinstveno za sve vrste platformi..."
ISP5: "...pa u principu utiče na izbor tehnologija...nije svejedno kakvu tehnologiju ćemo koristiti, jer su neke tehnologije bolje a neke lošije za rešavanje određenih problema i razvoj određenih vrsta softvera...".

Kompleksnost zahteva ogleda se i u broju korisnika, transakcija i količini podataka koja se obrađuje:

ISP13:"... kad govorimo o nečem gde imate na stotine, hiljade, milione nekih transakcija onda vi morate znati kako ćete skaliranje postaviti, kako ćete redudantnost sistema postaviti... šta je to što pravite, koliko je to mission critical aplikacija. Sama priroda aplikacije utiče na milion faktora i na kvalitet i na samu arhitekturu...".
ISP2: "...sistem je kompleksan ako zahteva veliki broj korisnika, recimo 10 miliona korisnika, ili možda radi sa velikom količina podataka... od toga zavisi koja količina vremena će se potrošiti na takve neke aspekte, tipa skalabilnost sistema. Da li je to sistem u cloud-u, pa mu treba auto skalabilnost i sl....".

Ispitanici se slažu da je kompleksnost u direktnoj vezi sa većim obimom up front arhitekturalnih napora, a rešenje vide u iterativno inkrementalnom razvoju arhitekture, čime se izbegava ekstremna strategija BDUF:

ISP9: ".. što je kompleksniji domen to bih ja možda hteo što kasnije da donosim neke odluke. Što kasnije donesem odluku manja je šansa da će ona biti pogrešna... ako je u pitanju vrlo kompleksan domen ulazimo u rizik da donesemo odluku kada još ne znam dovoljno o njemu. U praksi ako kažem da imam prvo 3 meseca za neki upfront dizajn onda imam 3 godine implementacije, imam veliki rizik da posle ta 3 meseca ne znam dovoljno o tom domenu i da donesem pogrešne odluke. Umesto toga preporučljivo je naći način da se sve to radi inkrementalno...".

Upotrebi frejmvorka i predefinisanih arhitektura je takođe strategija koja doprinosi smanjivanju up front arhitekturalnih napora:

ISP19: "...zadnjih 5 ili 10 godina koliko radimo, arhitekture se nisu drastično menjale. Čim se vidi kakav je problem možemo ga svrstati u neki od već postojećih tipova arhitekture i onda samo detalji se razlikuju...".

Ovakvi stavovi ispitanika se podudaraju sa mišljenjem Mirakhorli i Cleland-Huang (2013), koji smatraju da se veća agilnost može postići upravo korišćenjem poznatih arhitekturalnih rešenja tj. referentne arhitekture. Pod pojmom referentne arhitekture podrazumeva se upotreba frejmvorka i predefinisanih arhitekturalnih rešenja. Upotrebom referentnih arhitektura u startu je veći deo arhitekture već definisan, što omogućava postizanje veće agilnosti na projektu (Kruchten, Obbink, Stafford, 2006).

B. Kategorija zahtevi

Kategoriju *zahtevi* sačinjavaju sledeće podkategorije: *nepotpuni zahtevi* i *promenljivi zahtevi.* Zahtevi predstavljaju važan faktor pri odabiru arhitekturalne strategije u praksi, jer njihove osobine utiču na njeno pomeranje ka jednom od ekstrema. Većina ispitanika je istakla da uglavnom radi u nestabilnim uslovima, jer klijenti vrlo često na početku projekta ne znaju šta tačno žele. To implicira zaključak da su i zahtevi, sa kojima raspolaže softver arhitekta na početku projekta, nepotpuni:

ISP8: "...često vam dođe klijent sa idejom koju on nije razradio. U našem je interesu, a i njegovom, da ga nateramo da je u svojoj glavi razradi, i to nekada potraje. Mnogi dođu i sednu i žele da projekat što pre počne, ali uglavnom ne može tako. Vaš posao je da ga naterate da o tome razmisli...".
ISP6: "...najveći problem je u tome što je dosta stvari implicitno za tog nekog kome rešavate problem...".

Klijenti često nisu u stanju da sami definišu sopstvene zahteve i potrebe, što predstavlja dodatni razlog zbog kojeg se vreme ulaska u fazu implementacije produžava:

ISP10: "...naravno, vrlo često smo u situaciji da mušterija ne zna šta hoće, i onda i kad zna šta hoće, ne zna kolike su implikacije tih zahteva. Upravo je tu krucijalna uloga softver arhitekte, koji već iz tih zahteva može da prepozna drastične promene u pristupu i u samoj arhitekturi, i šta će biti izvodljivo a šta ne....".

Kvalitet zahteva direktno utiče na dužinu vremena koje softver arhitekta mora provesti u up front arhitekturalnoj analizi, kako bi utvrdio granice sistema i arhitekturalno značajne zahteve na osnovu kojih može postaviti arhitekturu za glavni deo softvera:

ISP8: "...ne može da bude pod znakom pitanja ono što je core projekta...".
ISP4: "...kvalitet requirementa, podrazumeva da je dobro identifikovan skoup, da stejkholderi i klijent nisu ništa bitno izostavili...ako su requirementi kvalitetni, biće više stvari poznatih unapred. U drugoj situaciji vi morate da dođete na taj nivo, morate da uložite više efforta, da biste ih dobro identifikovali i to produžava tu početnu fazu...bavite se zapravo requirement analizom, a ne arhitekturom, ali je to neophodno da biste napravili arhitekturu...tako da se u našoj kompaniji od arhitekte očekuje da radi i kao konsultant...".

Opisan problem je naročito izražen kod nefunkcionalnih zahteva, koji su krucijalni za arhitekturu, a kojih klijenti uglavnom nisu ni svesni ili pak ne umeju da ih pretoče u oblik koji je razumljiv softver arhitekti:

ISP3: "...retko kad sam sretao klijente koji mogu te zahteve da postave, oni nisu ni svesni tih zahteva. A veoma su bitni za arhitekturu i deo su svakog rešenja...".
ISP4: "...sa klijentom identifikujete nonfuctional requiremente, zato što to klijenti ne razumeju. Mi kad pričamo o performansama aplikacije, vi morate to da kvantifikujete da bi mogli posle da proverite da je to to. Kako to kvantifikujete? Ni jedan klijent vam neće reći ovo je web aplikacija, u piku očekujemo 10000 konkurentnih korisnika, i response time mora da bude maximum 100ms po stranici. To nećete dobiti od klijenata, mislim ako dobijete, to je super, to su vam sjajni klijenti. To su stvari koje morate da im objasnite, i da definišete zajedno. Jer to je jako bitno za arhitekturu, mislim da ima veći impact nego functional requirements, zato što je trend koji se tiče functional requirementa, da se rade servis orjentisana arhitektura, ali načini na koje radimo je drugačiji...".

Identifikovanje arhitekturalno značajnih zahteva na početku projekta znači višestruku korist, ne samo za razvojni tim već i za klijenta. Naime, postavka inicijalne arhitekture podrazumeva definisanje stvari koje su skupe i koje se tokom projekta ne trebaju menjati. Naknadno identifikovanje i izmena krucijalnih elementa arhitekture softvera, u kasnijim razvojnim

fazama, dovodi do izazova da li raditi ispočetka dizajn arhitekture ili pokušati sa refaktorisanjem postojećeg neadekvatnog rešenja. U oba slučaja enormno se povećavaju troškovi projekta i produžava vreme njegovog trajanja:

ISP8: "...npr. sa klijentom uopšte niste razbistrili priču oko toga koliki je IO tj. napad na storage servise. I vi ste to dizajnirali spram 10000 korisnika. Međutim kada posle tri meseca klijent možda i prvi put razmislio o tome, vi shvatite da ono što pravite za njega zahteva potpuno drugačiji koncept u domenu recimo upisivanje i iščitavanje podataka. Što znači da vi ceo taj data access layer morate da bacite i da pišete ponovo. Onaj koji dizajnira mora da zna koji je load ili teret koji očekuje od korisnika i spram toga da odabere bazu podataka, neke nestrukturirane tipove podataka, da smestite u cloud, da ukoliko je veliki load u čitavu arhitekturu ugradi skalabilnost. Jer te stvari su skupe i koštaju, i iziskuju dodatno vreme, tako da vi to nećete raditi ukoliko za tim nema potrebe. Međutim, ukoliko se ta potreba iskaže za tri meseca, a vi ste sve radili na taj način da to ne može da se uradi prosto, vi ste vrlo često pred izazovom da li je bolje da krenete skoro od početka ili da refaktorišete...".

Da bi predupredili opisane izazove, stručnjaci u praksi se često služe konceptom spike-ova. Koncept se implementira u dva slučaja: kada stejkholderi imaju neki zahtev, ali nisu sigurni šta je to što žele u pogledu funkcionalnosti ili kada softver arhitekta i razvojni tim nisu sigurni u način njegove implementacije. Tada se odvoji deo vremena kako bi se napravio neki prototip, analiza arhitekture i sl., da bi se saznalo što više o tehnologiji i načinu implementacije određenog zahteva *(ISP20).*

Promenljivost zahteva, kao druga identifikovana podkategorija zahteva, predstavlja drugi najčešći problem sa kojim se susreću softver arhitekte i agilni timovi u praksi. Promenljivost zahteva vezuje se za fazu implementacije i ogleda se u klijentovoj potpunoj promene ideje šta očekuju od softverskog rešenja, čime se može dovesti u pitanje održivost postavljene arhitekture:

ISP13: "...zahtevi koji dolaze od strane klijenata su jako često izvor problema, pošto radite agilno i njihovi zahtevi dolaze tokom razvoja softvera... proširuje se njihova vizija. Proširenjem vizije ljudi realno angažuju možđane ćelije, ono što bi se reklo, "just in time", baš kad moraju. Tako da tek kad im postavite prvu verziju softvera onda oni vide šta on može, pa se njihova percepcija šta softver treba da radi malo usklađuje sa time. Niko neće priznati da je pogrešio u proteklom vremenu, nego će samo slučajno ili namerno izmeniti svoj tok misli i tvrditi kako im je ta poslednja vizija ustvari ta koju su kao od početka nametali. Onda vi konstantno imate modifikaciju specifikacije arhitekture, kako bi se prilagodila tom nečemu što je korisnik tražio. U startu bi se moglo desiti da ste vi postavili osnovne temelje, uradili nad tim nešto i onda je nešto od tih temelja poremećeno zato što se vizija korisnika promenila, te su neophodne modifikacije, proširenja ili izrada arhitekture ispočetka..." .
ISP12: "...problem je u tome što se specifikacija i zahtevi menjaju i oni loše utiču na softversku arhitekturu...".

Mitigacija ovog rizika moguća je ulaganjem više napora u početnoj fazi projekta, pre postavke arhitekturalnog rešenja za glavni deo sistema, gde bi se sa stejkholderima postigao konsenzus po pitanju poslovne i arhitekturalne vizije. To ne podrazumeva detaljnu razradu svih zahteva na početku projekta, jer to u uslovima savremenog poslovanja nije ni moguće, već samo arhitekturalno značajnih zahteva za glavni deo sistema. Poslovni softver je danas ključak za uspeh biznisa i u skladu sa tim, razvoj softvera predstavlja kontinuirani proces, kao i svaki poslovni proces jedne organizacije. Poslovne promene impliciraju nove zahteve i stalno prilagođavanje softverskog rešenja, što implicira da i razvoj arhitekture mora biti kontinuiran, iterativno inkrementalni proces. Inicijalno postavljena arhitektura za glavni deo sistema evoluira tokom projekta, pri čemu se detaljna razrada određenog seta zahteva sprovodi u momentu planiranja iteracije kojom će se isti implementirati. Zaključak koji sledi je da se brojne arhitekturalne odluke moraju odložiti do momenta boljeg razumevanja zahteva, što se može nazvati JIT planiranjem arhitekture:

ISP6: "...biznis toliko brzo evoluira i tim mora stalno da bude aktivan i da aktivno menja aplikaciju. Sad trenutno kad se pravi neki biznis sistem, ne može da se dogodi da neki član tima uzme sve na sebe pa nešto kao napravi, pa se to kao uradi deploy i to će kao da radi i gotov projekat. Tako nešto ne može da postoji. Sada je softver biznis driver. Imamo slučajeva kada neko dođe kod nas sa nekom idejom, pa da ih mi konsultujemo, da im napravimo sistem koji će to da radi i da utvrdimo koliko će to da košta i to je to. Međutim, takav razvoj softvera ne postoji u 21. veku. Onda im kažemo da će to prvi mesec koštati toliko, drugi mesec toliko, računajte da ćete godišnje potrošiti toliko, itd. Pa onda pitaju kako to mislite? Zapravo ne razumeju da i auto koji kupiš pa ga voziš takođe mora da ide redovno na servis. Tako i softver kao biznis driver. treba ti konstatno tim, koji je potrebno skalirati tako da može da podrži zahteve biznisa. Taj tim konstantno radi na tom softveru i kako se taj biznis menja tako i tim vrši promene u softveru..."
ISP9: "...razmatranje arhitekture na početku projekta je neophodno i svakako se time smanjuju tehnički rizici, bez toga ništa ne bi moglo da se uradi. Međutim, ne treba preterati sa tim i ulaziti u previše detaljan dizajn... Ako smo pričali o ovom projektu koji bi mogao imati 3 meseca upfront dizajna i onda 3 godine razvoja, ja kažem ajde da radimo tako da se radi u incrementima od po tri meseca, gde arhitektura uvek radi malo unapred, jedan increment unapred u odnosu na samu implementaciju, ali ne 3 godine i 3 meseca unapred...".

Promena postavljene inicijalne arhitekture opravdana je samo usled velikog napretka poslovanja organizacije i veće upotrebe sistema u odnosu na planiranu, iz razloga što je tada ugrožena performantnost postavljenog arhitekturalnog rešenja. Realno je da ni klijent ni agilni timovi ne mogu da predvide koliki uspeh će doživeti neko softversko rešenje na tržištu:

ISP8: "...savršeno je normalno ako se taj projekat skalira i postane krupan i on dobije milione korisnika, onda će se i arhitektura menjati i to je potrebno negde predvideti. Ali je to jako teško predvideti, jer projekti koji uspeju na takav način oni direktno dolaze do novca koji je krupniji i arhitekture u takvom okruženju su potpuno drugačije...Retko kad organizujete i pravite arhitekturu spram toga da će te imati 5 miliona korisnika, jer je to vrlo skupo za razvoj u tom trenutku. Stoga je veliki uspeh biznisa prihvatljiv razlog da se arhitektura refaktoriše. Ali je neprihvatljivo da na neke manje skokove u broju korisnika vi morate sve da

refaktorišete. To vam je sigurna izlazna karta u komunikaciji sa klijentom. Klijent neće biti spreman na to, jer on ima svojih problema u domenu biznisa, novca i dr.. I onda mu se vi, posle njegovog inicijalnog uspeha, pojavljujete sa konstatacijom da ste pravili ovo za hiljadu ljudi, a ne za 200 hiljada i da treba sve ispočetka da se pravi...".

Iz dobijenih rezultata može se zaključiti da nepotpuni i promenljivi zahtevi predstavljaju faktor koji utiče na pomeranje strategija razvoja softverske arhitekture ka ekstremu nascentne arhitekture. Međutim, kvalitet zahteva utiče i na dužinu vremena koje softver arhitekta mora provesti u up front arhitekturalnoj analizi, kako bi utvrdio granice sistema i arhitekturalno značajne zahteve na osnovu kojih može postaviti arhitekturu za glavni deo softvera. Dakle, opisana priroda zahteva isključuje mogućnost detaljnog postavljanja arhitekture celog sistema, ali utiče na obim vremena koje se mora odvojiti za arhitekturalnu analizu na početku projekta.

C. Kategorija stejkholderi

Osobine stejkholdera predstavljaju važan faktor koji utiče na agilnost razvojnog tima, arhitekture i projekta u celini. Njihova spremnost na kontinuiranu saradnju, kroz aktivno učešće na projektu i redovno ispostavljanje povratnih informacija timu, utiče na izbor arhitekturalne strategije.

Agilniji stejkholderi impliciraju manje vremena i napora potrebnog za up front arhitekturalne aktivnosti. Pojedini ispitanici smatraju da je kontinuirana komunikacija sa stejkholderima ključ za uspešan razvoj agilne arhitekture:

ISP7: "...ako klijent ne želi da bude toliko uključen, mi ne uzimamo taj projekat...konstantna komunikacija sa klijentom obezbeđuje da je arhitektura na dobrom putu...".
ISP16: "...ključno je rad sa stejkholderima, sa product owner-ima, čak direktno sa klijentima, pošto product owner-i jesu tehnička lica, ali nisu na nivou arhitekti, tako da ne mogu ni oni da prenesu baš 100% svih zahteva...".

Pored spremnosti stejkholdera na aktivno učešće tokom trajanja projekta, bitan je i nivo kvaliteta njihovog učešća:

ISP4: "...morate da procenite i koliko kvalitetnu komunikaciju ćete imati sa klijentom, koliko je on spreman da aktivno učestvuje i koji je value tog učestvovanja. Oni često hoće da učestvuju, ali ne mogu mnogo da vam pomognu - morate da potrošite mnogo vremena da izvučete informacije iz njih...".
ISP13: "...itekako je važno da li su neki stejkholderi ljudi koji samo razumeju problematiku posla, ali ne znaju kako to da izraze, ili pak znaju i problematiku posla i znaju kako to da izraze. Imate ljudi koji razmišljaju kockasto, koji razmišljaju da bilo koji predlog ne može proći i da sve što se ne uklapa u njihovu sliku ne može adekvatno da prođe njihove filtere...".

Veliki deo problema na projektu i na samoj arhitekturi potiče od nestručnosti stejkholdera, koji treba da obezbede adekvatne zahteve:

ISP7: "...a ti problemi se uglavnom javljaju ako pričate sa nestručnom osobom sa druge strane...".
ISP8: "...u 90% slučajeva problemi su mogli biti rešeni da je komunikacija bila bolja i da su zahtevi bili definisani kako treba...".

Stoga je veoma važno, za uspeh i arhitekture i projekta u celini, na početku projekta identifikovati ključne stejkholdere:

ISP13: "...identifikovati ko od njih ima najveći uticaj na taj softver, obično to ne mora biti direktor koji je dodeljen, nego to može biti neko drugi, neko treći..., jer imate dosta politike..., jer softver ne bude implementiran iz prostog razloga što ta grupa ljudi nije bila u startu prepoznata kao bitan stejkholder, pa da se vide njihovi interesi, pa da ti ineresi budu i menadžmentu predočeni (npr. super mi ćemo vam to napraviti ali imate ovde dosta veliku grupu ljudi koji neće biti stimulisani da to koriste) kako bi napravili rešenje tako da i njima bude lakša upotreba softvera...".

Izbor arhitekturalnog rešenja treba da bude proizvod tesne kolaboracije stejkholdera i softver arhitekte. Uloga arhitekte je da stejkholderima prikaže moguće varijante arhitekturalnih rešenja, uz opis očekivanih rezultata koje će data rešenja obezbediti *(ISP3)*, kao i opis troškova i rizika koje podrazumevaju *(ISP8)*. Uloga stejkholdera je da donese konačnu odluku kojim će se "putem" krenuti *(ISP3)*.

Teško je, ili gotovo nemoguće, primenjivati arhitekturalnu strategiju koja je bliska nascentnoj arhitekturi, ukoliko se radi za neagilnog stejkholdera (klijenta/korisnika). Stejkholderi ne prihvataju agilne principe razvoja *"...prvenstveno zato što hoće da se osiguraju, hoće u ugovor da stave scope, timeline i kvalitet, a kada sve to fiksiraju ugovorom, ne vide način da rade agilno..." (ISP9).*

Klijenti velikih sistema uglavnom imaju nepoverljiv stav prema agilnim procesima razvoja, zbog eliminisanja faze planiranja i razvoja potpuno nascentne arhitekture: *"...oni se čak i uplaše kad im kažete da počinjete razvoj za 3 dana i misle da tu onda prosto nešto nije u redu..." (ISP8). S*a druge strane, klijenti start up projekata nemaju novca za postavljanje arhitekturalne strategije na početku projekta i fokusirani su samo na ranu isporuku vrednosti, a ako proizvod/servis na tržištu doživi masovnu upotrebu, spremni su na potpunu promenu arhitekturalne strategije:

ISP8: "... vrlo često sam bio u start up-u, tj. mnogo puta sam kretao od nule i u 95% slučajeva takvi projekati propadaju. Takva je priroda start up industrije, mali broj start up-a prežive 4 ili 5 godina".
ISP7: "...vodimo se minimum viable product (MVP), znači da se sa 20% napora i uloženog truda, pokrije 80% stvari. Jer je to jako brz feedback klijenta, tj. obezbeđena je implementacija cora te aplikacije...".

D. Kategorija tim

Narednim tekstom biće diskutovano kako osobine tima, kao što su: broj članova, znanje u rešavanju konkretnog problema, znanje iz domena problema, poznavanje tehnologija i trenda opcija, iskustvo članova tima, dobra komunikacija i kolaboracija članova tima i sl., utiču na razvoj arhitekture kompleksnih sistema u agilnim procesima.

Ukoliko je tim rešavao sličan problem ranije, onda poseduje iskustvo i znanje, što doprinosi smanjenju vremena i napora u up front arhitekturalnoj analizi i dizajnu:

ISP11: "...ako smo već sličan problem rešavali, možda ne u ovom domenu, možda u nekom drugom domenu, ali opet bilo je slično, tada imamo neko znanje od pre i sada lakše radimo... ne trebaju nam prototipi, ne trebaju neki dodatni sastanci i razmatranja za nešto što već znamo kako ćemo da koristimo. A sa druge strane ako radimo nešto totalno novo, nešto sa čim se niko u kompaniji nije sretao, za to unapred znamo da će nam trebati puno vremena. E sad kako tu uvek vremena nema uglavnom takve feature krećemo da razvijamo sa nedovršenim dizajnom i arhitekturom, što kasnije za posledicu ima da se vraćamo unazad, pa refaktoring, redizajn i slično...".

Na smanjivanje vremena i napora u up front arhitekturalnoj analizi i dizajnu utiče i znanje tima iz domena problema, kao i dobro poznavanje tehnologije sa kojom treba da radi:

ISP19: "...pitanje je da li ti ljudi koji počinju da planiraju stvarno imaju znanja koja misle da imaju. Mi smo mislili da znamo dovoljno o kladionicama i imali smo podršku jedne velike kladionice, međutim ni oni nisu bili svesni šta sve drugi rade. Znači bitno je ne samo koliko ti znaš o svom proizvodu nego koliko i konkurencija ima nešto već urađeno. U konkretnom primeru, "On line betting" je prešao kompletno sa jedne vrste na online. Tipovi na koje se ljudi klade, zašto se klade, je potpuno drugačije. Kontrola se potpuno promenila...".
ISP3: "... ako imate tim Java programera, sigurno nećete raditi rešenje koje je specifično za Microsoft tehnologije ili .NET, i obrnuto, ukoliko imate Microsoft programere, nikad nećete uključivati rešenja koja koriste neke druge tehnologije. Možda je čak i prvi korak u svemu tome tim sa kojim će se raditi, jer je osnova celog rešenja u tome ko će raditi implementaciju. Ne može da se radi arhitektura ako nemamo ljude koji će je sprovesti u delo...".

Međutim, u praksi su neretke situacije da klijenti nameću svoj tehnološki stek. U tom slučaju, timu je potrebno dodatno vreme za istraživanje date tehnologije i konsultovanje sa ljudima i kompanijama koje imaju iskustva sa datim tehnologijama. Sve ovo utiče na odlaganje početka implementacije funkcionalnosti:

ISP14: "...u outsourcing-u česta je situacija da se ne može birati tehnološki stek...".
ISP3: "...naravno ukoliko ste arhitekta koji je specijalizovan za jednu platformu ili neki skup rešenja, onda uključivanje bilo kojeg spoljnjeg, nekog novog dela, zahteva prilično izučavanje. Barem je tako meni bilo kad

sam se uključivao u sve ovo... kako se širi spektar alata, svaki alat mora dobro da se ispita da li stvarno može da uđe u rešenje ili ne može i u kojoj meri i na koji način da se iskoristi u rešenju...".
ISP18: "...imali smo par ljudi koje smo pitali za njihovo iskustvo u radu sa nekim tehnologijama, koje smo hteli da primenimo, onda smo mogli mnogo efikasnije da uradimo naš projekat...".

Ukoliko su članovi tima iskusni pojedinci u domenu tehničko-tehnoloških opcija, njihovih mogućnosti i ograničenja, znatno se ubrzava up front planiranje arhitekture i obezbeđuje izbor tehnologije koja je adekvatna za problem koji se rešava:

ISP2: "...da budu ljudi upoznati sa tehnologijom, tehnološkim inovacijama i možda nekim komponentama koje su dostupne, da se ne pravi sve ponovo. Puno stvari se radi, u principu postoje puno besplatnih...tako da se puno stvari može iskoristiti, ne mora se sve ponovo raditi. To omogućuje brži razvoj, jeftinije rešenje...".
ISP5: "...mislim da je jako važno da softver arhitekta, a i tim, imaju širinu u poznavanje raznih tehnologija...".

Takođe je vreme uloženo u up front arhitekturalne aktivnosti mnogo manje, ukoliko razvojni tim čine iskusni pojedinci u domenu arhitekturalnih znanja. Razlog je što iskusni softver inženjeri mogu dosta odluka da donesu implicitno. Iskusni arhitekte imaju širinu u pogledu znanja, zbog čega su svesni mogućih opcija za implementaciju rešenja nekog problema. Pošto bolje razumeju šta će raditi, a šta neće, oni za razliku od neiskusnih, donose bolje i brže arhitekturalne odluke:

ISP5:"... što su mlađi ljudi, treba više investirati u razvoj arhitekture up front. Ja sam imao jedan sjajan tim, i dalje plačem zbog toga što smo se rasuli po kompaniji. Bilo nas je šestoro i dovoljan je bio list papira da se dogovorimo, jer smo se bili ušemili na prethodnim projektima (svi smo pratili isto kodni guideline, isti način i prenos parametara, rukovanje greškama, definisanje modula i paketa itd.) i tada treba mnogo manje tog up fronta-a. U suprotnom, što je mlađi tim i neiskusniji, to treba više više vremena i napora uložiti u up front aktivnosti...".
ISP16: "...iskustvo oko dizajna softvera je preko 90% faktor uspeha...".

Podizanje nivoa znanja svih članova timova, u domenu agilnih procesa razvoja, bitan je faktor koji povećava agilnost tima i smanjuje up front planiranje:

ISP14: "...ukoliko ih pošaljete na kurs koji je ranga profesional scrum developer, ili bilo koji kurs koji daje dobru osnovu svim XP agile praksama, takav developer može da uđe u svaki tim i da bude sposoban da nosi odgovornost za arhitekturu. Edukacija i komunikacija su ključne...".

Navedeno mišljenje ispitanika u skladu je i sa nalazima u literaturi. Boehm i Turner (2003) su zaključili da agilan razvoj zahteva kritičnu masu eksperata, koje su definisali kao članove tima "koji su sposobni da revidiraju metod, postavkom pravila u skladu sa novim okolnostima i situacijom u praksi. Ekspert ne mora slepo da sledi pravila i uputstva, već treba da ima dovoljno znanja, veština i iskustva kako bi mogao i intuitivno da donosi odluke".

Broj timova/ljudi koji učestvuje u realizaciji nekog projekta, većina ispitanika je identifikovala kao faktor koji utiče na veći obim up front planiranja, prevashodno iz razloga što se moraju analizirati zavisnosti između timova *(ISP4).* Naime, "*...ako su njihove razvojne komponente međusobno povezane, to traži potpuno drugu dimenziju planiranja i koordinacije i arhitekture i načina sprovođenja i načina testiranja i organizacije...*" (*ISP19).* Ispitanik ISP19 navedeni razlog pojašnjava kroz sledeći primer:

ISP19: "... npr. šta ako ja u trećem sprintu zavisim od neke druge komponente u prvom sprintu i ako je oni ne završe...Arhitekte isto tako moraju unapred da pripreme dovoljno posla za sve i da znaju koje su komponente ključne na kritičnoj liniji. Tako da tu up front dizajn se drastično povećava, ali opet ni jedan gazda firme niti biznis owner nas neće platiti da mi sad up front dizajniramo 50% vremena. To je osnovni problem arhitekte u velikim timovima i velikim firmama, jer imaš standardan konflikt vremena i zahteva...".

Ispitanik ISP14 je istakao da je dobra komunikacija i kolaboracija između članova tima ključni faktor uspeha arhitekture i način na koji se može amortizovati veće up front planiranje u slučaju većeg broja timova:

ISP14: "...ukoliko je krosfunkcionalni tim koji je u stanju da se sam organizuje i da bude odgovoran za to što radi, znači ako nemamo neke velike disfunkcije timova....zato što je kultura izgrađena tako da je komunikacija dobra, da je sve vidljivo, da se ništa ne sakriva, onda to funkcioniše samo od sebe. Ako imamo te probleme onda to nije agilni tim. Onda je po meni potreban dobar agile coaching...".

Opisan nalaz istraživanja u skladu je sa nalazima u literaturi, gde je dobra komunikacija označena kao faktor koja bitno utiče na vreme i arhitekturalne napore na projektu. Naime Coplien and Bjornvig (2010) su istakli da se dobrom komunikacijom članova tima arhitekturalni napori mogu višestruko smanjiti, navodeći primer smanjenja sa šest meseci na dve nedelje.

Međutim, agilne kolaboracije, dobra komunikacija i iskustvo u agilnim timovima, ne mogu se postići od danas do sutra, već zahtevaju vreme koje će članovi tima provesti zajedno na projektima. Što su duže članovi zajedno, time se vreme potrebno za up front planiranje arhitekture skraćuje:

ISP19: "...mi smo taj problem prevazišli pošto imamo isti tim već oko 3 godine i sve je manje problema u komunikaciji između arhitekti i developera... sve manje moram u detalje da im pričam. Dogovorimo se koji patern ćemo koristiti, šta je specifično u tom i tom delu, kažem im da obrate pažnju na odredjene relacije gde može biti problem, prodiskutujemo o tome i onda oni programiraju. Ti programeri polako sa povećavanjem svog znanja bliže se arhitektama, tj. već znaju da iskroje neke delove sami...".

Dobijeni nalaz je u skladu sa nalazima u literaturi. Hoda (2010) je empirijski utvrdio da je neophodno određeno vreme da bi članovi nekog tima stekli iskustvo, samoorganizovanost, samovrednovanje i samopoboljšanje.

Takođe, agilni tim u pravom smislu reči podrazumeva i članove koji sede u istoj prostoriji i komuniciraju "licem u lice" na dnevnom nivou. To dalje znači da su agilni procesi predviđeni za "in house" razvoj projekata. Međutim, najveći broj projekata koji se razvija u Srbiji je outsourcing zbog čega je ispitanik ISP14 je zaključio da takvi projekti nikada ne mogu biti potpuno agilni:

ISP14: "...po meni outsourceing ne može nikada biti 100% agilan...za potpuno agilan razvoj trebao bi da bude „in house" razvoj projekta...".

4.2.1.2 Kategorija uloge i odgovornost donošenja arhitekturalnih odluka

U nastavku rada biće opisana uloga softver arhitekte u agilnim procesima razvoja kompleksnih softverskih rešenja. U tu svrhu biće integrisani nalazi iz postojeće literature i nalazi dobijeni empirijskim istraživanjem u doktoratu.

Empirijski rezultati istraživanja pokazuju da u svim agilnim timovima postoji formalna uloga softver arhitekte, koji je istovremeno i vrlo iskusan programer. Nalaz je u skladu sa stavovima i preporukama Coplien-a i Bjornvig-a (2010). Kao iskusan programer, softver arhitekta se uglavnom uključuje u razvojni tim na početku projekta, kako bi sa programerima postavio glavni deo softvera (ISP5). Uloga softver arhitekte, u agilnim procesima razvoja kompleksnih softverskih rešenja, bitno je izmenjena u odnosu na tradicionalnu, jer zahteva njegovo uključivanje kroz ceo razvojni proces. Softver arhitekta predstavlja neku vrstu mentora, koji tokom projekta savetuje i pomaže programerima oko rešavanja arhitekturalnih pitanja i problema (ISP11). Ovakvo stanje u praksi u skladu je sa stavovima u literaturi, koje zastupa Hadar (2012).

Faber (2010) smatra da arhitekte treba da obezbede vrednost klijentima, kroz ispunjavanje nefunkcionalnih zahteva sistema, a programerima, da pružaju kontinuiranu podršku tokom procesa implementacije rešenja. Agilni timovi u praksi svesni su ove činjenice i teže uspostavljanju idealne situacije, u kojoj bi arhitekte predstavljale provajdere usluga programerima i klijentu. Međutim, za ostvarivanje ovakve uloge softver arhitekte neophodan preduslov je *"...podizanje svesti, poverenja, veština i znanja iz domena problema i tehnologije..."* svih članova tima. Identifikovan problem u praksi je konstantan priliv novih ljudi i činjenica da je prosečan inženjer u timu na nivou juniora (ISP20). Podizanje nivoa tehničkog znanja članova tima najbolje se može postići njihovim uključivanjem na diskusije o arhitekturi, prilikom njene postavke na početku projekta i tokom planiranja iteracija. Na ovaj način se izbegava

usko grlo ljudi, koji postaju specijalisti za arhitekturu, dok ostali članovi tima nemaju ni osnovno arhitekturalno znanje niti celovitu sliku rešenja (ISP20).

Ispitanici smatraju da i sam arhitekta mora izgraditi/nadograditi svoje sistemsko, tehnološko i domensko znanje, na samom početku projekta, kako bi mogao pružati kontinuiranu pomoć programerima i stejkholderima (ISP19). Iz tih razloga bitno je da arhitekta učestvuje na sastancima sa stejkholderima, kako bi iz prve ruke čuo koji su problemi, jer u suprotnom nikada ne dobije potpune informacije putem dokumentacije koju mu dostavi vlasnik proizvoda (ISP7). Iako su vlasnici proizvoda uglavnom tehnička lica, ipak nisu na nivou arhitekti, tako da nisu u stanju da prenesu apsolutno 100% svih zahteva (ISP16).

Hopkins i Harcombe (2014) smatraju da je za uspeh softverske arhitekture važno da softver arhitekta na početku projekta sagleda problem koji se rešava, i to iz više različitih perspektiva, jer je svaki poslovni problem drugačiji i ima jedinstvene arhitekturalne aspekte. Ispitanici imaju slično mišljenje, navodeći da je u rešavanju problema najbitnije da arhitekta razume poslovanje ciljne organizacije (ISP13). Razumevanje poslovnih procesa ciljne organizacije, osnova je za identifikovanje arhitekturalno značajnih zahteva (ISP12).

Razvoj kompleksnih sistema zahteva postojanje formalne uloge u timu, koja se bavi identifikacijom zahteva (vlasnik proizvoda), pri čemu to ne znači da i softver arhitekta ne treba da učestvuje definisanju njihovih prioriteta (ISP16). Naprotiv, ispitanici su mišljenja da vlasnik proizvoda treba da vrši definisanje prioriteta zahteva sa aspekta vrednosti za korisnika, a arhitekta sa aspekta troškova, rizika i tehničkih zavisnosti rešenja (ISP20). Razvoj agilne arhitekture zahteva generisanje jedinstvene liste prioritetnih funkcionalnih i nefunkcionalnih zahteva sistema (ISP16).

Može se reći da je ovakvo mišljenje agilnih timova u praksi na liniji Madison-ovog (2010) pogleda na ulogu softver arhitekte, koji treba da uspostavlja balans između poslovnih i arhitekturalnih prioriteta, kako bi se proizvela agilna arhitektura.

Blair, Watt i Cull (2010) smatraju da je tesna kolaboracija softver arhitekte sa razvojnim timom ključ uspeha projekta u celini. Kruchten (2013) po tom pitanju posebno ističe značaj kolaboracije arhitekte sa poslovnim analitičarem, sa menadžerom projekta i sa programerima. Ispitanik ISP19 uspeh arhitekture vezuje za kolaboraciju arhitekte sa vlasnikom proizvoda i programerima. Takođe ističe i da promašaj bilo koje od ovih uloga može biti katastrofalan po arhitekturu, jer ukoliko vlasnik proizvoda ne razume dobro šta treba da se radi, mali promašaj u arhitekturi, od strane arhitekte, može da izazove mesece dodatnog rada. Isto tako, nedovoljno kvalitetini i nemotivisani programeri mogu dovesti u pitanje održivost arhitekturalnog rešenja (ISP19)

Babar (2009b) naglašava značaj interakcije softver arhitekte sa klijentima, u cilju identifikovanja zahteva, njihovih prioriteta i generisanja artifakta Software Architectural

Overall Plan (u nastavku SAOP). Buschmann (2012) kao ključni faktor uspeha razvoja arhitekture ističe aktivno učešće svih stejkholdera. Rezultati empirijskog istraživanja pokazuju, da softver arhitekte ne samo da imaju interakcije sa stejkholderima, već su često prinuđene i da im pomažu prilikom identifikacije arhitekturalno značajnih zahteva. Razlog je što stejkholderi uglavnom ne znaju šta hoće, ili ako i znaju šta hoće, ne mogu da sagledaju implikacije traženih zahteva. Tu je krucijalna uloga arhitekte koji na osnovu zahteva treba da prepozna drastične promene u pristupu i u samoj arhitekturi, te da klijentu saopšti šta je izvodljivo, a šta ne (ISP10). Drugim rečima, dužan je da stejkholderima predoči vrednost koju određeno arhitekturalno rešenje obezbeđuje korisniku, ali i troškove i rizike koje ono podrazumeva (ISP16). Odgovornost za izbor arhitekturalnog rešenja uglavnom je na klijentu (ISP3). Ukoliko klijent insistira na razvoju rizičnog rešenja softver arhitekta tada vodi računa da isto lako može prebaciti na ono koje je po njemu bolje (ISP1).

Empirijski rezultati pokazuju, da je u organizacijama sa velikim brojem timova, odgovornost za razvoj arhitekture divezifikovana na više različitih uloga arhitekte. Pored softver arhitekte, koji sa timovima radi na samim detaljima softverske arhitekture, postoji i uloga sistem arhitekte, koji je odgovoran za postavku arhitekture celog proizvoda. Arhitekta rešenja je uloga koja je odgovorna za implementaciju arhitekturalnog rešenja u ciljnoj organizaciji (ISP17).

Za razliku od Babar-a (2009b) koji je ulogu arhitekte rešenja vezao za odgovornosti fokusirane na menadžerske aspekte, ispitanici u praksi ove odgovornosti dodeljuju sistem arhitekti, dok ulogu arhitekte rešenja ispitanici vezuju za tim koji isporučuje softversko rešenje. Babar (2009b) je identifikovao i ulogu arhitekta implementacije, koja ima odgovornost da prati implementaciju korisničkih priča, pruža tehničko mentorstvo programerima i sagledava da li refaktoring ima negativne efekte. Ispitanici za ove odgovornosti vezuju ulogu softver arhitekte.

Rezultati istraživanja upućuju na zaključak da vlasnici arhitekture (sistem arhitekta, softver arhitekta, arhitekta rešenja i dr.) na kompleksnim projektima, moraju posedovati vrhunsko tehničko znanje i solidno znanje poslovnog domena. Odgovornosti vlasnika arhitekture su sledeće:

- Da na početku projekta, zajedno sa klijentom, identifikuje inicijalne arhitekturalno značajne zahteve.
- Da postavi inicijalno arhitekturalno rešenje za glavni deo softvera.
- Da kontinuirano sagledava arhitekturu u smislu ispunjenosti nefunkcionalnih zahteva sistema (putem kontinuirane integracije koda, seta metrika i testova).
- Da ima dobru kolaboraciju sa članovima tima u cilju deljenja ideja i rešavanja problema.
- Da vodi tehničke diskusije sa ostalim članovima tima.
- Da uči ostale članove tima i pruža mentorstvo u rešavanja arhitekturalnih problema tokom faze implementacije.

- Da razume postojeću infrastrukturu, standarde i tehnička rešenja ciljnih organizacija.
- Da poznaje mogućnosti i ograničenja što većeg broja tehnologija.
- Da prati trend u smislu mogućih opcija arhitekturalnog rešenja.
- Da obezbedi vidljivost arhitekturalnih zahteva na listi zadataka proizvoda.
- Da od početka projekta upravlja tehničkim dugom.
- Da pravovremeno donosi odluke.

Kruchten (2009) je identifikovao set osobina koje treba da poseduje softver arhitekta: dobar vizionar (kako bi sagledao i postavio sliku sistema u globalu), sposoban za donošenje odluka, vešt komunikator, sposoban za rešavanje arhitekturalnih problema koje programeri ne mogu da reše. Ispitanici su pored ovih, istakli i značaj njegovih liderskih veština u timu (ISP8), kao i moć apstrakcije (ISP5).

4.2.1.3 Kategorija eksplicitne arhitekturalne aktivnosti

Kategorija *eksplicitne arhitekturalne aktivnosti* sadrži brojne koncepte koji su grupisani u 3 podkategorije: *up front odluke na početku projekta, up front odluke pre razvoja release-a i arhitekturalno dokumentovanje.* Analiza ovih kategorija i njihovih koncepata predstavljala je ključni input za uzradu upitnika drugog kruga istraživanja. Cilj upitnika bio je da kvantifikuje značajnost identifikovanih eksplicitnih arhitekturalnih aktivnosti.

Diskusija kvalitativnih rezultata vezanih za ovu kategoriju, sprovešće se u korelaciji sa dobijenim kvantitativnim podacima koji se na nju odnose. Integracijom kvantitativnih i kvalitativnih rezultata, vezanih za eksplicitne arhitekturalne aktivnosti, biće dat odgovor na prvo istraživačko pitanje i stoga se njihov opis i diskusija nalaze u 5. poglavlju rada: *Rezultati empirijskih istraživanja, interpretacija i evaluacija razvijenog metodološko radnog okvira.*

4.2.2 Rezultati drugog kruga istraživanja

Najznačajniji rezultat drugog kruga istraživanja predstavljaju arhitekturalne prakse koje su ispitanici (sa nivoom saglasnosti >=70%) ocenili kao značajne u razvoju kompleksnih poslovnih softverskih rešenja. Dobijeni rezultati su korisni za davanje odgovora na postavljena istraživačka pitanja, te će se o njima opsežnije diskutovati u 5. poglavlju doktorske disertacije.

Pitanja iz upitnika koja nisu zadovoljila potrebni nivo saglasnosti od 70% (tabela 4.5), bilo je neophodno ponovo ispitati i oceniti njihovu značajnost u okviru trećeg kruga istraživanja, te iz tog razloga neće biti razmatrana sada.

Ocenjena značajnost pojedinih pitanja i njihovih stavki, iz upitnika drugog kruga, prikazana je putem pokazatelja proporcije ocenjivača koji dato pitanje/stavku ocenjuju kao značajnu. Način izračunavanja ovog pokazatelja opisan je u prethodnom poglavlju rada.

Tabela 4.9 Značajnost pojednih stavki 29. pitanja: ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa

Stavke	Proporcija ocenjivača koji stavku ocenjuju kao značajnu
Formiranje odgovarajućeg tima i izbor softver arhitekte u odnosu na problem koji se rešava	0.95
Razumevanje poslovnog problema	0.95
Code review	0.95
Aktivne diskusije sa stejkholderima u cilju analize zahteva i razumevanja biznisa	0.9
Identifikovanje arhitekturalno značajnih zahteva	0.9
Analiza rizika, kako bi se identifikovale i izolovale oblasti kompleksnosti	0.9
Istraživanje tehnologije pogodne za implementaciju	0.9
Identifikacija i definisanje osnovnih struktura (modula) za core sistema i njihovih veza	0.9
Testiranje performansi sistema ili dr. kritičnih nefunkcionalnih zahteva	0.9
Identifikovanje granica projekta	0.85
Analiza zavisnosti fukcionalnih zahteva od arhitekturalnih elemenata u planiranju relese-a	0.85
Kontinuirana podrška arhitekte kroz rešavanje ključnih pitanja dizajna tokom razvoja	0.85
Upravljanje konfiguracijom	0.85
Formiranje zajedničke liste prioriteta funkcionalnih i nefunkcionalnih zahteva	0.8
Definisanje osnovne arhitekture podataka	0.8
Razmatranje i postavka deployment modela	0.8
Validacija kritičnih arhitekturalnih zahteva i koncepata dizajna praksom razvoja prototipova	0.8
Specifikacija testova integracije	0.8
Identifikovanje ključnih stejkholdera sistema	0.75
Formalni review arhitekture	0.75
Specifikacija test case-ova	0.75
Planiranje release-a sa strategijom razmatranja legacy sistema, zavisnosti od drugih partnerskih ili third party proizvoda i backward compatibility podataka	0.7
Specifikacija testova prihvatljivosti	0.7
Kreiranje/razmatranje QA testova	0.7

Tabela 4.10 Značajnost pojednih stavki 26. pitanja: koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja?

Stavke	Proporcija ocenjivača koji praksu ocenjuju kao značajnu
Kao input za kasniji dizajn sistema i razvojne aktivnosti	0.9
Da dokumentuje tačke fleksibilnosti ili ograničenja sistema za buduće zahteve	0.9
Za analizu i procenu alternativnih arhitekturalnih rešenja	0.85
Da pomogne planiranje tranzicije sa postojećeg sistema na novi	0.85
Za operativnu i infrastrukturnu podršku; upravljanje konfiguracijom i prepravkama; redizajn i održavanje sistema, podsistema i komponenti	0.8
Kao specifikacija za grupu sistema koji dele set funkcija	0.75
Da podrži skaliranje agilnih praksi na velikim i kompleksnim projektima	0.75
Da uspostavi kriterijume za sertifikaciju implementacije u cilju usklađenosti sa postavljenom core arhitekturom sistema	0.7

Tabela 4.11 Značajnost pojednih stavki 27. pitanja*: ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta*

Stavke	Proporcija ocenjivača koji praksu ocenjuju kao značajnu
Smanjuje tehnički rizik, jer tim ima viziju koja ga vodi	0.9
Rano identifikovanje najkritičnijih nefunkcionalnih zahteva	0.9
Poboljšava produktivnost, anticipacijom kritičnih tehničkih pitanja umanjujući rizik kretanja razvoja pogrešnim putem	0.75
Smanjuje vreme razvoja u smislu izbegavanja velikih i skupih prepravki	0.75
Poboljšava integraciju razvoj/operacije (DevOps)	0.75
Skaliranje agilnog razvoja softvera	0.75

Tabela 4.12 Značajnost pojednih stavki 4.pitanja: *koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti*

Stavke	Proporcija ocenjivača koji stavke ocenjuju kao značajne
Razmatranje postojećih arhitekturalnih rešenja	0.95
Istraživanje odgovarajućeg frejmvorka za implementaciju	0.8
Razmatranje postojeće infrastrukture u ciljnoj organizaciji	0.8
Istraživanje third party biblioteka	0.75
Istraživanje tržišta i širenje znanja iz domena problema u cilju bolje identifikacije arhitekturalnih zahteva	0.7

Tabela 4.13 Značajnost pojednih stavki 21.pitanja: *koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture*

Stavke	Proporcija ocenjivača koji stavke ocenjuju kao značajne
Mišljenja architectural board-a (eksperti za razne aspekte: stručnjaci za deployment, stručnjaci za sigurnost, za kvalitet i druge razne aspekte arhitekture)	0.85
Scenario analiza interakcije stejkholdera sa sistemom, sa fokusom na nefunkcionalne zahteve	0.85
Arhitekturalni spike-ovi	0.8
Prototipovi	0.8
Regresioni testovi	0.75
Simulacija opterećenja buduće arhitekture sistema	0.75
Vremenski ograničen proof of concept	0.7

Tabela 4.14 Značajnost pojednih stavki 14.pitanja: da li je prilikom planiranja release-a značajno sprovoditi sledeće aktivnosti

Stavke	Proporcija ocenjivača koji stavke ocenjuju kao značajne
1. Analizu međusobnih zavisnosti funkcionalnosti	1
2. Analizu zavisnosti funkcionalnih i nefunkcionalnih zahteva	0.7

Tabela 4.15 Značajnost pitanja iz upitnika koja ne sadrže stavke

Pitanje	Proporcija ocenjivača koji pitanje ocenjuju kao značajno
P3: Koliko je za kvalitet softverske arhitekture važno uključivanje softver arhitekte kroz ceo razvojni proces i njegova tesna kolaboracija sa razvojnim timom uz kontinuirano deljenje ideja tokom trajanja projekta?	1
P8: Da li smatrate da je značajno da nefunkcionalni zahtevi sistema budu eksplicitno identifikovani?	0.9
P10: Koliko je značajno prilikom utvrđivanja prioriteta na jedinstvenoj listi zahteva, razmatrati njihove vrednosti, ne samo sa aspekta biznisa, već i sa aspekta nivoa rizika i uticaja na arhitekturu?	0.9
P2: Koliko je dobra komunikacija softver arhitekte sa stejkholderima značajna za izgradnju softverske arhitekture?	0.85
P5: Da li se slažete sa sledećim stavom: arhitekturalno značajni zahtevi su rezultat diskusija sa stejkholderima, analize funkcionalnih i nefunkcionalnih zahteva i anticipiranja budućih pravaca razvoja biznisa.	0.85
P9: Da li smatrate korisnim pravljenje jedinstvene liste (product backlog-a celog proizvoda) funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena?	0.85
P13: Da li je značajno planiranje arhitekture iznad jednog release-a, u cilju identifikovanja i razvoja arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera?	0.85
P18: Da li je pre početka novog release-a potrebno eksplicitno razmatrati da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a?	0.85

P20: Da li sprovođenje kontinuiranog review-a arhitekture može, pored standardnih agilnih praksi (kontinuirano testiranje, kontinuirana analiza koda, kontinuirana integracija koda, kontinuirano refaktorisanje i programiranje u paru) obezbediti dodatnu kontinuiranu kontrolu kvaliteta?	0.85
P7: Da li je bitno unapred identifikovati tačke arhitekture koje trebaju biti fleksibilnije u cilju budućih promena i odluka u kontekstu proširivosti softvera?	0.8
P6: Koliko je za arhitekturu značajno identifikovanje budućih ciljeva i promena, pravaca razvoja biznisa?	0.75
P19: Da li je značajno unapred (tokom tekućeg release-a) identifikovati potrebne arhitekturalne izmene pre implementacije narednog release-a?	0.7
P30: Koliko je značajno eksplicitno identifikovati zahteve koji se odnose na geografsku dislociranost delova budućeg sistema?	0.7

Nekolicina pitanja iz upitnika davala su mogućnost višestrukog izbora ispitanika (putem čekliste), te će narednim tekstom biti diskutovani dobijeni kvantitativni rezultati koji se odnose na ovaj tip pitanja.

Primer rezultata kvantitativne analize ove vrste pitanja dat je u tabeli 4.16. Druga kolona date tabele prikazuje procenat učestalosti izbora ponuđenih odgovora, na sledeće pitanje iz upitnika:

Da bi razumeli biznis i poslovne procese, identifikovali zahteve značajne za arhitekturu i rešili eventualne kontradiktornosti i konflikte stejkholdera, arhitekte treba da imaju:

- Direktnu i kontinuiranu komunikaciju sa stejkholderima sistema (37.5% odgovora).
- Da zajedno sa biznis analitičarem povremeno budu uključeni u komunikaciju sa stejkholderima (50% odgovora).
- Kontinuirano komuniciraju samo sa biznis analitičarem.(12.5% odgovora).

Rezultati pokazuju da je 50% ispitanika mišljenja da softver arhitekta treba povremeno da bude uključen u komunikaciju sa stejkholderima i to zajedno sa biznis analitičarem, a 37.5% ispitanika je mišljenja da softver arhitekta treba da ima direktnu i kontinuiranu komunikaciju sa stejkholderima. Ovi rezultati impliciraju zaključak da ukupno 87.5% ispitanika smatra da je direktna uključenost softver arhitekte u razumevanje problema i identifikaciju arhitekturalno značajnih zahteva korisnija nego da iste dobija posrednim putem, od biznis analitičara. Razlozi za ovakav stav mogu se najbolje sagledati u izjavi sledećeg ispitanika:

ISP7: "...mislim da to prođe kroz više slojeva dok konačno ne dođe do arhitekte i nikada se ne dobiju potpune informacije, jer se prenošenjem od sektora do sektora nešto izgubi. Nikad to ne bude ni lepo dokumentovano. Tako da je po meni najbolje da arhitekta učestvuje u sastancima, da čuje pravo od klijenta koji su tačno problemi...".

Nalazi su u skladu i sa literaturom u kojoj se ova eksplicitna arhitekturalna aktivnost ističe kao neophodna u agilnim procesima (Friedrichsen, 2014).

Tabela 4.16 Učestalost odgovora na 1. pitanje iz upitnika

		Responses		Percent of Cases
		N	Percent	
$P1[a]	P1.1	9	37,5%	45,0%
	P1.2	12	50,0%	60,0%
	P1.3	3	12,5%	15,0%
Total		24	100,0%	120,0%

Na pitanje iz upitnika: *kada je u agilnom procesu potrebno sprovoditi review arhitekture?,* ispitanici su odgovorili na sledeći način: najveći broj ispitanika (42.9%) je mišljenja da je review arhitekture potrebno sprovoditi na kraju/početku svake iteracije, a svega 28.6% ispitanika smatra da je review arhitekture potrebno vršiti nakon implementacije seta funkcionalnosti i pre početka novog release-a. Iz ovoga se može zaključiti da je najzastupljenije mišljenje da arhitekturu treba kontinuirano sagledavati, kako bi se na vreme uočili problemi u kvalitetu dizajna i sproveo neophodni refaktoring.

Na pitanje iz upitnika: *kada se treba fokusirati na softversku arhitekturu u kontekstu agilnog razvoja?*, ispitanici su imali sledeće ponuđene opcije: "nikada", "uvek" "kada je sistem kompleksan". Kvantitativnom analizom odgovora utvrđeno je da 90 % ispitanika smatra da se na arhitekturu treba fokusirati "uvek", a svega 10% - samo u slučaju "kada je sistem kompleksan". Dobijeni rezultati se bitno razlikuju od rezultata do kojih su došli Falessi i kolege (2010). Naime, njihovi rezultati pokazuju da je 50% ispitanika mišljenja da se na arhitekturu treba fokusirati "kada je sistem kompleksan".

Razlog ovako različitih nalaza istraživanja može biti u profilu učesnika istraživanja. Fallesi i kolege su ispitivali programere, dok je ovo istraživanje uključivalo eksperte na poziciji softver arhitekte ili ljude na upravljačkim pozicijama, koji su ranije u svojoj karijeri bili na poziciji softver arhitekte. Komparacijom dobijenih rezultata i učesnika istraživanja, zaključak koji se može izvesti jeste, da profil programera ima drugačiju svest i znanje o važnosti softverske arhitekture. Drugi zaključak koji se nameće, jeste da razvojni timovi u IBM-u (gde su Fallesi i kolege sprovele istraživanje) imaju u većoj meri implementirane agilne principe i vrednosti. Razlog za to je što IBM ima "in hous" razvoj i razvijaju sopstveni proizvod, dok agilni timovi u Srbiji najvećim delom učestvuju na "outsourcing" projektima, što umanjuje mogućnost razvoja softvera na potpuno agilan način.

Razlika u dobijenim rezultatima istraživanja vezana je i za pitanje: *šta utiče na kompleksnost projekta...,* pri čemu je 15.8% ispitanika ovog istraživanja odgovorilo da je to "velik broj komponenti i njihovih međusobnih odnosa", 12.5% da je to"interakcija sa drugim sistemima i spoljnim postojećim servisima", a 13.3% - "kompleksnost rešenja". Falessi i kolege (2010)

došli su do nalaza da 33% programera smatra da kompleksnost potiče od "broja zahteva i linija koda", dok je u ovom istraživanju samo 9.2% ispitanika bilo takvog mišljenja. Falessi i kolege su na drugo mesto stavili "broj stejkholdera" (sa 29%), dok je u ovom istraživanju svega 5.8% eksperata odabralo taj odgovor. Treći faktor, koji prema Fallesi-ju i kolegama utiče na kompleksnost (sa 19%), jeste "geografska distribuiranost", dok ispitanici ove doktorske disertacije dati faktor ocenjuju sa 6.7%.

Treća grupa rezultata drugog kruga istraživanja Delfi metodom, bila je značajna sa aspekta sagledavanja stepena, tj. proporcije međusobnog slaganja ispitanika. Rezultati su dobijeni izračunavanjem vrednosti Cohen-ovog kappa koeficijenta (deo rezultata dat u prilogu 13), a prikazani su u tabeli 4.17. Analizom statistički značajnih vrednosti kappa koeficijenta iz tabele 4.17 (vrednosti sa "*" u eksponatu broja), može se zaključiti da je najveći stepen međusobnog slaganja postojao između sledećih ispitanika: ISP9 i ISP16; ISP7 i ISP10; ISP10 i ISP16; ISP1 i ISP12, dok je najmanji stepen međusobnog slaganja bio između ispitanika: ISP10 i ISP8; ISP17 i ISP5; ISP16 i ISP7.

Tabela 4.17 Vrednosti Cohen-ovog kappa koeficijenta

	ISP1	ISP17	ISP15	ISP14	ISP11	ISP9	ISP12	ISP3	ISP16	ISP20	ISP4	ISP5	ISP18	ISP6	ISP7	ISP19	ISP2	ISP10	ISP8	ISP13
ISP1		-,158	,396	,127	,431*	,233	**,585***	,182	,233	,313	-,023	,020	,377	,052	,225	,000	-,222	,421*	,455*	,000
ISP17			,179	-,015	,214	,241	,298	-,182	,241	,132	,241	**,353***	-,170	,018	-,065	,000	,327	-,100	-,165	,000
ISP15				-,299	,139	,025	,466*	,091	,304	,396	,025	,320	-,175	,139	,279	,000	-,205	,507*	,093	,000
ISP14					,098	-,031	,113	,182	-,031	-,048	-,031	-,144	,290	,279	,076	,000	,233	-,015	-,144	,000
ISP11						,228	,441*	,000	,035	,052	,228	,342	,068	,083	-,090	,000	-,071	,018	,154	,000
ISP9							,340	-,182	**,694***	,488*	-,222	,172	-,100	-,158	-,078	,000	,154	,241	,172	,000
ISP12								,000	,340	,585	-,100	,294	,214	,068	,154	,000	-,021	,298	,294	,000
ISP3									,000	,000	,182	,273	,000	,182	,091	,000	-,091	,182	,091	,000
ISP16										,488*	-,222	,172	-,100	,035	**,353***	,000	,154	**,621***	,172	,000
ISP20											-,023	,020	,170	-,138	,225	,000	-,222	,421*	,238	,000
ISP4												,172	,120	,035	-,078	,000	,154	-,138	-,065	,000
ISP5													-,312	-,222	-,086	,000	,011	,094	-,048	,000
ISP18														,068	,154	,000	-,021	,064	,495*	,000
ISP6															,108	,000	,124	,018	-,222	,000
ISP7																,000	-,073	**,645***	,185	,000
ISP19																	,000	,000	,000	,
ISP2																		-,122	-,236	,000
ISP10																			**,353***	,000
ISP8																				,000
ISP13																				

Napomena: " * " u eksponatu broja označava da je ta vrednost kappa koeficijenta statistički značajna, na nivou 0.05.

4.2.3 Rezultati trećeg kruga istraživanja

Rezultati istraživanja trećeg kruga Delfi metodom, pokazuju kako su eksperti ocenili značajnost pitanja/stavki za koje nije postojao potreban nivo saglasnosti u prethodnom krugu istraživanja i prikazani su tabeli 4.18 (deo rezultata kvantitativne analize dat je u prilogu 14). Prema proceduri opisanoj u poglavlju rada 4.1.3.4, na osnovu rezultata kvantitativnih analiza, izračunata je proporcija ocenjivača koji su dato pitanje/stavku iz upitnika ocenili kao značajnu.

Analizom rezultata prikazanih u tabeli 4.18, zaključuje se da za svega jednu stavku (varijablu), od početnih 115 stavki (varijabli), nije postignut potreban nivo saglasnosti eksperata od 70%. Ovaj podatak implicira zaključak da ne postoji potreba za sprovođenjem dodatnog kruga istraživanja.

Tabela 4.18 Značajnost pitanja/stavki iz upitnika trećeg kruga

Rbr. pitanja	Rbr. stavke	Pitanje	Proporcija ocenjivača koji pitanje/stavku ocenjuju kao značajnu
29.	13.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema	0.95
26.	1.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju među organizacijama koje su uključene u razvoj, produkciju, operativne aktivnosti i održavanje sistema	0.9
15.		Koliko je značajno pre implementacije release-a utvrditi zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti?	0.9
21.	3.	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Test case-ovi	0.9
26.	6.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži pregled (review), analizu i evaluaciju sistema	0.9
29.	18.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem interreaguje tokom release-a	0.9
21.	6.	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Statička analiza koda u cilju analize struktura i pravila kodiranja	0.9
22.		Da li smatrate korisnom strategiju Test Driven Development sa fokusom na nefunkcionalne zahteve?	0.9

11.		Da li je značajno obezbediti vidljivost arhitekturalno značajnih zahteva i testova prihvatljivosti kroz postavku arhitekturalnih zadataka na listi zadataka (backlogu) ili Kanban tabli kroz ceo proces razvoja?	0.85
23.		Ocenite značaj strategije koja momenat donošenja i sprovođenja arhitekturalnih odluka zasniva na proceni relativnih troškova, sa jedne strane - usled kašnjenja implementacije i troškova usled eventualne dorade ili redizajna rešenja u fazi implementacije.	0.85
26.	3.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za dokumentovanje osnovnih postavki sistema, njegove namene i okruženja	0.85
26.	16.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži pripremu dokumentacije za akviziciju	0.85
26.	15.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži planiranje sistema i budžetske aktivnosti	0.85
27.	4.	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Poboljšava svesnost razvojnog tima o vezi sistema i organizacije koja ga implementira (njenih ciljeva, infrastrukture, strategije i dr.)	0.85
29.	12.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Generisanje top level dokumentacije	0.85
16.		Da li je značajno sprovoditi estimaciju veličine tehničke korisničke priče i brzine njenog razvoja?	0.8
27.	7.	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Poboljšava komunikaciju, prezentujući stejkholderima šta će se graditi i na koji način	0.75
27.	3.	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Izbegavanje tehničkog duga	0.75
28.		Koliko je značajno da ceo arhitekturalni proces tokom razvoja bude praćen nekim od alata (Jira ili sl.)?	0.7
29.	16.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji	0.6
12.		Da li ceo projekat ugrožen usled razmatranja arhitekture na nivou samo jedne iteracije (sprinta)?	0.25
26.	5.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju sa stejkholderima sistema (klijentima, korisnicima, integratorima i dr.)	0.2
26.	17.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Kao input za izbor generatora sistema i alata za analizu	0.2
29.	22.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje detaljnog dizajna svakog modula	0.15

29.	24.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Razmatranje detaljnog dizajna i korekcije	0.15
26.	11.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Sredstvo razvoja i održavanja dokumentacije	0.15
29.	23.	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Dokumentovanje detaljnog dizajna	0.1
26.	14.	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju između klijenata, korisnika i programera, kao deo dogovorenog ugovora	0.1

Rezultati prikazani u tabeli 4.18 pokazuju da jedino stavka: *upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji,* nema potrebni nivo saglasnosti eksperata tj. niti je ocenjena kao "značajna" niti kao "beznačajna". Razlozi mogu biti ili da eksperti nisu razumeli svrhu pitanja ili je sama formulacija pitanja nejasna. Utemeljenja za ovakve razloge mogu se naći u komentaru ispitanika ISP14 koji je ovu praksu ocenio kao "nije značajna": *"Tim kontinuirano radi na uklanjanju tehničkog duga"*. Pitanje je upravo imalo za cilj da naglasi kontinuiranost upravljanja tehničkim dugom, zbog čega je i naglašeno da se sprovodi u svakoj iteraciji, koja u agilnim procesima traje od 2-3 nedelje. Navedeno tumačenje rezultata potkrepljuju i mišljenja sledećih ispitanika:

ISP20: "...ovo je slika i prilika naše kompanije...ovde se tehnički dug ignoriše, a sa njim i nefunkcionalni zahtevi".
ISP18: "...upravljanje tehničkim dugom je značajno kod razvoja kompleksnih sistema, zbog problema koji nastaju ako se ne upravlja".
ISP16: "...to je vrlo bitno kod složenih sistema, gde se sva složenost i ne može detektovati u samom startu dizajna/razvoja".

Zanimljivo je takođe, ovo pitanje uporediti sa pitanjem: *ocenite sledeće benefite od inicijalne postavke arhitekture na početku agilnog projekta: izbegavanje tehničkog duga.....,* koje je ocenjeno kao značajno, sa proporcijom ocenjivača od 0.75. Njihovom komparacijom mogao bi se izvući pogrešan zaključak da se up front arhitekturalnim aktivnostima na početku projekta izbegava tehnički dug i da nije potrebno njime upravljati tokom svih iteracija u toku razvoja, što je u suprotnosti i sa tvrđenjem ispitanika ISP14 koji naglašava potrebu kontinuiteta u uklanjanju tehničkog duga. U prilog ovakvog tumačenja rezultata je i podatak iz intervjua:

ISP1: "...najveći problem u bivšoj firmi je bio taj što smo imali veliki tehnički dug, to je dug koji smo napravili ad hoc razvojem, gde nije bilo pravila, gde nije bilo nikakve metodologije razvoja, gde smo samo nabacivali lopatom kod i onda smo tako napravili gomilu bagova. Niko nije vodio računa o dizajnu, niko nije vodio računa o raznim stvarima i onda kad je došlo vreme da se radi agilni razvoj, onda smo uvideli koliko tu ima problema, pa smo malo usporili sa celom tom mašinerijom, da bi se malo više posvetili

kvalititu. Onda smo naišli na razne probleme, gde kao što kažete svašta treba da se refaktoriše. Uvideli smo da ne valja uglavnom kostur cele arhtekture, što je onda impliciralo veće troškove. Mi smo u principu tada ulagali u kvalitet i u bolji kod, da je više čitljiv, održiv, da je maintainability mnogo bolji i da u neku ruku poboljšamo i performanse. Funkcionalnosti tada nismo razvijali...".

Poslednjih osam stavki iz tabele 4.18 eksperti su ocenili kao "beznačajne". Četiri stavke odnose se na pitanja o nameni softverske arhitekture, a četiri na značajnost pojedinih arhitekturalnih praksi u razvoju kompleksnih poslovnih softverskih rešenja. Može se zaključiti da eksperti smatraju da su arhitekturalne aktivnosti, koje se odnose na definisanje i dokumentovanje detaljnog dizajna, "beznačajne" sa aspekta arhitekture. Odnosno, da eksperti aktivnosti vezane za detaljan dizajn posmatraju kao zadatak programera, a ne softver arhitekte, što potkrepljuju i mišljenja sledećih eksperata:

ISP16: "...ako smo stvarno agilni imamo i developere kojima ne treba detaljni dizajn...".
ISP3: "...u agilnim metodama uglavnom je arhitektura i planiranje na visokom nivou, znači ne ide se u taj nivo detalja u koji bi se išlo u tradicionalnom razvoju, gde ipak ima više vremena i gde se veći akcenat stavljaju na to...".
ISP4:"...ranije je bila intenca da vi imate nekog ko napravi dizajn, a onda neko ko iskodira. Sad se to prosto ne radi, jer ja očekujem od mojih developera da poznaju dizajn paterne, da su u stanju da prepoznaju, radimo kod review, ali niko im neće dizajnirati, u smislu da će im nacrtati class dijagram, prosto je to suviše sporo...".

Zanimljiv rezultat dobijen je i za pitanje*: da li je ceo projekat ugrožen usled razmatranja arhitekture na nivou samo jedne iteracije (sprinta)?* Eksperti su sa 75% saglasni da je ova aktivnost "beznačajna", što nije u skladu sa nalazima u literaturi. Ispitanik koji je ovu aktivnost ocenio kao "značajna", svoj odgovor je obrazložio na sledeći način: *"...ukoliko se arhitektura razmatra samo na nivou sprinta, za veće proizvode može da napravi probleme arhitekture globalnog rešenja" (ISP11).* Na osnovu ovog komentara možemo zaključiti da bi preciznija formulacija pitanja, u smislu ugrožavanja arhitekture, a ne celog projekta, proizvela drugačije odgovore.

Drugi značajan rezultat, trećeg kruga istraživanja Delfi metodom, dobijen je primenom indeksa induvidualne stabilnosti, koji je pokazatelj ustaljenosti odgovora ispitanika od drugog do trećeg kruga istraživanja. Stoga se, pored kriterijuma postignutog nivoa saglasnosti eksperata od 70%, koristio i kao dodatni kriterijum za donošenje odluke o obustavljanju iteracija istraživanja. Naime, ukoliko je indeks individualne stabilnosti imao vrednosti >0.5 moglo se zaključiti da je stabilnost prihvatljiva i da nema potrebe za narednim krugom istraživanja. Drugim rečima, što je vrednost indeksa bila bliža vrednosti 1, to su ocene nekog pojedinačnog eksperta, između dve iteracije Delfi tehnike, bile stabilnije.

Analizom rezultata iz tabele 4.19 zapaža se da svega 8 varijabli, od početnih 115, ne zadovoljava ovaj uslov. To je impliciralo zaključak da rezultati indeksa individualne stabilnosti potvrđuju da nema potrebe za novim iteracijama istraživanja.

Tabela 4.19 Indeksi individualne stabilnosti

Rbr. pitanja i stavke	Pitanje/stavka pitanja	indeks induvidualne stabilnosti
29.13	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema	0,30
26.06	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži pregled (review), analizu i evaluaciju sistema	0,33
22	Da li smatrate korisnom strategiju Test Driven Development sa fokusom na nefunkcionalne zahteve?	0,36
29.18	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem interreaguje tokom release-a	0,36
26.01	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju među organizacijama koje su uključene u razvoj, produkciju, operativne aktivnosti i održavanje sistema	0,44
26.15	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži planiranje sistema i budžetske aktivnosti	0,44
29.12	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Generisanje top level dokumentacije	0,46
29.23	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Dokumentovanje detaljnog dizajna	0,46
15	Koliko je značajno pre implementacije release-a utvrditi zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti?	0,50
26.14	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju između klijenata, korisnika i programera, kao deo dogovorenog ugovora	0,50
29.16	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji	0,50
26.16	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Da podrži pripremu dokumentacije za akviziciju	0,5
21.03	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Test case-ovi	0,54
21.06	Koja od sledećih tehnika je značajna i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: Statička analiza koda u cilju analize struktura i pravila kodiranja	0,54
26.03	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za dokumentovanje osnovnih postavki sistema, njegove namene i okruženja	0,54
29.24	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Razmatranje detaljnog dizajna i korekcije	0,57
26.17	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Kao input za izbor generatora sistema i alata za analizu	0,58

23	Ocenite značaj strategije koja momenat donošenja i sprovođenja arhitekturalnih odluka zasniva na proceni relativnih troškova, sa jedne strane usled kašnjenja implementacije i troškova usled eventualne dorade ili redizajna rešenja u fazi implementacije.	0,60
12	Da li ceo projekat ugrožen usled razmatranja arhitekture na nivou samo jedne iteracije (sprinta)?	0,61
26.11	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Sredstvo razvoja i održavanja dokumentacije	0,61
27.04	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Poboljšava svesnost razvojnog tima o vezi sistema i organizacije koja ga implementira (njenih ciljeva, infrastrukture, strategije i dr.)	0,61
27.03	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Izbegavanje tehničkog duga	0,64
11	Da li je značajno obezbediti vidljivost arhitekturalno značajnih zahteva i testova prihvatljivosti kroz postavku arhitekturalnih zadataka na listi zadataka (backlogu) ili Kanban tabli kroz ceo proces razvoja?	0,69
26.05	Koliko je značajna svaka od sledećih namena softverske arhitekture, u razvoju kompleksnih poslovnih softverskih rešenja: Za komunikaciju sa stejkholderima sistema (klijentima, korisnicima, integratorima i dr.)	0,70
27.07	Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: Poboljšava komunikaciju, prezentujući stejkholderima šta će se graditi i na koji način	0,70
16	Da li je značajno sprovoditi estimaciju veličine tehničke korisničke priče i brzine njenog razvoja?	0,73
29.22	Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: Definisanje detaljnog dizajna svakog modula	0,75
28	Koliko je značajno da ceo arhitekturalni proces tokom razvoja bude praćen nekim od alata (Jira ili sl.)?	0,77

Analizom ovih osam stavki iz tabele 4.19, zapaža se da se svega jedna stavka poklapa sa rezultatima iz tabele 4.18: *ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: dokumentovanje detaljnog dizajna,* gde je ona ocenjena kao beznačajna. Vrednost indeksa induvidualne stabilnosti od 0.46 i pokazatelji proporcije značajnosti u drugom krugu (0.4) i u trećem (0.1) pokazuju da je već u prvom krugu preovladavalo mišljenje eksperata da dokumentovanje detaljnog dizajna ne spada u važnu arhitekturalnu praksu koju treba inkorporirati u agilne procese razvoja. Nakon trećeg kruga istraživanja, trend je bitno naglašeniji, jer je većina eksperata promenila svoj originalni odgovor ukoliko se razlikovao od grupnog. O razlozima niskog vrednovanja ove prakse govori mišljenje sledećeg ispitanika:

ISP14: "..u agilnom razvoju softwera cenimo komunikaciju ispred dokumentacije. Ovo znači da treba imati što je moguće manje dokumentacije. Da bi se ovo postiglo konstantno moramo revidirati postojeću dokumentaciju izbacivati manje važne i dodavati važne stavke".

Stavka koja je u tabeli 4.18 ocenjena kao značajna i to sa proporcijom ocenjivača od 0.95, imala je najmanji indeks individualne stabilnosti (0.30): *definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema*. To pokazuje da su na ovom pitanju ispitanici u najvećoj meri promenili svoj originalni odgovor iz drugog kruga (proporcija ocenjivača iznosila 0.60) i prihvatili grupni odgovor. Međutim ispitanik ISP14 nije podržao većinsko mišljenje istakavši:

ISP14: "...tim je taj koji najbolje zna kako da razvija softver i ima sva potrebna znanja i veštine. U suprotnom ne radimo agilno".

Druga arhitekturalna aktivnost sa najmanjim indeksom individualne stabilnosti od 0.36 odnosi se na pitanje: *da li smatrate korisnom strategiju Test Driven Development sa fokusom na nefunkcionalne zahteve?*. U Tabeli 4.18 zapaža se da je i ona ocenjena kao značajna, sa proporcijom ocenjivača u trećem krugu istraživanja od 0.90, dok je u drugom krugu proporcija ocenjivača iznosila svega 0.45. Razlog za veliki procenat promene prvobitnog odgovora ispitanika dao je ispitanik ISP17: *"...korišćenjem Test Driven Development-a se čvrsto definišu nefunkcionalni zahtevi u startu i kontinualno se validiraju tokom razvoja"*.

Pitanje za koje jedino nije ostvaren nivo saglasnosti eksperata od 70% i po završetku trećeg kruga istraživanja: *ocenite značaj eksplicitne arhitekturalne aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji,* imalo je indeks individualne stabilnosti od 0.5. To ukazuje da se nije značajno promenio odgovor ispitanika iz drugog (proporcija ocenjivača značajnosti iznosila je 0.65) u treći krug istraživanja (proporcija ocenjivača značajnosti iznosila je 0.60). Diskusija ovog rezultat je data na početku ovog poglavlja.

U trećem krugu istraživanja sprovedena je i kvantitativna analiza podataka putem McNemar-ovog i McNemar-Bowker-ovog testa (deo razultata dat u prilogu 15). Cilj analize bio je da se utvrdi kod kojih ispitanika je postojala statistički značajna promena u mišljenju (vrednosti testa veće od 0.05), iz drugog u treći krug istraživanja. Dobijeni rezultati prikazani su u tabeli 4.20. i impliciraju zaključak da je kod ispitanika ISP4, ISP9, ISP5, ISP19, ISP7, ISP17 i ISP11 postojala statistički značajna promena u mišljenju iz drugog u treći krug istraživanja.

Međutim, dobijeni rezultati ne utiču na krajnje nalaze istraživanja, tj. ne ukazuju na postojanje konformiteta u mišljenju eksperata, iz razloga što je vrlo mali broj varijabli bio predmet ocene značajnosti u trećem krugu istraživanja (svega 29 od ukupno 115 varijabli). Drugim rečima, za preko 75% ocenjivanih varijabli postignut je nivo saglasnosti eksperata od 70%, već u drugom krugu istraživanja, kada ispitanici nisu imali uvid u grupno mišljenje.

Tabela 4.20 Promena u mišljenju eksperata iz drugog u treći krug istraživanja

Ekspert	McNemerov test	Statistička značajnost
ISP16	.000	1.00
ISP18	.091	0.763
ISP14	1.000	0.607
ISP1	4.000	0.406
ISP8	3.000	0.392
ISP15	2.000	0.368
ISP2	1.000	0.317
ISP3	2.000	0.157
ISP12	5.000	0.082
ISP4	4.000	0.046
ISP9	4.000	0.046
ISP5	6.500	0.039
ISP19	4.500	0.034
ISP7	10.000	0.019
ISP17	8.000	0.018
ISP11	8.333	0.004
ISP6	.	1.000*
ISP10	.	1.000*
ISP13	.	1.000*
ISP20	.	0.250*

Napomena: vrednosti sa " * " u eksponatu ukazuju na podatak da je korišćena binomna distribucija.

5.

Rezultati empirijskih istraživanja, interpretacija i evaluacija razvijenog metodološko-radnog okvira

Poglavlje sadrži prikaz i interpretaciju rezultata sprovedenog empirijskog istraživanja, prema postavljenim istraživačkim ciljevima. U kontekstu trećeg istraživačkog cilja, predstavljen je metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima, kao i njegova procena putem evaluacione studije. Poglavlje završava elaboracijom ograničenja istraživanja.

5.1 Zaključci izvedeni iz empirijskih istraživanja sa osvrtom na ekonomske implikacije

Spregom rezultata, dobijenih kvantitativnim i kvalitativnim analizama podataka, izveden je niz zaključaka, koji daju odgovor na prvo i drugo istraživačko pitanje. Diskusija rezultata sprovodiće se posebno za prvi, a potom drugi istraživački cilj rada.

5.1.1 C1: Identifikovane tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja

Navedeni cilj istraživanja, u naslovu poglavlja, vezan je za prvo istraživačko pitanje doktorske disertacije:

IP1: Da li je moguće ostvariti pozitivan uticaj na kvalitet poslovnog softverskog rešenja inkorporacijom tradicionalnih arhitekturalnih praksi u agilan proces razvoja?

Odgovor na postavljeno istraživačko pitanje je pozitivan tj. inkorporacijom tradicionalnih arhitekturalnih praksi u agilan proces razvoja, moguće je ostvariti pozitivan uticaj na kvalitet poslovnog softverskog rešenja, što je detaljno i elaborirano u nastavku poglavlja. Odgovor je proizašao na osnovu rezultata sprovedenog kvalitativno-kvantitativnog empirijskog istraživanja, opisanog u prethodnom poglavlju rada. Rezultati istraživanja, u vidu identifikovanih tradicionalnih arhitekturalnih praksi, prezentovani su u tabeli 5.1, uz prikaz proporcije ocenjivača, kojom su iste ocenjene kao značajne za razvoj softverske arhitekture u agilnim procesima razvoja. Prikazani rezultati potvrđuju realizaciju prvog cilja istraživanja.

Tabela 5.1 Empirijski identifikovane tradicionalne arhitekturalne prakse, značajne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja

Empirijski identifikovane tradicionalne arhitekturalne prakse	**Proporcija ocenjivača koji praksu ocenjuju kao značajnu**
Analiza međusobnih zavisnosti funkcionalnosti.	1
Formiranje odgovarajućeg tima i izbor softver arhitekte u odnosu na problem koji se rešava.	0.95
Razumevanje poslovnog problema.	0.95
Code review.	0.95
Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema.	0.95
Razmatranje postojećih arhitekturalnih rešenja.	0.95
Aktivne diskusije sa stejkholderima u cilju analize zahteva i razumevanja biznisa.	0.9
Identifikovanje arhitekturalno značajnih zahteva.	0.9
Analiza rizika, kako bi se identifikovale i izolovale oblasti kompleksnosti.	0.9
Istraživanje tehnologije pogodne za implementaciju.	0.9
Identifikacija i definisanje osnovnih struktura (modula) za core sistema i njihovih veza.	0.9
Testiranje performansi sistema ili dr. kritičnih nefunkcionalnih zahteva.	0.9

Utvrđivanje zajedničkih komponente i zajedničke infrastrukture seta funkcionalnosti pre implementacije release-a.	0.9
Upotreba tehnike test case-ova.	0.9
Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem interreaguje tokom release-a.	0.9
Primena tehnike statička analiza koda u cilju analize struktura i pravila kodiranja.	0.9
Test Driven Development sa fokusom na nefunkcionalne zahteve.	0.9
Eksplicitno identifikovanje nefunkcionalnih zahteva sistema.	0.9
Utvrđivanje prioriteta na jedinstvenoj listi zahteva, razmatranjem njihove vrednosti sa aspekta biznisa, ali i sa aspekta nivoa rizika i uticaja na arhitekturu.	0.9
Identifikovanje granica projekta.	0.85
Analiza zavisnosti fukcionalnih zahteva od arhitekturalnih elemenata u planiranju relese-a.	0.85
Upravljanje konfiguracijom	0.85
Postavka arhitekturalnih zadataka na listi zadataka (backlogu) ili Kanban tabli kroz ceo proces razvoja.	0.85
Momenat donošenja i sprovođenja arhitekturalnih odluka zasniva se na proceni relativnih troškova, sa jedne strane - usled kašnjenja implementacije i troškova usled eventualne dorade ili redizajna rešenja u fazi implementacije.	0.85
Generisanje top level dokumentacije.	0.85
Arhitekturalno značajni zahtevi su rezultat diskusija sa stejkholderima, analize funkcionalnih i nefunkcionalnih zahteva i anticipiranja budućih pravaca razvoja biznisa.	0.85
Izrada jedinstvene liste funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena.	0.85
Planiranje arhitekture iznad jednog release-a, u cilju identifikovanja i razvoja arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera.	0.85
Eksplicitno razmatranje, pre početka novog release-a, da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a.	0.85
Kontinuirani review arhitekture, pored standardnih agilnih praksi (kontinuirano testiranje, kontinuirana analiza koda, kontinuirana integracija koda, kontinuirano refaktorisanje i programiranje u paru) obezbeđuje dodatnu kontinuiranu kontrolu kvaliteta.	0.85
Formiranje zajedničke liste prioriteta funkcionalnih i nefunkcionalnih zahteva.	0.8
Definisanje osnovne arhitekture podataka.	0.8
Razmatranje i postavka modela raspoređivanja.	0.8
Validacija kritičnih arhitekturalnih zahteva i koncepata dizajna praksom razvoja prototipova.	0.8
Specifikacija testova integracije.	0.8
Estimacija veličine tehničke korisničke priče i brzine njenog razvoja.	0.8
Istraživanje odgovarajućeg frejmvorka za implementaciju.	0.8

Razmatranje postojeće infrastrukture u ciljnoj organizaciji.	0.8
Identifikovanje tački arhitekture koje trebaju biti fleksibilnije u cilju budućih promena i odluka u kontekstu proširivosti softvera.	0.8
Arhitekturalni spike-ovi	0.8
Prototipovi	0.8
Evaluacija arhitekture od strane arhitekturalnog odbora	0.85
Scenario analiza interakcije stejkholdera sa sistemom, sa fokusom na nefunkcionalne zahteve	0.85
Identifikovanje ključnih stejkholdera sistema.	0.75
Formalni review arhitekture.	0.75
Regresioni testovi	0.75
Simulacija opterećenja buduće arhitekture sistema	0.75
Specifikacija test case-ova.	0.75
Istraživanje third party biblioteka.	0.75
Identifikovanje budućih ciljeva i promena, pravaca razvoja biznisa u cilju razvoja arhitekture.	0.75
Planiranje release-a sa strategijom razmatranja legacy sistema, zavisnosti od drugih partnerskih ili third party proizvoda i backward compatibility podataka.	0.7
Specifikacija testova prihvatljivosti.	0.7
Kreiranje/razmatranje QA testova.	0.7
Vremenski ograničen proof of concept	0.7
Istraživanje tržišta i širenje znanja iz domena problema u cilju bolje identifikacije arhitekturalnih zahteva.	0.7
Analiza zavisnosti funkcionalnih i nefunkcionalnih zahteva.	0.7
Identifikovanje (tokom tekućeg release-a) potrebnih arhitekturalnih izmena pre implementacije narednog release-a.	0.7
Eksplicitno identifikovanje zahteva koji se odnose na geografsku dislociranost delova budućeg sistema	0.7

Rezultati prezentovani u tabeli 5.1 organizovani su i u okviru opšteg modela za analizu metoda za razvoj softverske arhitekture (arhitekturalna analiza, arhitekturalna sinteza i arhitekturalna evaluacija), koji su postavili Hofmeister i dr. (2007). Cilj ovakve organizacije dobijenih rezultata, bio je da se jasnije sagleda, kako su značajne tradicionalne arhitekturalne aktivnosti raspoređene, prema tradicionalnim fazama razvoja softverske arhitekture.

Narednim tekstom biće diskutovane identifikovane značajne tradicionalne arhitekturalne aktivnosti, predstavljene u tabeli 5.2., uz upotrebu kvalitativnih rezultata istraživanja, putem navoda ispitanika, čime se i argumentovano potvrđuje odgovor dat na prvo istraživačko pitanje.

Kvalitativni rezultati istraživanja, vezani za prvi cilj istraživanja, grupisani su u okviru kategorije: *eksplicitne arhitekturalne aktivnosti*. Data kategorija sadrži tri podkategorije: *arhitekturalna analiza, arhitekturalna sinteza i arhitekturalna evaluacija* (slika 4.1.), što je u skladu i sa načinom prikaza rezultata u tabeli 5.2.

Arhitekturalno dokumentovanje predstavlja, posebnu podkategoriju eksplicitnih arhitekturalnih aktivnosti, iz razloga što reprezentuje aktivnost, koja prožima sve druge arhitekturalne aktivnosti, tokom trajanja projekta. Zbog toga će se diskusija rezultata, u kontekstu dokumentovanja, sprovesti nezavisno od prethodno pomenutih kategorija, na kraju ovog poglavlja. Diskusijom će biti integrisani kvalitatitivni i kvantitativni rezultati (prikazani u tabeli 5.3).

Tabela 5.2 Eksplicitne arhitekturalne aktivnosti agilnih timova u kontekstu opšteg modela analize metoda za razvoj softverske arhitekture

Opšti model analize (Hofmeister et al., 2007)	**Eksplicitne arhitekturalne aktivnosti agilnih timova u praksi (identifikovane iz kvalitativno-kvantitavnih rezultata empirijskog istraživanja)**
ARHITEKTURALNA ANALIZA	1. Razumevanje poslovnog problema 2. Identifikovanje budućih ciljeva i pravaca razvoja biznisa 3. Identifikovanje ključnih stejkholdera sistema 4. Aktivne diskusije sa stejkholderima u cilju razumevanja poslovanja i poslovnih potreba 5. Istraživanje tržišta i širenje znanja iz domena problema u cilju bolje identifikacije arhitekturalnih zahteva 6. Identifikovanje arhitekturalno značajnih zahteva 7. Identifikovanje granica sistema 8. Analiza rizika, kako bi se identifikovale i izolovale oblasti kompleksnosti 9. Istraživanje tehnologije i trenda opcija 10. Istraživanje odgovarajućeg frejmvorka za implementaciju 11. Istraživanje postojećih biblioteka 12. Razmatranje postojeće infrastrukture u ciljnoj organizaciji 13. Formiranje odgovarajućeg tima i izbor softver arhitekte u odnosu na problem koji se rešava 14. Identifikovati tačke arhitekture koje trebaju biti fleksibilnije u cilju budućih promena i odluka u kontekstu proširivosti softvera za buduće zahteve 15. Eksplicitno identifikovanjei zahteve koji se odnose na geografsku dislociranost delova budućeg sistema 16. Formiranje jedinstvene liste prioritizovanih funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena 17. Utvrđivanja prioriteta na jedinstvenoj listi zahteva, razmatranjem njihove vrednosti sa aspekta biznisa, ali i sa aspekta rizika i uticaja na arhitekturu 18. Analiza međusobnih zavisnosti funkcionalnosti 19. Analiza zavisnosti funkcionalnih i nefunkcionalnih zahteva

	20. Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem interreaguje tokom release-a 21. Razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a 22. Identifikovanje potrebnih arhitekturalnih izmena, pre implementacije narednog release-a 23. Identifikovanje arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera 24. Utvrđivanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti 25. Razmatranja nasleđenog sistema, zavisnosti od drugih partnerskih ili otočnih proizvoda i kompatibilnosti podataka
ARHITEKTURALNA SINTEZA	26. Identifikovanje i analiza postojećih, potencijalnih arhitekturalnih rešenja 27. Identifikovanje i definisanje osnovnih struktura (modula) za glavni deo sistema, kao i njihovih veza (postavka arhitekture) 28. Definisanje osnovne arhitekture podataka 29. Razmatranje i postavka inicijalnog modela raspoređivanja
ARHITEKTURALNA EVALUACIJA	30. Code review 31. Formalni kontinuirani review arhitekture u cilju validacija kritičnih arhitekturalnih zahteva i koncepata dizajna 32. Testiranje performansi sistema ili dr. kritičnih nefunkcionalnih zahteva Upotreba: test case-ova, testova prihvatljivosti, QA testova, testova integracije, regresionih testova; statička analiza koda, mišljenja architectural board-a (eksperti za razne aspekte: stručnjaci za deployment, stručnjaci za security, za kvalitet i dr. razne aspekte arhitekture); scenario analiza interakcije stejkholdera sa sistemom, sa fokusom na nefunkcionalne zahteve; arhitekturalnih spike-ova; prototipova; simulacija opterećenja buduće arhitekture sistema; time boxed proof of concept, eksperiment.

Tabela 5.3 Arhitekturalno dokumentovanje agilnih timova u praksi

ARHITEKTURALNO DOKUMENTOVANJE	1. Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema 2. Dokumentovanje arhitekture visokog nivoa (arhitekturalno značajnih zahteva, ograničenja okruženja, tehnološkog steka i arhitekturalnog rešenja)

Arhitekturalna analiza

Arhitekturalna analiza obuhvata set eksplicitnih arhitekturalnih aktivnosti (prikazanih u tabeli 5.2), čiji rezultati direktno utiču na izbor arhitekturalnog rešenja i postavku inicijalne arhitekture. Arhitekturalna analiza podrazumeva informacije o kontekstu, radnom okruženju organizacije i načinu poslovanja organizacije, u kojoj će budući sistem biti implementiran. Kao osnova svega navedenog je razumevanje poslovnog problema koji se rešava (1. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP7: "... potrebno je da shvatimo suštinu njihovog poslovnog načina rada... tj. njihov poslovni model... da bi dobro napravili arhitekturu. Poslovna arhitektura i softverska arhitektura se naslanjaju jedna na drugu..."

ISP19: "...način na koji oni funkcionišu da li je njima za biznis, npr. neki server side ključan ili koliko im je bitan interfejs; šta će se prvo razvijati, a nekada nam i daju koji patern hoće da koriste tj. to moramo da koristimo da bi se u klopili u njihovu infrastrukturu. Da li je to neki service bass na koji treba da povežemo ili nešto slično. U principu pored njihovih već postojećih arhitektura moramo uklopiti i njihov biznis proces, kompleksnost, ponovljivost, brzinu izvršavanja, jer sve to direktno utiče na arhitekturu..."
ISP13: "...dokumentacija tih informacija ide kroz business case koji se u startu identifikuje. Postoji projektna specifikacija u okviru koje je navođeno, prvo, zbog čega se uvodi taj softver, koje probleme on treba da reši, koji su ciljevi, ko su ti ključni učesnici..."

Takođe, ispitanici smatraju da je u okviru arhitekturalne analize potrebno sagledati i pravac širenja poslovanja organizacije, jer šira slika sistema tj. njegov budući razvoj, itekako utiče na postavku arhitekturalnog rešenja na početku projekta (2. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP7: "...tražimo i širu sliku šta će taj sistem biti za 2 godine. Pa da imamo okvir, jer možda ne bih izdvojio nešto kao poseban moduo aplikacije, ali zbog budućnosti znamo da će se to integrisati sa nečim drugim i onda je bolje da ga odmah izdvojimo, jer će nam sutra biti lakše da nešto napravimo nego da radimo refaktoring..."

Za uspeh projekta vrlo je važno identifikovati ključne stejkholdere sistema (3. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Naime, razumevanje poslovnog problema zasnovano je na eksplicitnoj arhitekturalnoj aktivnosti, koja podrazumeva razgovor sa ključnim stejkholderima sistema, bilo na grupnim sastancima ili pojedinačnim intervjuima. Ispitanici su istakli da je važno da na sastancima pored vlasnika proizvoda učestvuje i softver arhitekta, kako bi iz prve ruke čuo koji su problemi, jer u suprotnom nikada ne dobije potpune informacije, putem dokumentacije koja mu se dostavi (ISP.7) (5. eksplicitna arhitekturalna aktivnost u tabeli 5.2).

Cilj razgovora sa stejkholderima je razumevanje poslovnih potreba ključnih za rešavanje poslovnog problema. Definisane poslovne potrebe osnova su za generisanje liste karakteristika budućeg rešenja i arhitekturalno značajnih zahteva. Pored toga ispitanici smatraju da je za identifikaciju arhitekturalnih zahteva važno i istraživanje tržišta i širenje znanja iz domena problema (5. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP13. "...obično nemate jednog ključnog korisnika, nego trebate intervjuisati u firmi dosta njih, pa identifikovati ko od njih ima najveći uticaj na taj softver... obično to ne mora biti direktor koji je dodeljen, nego to može biti neko drugi, neko treći.... recimo ima li smo jedan softver, ja ga zovem propalim... iz prostog razloga što jedna grupa ljudi nije bila u startu prepoznata kao bitan stejkholder pa nisu razmatrani njihovi interesi... ".
ISP12: "... sa stejkholderima komuniciramo kroz priču... mi idemo kod njih, imamo sastanke, jedan sastanak, drugi sastanak….najveći problem je od njih izvući big picture tj. šta oni hoće. Onda kad mi shvatimo šta oni hoće, e onda pokušavamo da od cele priče raščlanimo delove tj. šta sve ima u toj velikoj slici...".

Cilj intenzivne komunikacije i kolaboracije sa stejkholderima je i definisanje kriterijuma prihvatljivosti (koji su osnova testova prihvatljivosti) i kriterijuma kvaliteta:

ISP13: "...stejkholderi treba da definišu šta to znači kvalitetan softver, šta znači kvalitetno rešenje tj. šta njima znači kvalitetno rešenje, jer kvalitet proizilazi iz toga...".

Tradicionalna strategija prikupljanja zahteva podrazumeva, da se na početku projekta sprovede njihova detaljna specifikacija (Big Requirement Up Front, u nastavku BRUP), dok agilna strategija predstavlja drugi ekstrem. Rezultati empirijskog istraživanja upućuju na zaključak da je strategija definisanja zahteva, u razvoju kompleksnih poslovnih softverskih rešenja, između ova dva ekstrema. Drugim rečima, na početku projekta je neophodno izvršiti postavku inicijalnog seta arhitekturalno značajnih zahteva, čije detaljno razrađivanje će se sprovoditi u momentu planiranja iteracije, u okviru faze detaljnog dizajna (6. eksplicitna arhitekturalna aktivnost u tabeli 5.2).

Ovakav pristup u skladu je sa Lean principom JIT i odlaganjem odluke do poslednjeg odgovornog momenta. Razlog za primenu ovakve strategije jesu promenlji zahtevi tokom razvoja, kao i nemogućnost njihovog tačnog i sveobuhvatnog razmatranja na samom početku projekta, te svaka njihova detaljna razrada predstvalja otpad i prekomeran WIP.

ISP20: "...vodi se konstantno bitka između agilnog razvoja i ostatka tj. "just in time - just enough details". Just in time znači da ja kad imam informacije da te informacije i dalje budu relevantne, da nisu zastarele. Sa druge strane, ne možete uči u razvoj softvera a da nemate neku analizu arhitekture itd....".
ISP14: "...ako je klijent spreman da radi agilno onda nema velike analize biznis procesa, nego će se raditi just in time onda kada je to potrebno...".

Tradicionalni timovi najveći deo planiranja sprovode up front, dok JIT planiranje nastoje u što većoj meri da smanje. Agilni timovi se rukovode isključivo JIT planiranjem, dok rezultati istraživanja impliciraju zaključak da je, za razvoj kompleksnih softverskih rešenja, neophodna strategija sa dovoljno up front planiranja i dovoljno JIT planiranja, tokom celog trajanja projekta. Vreme potrebno za up front planiranje, agilni timovi u praksi, procenjuju u odnosu na kontekstualne faktore, opisane u prethodnom delu rada. Takođe, razumevanje granica sistema, je mera za utvrđivanje vremena potrebnog za up front definisanje arhitekturalnih zahteva (7. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP4: "...ako su requirementi kvalitetni, biće više stvari poznatih unapred, u drugoj situaciji, vi morate da dođete na taj nivo, morate da uložite više efforta, da bih ih dobro identifikovali i to produžava tu početnu fazu, jer se bavite requirement analizom, a ne arhitekturom... ako requirementi nisu na odgovarajućem nivou morate klijentu da pomognete da do toga dođu, jer ne vidim kako možete da napravite arhitekturu bez da razumete scoup i requiremente. Tako da se u našoj kompaniji od arhitekte očekuje da radi i kao konsultant...".

Razumevanje granica sistema svodi se na: analizu glavnih poslovnih entiteta i veza između njih, (ER dijagrami, UML model domena); analizu ključnih poslovnih procesa i toka podataka organizacije; analizu načina na koji korisnici intereaguju sa sistemom; analizu korisničkog interfejsa (UI dijagram); nefunkcionalnih zahteva:

ISP18: "...poslovnim procesima gledamo da li će oni u pozadini birati glavnu knjigu, finansijski deo, da li će menjati zalihe, da li će neke druge stvari imati izdvojeno. I onda koncipiramo i module i naše klase da preslikaju tu sliku...".
ISP12: "...procesi koje klijent ima i na koji način planira da ih sprovodi, utiče na arhitekturu...".

Identifikacija rizika je takođe značajna eksplicitna arhitekturalna aktivnost (8. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Inicijalna lista rizika generiše se na samom početku projekta, a ažurira se tokom celog perioda trajanja projekta:

ISP16: "...da, vršimo procenu rizika, stim što je sasvim jasno da samo deo rizika može da se vidi u samom startu, analizom problema. Ta lista se ažurira u toku razvoja softvera. Postoji i tabela oko toga koliko je ustvari nešto rizično, kakve posledice ima, koliko košta ukidanje tog rizika. Na osnovu više parametara se određuje da li će nešto da se rešava u određenom trenutki, da li će da sačeka, da li nešto i ne može da se reši na našem nivou, itd....".
ISP13: "...identifikujemo koje su to rizične tačke, šta može da dovede do toga da rešenje ne radi, a potom u startu odredimo šta ćemo da radimo za svaku identifikovanu tačku rizika... ta lista rizika se konstantno menja, dopunjava...".

Pod arhitekturalnim rizicima, ispitanici podrazumevaju npr. kupovinu otočnih komponenti, koje mogu biti preskupe ili nedovoljno ispitane, pa je upitno da li će ispuniti namenu za koju se kupuju (ISP2). ISP4 arhitekturalne rizike vezuje za momenat donošenja odluke oko izbora opcije arhitekturalnog rešenja, dok ISP8 procenu rizika vezuje za pitanje granica sistema, u smislu koliki broj korisnika ili procesa sistem može da podrži, bez da se menja iz korena. Rizik se vezuje i za pitanje da li odabrana platforma može da podrži ono što je na projektu potrebno.

Ispitanici ističu da, ukoliko razvoj nekih elemenata predstavlja visok rizik, često se isti uopšte ne razvijaju, pa čak i ako su visokog prioriteta. Takođe, ispitanici primenjuju strategiju kojom na početku nekog release-a prvo razvijaju najrizičnije delove da bi se do kraja release-a rešenja najrizičnijih delova stabilizovala:

ISP11: "...naravno da vršimo procenu rizika. Neke stvari odbacujemo, iako znamo da su bitne i prepoznate su da su visokog prioriteta, ako procenimo da nose veliki rizik. I trudimo se da na početku nekog razvoja nove verzije softvera odmah na startu radimo rizičnije stvari, pošto znamo da ćemo imati više vremena do kraja razvoja te verzije da ih stabilišemo, rešimo sve bagove...".
ISP13: "...tamo gde je najveći rizik tu ćete gledati da što pre razvijete, da imate vremena, ako vam fali neko znanje itd...".

Vrlo često na projektima izostaje sistematsko dokumentovanje liste identifikovanih rizika i ona postoji samo u glavama arhitekti:

ISP13: "...pošto nije bilo vremena uglavnom sam u glavi vodio tu listu rizika, identifikovao za koji šta treba da se uradi i to realizovao. To nije standardna analiza rizika jer nemamo ni dokumentaciju koja to treba da isprati to....uglavnom zato što nemamo vremena...".

Empirijski rezultati pokazuju da ispitanici, značajnom eksplicitnom arhitekturalnom aktivnošću vide istraživanje tehnologija, okvira i postojećih biblioteka, koje će se koristiti u implementaciji rešenja (9.; 10.; 11. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Ove tehničko tehnološke aspekte vide čak i kao ključni faktor uspeha u razvoju arhitekture:

ISP1: "...ključni faktor uspeha je ostaviti dovoljno vremena za istraživanje tehnologija i frejmorka koji bi se koristili u implementaciji softvera...".
ISP9: "...kad svaki put radimo nešto drugo treba da imamo vremena za research različitih tehnologija, za pravljenje možda nekog proof of concept, evaluaciju raznih rešenja itd. Ovde kada smo pravili novi proizvod, da onda smo se rukovodili npr. radilo se o izboru tehnologija pa je bilo pitanje da li hoćemo da investiramo više u tradicionalnije tehnologije, koje nam možda mogu pružiti bolje performanse, ili hoćemo modernije tehnologije koje nam pružaju rapid development, brži learning curve za developere i naravno poslove za budućnost...pa se odabrala druga varijanta, iako sada imamo nekih problema sa performansama i drugim stvarima...".

Na arhitekturalno rešenje utiču i sledeća ograničenja okruženja: budžet, tehnologije koju klijent već ima u organizaciji, softverske i hardverske komponente kojima klijent raspolaže (12. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Stoga njihova identifikacija predstavlja značajnu up front eksplicitnu arhitekturalnu aktivnost:

ISP8:"...često se raspitujemo kojim softverskim paketima su realizovali stvari koje oni već imaju. Vrlo je često da te veće organizacije već imaju tu microsoft-ovu ili oracl infrastrukturu i ne bi bilo loše da i vi to uradite sa vaše strane, jer će to posle da se hostuje na njihovim serverima. Ako vi to radite u nekoj vašoj priči koja je specifična za vas i okruženje, to će njima biti problem. Oni će lakše integrisati to što vi pravite ukoliko je to što vi radite ustvari kontinuitet u odnosu na ono što oni već imaju... to koliko vi možete da iskoristite već nešto što oni imaju u svojoj internoj infrastrukturi, kupljeno ili napravljeno, je ono što po meni utiče na razvoj. Ta njihova postojeća arhitektura utiče na razvoj arhitekture novog sistema...".
ISP19: "...često oni već imaju neka softverska rešenja sa kojima se treba integrisati, način na koji rade, tehnologije koje su nam dozvoljene...".
ISP8: "...budžet koji oni imaju utiče vrlo direktno na to kako će arhitektura da izgleda i vi spram toga skalirate i tim i cenu hostinga, trajanje razvoja...u zavisnosti koliki je budžet takvo će vam biti i uspostavljanje arhitekturalne priče. Ukoliko imate budžet koji je veći, onda ćete i arhitekturu pripremiti da bude robusnija, skalabilnije i da više vremena izdvojite na testiranje...".

Ključan faktor uspeha, kako arhitekture, tako i projekta u celini, predstavljaju *"pravi ljudi na pravom mestu" (ISP17),* što znači da u zavisnosti od problema koji se rešava, arhitekturalnih

zahteva i tehnološkog steka, treba formirati odgovarajući tim ljudi na projektu (ISP13) (13. eksplicitna arhitekturalna aktivnost u tabeli 5.2).

Takođe, bitno je *"postići flesibilnost, ne ograničiti se, ne vezati sebi ruke na startu. U tom smislu razdvajanje odgovornosti komponenti je nešto što je uvek dobro. Takođe, korišćenje web servisa i mikro servisa koji su više manje nezavisne celine, zato što se dobija na fleksibilnosti i zato što je lako svaku komponentu i razviti i testirati i u krajnjem slučaju zameniti ako ne odgovaraju" (ISP10).* Ovakav stav implicira zaključak da je neophodno eksplicitno identifikovati tačke arhitekture koje trebaju biti fleksibilnije, u cilju budućih promena i odluka, u kontekstu proširivosti softvera usled novih zahteva (14. eksplicitna arhitekturalna aktivnost u tabeli 5.2).

Ispitanici predlažu eksplicitno identifikovanje arhitekturalno značajnih zahteva i njihovo dokumentovanje u formi arhitekturalnih, tehničkih priča i epika. Arhitekturalne priče trebaju biti sastavni deo liste zahteva proizvoda, na osnovu kojeg se generiše lista arhitekturalnih zadataka za iteracije. Lista zahteva proizvoda treba da predstavlja jedinstvenu listu prioritizovanih funkcionalnih i nefunkcionalnih zahteva sistema (16. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Bitan input za dizajn arhitekture predstavljaju i zahtevi koje se odnose na geografsku dislociranost budućeg sistema (15. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP20: "...treba prepoznati zahteve što se tiče arhitekture, kao nešto što je sastavni deo backlog-a za celokupan proizvod..." .

Rezultati istraživanja upućuju na zaključak da prioritizacija zahteva treba da se sprovodi ne samo sa aspekta vrednosti za korisnika, već i sa aspekta rizika, troškova i tehničkih zavisnosti (17.; 18.; 19.; 20. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Tehničke zavisnosti podrazumevaju analizu međusobne zavisnosti funkcionalnih zahteva, ali i zavisnost funkcionalnih od nefunkcionalnih zahteva. Takođe, neophodno je razmotriti i zavisnosti koje potiču od spoljnih sistema sa kojima sistem interreaguje:

ISP20: "...nekoliko faktora utiču na rankiranje i samu proritizaciju zahteva...krećemo od biznis valua naravno, dalje uzimamo u obzir cost, znači estimaciju, potom rizike koje adresiramo, znanje koje stičemo o samom proizvodu, a i same tehnologije. I dodao bih neke tehničke zavisnosti. Znači bez što su neki zahtevi poređani po nekom biznisu, često se dešava da neki zahtev koji je potreban ovde iziskuje neke promene koje su verovatno u nekom drugom delu itd...".

Ukoliko nije u pitanju prvi release, ispitanici smatraju da je u okviru faze planiranja release-a takođe neophodno doneti set up front arhitekturalnih odluka, koji se tiču utvrđivanja granica, analize arhitekturalnog rešenja i identifikacije rizika:

ISP20: "..imamo taj neki release planning, gde pravimo neku ratio tabelu gde pokušavamo da identifikujemo sve stvari koje su nam neophodne da bi smo imali uspešan release. A to su recimo, prva stavka, da li smo

definisali scope, koja nam je i šta nam je vizija, da li je urađena analiza arhitekture, da li će arhitektura biti zrela da se nosi sa zahtevom, da li postoji set dokumenta koji opisuju na nekom high level nivou, koji impact imamo na arhitekturu, šta još treba da se povede računa što se tiče implementacije, kako će uticati na neki dizajn, pa da li smo identifikovali sve rizike, da li smo napravili backlog, da li imamo enterprise test sistem, pošto nam je ideja da sve reqiuremente testiramo na nekom enterprise test okruženju…".

Arhitekturalnom analizom potrebno je proveriti da li arhitekturalno rešenje može da podrži željene funkcionalnosti budućeg relese-a (21. eksplicitna arhitekturalna aktivnost u tabeli 5.2) i koji su potencijalni problemi. Neophodne arhitekturalne izmene postaju deo formirane jedinstvene liste zadataka, koja se na taj način kontinuirano ažurira (22. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Čest je slučaj da implementacija nekog nefunkcionalnog zahteva predstavlja neophodan uslov za implementaciju seta funkcionalnih zahteva:

ISP20: "...svaki release ima tri faze: inception faza, developement faza i maintenance faza. U toku inception faze, što se tiče arhitekture, ne primenjujemo nikakve iteracije nismo na tom nivou još uvek, ali u nekom vremenskom intervalu od nekoliko meseci, arhitekte dobijaju određene oblasti koje analiziraju i kreiraju neke dokumente koje drugi deo organizacije isto revidiraju, zarad impakta na test okruženje, developement okruženje, sisteme, itd. Imamo backlog stvari koje trebamo analizirati, to se radi po nekom ranku i prioritetu, u zavisnosti od biznisa, i onda se vrši analiza i kroz analize treba da dobijemo odgovor da li je potrebno napisati neke arhitektonske epic-e, nešto uraditi, šta je potrebno refaktorisati, itd. To je ono što se tiče te inception faze, i onda obično traje od nekoliko nedelja do nekoliko meseci u zavisnosti od veličine...".

U agilnim procesima razvoja praksa je, da se up front sagledavaju zahtevi za nekoliko iteracija, a ne samo za jednu. U razvoju kompleksnih poslovnih softverskih rešenja, ispitanici smatraju da se ovaj koncept treba proširiti na analizu zahteva i razmatranje arhitekture iznad tekućeg release-a. Razmatranje arhitekture iznad jednog release-a, ima za cilj identifikovanje i razvoj arhitekturalnih elemenata, koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera (23. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Takođe, cilj je i utvrditi zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti, kao i zavisnosti od partnerskih ili drugih postojećih proizvoda i kompatibilnost podataka (24.; 25. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Analiza arhitekture sprovodi se ad hoc, bez primene nekog formalnog procesa u okviru agilnog razvoja, što su ispitanici identifikovali kao aktuelni izazov u praksi:

ISP20: "...neophodno je voditi računa gde će se širiti biznis. Kao što u agilnom u scrum-u ne gledamo samo jedan sprint nego nekoliko unapred, tako moramo i da razmišljamo šta mi taj novi release donosi zato smo mi uveli taj koncept sistem arhitekti koji analiziraju te nove zahteve, kako može da se implementira, šta je potrebno promeniti u arhitekturi, definisati ne samo biznis nego i te neke arhitektonske zahteve koji bi pratili sve to i omogućili i jedno i drugo, da bi sprečili pojavu da mi ne možemo više da menjamo neke stvari....Jedna od stvari koja se dešava kod te tzv. "feasibility study" to mora da bude nekako inkorporirano u agilnom razvoju. Naravno nije potrebno sve uraditi unapred, ali je potrebno uraditi u nekom trenutku u svakom slučaju, ali ga ne treba zapostaviti dalje...".

ISP20: "...ideja je da, kada recimo krećemo sa sledećom verzijom softvera, sistem arhitekte već sad na ovom nivou počinju analizu za sledeći release koji je za nekih godinu dana. Ideja je da pokušamo da uočimo koji su to nedostaci koji postoje u arhitekturi, da identifikujemo te neke arhitektonske epic-e, neke refaktorizacije koje su nam neophodne da bismo mogli, kada dodje red na taj release, da bismo mogli da krenemo u detaljnije analize... sistem arhitekte kod nas imaju zadatak da analiziraju i dokumentuju neke stvari, koje su neophodne kao input za sledeći release, da bi mi mogli da vidimo u kom pravcu ćemo razvijati softver, koje timove odnosno kakav set up timove će nam trebati za sledeći release. To je neka analiza koju mi radimo što se tiče arhitekture. Što se tiče načina kako mi to radimo, još uvek je to na nivou ad hoc, nemamo nikakvu metodologiju...".

Rezultati kvalitativne analize iznedrili su brojne koncepte u okviru kategorije *arhitekturalna analiza*. Diskusijom ovih koncepata u prethodnom tekstu, uočen je set dodatnih značajnih eksplicitne arhitekturalnih aktivnosti (prikazani u tabeli 5.4), koje su komplementarne sa kvantitativno identifikovanim aktivnostima iz tabele 5.2.

Tabela 5.4 Identifikovane eksplicitne arhitekturalne aktivnosti u okviru kategorije *arhitekturalna analiza*

Razmatranje poslovne i informacione arhitekture ciljne organizacije
Razmatranje softverske i tehničke infrastrukture ciljne organizacije i identifikovanje elemenata koji mogu biti ponovo korišćeni
Opis problema i ciljeva njegovog rešavanja
Identifikovanje poslovnih slučajeva upotrebe
Definisanje kriterijuma prihvatljivosti
Identifikovanje ograničenja koja će biti nametnuta budućem sistemu, a potiču iz realnog okruženja i mogu biti različitog porekla: politička, ekonomska, pravna, tehnička, funkcionalna
Formiranje inicijalne liste rizika
Ažuriranje inicijalne liste rizika
Ažuriranje jedinstvene liste prioritizovanih funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena

Arhitekturalna sinteza

Kodiranjem kvalitativnih podataka iz intervjua, utvrđena je i kategorija *arhitekturalna sinteza*, koja uključuje dve podkategorije: *izbor arhitekturalnog rešenja* i *postavka arhitekturalnog rešenja*. U narednom tekstu biće diskutovane značajne eksplicitne arhitekturalne aktivnosti, u okviru arhitekturalne sinteze (navedene u tabeli 5.2), uz korišćenje navoda ispitanika iz intervjua. Ovakvom diskusijom obezbediće se integrisanje kvantitativnih i kvalitativnih rezultata istraživanja.

Izbor arhitekturalnog rešenja

U izboru arhitekturalnog rešenja, softver arhitekte koriste izlaze dobijene u fazi arhitekturalne analize. Oslanjaju se na sopstveno iskustvo i uglavnom na poznata, postojeća rešenja, koja je potrebno na početku projekta analizirati, sa ciljem da se odabere najadekvatnije u odnosu na

problem koji se rešava (26. eksplicitna arhitekturalna aktivnost u tabeli 5.2). Pored iskustva arhitekte, u odabiru adekvatnog arhitekturalnog rešenja neophodno je poznavati trend i opcije koje postoje za problem koji se rešava, profil klijenta, mogućnosti razvojnog tima, arhitekturalno značajne zahteve sistema i budućnost razvoja sistema:

ISP4: "...morate znati kada pravite arhitekturu šta postoji od opcija... u suštini nema tu mnogo opcija, jer za svaki problem imate tačno te i te opcije, gde sad morate da izmerite koja je bolja, ali bez toga ni ne možete da krenete... identifikovati šta je najrelevantnije za klijenta...".
ISP4: "...morate da pratite šta je trend, morate razumete, recimo mikroservise - šta to znači, šta to znači za vašu kompaniju, šta to znači za klijenta, koji su benefiti, koje su loše strane, morate da budete kritični...potom profil klijenta... i na kraju da li su vam ljudi sposobni da koriste tu tehnologiju...".
ISP11: "...izbor nekih poznatih metodologija i nekih poznatih rešenja, u smislu šta od toga koristiti, šta konkretno ugraditi u taj dizajn, na šta se osloniti, šta praviti od početka, šta uzeti od tehnologija koje su poznate tj, postojeće i dugo godina na tržištu...".
ISP2: "...trudimo da iskoristimo naše znanje, našu ekspertizu, tako da uvek kad rešavamo neki sistem i pravimo arhitekturu pravimo ga u nekoj viziji naših prethodnih projekata, komponenti koje možemo iskoristiti, znanja koja imamo...".
ISP10: "...budućnost razvoja te aplikacije umnogome može da menja pristup u razvoju arhitekture. Znači dobro razumenja poslovnog konteksta aplikacije, šta ta aplikacija treba da radi. Koliko korisnika će je koristiti, koje su dalje mogućnosti za razvoj i nadograđivanje i unapređivanje, jer od te cele kombinacije zavisi šta će se odabrati, da li će se odabrati neko lako i jednostavno rešenje ili nešto što je kompleksnije i zahtevnije, pogotovo u startu, da bi imalu tu fleksibilnost u daljem razvoju...".

Ukoliko je u pitanju projekat sa novim domenom problema, u kojem tim nema prethodnog iskustva, arhitekta se prilikom analiza mogućih opcija, pomaže POC konceptom koji mora biti vremenski ograničen:

ISP4: "...kada se rade nove stvari, morate da džastifikujete, i napraviti balans u korišćenju novih tehnologija... pored ove stvari treba vam i domensko znanje, neko stručno znanje u pogledu mogućih rešenja nekog tipa problema. Međutim, kako to da uklopite nije nešto što možete da otkrijete za nedelju dana. Onda se pomažete sa proof of concept-ima...".
ISP4: "...proof of concept, mora da bude time boxed, morate da identifikujete ciljeve i morate da identifikujete šta ćete da probate, pa da onda napravite analizu...".

Drugi pristupi kojima se agilni timovi u praksi koriste, pri izboru arhitekturalnog rešenja, su: "misaono istraživanje" arhitekte, diskusija sa drugim arhitektama i članovima tima, izučavanje iskustava drugih kompanija sa rešenjem koje se analizira:

ISP8: "...više je to kroz komunikacije sa drugim ljudima, jer nismo mi velika grupa ljudi i prosto smo svi vezani i dosta razgovaramo na tu temu. Dosta čitam, i to je defakto stanje. Literatura je dostupna i sve to negde već piše. A idealno je ako ste vi to već negde probali. Međutim ako te stvari predlaže neko ko je

kredibilan i čije se mišljenje uvažava i neko na koga ste se možda oslanjali u prošlosti, ne testiram ja tu mnogo...".
ISP9: "...najčešće se radi više o misaonoj vežbi, research u smislu pročita se šta se pročita, sam čovek ima tehnička iskustva iz nečega ili nema pa pita nekog koga ima, i takva rešenja prezentujemo nekoj široj grupi i vidimo šta ta grupa misli i te se stvari uvrste...".
ISP18: "...vidimo iskustva drugih firmi koja su radila na sličnim projektima, pogledamo malo na internetu kako se pokazala takva rešenja...".

Često je nužna i izrada prototipova, kako bi se proverilo da li arhitekturalno rešenje može da podrži ključne nefunkcionalne zahteve sistema:

ISP11: "...za neke stvari radimo neke prototipe, što bi rekli isprobavamo, ali ne čitav dizajn, nego samo neke delove tog dizajna, da li je to ono što želimo da postignemo...".
ISP20: "...radimo neki prototip, pokušavamo da izvučemo šta je to što je najosnovnije što da možemo da uradimo, čisto da vidimo da li će to arhitektonsko rešenje biti zadovoljavajuće ili ne. I dešavalo se da smo krenuli sa tim razvojom i posle par sprintova smo shvatili da poslednjom arhitekturom ne može da zadovolji to, pa smo to rešenje povukli...".

ili sprovođenje vremenski ograničenog eksperimenta, sa unapred definisanim metrikama i kriterijumima za uspeh i tabelom u koju se unosi troškovi i benefiti za rešenje koja se razmatra:

ISP14: "...radimo eksperimente. Dozvolimo sebi da ako nešto ne znamo, da to isprobamo. Nismo u onoj situaciji da kao ne znamo šta će biti rešenje, zbog toga da planiramo mesec dana, idemo da probamo pa da vidimo šta će biti rezultat. Ali onda sebe time box-ujemo. Za sve što nam je bila velika nepoznanica u domenu nefunkcionalnih zahteva, isprobavali smo oko 4 sata do jednog dana. Nakon toga praksa je da prezentujemo timu šta je odlučeno, u smislu radimo dalje to ili ne radimo... ukoliko je nešto novo onda bih definitivno preporučio eksperiment tj. analizu više rešenja ne samo jedno sa unapred određenim metrikama i kriterijumima za uspeh...".
ISP16: "...tabeli evidentiramo cost-benefit odnos analiziranog potencijalnog rešenja i tako realizujemo vrednovanje rešenja...".

U slučaju postojanja nasleđenog sistema neophodno je prvo napraviti analizu njegove arhitekture, a potom analizu njene usklađenosti sa arhitekturalnim rešenjem novog sistema:

ISP20: "...kod nekih postojećih rešenja, pokušavamo prvo da utvrdimo gde smo ... tj. koliki je gap između našeg rešenja i rešenja koje treba. I onda koji su koraci da premostimo taj gap...radimo neku analizu prvo impakta koji možemo imati na postojeću arhitekturu. Na osnovu tog impacta na postojeću arhitekturu i nekih funkcionalnih zahteva koje imamo, imamo čoveka koji treba da predloži arhitekturalno rešenje, naravno uz konsultaciju sa drugim ljudima. Imamo mali tim ljudi koji radi na svemu tome...".

Pravilo kojim se arhitekte rukovode pri izboru arhitekturalnog rešenja jeste biznis vrednost koju obezbeđuje dato rešenje. U slučajevima kada dva potencijalna rešenja imaju istu vrednost sa aspekta biznisa, odabira se ono koje omogućava lak povratak na prethodno stanje:

ISP20: "...pravilom kojim se ja vodim je sledeće: da ukoliko postoje dva moguća rešenja koji imaju sličnu biznis value, treba se ići sa onim rešenjem iz koga se najlakše moguće rekaverovati. Znači tom se logikom vodim odabrati rešenje koje mogu lakše da se vratim na prethodno stanje, da bih mogao da krenem nekim drugim putem, ako se pokaže da nije dobro...".

Postavka arhitekturalnog rešenja

Kategorija *postavka arhitekturalnog rešenja* sadrži podkategorije i koncepte koji reprezentuju proces kojim agilni timovi u praksi postavljaju inicijalnu arhitekturu. Rezultati istraživanja pokazuju, da se kompleksni sistemi ne grade isključivo na konceptu nascentne ahitekture, već da zahtevaju up front postavku arhitekturalnog skeleta za glavni deo sistema, što se još naziva i arhitekturom visokog nivoa. Inicijalna arhitektura, postavljena na početku projekta, iterativno-inkrementalnim razvojem se nadograđuje detaljima. Ispitanik ISP16 arhitekturalni skelet objašnjava kroz sledeći primer:

ISP16: "...konkretno, ako mi treba 5 matematičkih funkcija koje treba da se izvršavaju, ne moram ja u startu da implementiram svih 5. Recimo da postoji jedna koja je glavna na koju se sve ostale 4 oslanjaju, čak ne mora ni da bude. Ja ću izdizajnirati kako mi izgleda "engine" za pokretanje funkcija koji će da dobavlja podatke, da rezultate vraća, itd. Znači taj engine ću da napravim, i moram da ga izdizajniram u startu. Napraviću koji su mi interfejsi ka tim plug-in-ima i ostalim stvarima koje su mi potrebne. Na početku ću samo taj core infrastrukture da izdizajniram. A onda ću dati developerskim timovima da rade na tome...".

Nakon identifikacije modula za glavni deo sistema, identifikuju se veze između modula, u cilju definisanja interfejsa putem kojih će moduli međusobno komunicirati (27. eksplicitna arhitekturalna aktivnost u tabeli 5.2):

ISP12: "...kada napravimo module, onda sagledavamo koje su veze između tih modula. Ako neki od modula ili entiteta, sa čim se sistem bavi, ima slične funkcionalnosti, slične osobine, gledamo da li možemo da ih spajamo ili ne spajamo. I to su nam uvek diskusije da li treba ili ne treba... Kad definišemo entitete, pokušavamo da definišemo veze između njih...".
IP16: nakon postavke infrastrukture za core sistema, definišu se interfejsi tj. definiše se komunikacija između modula..." .

Sledeći korak je definisanje podsistema i lejera za identifikovane module sistema. Prikazuju se upotrebom neke vrste dijagrama komponenti, ili putem grubog sistemskog dijagrama:

ISP1: "...podsistemi i lejeri se dokumentuje kroz onaj dizajn dokument i tu treba da se opiše sve, i component dijagram i grubi sistemski dijagram gde se prikazuju razne kockice kako su povezane...".

a česta je njihova izrada i u okviru alata kao što je Jira:

ISP7: "...identifikujemo ih pre implementacije. Pošto obično koristimo tu Jiru, tamo možemo da definišemo komponente sistema. U zavisnosti od toga šta je komponenta, pošto je za njih komponenta u Jiri nešto apstraktno, nešto zovemo komponenta nešto layer. To je po mojoj proceni da li je to velika stvar ili neka manja stvar. Pošto komponenta kod nas može pokriti više ili par nekih modula aplikacija, koji su raslojeni po različitim podkomponentama...".

U slučajevima kada su komponente kompleksnije, pristupa se i njihovoj razradi na podkomponente i interfejse:

ISP10: "...u zavisnosti od same funkcionalnosti komponente, ako je nešto kompleksnije, ide se dalje u razradu, znači sama komponenta može da se razradi, ako ima neko veće procesiranje, ako ima nešto što je zahtevnije, što treba razbiti, onda se to raščlanjuje na komponente, i uz tu generalnu arhitekturu, naznačava se kojim tehnologijama, ili kojim tehničkim rešenjima će biti rešavani problemi, kako će se komponenta praviti, koja tehnologija, koji pristupi... Pre implementacije, u samom pravljenju arhitekture i dizajna, tu se identifikuju i slojevi i interfejsi i pristupi koji će biti korišćenji u realizaciji. Identifikujemo ih na osnovu funkcionalnih zahteva svake komponente...".

Definisanje klasa, njihovih metoda i atributa smatra se delom posla koji obavljaju agilni timovi, tokom iteracija:

ISP20: "...ja to vidim tako da neki deo toga treba naravno identifikovati ranije na nivou paketa, na nivou stvarno nekih lejera. A nivo, tipa klase, API-ja itd. moguće je razvijati tokom sprintova...".

Razmatranje modela raspoređivanja takođe spada u važne up front arhitekturalne odluke (29. eksplicitna arhitekturalna aktivnost u tabeli 5.2), iako se u praksi ovaj model često ne sagledava na početku projekta, zbog što ranijeg ulaska u razvoj funkcionalnosti. To rezultuje kompleksnijim modelom na kraju projekta, većim troškovima i dodatnim vremenom za njegovu popravku i implementaciju:

ISP6: "...up front, iz razloga što je to jako veliki problem ako se na početku ne razmišlja o tome. Nije retka situacija da se napravi sistem za koji je jako kompleksno uraditi deploy i koji će imati probleme performansi, a to se desi ako se na početku projekta ne misli deployment modelu tj. kako će sistem da izgleda u produkciji. Često se pravi sistem, a da se na početku projekta ne razmisli kako će izgledati u produkciji...".
ISP11:"...ako bi u startu bio definisan mogao bi da utiče na sam dizajn i na samo krajnje rešenje. Opet praksa je malo drugačija, to se retko definiše na startu... što onda uglavnom u prvoj verziji softvera dovede do toga da imamo komplikovan deployment, da smo napravili sve super, a treba nam 15 ljudi da sve to isporučimo. Pa se tada krene raditi na tome da se sve poboljša i to je onda velika borba. U startu nemamo vremena, zato što trebamo napraviti funkcionalnosti, a kasnije nemamo vremena zato što to postaje kompleksno i treba još više ljudi i vremena da to popravimo. Po meni bi trebao postojati na početku, ako ne u detaljima, onda barem u nekim konceptima, da se zna šta se želi postići...".

Up front vreme i napori potrebni za inicijalnu postavku modela raspoređivanja softvera, u direktnoj su zavisnosti od tehnologije koja se koristi. Izbor tehnologije određuje da li je potrebno napraviti sopstveno rešenje ili je mogući koristiti već postojeća:

ISP3: "...nekad smo razvijali sopstvena rešenja za to, u slučajevima nekih Java aplikacija, a nekad već imate rešenja, kao u slučaju Visual studija, ili Azura...Rubia - gde imate data rešenja za deployment...".

Danas cloud koncept preuzima dominaciju u domenu hostinga, jer podrazumeva virtualni hardver i potrošnju u skladu sa potrebama i zahtevima korisnika:

ISP1: "...sve to rešava za nas Microsoft Azure cloud tehnologija, gde mi samo publikujemo neke instance servisa, a Microsoft sam odredi gde će da ih stavi, na koji računar. Mi kažemo hoću 5 instanci ovog servisa da imam, kako bi se posao rasteretio, ako ima puno posla onda 5 paralelno ako radi biće brže i efikasnije. Mogu reći želim 5 ili 10 instanci ovog servisa, a Microsoft sam odredi gde će ih staviti, to mene ne interesuje. Tako da ta Azure cloud tehnologija umnogome olakšava posao... Tipa skaliramo ako nemamo šta da obrađujemo, stavimo najveći broj instanci na 1 ili 0, ako imamo dosta posla da obrađujemo, dosta podataka, onda ćemo povećati broj instanci na 10, 20, kako bi to brže završili. Sad postoji i način da se to automatski skalira, to opet Azure cloud nam to omogućava... potpuno je drugačiji princip razvoja sa Azure cloudom i bez njega...".
ISP10: "...u cloud rešenjima, lakše je osloniti se na postojeće platforme i postojeća rešenja, i virtualizovana rešenja, gde se dobija veća fleksibilnost, zato što je u samoj toj virtualizaciji lakše skalirati stvari i manje mora da se razmišlja o fizičkom hardveru, a više o tom virtualnom. I u tom kontekstu, uopšte pristup deployment-u sa virtualnim hardverom je mnogo fleksibilniji, jer uz neke možda malo veće troškove dobijate veću fleksibilnost, zato što ne morate da se vezujete za neku fizičku varijantu ili količinu servera, već možete da prema vašoj potrebi da vaše rešenje prilagođavate potrebama, tj zahtevima korisnika...".

Preporučljivo je prilikom razmatranja modela raspoređivanja, već na početku projekta uključiti i ljude koji obavaljaju operativne aktivnosti u ciljnoj organizaciji, zbog informacija koje mogu pružati, a koje imaju uticaja na postavku arhitekture:

ISP4: "... idealno po meni bi bilo da na početku involvirate ljude iz Ops-a da urade review vaše arhitekture i da zajedno dođete do modela raspoređivanja, zato što sad nemate više fizički hardver, sve je virtualizovano. U principu bitno je, jer vam ljudi koji rade na tom delu posla mogu dati neki input koji može da vam utiče na arhitekturu. Idealno bi bilo da su te stvari transparente... mislim da je dobro da up front znate to, i nekad i morate znati zbog sizinga hardvera, budžeta, i tako tih stvari, nekad vam i traže...".

Na početku projekta, prilikom postavke arhitekture, bitno je identifikovati i potrebu za nekim od mehanizama dizajna, dok se odluka o načinu njegove implementacije odlaže za kasnije, kada arhitekta bude raspolagao sa dovoljno informacija. Tako naprimer, mehanizam perzistentnosti značajan je na projektima koji koriste velike količine podataka, jer odgovara na zahteve kao što su brzina pristupa podacima, brzina zapisa podataka, obrasci za čitanje i pisanje i dr.:

ISP: "...radi se up front. To su primeri nonfuctional requirementa, to su oni crosscuting concern. kako ćete raditi logovanje, kako ćete error hendling, kako ćete da kolektujete metriku, kako ćete da znate da vam je node pao. Ja sam uvek da za to da koristim alate. Idealna moja aplikacija će da baca evente, neko drugi će to da skupi tj. specijalizovani alati koji to rade...nema svrhe to razvijati, sve te stvari su standardne. Vi morate da obezbedite support, mi sad imamo tendenciju da koristimo frejmorke, pošto nikad te stvari ne možete sve da isplanirate, pa zato koristite frejmvorke za koje je sedela grupa ljudi i smislila šta treba...".

Odluke o modelu implementacije spadaju takođe u značajne up front eksplicitne arhitekturalne odluke. Odluke imaju za cilj da eliminišu probleme vezane za upravljanje konfiguracijom i da omoguće nesmetanu implementaciju, integraciju i proces izgradnje:

ISP13: "...on baš dosta utiče i na samu arhitekturu i na razvoj, jer ako ste vi ti koji morate da nosite kompletnu poslovnu implementaciju, te neke troškove treba u startu anticipirati i minimizovati kroz samu arhitekturu softvera...".

Njegovo razmatranje na početku projekta, važno je zbog donošenje odluke o korišćenju otočnih servisa i za postavku inicijalnog modela podataka:

ISP8: "...kada se dođe do programiranja tada je već sve gotovo, već se sve zna. Ono o čemu jedino razmišljam kad mislim o arhitekturi je – da nešto izložim u modul za koji znam da je već negde napravljen da ne bi to mi pravili...".
ISP12:"...bitno mi je kako će taj interfejs da izgleda , jer to utiče na data model, npr.da li ćemo da biramo listu, više, multiple choice, inner choice,to utiče kako ćemo da pakujemo podatke u bazu...".

Razmatranje modela podata ispitanici takođe smatraju značajnom eksplicitnom arhitekturalnom aktivnošću, (28. eksplicitna arhitekturalna aktivnost u tabeli 5.2), koju treba sprovoditi od samog početka projekta, paralelno sa analizom poslovnih procesa i toka podataka ciljne oranizacije:

ISP20: "...potrebno je podatke analizirati unapred...imamo nekakve šeme, modele...insistiram da sve uđe u backlog, sve te aktivnosti moraju da budu deo backloga i ako tim na nečemu radi on mora da ima vremena da to uradi. Ne može to da radi u backgroundu...".
ISP8: "...model podataka je deo onoga što sistem može da izdrži...tip podataka ima svoje ograničenje i on je vrlo limitiran u onome što može da uradi. Baza podataka, koliko god je vi dobro indeksirali, ima granice do kojih vi možete da idete u bazu, a da sistem liči na nešto, da bude koliko toliko performantan. Ukoliko vi imate milione korisnika, desetine ili stotine, onda su baze neupotrebljive. Ali toga morate da budete svesni na početku tj. šta ćete spuštati u bazu, da ne upisujete sliku ili tako nešto. To je jako važno za performantnost sistema. Model podataka koji ćete odabrati je važan, jer i on mora da bude brz. Ako je sistem data driven, onda storidž mora da bude brz. Obično kod poslovnih softverskih aplikacija je upravo takav slučaj...".

Najveći identifikovani problem u praksi odnosi se na pitanje kompatibilnosti podataka, usled promena up front inicijalnog modela podataka. Model podataka se iz tog razloga često

posmatra kao interni model aplikacije, čiji se delovi klijentima daju na korišćenje, putem interfejsa:

ISP4: "...model podataka je za mene interni model, ako pričamo o domain modelu to je stvar koja će nastati. Ja neću da ga eskpozujem spolja, ako sam ga ekspozovo spolja kroz neki API, ja sam onda prinuđen da garantujem backward compatibility. Ako napravim neki model podataka i isti taj model podataka ekspozujem klijentima, a ima ih raznih i u datom momentu napravim novu verziju modela, mogu da budem u problemu ako korisnici neće da pređem na novu verziju. Time ste sebi limitirali mogućnost menjanja. Za mene je data model uvek bio interni za aplikaciju, delove njega ću da ekspozujem kroz neke interfejse, ali tu onda kontrolišem backward compatibility i u tom smislu ne moram da ga znam na početku projektu, sem u slučaju kada imamo bazu podataka koja već postoji, oni vam kažu mi imamo bazu podataka, vi onda sad gledate šta ćete da radite sa tim...".

Interpretirani rezultati kvalitativne analize podataka, iznedrili su set dodatnih značajnih eksplicitnih arhitekturalnih aktivnosti (prikazani u tabeli 5.5), koje su komplementarne sa kvantitativno identifikovanim aktivnostima iz tabele 5.2.

Tabela 5.5 Identifikovane eksplicitne arhitekturalne aktivnosti u okviru kategorije arhitekturalna sinteza

Sagledavanje ekspertize razvojnog tima
Identifikovanje komponenti koje su već razvijane i mogu biti ponovo korišćene
Identifikovanje potrebnih mehanizama dizajna, uz odlaganje odluke o načinu njihove implementacije
Definisanje podsistema i lejera za identifikovane module
Ključne odluke o modelu implementacije (dijagram komponenti, koji prikazuje identifikovane slojeve i alokaciju podsistema implementacije na slojeve, kao i uvozne zavisnosti između podsistema)

Arhitekturalna evaluacija

Ispitanici su pored standardnih agilnih praksi: code review, integracija koda, regresioni testovi, statička i dinamička analiza koda, istakli značaj i nekih tradicionalnih praksi: razvoj prototipa, vremenski ograničenog POC-a, formalni review od strane tima stručnjaka za arhitekturu. Istaknuta je primena i arhitekturalnih spike-ova, koja podrazumeva izdvajanje vremena za iteraciju kojom bi se arhitekturalno rešenje ili neki nejasni zahtevi potvrdili putem izvršnog koda, pre faze konstrukcije. Metrike i testovi su najbolji način sagladevanja arhitekture tj. ispunjenosti nefunkcionalnih zahteva, istakao je ispitanik ISP20: *"...od samog starta potrebno je postaviti pitanje kako ćemo mi znati da nam je arhitektura dobro postavljena posle nekog vremena? Kako ćemo meriti, koji će nam biti indikatori koji će reći da je rešenje koji smo izabrali adekvatno? Ako ne postoje neki podaci, neki testovi koje pokrećemo, ili neke aplikacije koje pišemo i koje nam daju neke rezultate, mi nećemo nikad znati nego će sve biti na nivou teorije ili nečijeg iskustva...".*

Primena ovih tehnika objašnjena je u delu poglavlja kojim se interpretiraju rezultati istraživanja vezanih za izbor arhitekturalnog rešenja. Navedene tehnike arhitekturalne evaluacije koriste se i za verifikaciju arhitekture u odnosu na arhitekturalno značajne zahteve.

Pored navedenih tehnika, kvalitativnom analizom podataka, dodatno je identifikovana tehnika "misaono istraživanje":

ISP9: "...najčešće se radi više o misaonoj vežbi, research u smislu pročita se šta se pročita, sam čovek ima tehnička iskustva iz nečega, ili nema, pa pita nekog koga ima i takva rešenja prezentujemo nekoj široj grupi i vidimo šta ta grupa misli i te se stvari uvrste...".

Dokumentovanje

Dokumentovanje spada u važnu eksplicitnu arhitekturalnu aktivnost, koja je prenaglašena u tradicionalnom razvoju, dok je u agilnim procesima gotovo zamenjena praksom "izvorni kod kao ultimativna dokumentacija", u skladu sa proklamovanim agilnim principom: "primenljiv softver iznad detaljne dokumentacije". Međutim, u razvoju kompleksnih poslovnih softverskih rešenja, kvalitetan kod ne može biti jedina praksa dokumentovanja, ali je važno sa merom inkorporirati neke od eksplicitnih praksi dokumentovanja.

Kvantitativni rezultati istraživanja pokazuju da agilni timovi smatraju važnim dokumentovanje arhitekture, jer su ovu eksplicitnu arhitekturalna aktivnost ocenili kao značajnu, sa proporcijom ocenjivača od 0.85 (tabela 4.18). Analizom kvalitativnih podataka, koji se odnose na identifikovanu kategoriju *arhitekturalno dokumentovanje* (slika 4.2), može se zaključiti da timovi izrađuju set neformalnih dokumenata softverske arhitekture:

ISP16: "...postoji neki arhitekturalni dokument za arhitekturu celog sistema. I postoje dokumenti koji su vezani za neke osnovne procedure prilikom razvoja softvera u vezi kodnih standarda, error handlinga, nekih stvari koje treba da budu zajedničke za kompletan softver. Tako da postoji neki set dokumenta koji zajedno pravi neku celinu, nije sve u jednom dokumentu. Nije ni prirodno da bude u jednom dokumentu...".

Za razliku od tradicionalnog načina dokumentovanja, koje podrazumeva izradu formalnog artifakta, sa pogledima koji opisuju različite perspektive arhitekture (npr. RUP 4+1 pogled), agilni timovi uglavnom koriste wiki i to bez neke unapred definisane forme:

ISP4: "...najviše za dokumentovanje koristimo wiki, i ne bih rekao da imamo neku eksplicitnu formu, već je i to agilno, radimo tako da i drugi razumeju...".

Donošenje arhitekturalnih odluka i obrazlaganje istih članovima tima, uglavnom se realizuje pred tablom, oko koje su svi članovi okupljeni. Tako je i najčešći vid dokumentovanja arhitekturalnih odluka i razloga, putem izrade neformalnih dijagrama na tabli, koji se slikaju i

u takvoj formi arhiviraju na wiki ili se putem mejla šalju stejkholderima sistema. Dokumenti u formi slika, koriste se i za komunikaciju i kolaboraciju sa stejkholderima:

ISP13: "...arhitekturu uglavnom dokumentujemo agilno ...Imamo crtež na tabli koji smo slikali, poslali smo ga na mejil svim učesnicima u projektu za koje je to bitno, ponekad ga ubacimo u wiki i uglavnom je to vid dokumentovanja arhitekture...".

Ispitanik ISP19 je istakao i upotrebu alata, kao podršku za sprovođenje aktivnosti dokumentovanja*: "...u većim projektima koristimo neke tool-ove za lakše održavanje dokumentacije...".* Najčešće korišćeni alati su Jira, Github, Word i Excel:

ISP7: "...nama se jako puno svodi na Jiru, u kojoj klijenti pišu user story...znači user story je neka šira slika šta bi korisnici želeli i onda se oni kasnije razbijaju na taskove, podtaskove. Ali oni nam ostaju u vrhu kao under umbrella, znači ispod toga ostane gomila taskova koja je uređena. Trudimo se da ne gubimo previše vremena na neku papirologiju, već da iz Jire izvučemo neki fini izveštaj...".
ISP8:"...Word i Excel su nam osnovni za pravljenje arhitekturalnih dokumenata, pa to šaljem i klijentu i programerima da ga vide... kroz GitHub na kraju formalizujemo konkretno zahteve, vrlo je praktičan i mnogo nam je pomogao...".

Tradicionalni razvoj karakteriše izrada formalnih modela i upotreba alata za tu svrhu, kao što su RUP i Enterprise Architect. Za razliku od tradicionalnog razvoja, gde je implementaciji prethodila sveobuhvatna izrada dijagrama, agilni timovi su istakli da je to gubitak resursa i vremena, iz razloga što njihov razvoj i održavanje nije moguć u uslovima turbolentnih promena i dinamičnog poslovanja:

ISP6: "...razvoj prave biznis aplikacije treba da bude toliko brz da uopšte nema potrebe da se troši vreme na pisanje dokumentacije. Zato što biznis toliko brzo evolvira i tim mora stalno da bude aktivan i da aktivno menja aplikaciju...".
ISP18: "...prvobitno smo imali u Rational Rose nacrtane dijagrame, pa onda smo u poslednje vreme krenuli da koristimo Tejlor, koji podržava UML 2 model. S tim što je osnovna ideja bila tu da krenemo od modela, pa da širimo dalje ka kodu, međutim, brzina razvoja je to malo okrenula, pa smo mi u suštini dobili neki naš XML fajl koji mapira taj model što se tiče baze, pa njega vraćamo nazad u model da bi to bilo i prikazano, da bi neko mogao i u grafičkom okruženju da shvati tu strukturu klasa. Nije baš najbolje rešenje, ali jednostavno brzina izmena nas je naterala...".
ISP17: "...jako je veliki effort potreban da se to održava, tu se stvari jako često menjaju, na kraju krajeva za ovako veliki proizvod mi smo imali pokušaja upotrebe UML dijagrama, ali su oni postali toliko veliki i komplikovanii struktura njihova da je bilo jako teško snaći se u tome...".

Takođe, ispitanici su istakli da vrlo malo koriste UML, ili neki drugi formalni jezik za modelovanje i dokumentovanje sistema, iz i razloga što nije efikasno sredstvo za komunikaciju sa stejkholderima:

ISP4: "...UML se izbegava jer ga poslovni ljudi ne razumeju...".

Osnovni UML dijagrami izrađuju se jedino u cilju komunikacije sa članovima tima, kada ne postoji bolji način da se neko rešenje predoči razvojnom timu:

ISP3: "...ja uglavnom koristim open source dijagrame. Dijagramom opisujem celu arhitekturu, sve module koji će biti uključeni u softversko rešenje...".
ISP2: "...imamo dokumente koji opisuju arhitekturu: crteži, UML dijagrami, opis interfejsa, dijagram sekvenci, dijagram data model neki, statički model komponenti, neke klase I tako. U principu nam dijagram sekvenci pomaže. Ne radima puno, radimo ono najosnovnije...".

Arhitekturalna dokumentacija uključuje opis problema koji se rešava, opis arhitekturalno značajnih funkcionalnih i nefunkcionalnih zahteva, kao i odluku o tehnološkom steku:

ISP9: "...o tehnološkim pitanjima moramo odluku da donesemo mnogo pre implementacije...dokumentovanje odluke o tehnološkom steku...".
ISP16: "...svaki dizajn dokumenta počinje sa time šta je to što pokušavamo da rešimo, znači neki popis zateva morate da bude i to se potvrdi od strane produkt menadžera...".
ISP11: "...u suštini pravimo dokumente za svaku funkcionalnost pojedinačno. Trenutno pišemo po jedan dizajn dokument za jednu funkcionalnost i izbacujemo te dokumente po svakoj verziji softvera. Dokument u suštini treba da sadrži jasne zahteve product owner-a za funkcionalnost...".
ISP4: "...nonfunctional requiremente obrazložim u smislu kako je taj zahtev relevantan za taj konkretan projekat, aplikaciju, sistem...".
ISP15:"... neki od nefunkcionalnih zahteva proističu iz opisnih elemenata šta oni očekuju od njihovog sistema u toku njihovog peiroda korišćenja i onda ih mi specifikujemo kao ne funkcionalne zahteve i u procesu verifikacije sa klijentom proveravamo njihovu validnost...".

Takođe, sadrži i postavku inicijalnog arhitekturalnog rešenja, do najviše srednjeg nivoa apstrakcije:

ISP1: "...uglavnom nemamo vremena, ni resursa da pišemo stvarno do najsitnijih detalja i delova, da specificiramo sve do interfejsa i do najsitnijih komponenti. Uglavnom to bude srednje apstraktno...".

Definisanje klasa i njihovih atributa i metoda nije uobičajena praksa, izuzev ako je sistem izuzetno kompleksan, a članovi razvojnog tima ujedno nedovoljno iskusni i nevešti u dizajnu:

ISP1: "...ako je to nešto jako kompleksno na tom nivou da bukvalno moramo specificirati svaki interfejs i svaku klasu i atribut, onda dolazimo do toga da pišemo dotle. Ako nešto gde je potrebno otprilike da se postavi koncept rešenja i da se postavi taj neki high level arhitektura i odakle je već jasno nekome kako treba napisati klase, tu se onda zaustavljamo kod ključnih podsistema. Onda se high level arhitektura predstavi dijagramom tog nekog globalnog rešenja...".

Arhitekturalni dokument najčešće sadrži i neku inicijalnu postavku modela raspoređivanja, inicijalni model baze podataka. Tokom razvoja dokumentuju se informacije koje se tiču analize korisničke priče koja se implementira. Najčešće se unose primeri pozitivnog i negativnog testa za datu korisničku priču, sagledava uticaj na sigurnosne aspekte arhitekture i sl.:

ISP1: "...sadrži odgovore na pitanja sa aspekta deploymenta, da li će taj user story uticati na drugačiji deployment softvera i na koji način...da li će taj user story koji se implementira uticati na QA, znači na testiranje, da li treba specifično nešto da se testira, i ako treba šta. Znači navesti neke primere pozitivnog i negativnog testa. Da li taj user story kada se implementira, da li će uticati na sequrity, znači kako sad sve te komponente ili jedna komponenta, koliko god komponenti da se razvija tokom pravljenja tog user storiy-a, da li će te komponente na neki način imati uticaj na sequrity...".
ISP1: "...svaki arhitekta je trebao da nabroji neke pozitivne i negativne use case-ove, da bi se testirala ta funkcionalnost, trebalo bi opisati svaku komponentu, tipa i bazu podataka, isto dijagrame tabela i tako neke stvari...".

Ispitanik ISP14 proces dokumentovanja počinje kroz pisanje testova prihvatljivosti u Word dokumentu, a završava testovima integracije, koji su krajnja specifikacija proizvoda:

ISP14: "...pišemo acceptance testove bez njihove automatizacije. Mi radimo integration testing u tom nekom behaviour stilu, međutim, ono što se ispostavilo je da kada se koristi kombinacija koda i toga, onda to nije razumljivo biznis osobama. Tako da smo krenuli sa acceptance-om u word formatu i to jako dobro radi posao u samoj analizi. Odnosno kada krenemo da radimo na tiketu imamo acceptance kriterijume, imamo acceptance testove i onda otprilike znamo sve šta treba da uradimo. Acceptance testovi su dokumentacija koja kada je story implementiran više nisu toliko validni, zato što su automatizovani. Jer automatizovani su kroz integration i samim tim na kraju integration test je krajnja specifikacija proizvoda i on je biznis pravilo..." .

Nekolicina ispitanika je istakla da je poželjno i korisno da arhitekturalna dokumentacija prođe formalni pregled i verifikovanje, od strane tima stručnjaka, i da se tek nakog njenog odobravanja otpočne faza implementacije rešenja. Međutim, u praksi to uglavnom nije slučaj:

ISP11: "... taj dokument kad ga jednom napiše softver arhitekta, nije dobra praksa da se uploaduju na neki interni sajt i to je to, nego je poželjno da se on šalje na review drugim softver arhitektama, QA šefu i sequrity šefu, da ga verifikuju. I ako oni kažu to je to, na kraju se šalje i glavnom šefu svih arhitekti na aproval. Kada i on pročita ceo taj dokument i odobri ga, onda je on ok za implementaciju. Tek posle aprovala bi trebalo da se sme krenuti u implementiranje user storiy-a, ali to nije slučaj, zato što glavni šef za arhitekturu nema vremena da pročita taj dokument i svaki dokument je ostao tako da čeka aproval...".

Značaj arhitekturalnog dokumentovanja ispitanici vezuju i za uključivanje novih članova u tim:

ISP1: "...da kad neko dođe sledeći mu samo damo dokument i on bude upućen u to šta je šta i gde se nalazi, da ne moramo sve iz glave da mu prenosimo...".

Kvantitativnom analizom odgovora ispitanika utvrđeno je da eksplicitna arhitekturalna aktivnost, *dokumentovanje detaljnog dizajna,* ne spada u prakse koje treba inkorporirati u agilne procese razvoja. Proporcija ocenjivača koji su ovu praksu ocenili kao značajnu bila je 0.1. Drugim rečima, postignuti nivo saglasnosti eksperata, po pitanju njene beznačajnosti, iznosio je 90%.Visok procenat slaganja ekserata implicira zaključak da praktičari dokumentovanje detaljnog dizajna smatraju "prekomernim" za agilne procese. Jedan od razloga je resurs vremena i napora, a drugi je svakako i agilan tim u kojem programeri moraju biti vešti da samostalno, ili eventualno uz tehničko mentorstvo softver arhitekte izrade detaljan dizajn:

ISP16: "...ako smo stvarno agilni, i uz to imamo i developere kojima ne treba detaljni dizajn...".
ISP3: "...dijagrami klasa ne...ja lično ne volim da idem u toliko detaljisanje, to je deo koji volim da ostavim programerima, seniorima ili vođama grupa, timova koji će u svojim oblastima zadataka ići detaljnije u ceo razvoj...".

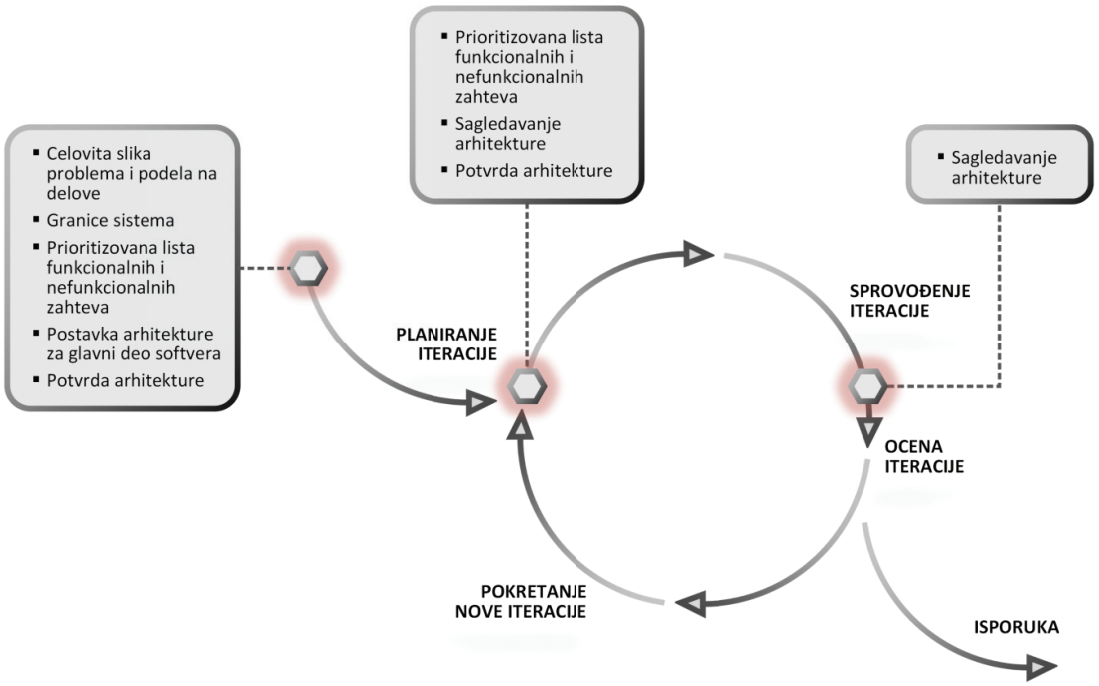
5.1.2 C2: Identifikovana kritična mesta u agilnom procesu razvoja pogodna za primenu tradicionalnih arhitekturalnih praksi

Na osnovu rezultata, sprovedenog kvalitativno-kvantitativnog empirijskog istraživanja, identifikovano je 6 kritičnih mesta agilnih procesa razvoja, sa aspekta arhitekture. Identifikovana kritična mesta u agilnom razvoju, prikazana su na slici 5.1 i predstavljaju odgovor na drugo istraživačko pitanje:

IP2. Koja su kritična mesta u procesu agilnog razvoja koja je potrebno proširiti tradicionalnim arhitekturalnim praksama?

U nastavku se daje grafički prikaz (od dijagrama 5.1, do dijagrama 5.6) empirijskih rezultata, na osnovu kojih su utvrđene pojedine kritične tačke agilnog razvoja. Svaka identifikovana kritična tačka, podrazumeva inkorporiranje seta značajnih eksplicitnih arhitekturalnih aktivnosti, prilikom razvoja kompleksnih poslovnih softverskih rešenja, čime se uspostavlja arhitekturalni proces za razvoj agilne arhitekture:

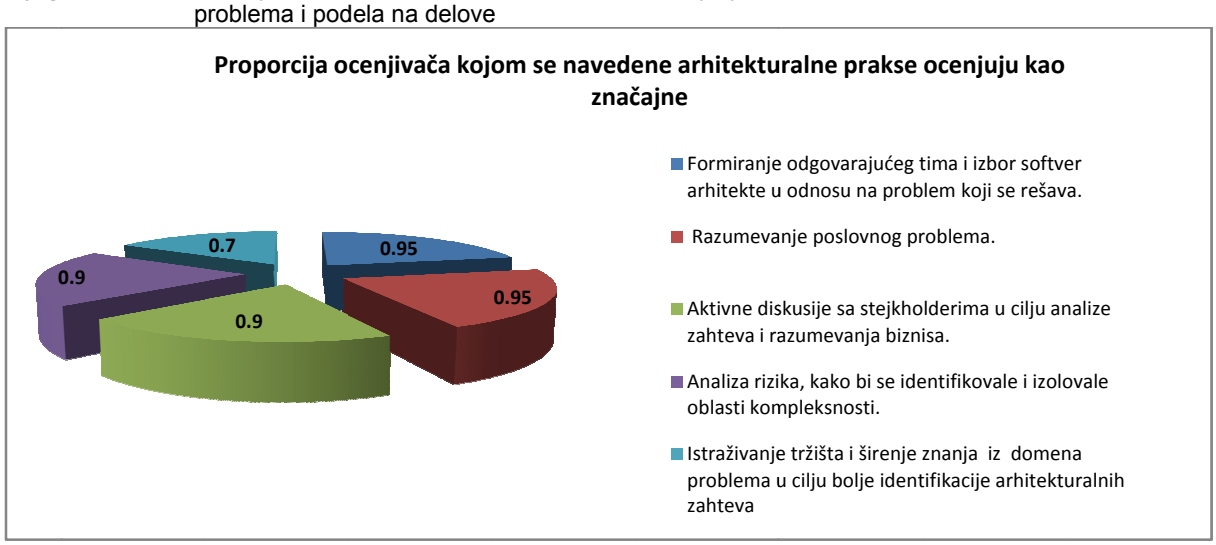
1. Celovita slika problema i podela na delove.
2. Granice sistema.
3. Postavka arhitekture za glavni deo softvera.
4. Potvrda arhitekture.
5. Prioritizova lista funkcionalnih i nefunkcionalnih zahteva.
6. Sagledavanje arhitekture.



Slika 5.1 Kritične tačke agilnog razvoja
Izvor: Prilagođeno prema (Matković, 2011)

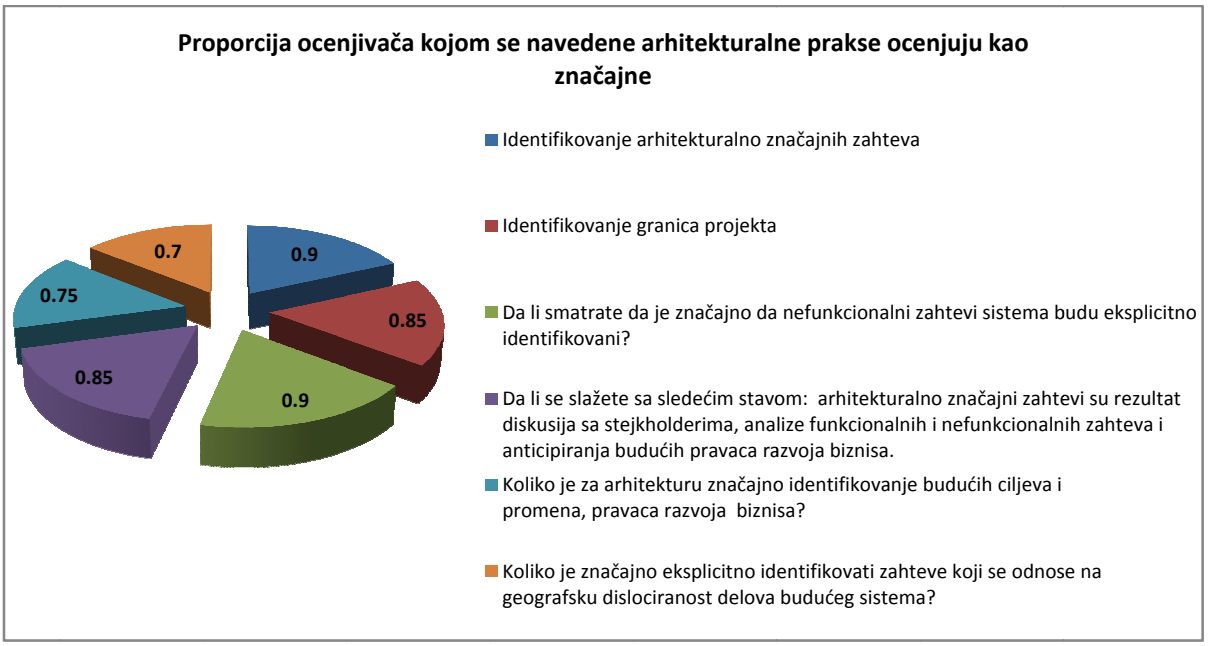
Celovita slika problema i podela na delove predstavlja kritičnu tačku u agilnim procesima razvoja, na koju je potrebno inkorporirati set eksplicitnih arhitekturalnih aktivnosti, pre početka implementacije funkcionalnosti. Agilna mantra "just do it", ne može se slediti na razvoju kompleksnih poslovnih softverskih rešenja, jer je neophodan set up front arhitekturalnih odluka, koje će obezbediti celovitu sliku problema, na osnovu koje će se postaviti celovita slika sistema koji se razvija. Celovita slika sistema obezbeđuje jasnu viziju programerima u fazi implementacije i u skladu je sa Lean konceptom "optimizuj celinu". Ova tačka u agilnim procesima razvoja značajna je, kako na samom početku projekta, tako i kasnije u razvojnom procesu, prilikom planiranja budućih release-ova. Dijagram 5.1 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

Dijagram 5.1 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: celovita slika problema i podela na delove



Granice sistema je naziv za drugu kritičnu tačku agilnih procesa razvoja i podrazumeva identifikovanje ključnih stejkholdera sistema, aktivne diskusije sa njima, identifikovanje budućih ciljeva i pravaca razvoja biznisa, analizu rizika i identifikovanje arhitekturalno značajnih zahteva sistema. Ova tačka u agilnim procesima razvoja značajna je, kako na samom početku projekta, tako i kasnije tokom razvojnog procesa, prilikom planiranja budućih release-ova. Dijagram 5.2 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

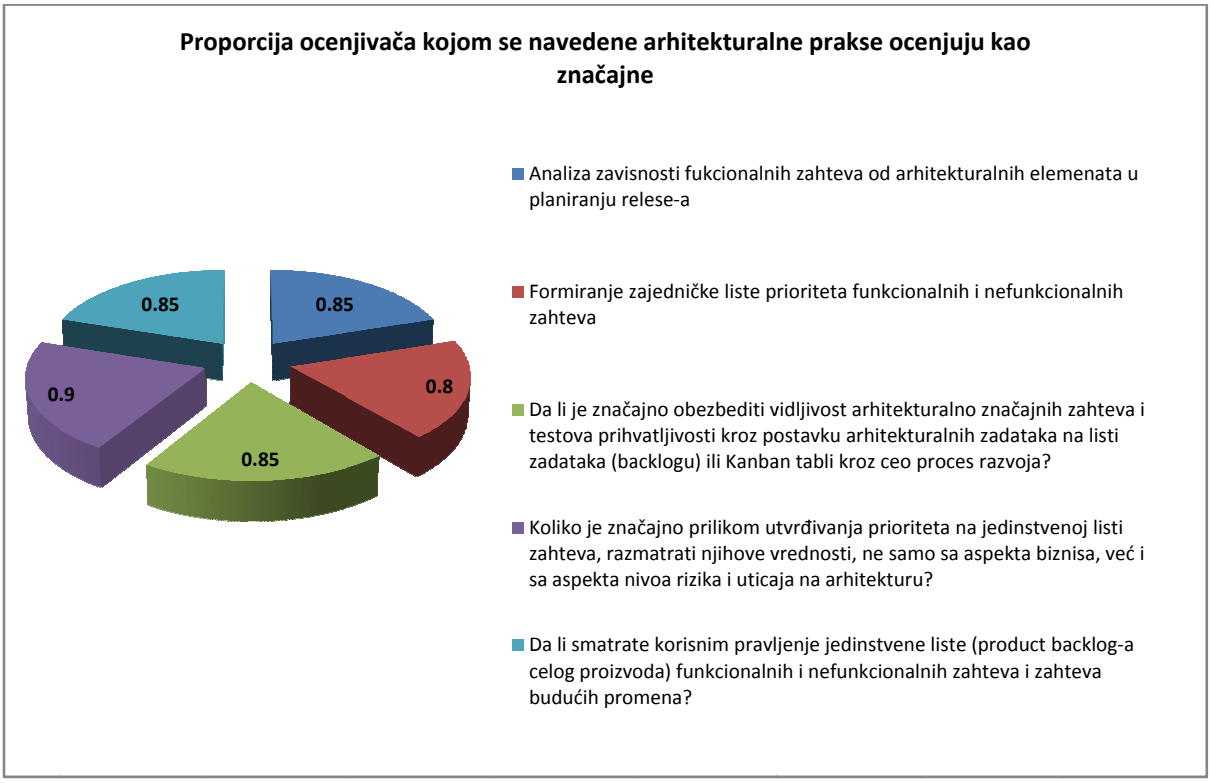
Dijagram 5.2 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: granice sistema



Prioritizovana lista funkcionalnih i nefunkcionalnih zahteva predstavlja kritičnu tačku agilnih procesa, iz razloga što agilan razvoj u fokusu ima identifikaciju i razvoj pre svega funkcionalnih zahteva. Nefunkcionalni zahtevi, u agilnim procesima, sagledavaju se i ocenjuju tek u fazi održavanja sistema, što je u skladu sa strategijom nascentne arhitekture. Međutim, nascentna arhitektura nije dovoljna arhitekturalna praksa, u slučaju razvoja kompleksnih poslovnih softverskih rešenja. Kritična tačka, prioritizovana lista funkcionalnih i nefunkcionalnih zahteva, podrazumeva inkorporiranje seta eksplicitnih arhitekturalnih aktivnosti i odluka: formiranje jedinstvene liste funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena; utvrđivanje prioriteta na jedinstvenoj listi zahteva, razmatranjem njihove vrednosti sa aspekta biznisa, ali i analize sa aspekta rizika i uticaja na arhitekturu; analizu zavisnosti funkcionalnih i nefunkcionalnih zahteva i upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem intereaguje tokom release-a. Opisana kritična tačka ima za cilj da obezbedi vidljivost i status arhitekturalnih zadataka, tokom celog perioda trajanja projekta. Tačka je značajna kako za početak projekta, tako i za događaj planiranja relese-a i planiranje iteracije. Nivoi planiranja razlikuje se po arfikatima koji se generišu. Tako se na početku projekta izrađuje artifakt prioritizovana lista poslovnih zahteva, tokom

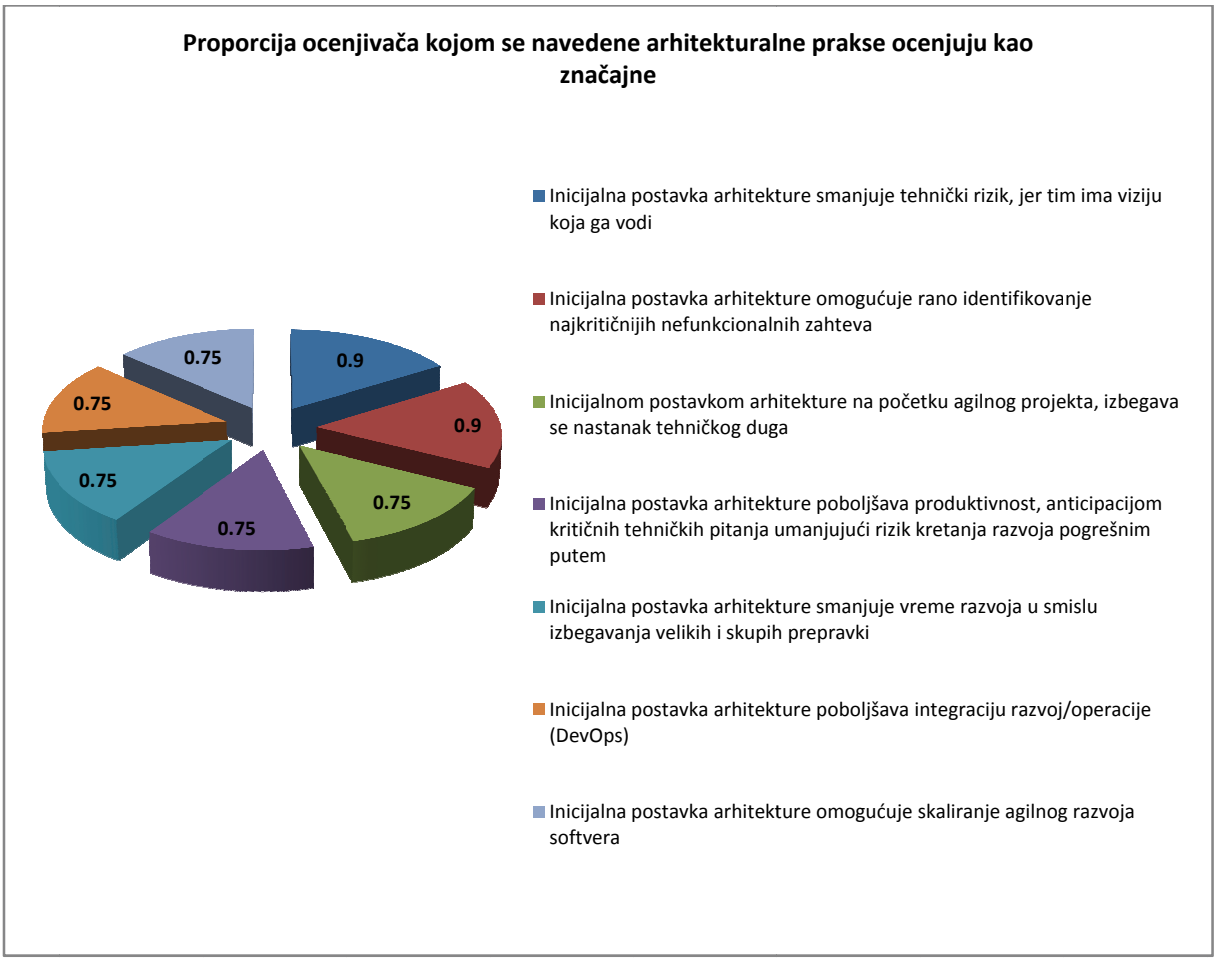
planiranja release-a - artifakt prioritizovana lista zahteva release-a, a tokom planiranje iteracije - artifakt prioritizovana lista zadataka iteracije. Dijagram 5.3 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

Dijagram 5.3 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: prioritizovana lista funkcionalnih i nefunkcionalnih zahteva



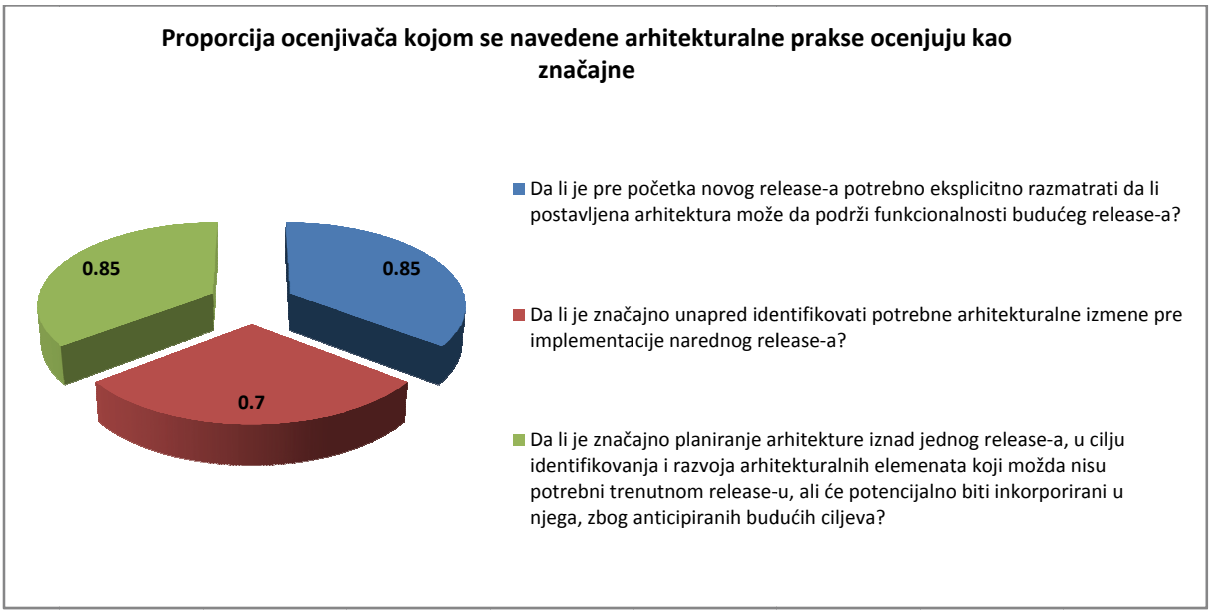
Postavka arhitekture za glavni deo softvera je kritična tačka u agilnim procesima razvoja koja podrazumeva postavku arhitekturalnog skeleta. Tačka obuhvata set eksplicitnih arhitekturalnih aktivnosti, kao što su: istraživanje tehnologije pogodne za implementaciju; istraživanje odgovarajućeg frejmvorka za implementaciju; istraživanje third party biblioteka; identifikovanje i analizu potencijalnih arhitekturalnih rešenja; identifikovanje i definisanje osnovnih struktura (modula) za glavni deo sistema i njihovih veza; definisanje osnovne arhitekture podataka; postavku inicijalnog modela raspoređivanja; identifikovanje tački arhitekture koje trebaju biti fleksibilnije u cilju budućih promena i odluka, u kontekstu proširivosti softvera spram budućih zahteva; identifikovanje arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera. Tačka je značajna kako na samom početku projekta, tako i za događaj planiranja relese-a. Dijagram 5.4 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

Dijagram 5.4 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: postavka arhitekture za glavni deo softvera



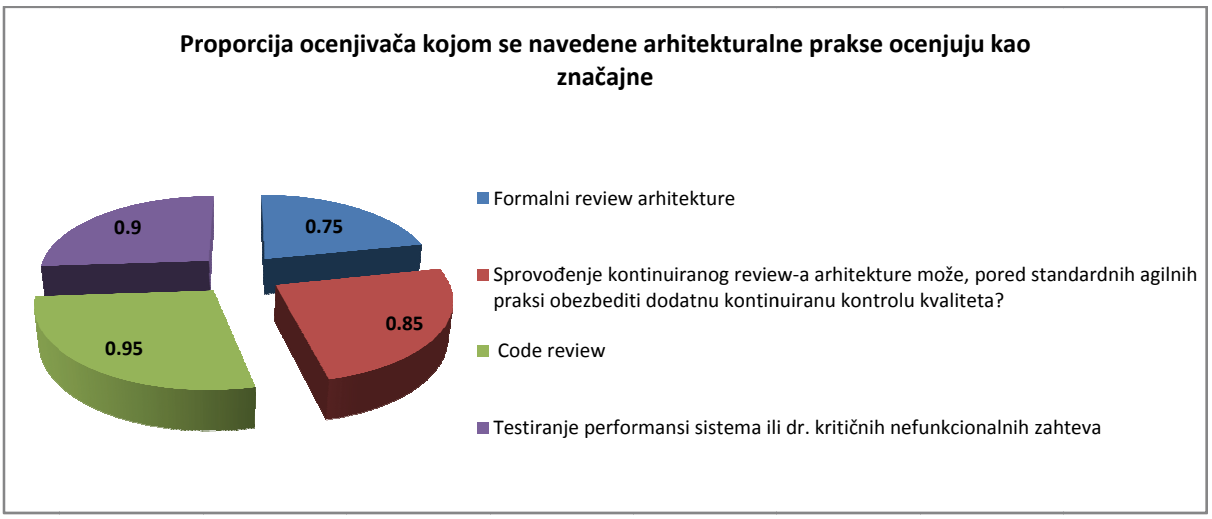
Sagledavanje arhitekture je kritična tačka agilnih procesa i to, kako na nivou release-a, tako i na nivou iteracija. Njeno mesto u agilnom razvoju vezuje se kako za događaj planiranja tako i za događaj ocene iteracije/relelease-a. Fokus ove kritične tačke u agilnom razvoju usmeren je na razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a; na identifikciju potrebnih arhitekturalnih izmena pre implementacije narednog release-a; na utvrđivanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti; na razmatranje nasleđenog sistema i zavisnosti od partnerskih ili otočnih proizvoda i kompatibilnost podataka; na ažuriranje liste rizika. Dijagram 5.5 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

Dijagram 5.5 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: sagledavanje arhitekture



Potvrda arhitekture: je kritična tačka agilnih procesa koja podrazumeva inkorporiranje eksplicitnih arhitekturalnih aktivnosti kao što je izrada prototipa, arhitekturalnog spike-a, testiranje performansi sistema ili dr. kritičnih nefunkcionalnih zahteva, formalni review arhitekture (u cilju validacije kritičnih arhitekturalnih zahteva i koncepata dizajna), diskusije vlasnika proizvoda, razvojnog tima i ključnih stejkholdera sa vlasnikom arhitekture. Dijagram 5.6 prikazuje empirijske rezultate na osnovu kojih je utvrđena data kritična tačka.

Dijagram 5.6 Značajne arhitekturalne aktivnosti na osnovu kojih je utvrđena kritična tačka: potvrda arhitekture



5.2 C3: Razvijen radni okvir za inkorporaciju tradicionalnih arhitekturalnih praksi u agilni proces razvoja

Stav da niska ceremonijalnost i visoka iterativnost, kao ključne karakteristike agilnog razvoja, nemaju svojstva panacea u razvoju kompleksnih poslovnih softverskih rešenja, potvrđuju dobijeni empirijski rezultati. Drugim rečima, jedini način da se agilni procesi stave u funkciju rešavanja izazova koje im nameće razvoj visoko kompleksnih sistema, jeste prilagođavanje i harmonizacija seta identifikovanih ekspicitnih (tradicionalnih) arhitekturalnih aktivnosti, pre njihove inkorporacije u agilni proces razvoja, čime se ne bi narušili osnovni principi agilnosti.

Realizacijom prvog i drugog istraživačkog cilja stvoreni su preduslovi za realizaciju trećeg istraživačkog cilja odnosno, postavku metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima.

Inkorporiranjem identifikovanih, značajnih eksplicitnih arhitekturalnih aktivnosti (prvi istraživački cilj), na odgovarajuća mesta u agilnom procesu razvoja (drugi istraživački cilj), putem predloženog okvira (treći istraživački cilj), formalizovan je proces razvoja agilne arhitekture, čime se daje odgovor na treće istraživačko pitanje:

IP3. Kako izabrati i integrisati tradicionalne arhitekturalne prakse u agilni proces razvoja poslovnog softvera?

Predloženi okvir uspostavlja arhitekturalni proces, koji zajedno sa agilnim procesom razvoja, omogućava razvoj agilne arhitekture. Agilna arhitektura podrazumeva arhitekturu koja je tako dizajnirana da može biti lako izmenjena tj. može da reaguje na promene zahteva u okruženju. Formalizovan arhitekturalni proces ima za cilj iterativno-inkrementalni razvoj agilne arhitekture, u kojem je nascentna arhitektura zasnovana na dovoljnom obimu up front arhitekturalnih odluka.

Okvir reprezentuje arhitekturalni proces u agilnom razvoju, prikazom toka arhitekturalnih aktivnosti, uloga koje ih realizuju i artifakata koji ih dokumentuju. Primenom okvira teži se uspostavljanju balansa između agilnog i tradicionalnog načina razvoja softverske arhitekture, tačnije, između primene eksplicitnih arhitekturalnih praksi i agilnosti razvojnog procesa.

Narednim tekstom daje se tumačenje (smisao) ključnih pojmova okvira:

- Uloga - predstavlja centralni koncept u procesu. Defiše ponašanje i odgovornost osobe ili grupe, odnosno tima koji radi zajedno. Ponašanje predstavlja aktivnost koje uloge izvršavaju, a svaka uloga je zadužena za određeni set aktivnosti.

- Artifakt - je informacija koju modifikuje ili koristi neka aktivnost. Koristi se kao inputi za realizaciju aktivnosti od strane uloga, a ujedno je i izlazni rezultat neke druge aktivnosti.
- Aktivnost - aktivnost je deo procesa rada koju neka uloga izvršava. Aktivnosti se odnose na kreiranje i modifikovanje artifakata i svaka aktivnost je sastavni deo jedne uloge.

Predloženi metodološko-radni okvir podrazumeva sve standardne uloge agilnih timova, koje su u radu već opisane, prilikom teoretskog razmatranja i analize agilnih procesa razvoja (treće poglavlje rada). Razvoj kompleksnih poslovnih softverskih rešenja zahteva, međutim, i dodatnu formalnu ulogu, koja je zadužena za softversku arhitekturu.

U postavljenom okviru ova uloga nazvana je vlasnik arhitekture (pojedinac/tim), iz razloga što su empirijski podaci pokazali da u praksi postoje uloge kao što su softver arhitekta, arhitekta rešenja, sistem arhitekta i sl.

Vlasnik arhitekture je istovremeno vrlo iskusan programer, te se iz tog razloga uključuje u razvojni tim na početku projekta, kako bi sa programerima postavio glavni deo softvera. Razvoj kompleksnih softverskih rešenja, zahteva njegovo angažovanje kroz ceo razvojni proces, u formi mentora, koji savetuje i pomaže programerima u rešavanju arhitekturalnih pitanja i problema.

Vlasnik arhitekture mora posedovati vrhunsko tehničko znanje i solidno znanje poslovnog domena. Okvirom se definišu sledeće odgovornosti vlasnika arhitekture:

- da na početku projekta, zajedno sa klijentom, identifikuje inicijalne arhitekturalno značajne zahteve,
- da postavi inicijalno arhitekturalno rešenje za glavni deo softvera,
- da kontinuirano sagledava arhitekturu u smislu ispunjenosti nefunkcionalnih zahteva sistema (putem kontinuirane integracije koda, seta metrika i testova),
- da ima dobru kolaboraciju sa članovima tima u cilju deljanja ideja i rešavanja problema
- da vodi tehničke diskusije sa ostalim članovima tima,
- da uči ostale članove tima i pruža mentorstvo u rešavanja arhitekturalnih problema tokom faze implementacije,
- da razume postojeću infrastrukturu, standarde i tehnička rešenja ciljnih organizacija,
- da poznaje mogućnosti i ograničenja što većeg broja tehnologija,
- da prati trend u smislu mogućih opcija arhitekturalnog rešenja,
- da obezbedi vidljivost arhitekturalnih zahteva na listi zahteva proizvoda i listi zahteva iteracije,
- da od početka projekta upravlja tehničkim dugom,
- da pravovremeno donosi odluke.

Narednim tabelama dat je prikaz osnovnih elemenata postavljenog metodološko-radnog okvira: eksplicitne arhitekturalne aktivnosti i uloge koje ih realizuju, prema kritičnim mestima u agilnom razvoju.

Tabela 5.6 Kritična tačka celovita slika problema i podela na delove

Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Celovita slika problema i podela na delove	Ključni stejkholderi, vlasnik proizvoda, vlasnik arhitekture	Razmatranje poslovne i informacione arhitekture ciljne organizacije
		Razmatranje softverske i tehničke infrastrukture ciljne organizacije i identifikovanje elemenata koji mogu biti ponovo korišćeni
		Identifikovanje ključnih stejkholdera sistema
		Aktivne diskusije sa ključnim stejkholderima u cilju razumevanja poslovanja
		Opis problema i ciljeva njegovog rešavanja
		Identifikovanje poslovnih potreba i ključnih poslovnih slučajeva upotrebe
		Formiranje inicijalne liste rizika
		Inicijalno formiranje odgovarajućeg tima

Tabela 5.7 Kritična tačka granice sistema

Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Granice sistema	Ključni stejkholderi, vlasnik proizvoda, vlasnik arhitekture	Aktivne diskusije sa ključnim stejkholderima u cilju analize zahteva
		Ažuriranje inicijalne liste rizika
		Identifikovanje budućih ciljeva i pravaca razvoja biznisa
		Identifikovanje ograničenja koja će biti nametnuta budućem sistemu - potiču iz realnog okruženja i mogu biti različitog porekla: politička, ekonomska, pravna, tehnička, funkcionalna
		Identifikovanje zahteva koji potiču od geografske dislociranosti delova budućeg sistema
		Identifikovanje arhitekturalno značajnih zahteva sistema (funkcionalnih i nefunkcionalnih)

Tabela 5.8 Kritična tačka prioritizovana lista funkcionalnih i nefunkcionalnih zahteva

Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Prioritizovana lista funkcionalnih i nefunkcionalnih zahteva	Ključni stejkholderi, vlasnik proizvoda, vlasnik arhitekture	Formiranje jedinstvene liste funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena
		Utvrđivanje prioriteta na jedinstvenoj listi zahteva razmatranjem njihove vrednosti sa aspekta biznisa, ali i analize sa aspekta rizika i uticaja na arhitekturu
		Ažuriranje inicijalne liste rizika
		Dodavanje članova tima
		Analiza zavisnosti funkcionalnih i nefunkcionalnih zahteva
		Identifikovanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti
		Analiza međusobne zavisnosti funkcionalnih zahteva
		Identifikovanje arhitekturalnih elemenata koji će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera.
		Analiza zavisnosti koje potiču od spoljnih sistema sa kojima sistem intereaguje

Tabela 5.9 Kritična tačka postavka arhitekture za glavni deo softvera

Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Postavka arhitekture za glavni deo softvera	Vlasnik arhitekture, razvojni tim	Istraživanje tehnologije i trenda opcija; istraživanje odgovarajućeg frejmvorka za implementaciju; istraživanje postojećih biblioteka
		Analiza postojećih arhitekturalnih rešenja
		Analiza potencijalnih arhitekturalnih rešenja
		Identifikovanje komponenti koje su već razvijane i mogu biti ponovo korišćene
		Ažuriranje inicijalne liste rizika
		Formiranje konačnog tima
		Sagledavanje ekspertize i znanja razvojnog tima
		Identifikovanje i definisanje osnovnih struktura (modula) za glavni deo sistema i njihovih veza
		Postavka inicijalne arhitekture podataka
		Postavka inicijalnog modela raspoređivanja
		Identifikovanje tački arhitekture koje trebaju biti fleksibilne Identifikovanje potrebnih mehanizama dizajna, uz odlaganje odluke o načinu njihove implementacije Definisanje podsistema i lejera za identifikovane module
		Ključne odluke o modelu implementacije

Tabela 5.10 Kritična tačka sagledavanje arhitekture

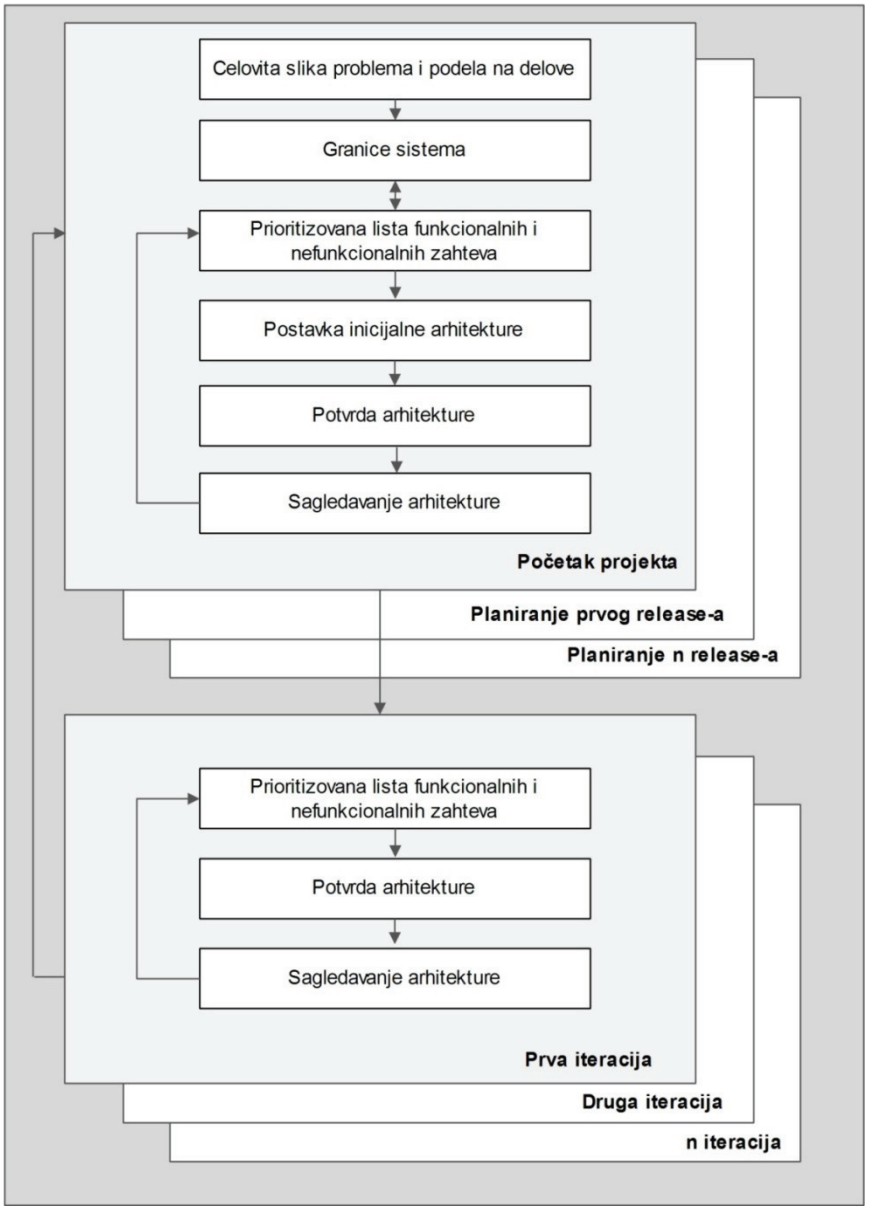
Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Sagledavanje arhitekture	Vlasnik arhitekture, razvojni tim	Razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a/iteracije
		Identifikovanje potrebnih arhitekturalnih izmena pre implementacije narednog release-a/iteracije
		Ažuriranje inicijalne liste rizika
		Ažuriranje prioritizovane liste funkcionalnih, nefunkcionalnih i zahteva budućih promena
		Identifikovanje arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera
		Utvrđivanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti

Tabela 5.11 Kritična tačka potvrda arhitekture

Kritična tačka agilnih procesa	Uloge	Eksplicitne arhitekturalne aktivnosti
Potvrda arhitekture	Ključni stejkholderi, vlasnik proizvoda, vlasnik arhitekture, razvojni tim	Code review
		Formalni kontinuirani review arhitekture u cilju validacija kritičnih arhitekturalnih zahteva i koncepata dizajna
		Testiranje kritičnih nefunkcionalnih zahteva
		Ažuriranje inicijalne liste rizika
		Tehnikama: test case-ova, testova prihvatljivosti, QA testova, testova integracije, regresionih testova; statička analiza koda, mišljenja architectural board-a (eksperti za razne aspekte: stručnjaci za raspoređivanje, stručnjaci za sigurnost, za kvalitet i dr. razne aspekte arhitekture); scenario analiza interakcije stejkholdera sa sistemom, sa fokusom na nefunkcionalne zahteve; arhitekturalnih spike-ova; prototipova; simulacija opterećenja buduće arhitekture sistema; vremenski ograničen proof of concept, eksperiment

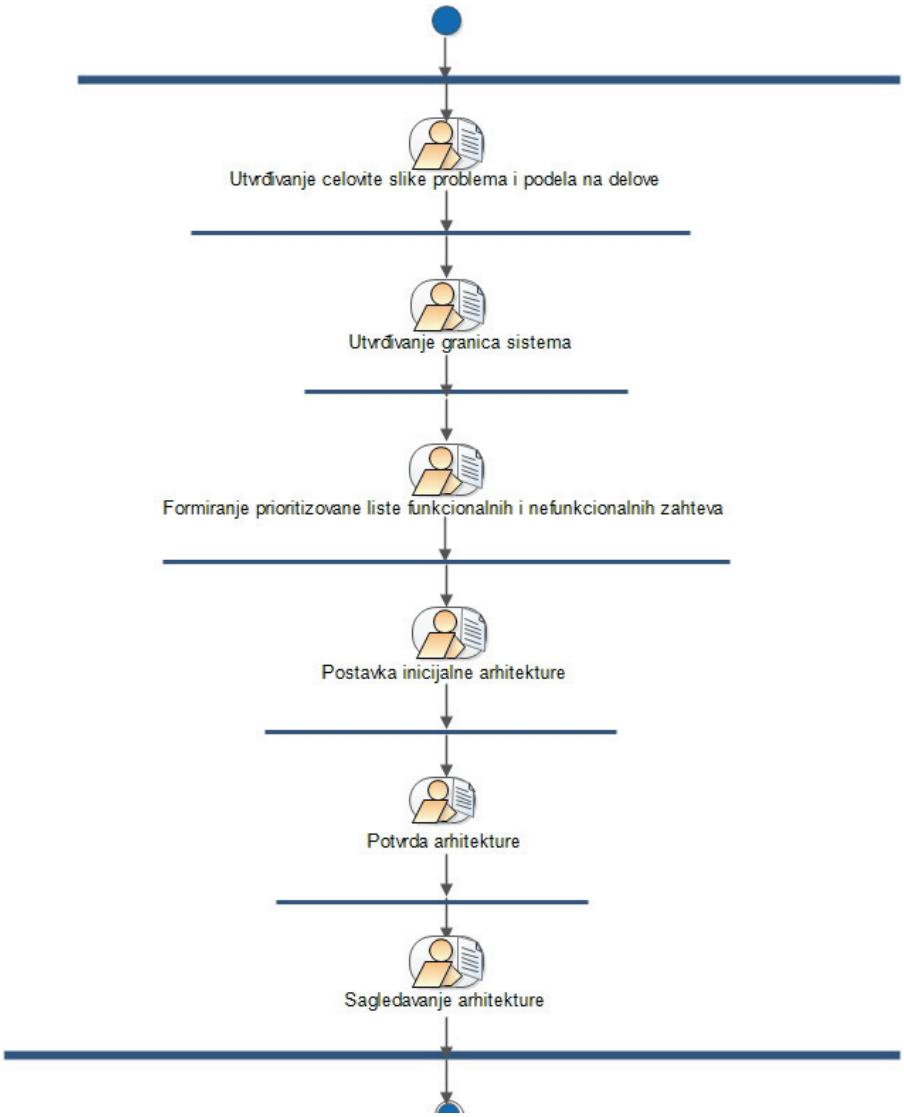
Na slici 5.2 dat je prikaz metodološko-radnog okvira na konceptualnom nivou, koji ima za cilj da pokaže, da su svih 6 identifikovanih kritičnih tačaka u agilnim procesima razvoja, sa setom eksplicitnih arhitekturalnih aktivnosti, neophodni na samom početku razvoja kompleksnih poslovnih softverskih rešenja, kao i prilikom planiranja release-a. Koraci od 3 do 6 (sa slike 5.2) sprovode se sve dok inicijalno arhitekturalno rešenje ne bude potvrđeno, u smislu da ispunjava inicajni set ključnih arhitekturalno značajnih zahteva (korak 3 na slici 5.2). Provera inicijalne arhitekture (korak 5 na slici 5.2), podrazumeva izradu izvršnog koda za ključne elemente postavljene arhitekture, kroz nekoliko iteracija, pre otpočinjanja faze implementacije sistema.

Planiranjem na nivou iteracija u agilan razvoj uključuje se set eksplicitnih arhitekturalnih aktivnosti, putem 3 kritične tačke, što je na slici 5.2 predstavljeno kroz tri koraka. Drugi korak se realizuje i tokom izvršavanja iteracije, dok se treći realizuje i na samom kraju iteracije. Drugim rečima, iteracije počinju planiranjem prioritizovanih lista zadataka tima, kojima se realizuju i funkcionalni i nefunkcionalni zahtevi tekuće iteracije. Tokom izvršavanja iteracije implementiraju se arhitekturalni elementi koji treba da ispune nefunkcionalne zahteve iteracije. Završetkom iteracije, neophodno je analizirati implementiranu arhitekturu (korak 3 na slici 5.2), identifikovati nedostatke i sl. Analiza arhitekture na kraju iteracije implicira ažuriranje liste zadataka za narednu iteraciju, ali i ažuriranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda.



Slika 5.2 Metodološko radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima – konceptualni nivo
Izvor: Autor

Slika 5.3 predstavlja korake metodološko-radnog okvira, koji će u nastavku rada biti objašnjeni kroz tri elementa: aktivnosti koraka, uloge koje realizuju aktivnosti koraka i artifakti koji se pritom generišu. Bitno je naglasiti da predloženi okvir podrazumeva obaveznu realizaciju svih ključnih arhitekturalnih aktivnosti (u okviru svakog koraka okvira) prilikom razvoja kompleksnih poslovnih softvreskih rešenja. Okvir nudi i preporuke za način njihove realizacije, u vidu seta podatktivnosti, koje nisu obavezujuće, već preporučljive. Realizacija preporučenih podaktivnosti zavisiće od tipa i konteksta projekta, kao i od znanja i iskustva vlasnika arhitekture i razvojnog tima.



Slika 5.3 Koraci metodološko- radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima
Izvor: Autor

Simboli grafičkog prikaza okvira:

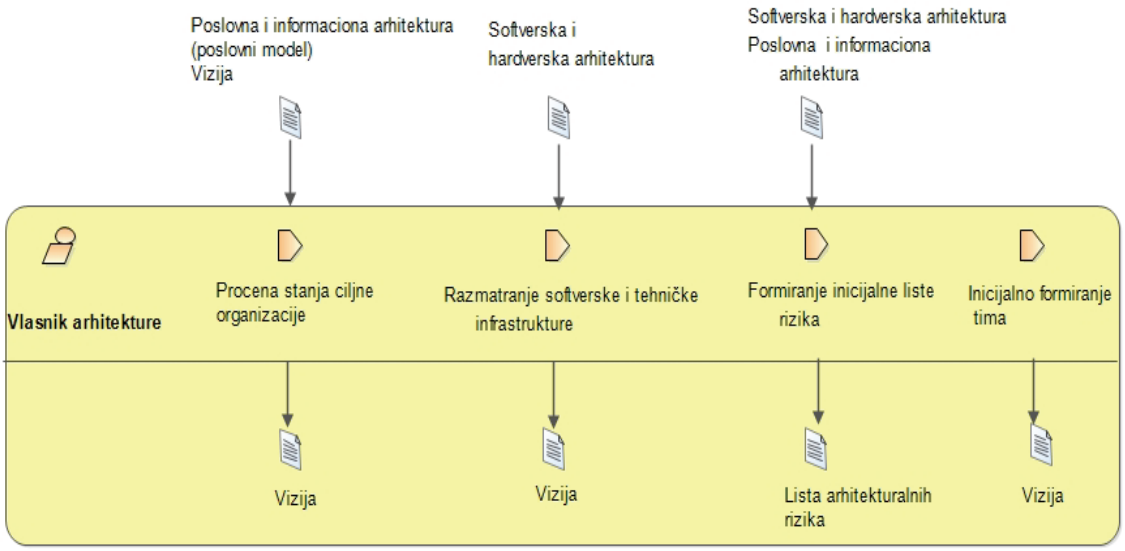
aktivnost

artifakt

uloga

Svaki korak sa slike 5.3 razrađen je dijagramima sa prikazanim aktivnostima koraka, ulogama koje ih realizuju i artifaktima.

KORAK 1: utvrđivanje celovite slike problema i podela na delove



Slika 5.4 Uloge, aktivnosti i artifakti u realizaciji prvog koraka: utvrđivanje celovite slike problema i podela na delove
Izvor: Autor

Izlazni artifakt koraka utvrđivanje celovite slike problema i podela na delove jeste dokument vizija sistema koji je dat u prilogu 16.

Korak utvrđivanje celovite slike problema i podela na delove, sastoji se od sledećih aktivnosti koje treba da realizuje vlasnik arhitekture:

1. **Procena stanja ciljne organizacije**, podrazumeva realizaciju sledećih podaktivnosti:

a) Ažuriranje liste ključnih stejkholdera sistema.
b) Aktivne diskusije sa ključnim stejkholderima u cilju razumevanja poslovnog problema i načina poslovanja.
c) Razmatranje poslovne i informacione arhitekture ciljne organizacije.
d) Identifikovanje budućih ciljeva i pravaca razvoja poslovanja organizacije.
e) Identifikovanje karakteristika sistema na osnovu utvrđenih poslovnih potreba.

Cilj eksplicitne arhitekturalne aktivnosti, procena stanja ciljne organizacije, jeste uvid u trenutno stanje organizacije, kako bi vlasnik arhitekture, na samom početku projekta, aktivno učestvovao u diskusijama sa stejkholderima u cilju razumevanja područja problema, načina poslovanja ciljne organizacije i korisničkih potreba, koje su osnova za definisanje karakteristika budućeg sistema.

Podaktivnost *ažuriranje liste ključnih stejkholdera* podrazumeva, identifikovanje stejkholdera koji, sa tačke gledišta vlasnika arhitekture, mogu imati uticaj na uspeh sistema koji se gradi. Stejkholderi mogu biti u okviru i izvan ciljne organizacije (kupci, konkurenti, partneri ili neka druga grupa koja može imati uticaj na uspeh sistema koji se gradi).

Podaktivnost *aktivne diskusije sa ključnim stejkholderima u cilju razumevanja poslovnog problema načina poslovanja*, podrazumeva identifikovanje i analizu sadašnjih i budućih očekivanja stejkholdera, putem workshop-ova, intervjua, upitnika ili nekih drugih tehnika. Direktna i kontinuirana komunikacija sa stejkholderima sistema je ključna za uspeh arhitekture koja se razvija Stoga, vlasnik arhitekture, zajedno sa poslovnim analitičarem treba povremeno da se učestvuje na sastancima sa stejkholderima.

Razmatranje poslovne i informacione arhitekture ciljne organizacije, svodi se na analizu i diskusije sa stejkholderima oko poslovnog modela ciljne organizacije. Cilj aktivnosti je da vlasnik arhitekture stekne uvid o celovitoj slici poslovnog problema i da razume način poslovanja organizacije, jer su to glavni inputi za postavku arhitekture sistema koji se gradi. Ulazni artifakt poslovni model, dovoljno je da bude izrađen neformalnim tipom modela (nacrtan slobodnim stilom na tabli ili papiru). Model treba treba da prikaže tok poslovanja ciljne organizacije, obezbeđujući uvid vlasniku arhitekture u ključne slučajeve upotrebe i tok ključnih poslovnih aktivnosti i informacija u organizaciji.

Podaktivnost *identifikovanje budućih ciljeva i pravaca razvoja poslovanja organizacije,* podrazumeva da vlasnik arhitekture sagleda pravac širenja poslovanja organizacije u cilju sticanja šire slike sistema tj. njegovog budućeg razvoja. Informacije su značajne za adekvatnu postavku arhitekturalnog rešenja na početku projekta.

Podaktivnost *identifikovanje karakteristika sistema na osnovu utvrđenih poslovnih potreba*, podrazumeva tabelarni prikaz identifikovanih poslovnih potreba, na osnovu kojih se utvrđuju karakteristike sistema, neophodne za njihovu realizaciju. Karakteristike sistema trebaju biti na opštem nivou, bez razmatranja kako će biti implementirane, već zašto će biti implementirane. Za sve karakteristike sisteme potrebno je sagladati vrednost (benefit) koju obezbeđuje korisniku i utvrditi njihov prioritet. Definisane karakteristike sistema osnova su za definisanje arhitekturalno značajnih zahteva u narednom koraku.

Realizacijom aktivnosti procena stanja ciljne organizacije, ažurira se izlazni artifakt dokument vizije i to u okviru tačaka 1,2,3,4 i 5 (prilog 16).

2. Razmatranje softverske i tehničke infrastrukture - je aktivnost koja podrazumeva opis postojeće aplikativne i tehničke arhitekture (tekstualno ili neformalnim modelom) ciljne organizacije. Opis aplikativne arhitekture podrazumeva opis softverskih aplikacija koje omogućavaju odvijanje poslovanja u organizaciji, uključujući opis načina korišćenja datih aplikacija i njihovu međusobnu povezanost. Opis tehničke arhitekture podrazumeva opis

postojeće hardverske infrastrukture ciljne organizacije, koja omogućuje izvršavanje softverskih aplikacija.

Dobijeni rezultati značajni su za donošenje odluke o izboru tehnološkog steka novog sistema, kao i prilikom postavke arhitekture, u kontekstu izbora softverskih elemenata koji mogu biti ponovo korišćeni. Rezultatima se ažurira artifakt vizija i to u delu tačka 6.

3. Formiranje inicijalne liste rizika je aktivnost koja podrazumeva identifikovanje rizika sa aspekta arhitekture, u cilju anticipiranja događaja koji se mogu pojaviti i koji će smanjiti verovatnoću da se isporučiti projekat sa pravim karakteristikama, potrebnog nivoa kvaliteta, na vreme i unutar budžeta. Artifakt lista arhitekturalnih rizika se ažurira tokom celog perioda trajanja projekta i sastavni je deo dokumenta vizija (tačka 7).

Pored identifikacije arhitekturalnih rizika, vlasnik arhitekture zadužen je i za njihovu analizu, rangiranje i definisanje inicijalne strategija za njihovu mitigaciju. Identifikovani rizici rezultat su iskustva vlasnika arhitekture, ali i aktivnih diskusija sa stejkholderima.

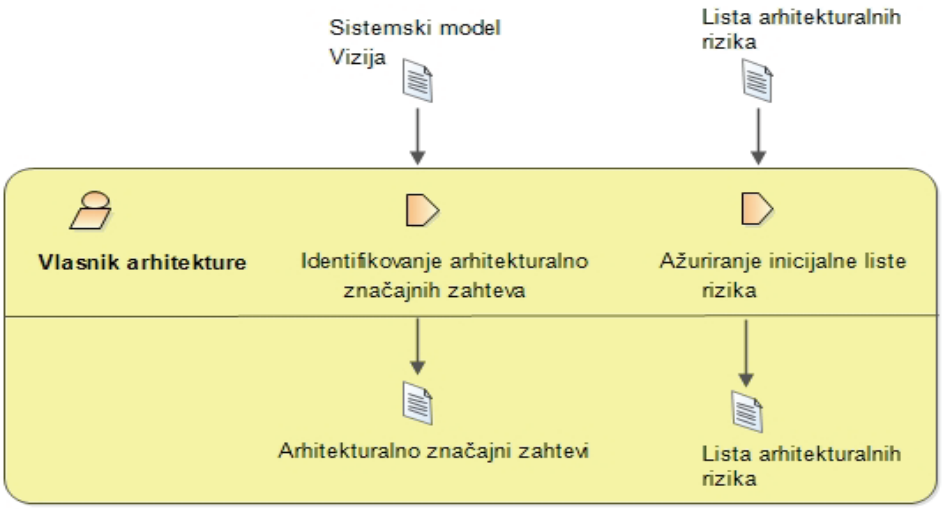
Prilikom analize rizika, vlasnik arhitekture treba da kvantifikuje izloženost arhitekture i projekta u celini, datom riziku. Da bi se utvrdila ekspozicija za svaki rizik ponaosob, potrebno je proceniti sledeće informacije: uticaj rizika i verovatnoću pojave rizike, na osnovu čega se može izračunati izloženost arhitekture datom riziku (množenjem uticaja i verovatnoće pojave rizika). Neophodno je sprovesti i njihovo rangiranje u odnosu na veličinu njihovog uticaja na uspeh arhitekture, kao i projekta u celini (npr. visok, značajan, srednje, minoran). Artifakt vizija treba da sadrži i informacije o strategijama za mitigaciju identifikovanih rizika.

Rizike koje vlasnik arhitekture treba da anticipira u ovom koraku potiču iz različitih izvora, (tehnologije, zahteva, ljudi, eksternih zavisnosti). Primeri rizika karakterističnih za ovu fazu razvoja su: da li je tehnologija u okviru ciljne organizacije dokazana i odgovarajuća i za novi sistem? da li se povećava rizik uvođenjem najnovijih tehnologija? da li je uspeh projekta zavisi od upotrebe najnovijih, nedovoljno ispitanih tehnologija? da li je opravdan cilj ponovne upotrebe softverskih komponenti koje postoje u ciljnoj organizaciji? da li su postojeće softverske komponente u organizaciji razvijane ili su kupovane? da li spoljne zavisnosti od drugih sistema uključuju i one izvan ciljne organizacije? da li već postoje interfejsi za njihovu komunikaciju ili ih je potrebno razviti novim sistemom?.

4. Inicijalno formiranje tima je aktivnost koju, pored menadžera projekta, realizuje i vlasnik arhitekture. Podrazumeva uključivanje neophodnih članova tima u projekat, pošto je na samom početku projekta dovoljno učešće vlasnika proizvoda, poslovnih stejkholdera i vlasnika arhitekture. Donešena odluka dokumentuje se u okviru artifakta vizija, u okviru tačke 8. Premda agilan razvoj naglasak stavlja na samoorganizujućem, krosfunkcionalnom timu, neophodno je dokumentovati za svakog člana tima informacije, kao što su: godine iskustva,

domenska znanja, znanja o agilnom razvoju, ekspertska tehnička znanja i veštine. Ekspertiza i kvalitet tima utiču na izbor i postavku arhitekturalnog rešenja.

KORAK 2: utvrđivanje granica sistema



Slika 5.5 Uloge, aktivnosti i artifakti u realizaciji drugog koraka: utvrđivanje granica sistema
Izvor: Autor

Korak utvrđivanje granica sistema sastoji se od dve ključne aktivnosti, čiji se rezultati dokumentuju u okviru izlaznog artifakta vizija sistema (prilog 16):

1. Identifikovanje arhitekturalno značajnih zahteva sistema - je aktivnost u okviru koje vlasnik arhitekture realizuje sledeći set podaktivnosti:

a) Aktivne diskusije sa ključnim stejkholderima u cilju analize funkcionalnih i nefunkcionalnih zahteva sistema.
b) Identifikovanje zahteva budućih promena sistema (na osnovu identifikovanih budućih ciljeva i pravaca razvoja biznisa).
c) Identifikovanje ograničenja koja će biti nametnuta budućem sistemu.
d) Identifikovanje zahteva koji potiču od geografske dislociranosti delova budućeg sistema.

Ulazni artifakti, na osnovu kojih vlasnik arhitekture realizuju navedene zadatke, su formirani dokument vizije sistema i sistemski model. Sistemski model može biti realizovan neformalnim tipom dijagrama (slobodnim stilom nacrtan na tabli i arhiviran u formi slike) i sadrži samo opštu sliku, putem jednostavnih grafičkih simbola. Model daje prikaz šta sistem treba da radi, a cilj njegove analize u ovom koraku jeste konačan dogovor i njegovo prihvatanje od strane korisnika, vlasnika arhitekture i razvojnog tima.

Drugim rečima, analiza sistemskog modela podrazumeva da se utvrdi konačan dogovor sa kupcima i ostalim stejkholderima o tome šta sistem treba da radi i zašto; da se jasno prikažu razvojnom timu zahtevi sistema; da se definišu granice sistema; da se stvori osnova za utvrđivanje tehničkog sadržaja budućih iteracija; da se stvori osnova za utvrđivanje troškova i vremena potrebnog za razvoj sistema; da se definiše model podataka na visokom nivou apstrakcije; da se definiše korisnički interfejs za sistem sa fokusom na želje i potrebe korisnika.

Osnovni elementi modela trebaju biti akter, slučaj upotrebe i relacije. Pomoću navedenih elemenata moguće je prikazati funkcionalnosti i najkompleksnijih sistema, što je osnovni input za realizaciju podaktivnosti *aktivne diskusije sa ključnim stejkholderima u cilju analize funkcionalnih i nefunkcionalnih zahteva sistema.* Akter predstavlja nekoga ili nešto što se nalazi izvan sistema, a u interakciji je sa njim. Akter može da bude nešto od sledećih pojmova: fizička ličnost ili korisnik sistema, drugi sistem, vreme. Na globalnom nivou dijagram slučajeva upotrebe je moguće predstaviti pomoću identifikovanih aktera i samog sistema, prikazujući njihovu interakciju relacijama. Pri detaljnoj razradi, potrebno je identifikovati funkcionalnosti koje se pojavljuju unutar sistema i utvrditi interakcije između identifikovanih aktera i funkcionalnosti sistema. Slučajevi upotrebe će imati istu namenu, a to je prikaz funkcionalnosti koje služe za zadovoljenje korisničkih zahteva, direktno ili indirektno. Pošto stejkholderi uglavnom nisu svesni nefunkcionalnih zahteva, vlasnik arhitekture ima odgovornost da ih identifikuje, na osnovu funkcionalnih zahteva i da ih potvrdi u razgovoru sa stejkholderima. Njegova uloga ogleda se i u rešavanju kontraktidkornosti u nefunkcionalnim zahtevima, koje su rezultat različitih interesa stejkholdera.

Podaktivnost *identifikovanje zahteva budućih promena sistema* podrazumeva eksplicitno definisanje zahteva budućih promena sistema, na osnovu utvrđenih i opisanih budućih ciljeva i pravaca razvoja biznisa u prethodnom koraku. Ova vrsta zahteva, zajedno sa ključnim funkcionalnim i nefunkcionalnim zahtevima, utiče na izbor i postavku inicijalnog arhitekturalnog rešenja.

Podaktivnost *identifikovanje ograničenja koja će biti nametnuta budućem sistemu*, podrazumeva utvrđivanje i kratak opis ograničenja koja potiču iz iz realnog okruženja i mogu biti različitog porekla: politička, ekonomska, pravna, tehnička, funkcionalna, kao i identifikovanje vrsta zavisnosti u slučaju postojanja nasleđenog sistema ili povezanosti sa nekim eksternim sistemom. Posebnu vrstu ograničenja čine zahtevi koji potiču od geografske dislociranosti delova budućeg sistema i zato se oni identifikuju u okviru podaktivnosti *identifikovanje zahteva koji potiču od geografske dislociranosti delova budućeg sistema.*

Izlazni artifakt, aktivnosti identifikovanje arhitekturalno značajnih zahteva sistema predstavlja lista i opis arhitekturalno značajnih zahteva, kojo se ažurira dokument vizija sistema (tačka 9 dokumenta, prilog 16).

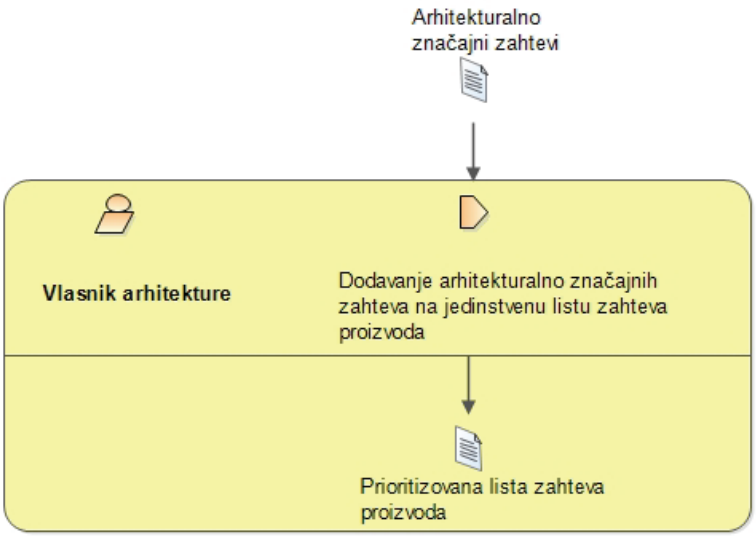
2. Ažuriranje inicijalne liste rizika - je aktivnost koja podrazumeva da vlasnik arhitekture ažurira ulazni artifakt lista arhitekturalnih rizika, sa novoidentifikovanim rizicima. Rizici koji, u ovom koraku arhitekturalnog procesa, mogu biti značajni za vlasnika arhitekture su: da li je zahtev za količinom transakcija koje treba da podrži sistem razuman? da li je količina podataka sa kojima sistem treba da radi razumna? da li postoje neuobičajeni ili tehnički zahtevni zahtevi, koje razvojni tim nije nikada rešavao do tada? da li su identifikovani zahtevi prilično stabilni i jasni? da li postoje neki ekstremno nefleksibilan nefunkcionalan zahtev (npr. da sistem ne sme nikada da propadne), da li su zahtevi kompleksni? da li su eksperti domena na raspolaganju? da li ima dovoljno ljudi, sa odgovarajućim veštinama i iskustvom, na raspolaganju?.

Odgovori na ova pitanja trebaju biti dokumentovana u artifaktu vizija sistema, u okviru tačke 7 (prilog 16).

KORAK 3: formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda

Korak formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda, čini set aktivnosti, prikazanih na slikama 5.6; 5.7 i 5.8. Svaka slika grupiše aktivnosti koje se izvršavaju paralelno. Prvo se realizuju aktivnosti predstavljene na slici 5.6, potom aktivnosti sa slike 5.7 i na kraju set aktivnosti sa slike 5.8. Na ovaj način će iste biti i diskutovane:

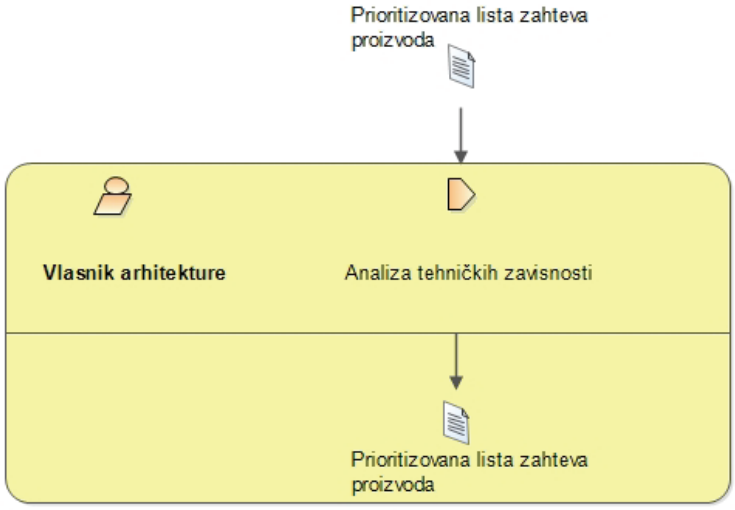
1. Dodavanje arhitekturalno značajnih zahteva na jedinstvenu listu zahteva proizvoda je aktivnost koja podrazumeva formiranje jedinstvene liste zahteva proizvoda: svih funkcionalnih, nefunkcionalnih i zahteva za budućim promenama sistema. Cilj ove aktivnosti je da, od samog početka projekta, arhitekturalni zahtevi budu vidljivi i transparentni i da se njihov značaj i momenat implementacije rangira zajedno sa funkcionalnim zahtevima proizvoda, kao uobičajene aktivnosti u agilnom razvoju. Ulazni artifakt za realizaciju ove aktivnosti je dokument arhitekturalno značajni zahtevi (koji je deo dokumenta vizija sistema), a izlazni artifakt je prioritizovana lista zahteva proizvoda.



Slika 5.6 Uloge, aktivnosti i artifakti u realizaciji trećeg koraka: formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda
Izvor: Autor

2. Analiza tehničkih zavisnosti je aktivnost koju realizuje vlasnik arhitekture, kroz izvršavanje tri ključne podaktivnosti:

a) Analiza međusobne zavisnosti funkcionalnih zahteva.
b) Analiza zavisnosti funkcionalnih od nefunkcionalnih zahteva.
c) Analiza zavisnosti od spoljnih sistema.



Slika 5.7 Uloge, aktivnosti i artifakti u realizaciji trećeg koraka: formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda
Izvor: Autor

Rangiranje i utvrđivanje prioriteta na jedinstvenoj listi zahteva, sprovodi se pre svega sa stanovišta vrednosti koju korisniku obezbeđuje implementacija nekog zahteva sa liste. Međutim, pored vrednosnog aspekta, neophodno je sprovoditi analizu zahteva i sa aspekta rizika i njihove međusobne zavisnosti.

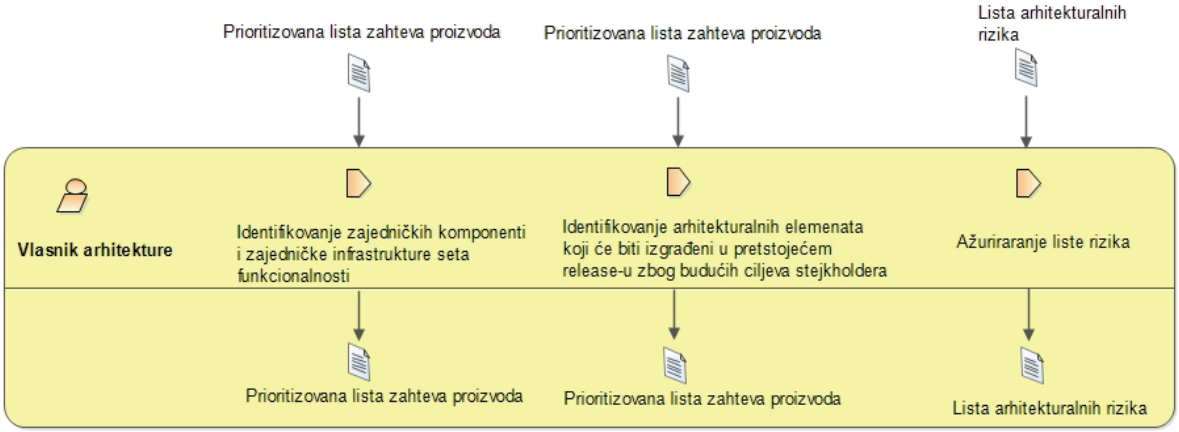
Podaktivnost *analiza zavisnosti funkcionalnih zahteva* može imati za posledicu da je zahteve nižeg ranga (prioriteta sa aspekta vrednosti), potrebno ranije implementirati jer od njih zavise zahtevi, funkcionalnosti koje su (od strane korisnika) ocenjene sa većim rangom odnosno većom vrednošću za biznis. Podaktivnost *analiza zavisnosti funkcionalnih od nefunkcionalnih zahteva* sa liste bitna je, jer je često implementacija nekog arhitekturalnog elementa preduslov za uspešnu implementaciju seta funkcionalnosti. Podaktivnost *analiza zavisnosti od spoljnih sistema* je takođe kriterijum koji je neophodan za utvrđivanje prioriteta u implementaciji zahteva sa jedinstvene liste proizvoda.

I ulazni i izlazni artifakt ovih eksplicitnih arhitekturalnih aktivnosti je dokument prioritizovana lista zahteva proizvoda. Jedinstvena lista zahteva proizvoda, sa utvrđenim prioritetima njihove realizacije treba da se generiše i održava u nekom od alata za upravljanje agilnim razvojem, gde postoje predviđeni šabloni za njenu izradu. To je svakako najagilniji način i za ažuriranje ovog dokumenta, kao i najefikasniji način za kolaboraciju celog tima. Pošto za tu svrhu postoji set agilnih alata i rezultatima istraživanja je potvrđeno da se oni u tu svrhu i koriste u praksi, frejmvorkom nije posebno definisana struktura ovog artifakta.

3. Identifikovanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti je aktivnost koju realizuje vlasnik arhitekture, sa ciljem da nakon utvrđenih zavisnosti definiše zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti.

4. Identifikovanje arhitekturalnih elemenata koji će biti izgrađeni u pretstojećem release-u zbog budućih ciljeva stejkholdera je aktivnost koju realizuje vlasnik arhitekture, sa ciljem da identifikuje arhitekturalne elementi koji nisu neophodni, ali će potencijalno biti implementirani u okviru pretstojećeg release-a zbog budućih ciljeva stejkholdera.

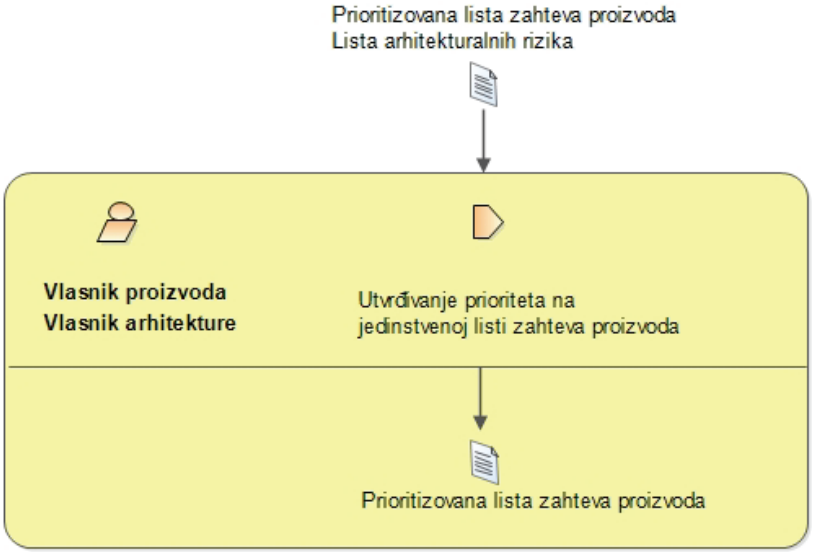
5. Ažuriranje liste rizika je aktivnost koja prati realizaciju prethodno opisanih arhitekturalnih odluka. Podrazumeva analizu rizika i troškova koji će nastati usled odložene isporuke vrednosti korisniku, zbog razvoja arhitekturalnih elemenata.



Slika 5.8 Uloge, aktivnosti i artifakti u realizaciji trećeg koraka: formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda
Izvor: Autor

6. Utvrđivanje prioriteta na jedinstvenoj listi zahteva proizvoda

U realizaciji aktivnosti, utvrđivanje prioriteta na jedinstvenoj listi zahteva proizvoda, učestvuju i vlasnik proizvoda i vlasnik arhitekture. Vlasnik proizvoda je predstavnik korisnika i prioritete utvrđuje sa aspekta vrednosti koja se isporučuje korisniku. Vlasnik arhitekture predlaže tehničke sadržaje i redosled njihove implementacije kroz iteracije, na osnovu prethodno sprovedenih analiza tehničkih zavisnosti i procene rizika. Tehnički predlog vlasnika arhitekture zavisi i od dostupnosti članova razvojnog tima, dostupnosti alata i sl..



Slika 5.9 Uloge, aktivnosti i artifakti u realizaciji trećeg koraka: formiranje prioritizovane liste funkcionalnih i nefunkcionalnih zahteva proizvoda
Izvor: Autor

Rangiranje arhitekturalnih zahteva vlasnik arhitekture zasniva na sledećim ključnim faktorima:

- Korisnošću koju stejkholderi imaju od implementacije nekog arhitekturalnog zahteva: kritičan/važan/koristan.
- Uticajem zahteva na arhitekturu: nema/proteže se/menja. Mogu postojati funkcionalni zahtevi koji su ocenjeni kao kritični (visokoprioritetni sa aspekta vrednosti koju obezbeđuju korisniku), a da pri tom nemaju uticaj na arhitekturu, kao i obrnuto.
- Rizicima koje treba ublažiti (performansa, dostupnost proizvoda, pogodnost komponenti).
- Drugim taktičkim ciljevima ili ograničenjima.

Prioritet za implementaciju imaju arhitekturalni zahtevi koji su visoko rizični, kako bi se obezbedilo dovoljno vremena za njihovu stabilizaciju, tokom trajanja projekta. Arhitekturalni zahtevi koji nisu dovoljno jasni, dodatno se analiziraju primenom nekog tipa prototipa ili arhitekturalnog spike-a.

Konačan rezultat koraka je jedinstvena lista zahteva proizvoda (za koju je odgovoran vlasnik proizvoda), sa definisanim prioretima njihove implementacije. Lista se ažurira tokom celog

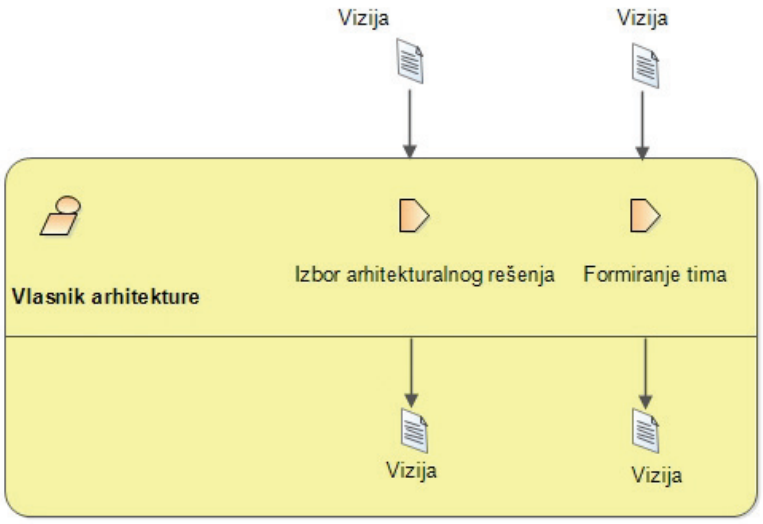
perioda trajanja projekta, nakon svake iteracije. Jedinstvena lista zahteva proizvoda, sa utvrđenim prioritetima njihove realizacije održava se u nekom od alata za upravljanje agilnim razvojem, gde postoje predviđeni šabloni za njenu izradu. Iz tog razloga sadržaj i strukturu ovog dokumenta nije bilo potrebno definisati ovim frejmvorkom.

KORAK 4: postavka inicijalne arhitekture

Korak podrazumeva realizaciju eksplicitnih arhitekturalnih aktivnosti koje se odnose na izbor arhitekturalnog rešenja (slika 5.10) i seta eksplicitnih arhitekturalnih aktivnosti u cilju postavke inicijalne arhitekture (slika 5.11):

1. Izbor arhitekturalnog rešenja je eksplicitna arhitekturalna aktivnost za koju je odgovoran vlasnik arhitekture. Realizuje se izvršavanjem seta podaktivnosti:

a) Istraživanje tehnologije i trenda opcija; istraživanje industrijskih okvira za implementaciju i biblioteka klasa.
b) Analiza postojećih industrijskih arhitekturalnih rešenja.
c) Sagledavanje ekspertize i znanja potencijalnih članova razvojnog tima.
d) Identifikovanje komponenti koje mogu biti ponovo korišćene.
e) Analiza i procena potencijalnih arhitekturalnih rešenja.
f) Potvrda odabranog arhitekturalnog rešenja i dokumentovanje odluke.



Slika 5.10 Uloge, aktivnosti i artifakti u realizaciji četvrtog koraka: postavka inicijalne arhitekture
Izvor: Autor

U izboru arhitekturalnog rešenja, vlasnik arhitekte se oslanja na sopstveno iskustvo i uglavnom na poznata, postojeća rešenja, koja je potrebno da na početku projekta analizira, sa ciljem da se odabere najadekvatnije u odnosu na problem koji se rešava. Pored iskustva, za

odabir adekvatnog arhitekturalnog rešenja, arhitekta mora analizirati trend i opcije u skladu sa problemom koji se rešava, profilom klijenta, mogućnostima razvojnog tima, arhitekturalno značajnim zahtevima sistema i budućnošću razvoja sistema. Vlasnik arhitekture treba da identifikuje i delove referentne arhitekture, koji mogu biti ponovo korišćeni: softverske komponente koje postoje u ciljnoj organizaciji, postojeći industrijski modeli, okviri, komponente. Izučava iskustva drugih kompanija sa određenim tehnologijama i arhitekturalnim rešenjima. Konačno, vlasnik arhitekture oslanjanjem na prikupljene informacije i na sopstveno iskustvo, donosi odluku o arhitekturalnom rešenju. Odabrano arhitekturalno rešenja je potrebno potvrditi nekom arhitekturalnih tehnika. Odluka odabranom arhitekturalnom rešenju se argumentovano dokumentuje u artifaktu dokument vizije (prilog 16).

2. Formiranje tima - je aktivnost koja se izvršava paralelno sa izborom arhitekturalnog rešenja, jer je to momenat kada je potrebno sagledati ekspertizu i znanja potencijalnih članova razvojnog tima, kako bi se odabralo rešenje koje će razvojni tim moći da implementira. Tim se formira u skladu sa potrebnim domenskim i tehničko-tehnološkim znanjem pojedinaca.

Sledeći set aktivnosti odnosi se na postavku inicijalne arhitekture, u skladu sa prethodno odabranim arhitekturalnim rešenjem. Postavka inicijalne arhitekture na početku projekta, podrazumeva razvoj opšteg pregleda arhitekture, na osnovu identifikovanih i rangiranih arhitekturalno značnih zahteva i odluka dokumentovanih artifaktom vizija sistema. Opšti pregled arhitekture razvija vlasnik arhitekture, u saradnji sa razvojnim timom i stejkholderima, u obliku neformalnih dijagrama.

3. Definisanje modula (za glavni deo sistema) i njihovih veza.

Opšti pregled arhitekture ilustruje suštinu predloženog arhitekturalnog rešenja i glavne gradivne blokove (elemente, module), od kojih će biti izgrađena arhitektura sistema. Predloženo arhitekturalno rešenje, ukoliko je moguće, zasniva se na referentnoj arhitekturi, arhitekturalnim obrascima i drugim već postojećim arhitekturalnim rešenjima (koji su analizirani i procenjeni u prethodnom koraku).

4. Definisanje podsistema i lejera za definisane module.

Vlasnik arhitekture, na početku projekta, definiše organizaciju podsistema za svaki identifikovani modul, što predstavlja nivo dizajna sistema na najvišem nivou apstrakcije.

5. Definisanje ključnih elemenata, interakcija i mehanizama dizajna.

Postavka arhitekture na početku projekta uključuje i prikaz identifikovanih ključnih apstrakcija dizajna – inicijalne elemente dizajna, koji će se tokom implementacije sistema modifikovati i obogaćivati detaljima, kao i interakcije između ključnih apstrakcija, čime je

uspostvaljen način komunikacije elemenata dizajna. Na početku projekta, prilikom postavke arhitekture, bitno je identifikovati i potrebu za nekim od mehanizama dizajna, dok se odluka o načinu njegove implementacije odlaže za kasnije, kada arhitekta bude raspolagao sa dovoljno informacija. Tako naprimer, mehanizam perzistentnosti značajan je na projektima koji koriste velike količine podataka, jer odgovara na zahteve kao što su brzina pristupa podacima, brzina zapisa podataka, obrasci za čitanje i pisanje i dr.

6. Razmatranje i inicijalna postavka arhitekture podataka.

Podrazumeva donošenje odluke o tipu baze podataka koja će se razvijati, razmatranje kompatibilnosti podataka i inicijalnu postavku modela.

7. Identifikovanje tački arhitekture koje trebaju biti fleksibilnije.

Na početku projekta potrebno je identifikovati i tačke arhitekture koje trebaju biti fleksibilnije u odnosu na identifikovane zahteve o budućim promenama sistema. Time se omogućuje laka modifikacija inicijalne arhitekture, što je važna osobina agilne arhitekture.

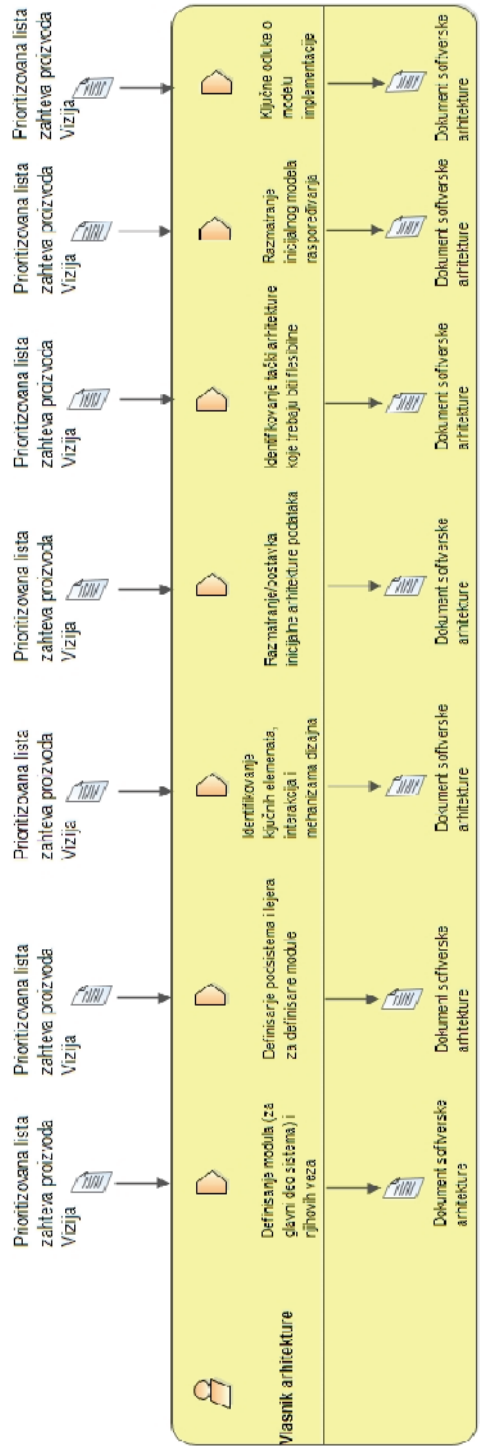
8. Razmatranje inicijalnog modela raspoređivanja.

Razmatranje modela rasporedjivanja važna je arhitekturalna aktivnost iz razloga što pruža informacije o produkcionom okruženju sistema, a ove informacije umnogome utiču na arhitekturalno rešenje. Razvoj modela raspoređivanja podrazumeva njegovu inicijalnu postavku, bez detalja. Značajan je iz više razloga: omogućava sagledavanje održivosti implementacije budućeg sistema; olakšava razumevanje geografske distribuiranosti i kompleksnosti sistema; predstavlja osnovu za ranu procenu troškova i napora. Up front vreme i napori potrebni za inicijalnu postavku modela raspoređivanja softvera, u direktnoj su zavisnosti od tehnologije koja se koristi. Izbor tehnologije određuje da li je potrebno napraviti sopstveno rešenje ili je mogući koristiti već postojeća. Preporučljivo je prilikom razmatranja modela raspoređivanja, već na početku projekta uključiti i ljude koji obavaljaju operativne aktivnosti u ciljnoj organizaciji, zbog informacija koje mogu pružati, a koje imaju uticaja na postavku arhitekture.

9. Ključne odluke o modelu implementacije.

Odluke o modelu implementacije imaju za cilj da eliminišu probleme vezane za upravljanje konfiguracijom i da omoguće nesmetanu implementaciju, integraciju i proces izgradnje sistema. Njegovo razmatranje na početku projekta, važno je zbog donošenje odluke o korišćenju otočnih servisa i za postavku inicijalnog modela podataka.

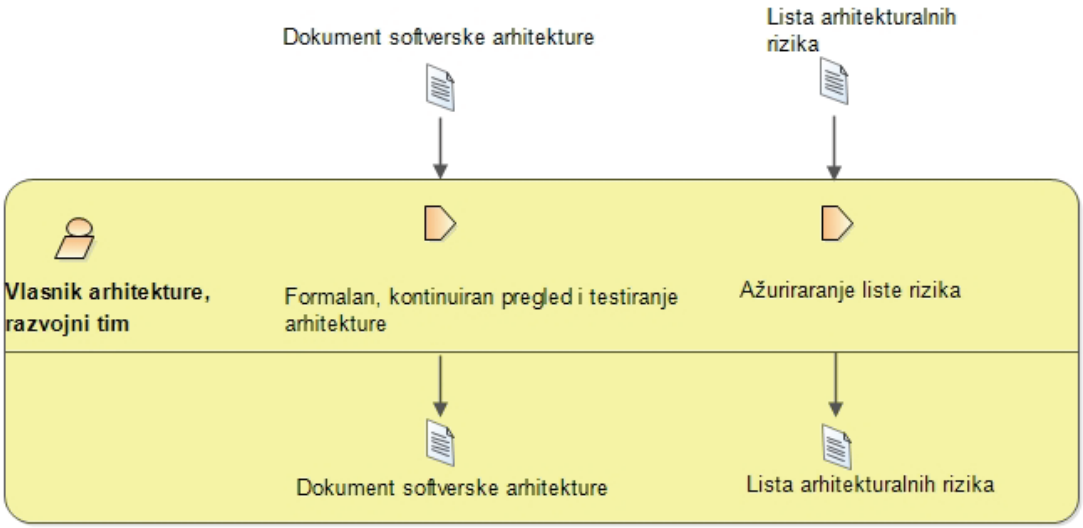
Ulazni artifakti neophodni za realizaciju koraka postavka inicijalne arhitekture su: prioretizovana lista zahteva proizvoda i dokument vizije sistema. Izlazni artifakt je dokument softverske arhitekture (prilog 16, sekcija B).



Slika5.11 Uloge, aktivnosti i artifakti u realizaciji četvrtog koraka: postavka inicijalne arhitekture
Izvor: Autor

KORAK 5: potvrda arhitekture

U realizaciji ovog arhitekturalnog koraka učešće uzima ceo tim, uključujući i stejkholdere, stim da je najveća odgovornost na vlasniku arhitekture i razvojnom timu. Ulazni artifakt za realizaciju ovog koraka jeste dokument softverske arhitekture, jer on sadrži sve modele i opise koji se odnose na postavljeno arhitekturalno rešenje.



Slika 5.12 Uloge, aktivnosti i artifakti u realizaciji petog koraka: potvrda arhitekture
Izvor: Autor

Korak potvrda arhitekture uključuje sledeće arhitekturalne aktivnosti:

1. Formalan, kontinuiran pregled i testiranje arhitekture

a) Izbor tehnika za potvrdu arhitekture.
b) Definisanje kriterijumima validacije i ciljeva koji se žele postići.
c) Generisanje rezultata validacije.

Aktivnost formalan, kontinuiran pregled i testiranje arhitekture obuhvata podaktivnost *izbor tehnika za potvrdu arhitekture*, jer potvrda arhitekture, pored izvršnog koda može uključivati i lepezu drugih tehnika, kao što su: proof of koncept, prototip, arhitekturalni spike, mišljenje odbora za arhitekturu i dr, u zavisnosti od realnih potreba projekta. Bitno je istaći da svaka od tehnika za proveru arhitekture mora da bude u skaladu sa agilnim razvojem, što dalje znači da je vremenski ograničena. Druga podaktivnost je eksplicitno *definisanje kriterijumima validacije i ciljeva koji se žele postići,* dok podaktivnost *generisanje rezultata validacije* podrazumeva tumačenje dobijenih rezultata validacije i obrazlaganje arhitekturalnih odluka koje se donose u skladu sa njima.

Na samom početku projekta, nakon postavke inicijalne arhitekture, neophodno je tzv. nultom iteracijom implementirati značajnu centralnu funkcionalnost i proveriti delikatne tačke postavljene arhitekture. Diskusije vlasnika proizvoda, razvojnog tima i ključnih stejkholdera sa vlasnikom arhitekture, takođe trebaju biti agilna praksa koja ima za cilj da elaborira izvodljivost, probleme i rizike u razvoju arhitekture i to kako na početku, tako i na kraju svake iteracije.

2. Ažuriranje liste rizika

Potvrda arhitekture je korak koji karakteriše i samo sprovođenje iteracija, jer se implementacijom arhitekturalnih elemenata, putem izvršnog koda, zapravo proverava postavljena arhitektura. Stoga je neophodno arhitekturalnu aktivnost *ažuriranje inicijalne liste rizika*, realizovati tokom trajanja iteracije, kako bi se završetkom iste, cilj nove iteracije definisao u skladu sa listom rizikom. Lista arhitekturalnih rizika je ulazni i izlazni artifakt aktivnosti ažuriranje liste rizika. Ažurirana lista dokumentuje se u okviru artifakta vizija sistema u okviru tačke 7 (prilog 16).

KORAK 6: sagledavanje arhitekture

Korak sagledavanje arhitekture podrazumeva realizaciju sledećih aktivnosti, prikazanih na slici 5.13:

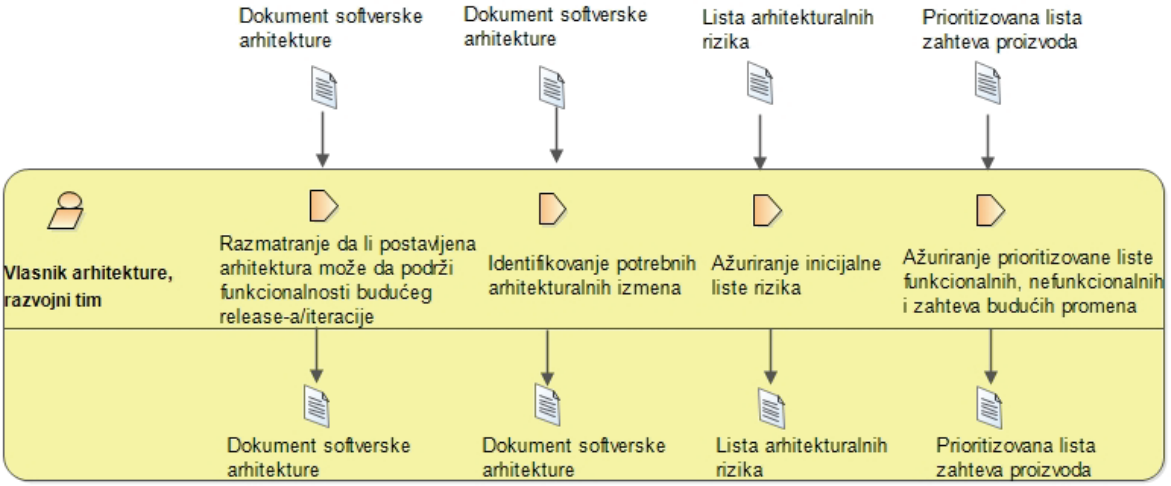
1. Razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a/iteracije.
2. Identifikovanje potrebnih arhitekturalnih izmena.
3. Ažuriranje inicijalne liste rizika.
4. Ažuriranje prioritizovane liste funkcionalnih, nefunkcionalnih i zahteva budućih promena.

Uloge koje realizuju aktivnosti ovog koraka su vlasnik arhitekture i razvojni tim. Sagledavanje arhitekture je kritična tačka agilnih procesa i to, kako na nivou release-a, tako i na nivou iteracija. Stoga je navedene aktivnosti potrebno sprovoditi prilikom događaja planiranja ali i događaja ocene iteracije/relelease-a.

Fokus koraka usmeren je na razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a; na identifikciju potrebnih arhitekturalnih izmena pre implementacije narednog release-a/iteracije; na utvrđivanje zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti; na razmatranje nasleđenog sistema i zavisnosti od partnerskih ili otočnih proizvoda i kompatibilnost podataka; na ažuriranje liste rizika i identifikovanje arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera.

Glavni ulazni i izlazni artifakt za aktivnosti: razmatranje da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a/iteracije i identifikovanje potrebnih arhitekturalnih izmena jeste dokument softverske arhitekture. Aktivnost ažuriranje inicajlne liste rizika, odvija se paralelno sa njima i kao rezultat daje novoidentifikovane arhitekturalne rizike. Aktivnost identifikovanje potrebnih arhitekturalnih izmena, podrazumeva i zadatak utvđivanja arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranja budućih ciljeva stejkholdera, kao i zadatak identifikovanja zajedničkih komponenti i zajedničke infrastrukture seta funkcionalnosti.

Korak sagledavanje arhitekture završava se aktivnošću ažuriranje prioritizovane liste funkcionalni, nefunkcionalnih i zahteva budućih promena, kako bi utvrđeni problemi, rizici i arhitekturalne izmene mogle biti transparentne na listi zahteva proizvoda, gde se konačno utvrđuje i njihov prioritet u implementaciji.



Slika 5.13 Uloge, aktivnosti i artifakti u realizaciji šestog koraka: sagledavanje arhitekture
Izvor: Autor

5.2.1 Evaluacija postavljenog metodološko-radnog okvira

Postavljeni metodološko-radni okvir podvrgnut je oceni, od strane osam eksperata (4 eksperta iz akademije i 4 iz prakse). Dopis koji je za tu svrhu poslat ekspertima, dat je u prilogu rada (prilog 17). Odabir eksperata vršio se prema definisanim kriterijumima, koji su rezultat modifikacije opštih kriterijuma postavljenih od strane grupe autora (Skulmoski, Hartman, Krahn, 2007; Ziglio, 1996):

- Znanje i praktična iskustva u domenu problema istraživanja: razvoj softverske arhitekture kompleksnih softverskih rešenja agilnim procesima razvoja.
- Kapacitet i spremnost učesnika da doprinese istraživanju.
- Potvrda da će imati vremena i biti dovoljno posvećen istraživanju.

- Najmanje akademski nivo obrazovanja.
- Višegodišnje profesionalno iskustvo (više od 10 godina).

Eksperti su evaluaciju postavljenog okvira sprovodili ocenjivanjem seta tvrdnji, ocenama od 1 do 6 (1 – u potpunosti se ne slažem sa tvrdnjom; 6 – u potpunosti se slažem sa tvrdnjom):

Tvrdnja 1: Postavljeni okvir u dovoljnoj meri proširuje agilne procese arhitekturalnim praksama, u cilju njihovog osposobljavanja za razvoj kompleksnih poslovnih sistema.

Tvrdnja 2: Postavljeni okvir je jasno definisan.

Tvrdnja 3: Postavljeni okvir je koristan na projektima razvoja kompleksnih sistema agilnim procesima.

Tvrdnja 4: Postavljeni okvir je primenljiv u organizacijama.

Tvrdnja 5: Upotreba postavljenog okvira doprinosi većem kvalitetu kompleksnih poslovnih softverskih rešenja u agilnim procesima.

Tvrdnja 6: Upotreba postavljenog okvira smanjila bi rizik erozije arhitekture.

Tvrdnja 7: Postavljeni okvir uravnotežava razvoj funkcionalnosti i arhitekture, neugrožavajući agilnost projekta.

Definisani kriterijumi evaluacije proizašli su iz teoretskih i empirijskih aspekata koje postavljeni metodološko-radni okvir treba da zadovolji. Prosečna ocena postavljenog metodološko-radnog okvira iznosila je 5.4.

Dobijeni rezultati evaluacije metodološko-radnog okvira potvrđuju i hipotezu istraživanja, navedenu u uvodnom delu rada.

5.2.2 Agilna arhitektura poslovnih softverskih rešenja i ekonomske implikacije

Agilni procesi razvoja imaju u fokusu da što ranije isporuče vrednost korisniku i stoga se rukovode principom "razvoj vođen vrednošću". U skladu sa tim, agilni procesi prenaglašavaju značaj ranog razvoja funkcionalnosti, dok arhitekturu posmatraju kao rezultat samog razvojnog procesa. Ovo su razlozi zbog kojih, na agilnim projektima, arhitektura često nije ekonomski opravdana, odnosno ne obezbeđuje ROI, već je često izvor rasta ukupnih projektnih troškova. Isključivanje formalnog arhitekturalnog procesa, zbog što veće agilnosti tima i što brže isporuke vrednosti korisniku, implicira veliki obim naknadnih prepravki arhitekture, kada ista ne može da podrži nove/promenjene zahteve. Troškovi koji se generišu

usled redizajna/novog dizajna arhitekturalnog rešenja, u kasnijim razvojnim fazama, su ogromni. Zbog toga razvoj kompleksnih poslovnih softverskih rešenja ne sme biti zasnovan na potpuno nascentnoj arhitekturi. Takav razvoj je visoko rizičan, jer može da dovede do tačke nakon koje postojeću arhitekturu nije više moguće popravljati tehnikom refaktorisanja. Takva situacija zahteva potpuno novi dizajn arhitekture, što implicira rapidan rast troškova i nezadovoljstva klijenata.

Ekonomski je opravdano da arhitekturalni proces bude iterativno-inkrementalan i da se zasniva na dovoljnom obimu up front odluka donešenih JIT. Drugim rečima, up front odluke, na svim nivoima, moraju biti opravdane nakon sagledavanja aspekta vrednosti, rizika i troškova. Troškovi up front arhitekturalnih odluka, koji proizilaze usled odloženog razvoja funkcionalnosti i kasnije isporuke vrednosti klijentu, moraju biti manji od troškova koji se indukuju usled kasnijih prepravki arhitekture. Tipični agilni timovi rukovode se isključivo maksimalnom isporukom rane vrednosti, jer je ona vrlo često klijentima bitnija od minimiziranja troškova. Iz tog razloga, uz opisanu analizu troškova potrebno je sagledati i trošak kašnjenja klijentovih prihoda.

Predloženi metodološko-radni okvir zasniva se na ekonomski opravdanom razvoju arhitekture u agilnim procesima. To dalje znači da uz agilan princip "razvoj vođen vrednošću" podjednako agilni timovi treba da uvažavaju i princip "razvoj vođen rizicima i troškovima arhitekture". Identifikacija i izbegavanje rizika je tradicionalna arhitekturalna aktivnost, koja se ogleda u ranom dokazivanju arhitekture putem izvršnog koda i neophodna je agilnim timovima prilikom razvoja kompleksnih poslovnih softverskih rešenja. Treći princip koji agilni timovi moraju uvažavati, prilikom razvoja arhitekture, jeste da je konačna odluka uvek klijentova, jer u krajnjoj instanci u pitanju je njegov novac. Ovi principi treba da budu ugrađeni u arhitekturalnu strategiju od samog starta projekta, pri izboru arhitekturalnog rešenja, preko prioritizacije zahteva na listi zadataka, pa do samog kraja projekta.
Može se zaključiti, da se predloženi metodološko-radni okvir za razvoj arhitekture kompleksnih poslovnih softverskih rešenja, zasniva na razmatranju ekonomskih aspekata, uz uvažavanje konačne odluke klijenta, koji je naručioc projekta i koji u krajnjoj liniji raspolaže budžetom.

Predloženi okvir agilne timove usmerava na kontinuirano sagledavanje arhitekture u cilju ispunjavanja novih funkcionalnih zahteva kao i na njenu kontinuiranu evaluaciju.

Uvažavajući kontekstualne faktore projekta, agilni timovi treba da donesu takvu arhitekturalnu strategiju kojom će se minimizirati arhitekturalni napori, a time i ukupni troškovi. Drugim rečima, agilni timovi treba da razmotre vreme neophodno za donošenje up front odluka, na svim nivoima, kao i vreme koje će biti potrebno za sprovođenje refaktorisanja, kako bi rešili određena arhitekturalna pitanja i probleme. Samo na takav način arhitektura može biti kontinuirano sagledavana, dorađivana i popravljana, spram novih/promenjenih zahteva, što mora biti cilj agilne arhitekture. Na ovakav način se

kontolišu troškovi koji potiču od kašnjenja isporučene vrednosti korisniku, ali i troškovi koji se generišu usled redizajna arhitekture. Takođe, sprečava se erozija arhitekture, koju nije moguće popraviti refaktoringom koda, već je neophodno pristupiti dizajnu potpuno novog arhitekturalnog rešenja. Samo up front odluke koje se donose onda kada je to neophodno (JIT), mogu obezbediti efikasnu nascentnu arhitekturu, koja se može dorađivati refaktoringom.

5.3 Ograničenja istraživanja

Poglavljem će biti prikazana ograničenja sprovedenog empirijskog istraživanja, koja potiču od Delfi metode, kojom je isto sprovedeno, kao i načini njihovog umanjenja ili potpunog eliminisanja. Takođe, u fokusu narednog teksta su i kriterijumi za evaluaciju kvaliteta sprovedenog istraživanja, Delfi metodom.

Brojni autori smatraju da Delfi metoda ne proizvodi probleme "grupnog mišljenja" i "dominantne ličnosti", koji za rezultat imaju konformitet u mišljenju respodenata panela (Moeller & Shafer, 1994; Veal, 1992; Fisher, 1978) iz razloga što respodenti ne donose odluke licem u lice. Druga grupa autora, međutim, navodi niz socijalno-psiholoških faktora koji mogu da utiču na konformitet u mišljenju respodenata panela (Bardecki, 1984; Sackman, 1975). Iz tog razloga, u radu su sprovedene dodatne kvantitativne analize, kako bi se utvrdile statistički značajne promene mišljenja respodenata iz drugog u treći krug istraživanja. Statistički značajna promena mišljenja utvrđena je kod 7 učesnika panela, međutim, bitno je naglasiti da su ispitanici već u drugom krugu istraživanja postigli nivo saglasnosti od 70%, za 87 varijabli od ukupnih 115. Tako da su u trećem krugu istraživanja, ispitanici ocenjivali značajnost svega 28 variabli, usled čega su imali uvid i u grupni odgovor za svaku varijablu ponaosob.

Može se zaključiti da je sindrom mogućeg konformiteta u mišljenju eksperata beznačajan sa stanovišta krajnjih rezultata istraživanja, jer je za preko 75% ocenjivanih varijabli postignut nivo saglasnosti od 70%, već u drugom krugu istraživanja, kada ispitanici nisu imali uvid u grupno mišljenje.

Delfi metoda je dugotrajna, administrativno zahtevna, skupa i iziskuje dosta vremena, rada i truda, kako od istraživača tako i od ispitanika (Yousuf, 2007; Fitzsimmons & Fitzsimmons, 2001; Zinn, Zalokowski, & Hunter, 2001). Jeffery, Hache i Lehr (1995) smatraju da dodatne otežavajuće faktore predstavljaju: broj krugova istraživanja (koji može biti i 25), dužina vremena koje protekne između krugova istraživanja i dužina trajanja svakog kruga istraživanja. Uvažavajući ove stavove, istraživanje je sprovedeno sa optimalna 3 kruga, uz zahtevani nivo saglasnosti ispitanika od 70%.

Prema Sandrey i Bulger (2008) vreme koje protekne između dva kruga istraživanja može predstavljati ozbiljno ograničenje istraživanja, ako panel čine mladi i neprofesionalni respodenti. Razlog za to je pad motivacije učesnika i njihovo osipanje tokom trajanja istraživačkog procesa. Uzimajući u obzir ovu vrstu ograničenja Delfi metode, panel su činili iskusni (sa prosečnim iskustvom od 13 godina) i zainteresovani eksperti, iz domena istraživačkog problema. To je impliciralo da tokom procesa istraživanja nije došlo do osipanja respodenata, usled čega je u sva tri kruga istraživanja, stopa odgovora iznosila 100%.

Uspostavljanje metodološke strogosti je od vitalnog značaja za svako istraživanje i tesno je povezano sa pojmovima pouzdanosti i validnosti. Pojmovi pouzdanosti i validnosti vezuju se prevashodno za evaluaciju kvantitativnih istraživanja, ali se njihova modifikovana primena može naći i u domenu kvalitativnih istraživanja. Delfi metod, kojim je sprovedeno empirijsko istraživanje u doktorskoj disertaciji, sadrži i kvantitativnu i kvalitativnu komponentu, zbog čega su validnost i pouzdanost razmatrani za sa oba aspekta.

Validnost same Delfi metode zasnovana je na pretpostavci da veći broj ljudi ima manje šanse da donese pogrešne odluke nego pojedinci. Veća validnost nalaza sprovedenog istraživanja, postignuta je insistiranjem da sve odluke respodenata budu argumentovane i obrazlagane. Takođe, validnost istraživanja povećana je i izborom respodenata koji imaju ekspertska znanja i zainteresovanost za problem istraživanja, u skladu sa preporukama (Goodman, 1987), kao i sprovođenjem uzastopnih krugova istraživanja, u kratkim vremenskim razmacima i stopom odgovora od 100% (u sva tri kruga istraživanja).

Validnost se u kvantitativnim istraživanjima odnosi na pitanje da li istraživač zaista meri ono što je nameravao, ili koliko su istiniti rezultati istraživanja. Validnost je, dakle, u tesnoj vezi sa instrumentom istraživanja tj. sa pitanjem da li instrument istraživanja (način merenja) omogućava ostvarivanje istraživačkih ciljeva (meri ono za šta je namenjen). Tehnika kojom je obezbeđena veća validnost razvijenih instrumenata istraživanja u doktoratu, je indeks sadžajne valjanosti upitnika, koji je izračunavan prema propisanoj proceduri od strane autora Polit i Beck (2006).

Pouzdanost se odnosi na pitanje stabilnosti istraživačkih uslova i procedura tj. u kojoj meri istraživački postupak daje slične rezultate, pod nepromenjenim istraživačkim uslovima. Delfi metod pouzdanost povećava na dva načina. Prvi način odnosi se na proces donošenja odluka, koji podrazumeva da učesnici nisu u istoj prostoriji (licem u lice), čime se izbegava grupna pristrasnost i mogućnost konformiteta u mišljenju. Drugi način odnosi se na veličinu panela, pri čemu rastom veličine panela raste i pouzdanost respodenata grupe. Veća pouzdanost rezultata istraživanja postignuta je bootstrap uzorkovanjem na 1000 replikacija.

Kirk i Miller (1986) su identifikovali tri kriterijuma pouzdanosti u kvantitativnim istraživanjima, koja se odnose na: stepen u kojem ponovljeno merenje daje iste rezultate, stabilnost merenja tokom vremena, sličnosti merenja u datom vremenskom periodu.

Charles (1995) pojam pouzdanosti vezuje direktno za pojam stabilnosti, u kontekstu nepromenjenih odgovora ispitanika, u različitim periodima vremena ili iteracijama istraživanja. Iz tog razloga je u doktoratu računat indeks individualne stabilnosti, koji je pokazatelj ustaljenosti odgovora ispitanika od drugog do trećeg kruga istraživanja. Dati pokazatelj se, pored kriterijuma postignutog nivoa saglasnosti eksperata od 70%, koristio i kao dodatni kriterijum za donošenje odluke o obustavljanju iteracija istraživanja.

Pouzdanost se može meriti i poređenjem rezultata dve različite grupe respodenata panela. Međutim, ovakva merenja pouzdanosti nisu sprovedena, zbog vremenskog ograničenja izrade doktorske disertacije. Drugim rečima, ovakva merenja podrazumevaju sprovođenje potpuno novog empirijskog istraživanja, za šta je potrebno dodatnih 12 meseci. Iz tog razloga, kontrolno istraživanje predstavlja jedan od pravaca budućih istraživanja.

Pouzdanost je koncept kojim se, u kvalitativnim istraživanjima, procenjuje "svrha objašnjavanja". Međutim, dobro kvalitativno istraživanje je ono koje omogućava razumevanje sitaucije ili fenomena koji nisu bili dovoljno istraženi i jasni. Razlika u svrsi evaluacije kvaliteta kvantitativnih i kvalitativnih istraživanja, je razlog zbog kojeg je koncept pouzdanosti, u svom osnovnom obliku, irelevantan u kvalitativnim istraživanjima. Brojni autori smatraju da je obezbeđivanje kriterijuma validnosti, dovoljno za uspostavljanje pouzdanosti u kvalitativnim istraživanjima.

Lincoln i Guba (1985) smatraju da su validnost i pouzdanost esencijalni kriterijumi kvaliteta u kvantitativnim istraživanjima, dok su prenosivost, konzistentnost, neutralnost i kredibilitet, esencijalni ktiterijumi za kvalitet kvalitativnih istraživanja. Navedeni kriterijumi predstavljaju zamenske, ili drugim rečima, modifikovane kriterijume pouzdanosti i validnosti, shodno prirodi kvalitativnih istraživanja.

U cilju postizanja većeg kredibiliteta nalaza i zaključaka kvalitativnog istraživanja, u doktoratu su korišćeni kriterijumi koje su predložili Lincoln i Guba (1985). Njihova primena podrazumevala je četiri glavna pitanja:

- da li je ostvariva prenosivost nalaza tj. njihova primenljivost i u drugim kontekstima?,
- da li postoji konzistentnost nalaza?,
- u kom stepenu su nalazi neutralni tj. da li nalazi odražavaju mišljenje i stavove ispitanika ili su rezultat pristrasnosti i pretenzija istraživača?,
- da li su dobijeni nalazi istiniti i obezbeđuju li kredibilitet izvedenih zaključaka?.

Za obezbeđivanje odgovora na navedena pitanja korišćen je sledeći set tehnika (Lincoln & Guba, 1985):

1. Tehnika korišćena za obezbeđivanje kriterijuma primenljivosti nalaza, podrazumevala je opisivanje istraživanog fenomena dovoljno detaljno, kako bi spoljni evaluator mogao da

proceni u kojoj meri su zaključci primenljivi na druge situacije, vreme, postavke i ljude. Na ovaj način obezbeđene su osnove za spoljnu validnost nalaza istraživanja.

2. Tehnika korišćena za obezbeđivanje kriterijuma konzistentnosti nalaza, podrazumevala je spoljnu reviziju od strane 2 istraživača, doktora nauka na Ekonomskom fakultetu u Subotici. Cilj je bio da se proceni tačnost i da li su nalazi, njihove interpretacije i zaključci izvedeni na osnovu podataka. Na ovaj način postignuta je veća tačnost i validnost sprovedenog istraživanja i dobijenih nalaza.

3. Tehnika korišćena za obezbeđivanje kriterijuma neutralnosti nalaza, podrazumevala je detaljan opis svih istraživačkih koraka. Opisi sadrže sve informacije o samom toku istrage (razvoj instrumenata istraživanja, beleške sa terena, sirove podatke i dr.) i o tome šta je urađeno tokom njega (analiza kvalitativnih podataka, sinteza, identifikovane teme, koncepti, kategorije i dr.).

4. Tehnika korišćena za obezbeđivanje kriterijuma kredibiliteta nalaza, podrazumevala je da se deo empirijskih podataka arhivira i da se u prvom krugu ne analizira. Analiza se prvobitno sprovodila nad preostalim podacima, na osnovu čega su generisani preliminarni rezultati. Generisanje preliminarnih rezultata uslovilo je otpočinjanje analize arhiviranih podataka, čime je proveravana ispravnost donetih zaključaka.

Posebnu vrstu ograničenja sprovedenog empirijskog istraživanja predstavlja činjenica da u Srbiji, do sada, ne postoje objavljeni rezultati sličnog empirijskog istraživanja, zbog čega dobijeni rezultati ne mogu da se uporede sa rezultatima do kojih su došli drugi istraživači na teritoriji Srbije.

6.

Zaključna razmatranja i budući pravci istraživanja

Pristalice agilnog razvoja pod pojmom "agilna arhitektura" podrazumevaju arhitekturu, koja u potpunosti nastaje kao rezultat samog razvojnog procesa. Reč je o pojavnoj, nascentnoj arhitekuri (Madison, 2010; Hadar & Silberman, 2008; Isham, 2008). Postojeća literatura o agilnim procesima razvoja ne sadrži smernice za arhitekturalne aktivnosti, kao ni uputstva o artifaktima kojima se arhitekturalne odluke trebaju dokumentovati. Agilni procesi razvoja arhitekturalni dizajn vezuju isključivo za tehniku refaktorisanja koda i neformalnu komunikaciju članova tima, licem u lice. Osim verbalnih rasprava, po pitanju odluka dizajna i opšte arhitekture sistema, agilni procesi razvoja ne pružaju nikakve smernice za njihovu skalabilnost za razvoj kompleksnih sistema (Babar, 2009).

Kruchten i Brown pod pojmom "agilna arhitektura" podrazumevaju arhitekturu koja je tako dizajnirana da može biti lako izmenjena tj. može da reaguje na promene zahteva u okruženju. Autori su istakli da razvoj softverskih rešenja agilnim procesima razvoja, ne garantuje da je rezultat nužno agilna softverska arhitektura (Brown, 2013; Kruchten, 2010).

Postavljeni i realizovani istraživački ciljevi doktorske disertacije, na liniji su stavova Kruchten-a i Brown-a. Naime, predloženi okvir uspostavlja arhitekturalni proces, koji zajedno sa agilnim procesom razvoja, omogućava razvoj agilne arhitekture. Drugim rečima, inkorporiranjem identifikovanih, značajnih eksplicitnih arhitekturalnih aktivnosti (prvi istraživački cilj), na odgovarajuća mesta u agilnom procesu razvoja (drugi istraživački cilj), putem predloženog okvira (treći istraživački cilj), formalizovan je proces razvoja agilne arhitekture. Formalizovan arhitekturalni proces ima za cilj iterativno-inkrementalni razvoj

agilne arhitekture, u kojem je nascentna arhitektura zasnovana na dovoljnom obimu up front arhitekturalnih odluka.

Postavljeni okvir obezbeđuje razvoj agilne arhitekture koja podržava agilnost tima, svojom tolerancijom na promene i nascentnu arhitekturu usmerava i kontroliše setom up front arhitekturalnih odluka. Up front arhitekturalne odluke donose se u kraćim periodima planiranja, tokom celog razvojnog procesa. Okvirom se obezbeđuje kontinuirano sagledavanje agilne arhitekture, kako bi se pravovremeno otkrivali arhitekturalni problemi i implementirao potreban refaktoring. Ovakvim razvojem agilne arhitekture obezbeđeno je minimiziranje tehničkog duga i kontinuirano očuvanje/poboljšanje kvaliteta dizajna i njegovo lako prilagođavanje promenljivim zahtevima.

Takođe, rezultati sprovedenog istraživanja impliciraju zaključak da inicijalna postavka arhitekture kompleksnih poslovnih softverskih sistema na početku projekta, obezbeđuje niz benefita. Sledi prikaz benefita i proporcije ocenjivača (broj u zagradi), koji su dati benefit ocenili kao značajan:

- Smanjuje tehnički rizik, jer tim ima viziju koja ga vodi (0.9).
- Obezbeđuje rano identifikovanje najkritičnijih nefunkcionalnih zahteva (0.9).
- Poboljšava produktivnost tima, jer anticipacijom kritičnih tehničkih pitanja umanjuje rizik kretanja razvoja pogrešnim putem (0.75).
- Smanjuje vreme razvoja u smislu izbegavanja velikih i skupih prepravki arhitekture (0.75).
- Poboljšava integraciju razvoj/operacije (devops) (0.75).
- Obezbeđuje efikasno skaliranje agilnog razvoja softvera (0.75).

Kako predloženi metodološko-radni okvir uključuje inicijalnu postavku arhitekture, zaključuje se da navedeni benefiti karakterišu i sam okvir.

Tradicionalni timovi najveći deo planiranja sprovode up front, dok JIT nastoje da u što većoj meri eliminišu. Agilni timovi se rukovode suprotnom strategijom tj. isključivo JIT planiranjem. Postavljenim okvirom implementira se strategija razvoja kompleksnih softverskih rešenja, zasnovana na dovoljno up front i dovoljno JIT planiranju, tokom celog perioda trajanja projekta. Vreme potrebno za up front planiranje u direktnoj je zavisnosti od kontektualnih faktora projekta.

Rezultati sprovedene evaluacije postavljenog metodološko-radnog okvira, upućuju na sledeće zaključke:

- Postavljeni okvir u dovoljnoj meri proširuje agilne procese arhitekturalnim praksama, u cilju njihovog osposobljavanja za razvoj kompleksnih poslovnih sistema.
- Postavljeni okvir je jasno definisan.

- Postavljeni okvir je koristan na projektima razvoja kompleksnih sistema agilnim procesima.
- Postavljeni okvir je primenljiv u organizacijama.
- Upotreba postavljenog okvira doprinosi većem kvalitetu kompleksnih poslovnih softverskih rešenja u agilnim procesima.
- Upotrebom postavljenog okvira smanjuje se rizik erozije arhitekture.
- Postavljeni okvir uravnotežava razvoj funkcionalnosti i arhitekture, neugrožavajući agilnost projekta.

Postavljeni metodološko-razvojni okvir zasnovan je na strategiji koja momenat donošenja i sprovođenja arhitekturalnih odluka zasniva na proceni relativnih troškova. Procena relativnih troškova podrazumeva sagledavanje troškove usled kašnjenja implementacije i odložene isporuke vrednosti klijentu i troškova koji potiču od buduće dorade ili redizajna arhitekture u kasnijim razvojnim fazama. Stoga se može zaključiti da se primenom postavljenog metodološko-radnog okvira ujedno razmatraju i ekonomske implikacije razvoja agilne arhitekture.

Takođe, bitno je istaći da je okvir za razvoj agilne arhitekture uvažava i sledeće Lean principe:

- Odložiti odluku do poslednjeg odgovornog momenta.
- Isporučiti što je brže moguće.
- Sagledaj i optimizuj celinu.
- Radi iterativno i inkrementalno.

Sprovedeno istraživanje obezbeđuju značajne doprinose, kako praktičarima, tako i istraživačima, u naporima premošćivanja jaza između agilnih procesa i arhitekturalnih aktivnosti, prilikom razvoja kompleksnih poslovnih sistema:

- Prikazuje dizajn i rezultate empirijskog istraživanja, koje je imalo za cilj da istraži arhitekturalne prakse koje agilni timovi u praksi smatraju značajnim u razvoju kompleksnih poslovnih softverskih rešenja.
- Pruža empirijske rezultate o tome na koji način agilni timovi pristupaju arhitekturalnim aspektima i koja su kritična mesta agilnog razvoja na kojima su neophodne eksplicitne arhitekturalne prakse, prilikom razvoja kompleksnih sistema.
- Obezbeđuje okvir koji sistematizuje proces razvoja agilne arhitekture.
- Rezultati evaluacione studije potvrđuju korisnost i primenljivost okvira u industriji.
- Industrija poslovnog softvera biće upoznata sa rezultatima empirijskog istraživanja, kroz njihovu desiminaciju u domaćim i međunarodnim naučnim i stručnim časopisima, kao i učešćem autora na konferencijama.
- Identifikovani su arhitekturalni izazovi i problemi sa kojima se agilni timovi u praksi suočavaju, prilikom razvoja kompleksnih sistema agilnim procesima razvoja, što će predstavljati buduće pravce istraživanja.

- Rezultati sprovedenog sistematskog pregleda literature značajni su, jer pokazuju da je do sada obavljeno vrlo malo empirijskih istraživanja iz oblasti razvoja softverske arhitekture u agilnim procesima.
- Rezultati pokazuju da u srbiji do sada nije sprovedeno ni jedno slično empirijsko istraživanje i da je neophodna veća angažovanost naučne zajednice. U tom kontekstu, rezultati istraživanja doprineće kumulativnom rastu znanja iz predmetnog područja.

Realizacijom definisanih istraživačkih ciljeva, obezbeđene su osnove za veću i efikasniju primenu agilnih procesa u razvoju kompleksnih poslovnih softverskih rešenja. Ostvarena su poboljšanja agilnih procesa razvoja u sledećim segmentima:

- sagledavanje celovite slike problema,
- razlaganje problema na delove, na samom početku projekta,
- eksplicitna identifikacija i specifikacija arhitekturalno značajnih zahteva,
- postavka arhitekturalnog "skeleta" za glavni deo sistema, na samom početku projekta,
- iterativan razvoj identifikovanih delova problema i inicijalnog arhitekturalnog "skeleta",
- iterativno testiranje arhitekture,
- kontinuirano sagledavanje arhitekture, čime je gap između nastanka arhitekturalnog problema/greške i njegovog otkrivanja, značajno umanjen,
- uravnotežen odnos razvoja funkcionalnosti vs. arhitekture sistema,
- razvoj arhitekture zasnovan na ekonomskim principima (vrednost/trošak/rizik).

C4: Identifikovani budući pravci istraživanja

Četvrti postavljeni istraživački cilj jesu identifikovani budući pravci istraživanja. Narednim tekstom opisaće se ukratko koja će se pitanja i izazovi istraživati u budućnosti.

Jedan od budućih pravaca biće usmeren na dizajn i sprovođenje empirijskog istraživanja u okviru visokoobrazovnih institucija u Srbiji, koje se bave obrazovanjem studenata u domenu razvoja softvera. Istraživanje će u fokusu imati empirijski identifikovan problem, koji se odnosi na nedovoljno izučavaju znanja i veština iz domena razvoja softverske arhitekture i agilnih procesa. Rešavanje identifikovanog problema od značaja je za IT industriju, jer je u direktnoj vezi sa obezbeđivanjem kvalitetnih, kros-funkcionalnih agilnih timova u praksi, koji su "conditio sine quanon" da bi odgovornost za arhitekturu imali svi članovi tima. Cilj istraživanja biće identifikovanje neophodnih promena i prilagođavanja trenutnih nastavnih planova i programa fakulteta, u odnosu na potrebe IT industrije u Srbiji.

Drugi pravac istraživanja biće usmeren na sprovođenje empirijskog istraživanja, zasnovanog na Grounded theory naučnoj metodi, kako bi se dublje istražili problemi i izazovi u razvoju agilne arhitekture kompleksnih sistema. Rezultati bi bili značajni sa aspekta komparacije sa rezultatima istraživanja sprovedenog u okviru ove doktorske diseratcije, putem Delfi metode.

Pouzdanosti spovedenog istraživanja može se dokazati najefektivnije putem metode ponovnog testiranja. Metod ponovnog testiranja podrazumeva sprovođenje novog empirijskog istraživanja, ali na novom panelu respodenata, kako bi se izmerila konzistentnost dobijenih rezultata. Pošto ovakva merenja pouzdanosti nisu mogla biti sprovedena prilikom izrade doktorske disertacije, jer iziskuju dodatno vreme od 12 meseci, ova vrsta kontrolnog istraživanja predstavlja budući pravac istraživanja. Cilj je da se dobijeni nalazi i izvedeni zaključci dodatno potvrde.

Četvrti pravac istraživanja biće usmeren na rešavanje empirijski identifikovanog problema, koji se odnosi na upravljanje arhitekturalnim znanjem u agilnim procesima, jer je to izazov sa kojim se sučeljavaju agilni timovi u praksi i za koji još uvek nemaju adekvatno rešenje. Upravljanje arhitekturalnim znanjem i njegova ponovna upotreba, u značajnoj meri bi doprinela smanjivanju vremena neophodnom za donošenje up front arhitekturalnih odluka. Istraživanje će imati za cilj da definiše i set kriterijuma za ocenu kvaliteta arhitekturalne dokumentacije.

Peti pravac istraživanja biće fokusiran isključivo na outsoursing projekte, jer oni čine najveći deo industrije razvoja softvera u Srbiji. Cilj empirijskog istraživanja biće identifikovanje izazova i problema u razvoju arhitekture, u kontekstu specifičnih kontekstualnih faktora za ovaj tip projekata. Rezultat istraživanja biće set strategija za razvoj agilne arhitekture na outsoursing projektima, koji će predstavljati dopunu predloženom okviru u doktorskoj disertaciji.

Arhitekturalne aktivnosti zahtevaju velike investicije (sa aspekta svih vrsta resursa), što je ključni razlog za njihovu eliminaciju iz agilnog razvoja i otklon ključnih stejkholdera. Rešenje ovog problema je u merenju njihovi pozitivnih efekata i vrednosti koju obezbeđuju ciljnoj organizaciji. Na ovaj način bi se arhitekturalne aktivnosti lakše pravdale kod stejkholdera, jer bi njihova vrednost bila transparentna. Akademska zajednica pitanje benefita i vrednosti, koje arhitekturalne aktivnosti obezbeđuju oraganizaciji, zanemaruje i stoga je budući pravac istraživanja usmeren na definisanje kriterijuma za njihovo merenje. Istraživanje je komplementarno sa postavljenim okvirom, jer bi obezbedilo uputstva za merenje vrednosti arhitekturalnih aktivnosti i benefita koji pružaju organizaciji.

7.

Literatura

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and Architecture: Can They Coexist? *IEEE Software*, *27*(2), 16–22.

Abrahamsson, P., Salo,O., Ronkainen,J., Warsta, J. (2002). Agile Software Development Methods- Review and Analysis. *VTT Electronic*, *478*, 1–107.

Ahlstrom, P. (1998). Sequences in the Implementation of Lean Production. *European Management Journal*, *16*(3), 327–334.

Akins, R., Tolson, H., Cole, B. R. (2005). Stability of response characteristics of a Delphi panel: application of bootstrap data expansion. *BMC Medical Research Methodology*, *5*(37).

Ambler, S. W. (2001). *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process* (1st ed.). New York, NY: Wiley.

Ambler, S. W., & Lines, M. (2013). *Disciplined agile delivery* (1st ed.). Boston, MA: IBM Press.

America, P., Obbink, H., Rommes, E. (2003). Multi-view variation modeling for scenario analysis. In *Proceedings of Fifth International Workshop on Product Family Engineering (PFE-5)* (pp. 44–65). Siena: Springer-Verlag.

Babar, M. A. (2009). An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches. In *Software Architecture, 2009 European Conference on Software Architecture. Joint Working IEEE/IFIP Conference* (pp. 81 – 90). IEEE.

Babar, M. A. (2014). Making Software Architecture and Agile Approaches Work Together. In M. A. Babar, A. W. Brown, & I. Mistrik (Eds.), *Agile software architecture* (1st ed., pp. 43–76). Waltham, MA: Elsevier.

Babar, M., & Abrahamsson, P. (2008). Architecture-centric methods and agile approaches. In *Proceedings of the 9th international conference on agile processes and eXtreme programming in software engineering* (pp. 238–243). Limerick.

Babar, M. l., Brown,A.W.,Mistrik, I. (2014). *Agile Software Architecture Aligning Agile Processes and Software Architectures* (1st ed.). Waltham, MA: Elsevier.

Balduino, R. (2007). Introduction to OpenUP (Open Unified Process). Retrieved from http://www.eclipse.org/epf/general/OpenUP.pdf

Bardecki, M. J. (1984). Participants' responses to the Delphi method: an attitudinal perspective. *Technological Forecasting and Social Change*, *5*, 281–292.

Barraza, M. F. S., Smith, T ., Dahlgaard- Park, S. M. (2009). Lean-Kaizen public service: an empirical approach in Spanish local governments. *The TQM Journal*, *21*(2), 143–167.

Bass, L., Clements, P., Kazman, R. (2013). *Software architecture in practice* (3rd editio.). Upper Saddle River, NJ: Addison-Wesley.

Bass,L., Klein,M., Bachmann, F. (2001). Quality Attribute Design Primitives and the Attribute Driven Design Method. In *4th International Workshop on Software ProductFamily Engineering* (pp. 169–186). Bilbao, Spain: Springer.

Beck, K. (1999). Embracing Change with Extreme Programming. *IEEE Computer*, *32*(10), 70–80.

Beck, K. (2003). *Test Driven Development: By Example*. Boston: Addison Wesley.

Beck, K. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Boston, MA: Addison-Wesley.

Beck, K., Andres, C. (2004). *Extreme programming explained: embrace change* (2nd ed.). Boston: Addison Wesley.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, J. (2001). The Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

Bellomo, S., Nord, R. L., & Ozkaya, I. (2013). A Study of Enabling Factors for Rapid Fielding Combined Practices to Balance Speed and Stability. In *35th International Conference on Software Engineering (ICSE)* (pp. 982–991). New York, NY: IEE.

Bellomo, S., Ozkaya, I., & Nord, R. (2013). A Study of Enabling Factors for Rapid Fielding Combined Practices to Balance Speed and Stability. In *ICSE Conference 2013* (pp. 982–991). San Francisco: IEEE.

Blair, S., Watt, R., & Cull, T. (2010). Responsibility-Driven Architecture. *IEEE Software*, *27*(2), 26–32.

Boehm, B., Lane, J., Koolmanojwong, S., & Turner, R. (2010). Architected Agile Solutions for Software-Reliant Systems. In *Agile SoftwareDevelopment: Current Research and Future Directions* (pp. 165–184). Springer.

Boehm, B., Basili, V. (2001). Software Defect Reduction Top 10 List. *IEEE Computer*, *34*(1), 135–137.

Boehm, B., Turner, R. (2003). Using risk to balance agile and plandriven methods. *IEEE Computer*, *35*(6), 57–66.

Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B. J. (2007). *Object-Oriented Analysis and Design with Applications* (3rd editio.). Westford, Massachussets: Addison-Wesley.

Bosch, J. (2000). *Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach* (1st editio.). New York, NY: Addison-Wesley.

Brown, N., Nord, R., & Ozkaya, I. (2010). Enabling Agility Through Architecture. *CrossTalk*, 12–16.

Brown, S. (2013). *Software Architecture for Developers*. LeanPub.

Buchgeher, G., Weinreich, R. (2014). Continuous Software Architecture Analysis. In I. Babar, M. l., Brown, A.W., Mistrik (Ed.), *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed., pp. 161–188). New York, NY: Elsevier.

Calderone, C. (2010). Thinking-lean-power-standardized-work-practices.

Chandra, S., Bhattaram, S. (2003). Patterns approach to building software systems. In *STRAW o3 Second International Software Requirements to Architectures Workshop* (p. 200). Oregon, Portland.

Charles, C. M. (1995). *Introduction to educational research* (2nd ed.). San Diego: Longman.

Chen, L.,Babar, M. A. (2014). Towards an Evidence-Based Understanding of Emergence of Architecture Through Continuous Refactoring in Agile Software Development. In *Software Architecture (WICSA), 2014 IEEE/IFIP* (pp. 195 – 204). IEEE.

Clements,P., Bachmann,F., Bass,L., Garlan,D., Ivers,J., Little,R., Merson,P., Nord,R., Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison Wesley.

Clerc, V., Lago, P.,Vliet, H. (2011). Architectural Knowledge Management Practices in Agile Global Software Development. In *Sixth IEEE International Conference on Global Software Engineering Workshops* (pp. 1–8). ACM.

Cockburn, A. (2007). *Agile Software Development: The Cooperative Game* (2nd ed.). Boston, MA: Addison Wesley.

Cohen, D., Lindvall, M., Costa, P. (2004). An Introduction to Agile Methods. *Advances in computers*, *62*(1), 1–66.

Cohn, M. (2005). *Agile Estimating and Planning* (1st ed.). New York, NY: Prentice Hall.

Cohn, M. (2009). *Succeeding with Agile: Software development Using Scrum* (1st ed.). New York, NY: Addison Wesley.

Cooney, C.F., Stebbings, S.N., Roxburgh, M., Mayo, J., Keen, N., Evans, E., & Meehan, T. C. (1995). Integrating nursing research and practice: Part II – a Delphi study of nursing practice priorities for research-based solutions. *Nursing Praxis in New Zealand*, *10*(1), 22–27.

Coplien, J.O., Bjornvig, G. (2010). *Lean architecture for agile software development* (1st ed.). Cornwall: John Wiley and Sons.

Corporation IBM. (2006). Role Set: XP Roles. Retrieved from http://epf.eclipse.org/wikis/xp/xp/rolesets/xp_roles_AC740340.html

Creswell, J. W. (2009). No Title. *Journal of Mixed Methods Research*, *3*(1), 107–111.

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2010). The Scrum Primer. Retrieved from http://www.goodagile.com/scrumprimer/scrumprimer.pdf

Delbecq, A.L., Van de Ven, A., Gustafson, D. (1975). *Group Techniques for Program Planning: A Guide to Normal Group and Delphi Processes* (1st ed.). Glenview, Illinois: Scott, Foreman and Company.

Dingsøyr, T., Nerur, S., Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems & Software*, *85*(1), 1213–1221.

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*, 833–859. doi:10.1016/j.infsof.2008.01.006

Ebert, C., Abrahamsson, P., Oza, N. (2012). Lean software development. *IEEE Computer Society*, *29*(5), 22–25.

Eloranta, V.P., Koskimies, K. (2012). Aligning Architecture Knowledge Management with Scrum. In *Proceedings of the WICSA/ECSA 2012* (pp. 112–115). ACM.

Erickson, J., Lyytinen, K., Siau, K. (2005). Agile modeling, agile software development, and extreme programming. *Journal of Database Management*, *16*(1), 88–100.

Faber, R. (2010). Architects as Service Providers. *IEEE Software*, *1*, 33–40.

Falessi, D., Cantone, G., Sarcia', S. A., Calavaro, G., Subiaco, P., & D'Amore, C. (2010). Peaceful Coexistence: Agile Developer Perspectives on Software Architecture. *IEEE Software*, *27*. doi:10.1109/MS.2010.49

Feigenbaum, A. V. (2008). *Total Quality Control, vol 2* (1st ed.). New York, NY: McGraw-Hill.

Fisher, R. G. (1978). The Delphi method: a description, review and criticism. *Journal of Academic Librarianship*, *4*(1), 64–70.

Fitzsimmons, J. A., & Fitzsimmons, M. J. (2001). *Service Management: Operations, Strategy and Information Technology* (4th ed.). Boston: McGraw-Hill.

Fowler, M. (2004). Is Design Dead? Retrieved from http://www.martinfowler.com/articles/designDead.html

Fowler, M. (2006). Continuous Integration. Retrieved from http://martinfowler.com/articles/continuousIntegration.html

Fowler, M., Beck, K., Brant, K., (Author), Opdyke, W., Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code* (1st ed.). Boston: Addison Wesley.

Friedrichsen, U. (2014). Opportunities, Threats, and Limitations of Emergent Architecture. In I. Babar, M.A., Brown, A. W., Mistrik (Ed.), *Agile Software Architecture Aligning Agile Processes and Software Architectures* (1st ed., pp. 335–355). New York, NY: Elsevier.

Gacek, C., Abd-Allah, A., Clark, B., Boehm, B. (1995). On the Definition of Software Architecture. In *Proceedings of the First International Workshop on Architectures for Software Systems* (pp. 85–95). Seattle, WA.

Garlan, D., & Shaw, M. (1994). An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, *1*(1), 37.

Garland,J., Anthony, R. (Author). (2003). *Large-Scale Software Architecture: A Practical Guide using UML*. New York, NY: Wiley.

Glass, R. (2001). Agile versus traditional: Make love, not war. *Cutter IT Journal*, *14*(12), 12–18.

Goodman, C. M. (1987). The Delphi technique: a critique. *Journal of Advanced Nursing*, *12*, 729–734.

Gordon, T. J. (1992). Gordon, T.J. *Annals of the American Academy of Political and Social Science*, *522*(1), 25–35.

Gorton, I. (2006). *Essential Software Architecture* (1st ed.). Berlin: Springer.

Grady, R. (1992). *Practical software metrics for project management and process improvement* (1st ed.). Upper Saddle River, NJ: Prentice-Hall.

Hadar, I., & Sherman, S. (2012). Agile vs. Plan-Driven Perceptions of Software Architecture. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop* (pp. 50–55). Zurich, Switzerland: IEEE.

Hadar, E., Silberman, G. M. (2008). Agile Architecture Methodology: Long Term Strategy Interleaved with Short Term Tactics. In *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications* (pp. 641–651). ACM.

Hadar, I. , Sherman, S., Hadar, E., Harrison, J. J. (2013). Less is More: Architecture Documentation for Agile Development. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop* (pp. 121–124). IEEE.

Helmer, O., & Rescher, N. (1959). On the Epistemology of the Inexact Sciences. *Management Science*, *6*(1), 25–52.

Hevner, A.R., March, S.T., Park, J., Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105.

Highsmith, J. (2002). *Agile software development ecosystems*. Boston, MA: Addison-Wesley.

Highsmith J. (2002). What is agile software development? *Crosstalk*, 4–9.

Highsmith,J., Cockburn, A. (2001). Agile software development: the business of innovation. *IEEE Computer*, *34*(3), 120–127.

Hines P, Holweg, M., Rich, N. (2004). Learning to evolve. A review on contemporary lean thinking. *Int J of Operations and Production Management*, *24*(10), 994–1011.

Hines, P, Taylor, D. (2000). *Going lean. A guide to implementation*. Cardiff, Great Britain: Lean Enterprise Research Centre.

Hines, P., Lamming, R., Jones, D., Cousins, P., Rich, N. (1999). *Value Stream Management. Strategy and Excellence in the Supply Chain* (1st ed.). New York, NY: Financial Times/ Prentice Hall.

Hinsman, C., Sangal, N., & Stafford, J. (2009). Achieving Agility through Architecture Visibility. In *QoSA '09: Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems* (pp. 116–129). Springer-Verlag.

Hirotaka,T., Nonaka, I. (1986). *The new product development game*.

Hoda, R. (2010). *Self-organizing agile teams: a grounded theory*. Victoria University of Wellington.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., & America, P. (2007). A general model of software architecture design derived from five industrial approaches. *The Journal of Systems and Software*, *80*(1), 106–126. doi:10.1016/j.jss.2006.05.024

Hofmeister, C., Nord, R., Soni, D. (2000). *Applied Software Architecture*. Boston: Addison Wesley.

Holden, R. J. (2010). Lean Thinking in Emergency Departments: A Critical Review. *Annals of Emergency Medicine*, *59*(3), 266.

Hopkins, R., & Harcombe, S. (2014). Agile Architecting: Enabling the Delivery of Complex Agile Systems Development Projects. In I. Babar, M. l., Brown,A.W.,Mistrik (Ed.), *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed., pp. 291–314). New York, NY: Elsevier.

Hsieh, H., Shannon, S. E. (2005). Three Approaches to Qualitative Content Analysis. *Qual Itative Health Research*, *15*(9), 1277–1288.

Huang, JC., Czauderna, A., Mirakhorli, M. (2014). Driving Architectural Design and Preservation from a Persona Perspective in Agile Projects. In M. A. Babar, A. W. Brown, & I. Mistrik (Eds.), *Agile Software Architecture Aligning Agile Processes and Software Architectures* (1st ed., pp. 153–181). Waltham, MA: Elsevier.

Ihme, T., & Abrahamsson, P. (2005). The use of architectural patterns in the agile software development on mobile applications. In *ICAM 2005 Internetional Conference on Agility* (Vol. 8, pp. 1–16).

Imai, M. (2008). *Kaizen - ključ japanskog poslovnog uspeha* (1st ed.). Beograd: Mono i Manjana.

Isham, M. (2008). Agile architecture is possible – you first have to believe! In *In Agile Conference (AGILE '08) (Toronto, 2008)* (pp. 484–489.).

Ishikawa, K. (1985). *What is total quality control?The Japanese Way* (1st ed.). New Jersey, USA: Prentice-Hall.

ISO/IEC. (2007). *Systems and software engineering — Recommended practice for architectural description of software-intensive systems* (1st ed.). New York, NY: ISO/IEC.

ISO/IEC, 42010. (2011). *Systems and software engineering — Architecture description*. New York, NY.

Isotta-Riches, B., & Randell, J. (2014). Architecture as a Key Driver for Agile Success. In I. Babar, M. l., Brown,A.W.,Mistrik (Ed.), *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed., pp. 357–374). New York, NY: Elsevier.

Jacobson, I., Wei Ng,P., Spence, I. (2006). The Essential Unified Process. Retrieved from http://www.drdobbs.com/architecture-and-design/the-essential-unified-process/191601687

Jacobson,I., Booch,G., Rumbaugh, J. (1999). *The Unified Software Development Process* (1st ed.). Boston: Addison Wesley.

Jeffery, G., Hache, G., & Lehr, R. (1995). A group based Delphi application: defining rural career counselling needs. *Measurement and Evaluation in Counselling and Development*, *28*, 45–60.

Jeffries, R. E. (2014). What is Extreme Programming? Retrieved from http://xprogramming.com/what-is-extreme-programming/

Jeon, S., Han, M., Lee, E., Lee, K. (2011). Quality attribute driven agile development. In *Proceedings - 2011 9th International Conference on Software Engineering Research, Management and Applications, SERA 2011* (pp. 203–210). doi:10.1109/SERA.2011.24

Jones, H.,Twiss, B. C. (1978). *Forecasting Technology for Planning Decisions*. New York, NY: Petrocelli Books.

Kaikkonen, H. (2011). *Applying lean thinking in software development organizations*. University of Oulu.

Karlsson, C., Ahlstrom, P. (1996). Assessing changes towards lean production. *International Journal of Operations & Production Management*, *16*(2), 24–41.

Kazman, R., Kruchten, P., Nord, R.L., Tomayko,J., E. (2004). Integrating Software- Architecture-Centric Methods into the Rational Unified Process, (July).

Keeney, S., Hasson, F., & McKenna, H. (2011). *The Delphi Technique in Nursing and Health Research* (1st ed.). London: Wiley-Blackwell.

Keuler, T., Wagner, S., Winkler, B. (2012). Architecture-aware Programming in Agile Environments. In *2012 Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture* (pp. 229–233). IEEE.

Kirk, J., & Miller, M. L. (1986). *Reliability and validity in qualitative research*. Beverly Hills, CA: Sage Publications.

Kitchenham, B. (2004). *Procedures for performing systematic reviews*.

Kroll,P.,Kruchten, P. (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP: A Practitioner's Guide to the RUP* (1st ed.). Boston, MA: Addison Wesley.

Kruchten, P. (1995). Architectural blueprints—the "4+1" view model of software architecture. *IEEE Software*, *12*(6), 42–50.

Kruchten, P. (2004). *The Rational Unified Process: An Introduction* (2nd ed.). Boston, MA: Addison Wesley.

Kruchten, P. (2008). Situated agility: context does matter, a lot. In *9th International conference on agile processes and eXtreme programming in software engineering*. Limerick.

Kruchten, P. (2010). Software architecture and agile software development. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10* (Vol. 2, pp. 497–498). doi:10.1145/1810295.1810448

Kruchten, P. (2010). Software Architecture and Agile Software Development—A Clash of Two Cultures? In *Software Engineering, 2010 ACM/IEEE 32nd International Conference* (pp. 497 – 498). IEEE.

Kruchten, P. (2012). Complexity made simple. In *In Proceedings of the Canadian Engineering Education Association*.

Kruchten, P. (2013). Contextualizing agile software development. *Journal of Software: Evolution and Process*, *25*(1), 351–361.

Kruchten, P., Obbink, H., Stafford, J. (2006). The past, present, and future for software architecture. *IEEE Software*, *23*(2), 22–30.

Kruchten, P., Obbink, H., Staord, J. (2006). he past, present, and future for software architecture. *IEEE Software*, *23*(2), 22–30.

Landis, J.R., Koch, G. . (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, *33*(1), 159–174.

Larman, C., Vodde, B. (2009). Lean Primer.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises* (1st ed.). MA: Addison Wesley.

Liker, J. K. (2004). *The Toyota Way* (1st ed.). New York, NY: McGraw-Hill.

Liker, J.K., Meier, D. (2006). *The Toyota Way Fieldbook. A practical guide for implementing Toyota"s 4Ps*. New York, NY: McGraw-Hill.

Liker, J.K., Morgan, J. M. (2006). The Toyota Way in Services: The Case of Lean Product Development. *Academy of Management Perspectives*, *20*(2), 5–20.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry* (1st ed.). Beverly Hills, CA: Sage.

Linstone,H.A.,Turoff, M. (1975). *The Delphi Method Techniques and Applications* (1st ed.). Boston, MA: Addison Wesley.

Madison, J. (2010). Agile–Architecture Interactions. *IEEE Software*, *27*, 41–48. doi:10.1109/MS.2010.35

Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. *The MIT Press*, 31–2.

Martin, J. (1984). *An Information Systems Manifesto* (1st. ed.). New Jersey, USA: Prentice Hall.

Matković, P. (2011). *Metodološko radni okvir za realizaciju softverskih projekata male i srednje veličine*. Univerzitet u Novom Sadu, Ekonomski fakultet Subotica.

Matković, P., Tumbas, P., & Sakal, M. (2011). The RSX model: traditionalisation of agility. *Strategic Management*, *16*(2), 74–83.

McKenna, H.P., Keeney, S., Bradley, M. (2003). Generic and specialist nursing roles: views of community nurses, general practitioners, senior policy makers and members of the public. *Health and Social Care in the Community*, *11*(6), 537–545.

Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *EEE Trans. Softw. Eng.*, *26*(1), 70–93.

Melnik,G., Maurer, F. (2004). Introducing Agile Methods: Three Years of Experience. In *Proceedings of the 30th Euromicro Conference. IEEE.* (pp. 334–341).

Merisalo-Rantanen, H., Tuure, T., Matti, R. (2005). No TitleIs extreme programming just old wine in new bottles: a comparison of two cases. *Journal of Database Management*, *16*(4), 41–61.

Middleton, P. (2001). Lean Software Development: Two Case Studies. *Software Quality Journal*, *9*(1), 241–252.

Middleton, P., Flaxel, A., Cookson, A. (2005). Lean Software Management Case Study: Timberline Inc. *Extreme Programming and Agile Processes in Software Engineering*, *3556*, 1–9.

Miles, M. B., & Huberman, A. . (1994). *Qualitative data analysis: An expanded sourcebook* (2nd ed.). Sage.

Miller, L. M. (2005). Lean Teams Developing the Team-Based Organization: The Skills and Practices of High Performance Business Teams.

Milunsky, J. (2009). Agilesoftwaredevelopment.

Mirakhorli, M., Cleland-Huang, J. (2013). Traversing the twin peaks. *IEEE Software*, *30*(2), 30–36.

Moe,N.B., Dingsøyr,T., Dybå, T. (2009). No Title. *IEEE Software*, *26*(1), 20–26.

Moeller, G. H., & Shafer, E. L. (1994). The Delphi technique a tool for long-range travel and tourism planning. In: Travel, Tourism, and Hospitality Research. In J. R. B. Goeldner & RichieC.R. (Eds.), *A Handbook for Managers and Researchers* (pp. 473–480). Washington, DC,: USDA Forest Service.

Naftanaila, I., Paul, B. (2009). Lean principles applied to software development – avoiding waste. *Economia. Seria Management*, *12*(2), 162–170.

Nord, R. L., Ozkaya, I., & Sangwan, R. S. (2012). Making Architecture Visible to Improve Flow Management in Lean Software Development. *IEEE Software*, *1,* 33–39.

Nord, R.L., Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *Software IEEE*, *23*(2), 47–53.

Northouse, P. G. (2015). *Introduction to Leadership* (3rd ed.). SAGE Publications, Inc.

Nuseibeh, B. (2001). Weaving together requirements and architecture. *Computer*, *34*(3), 115–119.

Ohno, T. (1988). *Toyota production system: Beyond large-scale production* (1st ed.). Massachusetts, USA: Productivity press.

Owens, C., Ley, A., Aitken, P. (2008). Do different stakeholder groups share mental health research priorities? A four-arm Delphi study. *Health Expectations*, *11*(4), 418–431.

Pareto, L., Sandberg, A., Eriksson, P., Ehnebom, S. (2011). Prioritizing Architectural Concerns. In *Software Architecture (WICSA),2011 Ninth Working IEEE/IFIP Conference* (pp. 22–31). IEEE.

Parnas, D. L. (1974). On a "Buzzword": Hierarchical Structure. In *Proceedings of the IFIP Congress'74* (pp. 336–339).

Parsons, R. (2008). Architecture and agile methodologies—how to get along. In *WICSA*.

Pérez,J., Díaz,J., Garbajosa,J., Yagüe, A. (2014). Bridging User Stories and Software Architecture: A Tailored Scrum for Agile Architecting. In I. Babar, M. l., Brown,A.W.,Mistrik (Ed.), *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed., pp. 215–241). New York, NY: Elsevier.

Perry, D.E., Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, *17*(4), 40–52.

Polit, D. F., Beck, C. T. (2006). The Content Validity Index: Are You Sure You Know What's Being Reported? Critique and Recommendations. *Research in Nursing & Health*, *29*, 489–497.

Poppendieck, M., Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *Software IEEEEEE*, *29*(5), 26–32.

Poppendieck, M.,Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Upper Saddle River, NJ: Addison-Wesley.

Porter, M. (2008). *Competitive advantage: Creating and sustaining superior performance* (reprint.). New York, NY: Simon&Schuster.

Punch, K. (2005). *Introduction to social research: Quantitative and qualitative approaches* (2nd ed.). London: SAGE.

Putman,J., R. (2000). *Architecting with RM-ODP* (1st ed.). New Jersey, USA: Prentice Hall.

Qureshi, M. R. J. (2012). Agile software development methodology for medium and large projects. *IET Software*. doi:10.1049/iet-sen.2011.0110

Rational Team. (2010). Rational Unified Process, v7.5.1. Armonk: IBM Rational Software Corporation.

Ristić, Ž. (2011). *Kvantitativna, kvalitativna i mešovita istraživanja* (prvo.). Novi Sad: Univerzitet u Novom Sadu.

Rother, M., Shook, J. (1999). *Learning to see. Value stream mapping to create value and eliminate muda*. New York, NY: Lean Enterprise Institute.

Rozanski, N., Woods, E. (2012). *Software Systems Architecture* (Second Edi.). Upper Saddle River, NJ: Addison-Wesley.

Rubin, K., S. (2012). *Essential Scrum* (1st ed.). New York, NY: Addison Wesley.

Sackman, H. (1975). *Delphi-Critique: Expert Opinion, Forecasting and Group Process*. Massachusetts, USA: Lexington Books.

Sandrey, M., & Bulger, S. (2008). The Delphi method: an approach for facilitation evidence based practice in athletic training. *Althetic Training Educaci´on Journal*, *3*(4), 135–142.

Schwaber, K. (2004). *Agile project management with scrum. Microsoft Press* (1st ed.). Washington: Microsoft Press.

Schwaber, K., Beedle, M. (2002). *Agile Software Development with Scrum* (1st ed.). New York, NY: Prentice Hall.

Schwaber,K., Sutherland, J. (2013). *The Definitive Guide to Scrum: The Rules of the Game*.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline* (First Edit.). New York, NY: Prentice Hall.

Shuja,A.,K., Krebs, K. (2008). *IBM Rational Unified Process Reference and Certification Guide: Solution Designer (RUP)* (1st ed.). Boston, MA: IBM Press.

Skulmoski, G. J., Hartman, F. T., & Krahn, J. (2007). The Delphi method for graduate research. *Journal of Information Technology Education*, *6*(1), 1–21.

Sommerwille, I. (2010). *Software Engineering 9* (9th ed.). Boston, MA: Addison Wesley.

Stal, M. (2014). Refactoring Software Architectures. In *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (pp. 130–152). W: Elsevier.

Stapleton, J. (1997). *DSDM Dynamic Systems Development Method* (1st ed.). UK: Addison Wesley.

Stephens, M., Rosenberg, D. (2003). *Extreme Programming Refactored: The Case Against XP*. California: Apress.

Sutherland, J., Schwaber, K. (1995). Business object design and implementation. In *OOPSLA '95 Workshop Proceedings* (p. 118). Austin,Texas.

Taylor-Powell, E., & Renner, M. (2003). *Analyzing Qualitative Data*.

Thapparambil, P. (2005). Agile architecture: pattern or oxymoron? *Agile Times*, *6*(1), 43–48.

Tyree, J., Akerman, A. (2005). Architecture Decisions: Demystifying Architecture. *IEEE Software*, *22*(2), 19–27.

van der Ven, S.J., Bosch, J., Groningen, F. (2014). Architecture Decisions: Who, How, and When? In *Agile Software Architecture Aligning Agile Processes and Software Architectures* (pp. 182–212). Waltham, MA: Elsevier.

Veal, A. J. (1992). *ResearchMethods for Leisure and Tourism: a Practical Guide*. Edinburgh Gate: Harlow, United Kingdom: Pearson Education.

Wanga, X., Conboyb, K., Cawleyc, O. (2012). Leagile software development: An experience report analysis of the application of lean approaches in agile software development. *The Journal of Systems and Software*, *85*(1), 1287–1299.

Waterman, M., Noble, J., Allan, G. (2012). How much architecture? Reducing the up-front effort. In *AGILE India, 2012,* (pp. 56 – 59). IEEE.

Weitzel, B., Rost, D., & Scheffe, M. (2014). Sustaining Agility through Architecture. In *2014 IEEE/IFIP Conference on Software Architecture* (pp. 53–56). IEEE.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord,R., Wood, B. (2006). *Attribute-Driven Design (ADD), Version 2.0*. MA.

Womack, J., Jones, D. (2003). *Lean Thinking: Banish Waste and Create Wealth in your Corporation* (2nd ed.). New York, NY: Free Press.

Womack, J., Jones, D. (2009). *Can, Lean solutions: How companies and customers Together, create value and wealth* (1st ed.). New York, NY: Free Press.

Womack, J., Jones, D., Roos, D. (1990). *The Machine that changed the world* (1st ed.). New York, NY: Macmillan Publishing.

Wout,J., Waage,M., Hartman,H., Stahlecker,M., Hofman, A. (Author). (2010). *The Integrated Architecture Framework Explained: Why, What, How* (2010 editi.). Springer.

Yavuz, M. (2010). Fuzziness in JIT and Lean Production Systems. *Production Engineering and Management under Fuzziness Studies in Fuzziness and Soft Computing*, *252*(1), 59–75.

Yousuf, M. I. (2007). The Delphi technique. *Essays in Education*, *20*, 80–89.

Zhang, Y., & Wildemuth, B. M. (2009). Qualitative analysis of content. In *Applications of Social Research Methods to Questions in Information and Library Science* (1st ed., pp. 308–319). Westport: CT: Libraries Unlimited.

Ziglio, E. (1996). The Delphi method and its contribution to decision-making. In E. Adler,M., Ziglio (Ed.), *Gazing Into the Oracle: the Delphi Method and Its Application to Social Policy and Public Health* (1st ed., pp. 3–33). London: Jessica Kingsley Publishers.

Zinn, J., Zalokowski, A., & Hunter, L. (2001). Identifying indicators of laboratory management performance: a multiple consistency approach. *Health Care Management Review*, *26*, 40–53.

8.

Prilozi

Prilog 1: Pozivno pismo ekspertima za učešće u istraživanju

Poštovani **<<ime i prezime eksperta>>,**

u okviru izrade doktorske disertacije: "Metodolosko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima" sprovodi se empirijsko istraživanje, kako bi se ostvarili sledeći istraživački ciljevi:

C1. Identifikovane tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja.

C2. Identifikovana kritična mesta u agilnom procesu razvoja pogodna za primenu tradicionalnih arhitekturalnih praksi.

C3. Razvijen radni okvir za inkorporaciju tradicionalnih arhitekturalnih praksi u agilni proces razvoja.

Postavljeni ciljevi treba da doprinesu rešavanju sledećeg postavljenog istraživačkog problema:
pronaći adekvatan balans između unapred planiranog razvoja arhitekture i implementacije funkcionalnosti, prilikom razvoja kompleksnog poslovnog softvera agilnim procesima.

Istraživanje će se sprovoditi Delfi tehnikom, zbog čega su predviđene tri iteracije istraživanja, u cilju postizanja saglasnosti eksperata od 70%. Prva iteracija istraživanja će se sprovoditi direktnim intervjuisanjem, za šta je potrebno da izdvojite oko 60 minuta. Nakon obrade tako prikupljenih podataka, otpočeće druga iteracija istraživanja, koja podrazumeva popunjavanje e-upitnika, za šta će biti dovoljno da izdvojite oko 30 minuta. Nakon obrade podataka iz drugog kruga istraživanja nastupiće treća, završna iteracija. Treća iteracija takođe podrazumeva popunjavanje e-upitnika, a planirano vreme trajanja je oko 15 minuta.

Učešće u ovom istraživanju podrazumeva da zadovoljavate sledeće kriterijume:

- znanje i praktična iskustva u domenu predmeta istraživanja: softverska arhitektura i agilni procesi u razvoju poslovnih softverskih rešenja,

- kapacitet i spremnost učesnika da doprinese istraživanju,
- potvrda da će imati vremena i biti dovoljno posvećen u istraživanju,
- dobre komunikacione veštine,
- akademski nivo obrazovanja u domenu informacionih tehnologija,
- višegodišnje profesionalno iskustvo (više od 5 godina).

Vaše učešće u ovom istraživanju je potpuno dobrovoljno i anonimno, te se ni objavljivanjem rezultata istraživanja neće navoditi imena učesnika niti kompanija u kojima rade, kao ni bilo koje druge informacije koje bi mogle da ukazuju na njihovu identifikaciju. Iskreno se nadam Vašem učešću tokom celog postupka istraživanja i unapred sam Vam zahvalna na vremenu, volji i naporu koji ćete uložiti.

Srdačan pozdrav,
Mirjana Marić.

Prilog 2: Indeksi sadržajne valjanosti pitanja, upitnika prvog kruga istraživanja

Tabela P-2.1 Indeksi sadržajne valjanosti pitanja prve verzije upitnika

Pitanje	Indeks sadržajne valjanosti pitanja
Pitanje 1	0.6
Pitanje 2	1
Pitanje 3	1
Pitanje 4	0.8
Pitanje 5	0.8
Pitanje 6	0.8
Pitanje 7	0.8
Pitanje 8	1
Pitanje 9	0.8
Pitanje 10	1
Pitanje 11	0.8
Pitanje 12	1
Pitanje 13	0.4
Pitanje 14	0.4
Pitanje 15	0.6
Pitanje 16	1
Pitanje 17	0.6
Pitanje 18	0.4
Pitanje 19	0.8
Pitanje 20	0.6
Pitanje 21	0.6
Pitanje 22	0.8
Pitanje 23	0.8
Pitanje 24	0.8
Pitanje 25	0.8
Pitanje 26	0.4
Pitanje 27	0.6
Pitanje 28	0.4
Pitanje 29	0.2
Pitanje 30	1
Pitanje 31	0.8
Pitanje 32	1
Pitanje 33	0.4
Pitanje 34	0.8
Pitanje 35	0.4
Pitanje 36	0.8
Pitanje 37	0.8

Pitanje 38	0.6
Pitanje 39	0.6
Pitanje 40	0.8
Pitanje 41	0.6
Pitanje 42	0.8
Pitanje 43	0.8
Pitanje 44	1
Pitanje 45	1
Pitanje 46	1
Pitanje 47	1
Pitanje 48	0.8
Pitanje 49	1
Pitanje 50	1
Pitanje 51	0.8
Pitanje 52	1
Pitanje 53	1
Pitanje 54	0.8
Pitanje 55	0.2
Pitanje 56	1
Pitanje 57	1

Prilog 3: Dopis ekspertima za ocenu značajnosti pitanja polustrukturiranog intervjua

Poštovani **<<ime i prezime eksperta>>**,

u okviru izrade doktorske disertacije pod nazivom "Metodološko radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima" biće sprovedeno empirijsko istraživanje putem polustrukturiranog intervjua sa ekspertima. Istraživanje će biti zasnovano na Delfi tehnici, a predložena struktura intervjua će biti korišćena u I iteraciji istraživanja. Molim Vas da u nastavku ocenite značajnost svakog predloženog pitanja u odnosu na istraživačka pitanja i postavljeni cilj istraživanja čiji opis sledi u nastavku. Pitanja možete i komentarisati na linijama koje su predviđena za komentare.

Problem istraživanja: iznalaženje balansa između planiranog razvoja arhitekture i razvoja funkcionalnosti u agilnim procesima, u slučaju kompleksnih poslovnih softverskih rešenja.

Definisani ciljevi istraživanja:

C1. Identifikovane tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja.

C2. Identifikovana kritična mesta u agilnom procesu razvoja pogodna za primenu tradicionalnih arhitekturalnih praksi.

C3. Razvijen radni okvir za inkorporaciju tradicionalnih arhitekturalnih praksi u agilni proces razvoja.

Unapred zahvalna,

Mirjana Marić

Prilog 4: Konačan set pitanja upitnika za sprovođenje prvog kruga istraživanja, polustrukturiranim intervjuom

Tabela P-4.1 Instrument I kruga istraživanja

I Podaci o ispitaniku i kontekstu

1. Koliko godina iskustva imate u razvoju softvera?

2. Koje ste uloge imali u projektnim timovima razvoja do sada?

3. Koliko dugo radite (ste radili) kao softver arhitekta?

4. Da li imate iskustva u agilnom/tradicionalnom razvoju softvera?

5. Opišite ukratko na kojim razvojnim projektima ste do sada radili i na kojim aktivnostima?

6. Šta je za Vas softverska arhitektura?

7. Šta je za Vas agilni, a šta tradicionalni pristup razvoja softvera?

II Podaci o modelima razvojnih procesa

1. Koje modele razvojnih procesa koristite u Vašim projektima?

2. Kako su organizovani ljudski resursi (projektni timovi) involvirani u razvoj?

3. Koliko iskustva na projektima agilnog razvoja su imali ljudski resursi (članovi tima)?

4. Opišite proces razvoja softverske arhitekture u agilnom razvoju koji koristite.

- Koje tehnike agilnog razvoja koristite u razvoju softverske arhitekture?Zašto?
- Da li su one dovoljne u razvoju kompleksnih poslovnih softverskih rešenja? Ili koristite i neke prakse karakteristične za tradicionalan razvoj softverske arhitekture?

III Podaci o identifikovanim problemima i uzrocima

1. Navedite i opišite sa kojim problemima ste se susreli u procesu dizajna softverske aritekture na projektima agilnog razvoja?

2. Šta su, po Vašem mišljenju, bili uzroci tih problema? Objasnite navedene uzroke.

3. Da li su po Vašem mišljenju identifikovani problemi mogli biti sprečeni ili ublaženi?

4. Da li su na početku projekta ovi problemi bili identifikovani kao rizici projekta, kojima treba upravljati?

5. Koje faktore uspeha smatrate presudnim u razvoju arhitekture kompleksnih sistema?

6. Šta prema Vašem mišljenju utiče na potrebu većeg planiranje arhitekture na početku projekta?

IV Podaci o softverskoj arhitekturi

1. Da li se na projektima bavite poslovnim modelovanjem u cilju sticanja informacija o kontekstu i radnom okruženju organizacije u kojoj će sistem biti implementiran?

- Kako dokumentujete te informacije?

- Na koji način uspostavljate vezu između identifikovanih poslovnih problema (ciljne organizacije) i arhitekture budućeg sistema?
- Kakva je veza poslovne arhitekture ciljne organizacije i arhitekture budućeg sistema?

2. Da li softver arhitekta učestvuje u identifikaciji zahteva? Zašto? Kojim tehnikama?

3. Na koji način utvrđujete arhitekturalno značajne zahteve (funkcionalne i nefunkcionalne) i kako ih dokumentujete?

4. Da li ste se susretali sa arhitekturalnim konfliktima u zahtevima stejkholdera i na koji način ste ih rešavali?

5. Šta po Vama usmerava razvoj arhitekture *(architectural drivers)*?

6. Čime se vodite prilikom utvrđivanja redosleda razvoja funkcionalnosti sistema? Da li razmatrate njihov značaj sa aspekta arhitekture?

7. Da li i na koji način vršite procenu rizika na početku projekta? Kako njima upravljate?

8. Na koji način identifikujete potencijalna arhitekturalna rešenja? Kako vršite izbor?

9. Kojim artifaktima dokumentujete softversku arhitekturu?

10. Da li koristite neki frejmvork prilikom razvoja softverske arhitekture? A arhitekturalne stilove i uzore (paterne)? Zašto?

11. Kada i na koji način identifikujete podsisteme i slojeve budućeg sistema? A kada klase i pakete klasa? Kako ih dokumentujete?

12. Kako identifikujete i dokumentujete mehanizme dizajna?

13. Kada donosite odluku o mapiranju softvera i hardvera tj. raspoređivanju softverskih elemenata na odgovarajuću mrežu fizičkih elemenata – čvorova na kojima će se izvršavati (model raspoređivanja)?

14. Da li je model podataka značajan u razvoju softverske arhitekture kompleksnih poslovnih softverskih rešenja? Kada ga razmatrate?

15. Kada razmatrate način transformacije identifikovanih elemenata dizajna u elemente implementacije (model implementacije)?

V Kontekstualni faktori

1. Kakva je veza između kompleksnosti (domena) i planiranja softverske arhitekture na početku projekta?

2. Da li obimnije razmatranje arhitekture na početku projekta smanjuje tehničke rizike na nivo koji je prihvatljiv timu i stejkholderima?

3. Da li povezivanje sistema sa nasleđenim sistemima i podacima utiče na potrebu većeg planiranja arhitekture na početku projekta? Objasnite.

4. Na koji način tip projekta utiče na obim planiranja softverske arhitekture na početku projekta?

5. Da li je efikasno funkcionisanje i održavanje kompleksnog softverskog rešenja uslovljeno većim planiranjem arhitekture na početku projekta?

6. Da li višeplatformsko rešenje utiče na obimnije planiranje arhitekture? Objasnite.

7. Da li osobine i broj agilnih timova na projektu utiču na potrebu obimnijeg planiranja softverske arhitekture na početku projekta?

8. Da li su veličina projekta i veličina organizacije koja će implementirati budući sistem faktori koji utiču na potrebu obimnijeg planiranja arhitekture na početku projekta?

Prilog 5: Instrukcije za ocenjivanje sadržajne valjanosti pitanja upitnika, instrumenta drugog kruga istraživanja

Poštovani eksperte,

drugi krug istraživanja Delfi metodom, ima za cilj da putem novog instrumenta istraživanja – upitnika, sumira sve odgovore eksperata, koji su učestvovali u prvom krugu istraživanja. Izradi upitnika prethodila je analiza svih sprovedenih intervjua, na osnovu čega su generisana pitanja za upitnik, vodeći računa da su na liniji postavljenih istraživačkih ciljeva doktorske disertacije:

C1. Identifikovati tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja.
C2. Identifikovati kritična mesta u agilnom procesu razvoja, koja su pogodna za primenu tradicionalnih arhitekturalnih praksi.

Cilj upitnika je i da približi stavove i mišljenja intervjuisanih eksperata, po pitanju razvoja softverske arhitekture kompleksnih poslovnih softverskih rešenja upotrebom agilnih procesa razvoja.

U skladu sa metodologijom istraživanja, potrebno je da predloženi instrument istraživanja (upitnik) oceni nekoliko eksperata.Vaš zadatak, kao eksperta, je da ocenite značajnost svakog pitanja (stavke pitanja) na upitniku za ostvarivanje navedenih istraživačkih ciljeva doktorske disertacije (C1 i C2).

Uz svako pitanje na upitniku stoji skala od 1-4, na kojoj treba da ocenite njegovu značajnost:

1. nije značajno
2. delimično je značajno
3. značajno
4. ekstremno značajno

Pojedina pitanja su sačinjena od više stavki (podpitanja), te je potrebno da svaku pojedinačnu stavku rangirate na skali od 1-4. Imate mogućnost i da za svako pitanje napišete svoj komentar ili sugestiju.

Molim Vas da ocenite sva pitanja, kako bih mogla sprovesti validnu analizu i dobila kvalitetan instrument za sprovođenje druge faze istraživanja.

Unapred zahvalna na izdvojenom vremenu i uloženom naporu.

Srdačan pozdrav,
Mirjana Marić.

Prilog 6: Indeksi sadržajne valjanosti pitanja upitnika za drugi krug istraživanja

Tabela P-6.1 Indeksi sadržajne valjanosti pitanja (ISVP)

Pitanje/stavka upitnika	ISVP
1. Da bi razumeli biznis i poslovne procese, identifikovali zahteve značajne za arhitekturu i rešili eventualne kontradiktornosti i konflikte stejkholdera, arhitekte treba da: imaju direktnu i kontinuiranu komunikaciju sa stejkholderima sistema,ili da zajedno sa biznis analitičarem povremeno budu uključeni u komunikaciju sa stejkholderima.	1
2. Koliko je dobra komunikacija softver arhitekte sa stejkholderima značajna za uspostvaljanje ravnoteže između up front i emergent arhitekture?	0.8
3. Koliko je za kvalitet softverske arhitekture važno uključivanje softver arhitekte kroz ceo razvojni proces i njegova tesna kolaboracija sa razvojnim timom uz kontinuirano deljenje ideja tokom trajanja projekta?	1
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [1. Istraživanje tehnologija i tehnoloških inovacija]	1
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [2. Izučavanje iskustava drugih korisnika određene tehnologije]	0.4
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [3.Istraživanje odgovarajućeg frejmvorka za implementaciju]	0.8
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [4. Istraživanje postojećih biblioteka klasa]	0.8
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [5. Razmatranje postojećih arhitekturalnih rešenja]	0.8
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [6.Razmatranje arhitekturalnih rešenja koje nude prodavci]	0
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [7. Razmatranje postojeće infrastrukture u ciljnoj organizaciji]	1
4. Koliko je značajno prilikom postavke arhitekture, na početku projekta sprovesti sledeće aktivnosti: [8. Istraživanje tržišta iz domena problema u cilju bolje spoznaje arhitekturalnih zahteva]	0.8
5. Da li se slažete sa sledećim stavom: arhitekturalno značajni zahtevi su rezultat diskusija sa stejkholderima, analize funkcionalnih i nefunkcionalnih zahteva i anticipiranja budućih pravaca razvoja biznisa.	1
6. Koliko je za arhitekturu značajno identifikovanje budućih ciljeva i promena, pravaca razvoja biznisa?	0.8
7. Da li je bitno unapred identifikovati tačke arhitekture koje trebaju biti fleksibilnije u cilju budućih promena i odluka u kontekstu proširivosti softvera?	0.8
8. Da li smatrate da je značajno da nefunkcionalni zahtevi sistema budu eksplicitno identifikovani?	1
9. Da li smatrate korisnim pravljenje jedinstvene liste (product backlog-a celog proizvoda) funkcionalnih i nefunkcionalnih zahteva i zahteva budućih promena?	0.8
10. Koliko je značajno prilikom utvrđivanja prioriteta na jedinstvenoj listi zahteva, razmatrati njihove vrednosti ne samo sa aspekta biznisa, već i nivo rizika i uticaja na arhitekturu?	0.8
11. Da li je značajno obezbediti vidljivost arhitekturalno značajnih zahteva i testova prihvatljivosti kroz postavku arhitekturalnih zadataka na listi zadataka (backlogu) ili Kanban tabli kroz ceo proces razvoja?	0.8
12. Da li ceo projekat ugrožen usled razmatranja arhitekture na nivou samo jedne iteracije?	0.8
13. Da li je značajno planiranje arhitekture iznad jednog release-a, u cilju identifikovanja i razvoja arhitekturalnih elemenata koji možda nisu potrebni trenutnom release-u, ali će potencijalno biti inkorporirani u njega, zbog anticipiranih budućih ciljeva stejkholdera?	0.8
14. Da li je prilikom planiranja release-a značajno sprovoditi: [1. Analizu međusobnih zavisnosti funkcionalnosti]	1

14. Da li je prilikom planiranja release-a značajno sprovoditi: [2. Analizu zavisnosti funkcionalnih i nefunkcionalnih zahteva]	0.8
15. Koliko je značajno pre implementacije release-a utvrditi zajedničke komponente i zajedničku infrastrukturu seta funkcionalnosti?	0.8
16. Da li je značajno sprovoditi estimaciju veličine tehničke korisničke priče i brzine njenog razvoja?	0.8
17. Kada je u agilnom procesu potrebno sprovoditi pregled (review) arhitekture?	1
18. Da li je pre početka novog release-a potrebno eksplicitno razmatrati da li postavljena arhitektura može da podrži funkcionalnosti budućeg release-a?	0.8
19. Da li je značajno unapred (tokom tekućeg release-a) identifikovati potrebne arhitekturalne izmene pre implementacije narednog release-a?	0.8
20. Da li sprovođenje kontinuiranog pregleda (review) arhitekture može, pored standardnih agilnih praksi (kontinuirano testiranje, kontinuirana analiza koda, kontinuirana integracija koda, kontinuirano refaktorisanje i programiranje u paru) obezbediti dodatnu kontinuiranu kontrolu kvaliteta?	1
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [1. arhitekturalni spike]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni review arhitekture: [2. vremenski ograničen proof of concept]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [3. test case-ovi]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [4. regresioni testovi]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [5. prototipovi]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [6. statička analiza koda u cilju analize struktura i pravila kodiranja]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [7. simulacija konteksta sistema]	0.8
21. Koja od sledećih tehnika je 3 i korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [8. mišljenje odbora za arhitekturu]	0.8
21. Koja od sledećih tehnika je korisna za proveru ključnih/kritičnih nefunkcionalnih zahteva sistema i formalni pregled (review) arhitekture: [9. scenario analiza interakcije stejkholdera sa sistemom sa fokusom na nefunkcionalne zahteve]	0.8
22. Da li smatrate korisnom strategiju Test Driven Development sa fokusom na nefunkcionalne zahteve?	1
23. Ocenite značaj strategije koja momenat donošenja i sprovođenja arhitekturalnih odluka zasniva na proceni relativnih troškova, sa jedne strane usled kašnjenja implementacije i troškova usled eventualne dorade ili redizajna rešenja u fazi implementacije.	0.8
24. Kada se treba fokusirati na softversku arhitekturu u kontekstu agilnog razvoja?	1
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [1. Za komunikaciju među organizacijama koje su uključene u razvoj, produkciju, operativne aktivnosti i održavanje sistema]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [2. Kao input za kasniji dizajn sistema i razvojne aktivnosti]	1
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [3. Za dokumentovanje osnovnih postavki sistema, njegove namene i okruženja]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [4. Za analizu i procenu alternativnih arhitekturalnih rešenja]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [5. Za komunikaciju sa stejkholderima sistema]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [6. Da podrži pregled (review), analizu i evaluaciju sistema]	0.8

26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [7. Da pomogne planiranje tranzicije sa postojeće (legacy) arhitekture sistema na novu]	1
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [8. Kao specifikacija za grupu sistema koji dele set funkcija]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [9. Da podrži skaliranje agilnih praksi na velikim i kompleksnim projektima]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [10. Da dokumentuje tačke fleksibilnosti ili ograničenja sistema za buduće zahteve]	1
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [11. Sredstvo razvoja i održavanja dokumentacije]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [12. Za operativnu i infrastrukturnu podršku; upravljanje konfiguracijom i prepravkama; redizajn i održavanje sistema, komponenti]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [13. Da uspostavi kriterijume za sertifikaciju implementacije u cilju usklađenosti sa postavljenom arhitekturom za glavni deo sistema]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [14. Za komunikaciju između klijenata, korisnika i programera, kao deo dogovorenog ugovora]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [15. Da podrži planiranje sistema i budžetske aktivnosti]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [16. Da podrži pripremu dokumentacije za akviziciju]	0.8
26. U kontekstu agilnog razvoja, koliko je značajna svaka od sledećih namena softverske arhitekture? [17. Kao input za izbor generatora sistema i alata za analizu]	0.2
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [1. Poboljšava produktivnost, anticipacijom kritičnih tehničkih pitanja umanjujući rizik kretanja razvoja pogrešnim putem]	1
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [2. Smanjuje tehnički rizik, jer tim ima viziju koja ga vodi]	1
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [3. Izbegavanje tehničkog duga]	0.8
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [4. Poboljšava svesnost razvojnog tima o vezi sistema i organizacije koja ga implementira (njenih ciljeva, infrastrukture, strategije i dr.)]	1
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [5. Smanjuje vreme razvoja u smislu izbegavanja velikih i skupih prepravki]	0.8
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [6. Rano identifikovanje najkritičnijih nefunkcionalnih zahteva]	1
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [7. Poboljšava komunikaciju, prezentujući stejkholderima šta će se graditi i na koji način]	0.8
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [8. Poboljšava integraciju razvoj/operacije (DevOps)]	1
27. Ocenite sledeće benefite od inicijalne postavke (envision) arhitekture na početku agilnog projekta: [9. Skaliranje agilnog razvoja softvera]	1
28. Koliko je značajno da ceo arhitekturalni proces tokom razvoja bude praćen nekim od alata (Jira ili sl.)?	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [1. Identifikovanje ključnih stejkholdera sistema]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [2. Formiranje odgovarajućeg tima i izbor softver arhitekte u odnosu na problem koji se rešava]	0.8

29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [3. Razumevanje poslovnog problema]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [4. Identifikovanje granica projekta]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [5. Aktivne diskusije sa stejkholderima u cilju analize zahteva i razumevanja biznisa]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [6. Identifikovanje arhitekturalno značajnih zahteva]	1
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [7. Formiranje zajedničke liste prioritizovanih funkcionalnih i nefunkcionalnih zahteva]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [8. Analiza rizika, kako bi se identifikovale i izolovale oblasti kompleksnosti]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [9. Istraživanje tehnologije pogodne za implementaciju]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [10. Identifikacija i definisanje osnovnih struktura (modula) za glavni deo sistema i njihovih veza (envision arhitekture)]	1
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [11. Definisanje osnovne arhitekture podataka]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [12. Generisanje top level dokumentacije]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [13. Definisanje uputstava za kodiranje i drugih uputstava za dizajn sistema]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [14. Razmatranje i postavka modela raspoređivanja]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [15. Analiza zavisnosti fukcionalnih zahteva od arhitekturalnih elemenata u planiranju relesea]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [16. Upravljanje tehničkim dugom, sa fokusom na nefunkcionalne zahteve u svakoj iteraciji]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [17. Planiranje release-a sa strategijom razmatranja legacy sistema, zavisnosti od drugih partnerskih ili third party proizvoda i compatibilnost podataka]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [18. Upravljanje zavisnostima koji potiču od spoljnih sistema sa kojima sistem intereaguje tokom release-a]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [19. Validacija kritičnih arhitekturalnih zahteva i koncepata dizajna praksom razvoja prototipova]	1
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [20. Formalni pregled arhitekture]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [21. Plan testiranja]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [22. Definisanje detaljnog dizajna svakog modula]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [23. Dokumentovanje detaljnog dizajna]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [24. Razmatranje detaljnog dizajna i korekcije]	0.8

29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [25. Kreiranje/razmatranje unit testova i testova prihvatljivosti]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [26. Kreiranje/razmatranje QA testova]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [27. Testiranje performansi sistema ili dr. kritičnih nefunkcionalnih zahteva]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [28. Kreiranje/razmatranje testova integracije]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [29. Code review]	0.8
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [30. Kontinuirana podrška arhitekte kroz rešavanje ključnih pitanja dizajna tokom razvoja]	1
29. Ocenite značaj sledećih eksplicitnih arhitekturalnih aktivnosti u razvoju kompleksnih sistema, upotrebom agilnih procesa: [31. Upravljanje konfiguracijom]	1
30. Koliko je značajno eksplicitno identifikovati zahteve koji se odnose na geografsku dislociranost delova budućeg sistema?	0.8

Prilog 7: Pozivno pismo ekspertima za drugi krug istraživanja

Poštovani **<<ime i prezime eksperta>>,**

zahvaljujem Vam se na kvalitetnim i iscrpnim odgovorima koje ste pružili u prvom krugu istraživanja i što ste nastavili svoje učešće u ovom drugom krugu.

Drugi krug istraživanja zahteva da popunite upitnik, koji je drugačiji od upitnika u prvom krugu istraživanja, jer nema otvorenih pitanja, već je potrebno da odaberete neku od ponuđenih opcija za svako pitanje u upitniku. Upitnik je izrađen na osnovu analiziranih podataka tj. odgovora koje ste Vi i drugi eksperti dali tokom intervjuisanja u prvom krugu istraživanja.

U prilogu Vam, pored linka na upitnik, šaljem i kratke instrukcije za popunjavanje upitnika, kako bi što uspešnije odgovorili na sva pitanja upitnika.

Molim Vas da popunite sva pitanja upitnika, jer je to od izuzetne važnosti za dobijanje validnih rezultata, a time i postizanje ciljeva ovog istraživanja.

Unapred zahvalna na vremenu, volji i naporu koji ćete uložiti, dajući kvalitetne odgovore.

Srdačan pozdrav,
Mirjana Marić.

Prilog 8: Instrukcije za popunjavanje upitnika drugog kruga istraživanja

Drugi krug istraživanja Delfi metodom ima za cilj da, putem upitnika, sumira sve odgovore eksperata koji su intervjuisani u okviru prvog kruga istraživanja. Iz tog razloga, izradi upitnika prethodila je kvalitativna analiza svih sprovedenih intervjua, čiji su rezultati predstavljali osnovu za generisanje pitanja upitnika drugog kruga. Pitanja upitnika u skladu su sa postavljenim istraživačkim ciljevima:

C1. Identifikovati tradicionalne arhitekturalne prakse korisne u agilnom procesu razvoja kvalitetnih, kompleksnih softverskih rešenja.
C2. Identifikovati kritična mesta u agilnom procesu razvoja pogodna za primenu tradicionalnih arhitekturalnih praksi.

Uz svako pitanje upitnika stoji skala za ocenjivanje od 1-4, a od Vas se očekuje da označite jednu od mogućih varijanti odgovora:

1. nije značajno
2. delimično značajno
3. značajno
4. ekstremno značajno

Pojedina pitanja su sačinjena od više stavki tj. podpitanja, te je potrebno da svaku pojedinačnu stavku rangirate na skali od 1-4.

Molim Vas da popunite sva pitanja, kako bih mogla sprovesti validnu analizu podataka i dobiti kvalitetne rezultate istraživanja.

Prilog 9: Deo rezultata kvantitativne analize podataka drugog kruga istraživanja (primer dobijenih rezultata za 3 stavke 29. pitanja iz upitnika)

Bootstrap

Notes

Output Created		04-JUN-2015 08:26:00
Comments		
Input	Data	C:\Users\Maric\Documents\Mirjana Marić\PODACI ZA ANALIZU_II_KRUG\PODACI_2.sav
	Active Dataset	DataSet7
	Filter	<none>
	Weight	<none>
	Split File	<none>
Syntax		BOOTSTRAP /SAMPLING METHOD=SIMPLE /VARIABLES INPUT=P29.01 P29.02 P29.03 P29.04 P29.05 P29.06 P29.07 P29.08 P29.09 P29.10 P29.11 P29.12 P29.13 P29.14 P29.15 P29.16 P29.17 P29.18 P29.19 P29.20 P29.21 P29.22 P29.23 P29.24 P29.25 P29.26 P29.27 P29.28 P29.29 P29.30 P29.31 /CRITERIA CILEVEL=95 CITYPE=PERCENTILE NSAMPLES=1000 /MISSING USERMISSING=EXCLUDE.
Resources	Processor Time	00:00:00,08
	Elapsed Time	00:00:00,22

Bootstrap Specifications

Sampling Method	Simple
Number of Samples	1000
Confidence Interval Level	95,0%
Confidence Interval Type	Percentile

```
FREQUENCIES VARIABLES=P29.01 P29.02 P29.03 P29.04 P29.05 P29.06 P29.07 P29.08 P29.09
P29.10 P29.11 P29.12 P29.13 P29.14 P29.15 P29.16 P29.17 P29.18 P29.19 P29.20 P29.21 P29.22 P29.23
P29.24 P29.25 P29.26 P29.27 P29.28 P29.29 P29.30 P29.31
  /NTILES=4
  /STATISTICS=STDDEV RANGE MEAN MEDIAN
  /ORDER=ANALYSIS.
```

Frequencies

Notes

Output Created		04-JUN-2015 08:26:00
Comments		
Input	Data	C:\Users\Maric\Documents\Mirjana Marić\PODACI ZA ANALIZU_II_KRUG\PODACI_2.sav
	Active Dataset	DataSet7
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	12871
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics are based on all cases with valid data.
Syntax		FREQUENCIES VARIABLES=P29.01 P29.02 P29.03 P29.04 P29.05 P29.06 P29.07 P29.08 P29.09 P29.10 P29.11 P29.12 P29.13 P29.14 P29.15 P29.16 P29.17 P29.18 P29.19 P29.20 P29.21 P29.22 P29.23 P29.24 P29.25 P29.26 P29.27 P29.28 P29.29 P29.30 P29.31 /NTILES=4 /STATISTICS=STDDEV RANGE MEAN MEDIAN /ORDER=ANALYSIS.
Resources	Processor Time	00:00:34,34
	Elapsed Time	00:00:31,82

Statistics

			Statistic	Bootstrap[a]			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
N	Valid	P29.01	20	0	0	20	20
		P29.02	20	0	0	20	20
		P29.03	20	0	0	20	20
		P29.04	20	0	0	20	20
		P29.05	20	0	0	20	20
		P29.06	20	0	0	20	20
		P29.07	20	0	0	20	20
		P29.08	20	0	0	20	20
		P29.09	20	0	0	20	20
		P29.10	20	0	0	20	20
		P29.11	20	0	0	20	20
		P29.12	20	0	0	20	20
		P29.13	20	0	0	20	20
		P29.14	20	0	0	20	20
		P29.15	20	0	0	20	20
		P29.16	20	0	0	20	20
		P29.17	20	0	0	20	20
		P29.18	20	0	0	20	20
		P29.19	20	0	0	20	20
		P29.20	20	0	0	20	20
		P29.21	20	0	0	20	20
		P29.22	20	0	0	20	20
		P29.23	20	0	0	20	20
		P29.24	20	0	0	20	20
		P29.25	20	0	0	20	20
		P29.26	20	0	0	20	20
		P29.27	20	0	0	20	20
		P29.28	20	0	0	20	20
		P29.29	20	0	0	20	20
		P29.30	20	0	0	20	20
		P29.31	20	0	0	20	20

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
N	P29.01	0	0	0	0	0
	P29.02	0	0	0	0	0
	P29.03	0	0	0	0	0
	P29.04	0	0	0	0	0
	P29.05	0	0	0	0	0
	P29.06	0	0	0	0	0
	P29.07	0	0	0	0	0
	P29.08	0	0	0	0	0
	P29.09	0	0	0	0	0
	P29.10	0	0	0	0	0
	P29.11	0	0	0	0	0
	P29.12	0	0	0	0	0
	P29.13	0	0	0	0	0
	P29.14	0	0	0	0	0
	P29.15	0	0	0	0	0
Missing	P29.16	0	0	0	0	0
	P29.17	0	0	0	0	0
	P29.18	0	0	0	0	0
	P29.19	0	0	0	0	0
	P29.20	0	0	0	0	0
	P29.21	0	0	0	0	0
	P29.22	0	0	0	0	0
	P29.23	0	0	0	0	0
	P29.24	0	0	0	0	0
	P29.25	0	0	0	0	0
	P29.26	0	0	0	0	0
	P29.27	0	0	0	0	0
	P29.28	0	0	0	0	0
	P29.29	0	0	0	0	0
	P29.30	0	0	0	0	0
	P29.31	0	0	0	0	0

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
	P29.01	2,8000	-,0043	,1849	2,4500	3,1500
	P29.02	3,4000	-,0029	,1324	3,1500	3,6500
	P29.03	3,4500	-,0002	,1303	3,2000	3,7000
	P29.04	3,3000	,0024	,1863	2,9000	3,6500
	P29.05	3,3000	,0022	,1380	3,0013	3,5500
	P29.06	3,4500	-,0015	,1477	3,1500	3,7000
	P29.07	3,1500	-,0015	,1713	2,8000	3,5000
	P29.08	3,2500	,0031	,1410	2,9500	3,5487
	P29.09	3,1500	-,0029	,1230	2,9000	3,4000
	P29.10	3,55	,00	,15	3,25	3,85
	P29.11	3,0000	-,0035	,1395	2,7000	3,2500
	P29.12	2,6500	-,0061	,2095	2,2013	3,0500
	P29.13	2,6000	-,0025	,1756	2,2500	2,9500
	P29.14	2,8500	,0003	,1786	2,5000	3,2000
	P29.15	2,9500	,0022	,1530	2,6500	3,2500
Mean	P29.16	2,5500	,0027	,2026	2,1500	3,0000
	P29.17	2,7500	,0001	,1917	2,3500	3,1000
	P29.18	2,6000	,0008	,1626	2,3000	2,9000
	P29.19	3,1000	-,0049	,1504	2,8000	3,4000
	P29.20	2,80	,00	,15	2,50	3,10
	P29.21	2,8000	-,0030	,2072	2,4000	3,2000
	P29.22	2,2500	-,0061	,1838	1,9000	2,6000
	P29.23	2,1500	-,0073	,2039	1,7500	2,5500
	P29.24	2,2500	-,0050	,2318	1,8000	2,7487
	P29.25	2,9000	-,0026	,1915	2,5013	3,2500
	P29.26	2,7500	,0005	,1890	2,3500	3,1000
	P29.27	3,2000	-,0049	,1636	2,8500	3,5000
	P29.28	2,9500	-,0024	,1666	2,6000	3,2500
	P29.29	3,2000	-,0030	,2035	2,8000	3,6000
	P29.30	3,40	,00	,19	3,00	3,75
	P29.31	3,0000	-,0043	,1903	2,6000	3,3500

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Median	P29.01	3,0000	-,0115	,0929	3,0000	3,0000
	P29.02	3,0000	,3185	,4219	3,0000	4,0000
	P29.03	3,5000	,0065	,4466	3,0000	4,0000
	P29.04	3,5000	-,0025	,4544	3,0000	4,0000
	P29.05	3,0000	,1705	,3381	3,0000	4,0000
	P29.06	4,0000	-,3460	,4324	3,0000	4,0000
	P29.07	3,0000	,0855	,2569	3,0000	4,0000
	P29.08	3,0000	,0875	,2533	3,0000	4,0000
	P29.09	3,0000	,0075	,0721	3,0000	3,0000
	P29.10	4,00	-,09	,26	3,00	4,00
	P29.11	3,0000	-,0015	,0474	3,0000	3,0000
	P29.12	3,0000	-,3385	,4334	2,0000	3,0000
	P29.13	3,0000	-,1740	,3374	2,0000	3,0000
	P29.14	3,0000	,0000	,0000	3,0000	3,0000
	P29.15	3,0000	,0010	,0223	3,0000	3,0000
	P29.16	2,5000	-,0080	,4490	2,0000	3,0000
	P29.17	3,0000	-,0270	,1442	2,5000	3,0000
	P29.18	3,0000	-,3315	,4178	2,0000	3,0000
	P29.19	3,0000	,0265	,1535	3,0000	3,5000
	P29.20	3,00	-,01	,08	3,00	3,00
	P29.21	3,0000	-,0090	,0996	3,0000	3,0000
	P29.22	2,0000	,1820	,3486	2,0000	3,0000
	P29.23	2,0000	,1430	,4034	1,5000	3,0000
	P29.24	2,0000	,1545	,4136	1,5000	3,0000
	P29.25	3,0000	-,0275	,1747	2,5000	3,0000
	P29.26	3,0000	-,0375	,1654	2,5000	3,0000
	P29.27	3,0000	,0805	,2384	3,0000	4,0000
	P29.28	3,0000	-,0020	,0447	3,0000	3,0000
	P29.29	3,0000	,3260	,4236	3,0000	4,0000
	P29.30	4,00	-,20	,37	3,00	4,00
	P29.31	3,0000	,0095	,0785	3,0000	3,0000

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Std. Deviation	P29.01	,83351	-,03563	,13969	,48936	1,05006
	P29.02	,59824	-,02164	,07504	,47016	,74484
	P29.03	,60481	-,02060	,07889	,47016	,75394
	P29.04	,86450	-,03973	,16071	,51042	1,12390
	P29.05	,65695	-,02848	,08800	,47016	,79472
	P29.06	,68633	-,02926	,09795	,47016	,83335
	P29.07	,74516	-,02527	,08134	,55012	,87509
	P29.08	,63867	-,01782	,08333	,47016	,76777
	P29.09	,58714	-,02177	,08835	,39403	,72548
	P29.10	,686	-,028	,119	,410	,865
	P29.11	,64889	-,02141	,08860	,44721	,78807
	P29.12	,93330	-,03640	,11770	,67082	1,12834
	P29.13	,82078	-,02988	,12244	,55012	1,03976
	P29.14	,81273	-,03357	,15637	,45883	1,05631
	P29.15	,68633	-,03841	,15301	,36635	,94451
	P29.16	,88704	-,02860	,11223	,64114	1,08921
	P29.17	,85070	-,03234	,12968	,55251	1,06992
	P29.18	,75394	-,02422	,10757	,51042	,94032
	P29.19	,71818	-,02209	,08242	,52315	,85220
	P29.20	,696	-,034	,129	,394	,918
	P29.21	,95145	-,03721	,14570	,60481	1,16416
	P29.22	,85070	-,02648	,11007	,60481	1,03999
	P29.23	,93330	-,02598	,10252	,69585	1,10501
	P29.24	1,06992	-,03075	,11925	,78824	1,26803
	P29.25	,85224	-,03274	,10931	,60481	1,02084
	P29.26	,85070	-,03916	,13284	,52315	1,05006
	P29.27	,76777	-,03597	,15309	,47016	1,02084
	P29.28	,75915	-,03430	,13674	,45883	,96791
	P29.29	,95145	-,03622	,17035	,51042	1,20961
	P29.30	,883	-,041	,171	,489	1,165
	P29.31	,85840	-,03532	,16242	,47016	1,11803

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
	P29.01	3,00				
	P29.02	2,00				
	P29.03	2,00				
	P29.04	3,00				
	P29.05	2,00				
	P29.06	2,00				
	P29.07	2,00				
	P29.08	2,00				
	P29.09	2,00				
	P29.10	2				
	P29.11	2,00				
	P29.12	3,00				
	P29.13	3,00				
	P29.14	3,00				
	P29.15	3,00				
Range	P29.16	3,00				
	P29.17	3,00				
	P29.18	3,00				
	P29.19	2,00				
	P29.20	3				
	P29.21	3,00				
	P29.22	3,00				
	P29.23	3,00				
	P29.24	3,00				
	P29.25	3,00				
	P29.26	3,00				
	P29.27	3,00				
	P29.28	3,00				
	P29.29	3,00				
	P29.30	3				
	P29.31	3,00				

			Statistic	Bootstrap[a]			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
		P29.01	2,2500	,1928	,5215	1,2500	3,0000
		P29.02	3,0000	,0005	,0488	3,0000	3,0000
		P29.03	3,0000	,0063	,0937	3,0000	3,0000
		P29.04	3,0000	-,1352	,3430	2,0000	3,0000
		P29.05	3,0000	-,0395	,1770	2,2500	3,0000
		P29.06	3,0000	-,0052	,2042	2,2500	3,2500
		P29.07	3,0000	-,3307	,4323	2,0000	3,0000
		P29.08	3,0000	-,0393	,1787	2,2500	3,0000
		P29.09	3,0000	-,0275	,1480	2,2500	3,0000
		P29.10	3,00	,11	,39	2,25	4,00
		P29.11	3,0000	-,3390	,4322	2,0000	3,0000
		P29.12	2,0000	-,0032	,2382	1,2500	2,2500
		P29.13	2,0000	,0248	,2782	1,2500	3,0000
		P29.14	3,0000	-,3605	,5148	1,2500	3,0000
		P29.15	3,0000	-,1450	,3286	2,0000	3,0000
Percentiles	25	P29.16	2,0000	-,0223	,2000	1,2500	2,0000
		P29.17	2,0000	,2485	,4697	1,2500	3,0000
		P29.18	2,0000	,0238	,1428	2,0000	2,2500
		P29.19	3,0000	-,3422	,4362	2,0000	3,0000
		P29.20	2,25	,22	,46	2,00	3,00
		P29.21	2,2500	,0737	,6743	1,0000	3,0000
		P29.22	2,0000	-,3465	,4396	1,0000	2,0000
		P29.23	1,0000	,2640	,4024	1,0000	2,0000
		P29.24	1,0000	,2637	,4019	1,0000	2,0000
		P29.25	2,0000	,2967	,4193	2,0000	3,0000
		P29.26	2,0000	,2645	,4668	1,2500	3,0000
		P29.27	3,0000	-,0332	,1662	2,2500	3,0000
		P29.28	3,0000	-,3347	,4353	2,0000	3,0000
		P29.29	3,0000	-,1758	,4254	1,2500	3,0000
		P29.30	3,00	-,08	,41	2,00	4,00
		P29.31	3,0000	-,1870	,4247	1,5000	3,0000

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Percentiles	P29.01	3,0000	-,0115	,0929	3,0000	3,0000
	P29.02	3,0000	,3185	,4219	3,0000	4,0000
	P29.03	3,5000	,0065	,4466	3,0000	4,0000
	P29.04	3,5000	-,0025	,4544	3,0000	4,0000
	P29.05	3,0000	,1705	,3381	3,0000	4,0000
	P29.06	4,0000	-,3460	,4324	3,0000	4,0000
	P29.07	3,0000	,0855	,2569	3,0000	4,0000
	P29.08	3,0000	,0875	,2533	3,0000	4,0000
	P29.09	3,0000	,0075	,0721	3,0000	3,0000
	P29.10	4,00	-,09	,26	3,00	4,00
	P29.11	3,0000	-,0015	,0474	3,0000	3,0000
	P29.12	3,0000	-,3385	,4334	2,0000	3,0000
	P29.13	3,0000	-,1740	,3374	2,0000	3,0000
	P29.14	3,0000	,0000	,0000	3,0000	3,0000
	P29.15	3,0000	,0010	,0223	3,0000	3,0000
50	P29.16	2,5000	-,0080	,4490	2,0000	3,0000
	P29.17	3,0000	-,0270	,1442	2,5000	3,0000
	P29.18	3,0000	-,3315	,4178	2,0000	3,0000
	P29.19	3,0000	,0265	,1535	3,0000	3,5000
	P29.20	3,00	-,01	,08	3,00	3,00
	P29.21	3,0000	-,0090	,0996	3,0000	3,0000
	P29.22	2,0000	,1820	,3486	2,0000	3,0000
	P29.23	2,0000	,1430	,4034	1,5000	3,0000
	P29.24	2,0000	,1545	,4136	1,5000	3,0000
	P29.25	3,0000	-,0275	,1747	2,5000	3,0000
	P29.26	3,0000	-,0375	,1654	2,5000	3,0000
	P29.27	3,0000	,0805	,2384	3,0000	4,0000
	P29.28	3,0000	-,0020	,0447	3,0000	3,0000
	P29.29	3,0000	,3260	,4236	3,0000	4,0000
	P29.30	4,00	-,20	,37	3,00	4,00
	P29.31	3,0000	,0095	,0785	3,0000	3,0000

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Percentiles	P29.01	3,0000	,1263	,3067	3,0000	4,0000
	P29.02	4,0000	-,0292	,1387	3,7500	4,0000
	P29.03	4,0000	-,0110	,0845	3,7500	4,0000
	P29.04	4,0000	-,0057	,0605	4,0000	4,0000
	P29.05	4,0000	-,0650	,2158	3,0000	4,0000
	P29.06	4,0000	-,0040	,0499	4,0000	4,0000
	P29.07	4,0000	-,1583	,3303	3,0000	4,0000
	P29.08	4,0000	-,1303	,3011	3,0000	4,0000
	P29.09	3,7500	-,2393	,4641	3,0000	4,0000
	P29.10	4,00	,00	,00	4,00	4,00
	P29.11	3,0000	,3132	,4289	3,0000	4,0000
	P29.12	3,0000	,3010	,4334	3,0000	4,0000
	P29.13	3,0000	,0305	,1556	3,0000	3,7500
	P29.14	3,0000	,1360	,3168	3,0000	4,0000
	P29.15	3,0000	,1418	,3207	3,0000	4,0000
75	P29.16	3,0000	,1385	,3519	3,0000	4,0000
	P29.17	3,0000	,1538	,3347	3,0000	4,0000
	P29.18	3,0000	,0290	,1656	3,0000	3,7500
	P29.19	4,0000	-,2815	,4047	3,0000	4,0000
	P29.20	3,00	,03	,16	3,00	3,75
	P29.21	3,0000	,3090	,4277	3,0000	4,0000
	P29.22	3,0000	-,0642	,2224	2,0000	3,0000
	P29.23	3,0000	-,0748	,2396	2,0000	3,0000
	P29.24	3,0000	,0722	,4164	2,0000	4,0000
	P29.25	3,7500	-,2378	,4640	3,0000	4,0000
	P29.26	3,0000	,1275	,3049	3,0000	4,0000
	P29.27	4,0000	-,1620	,3356	3,0000	4,0000
	P29.28	3,0000	,3248	,4342	3,0000	4,0000
	P29.29	4,0000	-,0237	,1230	3,7500	4,0000
	P29.30	4,00	,00	,01	4,00	4,00
	P29.31	3,7500	-,2255	,4610	3,0000	4,0000

Frequency Table

P29.01

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent[a]	
						Bias	Std. Error
Valid	1,00	2	10,0	10,0	10,0	,0	6,6
	2,00	3	15,0	15,0	25,0	,1	8,4
	3,00	12	60,0	60,0	85,0	,2	11,1
	4,00	3	15,0	15,0	100,0	-,3	7,8
	Total	20	100,0	100,0		,0	,0

P29.01

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	1,00	,0	25,0
	2,00	,0	35,0
	3,00	40,0	80,0
	4,00	,0	30,0
	Total	100,0	100,0

P29.02

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent[a]	
						Bias	Std. Error
Valid	2,00	1	5,0	5,0	5,0	,0	5,0
	3,00	10	50,0	50,0	55,0	,4	10,8
	4,00	9	45,0	45,0	100,0	-,3	11,0
	Total	20	100,0	100,0		,0	,0

P29.02

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	2,00	,0	15,0
	3,00	30,0	70,0
	4,00	25,0	65,0
	Total	100,0	100,0

P29.03

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent[a]	
						Bias	Std. Error
Valid	2,00	1	5,0	5,0	5,0	,0	4,9
	3,00	9	45,0	45,0	50,0	,1	11,5
	4,00	10	50,0	50,0	100,0	,0	11,3
	Total	20	100,0	100,0		,0	,0

P29.03

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	2,00	,0	15,0
	3,00	25,0	69,9
	4,00	25,0	70,0
	Total	100,0	100,0

Prilog 10: Pozivno pismo ekspertima za učešće u završnom krugu istraživanja

Poštovani **<<ime i prezime eksperta>>**,

obaveštavam Vas da predstoji treći, poslednji krug istraživanja i tim povodom Vam u prilogu mejla šaljem fajl sa instrukcijama kako da popunite upitnik, kao i sam upitnik.

Molim Vas da pažljivo pročitate instrukcije za popunjavanje upitnika. Upitnik sadrži samo ona pitanja/tvrđenja iz upitnika prethodnog kruga, za koja nije postignut potreban nivo saglasnosti (70%) svih eksperata o nivou njihove značajnosti.

Izražavam veliku zahvalnost na Vašoj posvećenosti i doprinosu koji ste dali u ovom istraživanju, kao i na vremenu koje ste izdvojili. Moja obaveza je da Vam, nakon završetka analiza podataka, predočim do kakvih sam rezultata došla.

Bila bih Vam zahvalna da upitnik popunite do 24.6 i da mi isti pošaljete mejlom. Planirano vreme za popunjavanje upitnika je oko 15 minuta.

Srdačan pozdrav,
Mirjana Marić.

Prilog 11: Instrukcije za popunjavanje upitnika u završnom krugu istraživanja

Poštovani **<<ime i prezime eksperta>>**,

ovo je treći, ujedno završni krug istraživanja Delfi metodom i podrazumeva ponovno davanje odgovora na pitanja/tvrdnje iz upitnika prethodnog kruga istraživanja, za koja nije postignut potreban nivo saglasnosti (od 70%) eksperata, po pitanju njihove značajnosti.

Za svako pitanje, upitnika trećeg kruga istraživanja, predviđene su 4 kolone:

Prva kolona pokazuje Vaš odgovor o značajnosti navedenog tvrđenja, koji ste dali u prethodnom krugu istraživanja (npr. nije značajno/delimično značajno/značajno/ekstremno značajno).

Druga kolona sadrži izračunatu medijanu tj. odgovor grupe eksperata za dato tvrđenje (npr. nije značajno/delimično značajno/značajno/ekstremno značajno).

Treća kolona je prazna i u nju unosite Vaš odgovor za ovaj krug istraživanja (npr. nije značajno/delimično značajno/značajno/ekstremno značajno). Stim da u ovom krugu istraživanja možete da uporedite Vaš odgovor iz prošlog kruga istraživanj,a sa odgovorom grupe u prošlom krugu istraživanja i da preispitate svoj prvobitni odgovor i eventualno ga promenite u skladu sa odgovorom grupe.

Četvrta kolona predviđena je za argumentovano obrazlaganje Vašeg konačnog odgovora (odgovora u trećem krugu istraživanja).

Cilj ovog kruga istraživanja je, dakle, da preispitate svoj originalni odgovor, u odnosu na odgovor grupe i promenite ga ukoliko mislite da ima smisla. Naravno, imate potpuno pravo da ne promenite svoj prvobitni odgovor, ali Vas molim da u kolonu pored odgovora date i kratko obrazloženje zašto ostajete pri svom prvobitnom odgovoru.

Srdačan pozdrav,
Mirjana Marić.

Prilog 12: Obrasci po kojima su se sprovodile procedure za kvantitativnu analizu podataka

1. Bootstrapping

Greška (bias) nekog statistika T je ocenjena pomoću sledećeg obrasca:

$$Bias(T) = B^{-1}\sum_{b=1}^{B} T_b^* - T$$

Standardna greška statistika T je ocenjena na osnovu standardne devijacije vrednosti dobivenih ponavljanim uzorkovanjem pomoću sledećeg obrasca:

$$SE \approx \sqrt{\frac{1}{B-1}\sum_{b-1}^{B}\left(T_b^* - B^{-1}\sum_{b=1}^{B} T_b^*\right)^2}$$

gde je:

B – broj uzoraka u ponavljanom uzorkovanju

T – statistik za bootstrap

T^*_b *b-ta* – bootstrap kopija statistika*T*

2. Ocenjivanje saglasnosti ocenjivača - Cohen-ov kappa koeficijent

Cohen-ov pokazatelj saglasnosti, kappa (*k*), za kvadratnu tabelu (*R*=*C*):

$$k = \frac{W\sum_{i=1}^{R} f_{ii} - \sum_{i=1}^{R} r_i c_i}{W^2 - \sum_{o=1}^{R} r_i\, c_i}$$

Opšte notacije:

f_{ij} - zbir za slučajeve u ćeliji (*i,j*)

$$r_i = \sum_{j=1}^{C} f_{ij} \qquad c_j = \sum_{i=1}^{R} f_{ij} \qquad W = \sum_{j=1}^{C} c_j \sum_{i=1}^{R} r_i$$

3. Chaffin-Talley-ev indeks individualne stabilnosti između dve uzastopne Delfi iteracije

Indeks individualne stabilnosti između dve uzastopne Delfi iteracije:

$$I = \frac{\sum_j \max_k O_{jk} - \max_k O_{.k}}{n - \max_k O_{.k}}$$

gde je:

O_{jk} - učestalost respondenata koji su glasali za j-ti interval u i-toj iteraciji, ali su glasali za k-ti interval u iteraciji $i + 1$;

$\max_k O_{jk}$ - najveća učestalost za *j*-ti interval u *i*-toj iteraciji;

$\max_k O_{.k}$ - najveća učestalost između *k*-tih intervala u iteraciji $i + 1$;

n - ukupne opažene učestalosti u tabeli kontingencije.

4. McNemar i McNemar-Bowker-ov test

Za dva zavisna uzorka iz kategorijskog polja sa dve vrednosti, McNemar-ov test proverava hipotezu:

H_0: dva uzorka imaju istu marginalnu distribuciju.

Neka je $n_{1,f}$ broj zapisa u kojima je x_i uspeh, a y_i neuspeh i $n_{2,f}$ - broj zapisa u kojima je x_i neuspeh, a y_i uspeh, uključujući pondere učestalosti.

Ako $n_{1,f,}+n_{2,f} \leq 25$, dvostrana egzaktna verovatnoća je:

$$p = 2\sum_{i=0}^{r} \binom{n_{1,f} + n_{2,f}}{i} 0{,}5^{n_{1,f}+n_{2,f}}$$

gde je:

$r - \min(n_{1,f}, n_{2,f})$

Ako je $n_{1,f}+ n_{2,f} > 25$, test statistik je:

$$\xi_c^2 = \frac{\left(|n_{1,f} - n_{2,f}| - 1\right)^2}{n_{1,f} + n_{2,f}}$$

Za dva zavisna uzorka iz kategorijskog polja sa više od dve vrednosti (*R*=*C* kvadratna tabela) McNemar-Bowkerov test proverava hipotezu:

H_0: $pij = pji$ za sve $(i<j)$

H_1: $pij \neq pji$ za najmanje jedan par (i,j)

McNemar-Bowkerov test statistik je određen obrascem:

$$\xi^2 = \sum_{i<j} \frac{I\left(n_{ij} + n_{ji} > 0\right)\left(n_{ij} - n_{ji}\right)^2}{n_{ij} + n_{ji}}$$

gde je I () funkcija indikatora.

Prilog 13: Deo rezultata kvantitativne analize podataka drugog kruga istraživanja (Cohen-ov kappa koeficijent)

Crosstabs

Notes

Output Created		04-JUN-2015 13:24:31
Comments		
Input	Data	C:\Users\Maric\Documents\PODACI_10_dihot_transp_P29.sav
	Active Dataset	DataSet1
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	31
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics for each table are based on all the cases with valid data in the specified range(s) for all variables in each table.
Syntax		CROSSTABS /TABLES=IP17 BY IP1 IP15 IP14 IP11 IP9 IP12 IP3 IP16 IP20 IP4 IP5 IP18 IP6 IP7 IP19 IP2 IP10 IP8 IP13 /FORMAT=AVALUE TABLES /STATISTICS=CHISQ PHI KAPPA /CELLS=COUNT /COUNT ROUND CELL.
Resources	Processor Time	00:00:00,13
	Elapsed Time	00:00:00,13
	Dimensions Requested	2
	Cells Available	174762

Case Processing Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
IP17 * IP1	31	100,0%	0	0,0%	31	100,0%
IP17 * IP15	31	100,0%	0	0,0%	31	100,0%
IP17 * IP14	31	100,0%	0	0,0%	31	100,0%
IP17 * IP11	31	100,0%	0	0,0%	31	100,0%
IP17 * IP9	31	100,0%	0	0,0%	31	100,0%
IP17 * IP12	31	100,0%	0	0,0%	31	100,0%
IP17 * IP3	31	100,0%	0	0,0%	31	100,0%
IP17 * IP16	31	100,0%	0	0,0%	31	100,0%
IP17 * IP20	31	100,0%	0	0,0%	31	100,0%
IP17 * IP4	31	100,0%	0	0,0%	31	100,0%
IP17 * IP5	31	100,0%	0	0,0%	31	100,0%
IP17 * IP18	31	100,0%	0	0,0%	31	100,0%
IP17 * IP6	31	100,0%	0	0,0%	31	100,0%
IP17 * IP7	31	100,0%	0	0,0%	31	100,0%
IP17 * IP19	31	100,0%	0	0,0%	31	100,0%
IP17 * IP2	31	100,0%	0	0,0%	31	100,0%
IP17 * IP10	31	100,0%	0	0,0%	31	100,0%
IP17 * IP8	31	100,0%	0	0,0%	31	100,0%
IP17 * IP13	31	100,0%	0	0,0%	31	100,0%

IP17 * IP1

Crosstab

Count

		IP1		Total
		,00	1,00	
IP17	,00	0	5	5
	1,00	4	22	26
Total		4	27	31

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)	Exact Sig. (1-sided)
Pearson Chi-Square	,883[a]	1	,347		
Continuity Correction[b]	,045	1	,833		
Likelihood Ratio	1,517	1	,218		
Fisher's Exact Test				1,000	,475
Linear-by-Linear Association	,855	1	,355		
N of Valid Cases	31				

a. 3 cells (75,0%) have expected count less than 5. The minimum expected count is ,65.

b. Computed only for a 2x2 table

Symmetric Measures

		Value	Asymp. Std. Error[a]	Approx. T[b]	Approx. Sig.
Nominal by Nominal	Phi	-,169			,347
	Cramer's V	,169			,347
Measure of Agreement	Kappa	-,167	,057	-,940	,347
N of Valid Cases		31			

a. Not assuming the null hypothesis.

b. Using the asymptotic standard error assuming the null hypothesis.

IP17 * IP15

Crosstab

Count

		IP15		Total
		,00	1,00	
IP17	,00	1	4	5
	1,00	5	21	26
Total		6	25	31

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)	Exact Sig. (1-sided)
Pearson Chi-Square	,002[a]	1	,968		
Continuity Correction[b]	,000	1	1,000		
Likelihood Ratio	,002	1	,968		
Fisher's Exact Test				1,000	,687
Linear-by-Linear Association	,002	1	,969		
N of Valid Cases	31				

a. 2 cells (50,0%) have expected count less than 5. The minimum expected count is ,97.

b. Computed only for a 2x2 table

Symmetric Measures

		Value	Asymp. Std. Error[a]	Approx. T[b]	Approx. Sig.
Nominal by Nominal	Phi	,007			,968
	Cramer's V	,007			,968
Measure of Agreement	Kappa	,007	,180	,040	,968
N of Valid Cases		31			

a. Not assuming the null hypothesis.

b. Using the asymptotic standard error assuming the null hypothesis.

Prilog 14: Deo rezultata kvantitativne analize podataka trećeg kruga istraživanja

Bootstrap

Notes

Output Created		04-JUL-2015 18:47:03
Comments		
Input	Data	C:\Users\Maric\Documents\Mirjana Marić\TREĆI KRUG\PODACI_ZA_III_KRUG\PODACI_IIIkrug_TRASP_T.sav
	Active Dataset	DataSet6
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	20
Syntax		BOOTSTRAP /SAMPLING METHOD=SIMPLE /VARIABLES INPUT=V11 V12 V15 V16 V21.03 V21.06 V22 V23 V26.01 V26.03 V26.05 V26.06 V26.11 V26.14 V26.15 V26.16 V26.17 V27.03 V27.04 V27.07 V28 V29.12 V29.13 V29.16 V29.18 V29.22 V29.23 V29.24 /CRITERIA CILEVEL=95 CITYPE=PERCENTILE NSAMPLES=1000 /MISSING USERMISSING=EXCLUDE.
Resources	Processor Time	00:00:00,06
	Elapsed Time	00:00:00,06

Bootstrap Specifications

Sampling Method	Simple
Number of Samples	1000
Confidence Interval Level	95,0%
Confidence Interval Type	Percentile

```
FREQUENCIES VARIABLES=V11 V12 V15 V16 V21.03 V21.06 V22 V23 V26.01 V26.03 V26.05 V26.06
V26.11 V26.14 V26.15 V26.16 V26.17 V27.03 V27.04 V27.07 V28 V29.12 V29.13 V29.16 V29.18
V29.22 V29.23 V29.24
  /NTILES=4
  /STATISTICS=STDDEV MEAN MEDIAN
  /ORDER=ANALYSIS.
```

Frequencies

Output Created		04-JUL-2015 18:47:03
Comments		
Input	Data	C:\Users\Maric\Documents\Mirjana Marić\TREĆI KRUG\PODACI_ZA_III_KRUG\PODACI_IIIkrug_TRASP_T.sav
	Active Dataset	DataSet6
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	12844
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics are based on all cases with valid data.
Syntax		FREQUENCIES VARIABLES=V11 V12 V15 V16 V21.03 V21.06 V22 V23 V26.01 V26.03 V26.05 V26.06 V26.11 V26.14 V26.15 V26.16 V26.17 V27.03 V27.04 V27.07 V28 V29.12 V29.13 V29.16 V29.18 V29.22 V29.23 V29.24 /NTILES=4 /STATISTICS=STDDEV MEAN MEDIAN /ORDER=ANALYSIS.
Resources	Processor Time	00:00:40,14
	Elapsed Time	00:00:39,88

Statistics

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
N	Valid	V11	20	0	0	20	20
		V12	20	0	0	20	20
		V15	20	0	0	20	20
		V16	20	0	0	20	20
		V21.03	20	0	0	20	20
		V21.06	20	0	0	20	20
		V22	20	0	0	20	20
		V23	20	0	0	20	20
		V26.01	20	0	0	20	20
		V26.03	20	0	0	20	20
		V26.05	20	0	0	20	20
		V26.06	20	0	0	20	20
		V26.11	20	0	0	20	20
		V26.14	20	0	0	20	20
		V26.15	20	0	0	20	20
		V26.16	20	0	0	20	20
		V26.17	20	0	0	20	20
		V27.03	20	0	0	20	20
		V27.04	20	0	0	20	20
		V27.07	20	0	0	20	20
		V28	20	0	0	20	20
		V29.12	20	0	0	20	20
		V29.13	20	0	0	20	20
		V29.16	20	0	0	20	20
		V29.18	20	0	0	20	20
		V29.22	20	0	0	20	20
		V29.23	20	0	0	20	20
		V29.24	20	0	0	20	20
	Missing	V11	0	0	0	0	0
		V12	0	0	0	0	0
		V15	0	0	0	0	0
			Statistic	Bootstrap[a]			

				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
N	Missing	V16	0	0	0	0	0
		V21.03	0	0	0	0	0
		V21.06	0	0	0	0	0
		V22	0	0	0	0	0
		V23	0	0	0	0	0
		V26.01	0	0	0	0	0
		V26.03	0	0	0	0	0
		V26.05	0	0	0	0	0
		V26.06	0	0	0	0	0
		V26.11	0	0	0	0	0
		V26.14	0	0	0	0	0
		V26.15	0	0	0	0	0
		V26.16	0	0	0	0	0
		V26.17	0	0	0	0	0
		V27.03	0	0	0	0	0
		V27.04	0	0	0	0	0
		V27.07	0	0	0	0	0
		V28	0	0	0	0	0
		V29.12	0	0	0	0	0
		V29.13	0	0	0	0	0
		V29.16	0	0	0	0	0
		V29.18	0	0	0	0	0
		V29.22	0	0	0	0	0
		V29.23	0	0	0	0	0
		V29.24	0	0	0	0	0
		V11	3,1500	,0048	,1457	2,8500	3,4500
		V12	1,9500	-,0030	,1914	1,5500	2,3500
		V15	3,1000	,0025	,1178	2,9000	3,3500
Mean		V16	2,8000	-,0063	,2239	2,3500	3,2500
		V21.03	3,0000	,0023	,1457	2,7000	3,2500
		V21.06	2,2000	-,0033	,1326	1,9500	2,4500

		Statistic	Bootstrap[a]			
			Bias	Std. Error	95% Confidence Interval	
					Lower	Upper
Median	V26.03	3,0000	,0350	,1591	3,0000	3,5000
	V26.05	2,0000	,0000	,0224	2,0000	2,0000
	V26.06	3,0000	,0000	,0000	3,0000	3,0000
	V26.11	2,0000	-,0795	,2470	1,0000	2,0000
	V26.14	2,0000	,0000	,0000	2,0000	2,0000
	V26.15	3,0000	,0000	,0000	3,0000	3,0000
	V26.16	3,0000	-,0005	,0158	3,0000	3,0000
	V26.17	2,0000	-,0050	,0836	2,0000	2,0000
	V27.03	3,0000	-,0105	,0874	3,0000	3,0000
	V27.04	3,0000	,0060	,0705	3,0000	3,0000
	V27.07	3,0000	-,0080	,0771	3,0000	3,0000
	V28	3,0000	-,0295	,1685	2,5000	3,0000
	V29.12	3,0000	,0010	,0223	3,0000	3,0000
	V29.13	3,0000	,0000	,0000	3,0000	3,0000
	V29.16	3,0000	-,1855	,3492	2,0000	3,0000
	V29.18	3,0000	,0000	,0000	3,0000	3,0000
	V29.22	2,0000	-,0015	,0274	2,0000	2,0000
	V29.23	2,0000	-,0360	,1665	1,5000	2,0000
	V29.24	2,0000	-,0360	,1665	1,5000	2,0000
	V11	,67082	-,02161	,09016	,45883	,80131
	V12	,88704	-,02826	,12633	,59824	1,11770
	V15	,55251	-,02225	,10081	,30822	,71818
Std. Deviation	V16	1,00525	-,03821	,16237	,60481	1,23085
	V21.03	,64889	-,03835	,16955	,22361	,93330
	V21.06	,61559	-,02477	,08747	,41039	,75880
	V22	,55251	-,02510	,09778	,30822	,71635

		V23	,56195	-,02241	,09740	,32444	,71818
		V26.01	,60481	-,04985	,17547	,22361	,85070
		V26.03	,78807	-,03584	,14782	,47016	1,03999
		V26.05	,60481	-,02178	,09601	,39403	,75915
		V26.06	,67082	-,04929	,21316	,00000	,94451
			Statistic	Bootstrap[a]			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
Std. Deviation		V26.11	,69585	-,02606	,08569	,50262	,82558
		V26.14	,45883	-,02674	,11729	,22361	,64072
		V26.15	,69585	-,04632	,18519	,22361	,94451
		V26.16	,58714	-,03633	,16871	,22361	,86419
		V26.17	,68633	-,02600	,08791	,48936	,82558
		V27.03	,69585	-,03063	,14023	,39403	,92338
		V27.04	,85840	-,04038	,17007	,44721	1,11803
		V27.07	,78640	-,03764	,14567	,44434	,99868
		V28	,89443	-,04077	,13081	,58714	1,09529
		V29.12	,72548	-,03165	,14873	,41039	,98809
		V29.13	,56195	-,05172	,20818	,00000	,87509
		V29.16	,60481	-,02177	,09343	,41039	,76089
		V29.18	,60481	-,04630	,18011	,22361	,89428
		V29.22	,60481	-,02370	,09866	,39403	,75915
		V29.23	,61559	-,02145	,08754	,41039	,75915
		V29.24	,78807	-,03596	,15273	,48936	1,05006
		V11	3,0000	-,1250	,3037	2,0000	3,0000
		V12	1,0000	,1437	,3188	1,0000	2,0000
		V15	3,0000	-,0415	,1829	2,2500	3,0000
		V16	3,0000	-,6815	,8773	1,0000	3,0000
		V21.03	3,0000	-,0348	,1721	2,2500	3,0000
		V21.06	2,0000	-,0275	,1505	1,2500	2,0000
		V22	2,0000	-,0357	,1683	1,2500	2,0000
Percentiles	25	V23	3,0000	-,1442	,3240	2,0000	3,0000
		V26.01	3,0000	-,0252	,1425	2,2500	3,0000
		V26.03	3,0000	-,1360	,3215	2,0000	3,0000
		V26.05	2,0000	-,1552	,3357	1,0000	2,0000
		V26.06	3,0000	-,0815	,3588	1,5000	3,0000
		V26.11	1,0000	,1557	,3260	1,0000	2,0000
		V26.14	2,0000	-,0408	,1794	1,2500	2,0000
		V26.15	3,0000	-,1920	,4331	1,2500	3,0000
			Statistic	**Bootstrap[a]**			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
Percentiles	25	V26.16	3,0000	-,1560	,3403	2,0000	3,0000
		V26.17	1,2500	,2177	,4619	1,0000	2,0000
		V27.03	2,2500	,2137	,4654	2,0000	3,0000
		V27.04	3,0000	-,1820	,4284	1,2500	3,0000
		V27.07	2,2500	,1783	,5200	1,2500	3,0000
		V28	2,0000	,2695	,4559	1,2500	3,0000
		V29.12	3,0000	-,1360	,3215	2,0000	3,0000
		V29.13	3,0000	-,0035	,0790	3,0000	3,0000
		V29.16	2,0000	,0665	,2242	2,0000	3,0000
		V29.18	3,0000	-,0355	,1747	2,2500	3,0000
		V29.22	2,0000	-,3280	,4323	1,0000	2,0000
		V29.23	1,0000	,3095	,4258	1,0000	2,0000
		V29.24	1,0000	,3095	,4258	1,0000	2,0000
		V11	3,0000	,0365	,1672	3,0000	3,5000
		V12	2,0000	-,0850	,2576	1,0000	2,0000
		V15	3,0000	,0010	,0223	3,0000	3,0000
		V16	3,0000	-,0010	,0500	3,0000	3,0000
	50	V21.03	3,0000	,0005	,0158	3,0000	3,0000
		V21.06	2,0000	,0280	,1440	2,0000	2,5000
		V22	2,0000	,0015	,0274	2,0000	2,0000
		V23	3,0000	,0000	,0000	3,0000	3,0000

		V26.01	3,0000	,0000	,0000	3,0000	3,0000
		V26.03	3,0000	,0350	,1591	3,0000	3,5000
		V26.05	2,0000	,0000	,0224	2,0000	2,0000
		V26.06	3,0000	,0000	,0000	3,0000	3,0000
		V26.11	2,0000	-,0795	,2470	1,0000	2,0000
		V26.14	2,0000	,0000	,0000	2,0000	2,0000
		V26.15	3,0000	,0000	,0000	3,0000	3,0000
		V26.16	3,0000	-,0005	,0158	3,0000	3,0000
		V26.17	2,0000	-,0050	,0836	2,0000	2,0000
		V27.03	3,0000	-,0105	,0874	3,0000	3,0000

			Statistic	Bootstrap[a]			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
Percentiles	50	V27.04	3,0000	,0060	,0705	3,0000	3,0000
		V27.07	3,0000	-,0080	,0771	3,0000	3,0000
		V28	3,0000	-,0295	,1685	2,5000	3,0000
		V29.12	3,0000	,0010	,0223	3,0000	3,0000
		V29.13	3,0000	,0000	,0000	3,0000	3,0000
		V29.16	3,0000	-,1855	,3492	2,0000	3,0000
		V29.18	3,0000	,0000	,0000	3,0000	3,0000
		V29.22	2,0000	-,0015	,0274	2,0000	2,0000
		V29.23	2,0000	-,0360	,1665	1,5000	2,0000
		V29.24	2,0000	-,0360	,1665	1,5000	2,0000
		V11	4,0000	-,2705	,4051	3,0000	4,0000
		V12	2,7500	-,2130	,4638	2,0000	3,0000
		V15	3,0000	,3278	,4311	3,0000	4,0000
		V16	3,0000	,3163	,4294	3,0000	4,0000
		V21.03	3,0000	,1580	,3360	3,0000	4,0000
		V21.06	3,0000	-,2980	,4137	2,0000	3,0000
		V22	2,0000	,3158	,4305	2,0000	3,0000
		V23	3,0000	,1413	,3219	3,0000	4,0000
		V26.01	3,0000	,0448	,1893	3,0000	3,7500
		V26.03	4,0000	-,2902	,4123	3,0000	4,0000
	75	V26.05	2,0000	,3237	,4329	2,0000	3,0000
		V26.06	3,0000	,0032	,0518	3,0000	3,0000
		V26.11	2,0000	,1450	,3211	2,0000	3,0000
		V26.14	2,0000	,0327	,1621	2,0000	2,7500
		V26.15	3,0000	,0008	,0237	3,0000	3,0000
		V26.16	3,0000	,0038	,0529	3,0000	3,0000
		V26.17	2,0000	,3205	,4331	2,0000	3,0000
		V27.03	3,0000	,0380	,1727	3,0000	3,7500
		V27.04	3,7500	-,2107	,4573	3,0000	4,0000
		V27.07	3,0000	,0413	,1838	3,0000	3,7500
		V28	3,0000	,3100	,4273	3,0000	4,0000

			Statistic	Bootstrap[a]			
				Bias	Std. Error	95% Confidence Interval	
						Lower	Upper
Percentiles	75	V29.12	3,0000	,3403	,4389	3,0000	4,0000
		V29.13	3,0000	,0395	,1756	3,0000	3,7500
		V29.16	3,0000	-,0010	,0316	3,0000	3,0000
		V29.18	3,0000	,0317	,1592	3,0000	3,7500
		V29.22	2,0000	,1515	,3275	2,0000	3,0000
		V29.23	2,0000	,0368	,1688	2,0000	2,7500
		V29.24	2,0000	,1420	,3310	2,0000	3,0000

a. Unless otherwise noted, bootstrap results are based on 1000 bootstrap samples

Frequency Table

V11

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent[a]	
						Bias	Std. Error
Valid	2,00	3	15,0	15,0	15,0	-,1	7,8
	3,00	11	55,0	55,0	70,0	-,3	11,4
	4,00	6	30,0	30,0	100,0	,4	10,5
	Total	20	100,0	100,0		,0	,0

V11

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	2,00	,0	30,0
	3,00	35,0	75,0
	4,00	10,0	50,0
	Total	100,0	100,0

a. Unless otherwise noted, bootstrap results are based on 1000 bootstrap samples

Prilog 15: Deo rezultata kvantitativne analize podataka u trećem krugu istraživanja - vrednosti McNemar-Bowker-ovog testa za ispitanika ISP20

Crosstabs

Output Created		04-JUL-2015 18:06:54
Comments		
Input	Data	C:\Users\Maric\Documents\Mirjana Marić\TREĆI KRUG\PODACI_ZA_III_KRUG\PODACI_IIIkrug.sav
	Active Dataset	DataSet1
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	28
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics for each table are based on all the cases with valid data in the specified range(s) for all variables in each table.
Syntax		CROSSTABS /TABLES=IP20D BY IP20T /FORMAT=AVALUE TABLES /STATISTICS=CHISQ KAPPA MCNEMAR /CELLS=COUNT /COUNT ROUND CELL.
Resources	Processor Time	00:00:00,03
	Elapsed Time	00:00:00,02
	Dimensions Requested	2
	Cells Available	174762

Case Processing Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
IP20D * IP20T	28	100,0%	0	0,0%	28	100,0%

IP20D * IP20T Crosstabulation

Count

		IP20T			Total
		2	3	4	
IP20D	1	1	0	0	1
	2	11	3	0	14
	3	0	11	0	11
	4	0	1	1	2
Total		12	15	1	28

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)
Pearson Chi-Square	31,167[a]	6	,000
Likelihood Ratio	28,403	6	,000
Linear-by-Linear Association	17,291	1	,000
McNemar-Bowker Test	.	.	.[b]
N of Valid Cases	28		

a. 9 cells (75,0%) have expected count less than 5. The minimum expected count is ,04.
b. Computed only for a PxP table, where P must be greater than 1.

Symmetric Measures

		Value	Asymp. Std. Error[a]	Approx. T[b]	Approx. Sig.
Measure of Agreement	Kappa	,688	,119	4,496	,000
N of Valid Cases		28			

a. Not assuming the null hypothesis.
b. Using the asymptotic standard error assuming the null hypothesis.

Prilog 16: Artifakti metodološko-radnog okvira za razvoj softverske arhitekture poslovnog softvera u agilnim procesima

A. Dokument vizija (sistema)

1. Problem koji se rešava

Opis problema	
Ključne prednosti uspešnog rešavanja problema	

2. Stejkholderi sistema

Naziv stejkholdera	Ključne odgovornosti	Sadašnja očekivanja/očekivanja od budućeg sistema

3. Poslovna i informaciona arhitektura ciljne organizacije

Prikaz toka poslovanja organizacije, neformalnim blok dijagramom.

4. Identifikovani budući ciljevi i pravac razvoja poslovanja organizacije

Kratak opis koji će biti osnova za kasnije generisanje liste zahteva budućih promena sistema.

5. Identifikovane karakteristike budućeg sistema

Poslovne potrebe	Karakteristike sistema (opšti nivo)	Vrednost koju obezbeđuje karakteristika sistema: kritična / važna / korisna	Rrizik	Napor

6. Softverska i tehnička infrastruktura ciljne organizacije

- Postojeće softverske aplikacije u organizaciji (kratak opis)
- Načina korišćenja postojećih softverskih aplikacija (kratak opis)
- Međusobna povezanost postojećih softverskih aplikacija
- Postojeća hardverska infrastruktura koja omogućuje izvršavanje softverske aplikacije

Nakon evidentiranja i kratke analize softverske i tehničke infrastukture (u tabeli), neophodno je dati odgovore na sledeća pitanja:

- Da li je tehnologija u okviru ciljne organizacije dokazana i odgovarajuća i za novi sistem?
- Da li se povećava rizik uvođenjem najnovijih tehnologija?
- Da li je uspeh projekta zavisi od upotrebe najnovijih, nedovoljno ispitanih tehnologija?
- Da li je opravdan cilj ponovne upotrebe softverskih komponenti koje postoje u ciljnoj organizaciji?
- Da li su postojeće softverske komponente u organizaciji razvijane ili su kupovane?
- Da li spoljne zavisnosti od drugih sistema uključuju i one izvan ciljne organizacije?
- Da li već postoje interfejsi za njihovu komunikaciju ili ih je potrebno razviti novim sistemom?

7. Lista arhitekturalnih rizika

Rizik	Izloženost arhitekture riziku	Rang rizika	Strategija za mitigaciju

7.1 Korak utvrđivanje granica sistema, podrazumeva ažuriranje liste rizika u tabeli, davanjem odgovora na sledeći set pitanja:

- da li je zahtev za količinom transakcija koje treba da podrži sistem razuman?
- da li je količina podataka sa kojima sistem treba da radi razumna?
- da li postoje neuobičajeni ili tehnički zahtevni zahtevi, koje razvojni tim nije nikada rešavao do tada?
- da li su identifikovani zahtevi prilično stabilni i jasni?
- da li postoje neki ekstremno nefleksibilan nefunkcionalan zahtev (npr. da sistem ne sme nikada da propadne), da li su zahtevi kompleksni?
- da li su eksperti domena na raspolaganju?
- da li tim ima dovoljno ljudi, sa odgovarajućim veštinama i iskustvom?

7.2 Rizici identifikovani tokom potvrde i implementacije arhitekture

8. Tim

Ime i prezime	Iskustvo, veštine, ekspertska znanja	Dodeljena pozicija/odgovornosti

9. Arhitekturalno značajni zahtevi

- Ključni funkcionalni zahtevi.
- Nefunkcionalni zahtevi.
- Zahtevi budućih promena.
- Zahtevi usled geografske dislociranosti.
- Ograničenja iz realnog okruženja.

10. Tehničko-tehnološka pitanja

- Odluke o tehnološkom steku.
- Rezultati analize trenda opcija.
- Odabran frejmvork za implementaciju.
- Odabran programski jezik.
- Odluka o korišćenju postojećih biblioteka klasa.
- Odluka o ponovnoj upotrebi postojećih softverskih komponenti.

11. Izbor arhitekturalnog rešenja

- Definisani kriterijumi za izbor arhitekturalnog rešenja
- Izbor tehnike za evaluaciju potencijalnih arhitekturalnih rešenja
- Rezultati analize i procene potencijalnih arhitekturalnih rešenja (prednosti/nedostaci).
- Obrazlaganje odluke o izabranom arhitekturalnom rešenju.

B. Dokument softverska arhitektura sistema

1. Moduli za glavni deo sistema i veze između njih

Grafički prikaz zasnovan na neformalnom tipu modela, izrađenom na papiru ili tabli i dokumentovanom u formi slike. Opšti pregled arhitekture ilustruje suštinu predloženog arhitekturalnog rešenja i glavne gradivne blokove (elemente, module), od kojih će biti izgrađena arhitektura sistema.

2. Definisanje podsistema i slojeva za identifikovane module

Grafički prikaz zasnovan na neformalnom tipu modela, izrađenom na papiru ili tabli i dokumentovanom u formi slike. Organizacija podsistema za svaki identifikovani modul treba da predstavlja nivo dizajna sistema na najvišem nivou apstrakcije.

3. Definisanje ključnih elemenata, interakcija i mehanizama dizajna

- Grafički prikaz zasnovan na neformalnom tipu modela, izrađenom na papiru ili tabli i dokumentovanom u formi slike. Prikaz uključuje inicijalne elemente dizajna, koji će se tokom implementacije sistema modifikovati i obogaćivati detaljima, kao i interakcije između ključnih apstrakcija, čime je uspostvaljen način komunikacije elemenata dizajna.
- Tačka uključuje i listu identifikovanih mehanizama dizajna.
- Odluka o načinu njihove implementacije dokumentuje se tokom faze implementacije.

4. Arhitekturalne odluke o modelu podataka

- Odluka o tipu baze podataka koja će se razvijati.
- Problemi i rešenja koja se odnose na kompatibilnost podataka.
- Inicijalna postavka modela podataka, koji se tokom implementacije proširuje i obogaćuje detaljima.

5. Identifikovane tačke arhitekture koje trebaju biti fleksibilnije

Odluka o identifikovanim tačkama arhitekture koje trebaju biti fleksibilnije u odnosu na identifikovane zahteve o budućim promenama sistema.

6. Arhitekturalne odluke o modelu raspoređivanja

- Odluka o rešenju modela raspoređivanja
- Inicijalna postavka modela raspoređivanja

7. Ključne odluke o modelu implementacije

Ključne odluke o modelu implementacije, bez razrade detalja.

8. Odluke i obrazloženja o izmenama arhitekture

Uključuje odluke koje su rezultat potvrde i sagladavanja arhitekture sistema, tokom njene implementacije.

Prilog 17: Dopis ekspertima za ocenu postavljenog metodološko-radnog okvira

Poštovani **<<ime i prezime eksperta>>**,

u okviru izrade doktorske disertacije pod nazivom „Metodološko-radni okvir za razvoj softverske arhitekture poslovnog softvera u agilnim procesima", na osnovu rezultata sprovedenog teoretsko-empirijskog istraživanja, dizajniran je okvir za razvoj arhitekture kompleksnih sistema u agilnim procesima.

Kao eksperta iz oblasti razvoja softverske arhitekture agilnim procesima, molim Vas da postavljeni okvir ocenite u celini, na osnovu tvrdnji koje se nalaze u tabeli 1 (ocena 1 - u potpunosti se ne slažem sa tvrdnjom; ocena 6 - u potpunosti se slažem sa tvrdnjom).

Unapred zahvalna,
Mirjana Marić.

Tabela P-17.1 Tvrdnje za ocenu okvira

	Tvrdnja	1	2	3	4	5	6
1.	Postavljeni okvir u dovoljnoj meri proširuje agilne procese arhitekturalnim praksama, u cilju njihovog osposobljavanja za razvoj kompleksnih poslovnih sistema.						
2.	Postavljeni okvir je jasno definisan.						
3.	Postavljeni okvir je koristan na projektima razvoja kompleksnih sistema agilnim procesima.						
4.	Postavljeni okvir je primenljiv u organizacijama.						
5.	Upotreba postavljenog okvira doprinosi većem kvalitetu kompleksnih poslovnih softverskih rešenja u agilnim procesima.						
6.	Upotreba postavljenog okvira smanjila bi rizik erozije arhitekture						
7.	Postavljeni okvir uravnotežava razvoj funkcionalnosti i arhitekture, neugrožavajući agilnost projekta.						