SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE
POSLIJEDIPLOMSKI STUDIJ ELEKTROTEHNIKE I INFORMACIJSKE TEHNOLOGIJE

Kvalifikacijski doktorski ispit

PRIKUPLJANJE, DOKUMENTIRANJE I VALIDACIJA KORISNIČKIH ZAHTJEVA NA PROGRAMSKU PODRŠKU

Split, svibanj 2017 mr.sc. Srđana Dragičević

SADRŽAJ

1. Sažetak i ključne riječi

1.1 Sažetak

Veliki broj projekata programske podrške obustavi se ili značajno premaši rokove i/ili proračun. Rezultati su još lošiji ako se u definiciju uspješnosti uključi zadovoljstvo korisnika. Ključni čimbenik uspjeha su potpuni, razumljivi i nedvosmisleni zahtjevi. Kvaliteta zahtjeva ovisi o aktivnostima prikupljanja, dokumentiranja i validacije zahjeva, ali i o komunikaciji među sudionicima. Strukturne i agilne metode razvoja koriste iste tehnike prikupljanja zahtjeva. Prisustvo korisnika na licu mjesta rezultira nešto boljim uspjehom agilnih metoda, ali one nisu pogodne za sve vrste projekata programske podrške. U radu su predstavljene aktivnosti i metode inženjerstva zahtjeva, kao i način na koji SEMATdefinira inženjerstvo zahtjeva.

1.2 Ključne riječi

Inženjerstvo zahtjeva, prikupljanje zahtjeva, agilni razvoj programske podrške, SEMAT

2. Uvod

Točnost i kvaliteta zahtjeva znatno doprinose uspjehu projekta programske podrške, a kvaliteta zahtjeva je ključni čimbenik za zadovoljstvo klijenata/korisnika proizvoda. Greške uslijed nepotpunih, pogrešnih i/ili dvosmislenih zahtjeva su najčešće i najskuplje pogreške u projektima, i glavni su krivac prekinutih projekata. Problem je u složenosti. Inženjerstvozahtjeva nije samo tehnički proces; na zahtjeve utječu klijentove želje, averzije i predrasude kao i politička i organizacijska pitanja. To su osnovne ljudske karakteristike i sama tehnologija ne može razriješiti ove probleme, iako način i aktivnosti prikupljanja zahtjevautječu na kvalitetu zahtjeva.

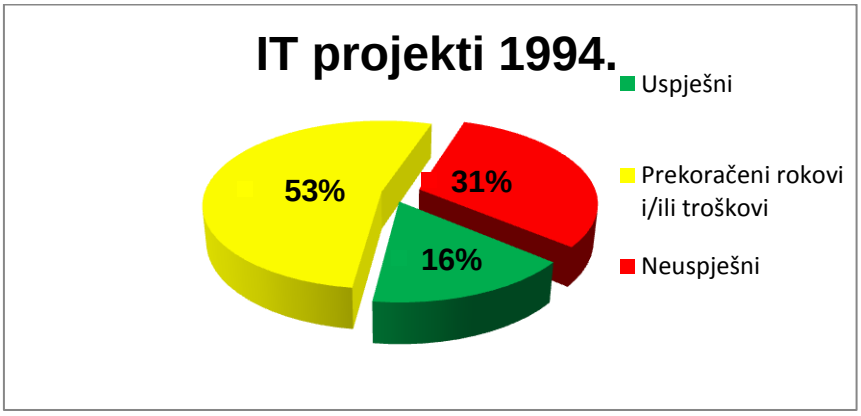
U projektima agilnog razvoja programske podrške korisnici su više uključeni, ali se koriste iste metode prikupljanja i upravljanja zahtjevima kao i kod strukturnog razvoja, pa su i problemi komunikacije i (ne)razumijevanja među sudionicima isti. Ipak, agilne metode razvoja pokazale su se uspješnijima od strukturnih metoda, iako ni njihovi rezultati nisu zadovoljavajući. Ali, nisu svi projekti prikladni za primjenu agilnih metoda. Čimbenici za odabiragilnog ili strukturnog pristupa navedeni suu Poglavlju 4.

Statistika uspješnosti projekata programske podrške i utjecaj inženjerstva zahtjeva na uspjeh projekta obrađeni su u Poglavlju 2. U Poglavlju 3 dan je kratak pregled osobina agilnog razvoja programske podrške s naglaskom na korištenim metodama prikupljanja zahtjeva. U Poglavlju 5 opisani su osnovni pojmovi vezani uz zahtjeve i procese inženjerstva zahtjeva kao i osobine neuspješnih projekata vezane uz definiranje zahtjevâ. U Poglavlju 6 opisana su stanja zahtjeva prema SEMAT-u te kontrolni popisi za lakšu procjenu stanja i napretka na projektu. Prije samog zaključka, u u Poglavlju7 obrađeni su rezultati pretraživanja znanstvene literature.

3. Uspješnost projekata programske podrške

Veliki broj projekata programske podrške nikad se ne realizira do kraja ili značajno premaši predviđene rokove i raspoložive resurse. 1994. godine Standish Group izvršilaje analizu 8.380 projekatarazvoja programske podrške u državnom i privatnom sektoru u SAD-u[1]. Za potrebe studije projekti su, prema rezultatu, razvrstani u jedan od slijedećatri tipa:

- tip 1 ili uspješan projekt (succeeded): projekt je završen na vrijeme, unutar predviđenog proračuna i sa svim izvorno specificiranim mogućnostima i funkcijama.
- tip 2 (challenged): projekt je završen i produkt je u uporabi, ali su prekoračeni vremenski rokovi i/ili troškovi i/ili su realizirane samo neke izvorno specificirane mogućnosti i funkcije.
- tip 3 ili obustavljen projekt (failed): projekt je obustavljen prije završetka.



Slika 1. Uspješnost projekata razvoja programske podrške 1994.

Studija je pokazala[1]:

- 31% projekata razvoja programske podrške je obustavljeno prije završetka (utrošena $81 milijarda)
- 53% je kompletirano s prosječnom cijenom koja je za oko 189% premašila predviđenu cijenu
- od ovih 53% samo 42% projekata je zadržalo njihove izvorne (originalne) značajke i funkcije
- 9% projekata je kompletirano na vrijeme i s predviđenim budžetom (velike tvrtke)

- 16% projekata je kompletirano na vrijeme i s predviđenim budžetom (male tvrtke).

Nakon velikog odjeka njihove studije iz 1994.g.,StandishGroup je nastavila provoditi ankete iobjavljivati godišnje izvještaje o uspješnosti projekata programske podrške. Treba naglasiti da nije bilo velikih promjena u rezultatima anketa. Rezultati su još lošiji,ako se zna da u anketama nije ispitvano zadovoljstvo korisnika. Korisnici mogu biti nezadovoljni čak i s projektima koji su završeni na vrijeme, unutar predviđenog proračuna, a realizirali su i specificirani opseg. Pokazalo se da su korisnici nezadovoljni sa 7% projekata uspješnih po kriterijima ankete. Stoga je, za 2015.g., proširena definicija uspješnog projekta: završen je na vrijeme, unutar predviđenog proračuna i korisnik je zadovoljan postignutim rezultatom[2].Rezultati uspješnosti projekata prema prema novoj definiciji prikazani su u *Tablica 1*.

Tablica 1. Uspješnost projekata razvoja programske podrške 2011-2015 [2]

	2011	2012	2013	2014	2015
Uspješni	29%	27%	31%	28%	29%
Prekoračeni rokovi i/ili troškovi	49%	56%	50%	55%	52%
Neuspješni	22%	17%	19%	17%	19%

Sudionici Standish Group ankete iz 1994.g. [1] identificirali su uzroke neuspjeha projekta. U *Tablica 2* su navedena tri glavna uzrokauslijed kojih dolazi do promjena u projektu, prema mišljenju sudionika ankete. Proizlazi da je glavni uzrokneuspjeha nedostatak jasnoće u prikupljanju i komuniciranju korisničkih zahtjeva te slabo upravljanje zahtjevima.

Tablica 2. Uzroci zbog koji dolazi do promjena projekta

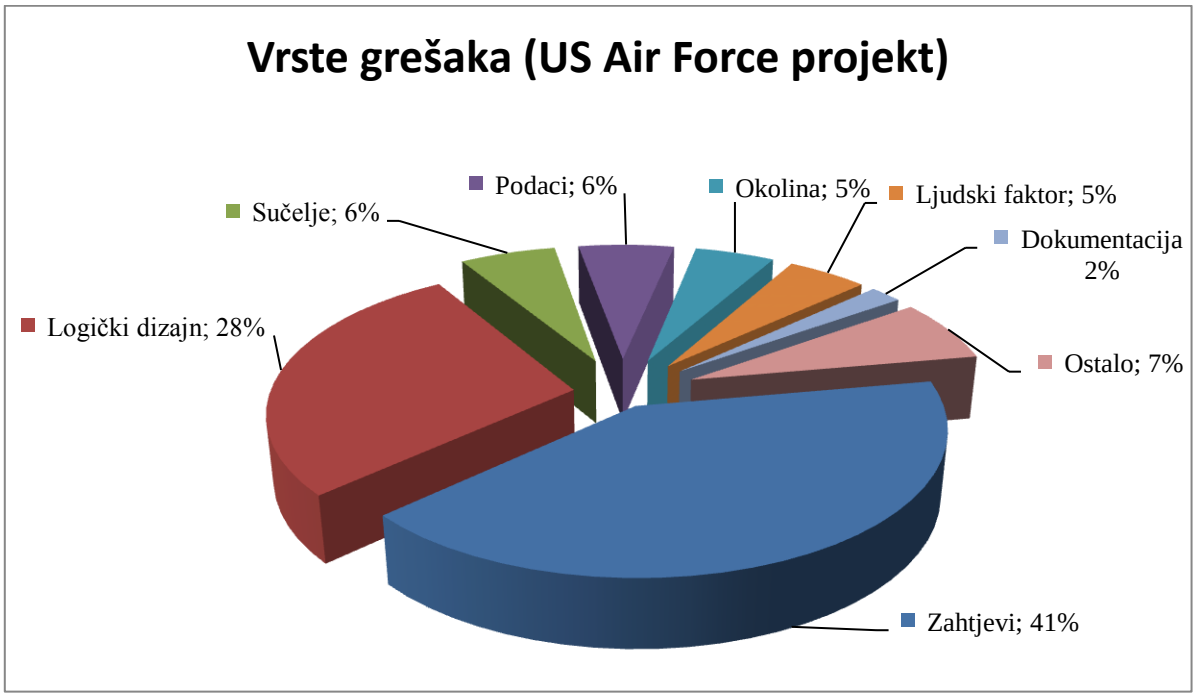
	Uzroci zbog koji dolazi do promjena projekta	% odgovora
1.	Nedostatak potrebnih korisničkih podataka	12,8%
2.	Nepotpuni zahtjevi i specifikacije	12,3%
3.	Promjene zahtjeva i specifikacija	11,8%

I druge studije potvrđuju da su greške napravljene u fazi definiranja zahtjeva najčešće greške ovih projekata. Sheldon [3] u svojoj studiji US Air Force projekata, navodi da su pogrešni zahtjevi uzrokovali 41%, a loš logički dizajn samo 28% ukupnih grešaka (Slika 2). Leffingwell [4] potvrđuje ove rezultate citirajući rezultate studije Tavolato – Vincena, gdje se navodi da 56% svih grešaka ima korijen u fazi definiranja zahtjevâ.

Istraživanje 12 tvrtki [5] potvrđuje ove rezultate: 48% problema na projektima programske podrške uzrokovano je nepotpunim/pogrešnim zahtjevima. Rezultat je jaz između očekivanja i potreba klijenata i mogućnosti stvarnog sustava.

I noviji radovi[6], [7],[8]potvrđuju utjecaj inženjerstva zahtjeva na uspjeh razvojnih projekata. Neuspjesi projekata razvoja programske podrške su globalni problem. Samo u Sjedinjenim Državama troškovi obustavljenih projekata dosežu desetke milijardi dolara, a različite studije pokazuju da su loša kvaliteta i loše upravljanje zahtjevima glavni čimbenici neuspjeha [6].

Uspjeh projekta ovisi o točnosti prikupljenih zahtjeva, ali i o učinkovitom upravljanju zahtjevima[7].Inženjerstvo zahtjeva treba biti polazište svakog projekta, jer pomaže sudionicima odrediti i usredotočiti se na cilj, te uskladiti međusobne potrebe, a kvaliteta zahtjeva je ključni čimbenik za zadovoljstvo kupaca / korisnika proizvoda. Kako bi se osigurao uspjeh i smanjili rizici, potrebno je upravljati zahtjevima u svakoj fazi razvoja programske podrške.

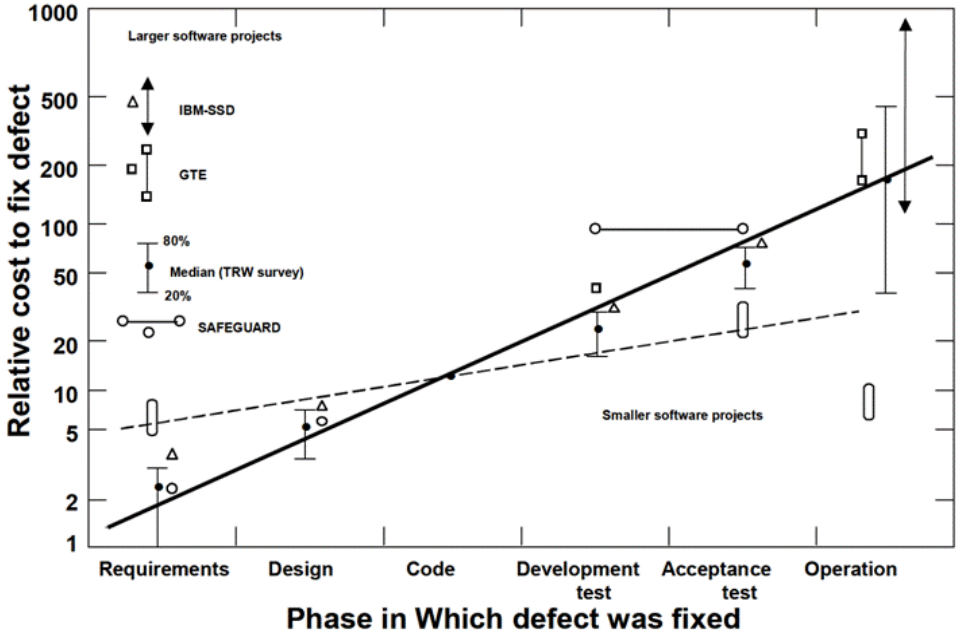


Slika 2. Vrste grešaka na US Air Force projektu[3]

Anketa [8]provedena među IT stručnjacima pokazala je da se greške nepotpunih/pogrešnih zahtjeva najčešće otkrivaju u kasnijim fazama razvoja: fazi projektiranja, implementacije i ispitivanja.

Ispravke grešaka nepotpunih/pogrešnih zahtjeva vrlo su jeftine ako se otklone u dovoljno ranoj fazi razvoja, ali troškovi dramatično rastu kako procesrazvoja programske podrške napreduje[4]. Boehm i Basili[9] smatraju da troškovi otklanjanja grešaka nepotpunih /pogrešnih zahtjeva rastu do 100 puta ako se otklone tek u završnim fazama razvoja. Leffingwell [4]navodi da se cca 40% od ukupnog proračuna projekta odnosi na troškove ponovne izrade (rework), pri čemu je 70% - 85% tih troškova direktno uzrokovano niskom kvalitetom zahtjeva (dokumentacije zahtjevâ).

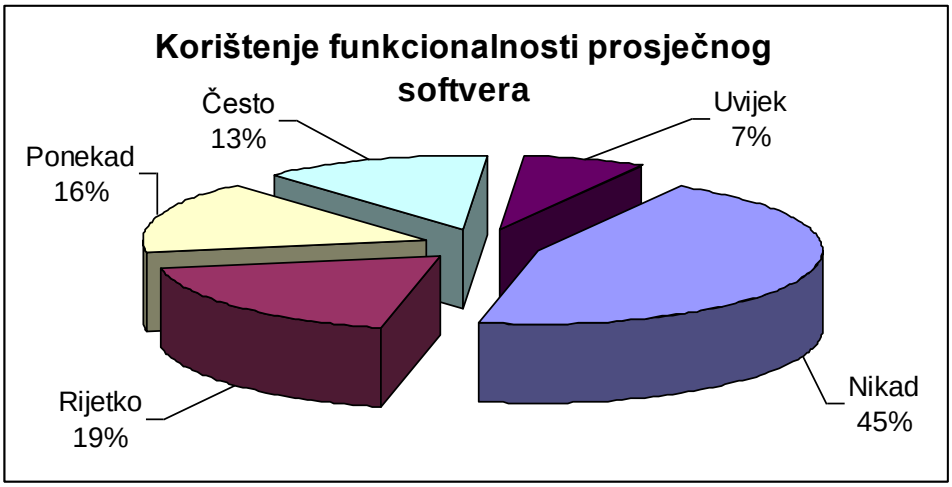
Pri tome u ove troškove nisu uključeni tkz. nematerijalni troškovi uzrokovani ovim greškama. Nematerijalni troškovi uključuju nedostatak funkcionalnosti koje su mogle biti isporučene da resursi projekta nisu potrošeni na ispravak grešaka, gubitak povjerenja od strane kupaca, popratni gubitak udjela na tržištu, a time i pad prihoda i dobiti.



Slika 3 . Troškovi ispravljanja grešaka po fazama projekta[10]

Inženjerstvo zahtjeva se smatra najkritičnijom fazom projekata programske podrške[11], [12] i loše implementirani zahtjevi predstavljaju glavni rizik neuspjeha projekta.

Složenost inženjerstva zahtjeva i utjecaj na trajanje i troškove projekata programske podrške, kao i na korisnost krajnjeg produkta naglašava studija Standish Group[13] iz 2002.g. Prema rezultatima te studije 45% funkcionalnosti prosječneprogramske podrške se nikad ne koristi, a vrlo rijetko 19% funkcionalnosti. Samo 20% razvijenih funkcionalnosti se redovito koristi (Slika 4).



Slika 4. Korištenje funkcionalnosti programske podrške

Iz navedenog proizlazi da je glavni uzrok neuspjeha projekata programske podrške nedostatak jasnoće u prikupljanju i komuniciranju korisničkih zahtjeva te slabo upravljanje zahtjevima.Stoga se ovaj rad bavi inženjerstvom zahtjeva s naglaskom na metode prikupljanja i dokumentiranja zahtjevâ.

Iako ni agilni projekti nemaju zadovoljavajući postotak uspješnosti, agilne metode razvoja pokazale su se uspješnijima od vodopadne metode[2], pa su u slijedećem poglavlju iznesene osnovne karakteristike agilnog razvoja programske podrške. I ovdje je naglasak na korištenim metodama prikupljanja zahtjeva.

4. Agilni razvoj programske podrške

Anketa StandishGroupza 2015.g.[2] pokazala je da je 39% agilnih projekata bilo je uspješno u odnosu na 11% projekata razvijenih vodopadnom metodom. Kod velikih projekata podaci su još dramatičniji: agilne metode su 600% uspješnije (*Tablica 3*).

Tablica 3. Uspješnost projekata prema razvojnim metodama (2011-2015.g) [2]

Veličina	Metoda	Uspješni	Prekoračeni rokovi i/ili troškovi	Neuspješni
Svi projekti	Agilna	39%	52%	9%
	Vodopad	11%	60%	29%
Veliki projekti	Agilna	18%	59%	23%
	Vodopad	3%	55%	42%
Projekti srednje veličine	Agilna	27%	62%	11%
	Vodopad	7%	68%	25%
Mali projekti	Agilna	58%	38%	4%
	Vodopad	44%	45%	11%

Agilni razvoj nije metodologija; to je termin kojim se opisuje nekoliko neovisno razvijenih metoda koje dijele iste vrijednosti[14],[15]:

- Pojedinci i interakcije ispred procesa i alata;
- Programska podrška koja radi ispred opsežne dokumentacije;
- Suradnja s klijentima ispred pregovaranje oko ugovora;
- Odgovor na promjene ispred striktnog pridržavanja plana.

Agilne metode temelje se na 12 načela [14]. Primjene agilnih načela i vrijednosti u određenoj situaciji i na određenom projektu, nazivaju se praksama. Primjeri takvih praksa su: refaktoriranje, programiranje u paru, cijepanje korisničke priče, mapiranje korisničke priče, zajedničko vlasništvo koda, kontinuirane isporuke, klijent na licu mjesta, dnevni sastanci, vlasnik proizvoda itd. Razvojni timovi obično ne koriste sve prakse odabrane metodologije, već biraju podskup agilnih praksi prikladan za određeni projekt[15].
Agilne metode temelje se na iterativnim i evolucijskim metodama. Fleksibilan i brz odgovor na promjene osiguran je kratkoćom razvojnih iteracija, minimalnim planiranjem i neprestanim poboljšanjima. Duljina iteracija je fiksna, obično od jednog tjedna do mjesec dana. Svaka iteracija je projekt sam po sebi. To uključuje sve aktivnosti razvojnog

ciklusa, od analiza zahtjeva do prihvaćanja klijenta, a završava (internim) objavljivanjem aplikacije. Nepotrebna dokumentacija se odstranjuje[15].

Agilna metodologija nastoji osigurati zadovoljstvo korisnika dostavljajući samo potrebne funkcionalnosti, a za to su neophodni točni i potpuni zahtjevi. To se pokušava postići uključivanjem korisnika, ali tehnike prikupljanja zahtjeva praktično se ne razlikuju od onih korištenih u strukturnom razvoju programske podrške [16], [17].

To znači da problem komunikacije između korisnika i razvojnog tima i dalje postoji, jer korisnik često ne zna objasniti što zapravo treba, ili klijent zanemaruje zahtjeve krajnjeg korisnika, već uvjetujeimplementaciju svojim prihvaćanjem.

Promjena zahtjeva tijekom projekta je uobičajena, i detaljna dokumentacija u ranoj fazi projekta može biti gubitak vremena i napora. U agilnom razvoju zahtjevi se definiraju na početku svake iteracije što štedi vrijeme i resurse, ali rezultira slijedećim nedostacima [16], [18], [19]:

- Implicitno prikupljanje ne-funkcionalnih zahtjeva; tehnike prikupljanja i upravljanja zahtjevima nisu ponuđene;
- Nedostatak šire slike; nedostaju veze među zahtjevima;
- Problemi u održavanju proizvoda zbogograničene dokumentacije;
- Nedostatak inženjerskih aktivnosti u definiranju specifikacije zahtjevâ.

Treba spomenuti i Lean pristup, koji ima korijene u procesu proizvodnje u japanskoj tvornici Toyota. Lean filozofija temelji se na razumijevanju kupca i njegovog poimanja vrijednosti. Mary i Tom Poppendieck [20] prilagodili su taj pristup procesu razvoja programske podrške i definirali su sedam Lean načela: optimizacija cjeline, eliminiranje gubitaka, ugradnja kvalitete, konstantno učenje, brza isporuka, uključivanje svih te neprestano poboljšanje. Autori navode da su lean načela temelj na kojem se mogu graditi agilne praksekako bi se povećala učinkovitost. Lean načela i vrijednosti omogućavaju timovima i tvrtkama visoke performanse, a agilni pristup pruža najbolju praksu i metode za rad prema vrijednostima Lean pristupa. U praksi, razvojni timovi kombiniraju agilna i lean načela nastojeći postići ekonomsku učinkovitost i fleksibilnost [21], [22].

5. Čimbenici za odabir agilnog ili strukturnog pristupa

Ne postoji idealna metoda razvoja programske podrške: i agilne i strukturne metode imaju svoje dobre i loše strane. Uvjeti na projektu definiraju prikladnost metoda. Složena priroda razvoja programske podrškei širok raspon metoda otežavaju usporedbu agilnog i strukturnog pristupa i čine je nepreciznom. Boehm i Turner [23]pronašli sučetiripodručja za koje postoje jasne razlike između agilnih i strukturnih metoda. To su:

- Aplikacija, uključujući primarne ciljeve projekta, veličinu projekta i okolinu u kojoj će aplikacija raditi.
- Upravljanje, uključujući odnose s klijentima, planiranje i kontrolu te komunikaciju na projektu.
- Tehničke osobine, uključujući prikupljanje zahtjeva i upravljanje zahtjevima, razvojne aktivnosti i testiranje.
- Osoblje, uključujući osobine korisnika, osobine razvojnog tima i organizacijsku kulturu.

Grupa uvjeta prikladnih za primjenu pojedinog pristupa, za svako navedeno područje, prikazana je u Tablica 4.

Tablica 4. Uvjeti prikladni za primjenu metoda agilnog i strukturnog razvoja

Osobine	Agilne metode razvoja	Strukturne metode razvoja
Aplikacija		
Primarni ciljevi	Brzi rezultati; odgovor na promjene	Predvidljivost, stabilnost, visoka sigurnost
Veličina	Manji timovi i projekti	Veći timovi i projekti
Okolina	Turbulentna; jako promjenljiva; usmjerena na projekt	Stabilna; slabo promjenjljiva; usmjerena na projekt / organizaciju
Upravljanje		
Odnosi s klijentima	Korisnici na licu mjesta; usredotočeni na prioritizaciju	Interakcija korisnikapo potrebi; usredotočenost na odredbe ugovora
Planiranje i kontrola	Internalizirani planovi; kvalitativna kontrola	Dokumentirani planovi, kvantitativna kontrola

Komunikacija	Prešutno znanje članova tima	Eksplicitno dokumentirano znanje
Tehničke osobine		
Zahtjevi	Prioritetne neformalne priče i test slučajevi; trpi nepredvidljive promjene	Formalizirani projekt, sposobnost, sučelje, kvaliteta, predvidljivaevolucija zahtjeva
Razvoj	Jednostavan dizajn; kratkeiteracije; refaktoriranje je jeftino	Opsežan dizajn; dulje iteracije; refaktoriranje je skupo
Testiranje	Kreirani testovi definiraju zahtjeve	Dokumentirani testni planovi i postupci
Osoblje		
Korisnici	Privrženi, na licu mjesta, CRACK* izvršitelji	CRACK* izvršitelji,ne uvijek na licu mjesta
Razvojni tim	Barem 30% vremena specijalisti razine 2 i 3 po Cockburn-u; bez osoblja razine 1B ili -1**	U početnoj fazi 50% vremena specijalisti razine 3 po Cockburn-u; 10% tijekom cijelog projekta; 30% vremena može raditi razina 1B; bez razine -1**
Organizacijska kultura	Ugodna okolina i veći stupanj slobode (uspješnostkroz kaos)	Ugodna okolina i jasno definirane uloge kroz postupke i procedure (uspješnostkroz red)
*CRACK – eng. Collaborative, Representative,Authorized, Committed, Knowledgeable (suradnik, predstavnik, ovlašten, predan, obrazovan). **Ovi brojevi variraju ovisno o složenosti aplikacije.		

Što je veća razlika između uvjeta na projektu i uvjeta za odabir određenog pristupa, to je veći rizik korištenja tog pristupa u čistom obliku, a pogodnije je uključivanje neke komplementarne prakse iz suprotne metode.

Kao "sažetak sažetaka", Boehm i Turner [23]zaključili su da postoji 5 ključnih čimbenika koji određuju prikladnost agilnog ili strukturnog pristupa na određenom projektu. Ti

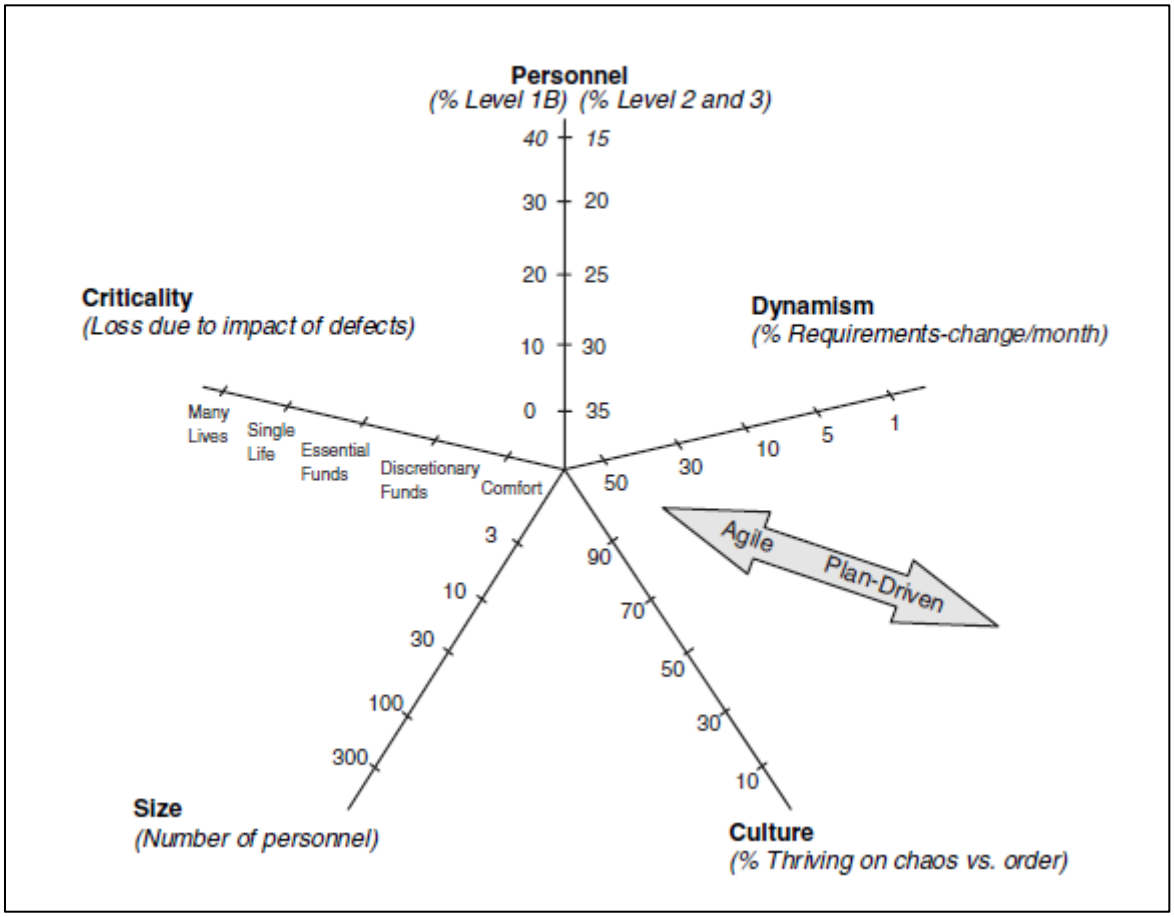
čimbenici su na veličina projekta, kritičnost, dinamika, osoblje i organizacijska kultura.U Tablica 5navedeno je kojesuvrijednosti ključnih čimbenika pogodne za primjenu agilnih,a koje za primjenu strukturnih metoda razvoja. Ako vrijednosti četiri čimbenika odgovaraju odabranom pristupu, ali ne i peti, potrebno je napraviti procjenu rizika i odabrati neku mješavinu agilnih i strukturnih metoda.

Tablica 5. Pet ključnih čimbenika za agilni ili strukturni pristup

Čimbenik	Agilne metode razvoja	Strukturne metode razvoja
Veličina projekta	Metode dobro prilagođene malim projektima i timovima. Oslanjanje na iskustveno znanje ograničava skalabilnost.	Metode razvijene za velike projekte i timove. Teško ih je prilagoditi za male projekte.
Kritičnost	Neprovjereno na projektima razvoja sigurnosno kritičnih proizvoda. Potencijalne teškoće s jednostavnim dizajnom i nedostatkom dokumentacije.	Metode razvijene za izradu visoko kritičnih proizvoda. Teško ih je prilagoditi razvoju proizvoda niske kritičnosti.
Dinamika	Jednostavno oblikovanje i kontinuirano refaktoriranje koda izvrsni su za vrlo dinamična okruženja, ali izvor potencijalno skupih naknadnih radova u visoko stabilnim okruženjima.	Detaljni planovi i dizajn problema prije implementacije izvrsni su za vrlo stabilno okruženje, ali izvor skupih preinaka u vrlo dinamičnim okruženjima.
Osoblje	Zahtijeva stalnu prisutnost kritične mase rijetkih stručnjaka (Cockburn razine 2 ili 3). Rizično je koristiti ljude koji nemaju dovoljnu razinu znanja i iskustva (razina 1B).	Potrebna je kritična masa rijetkih stručnjaka (Cockburn razine 2 i 3) tijekom definiranja projekta, ali kasnije mogu raditi i ljudi s manje znanja i iskustva - osim ako je okolina vrlo dinamična.
Organizacijska kultura	Metode su uspješne ako se	Metode su uspješne ako se

	ljudi osjećaju ugodno i imaju veći stupanj slobode.	ljudi osjećaju ugodno te ako su njihove uloge jasno definirane.

Grafički sažetak Tablica 5 prikazan je na Slika 5.



Slika 5. Dimenzije koje utječu na odabir metoda[23]

Ideju da se na istom projektu mogu (i trebaju) koristiti komplementarne prakse iz suprotnog pristupa podržava i novi način razmišljanja koji je iznio Ivar Jacobson u metodama i tehnologiji programskog inženjerstva (eng. *Software Engineering Methods and Technology - SEMAT*)[24]. U Poglavlju 6 navedeni su osnovni pojmovi vezani uz zahtjeve i procese inženjerstva zahtjeva na način kako su definirani u SEMAT-u.

6. Inženjerstvo zahtjevâ

6.1 Zahtjevi

Prema IEEE standardnom rječniku pojmova [25] zahtjev je:

- stanje ili sposobnost potrebna korisniku za rješenje problema ili postizanje cilja
- stanje ili sposobnost koju programski sustav ili komponenta sustavamora zadovoljavati ili posjedovati kako bi bio zadovoljen ugovor, standard, specifikacija, ili neki drugi, formalno definiran dokument
- dokumentirani prikaz (opis) stanja ili sposobnostidefiniranih pod a) ili b).

Zahtjevi programske podrškeopisuju funkcionalnosti i ograničenja programske podrške koji pridonosi zadovoljenju neke potrebe u stvarnom svijetu.Prema sadržaju zahtjevi se klasificiraju kao funkcionalni, nefunkcionalni i domenski zahtjevi [12]:

- funkcionalni zahtjevi opisuju *što*sustav treba raditi:
    - aktivnosti koje mora pružati,
    - kako sustav treba reagirati na određene ulazne poticaje,
    - kako bi sesustav trebao ponašati u određenim situacijama,
    - a u nekim slučajevima, i eksplicitne izjave što sustav ne treba obavljati.
- nefunkcionalni zahtjevi postavljaju ograničenja na sustav ili na proces oblikovanja sustava(kako). Obično se odnose na sustav kao cjelinu i uključuju:
    - vremenska ograničenja,
    - ograničenja u procesu razvoja i oblikovanja programske podrške, i
    - standarde.
- zahtjevi domene proizlaze iz područja primjene sustava, a ne iz specifične potrebe korisnika sustava. Mogu biti novi funkcionalni zahtjevi ili ograničenja na postojeće zahtjeve.Iskazuju specifičnosti sredine, iz čega proizlaze i specifični problemi implicitnosti i razumljivosti:
    - poslovni korisnici ne definiraju eksplicitno ove zahtjevejer, zbog dobrog poznavanjadomene primjene, smatraju da se oni podrazumijevaju;
    - razvojni tim traži detaljan opis zahtjeva jer ne razumije domenu primjene.

U stvarnosti, razlika između različitih vrsta zahtjeva nije tako jasna kao što ove jednostavne definicije sugeriraju.Dio problema u inženjerstvu zahtjeva ima uzrok u nedovoljno jasnom razdvajanju različitih razina opisa. Prema razini detalja razlikuju se:

- zahtjevi korisnika - opisuju funkcionalne i nefunkcionalne zahtjeve na način koji je razumljiv poslovnim korisnicima. Trebali bi opisivati samo vanjsko ponašanje i ograničenja, a ne i dizajn sustava. Pišu se u prirodnom jeziku i (intuitivnim) grafičkim dijagramima. Korištenje prirodnog jezika u opisu korisničkih zahtjeva može rezultirati raznim problemima:
    - pomanjkanje jasnoće – teško je precizno i nedvosmisleno opisati nešto, a da to ostane razumljivo i čitljivo;
    - zbrka u zahtjevima – ne postoji jasno razgraničenje između funkcionalnih i nefunkcionalnih zahtjeva te informacija o ciljevima i dizajnu sustava;
    - stapanje zahtjeva – nekoliko različitih zahtjeva može biti prikazano zajedno kao jedan zahtjev.
- zahtjevi sustava - vrlo su detaljna specifikacija programske podrške. To su korisnički zahtjevi koje je razvojni tim proširio dodatnim detaljima i objašnjenjem kako bi sustav trebaoomogućiti korisničke zahtjeve. Trebali bi opisivati samo vanjsko ponašanje i ograničenja sustava, a ne i kako će sustav biti dizajniran ili implementiran, ali je kod ovako detaljnih specifikacija praktično nemoguće izbjeći sve informacije o dizajnu. Za pisanje zahtjeva sustava koristi se strukturirani prirodni jezik, posebni jezici za oblikovanje sustava, grafički dijagrami i matematička notacija.

Sudionici u procesu prikupljanja i dokumentiranja zahtjeva obično se moraju nositi sa sljedećim vrstama odluka[26]:

- odluke o kvaliteti , npr.: je li zahtjev ne-redundantan, konkretan i razumljiv?
- odluke o prednosti (preferiranju), npr.: koje zahtjeve treba uzeti u obzir za slijedeću inačicu?
- odluke o klasifikaciji, npr.: kojoj temi, komponenti, odnosno grupi zahtjev pripada?
- odluka o svojstvima, npr.: je li procjena potrebnog rada za ovaj zahtjev realna?

Povećanje veličine i složenosti projekata programske podrške sve više otežava donošenje ovakvih odluka.

6.2 Metode prikupljanja zahtjevâ

Engleski izraz *requirementscapturing*(prikupljanje zahtjeva) u stručnoj je literaturi uglavnom zamijenjen izrazom*requirementselicitation* (izlučivanje zahtjevâ) ili u manjoj mjeri *requirementsdiscovery*(otkrivanje zahtjevâ) jer se i na taj način želi pokazati koliko teškog i mukotrpnog rada zahtjeva ova aktivnost[1]. Prikupljanjei razumijevanje korisničkih zahtjeva je teško iz nekoliko razloga[12]:

- sudionici samo vrlo općenito znaju što žele. Teško artikuliraju zahtjeve iliimaju nerealna očekivanja, jer nisu svjesni stvarne cijene zahtjevâ.
- sudionici smatraju da se neki zahtjevi podrazumijevaju, pa ih ne navode eksplicitno;
- različiti sudionici izražavaju zahtjeve na različite načine i često su zahtjevi međusobno konfliktni;
- organizacijski i politički faktori mogu utjecati na zahtjeve (npr. rukovoditelj traži da uvedeni sustav ima značajke koje povećavaju njegov utjecaj u organizaciji);
- ekonomsko i poslovnog okruženje je dinamično, pa tijekom procesa prikupljanja i analizezahtjeva dolazi do promjene tog okruženja koja može uvjetovati pojavu novih sudionika s novim, specifičnim zahtjevima.

Koliko je u praksi zahtjevna aktivnost prikupljanja zahtjeva možda najbolje pokazuje broj i raznovrsnost metoda prikupljanja koje se koriste, kao i činjenica da ni jedna metoda nije idealna za sve situacije, pa se na projektu najčešće kombinira više metoda. Stoga će se ovdje ukratko opisati najčešće korištene metode prikupljanja zahtjevâ.
Metode prikupljanja i dokumentiranja zahtjeva možemo svrstati u četiri osnovne kategorije [29]:

- konverzacijske metode – karakterizira ih verbalni način komunikacije među ljudima. Ove metode uključuju intervjue, radionice, upitnike i ankete, fokus grupe, brainstorming i sl.
- metode opažanja – promatra se radna rutina i stječu se znanja o domeni koje je teško objasniti verbalno. Obično traju dulje od ostalih metoda, ali rezultiraju detaljnim razumijevanjem socijalne / organizacijske kulture, radnih postavki i konteksta rada, problema u postojećim sustavima. Ove metode uključuju metodu analize protokola i etnografsku studiju.

[1] U hrvatskoj je literaturi uobičajen izraz *prikupljanje*, pa će se on koristiti i u ovom radu.

- analitičke metode - bave se analizom prikupljenih informacija putem prije navedenih tehnika. Koriste se za definiranje domene zahtjeva, obilježja korisničkih sučelja, organizacijske politike, standarda, zakonodavstva, specifikaciju naslijeđenih sustava itd.Ove metode uključuju tehniku ponovnog korištenja zahtjeva, metodu ljestvi, metodu repozitorija rešetki i metodu sortiranja kartica.
- sintetičke metode - nazivaju se i suradničke metode. Niti jedna metoda za prikupljanje zahtjeva nije savršena. Umjesto kombinacije pojedinih metoda, sintetičkametodaoblikuje koherentnu cjelinu sustavno kombinirajući razgovor, promatranje i analizu u jednu metodu.Ove metode uključuju tehniku scenarija, prototipiranje i JAD/RAD.

Najčešće korištene metode prikupljanja i dokumentiranja zahtjeva su [28], [29],[30]:

- intervjui – zbog jednostavnosti ovo je najčešće korištena tehnika. Mogu biti strukturiranog (zadan je kontekst i traže se precizni i jasni odgovori) ili otvorenog tipa (ostavljaju prostora osobi koja se ispituje da opisno i opsežno govori o nekoj temi, pri čemu se izbjegava prejudiciranje odgovora). Obično su intervjui kombinacija pitanja otvorenog i zatvorenog tipa. Ovom metodom moguće je:
    - prikupiti veliku količinu informacija;
    - otkriti stavove, mišljenja, osjećaje i ciljeve sudionika projekta;
    - sondirati neku temu u dubinu prateći što osoba govori i prilagođavajući tome pitanja;

  nedostaci su:
    - veliku količinu kvalitativnih podataka teško je analizirati;
    - teško je usporediti različite ispitanike; iskustvo, razumijevanje i pristranost osoba koje se intervjuiraju mogu utjecati na dobivene informacije;
    - teško je svladati vještinu razgovora.
- upitnici i ankete – koriste se kada treba uključiti veći broj ljudi. Prednosti su:
    - brzo prikupljanje informacije od velikog broja ljudi;
    - može se primijeniti na daljinu;
    - omogućuje prikupljanje stavova, uvjerenja, karakteristika;
    - mana je što pretpostavljene kategorije (usmjerena pitanja) ne dozvoljavaju korisnicima iskazivanje stvarnih potreba, a odgovore na pitanja otvorenog tipa je jako teško analizirati.

- promatranje korisnika – promatrač neposredno prati odvijanje poslovnih procesa u promatranoj radnoj sredini. Ova metoda otkriva pojedinosti koje druge metode ne mogu, ali:
    - zahtjeva veliki utrošak vremena;
    - teško je analizirati rezultate s velikim brojem detaljnih informacija;
    - ne govori mnogo o rezultatima predloženih izmjena.
- radionice - koriste se za dogovore, planiranje, dobivanje povratnih informacija, definiranje prioriteta, rješavanje problema, rješavanje sukoba, prezentaciju i analizu napretka i sl. Uključuju korisnike i uklanjaju organizacijske barijere, ali zahtijevaju pažljivu pripremu: ciljevi, dnevni red, distribucija materijala… Za različite vrste radionica (formalne prezentacije, brainstorming, JAD,…) primjenjuju se različita pravila.
- JAD/RAD (Joint/RapidApplicationDevelopment) – zahtjevi se definiraju i specificiraju na radionicama koje uključuju klijente, korisnike i razvojni tim, uz korištenje vizualnih pomagala i tehnika kao što su brainstorming i top-down analize. Ovo je efikasan način za rano definiranje korisničkih potreba ali ima i nedostataka [31]:
    - nelagoda izražavanja vlastitog mišljenja;
    - apsolutna dominacija svega par sudionika.
- brainstorming, fokus grupe – ove tkz. grupne tehnike prikupljanja omogućuju prirodnije interakcije među sudionicima od formalnog intervjua, a mogu rezultirati i kreativnim idejama. Nedostaci su:
    - skupine mogu biti neugodne za sudionike;
    - opasnost od grupnog mišljenja;
    - može pružiti samo površne odgovore na tehnička pitanja;
    - zahtijevaju visoko obučene voditelje.
- slučaj primjene –temelji se na tehnici scenarija iintegralni je dio UML notacije. Svrha je opisati ponašanje sustava i način na koji je sudionik u odnosu sa sustavom. Ova je metoda efikasna za prikupljanje i definiranje zahtjeva sa stajališta interakcija sudionika i sustava, ali je neprikladna za definiranje ograničenja sustava, ne-funkcionalnih zahtjeva i zahtjeva visoke poslovne razine.
- scenariji – opisuju jednu ili više interakcija korisnika sa sustavom. Počinje se sa skicom interakcije, a zatim se razrađuju detalji dok se ne utvrde stvarni zahtjevi.

Ova metoda osobito je korisna za dodavanje detalja u opis strukture zahtjeva i integralni je dio metoda agilnog programiranja [32].

- prototipiranje –je tehnika brze izrade grube inačice željenog sustava ili dijela tog sustava. Prototip ilustrira mogućnosti sustava i olakšava korisnicima shvaćanje interakcije sa sustavom. Prototipovi ponekad daju sliku i ostavljaju utisak kao da su programeri daleko odmakli u svome radu, dajući korisnicima pretjerano optimističan prikaz kompletiranih mogućnosti. Prototipovi se efikasno mogu kombinirati sa drugim pristupima, kao što je npr. JAD.
- etnografija – je tehnika promatranja koja se koristi za razumijevanje društvenih čimbenika i organizacijskih faktora. Promatrač prati dnevnu rutinu i bilježi aktivnosti, bez ometanja svakodnevne rutine. Posebno je korisna za otkrivanje dvije vrste zahtjeva:
    - zahtjeva koji proizlaze iz načina na koji korisnici zaista rade, a ne iz načina na koji bi trebali raditi (kako je propisano).
    - zahtjeva koji proizlaze iz suradnje s drugima i svijesti o aktivnostima drugih ljudi.

  Metoda može otkriti kritične informacije o procesu, koje se često ne mogu otkriti drugim tehnikama prikupljanja zahtjeva, ali ne predstavlja potpuni pristup prikupljanju zahtjeva. Zbog fokusiranja na krajnjeg korisnika, metoda nije prikladna za definiranje organizacijskih zahtjeva i zahtjeva domene, a problem je i identificiranje novih funkcionalnosti koje treba dodati sustavu. Stoga se kombinira s drugim metodama npr. analizom slučajeva primjene.

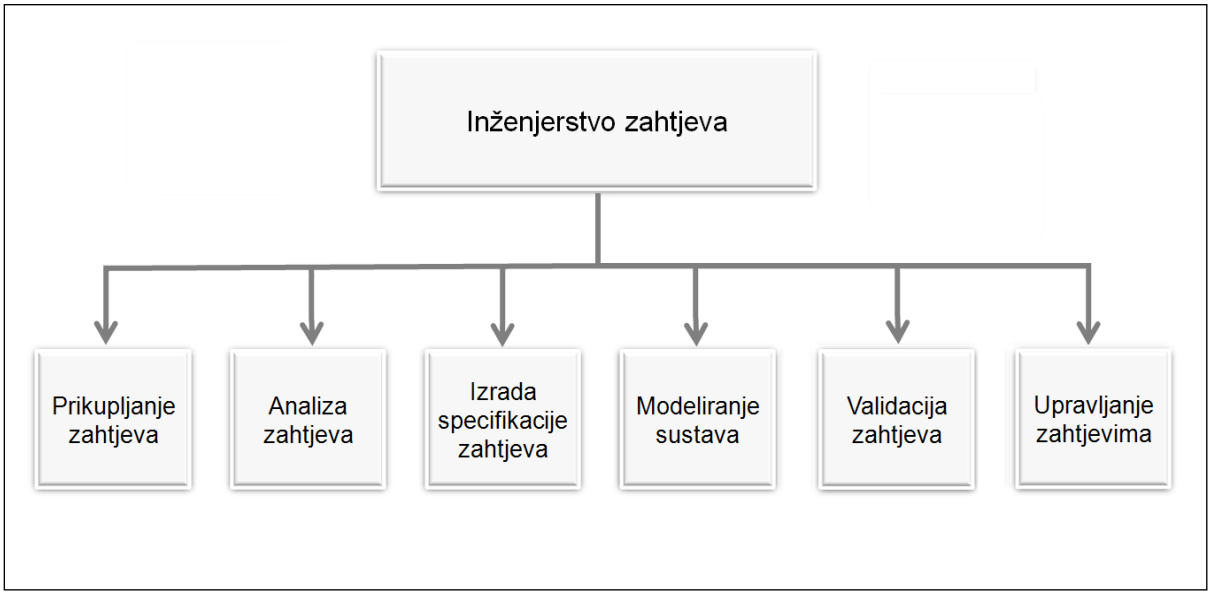
6.3 Aktivnosti inženjerstva zahtjevâ

Inženjerstvo zahtjeva (RE – RequirementsEngineering) definira što sustav programske podrške treba raditi, poželjne i suštinski opasne osobine sustava, kao i ograničenja na rad sustava i na procese razvoja tog sustava programske podrške [12].

Inženjerstvo zahtjeva je integralni dio programskog inženjerstva, jedno od ukupno petnaest područja znanja navedenih u posljednjoj verziji međunarodno priznatog Guide to the Software Engineering Body of Knowledge (SWEBOK V3.0)[27].

Proces u općem smislu je organiziran skup aktivnosti koji vodi nekom cilju, a proces inženjerstva zahtjeva je skup aktivnosti koje generiraju i dokumentiraju zahtjeve[28]:

- prikupljanje zahtjeva - moguće je realizirati kroz pitanja, ispitivanje ponašanja, modeliranje procesa, demonstriranje sličnih sustava;
- analizu zahtjeva i dogovaranje – utvrđuju senekonzistentnosti, konflikti, propušteni zahtjevi i preklapanja, a potom se zahtjevi usuglašavaju sa svim sudionicima i definiraju se prioriteti. Ova faza uključuje evidentiranje zahtjeva kroz modeliranje ponašanja sustava (dijagrami, izrada prototipova i sl.), amoguć je i povratak u fazu prikupljanja zahtjevâ;
- izradu specifikacije zahtjeva - određuje se koji će se dijelovi zahtijevanog ponašanja implementirati u sustav te se dokumentira ponašanje predloženeprogramske podrške;
- modeliranje sustava – razvijaju se apstraktni modeli sustava koji su dio detaljne specifikacije sustava;
- validaciju zahtjeva – provjerava se odgovara li specifikacija zahtjeva onom što korisnik zaista želi u finalnom proizvodu;
- upravljanje zahtjevima – definiraju se aktivnosti koje će osigurati kvalitetno praćenje promjena u zahtjevima.



Slika 6. Aktivnosti inženjerstva zahtjevâ

6.4 Specifikacija zahtjeva programske podrške

Osobine dobre specifikacije zahtjeva programske podrške [33]:

- ispravnost – zahtjevi su točno definirani;

- nedvosmislenost (jednoznačnost) – ne postoji mogućnost višestrukih tumačenja istih zahtjevâ;
- cjelovitost – uključeni su svi zahtjevi;
- dosljednost – međusobno usklađeni zahtjevi;
- kategoriziranje po važnosti i/ili stabilnosti;
- provjerljivost (dokazivost – koji se može dokazati);
- modificiranost – jednostavno je mijenjanje i prilagođavanje;
- sljedivost – jednostavno je praćenje zahtjeva prema drugim dokumentima.

6.5 Osobine (ne) uspješnih projekata

Prekinute projekte programske podrške i projekte koji su prekoračili rokove i/ili proračun i/ili nisu na zadovoljavajući način realizirali sve tražene funkcionalnosti karakterizira većina ili sve slijedeće osobine (vezane uz definiranje zahtjevâ) [34]:

- vizija i opseg projekta nisu jasno definirani;
- klijenti su prezaposleni za razradu zahtjeva sa članovima razvojnog tima;
- posrednici (razvojni menadžeri, proizvodni menadžeri itd.) ne prikazuju točno potrebe korisnika (iako to tvrde);
- zahtjevi postoje u glavama „stručnjaka“ i nikada nisu formalno zapisani;
- klijenti tvrde da su svi zahtjevi kritični, pa ne postavljaju prioritete;
- razvojni tim otkriva nedosljednosti te nedostatak informacija tek u fazi kodiranja;
- komunikacija klijent – razvojni tim fokusira se na korisničko sučelje, a ne na to što korisnik mora obavljati;
- klijenti odustanu od zahtjevâ, a zatim traže izmjene/ dopune;
- obim projekta se povećava prihvaćanjem promjena, ali rokovi „klize“ jer nisu osigurani dodatni resursi niti su neke funkcionalnosti izostavljene;
- neažurna tj. izgubljena evidencija o promjenama zahtjevâ, pa ni klijent ni razvojni tim ne zna točan status svih zahtijevanih promjena;
- klijenti zahtijevaju određene funkcionalnosti, ali ih po implementaciji nitko ne koristi;
- specifikacija zahtjeva je realizirana u potpunosti, ali je klijent nezadovoljan.

Greške u definiranju zahtjeva ne mogu se u potpunosti izbjeći, ali ih se može reducirati [4]:

- učinkovitijim prikupljanjem zahtjevâ;
- provjerom prikupljenih zahtjeva sa sudionicima (klijentima i krajnjim korisnicima);
- boljom organizacijom i dokumentiranošću zahtjevâ;
- postojanjem dostupnog repozitorija zahtjeva i njegovanjem poboljšane komunikacije među članovima tima;
- poboljšanim procesima izvješćivanja;
- testiranjem temeljenim na zahtjevima i sljedivošću zahtjevâ.

Višestruke su dobrobiti dobro definiranih zahtjevâ:

- manji broj pogrešaka u zahtjevima;
- manji opseg ponovne izrade (rework);
- manje nepotrebnih funkcionalnosti;
- niži troškovi unaprjeđenja;
- brži razvoj;
- manje problema (nesuglasica) u komunikaciji (manje pogrešnih interpretacija);
- manja mogućnost proširenja opsega projekta;
- manja vjerojatnost „probijanja“ rokova i proračuna i/ili isporuke neodgovarajućih funkcionalnosti;
- točnija procjena kod sustavnog testiranja;
- veće zadovoljstvo klijenata i članova razvojnog tima.

7. Metode i tehnologija programskog inženjerstva (SEMAT)

7.1 Uvod u SEMAT

Danas je razvoj programske podrške više zanatska,a manje inženjerska disciplina. Temelji se na dokazanoj praksi umjesto na znanstvenom temelju.SEMAT (eng. *Software Engineering Methods and Technology* - metode i tehnologija programskog inženjerstva)potiče proces preustroja programskog inženjerstva temeljenog na teoriji, dokazanim načelima i najboljim praksama[35].

SEMAT je zajednica ljudi (oko 2000), tvrtki (uključujući ABB, Ericsson, Fujitsu UK, Huawei i TCS) i sveučilišta širom svijeta (uključujući Carnegie Mellon West, Free University of Bozen Bolzano, Korea Advanced Institute of Science and Technology, KTH Royal Institute of Technology, Tsinghua University i University of Twente), koja podržava inicijativu za stvaranje zajedničkog jezika, jezgre ili temelja programskog inženjerstva - nešto što je do sada nedostajalo.

Bez obzira na kôd koji je napisan, programsku podršku koja se gradi, rješenje koje se konstruira, metode ili način rada zaposlenih ili uključene organizacije, postoji zajedničko područje - jezgra neophodnih elemenata koji su uvijek prisutni u bilo kojem programskom pothvatu.Ova jezgra bitnih elemenata naziva se i suština programskog inženjerstva (eng. *Essence*).

Jedan od prvih i najistaknutijih rezultata SEMAT zajednice je standard za rad s metodama programskog inženjerstva nazvan Essence, kojeg je OMG usvojio u lipnju 2014. U prosincu 2015.g izdana je posljednja verzija standarda *Essence - jezgra i jezik za metode programskog inženjerstva 1.1*[36].Ovaj dokument pruža sveobuhvatne definicije i opise jezgre i jezika za metode programskog inženjerstva.Jezgra je skup elemenata koji sačinjavaju zajednički temelj za opisivanje projekata programskog inženjerstva.Jezgra je opisana pomoću jezika koji definira apstraktnu sintaksu, dinamičku semantiku, grafičku sintaksu i tekstualnu sintaksu.

Za razumijevanje SEMAT-a važno je prepoznati i ono što Essence nije. Essence nije nova jedinstvena metodologija, niti novi meta-model, ni nova disciplina ni novi jezik modeliranja. Nije ni trik da bi ljudi gradili ili kupovali više alata. To je jednostavno mapa onoga što već postoji (npr. timovi i projekti), onoga što se već radi (npr. specificiramo i implementiramo), vještina kojesu potrebne (npr. programiranje i testiranje) i onoga što se

već proizvodi (npr. sustavi programske podrške). Fokus je na suštini onoga što je potrebno za uspješan razvoj programske podrške - bez obzira na način na koji radimo ili stupanj dokumentacije.

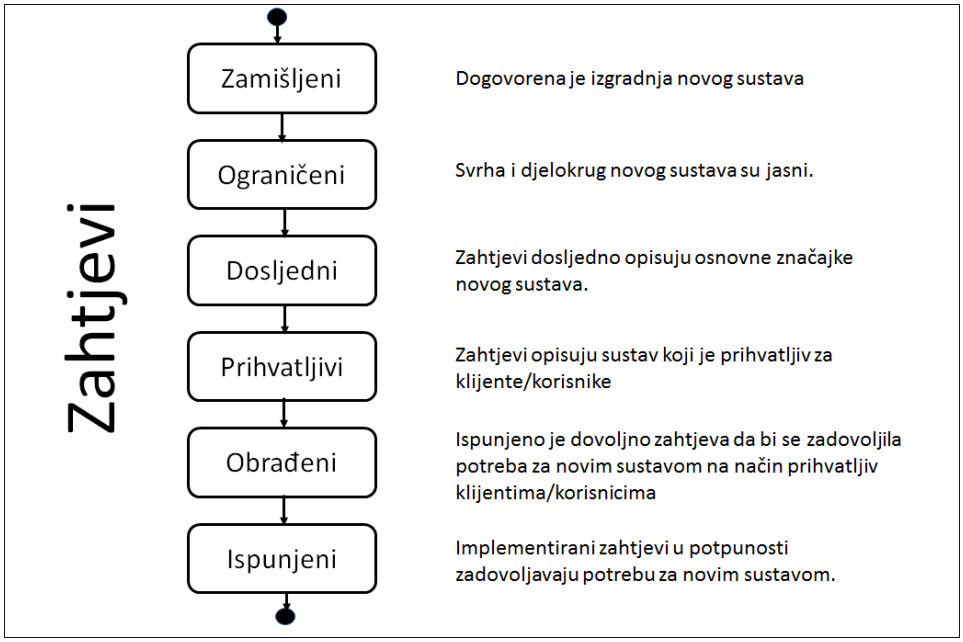
7.2 Inženjerstvo zahtjevau SEMAT-u

Jezgru sačinjavaju tri vrste elemenata - alfe, prostoriaktivnosti i kompetencije[36]:

- Alfe su subjekti u programskom inženjerstvu čiju evoluciju želimo razumjeti, pratiti, usmjeravati i kontrolirati. Alfe imaju stanja i kontrolne popise kako bismo mogli pratiti napredak. Stavke kontrolnog popisa mjere kritične ishode, od kojih svaki doprinosi postizanju određenog alfa stanja. Alfe, njihova stanja i kontrolni popisi mogu se koristiti za procjenu napretka projekta neovisno o primijenjenoj metodologiji i praksi[37], [38].Essence identificira sedam alfa subjekata: sudionici, mogućnost, zahtjevi, tim, način rada, rad i programski sustav.
- Prostori aktivnosti su mjesta rezervirana za skup aktivnosti kojima se ostvaruje napredovanje alfe. Rezervirano mjesto može se sastojati od nula do mnogo aktivnosti. Essence ne propisuje određene aktivnosti, već samo zamjenska mjesta za aktivnosti specifične za projekt. Aktivnosti se određuju u kontekstu određenog projekta i odabranog načina rada tima. Na primjer, veće razine istovremenih aktivnosti javljaju se u implementacijama agilnih metoda, a veći broj sekvencijalnih aktivnosti u implementacijama vodopadne metode.
- Kompetencije su karakteristike članova tima koje predstavljaju ključne sposobnosti potrebne za obavljanje poslova programskog inženjeringa. To su: *zastupanje klijenata/korisnika* (prikupljanje zahtjeva, komuniciranje, točan prikaz stavova), *analiza* (razumijevanje mogućnosti i potreba klijenata/korisnika te njihovo pretvaranje u zahtjeve),razvoj (dizajniranje i programiranje učinkovitih sustava prema dogovorenim standardima i normama), testiranje (testiranje sustava, verifikacija zahtjeva), vodstvo (inspiriranje i motiviranje suradnika) i upravljanje (koordiniranje, planiranje i praćenje rada razvojnog tima).Kompetencija se ocjenjuje razinom sposobnosti u rasponu od minimalne do maksimalne razine. Predložene razine kreću se od 0 (asistiranje) do 5 (inovacija).

Zahtjevi su jedan od alfa subjekata. Tijekom razvoja programske podrške zahtjevi prolaze kroz nekoliko stanja:zamišljeni, ograničeni, dosljedni, prihvatljivi, obrađeni i ispunjeni (Slika 7).Ta stanja opisuju kako evoluira razumijevanje tima o onome što

predloženi sustav mora raditi: od inicijalne ideje, preko razvoja zahtjeva do njihovog ispunjenja isporukom upotrebljivog sustava programske podrške.Razumijevanje trenutnog i željenog stanja zahtjeva može svima na projektu pomoći da razumiju što sustav treba učiniti i koliko je blizu dovršenosti.



Slika 7. Stanja zahtjeva

Kontrolni popis (
Tablica 6) pomaže u procjeni stanja zahtjeva i ocjeni napretka projekta.

Tablica 6. Kontrolni popis za zahtjeve

Stanje zahtjeva	**Spisak provjera**
Zamišljeni	Klijenti su suglasni da treba stvoriti novi sustav. Identificirani su korisnici novog sustava. Identificirani su klijenti koji će financirati početni rad na novom sustavu. Postoji jasna poslovna prilika za provedbu novog sustava.
Ograničeni	Identificirani su klijenti uključeni u razvoj novog sustava. Klijenti se slažu oko svrhe novog sustava. Jasno je što je uspjeh za novi sustav. Klijenti razumiju opseg predloženog rješenja. Dogovoreno je kako će se opisati zahtjevi. Mehanizmi za upravljanje zahtjevima su postavljeni.

	Shema prioriteta je jasna. Ograničenja su identificirana i razmotrena. Pretpostavke su jasno navedene.
Dosljedni	Zahtjevi su prikupljeni i podijeljeni s timom i klijentima. Podrijetlo zahtjeva je jasno. Razlog iza zahtjeva je jasan. Konfliktni zahtjevi su identificirani i razrješeni. Iz zahtjeva je jasno koje će bitne značajke sustava biti isporučene. Može se objasniti najvažniji scenarij uporabe sustava. Prioritet zahtjeva je jasan. Razumljiv je utjecaj implementacije zahtjeva. Razvojni tim razumije što mora biti isporučeno i pristaje to isporučiti.
Prihvatljivi	Klijenti prihvaćaju da zahtjevi opisuju prihvatljivo rješenje. Stopa promjene ugovorenih zahtjeva relativno je niska i pod kontrolom. Vrijednost dobivena implementacijom zahtjeva je jasna. Jasno je koji su dijelovi poslovne prilike zadovoljeni zahtjevima. Zahtjevi se mogu testirati.
Obrađeni	Implementirano je dovoljno zahtjeva da rezultirajući sustav bude prihvatljiv klijentima. Klijenti prihvaćaju zahtjeve koji točno opisuju što sustav radi, a što ne. Skup implementiranih zahtjeva osigurava jasnu vrijednost klijentima. Klijenati prihvaćaju sustav s implementiranim zahtjevima kao vrijedan rad.
Ispunjeni	Klijenti prihvaćaju da zahtjevi točno i potpuno opisuju potrebu za novim sustavom. Nema izuzetih zahtjeva, pa sustav u potpunosti zadovoljava zahtjeve. Klijenti prihvaćaju da sustav u potpunosti zadovoljava zahtjeve.

8. Pregled istraživanja iz područja inženjerstva zahtjevâ

Znanstveni projekti zahtijevaju ponovljivi transparentan proces pregleda literature, pri čemu se nastoji eliminirati utjecaj osobe koja provodi pretraživanje na rezultate tog pretraživanja. Sustavan pregled literatureomogućava temeljitost,potpunost i kvalitetu pretraživanja[39],[40],[41],a propisuje tri stadija rada:

- planiranje pregleda – definiraju se ciljevii područje pretraživanja, baze podataka, te kriteriji isključivanja (uključivanja);
- provođenjepregleda –definiraju se ključne riječi, evaluiraju se i sintetiziraju dobiveni podaci;
- dokumentiranje rezultata pregleda.

U ovom radu, sustavan pregled literaturekorišten jes ciljem identifikacije metodaza prikupljanje zahtjeva te nedostataka i prednosti tihmetoda.Pretraživanje nije ograničeno na računalne znanosti (ComputerScience), jer se zanimljive metode mogu naći i unutar drugih područja, tim prije što ljudski faktor ima veliki utjecaj na prikupljanje zahtjevâ.
Za ključnu riječpretraživanja odabranje izraz „requirements elicitation“. Korištenje engleski izraz jer se većina relevantne literature iz područja inženjerstva zahtjeva objavljuje na engleskom jeziku, pa su na tom jeziku i pretraživane baze podataka.
Relevantni izvori pretraživanja uključivali su:

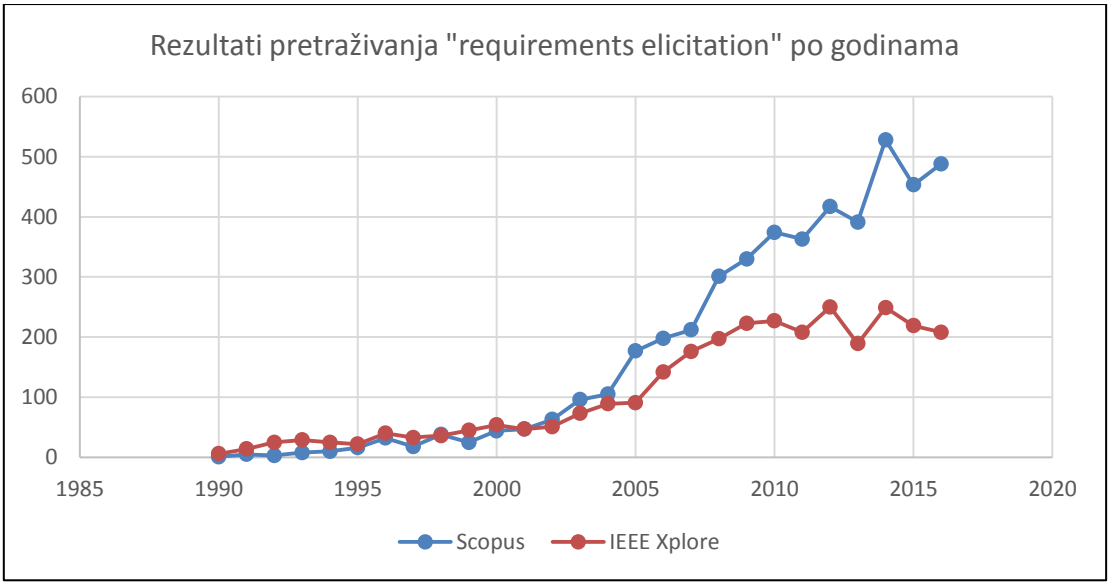
- članke
- zbornike konferencija
- knjige
- publikacije istraživačkih ustanova i tehnoloških tvrtki.

Pretraživane su posljednje tri godine (2015., 2016. i 2017.), kako bi se identificiralo trenutno zanimanje znanstvene zajednice za prikupljanje i obradu korisničkih zahtjevâ.
Rezultati pretraživanja za “requirements elicitation” (od (uključivo) 2015.g, sva polja, sva područjaznanosti), uz navedeni broj radova pronađenih u odgovarajućoj bazi podataka, dani su u Tablica 7.Pregled znanstvene literature u posljednje tri godine pokazuje postojanost u zanimanju znanstvene zajednice za prikupljanje i obradu korisničkih zahtjeva.
Rezultati istogpretraživanje (“requirements elicitation”, sva polja, sva područjaznanosti) u bazama Scopus i IEEE Xplore po godinama od 1990. do 2016. prikazanasu grafom na Slika 8. Rezultati za 2017. g. su ispušteni, jer je tek travanj i rezultati nisu konačni.

Vidljivo je da je posljednjih petnaestak godina višestruko povećan broj radova koji se bave prikupljanjem i obradom zahtjevâ.

Tablica 7. Rezultati pretraživanja baza podataka

	Baza podataka	Broj radova
1.	Scopus	1.059
2.	ACM	39
3.	Google Scholar	16.600
4.	ScienceDirect	249
5.	IEEE Xplore	440



Slika 8. Rezultati pretraživanja u bazama Scopus i IEEE Xplore

Područje istraživanja obuhvaća različita područja: od pregleda metoda na području prikupljanja zahtjeva [42], [43], dokumentiranja nefunkcionalnih zahtjeva[44], alata za otkrivanje i rješavanje konfliktnih zahtjeva [45], metode za generiranje dijagrama slučaja primjene iz postojeće aplikacije [46], metode za odabir najbolje tehnike prikupljanja.zahtjeva[43], [47], formalizacije lista zahtjeva u agilnom razvoju [48], pristupa za otkrivanje dvosmislenosti u tekstualnom opisu zahtjeva[49], do rasprave o važnosti inženjerstva zahtjeva[50].

Pregled literature [42]pokazao je da se identifikacija sudionika, prikupljanje potreba korisnika te izbor metode prikupljanja, smatraju ključnim aktivnostima u procesu prikupljanja. U istom radu predložena je metoda prikupljanja, koja pored spomenute tri,

uključuje još šest aktivnosti, u nastojanju da se obuhvate sve aktivnosti neophodne za definiranje točnih, sveobuhvatnih i nedvosmislenih zahtjeve. Za poboljšanje kvalitete zahtjeva, autori predlažu aktivnosti praćenja te pročišćavanja zahtjeva. Metoda nije validirana u praksi.

Sustavnim pregledom literature [43]identificirane su dokumentirane metode prikupljanja zahtjeva i analiziran je njihov sustavni metodološki proces.Predložena je metodologija prikupljanja zahtjeva pogodna za ICTD (eng. *Information and Communication Technologies and Development* - Informacijske i komunikacijske tehnologije i razvoj) projekte. Predloženi pristup pomaže programeru u boljem razumijevanju procesa prikupljanja zahtjeva te u odabiru odgovarajuće metode i alata.

Rad [44]se bavi razumijevanjem prirode nefunkcionalnih zahtjeva te načinom njihova dokumentiranja. Nefunkcionalni zahtjevi se često nadograđuju u procesu razvoja programske podrške ili se prikupljaju paralelno, ali odvojeno od funkcionalnih zahtjeva. To ima za posljedicu da se nefunkcionalnim zahtjevima uglavnom upravlja implicitno s malo ili bez ikakve analize posljedica, što dugoročno može rezultirati visokim troškovima održavanja.Temeljem analize i klasifikacije 530 nefunkcionalnih zahtjeva iz 11 specifikacija, autori su zaključili da 75% zahtjeva koji su označeni kao "nefunkcionalni" u razmatranim industrijskim specifikacijama opisuju ponašanje sustava, a samo 25% opisuje svojstva sustava. Drugim rječima, mnogi tkz. nefunkcionalni zahtjevi zapravo nisu nefunkcionalni i trebaju se prikupljati na isti način kao i funkcionalni.

U radu [45]opisan je alat koji omogućava lakše otkrivanje i rješavanje konfliktnih zahtjeva, nazvan grafikon proširenog cilja (eng. *Extended Goal Graph*). Zahtjevi i njihova međuzavisnost prikazani su kao stablo simetrične strukture. Alat omogućava sudionicima da sagledaju širu sliku i pronađu implicitna rješenja za ciljani sustav.

U fazi prikupljanja zahtjeva, dijagrami slučaja primjene (eng. Use Case Diagram) često se koriste za opisivanje zahtjeva korisnika. Problem je što ih kreiraju članovi razvojnog tima, pa ovi dijagrami ne predstavljaju uvijek odgovarajuće zahtjeve korisnika. Poželjno bi bilo da korisnici sami opišu zahtjeve, ali oni najčešće nemaju dovoljno znanja. U praksi najčešće već postoje različiti programi u domeni, pa korisnici mogu pregledati postojeće aplikacije kako bi pronašli funkcije i interakcije koje su slične njihovim zahtjevima.U radu [46]predlaže se metoda koja, u pet koraka,generiradijagram slučaja primjene iz postojeće aplikacije.

U radu [47]je predložen model koji olakšava odabir najbolje tehnike prikupljanja zahtjeva u određenoj situaciji. Model uzima u obzir značajke projekta (složenost projekta, veličina

zahtjeva, vrsta projekta, proračun), specifičnu situaciju na projektu (domena, klijenti, analitičari, politika tvrtke, korisnici) te prednosti i mane pojedinih tehnika.

Lista zahtjeva (eng.*product backlog*) predstavlja jedinstveni izvor zahtjeva za bilo kakve izmjene na proizvodu. Lista zahtjeva navodi sve značajke, funkcije, zahtjeve, poboljšanja i popravke koje će se izvršiti na proizvodu u budućim verzijama[48].Potreba za formalizacijom lista zahtjeva povećava se s veličinom projekta.Rad [48]opisuje uzorke za izgradnju i strukturu lista zahtjeva u agilnom razvoju koristeći rezultate prikupljanja i analize zahtjeva. Ciljana publika ovog rada prvenstveno su poslovni analitičari, ali uključuje i druge uloge koje sudjeluju u definiranju, pročišćavanju i upravljanju zahtjevima u agilnom razvoju, kao što su vlasnici proizvoda, programeri i arhitekti sustava.

Zahtjevi se najčešće opisuju tekstualno, rečenicama prirodnog jezika. Prirodni jezik je u svojoj osnovi dvosmislen, što dovodi do problema u kasnijem tumačenju zahtjeva. U radu[49] prezentiran je pristup za otkrivanje dvosmislenosti u tekstualnom opisu zahtjeva pomoću alata za analizu prirodnog jezika.Svrha ove analize je procijeniti kvalitetu zahtjeva obzirom na karakteristike kao što su nedvosmislenost, potpunost i razumljivost.

U radu [50]objašnjeni su osnovni pojmovi iz inženjerstva zahtjeva, kao i važnost ove faze programskog inženjerstva u razvoju programske podrške. Vrste zahtjeva koji se mogu prikupiti za razvoj programskog proizvoda su: poslovni zahtjevi, korisnički zahtjevi, zahtjevi sustava te funkcionalni i nefunkcionalni zahtjevi. Aktivnosti inženjerstva zahtjeva dijele se u dvije osnovne grupe: razvoj zahtjeva i upravljanje zahtjevima. Razvoj zahtjeva obuhvaća aktivnosti prikupljanja, analize, izrade specifikacije, modeliranja sustava te validacije zahtjeva.

Većina radova nema sustavni pristup, već se bavi pojedinim tehnikama prikupljana zahtjeva ili primjenom metoda u posebnim uvjetima, a zanemaruje se i uloga klijenata/korisnika.

9. Zaključak

Sve brži tehnološki razvoj nije značajnije utjecao na uspješnost projekata programske podrške. Troškovi obustavljenih projekata dosežu desetke milijardi dolara na godišnjoj razini. Većina problema uzrokovana je nejasnim, nepotpunim, dvosmislenim i pogrešnim zahtjevima.

Na kvalitetu zahtjeva utječu aktivnosti inženjerstva zahtjeva, ali glavni problem je komunikacija sudionika na projektu. Strukturne i agilne metode razvoja koriste iste tehnike prikupljanja zahtjeva, pa su isti i problemi. Agilne metode imaju nešto veći, iako nedovoljan, postotak uspješnih projekata, jer je (predstavnik) korisnika na licu mjesta. Ali, agilne metode nisu pogodne za sve vrste projekata razvoja programske podrške.

SEMAT definira šest stanja koja opisuju kako evoluira razumijevanje tima o onome što predloženi sustav mora raditi. Kontrolni popis pomaže u procjeni stanja zahtjeva i ocjeni napretka projekta.

Broj znanstvenih radova koji se bave prikupljanjem i obradom korisničkih zahtjeva višestruko je povećan u posljednjih petnaestak godina. Većina radova nema sustavni pristup, već se bavi pojedinim tehnikama prikupljana zahtjeva ili primjenom metoda u posebnim uvjetima, a zanemaruje se i uloga klijenata/korisnika.

LITERATURA

[1] https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf, *The Standish Group: The Chaos Report (1994)*, [Pristup:06-veljače-2017.]

[2] https://www.infoq.com/articles/standish-chaos-2015, *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*, [Pristup: 06-veljače-2017.]

[3] F. T. Sheldon, K. M. Kavi, R.C. Tausworth, J. T. Yu, R. Brettschneider, W. W. Everett, *Reliability Measurement from Theory to Practice*, IEEE Software, Vol. 10, No. 4, pp. 13-20, July / August 1992., doi:10.1109/52.143095

[4] D. Leffingwell, *Calculating the return on investment from more effective requirements management*, American Programmer, Vol. 10, No. 4, pp. 13-16, April 1997.

[5] S. Beecham, T. Hall, A. Rainer, *Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis, Empirical Software Engineering*, Vol. 8, No. 1, pp. 7-42, March 2003, doi:10.1023/A:1021764731148

[6] B. Davey, K. Parker, *Requirements elicitation problems: A literature analysis*, Issues in Informing Science and Information Technology, Vol.12, pp. 71-82, 2015.

[7] A. Hussain, E. O. C. Mkpojiogu, F. M. Kamal, *The Role of Requirements in the Success or Failure of Software Projects*, International Review of Management and Marketing, Vol. 6, No. 7S (Special Issue), pp.306-311, 2016.

[8] S. Ahmad, S. A. Asmai, N. A. Rosmadi, *A Significant Study of Determining Software Requirements Defects: A Survey*, 14th International Conference on Applied Computer and Applied Computational Science (ACACOS '15), pp. 117-122, 2015.

[9] B. Boehm, V.R. Basili, *Software defect reduction top 10 list*, IEEE Computer, Vol. 34, No. 1, pp. 135–137, January 2001.

[10] B. Boehm, *A View of 20th and 21st Century Software Engineering*, ICSE'06, Shanghai, China, pp. 12-29, May 20–28, 2006.

[11] K. Pohl, *Process-Centered Requirements Engineering*, John Wiley and Sons, New York, 1996.

[12] Sommerville, *Software Engineering*, 8th Edition, Pearson Education Limited, Harlow, 2007.

[13] http://www.featuredrivendevelopment.com/node/614, *Feature Driven Development*, [Pristup: 16-siječnja-2017.]

[14] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, et al., *The agile manifesto*, http://www.agilealliance.org/the-alliance/the-agile-manifesto/, 2001, [Pristup: 26-siječnja-2017.]

[15] L. Williams, *Agile software development methodologies and practices*, Advances in Computers, Vol. 80, pp. 1-44, 2010.

[16] A. Sillitti, G. Succi, *Requirements engineering for agile methods engineering and managing software requirements*, Part 2, Springer Berlin Heidelberg, pp. 309-326, 2005. DOI: 10.1007/3-540-28244-0_14

[17] I. Inayat, L. Moraes, M. Daneva, S. S. Salim, *A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed*, XP'15: Scientific Workshop Proceedings of the XP2015, Article No. 6, May 25-29, 2015. doi>10.1145/2764979.2764985

[18] W. Helmy, A. Kamel, O. Hegazy, *Requirements engineering methodology in agile environment*, IJCSI '12, Vol. 9, Issue 5, No 3, pp.293-300, September 2012. ISSN (Online): 1694-0814

[19] J. Nawrocki, M. Jasiński, B. Walter, A. Wojciechowski, *Extreme programming modified: embrace requirements engineering practices*, Proceedings of the IEEE

Joint International Conference on Requirements Engineering, IEEE CS Press, pp. 303–310, 2002.
[20] M. Poppendieck, T. Poppendieck, *Lean software development*, Addison-Wesley, 2003.
[21] P. Rodríguez, J. Markkula, M. Oivo, J. Garbajosa, *Analyzing the drivers of the combination of lean and agile in software development companies*, PROFES ’12: Proceedings of the 13th international conference on Product-Focused Software Process Improvement, pp. 145–159, 2012. doi>10.1007/978-3-642-31063-8_12
[22] M. Holm, C. R. Jakobsen, *Systematic, Building an Agile Enterprise with Lean Culture: Fast and Reliable Service to Customers*, https://systematic.com/media/282239/Building_an_Agile_Enterprise_with_Lean_Culture.pdf, [Pristup: 30-travnja-2017.]
[23] B. Boehm, R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley, October 2009.
[24] D. O'Neill, *The way forward: A strategy for harmonizing agile and CMMI*, CrossTalk, The Journal of Defense Software Engineering, Vol.29, No.4, pp. 4-9,·July/August 2016.
[25] The Institute of Electrical and Electronics Engineers, *IEEE Standard Glossary of Software Engineering Terminology: IEEE Std. 610.12-1990*, The Institute of Electrical and Electronics Engineers, New York, 1990.
[26] Felfernig, M. Schubert, M. Mandl, F. Ricci, W. Maalej, *Recommendation and decision technologies for requirements engineering*, RSSE '10: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, pp.11-15, 2010.
[27] IEEE Computer Society , P. Bourque , R. E. Fairley, eds,*Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0*, IEEE Computer Society Press, Los Alamitos,California, 2014.
[28] R.S. Pressman, *Software Engineering: A Practitioner’s Approach*, 5th Edition, McGraw-Hill, New York, 2001.
[29] Sajid, A. Nayyar, A. Mohsin, *Modern Trends Towards Requirement Elicitation*, NSEC '10: Proceedings of the 2010 National Software Engineering Conference, pp.1-10, October 2010.
[30] Sommerville, P. Sawyer, *Requirements Engineering: A good practice guide*, John Wiley and Sons, Chichester 1997.
[31] J. Goguen, C. Linde, *Techniques for Requirements Elicitation*, Proc. RE’93 - First IEEE Symposium on Requirements Engineering, San Diego, pp. 152-164, 1993.
[32] H. Obendorf, M. Finck, *Scenario-based usability engineering techniques in agile development processes*, CHI EA '08: CHI '08 extended abstracts on Human factors in computing systems, April 2008.
[33] The Institute of Electrical and Electronics Engineers, *IEEE Recommended Practice for Software Requirements Specifications: IEEE Std 830-1998*, The Institute of Electrical and Electronics Engineers, New York, 1998.
[34] K.E. Wiegers, *Software Requirements*, 2nd Edition, Microsoft Press, Redmond, 2003.
[35] http://semat.org/, *SEMAT*, [Pristup: 06-veljače-2017.]
[36] http://semat.org/essence-1.1, *Essence - Kernel and Language for Software Engineering Methods (Essence 1.1)*, December 2015. [Pristup: 06-veljače-2017.]
[37] I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence and S. Lidman, *The Essence of Software Engineering: Applying the SEMAT Kernel*, Pearson Education Inc., 2013.
[38] I. Jacobson, P. Ng, I. Spence and P. E. McMahon, *Major-League SEMAT-Why Should an Executive Care?*, Communications of the ACM, Vol.57, No.4, pp. 44-50, 2014.

[39] Tranfield, D. Denyer, P. Smart, *Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic Review*, British Journal of Management, Vol. 14, pp. 207-222, September 2003.
[40] M. Petticrew, H. Roberts, *Systematic Reviews in the Social Sciences: A Practical Guide*, Blackwell Publishing Ltd, Oxford, United Kingdom, 2006.
[41] http://pareonline.net/pdf/v14n13.pdf, *Practical Assessment, Research & Evaluation: A Guide to Writing the Dissertation Literature Review*, [Pristup: 06-prosinca -2011.]
[42] H. Bani-Salameh, N. Al jawabreh, *Towards a Comprehensive Survey of the Requirements Elicitation Process Improvements*, IPAC '15: Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, November 23-25, 2015. DOI: http://dx.doi.org/10.1145/2816839.2816872
[43] M. M. Hasan, *ICTD systems development: analysis of requirements elicitation approaches*, ICTD '15: Proceedings of the Seventh International Conference on Information and Communication Technologies and Development, May 15 - 18, 2015. doi>10.1145/2737856.2737886
[44] J. Eckhardt, A. Vogelsang, D. M. Fernández, *Are "non-functional" requirements really non-functional?: an investigation of non-functional requirements in practice*, ICSE '16: Proceedings of the 38th International Conference on Software Engineering, pp. 832-842, 2016. doi>10.1145/2884781.2884788
[45] N. Kushiro, T. Shimizu, T. Ehira, *Requirements Elicitation with Extended Goal Graph*, 20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems: KES2016, pp. 1691 – 1700, 5-7 September 2016. doi: 10.1016/j.procs.2016.08.217
[46] J. Shirogane, *Generation of Use Cases for Requirements Elicitation by Stakeholders*, Information Science and Applications 2017, Lecture Notes in Electrical Engineering 424, DOI 10.1007/978-981-10-4154-9_65
[47] H. Al-Zawahreh, K. Almakadmeh, *Procedural Model of Requirements Elicitation Techniques*, IPAC '15: Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, November 23 - 25, 2015. doi>10.1145/2816839.2816902
[48] L. B. Hvatum, R. Wirfs-Brock, *Patterns to Build the Magic Backlog*, EuroPLoP '15: Proceedings of the 20th European Conference on Pattern Languages of Programs, July 08 - 12, 2015. doi>10.1145/2855321.2855334
[49] A. Fantechi, S. Gnesi, L. Semini, Ambiguity Defects as Variation Points in Requirements, VaMoS ’17: 11th International Workshop on Variability Modelling of Software-intensive Systems, February 01-03, 2017. http://dx.doi.org/10.1145/3023956.3023964
[50] V. Shukla, D. Pandey, R. Shree, *Requirements Engineering: A Survey*, Communications on Applied Electronics (CAE), Vol. 3, No.5, pp. 28-31, November 2015.

PRILOZI

Prilog I. Kazalo slika

Prilog II. Kazalo tablica