Utvrđivanje zahtjeva u programskom inženjerstvu

Korenić, Francesca

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komuniakcijske tehnologije

FRANCESCA KORENIĆ

UTVRĐIVANJE ZAHTJEVA U PROGRAMSKOM INŽENJERSTVU

Završni rad

Pula, rujan 2016.

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

FRANCESCA KORENIĆ

UTVRĐIVANJE ZAHTJEVA U PROGRAMSKOM INŽENJERSTVU

Završni rad

JMBAG: 0242023468; redoviti student

Studijski smjer: Sveučilišni preddiplomski studij Informatika

Kolegij: Softversko inžinjerstvo

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan 2016.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Francesca Korenić, kandidat za prvostupnika infotmatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 21. rujna 2016.

IZJAVA

o korištenju autorskog djela

Ja, Francesca Korenić dajem odobrenje Sveučilištu Jurja Dobrileu Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Utvrđivanje zahtjeva u programskom inženjerstvu“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu sa Zakonom o autorskom pravu i drugim srodnim pravimai dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupaznanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 21. rujna 2016.

Potpis

_____

Sažetak

U ovom završnom radu, čija je tema „Utvrđivanje zahtjeva u programskom inženjerstvu“, cilj je prikazati te objasniti što je to inženjerstvo zahtjeva. Naime, bitno je razumjeti da je inženjerstvo zahtjeva cikličan proces koji uključuje pronalaženjei analizu zahtjeva, dokumentaciju zahtjeva te validaciju. Također, u ovom završnom radu se govori o raznim podjelama zahtjeva. Opisani su postupci utvrđivanja i analize zahtjeva poput pogleda, intervjua, scenarija te prototipiranja. Uza sve navedeno, naglašene su različite perspektive i aspekti modeliranja zahtjeva.

Ključne riječi: inženjerstvo zahtjeva, procesi inženjerstva zahtjeva, utvrđivanje i analiza zahtjeva, formalna specifikacija

Summary

In this final paper, whose topic is „Defining requirements in software engineering“, the aim is to show and explain what is engineering requirement. Specifically, it is important to understand that engineering demands a cyclic process which includes finding, and analysis of requirements, documentation of requirements, and validation. Also, this final paper discusses the various division of requests. It includes description of the procedures of identifying and analysing requirements, such as views, interviews, scripts, and prototyping. With all the above, this final paper mentions different perspectives and aspects of modelling requirements.

Keywords: engineering requirements, processes of engineering processes, identification and analysis of requirements, formal specification

Sadržaj

Uvod

Danas je softver sveprisutan u svim ljudskim djelatnostima, a često je presudan za ljudske živote i materijalna dobra. Softverska tehnologija ubrzano napreduje što rezultira velikim materijalnim troškovima i pogreškama. U stvaranju softvera, treba krenuti od postojećih teoretskih osnova te koristiti dokazane tehnike razvoja kao u drugim, zrelijim disciplinama. Dobre metode i tehnike iz drugih disciplina treba primijeniti na razvoj novog softvera ukoliko je to moguće. No, istraživanje i razvoj softvera je mlada disciplina, te se bitno razlikuje od ostalih.

U slučaju da sustav ima grešku, onda je ona unesena u fazi istraživanja ili razvoja. Kod softvera nema trošenja ili zamora materijala, te nema rezervnih dijelova, ali softver može zastarjeti, te se uočeni zaostaci moraju ispraviti ponovnim razvojem. Stoga, ograničen broj poznatih tehnika iz drugih disciplina moći će se jednostavno primijeniti na istraživanje i razvoj softvera, odnosno na softversko inženjerstvo.

U ovom završnom radu pisat će se o jednom dijelu stvaranja softvera, a to je utvrđivanju zahtjeva u programskom inženjerstvu. Naime, utvrđivanje zahtjeva, tj. specifikacija je prva osnovna aktivnost u softverskom procesu. Važno je napomenuti da je to korak u kojem se prikazuje odraz stvarnih potreba korisnika, te se definiraju potrebne karakteristike softvera. Specifikacija zahtjeva ima svrhu u postizanju međusobnog dogovora oko zahtjeva na temelju čega će se programski sustav dalje razvijati. Također, specifikacija se koristi i za sklapanje ugovora između naručitelja sustava i organizacije razvoja sustava, te kao ulaz u sljedeću fazu dizajna tehničke solucije. U ovom radu, pisat će se o zahtjevima koji predstavljaju što sustav treba raditi te definiraju ograničenja u implementaciji i radu samog sustava. Također, predstavit će se nekoliko podjela zahtjeva i problemi u samoj specifikaciji zahtjeva jer ne postoji jedinstveni standard za izražavanje zahtjeva.

U prvom poglavlju objasnit će se što je to inženjerstvo zahtjeva te će se predstaviti višedimenzijska klasifikacija zahtjeva.

U drugom poglavlju govori se o specifikaciji zahtjeva te je prikazana podjela zahtjeva prema razini detalja i prema sadržaju. Također, prikazani su i načini specifikacije zahtjeva, problemi specificiranja zahtjeva te promjene zahtjeva.

U trećem poglavlju opisan je VORD model, a u četvrtom poglavlju dokument zahtjeva.

U petom poglavlju obrađene su podaktivnosti unutar specifikacije, tj. procesi inženjerstva zahtjeva.

U šestom poglavlju govori se o modeliranju zahtjeva.

U sedmom poglavlju opisan je proces evidentiranja zahtjeva, a u osmom poglavlju govori se o studiji slučaja.

U devetom poglavlju obrađeno je utvrđivanje, tj. izlučivanje i analiza zahtjeva gdje su opisani pogledi, intervjui, scenariji i etnografija.

U desetom poglavlju prikazana je uporaba prototipa, a u jedanaestom poglavlju prikazana je validacija zahtjeva.

U dvanaestom poglavlju obrađeni su testiranje i održavanje, a u trinaestom poglavlju formalna specifikacija.

1. Inženjerstvo zahtjeva

Inženjerstvo zahtjeva je proces izrade specifikacije programske potpore. Može se opisati i kao proces utvrđivanja aktivnosti, tj. zahtjeva koje bi sustav trebao obavljati, kao i pod kojim uvjetima bi se te aktivnosti trebale obaviti. Inženjerstvo zahtjeva može se okarakterizirati kao primjena inženjerskih metoda u utvrđivanju zahtjeva sustava. Također, to je prva generička aktivnost tijekom svakog procesa u programskom inženjerstvu. Zahtjevi opisuju što programski sustav treba raditi kao i ograničenja u njegovom radu. Što se tiče procesa izrade specifikacije programske potpore, unutar toga postoje postupci pronalaženja, analiziranja, dokumentiranja i provjere funkcija i ograničenja u uporabi. Zahtjevi su sami po sebi kraći ili duži dokumenti koji specificiraju usluge sustava i ograničenja u upotrebi.

Proces inženjerstva zahtjeva temelji se na prikupljenim informacijama, informacijama koje se odnose na procjenu mogućnosti u odnosu na zahtjeve te se odlučuje da li je predloženi sustav isplativ ili nije. Kod utvrđivanja, tj. otkrivanja i analize zahtjeva, informatičko osoblje upoznaje aplikacijsku domenu uz pomoć korisnika, aktivnosti koje se od sustava očekuje i ograničenja u radu sustava. U procesu utvrđivanja zahtjeva mogu se uključiti krajnji korisnici, inženjeri koji rade na održavanju sustava, eksperti za pojedinačna područja,rukovoditelji, ponuđači opreme (eng. stakeholders) i sl.

Softverski zahtjevi se definiraju na razini sustava te se njima utvrđuju aktivnosti koje korisnici očekuju, tj. zahtijevaju od sustava i uvjeti pod kojima će se sustav razvijati kako bi udovoljio postavljenim zahtjevima. Sami zahtjevi predstavljaju opis aktivnosti sustava s pripadajućim ograničenjima koji se generiraju tijekom procesa definiranja zahtjeva. Što se tiče definiranja zahtjeva, oni se mogu definirati na najvišoj razini apstrakcije sustava ili detaljnije, matematički izraženim opisom funkcija sustava.

Korisnički zahtjevi se mogu razvijati do različitih razina detaljnosti opisa, te su pisani od strane informatičara na temelju iskazanih potreba, tj. zahtjeva krajnjih korisnika odnosno klijenata. Navedeni zahtjevi su prikazani u prirodnom jeziku te dijagramima koji opisuju koje se aktivnosti očekuju od sustava i uz koje uvijete funkcioniranja sustava.

Uza sve navedeno, treba spomenuti da zahtjevi sustava pružaju detaljni opis što bi sustav trebao činiti, tj. što se od sustava očekuje. Prikazani su u obliku strukturiranog dokumenta s detaljnim opisom svih aktivnosti sustava te su pisani za iskusnije tehničko obilježje i menadžment. Također, mogu poslužiti kao ugovor između klijenata i projektanta.

Specifikacija oblikovanja softvera zahtjeva još detaljniji opis te inženjerstvo zahtjeva i aktivnosti oblikovanja. Naime, specifikacija daje sažeti opis oblikovanja softvera koji može poslužiti kao osnova pri detaljnijem oblikovanju i implementaciji. Dokument je orijentiran na implementaciju softvera i pisan je za softverskog inženjera koji će razvijati sustav, tj. aplikaciju.

1.1. Višedimenzijska klasifikacija zahtjeva

U industrijskoj primjeni, pojam zahtjeva ne koristi se konzistentno. Zahtjevi su u nekim slučajevima apstrakte izjave o usluzi koju bi programski sustav trebao ponuditi ili o njegovim ograničenjima. Nasuprot tome, u drugim slučajevima se tu podrazumijeva detaljna, često formalna definicija funkcije programskog proizvoda. Sljedeći primjer ilustrira zašto dolazi do takvih razlika u shvaćanju zahtjeva, te se iz njega može zaključiti da se obje vrste specifikacije, i ona apstraktna i ona detaljna, smatraju specifikacijom zahtjeva.

U slučaju da neka tvrtka nudi natječaj o izradi složenog programskog projekta na otvorenom tržištu, tada ona nudi apstraktnu specifikaciju koja opisuje funkcije konačnog programskog proizvoda u najopćenitijim crtama. Na natječaju se javljaju ponuđači koji predlažu načine na koje se može riješiti zadani problem. Nakon toga, tvrtka pregledava prijedloge ponuđača i odabire izvođača projekta. Odabrani izvođač piše detaljnu specifikaciju sustava koju tvrtka vrednuje prije nego što se potpiše ugovor i kreće u izradu projekta. (Jović i sur., 2015., str. 20).

2. Specifikacija zahtjeva

Specifikacija je početna faza softverskog procesa gdje se analiziraju zahtjevi na budući sustav. Rezultat je dokument o zahtjevima koji opisuje što sustav treba raditi, po mogućnosti bez pretpostavke kako da se to postigne. Samom specifikacijom se bavi tim sastavljen od razvijača softvera i budućih korisnika. (Manger i Mauher, 2011., str. 16).

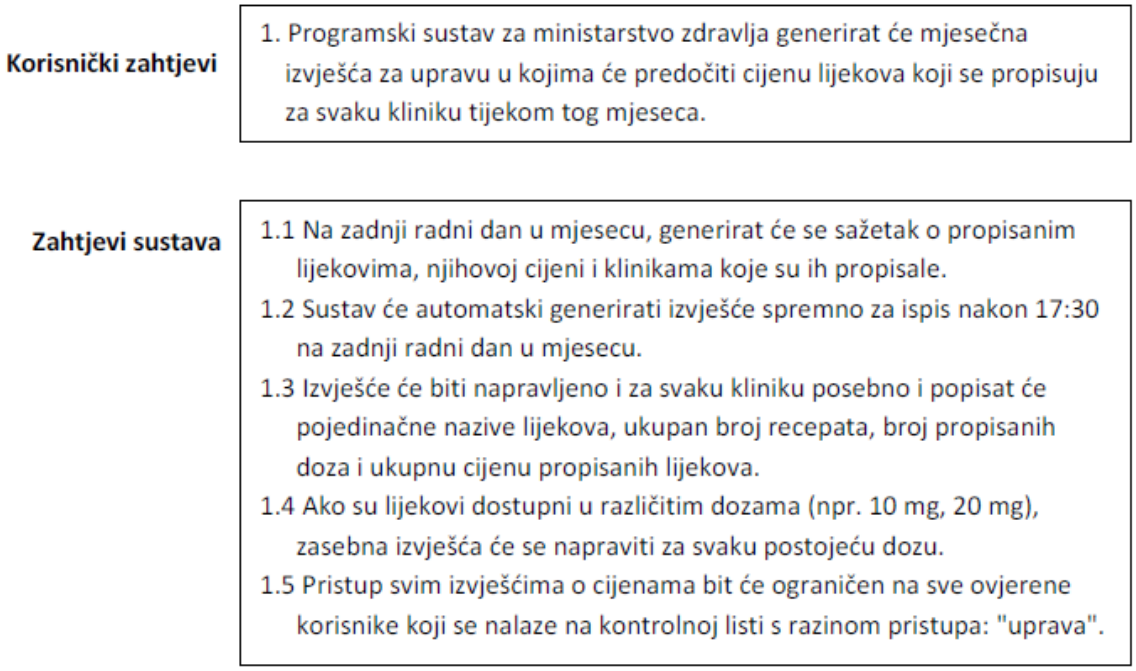
2.1. Podjela zahtjeva prema razini detalja

Mnogi projekti zapadnu u probleme kad se razine zahtjeva međusobno miješaju. Stoga, vrlo je bitno klasificirati zahtjeve prema razini detalja i prema sadržaju. Prema razini detalja, razlikujemo sljedeće zahtjeve (Jović i sur., 2015., str. 20):

- Korisničke zahtjeve (eng. user requirements)
- Zahtjeve sustava (eng. system requirements)
- Specifikaciju programske potpore (eng. software specification)

Korisnički zahtjevi su zahtjevi najviše razine apstrakcije. Njih najčešće zadaje korisnik i to uglavnom u neformalnom, tj. nestrukturiranom obliku, a pišu se u prirodnom jeziku i crtaju jednostavnim grafičkim dijagramima. No, često iz njih nije jasno kako će točno sustav funkcionirati, niti koji će biti njegovi dijelovi. Korisnički zahtjevi obično dolaze u okviru ponude za izradu programskog produkta, kao na sljedećoj slici.

Slika 1.: Korisnički zahtjevi i zahtjevi sustava



Izvor: Jović i suradnici, 2015.,str.21

Zahtjevi sustava predstavljaju detaljnu specifikaciju o funkcionalnosti i ograničenjima programske potpore, a pišu se strukturiranim prirodnim jezikom, posebnim jezicima za oblikovanje sustava, dijagramima i matematičkom notacijom. Na slici 1, možemo vidjeti primjer korisničkih zahtjeva i zahtjeva sustava. Može se primijetiti da su zahtjevi sustava detaljniji u svojoj razradi funkcionalnosti i ograničenja od korisničkih zahtjeva. Za razliku od korisničkih zahtjeva koji se ne zamaraju načinom izvedbe funkcionalnosti, zahtjevi sustava definiraju što programski proizvod treba raditi na način da se iz tako strukturiranih zahtjeva nadzire njegova buduća arhitektura.

Specifikacija programske potpore je najdetaljniji opis i objedinjuje korisničke zahtjeve i zahtjeve sustava. Također, ona može dodatno obuhvaćati tehničku specifikaciju koja opisuje detaljne zahtjeve na arhitekturu i programske dijelove.

Svaku razinu zahtjeva čitaju i dorađuju različiti sudionici na programskom projektu. U tablici 1., predočen je opis koji sudionik čita ili dorađuje zahtjeve, ovisno o razini detalja.

Tablica 1.: Sudionici koji čitaju ili dorađuju zahtjeve u ovisnosti o razini detalja zahtjeva

Razina zahtjeva	**Dionici**
Korisnički zahtjevi	Klijentski rukovoditelji i menadžeri Klijentski inženjeri Krajnji korisnici sustava Rukovoditelji za pisanje ugovora Specijalisti za oblikovanje sustava - arhitekti
Zahtjevi sustava	Klijentski inženjeri Krajnji korisnici sustava Specijalisti za oblikovanje sustava - arhitekti Specijalisti za razvoj programske potpore
Specifikacija programske potpore	Klijentski inženjeri (možda) Specijalisti za oblikovanje sustava - arhitekti Specijalisti za razvoj programske potpore

Izvor: Jović i sur., 2015., str. 22

Treba spomenuti da se evolucija zahtjeva razvija kroz tri koraka, a to su prepoznavanje potreba, tj. želja, otkrivanje ključnih karakteristika identificiranih potreba te preoblikovanje identificiranih karakteristika u zahtjeve i njihovo dokumentiranje. Sam korisnik ima viziju što bi sustav trebao raditi. Primjerice, korisnik želi poslove koji su se obavljali ručno automatizirati; želi poboljšati i/ili proširiti postojeći sustav; želi softver koji će obavljati obrade koji se do sada nisu obavljale ili će ih obavljati bolje. Cilj definiranja zahtjeva je razumijevanje problema, potreba i želja korisnika. Zahtjev je okrenut prvenstveno prema korisnicima te problemima, a ne prema rješenjima i implementaciji. Iz navedenog se može zaključiti da zahtjev označava *kakvo* ponašanje korisnik želi, bez izražavanja *kako* će se ponašanje ostvariti.

2.2. Podjela zahtjeva prema sadržaju

Prema sadržaju, zahtjeve se može podijeliti na (Jović i sur., 2015., str. 21-22):

- Funkcionalne zahtjeve (eng. functional requirements)
- Nefunkcionalne ili ostale zahtjeve (eng. non-functional, other requirements)
- Zahtjeve domene primjene (eng. domain requirements)

Funkcionalni zahtjevi (što?) opisuju funkcije sustava, „tj. usluge koje bi trebalo obavljati, izlaze koje daje za zadane ulaze te njegovo ponašanje u pojedinim situacijama.“ (Manger i Mauher, 2011., str. 16). Naime, to su izjave o uslugama koje programski proizvod mora pružati, tj. izvršiti, kako će sustav reagirati na određeni ulazni podražaj te kako bi se trebao ponašati u određenim situacijama. U nekim slučajevima, funkcionalni zahtjevi trebaju eksplicitno definirati i što sustav ne treba raditi. Može se reći da su funkcionalni zahtjevi kompletni ako sadrže opise svih zahtjevnih mogućnosti, dok su konzistentni ako ne sadržavaju konflikte ili proturječne tvrdnje. Potrebno je naglasiti da je u praksi nemoguće postići kompletan i konzistentan dokument o funkcionalnim zahtjevima.

Nefunkcionalnih zahtjeva (kako?) izražavaju ograničenja na funkcije sustava (primjerice, poštivanje određenih standarada, dopušteno zauzeće memorije te traženo vrijeme), tj. u uslugama i funkcijama programskog proizvoda. Oni opisuju karakteristike (ograničenja) koje softver mora imati te opisuju karakteristike koje sustav postavlja u odnosu na aktivnosti i funkcije koje sustav obavlja (primjerice, vremenska ograničenja, ograničenja u razvojnom procesu i sl.). Potrebno je napomenuti da je dobro da se većina nefunkcionalnih zahtjeva navede kvantitativno kako bi se kasnije mogli ispravno ispitati.

Najgrublja podjela je podjela na tri tipa, a to su (Jović i sur., 2015., str. 22):

1) Zahtjevi programskog proizvoda
2) Organizacijski zahtjevi
3) Vanjski zahtjevi

Zahtjevi programskog proizvoda specificiraju na koji se osobiti način treba ponašati isporučeni proizvod (primjerice, da ima odgovarajuće vrijeme odaziva, da radi na operacijskom sustavu Windows 10, koristi HTTPS protokol i sl.). Organizacijski zahtjev je rezultat organizacijskih pravila i procedura, primjerice uporaba propisanog normiranog procesa razvoja, korištenje uvijek točno određenog programskog jezika pri razvoju i sl. Vanjski zahtjevi proizlaze izvan sustava i razvojnog procesa, npr. postizanje međusobne operabilnosti s drugim sustavima, zakonske zahtjeve i sl.

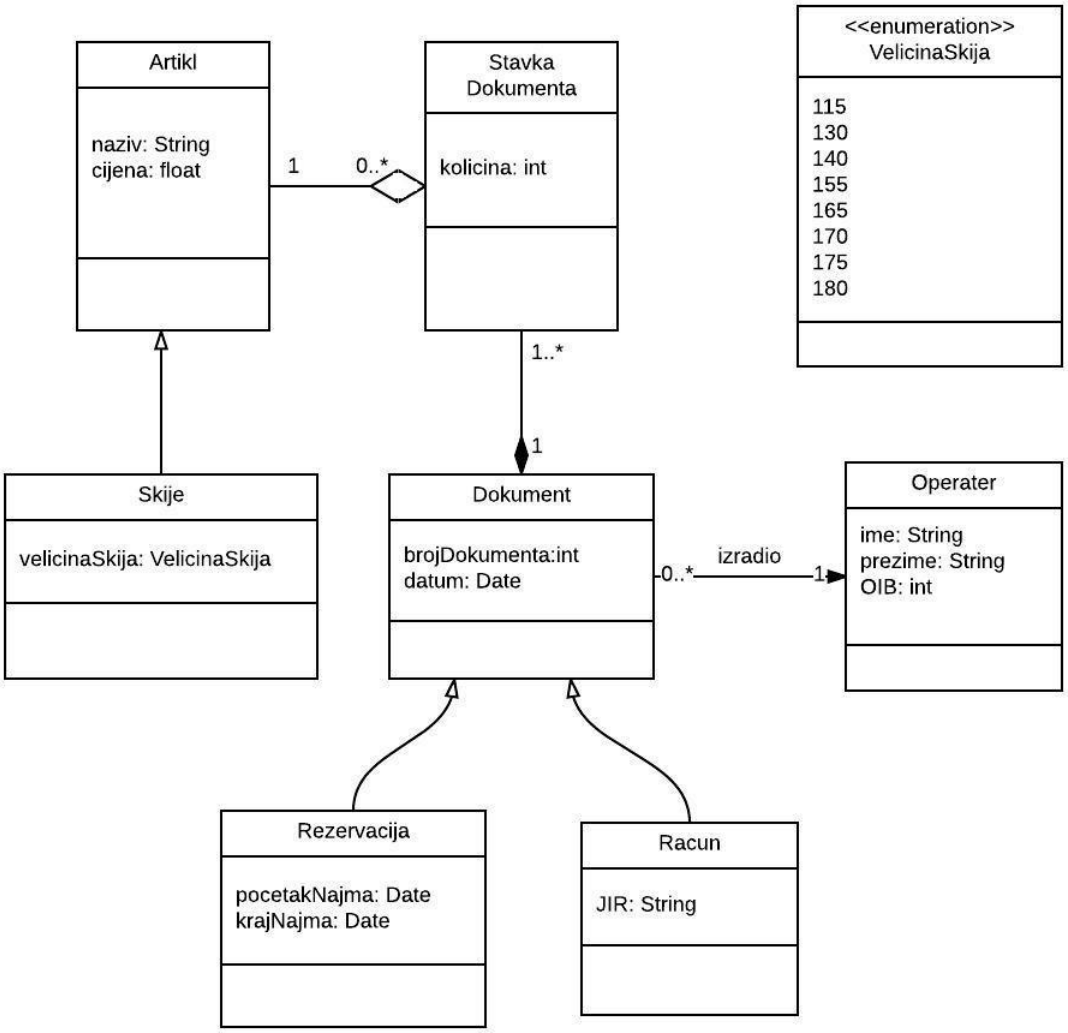
Zahtjevi domene primjene su zahtjevi koji su specifični za domenu primjene sustava. Navedeni zahtjevi mogu biti funkcionalni zahtjevi ili ograničenja na postojeće zahtjeve. Takvi zahtjevi se obično saznaju kasnije u procesu dobivanja, tj. otkrivanja jer ih je često teško izvući od stručnjaka. Naime, prilikom otkrivanja zahtjeva postoje dvije proturječnosti, a to su razumljivost i implicitnost. Razumljivost se odnosi na to da razvojni tim ne razumije domenu primjene te stoga traži detaljan opis zahtjeva. S druge strane, implicitnost se odnosi na to da specijalisti domene poznaju primjenu tako dobro da podrazumijevaju neke zahtjeve i ne navedu ih. Dok razvojni tim uspije izvući traženu funkcionalnost, sustav već može biti dovršen te to poskupljuje izmjene.

2.2.1. Oblikovanje funkcionalnih zahtjeva

Funkcionalne zahtjeve možemo oblikovati modeliranjem poslovnih procesa te modeliranjem klasa. Modeliranje poslovnih procesa ostvaruje se primjenom dijagrama aktivnosti. Za svaki poslovni proces identificiraju se funkcije i uloge te odgovarajuće aktivnosti i njihove međusobne interakcije. Aktivnosti se mogu podijeliti u više koraka i postupaka koji zajedno čine cjelovitost aktivnosti.

Oblikovanju zahtjeva putem modeliranja klasa pristupa se u slučaju kada se u oblikovanje uključuju i podaci. Dijagrami klasa ostvaruju dodatnu vrijednost za SOA procese jer omogućavaju optimiranje procesa definiranja formata poruka i njihovih transformacija. Slika 2 prikazuje dijagram razreda, tj. klasa.

Slika 2.: Prikaz dijagrama klasa



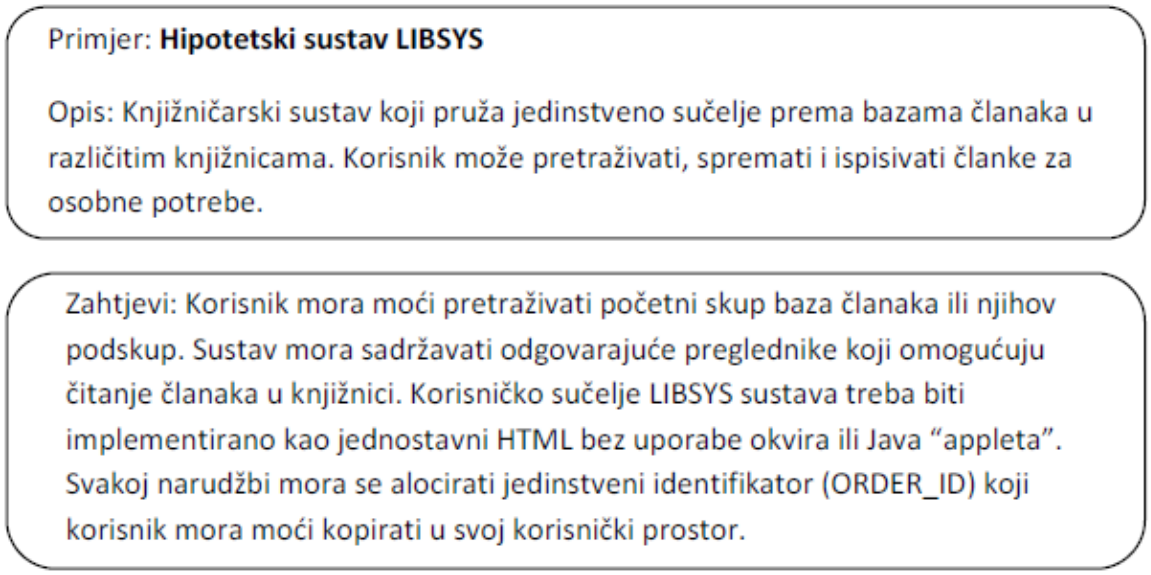
Izvor: Samostalni rad studenta

2.3. Razlikovanje tipa zahtjeva

U praksi, razlikovanje tipova zahtjeva po sadržaju često nije jednostavno. Korisnički zahtjev koji se tiče sigurnosti, kao što je iskaz da se pristup treba ograničiti samo na ovjerene korisnike, može se činiti nefunkcionalnim zahtjevom. No, kad se detaljnije razradi, navedeni zahtjev može generirati druge zahtjeve koji su jasno funkcionalni, kao što je potreba za ugradnjom programske potpore koja se bavi ovjerom korisnika.

Na slici 3, prikazan je primjer programskom sustava LIBSYS te je pomoću njega moguće pojasniti probleme koji se javljaju prilikom navođenja zahtjeva.

Slika 3.: Zahtjevi hipotetskog programski sustav LIBSYS



Izvor: Jović i sur., 2015., str. 23

U primjeru nije odmah jasno što se točno treba napraviti te je očit nedostatak jasnoće. Preciznost u zahtjevima je teško postići bez detaljiziranog, a samim time i naporno čitljivog dokumenta. Može se primijetiti i da se miješaju funkcionalni i nefunkcionalni zahtjevi. Tipični nefunkcionalni zahtjev je specifikacija točne vrste tehnologije koja se treba koristiti (HTML). Također, postoji nenamjerno objedinjavanje više zahtjeva u jednom što je vidljivo u zadnjoj rečenici zahtjeva (alokacija identifikatora i njegovo kopiranje). Na samom kraju, nejasno postavljeni zahtjevi mogu biti različito interpretirani od korisnika i razvojnih timova, što dovodi do problema u procesu razvoja i konačno, u kršenju ugovora. Tipičan primjer navedenog je izjava „odgovarajući preglednik“. Namjena korisnika je da postoji više preglednika posebne namjene za svaki tip dokumenta, ali razvojni tim često ide za time da smanji količinu potencijalno nepotrebnog posla.

2.4. Načini specifikacije zahtjeva

Zahtjevi sustava mogu se napisati na više načina pri čemu se uvijek nastoji izbjeći nestrukturirani prirodni jezik u obliku pisanog teksta. U tablici 2., prikazane su sve mogućnosti za pisanje specifikacije zahtjeva sustava. U praksi se koriste svi pristupi, a često se kombinira više od jednog načina specifikacije.

Tablica 2.: Mogućnosti specifikacije zahtjeva sustava

Notacija	Opis
Rečenice prirodnog jezika	Zahtjevi su pisani korištenjem pobrojanih rečenica prirodnog jezika. Svaka rečenica trebala bi izreći samo jedan zahtjev sustava.
Strukturirani prirodni jezik	Zahtjevi su pisani u prirodnom jeziku u normiranom obliku ili na obrascu. Svako polje daje informaciju o jednom vidu zahtjeva.
Specifični jezik za opis oblikovanja	Ovaj pristup koristi jezik sličan programskom jeziku, ali s apstraktnijim značajkama za definiranje operativnog modela programskog sustava. Primjer jezika je SDL (engl. *Specification and Description Language*). Danas se ovaj pristup rijetko koristi, osim za specifikaciju sučelja.
Grafička notacija	Grafički modeli koji su nadopunjeni tekstualnim opisom koriste se za definiciju funkcionalnih zahtjeva sustava. Pritom se najčešće koristi **UML dijagrami obrazaca uporabe** (engl. *use case diagram*) i **UML sekvencijski dijagrami** (engl. *sequence diagram*).
Matematička specifikacija	Notacija zasnovana na matematičkim konceptima kao što su strojevi s konačnim brojem stanja, skupovima, logici. Ovo je najstrože definirana specifikacija koju klijenti ne vole jer ju najčešće ne razumiju.

Izvor: Jović i sur., 2015., str. 24

2.5. Problemi specificiranja zahtjeva

Kod specifikacije može doći do određenih problema, a oni su sljedeći (Manger i Mauher, 2011., str. 19):

- Ako novi sustav treba promijeniti sadašnji način rada, teško ga je definirati jer s njim još nema iskustva.
- Razni korisnici imaju različite zahtjeve koji mogu biti u sukobu. Konačni zahtjevi su nužno kompromis.

- Financijeri sustava i njegovi korisnici obično nisu isti ljudi. Kupac postavlja zahtjeve motivirane organizacijskim i/ili budžetskim ograničenjima koji su u sukobu s korisničkim zahtjevima.
- Mijenja se poslovno i tehničko okruženje, čime se mijenjaju i zahtjevi.

Problemi specificiranja zahtjeva u prirodnom jeziku su izraženiji u detaljnijoj specifikaciji. U specifikaciji postoji mogućnost dvosmislenosti, prevelike fleksibilnosti, te pomanjkanja modularizacije. Dvosmislenost se odnosi na to da onaj tko piše i onaj tko čita zahtjeve moraju interpretirati istu riječ na isti način. Fleksibilnost se odnosi na istu stvar koja može biti spomenuta u specifikaciji na više različitih značenja, tj. načina. Pomanjkanje modularizacije se odnosi na strukture prirodnog jezika koje su neadekvatne za strukturiranje zahtjeva sustava. Stoga, da bi se otkrila posljedica promjene potrebno je pogledati sve zahtjeve.

SRS (eng. Software Requirement Specification) je dokument koji sadrži zahtjeve koje korisnik postavlja pred sistem te detaljan opis sistemskih (sustavnih) zahtjeva. Korisnici SRS sustava specificiraju zahtjeve, provjeravaju jesu li u dokumentu dobro iskazani njihovi zahtjevi te specificiraju eventualne (nove) promjene. S druge strane, menadžeri koriste dokument kako bi planirali ponudu, te proces razvoja sustava. Sistemski inženjeri pomoću zahtjeva mogu spoznati kako će određeni sustav raditi. Testeri koriste zahtjeve kako bi razvili testove za validaciju sustava. Postoje i inženjeri koji kasnije održavaju sustav i pomoću zahtjeva mogu spoznati sustav i odnose između njihovih dijelova.

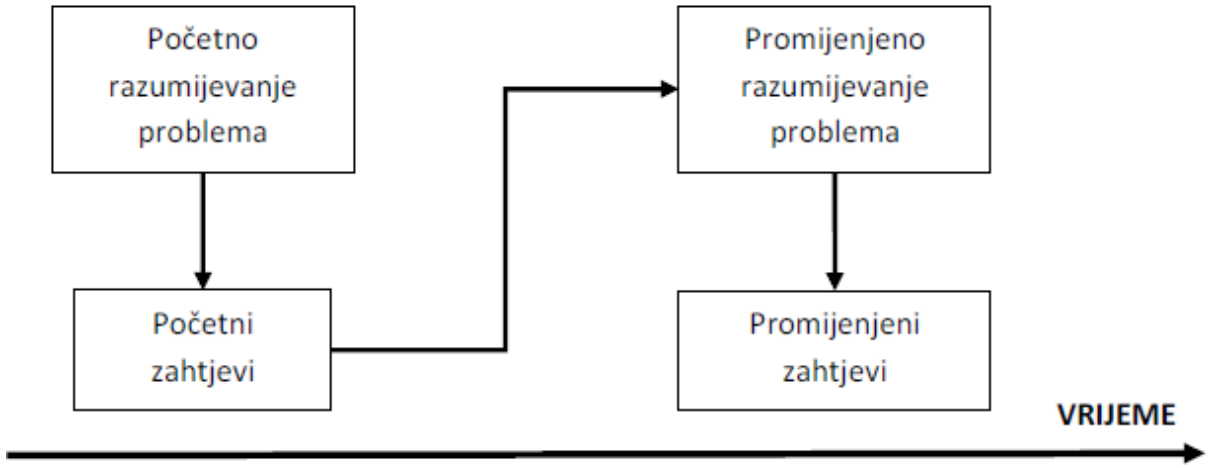
2.6. Rukovanje promjenama u zahtjevu

Prioritet zahtjeva mijenja se tijekom razvojnog procesa. Kupci mogu specificirati zahtjeve s poslovne perspektive te to može biti u konfliktu sa zahtjevima koje postavljaju krajnji korisnici. Poslovna i tehnička okolina sustava mijenja se tijekom razvoja sustava.

Zahtjevi mogu biti dugotrajni (postojani) te se izvode iz temeljne aktivnosti i direktno se odnose na domenu sustava. No, zahtjevi mogu biti promjenjivi (nepostojani) te se često mijenjaju tijekom razvoja sustava i nakon toga, kada je sustav u funkciji.

Rukovanje promjenama u zahtjevima je proces razumijevanja te kontroliranja promjena u zahtjevima sustava (Jović i sur., 2015., str. 33). Promjene često nastupaju zbog promijenjenog modela poslovanja, boljeg razumijevanja procesa tijekom razvoja ili konfliktnih zahtjeva u različitim pogledima koji nisu bili uočeni na vrijeme te je rukovanje promjenama nužno. Evolucija zahtjeva programskog sustava prikazana je na slici 4.

Slika 4.: Evolucija zahtjeva



Izvor: Jović i sur., 2015., str. 34

Naručitelj (korisnik) ima svoju viziju što bi sustav, tj. softver trebao raditi. Primjerice, naručitelj želi poboljšati i/ili proširiti postojeći sustav, želeći softver koji će obavljati obrade koje se do sada nisu obavljale ili će ih obavljati bolje i sl. Uz pomoć korisnika projektant, tj. osoba koja razvija aplikaciju, treba iz identificiranih potreba prepoznati ključne karakteristike. Navedene karakteristike u sljedećem koraku treba iskazati kao zahtjeve koji se postavljaju pred buduću aplikaciju. Zahtjev označava kakvo ponašanje naručitelj želi, bez izražavanja kako će se to ponašanje ostvariti. No, često se pojavljuju novi zahtjevi kada je sustav već instaliran i u svakodnevnoj uporabi. Razlog tome je što je korisnicima i klijentima teško predvidjeti sve učinke koje će imati novorazvijeni sustav na poslove ili proizvodne procese te se s vremenom uočavaju nove potrebe i prioriteti.

Moguće je klasificirati vrste promjena zahtjeva u sljedeće četiri kategorije (Jović i sur., 2015., str. 34):

1) Okolinom promijenjeni zahtjevi
2) Novonastali zahtjevi
3) Posljedični zahtjevi
4) Zahtjevi kompatibilnosti

Kod okolinom promijenjenih zahtjeva, promjena zahtjeva dešava se zbog promjene okoline u kojoj organizacija posluje. Primjerice, bolnica mijenja financijski model pokrivanja usluga. Novonastali zahtjevi se mogu opisati kao zahtjevi koji se pojavljuju nakon što kupac sve bolje razumije sustav koji se oblikuje. Posljedični zahtjevi su zahtjevi koji nastaju nakon uvođenja sustava u korištenje, a rezultat su primjene procesa rada u organizaciji nastalih upravo uvođenjem novoga sustava. Zahtjevi kompatibilnosti podrazumijevaju zahtjeve koji ovise o procesima drugih sustava u organizaciji. U slučaju da se ti sustavi mijenjaju, to traži promjenu sustava i na novo uvedenom sustavu.

Pri rukovanju promjenama potrebno je voditi računa o svim pojedinim zahtjevima i održavati veze između ovisnih zahtjeva tako da se može procijeniti značaj promjena. Zbog tog razloga često se na početku ustanovljava formalni proces za prijedlog uvođenja promjena i za povezivanjem promjena za zahtjevima sustava. Planiranje je prvi korak u provođenju rukovanja zahtjeva. Tijekom tog koraka odlučuje se o načinu identifikacije zahtjeva, procesu rukovanja promjenom, načinu sljedovitosti promjena i o alatima koji će poduprijeti rukovanje promjenama.

Kod velikih sustava potrebno je često dobro proanalizirati isplati se ili ne uvoditi neku promjenu. Posebnu pozornost treba dati onim promjenama koje utječu na jedan ili više ostalih sustava s kojima se trenutni sustav integrira. No, najproblematičnije su one promjene koje se moraju napraviti „pod hitno“. U tom je slučaju važno voditi računa o tome da se promjena provede na svim razinama i da su dokumenti o promjenama u potpunosti sljedivi kako bi se kasnije imalo uvida u to što je točno bilo izmijenjeno. U tom slučaju, puno pomažu odgovarajući specijalizirani CASE alati koji nastoje automatizirati proces evidencije promjena.

Treba spomenuti da proces upravljanja promjenama zahtjeva teče tijekom cijelog procesa inženjerstva zahtjeva i razvoja sustava.

3. VORD model

Metodu VORD (eng. Viewpoint Oriented Requirement Discovery (Description ili Definition – slovi D se u različitim situacijama koristi u više značenja)) razvili su Sommerville i Kotonya 1992. godine. Metoda se razvija kroz četiri koraka, a to su (Požgaj, str. 3):

- Identifikacija točki gledišta
- Strukturiranje točki gledišta
- Dokumentiranje točki gledišta
- Prikazivanje sustava (stvaranje scenarija) temeljenog na točkama gledišta

Kod identifikacije točki gledišta korisnici promatraju sustav sa svog aspekta uloge u sustavu, odnosno s aspekta što očekuju od sustava. Kod strukturiranja točki gledišta točke se grupiraju prema srodnosti i hijerarhiji. Identificirane aktivnosti prepoznate kao zajedničke za više točaka gledišta postavljaju se na hijerarhijski višu razinu. Dokumentiranje točki gledišta obuhvaća identificirane točke gledišta i pripadajuće aktivnosti te se pojašnjavaju detaljnim opisom. Prikazivanje sustava, tj. stvaranje scenarija temeljenog na točkama gledišta transformira se nekom metodom dijagrama u odgovarajući scenariji, danas najčešće primjenom UML-a (dijagram studije slučaja).

Rezultat primjene metode je odgovarajući scenariji koji daje prikaz kako se sustav koristi, tj. kako funkcionira u praksi. On opisuje stanje sustava na početku scenarija i tok događanja u scenariju te ukazuje na moguće neodgovarajuće situacije u sustavu i što u takvim situacijama treba poduzeti. Također, prikazuje stanje sustava nakon završetka odvijanja aktivnosti prikazanih u scenariju.

4. Dokument zahtjeva

Dokument zahtjeva programske potpore je službena izjava o tome što točno razvojni inženjeri trebaju ostvariti. On najčešće sadržava i neformalne korisničke zahtjeve i detaljniju specifikaciju zahtjeva sustava. Kad je to moguće, ponekad se obje vrste zahtjeva integriraju u jedinstveni opis. U ostalim slučajevima korisnički zahtjevi su definirani u uvodu, dok su sustavi definirani u ostatku dokumenata. U slučaju većeg broja zahtjeva sustava detaljni sustavi mogu se prikazati u zasebnom dokumentu. Što se tiče razine detalja koja je uključena u dokument zahtjeva, ona ovisi o vrsti sustava koji se razvija i o vrsti razvojnog procesa. Primjerice, kritični sustavi trebaju imati detaljnu razradu zahtjeva budući da su pitanje sigurnosti i zaštite od velike važnosti.

Prema normi instituta IEEE[1] iz 1998. god. te prema doradi Sommervillea, dokument zahtjeva trebao bi sadržavati sljedeće stavke (IEEE, 1998.):

1) Predgovor
2) Uvod
3) Rječnik pojmova
4) Definicije korisničkih zahtjeva
5) Specifikacija zahtjeva sustava
6) Arhitektura sustava
7) Modeli sustava
8) Evolucija sustava
9) Dodaci
10) Indeks (pojmova, dijagrama, funkcija)

Iz navedenog se može zaključiti da ovako strukturirani dokument zahtjeva već predviđa buduću arhitekturu i model sustav, kao i evoluciju sustava u vidu inačica i mogućih proširenja.

[1] Međunarodna neprofitna profesionalna organizacija za napredovanje tehnologije

5. Procesi inženjerstva zahtjeva

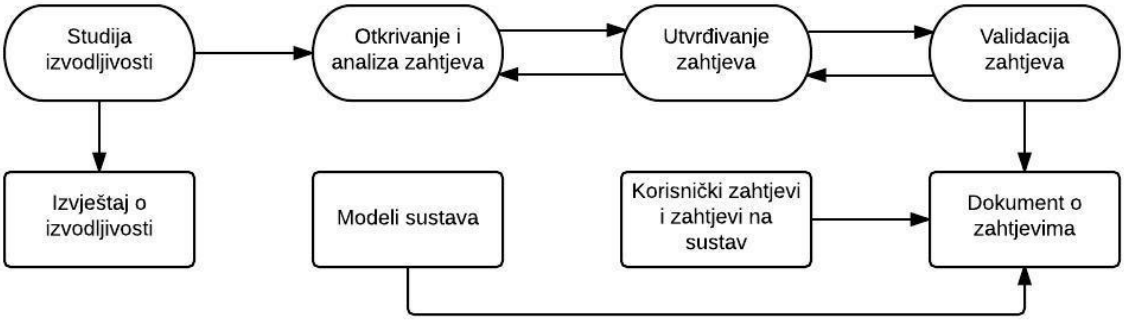
Procesi koji su u upotrebi u inženjerstvu zahtjeva programske potpore razlikuju se ovisno o domeni primjene, ljudskim resursima i organizaciji koja oblikuje zahtjeve. Pritom je važno naglasiti da nema jedinstvenog procesa inženjerstva zahtjeva koji bi bio primjenjiv neovisno o vrsti projekta. U okviru svakog procesa ipak postoje neke generičke aktivnosti inženjerstva zahtjeva koje se redom odvijaju kako bi se u konačnici definirao dokument zahtjeva. Aktivnosti specifikacije mogu se podijeliti na sljedeće podaktivnosti (Manger i Mauher, 2011., str. 18):

1) Studija izvodljivosti (eng. feasibility study)
2) Otkrivanje (izlučivanje) i analiza zahtjeva (eng. requirements elicitation an analysis)
3) Utvrđivanje (specifikacija) zahtjeva (eng. requirements specification)
4) Validacija zahtjeva (eng. requirements validation)

Osim navedenih četiri temeljene generičke aktivnosti, u novije vrijeme kao dodatna aktivnost navodi se i upravljanje promjenama u zahtjevu (eng. requirements management).

U studiji izvodljivosti procjenjuje se mogu li se potrebe korisnika zadovoljiti pomoću već dostupnih hardverskih i softverskih tehnologija. Također, procjenjuje se bi li predloženi sustav bio isplativ u poslovnom smislu i može li se sustav razviti uz raspoloživi budžet. U otkrivanju i analizi zahtjeva otkrivaju se zahtjevi na način da se promatraju postojeći sustavi, analiziraju se radni procesi, intervjuiraju se budući korisnici i sl. Na taj način se stvaraju modeli sustava te prototipovi zbog boljeg razumijevanja sustava kojeg treba stvoriti. Što se tiče utvrđivanja zahtjeva, informacije sakupljene analizom pretvaraju se u tekstove kojima se definiraju zahtjevi. Postoje dvije razine opisivanja zahtjeva, a to su korisnički zahtjevi i zahtjevi na sustav. U validaciji zahtjeva se provjerava jesu li zahtjevi realistični (mogu li se ostvariti raspoloživim budžetom i tehnologijom), konzistentni (da nisu u međusobnom sukobu) te potpuni (uključene su sve potrebne funkcije i ograničenja). Sljedeća slika prikazuje podaktivnosti koje čine aktivnosti specijalizacije.

Slika 5.: Podaktivnosti koje čine aktivnosti specijalizacije



Izvor: Manger i Mauher, 2011., str.18

5.1. Vrste dokumenata koji opisuju zahtjeve

Rezultat specifikacije su dokumenti koji opisuju zahtjeve. Dokumenti se razlikuju po svom sadržaju, načinu opisivanja i detaljnosti. Razlikuje se nekoliko vrsta dokumenata, a to su (Manger i Mauher, 2011., str. 18):

- Korisnički zahtjevi
- Zahtjevi na sustav
- Modeli sustava
- Dokument o zahtjevima
- Specifikacija softverskog dizajna

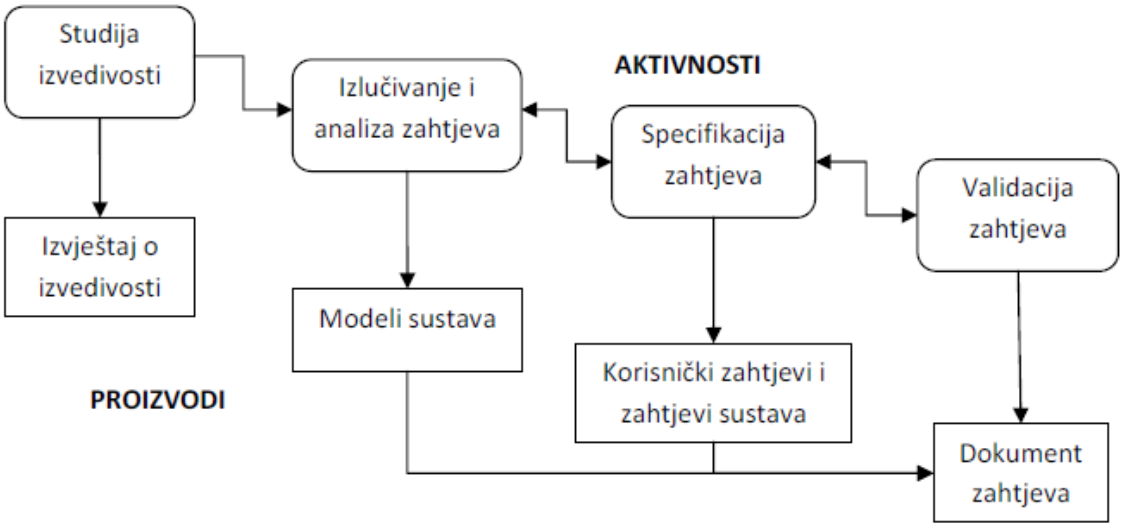
U korisničkim zahtjevima postoji manje precizan tekst u prirodnom jeziku koji je popraćen dijagramima i tablicama, a opisuju se funkcije sustava i ograničenja po kojima sustav radi. Korisnički zahtjev, kako mu i samo ime kaže, prilagođen je krajnjim korisnicima. Što se tiče zahtjeva na sustav, riječ je o detaljnom i preciznom opisu funkcija i ograničenja samog sustava. Zahtjev na sustav je namijenjen softverskom inženjerima kao polazište za oblikovanje, a može služiti i kao temelj ugovora između kupca i razvijača softvera te je pisan donekle strukturiranim prirodnim jezikom. Modeli sustava se odnose na dijagrame koji opisuju ponašanje sustava na precizniji i jezgrovitiji način od prirodnog jezika. Navedeni modeli nastaju tijekom analize zahtjeva kao sredstvo komunikacije između budućih korisnika i razvijača softvera. Dokument o zahtjevima je konačni rezultat specifikacije koji je dobiven kao spoj svih navedenih dokumenata. Specifikacija softverskog dizajna nije

obavezan dokument. Ona predstavlja apstraktni opis građe softvera i funkcije koje obavlja. Specifikacija softverskog dizajna je pisana formalnim jezikom, a pretpostavlja određenu arhitekturu sustava na osnovi koje dalje razrađuje zahtjeve na sustav. Također, stvara se most između aktivnosti i oblikovanja te služi razvijačima softvera kao polazište za daljnje oblikovanje i implementaciju.

5.2. Generičke aktivnosti inženjerstva zahtjeva

Generičke aktivnosti inženjerstva zahtjeva ne odvijaju se nužno slijedno, nego se mnogo češće isprepliću. Na slici 6, prikazan je klasični model procesa inženjerstva zahtjeva u kojem se konačni dokument zahtjeva postepeno slaže na temelju nekoliko manjih dokumenata, tj. proizvoda pojedinih generičkih aktivnosti. Iteracije su prisutne između aktivnosti izlučivanja i analize zahtjeva, specifikacije zahtjeva i validacije zahtjeva, sve dok se zahtjevi ne usuglase.

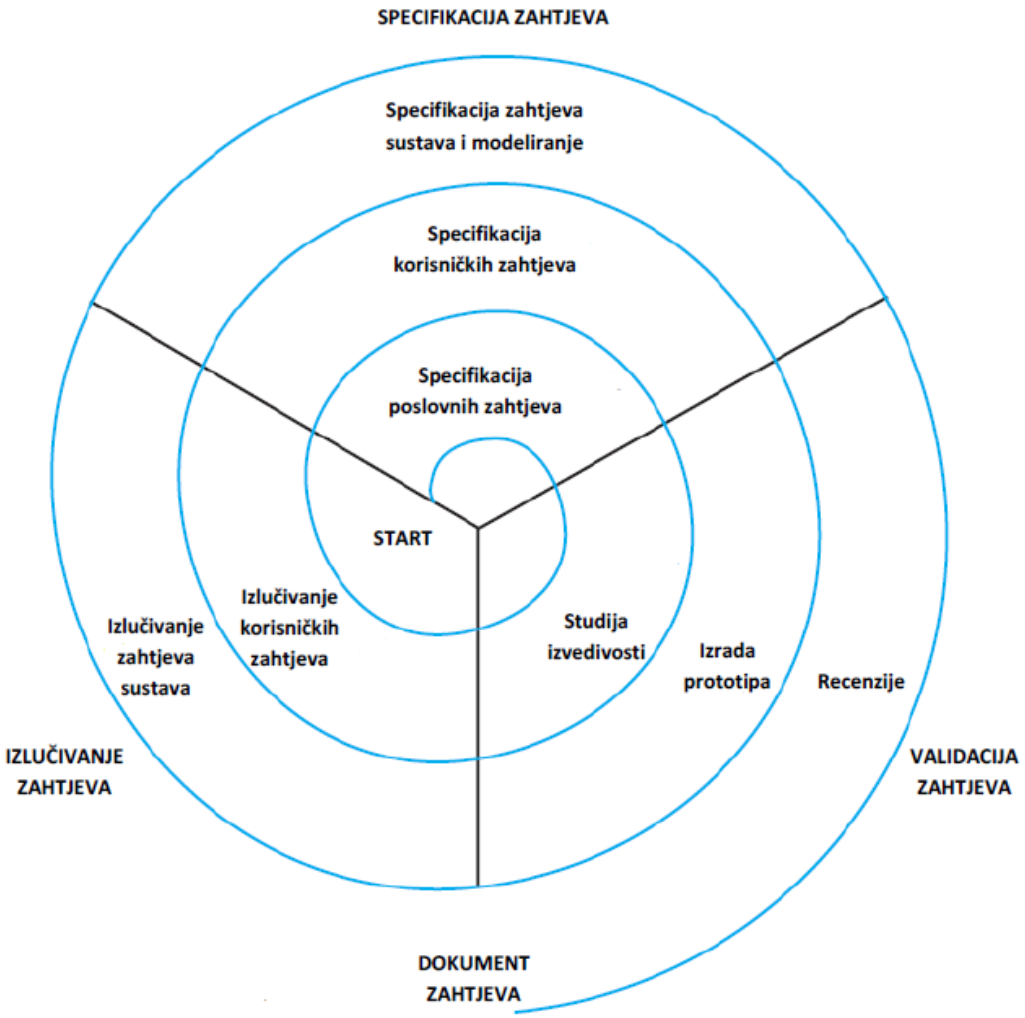
Slika 6.: Klasični model procesa inženjerstva zahtjeva



Izvor: Jović i sur., 2015., str. 26

Bolji model razvoja specifikacije programske potpore od klasičnog je spiralni kod kojeg vrijeme i trud uloženi u svaku aktivnost ovise o stupnju razvoja i vrsti programskog proizvoda, što prikazuje i slika 7. Spiralni model je trostupanjski ciklus koji se sastoji od ponavljanja specifikacije, validacije i izlučivanja zahtjeva.

Slika 7.: Spiralni model procesa inženjerstva zahtjeva



Izvor: Jović i sur., 2015., str. 27

U ranim fazama procesa, najveći intenzitet aktivnosti je na razumijevanju poslovnih i nefunkcionalnih zahtjeva visoke razine apstrakcije. Nakon toga se proučavaju korisnički zahtjevi i izrađuju prototipovi sustava, da bi se u vanjskim slojevima spirale prešlo na detaljno izlučivanje, analizu i specifikaciju svih zahtjeva sustava. Na taj način, spiralni model izravno podržava razradu zahtjeva prema razini detalja. Broj iteracija po spirali pritom može varirati. Kod agilnih metoda razvoja programske potpore, umjesto razvoja prototipa proizvoda, može se ići i na simultani razvoj specifikacije i implementacije.

6. Modeliranje zahtjeva

Kako bi se otkrili zahtjevi, na početku je potrebno razgovarati s korisnicima i proučiti njihove radne procese, dokumente te postojeći softver. Također, potrebno je proučiti relevantne zakone i propise. Kada postoje otkriveni zahtjevi, tada se organiziraju u taksonomiju te ih se dijeli na funkcionalne i nefunkcionalne, jasno se formuliraju i određuje se prioritet. Potrebno je izbjeći i moguće kontradikcije i nepotpunosti.

Neki od primjera funkcionalnih zahtjeva za poslovni sustav su kupci, proizvodi, narudžbe, kanali prodaje, plaćanja i sl. Primjeri nefunkcionalnih zahtjeva su performanse, kapacitet, raspoloživost, poštivanje standarada i sigurnost.

Primjer jasno formuliranih zahtjeva za bankomat je sljedeći (Manger, 2015., str. 5):

Funkcionalni zahtjevi:

1) Bankomat treba ispitati valjanost umetnute bankomatske kartice.
2) Bankomat treba provjeriti točnost PIN-a kojeg je upisao korisnik.
3) Bankomat unutar perioda od 24 h ne smije isplatiti više od 4,000 kuna za istu karticu.

Nefunkcionalni zahtjevi:

1) Softver za bankomat treba biti napisan u jeziku C++.
2) Bankomat treba komunicirati s bankom služeći se kriptiranjem koje je zasnovano na ključu duljine barem 256 bita.
3) Bankomat treba ispitati valjanost bankomatske kartice u roku od 3 sekunde ili brže.

6.1. Modeliranje zahtjeva u skladu s UP

UP[2] zanemaruje nefunkcionalne zahtjeve te se bavi samo funkcionalnim. Također, UP zahtijeva da se funkcionalni zahtjevi razrade i modeliraju kao studija slučaja (eng. use case). Iz navedenog razloga sastavlja se detaljna specifikacija za svaku studiju slučajeva u obliku tablice te se crtaju studije slučaja.

[2]Eng. Unfied Process, što se odnosi na metodologiju koja određuje kako se treba odvijati proces razvoja softvera. UP zahtijeva da se softverski proces sastoji od određenih faza, te propisuje rezultate koje pojedine faze trebaju proizvesti. No, ne određuje način kako se ti rezultati trebaju dokumentirati.

6.2. Priprema za modeliranje

Priprema za modeliranje uključuje utvrđivanje subjekata, pronalaženje aktera, pronalaženje studije slučaja, te se postupak pronalaženja odnosno utvrđivanja iterira. Utvrđivanje subjekta se odnosi na pronalaženje granice sustava. Granica određuje što je dio sustava, a što je izvan njega. Akteri predstavljaju uloge koje stvari, osobe ili pojave izvan sustava poprimaju onda kada su u neposrednoj interakciji sa sustavom. Studija slučaja je funkcija koju sustav izvodi u ime ili korist određenih aktera.

Da bi se postavila studija slučaja, moraju se postaviti sljedeća pitanja (Manger, 2015., str. 12):

- Koje funkcije određeni akter traži od sustava?
- Pohranjuje ili pronalazi li sustav informacije te koji akter pokreće takvo ponašanje sustava?
- Što se dešava ako sustav promijeni stanje (npr. da li krene ili stane te da li akteri to mogu primijetiti?)
- Utječu li vanjski događaji na sustav te tko ili što obavještava sustav o tim događajima?
- Stupa li sustav u interakciju s nekim drugim sustavom?
- Generira li sustav izvještaje?

7. Proces evidentiranja zahtjeva

Proces evidentiranja zahtjeva dijeli se na nekoliko koraka (Požgaj, str. 1):

1) Prepoznavanje (identificiranje)
2) Analiza
3) Specifikacija
4) Validacija

Prepoznavanje (identifikacija) podrazumijeva analitičara koji se bavi postavljanjem pitanja, ispitivanjem ponašanja, demonstriranjem sličnih sustava i sl. Također, prepoznavanje podrazumijeva informatičare koji uz pomoć korisnika upoznaju aktivnosti koje se od sustava očekuju i ograničenja u radu sustava. U ovaj proces se

mogu uključiti svi koji imaju direktan, ali i indirektan utjecaj na zahtjeve korisnika. U analizi, zahtjevi se evidentiraju kroz modeliranje željenog ponašanja što uključuje kreiranje dijagrama te prototipiranje. Specifikacija obuhvaća odlučivanje o tome koji će se dijelovi zahtijevanog ponašanja implementirati u sustav te dokumentiranje ponašanja predloženog softverskog sustava. Validacija odgovara na pitanje odgovara li specifikacija onom što korisnik želi u finalnom proizvodu. U analizi zahtjeva i validaciji postoji mogućnost povratka u fazu prepoznavanja.

Zahtjevi korisnika detaljno opisuju funkcionalne i nefunkcionalne zahtjeve. Problemi koji se mogu pojaviti su pomanjkanje jasnoće, zbrka zahtjeva i stapanje zahtjeva. Što se tiče zahtjeva sustava, oni pružaju detaljniju specifikaciju od korisničkih zahtjeva te mogu poslužiti kao dio ugovora. Sadržavaju *što* sustav treba raditi, a dizajn kasnije pokazuje *kako* će sustav raditi. Problem koji se može javiti je mogućnost dvosmislenosti. Zahtjevi domene prikazuju sredinu u kojoj će budući softverski proizvod funkcionirati te iskazuju specifičnosti same sredine. Oni mogu biti funkcionalni i nefunkcionalni, a mogući problemi su razumljivost i implicitnost.

Poteškoće koje se pojavljuju u iskazivanju zahtjeva su (Požgaj, str. 4):.

- Kupci teško „prevode“ svoje ciljeve u zahtjeve; izražavaju se svojom terminologijom
- Korisnici ne znaju što točno žele
- Postoje različite osobe s različitim zahtjevima te su mogući konflikti
- Kvantitativna verifikacija nefunkcionalnih zahtjeva
- Nefunkcionalni zahtjevi često su u konfliktu s funkcionalnim zahtjevima
- U dokumentu, funkcionalni i nefunkcionalni zahtjevi se evidentiraju odvojeno te ih je teško povezati

Treba naglasiti kako može doći do uplitanja organizacijskih i političkih faktora u sustav zahtjeva te je moguće da se zahtjevi mijenjaju tijekom analize (nova osoba i/ili poslovna okolina se mijenja).

8. Studija izvedivosti

Studija izvedivosti je kratka, fokusirana studija na početku procesa inženjerstva zahtjeva kojom se utvrđuje isplati li se predloženi sustav, tj. je li vrijedan uloženih sredstava. Ulazna informacija je preliminaran skup poslovnih zahtjeva procesa. Zadatak studije izvedivosti je odgovoriti na tri temeljna pitanja (Jović i sur., 2015., str. 26-27):

- Doprinosi li programski sustav ciljevima organizacije u koju se uvodi?
- Može li se programski sustav izvesti postojećom tehnologijom i predviđenim sredstvima?
- Može li se predloženi sustav integrirati s postojećim sustavima organizacije u koju se uvodi?

U slučaju da se utvrdi da je odgovor na barem jedno od navedenih pitanja negativan, nije poželjno nastaviti s detaljnim razvojem programskog sustava. Pri provođenju studije izvedivosti najprije se određuje koje su sve informacije potrebne kako bi se studija završila, zatim se te informacije prikupljaju te se na kraju piše izvješće. Studija izvedivosti često se provodi na terenu, u organizaciji gdje se sustav implementira. Neka od pitanja za ljude u organizaciji koja se često postavljaju u studiji su (Jović i sur., 2015., str. 27-28):

- Koji su trenutni problemi procesa organizacije?
- Kako bi predloženi sustav pomogao u poboljšanju procesa?
- Što ako se sustav ne implementira?
- Koji se problemi mogu očekivati pri integraciji novoga sustava?
- Je li potrebna nova tehnologija ili nove vještine?
- Koje dodatne resurse organizacija traži pri implementaciji novog sustava?

9. Izlučivanje i analiza zahtjeva

Izlučivanje i analiza zahtjeva najznačajnija je generička aktivnosti procesa inženjerstva zahtjeva. Tijekom ove aktivnosti, razvojni inženjeri i drugo tehnički obrazovano osoblje surađuje s klijentima i konačnim korisnicima sustava kako bi saznali što više o domeni primjene i o tome koje usluge trebaju biti podržane s

kakvim performansama i ograničenjima. Vrlo je bitno pri toj aktivnosti uključiti sve ili što više sudionika sustava, tj. osoba koje će razvijeni sustav direktno ili indirektno pogoditi. Primjerice, za sustav bankomata, mogući sudionici programskog sustava uključuju (Jović i sur., 2015., str. 28):

- Bankovne klijente
- Predstavnike drugih banaka
- Bankovne rukovoditelje
- Šalterske službenike
- Administratore baza podataka
- Rukovoditelje sigurnosti
- Marketinški odjel
- Inženjere održavanja sustava (sklopovlja i programske potpore)
- Regulatorna tijela za bankarstvo

Model aktivnosti izlučivanja i analize zahtjeva prikazan je na slici 8.

Slika 8.: Model aktivnosti izlučivanja i analize zahtjeva

Izvor: Jović i sur.,2015., str. 29

Svaka organizacija ima vlastiti način provedbe ovog modela koji ovisi o specijalizaciji osoblja, vrsti programskog proizvoda, korištenim normama razvoja i sl. Proces izlučivanja i analize zahtjeva je iterativan. Znanje o zahtjevima sustava se tijekom svakog ponavljanja ciklusa poboljšava dok se ne komplementira, a prvi korak je

izlučivanje, tj. otkrivanje zahtjeva. To je postupak interakcije sa svim sudionicima s ciljem otkrivenja njihovih zahtjeva. Zahtjevi primjene domene također se nastoji definirati u ovom koraku. Izvori informacija su dokumenti, sudionici te slični razvijeni sustavi. Tehnike koje se koriste za otkrivanje zahtjeva su uglavnom intervjuiranje, izrada scenarija i etnografija, o čemu će kasnije biti riječ. Drugi korak je klasifikacija i organizacija zahtjeva gdje se srodni zahtjevi grupiraju i organiziraju u klastere. U praksi se zahtjevi najčešće grupiraju prema podsustavu cijelog sustava koji opisuju. Treba spomenuti da je u praksi teško razgraničiti postupak specifikacije zahtjeva od razvoja arhitekture sustava. Tijekom trećeg koraka, ustanovljavanja prioriteta i pregovaranja, zahtjevi se razvrstavanju po prioritetima i razrješavaju se konflikti nastali zbog različitih pogleda različitih sudionika na sustav. U praksi, cilj je postići kompromis zahtjeva između sudionika. Na kraju, u četvrtom koraku, zahtjevi se specificiraju, pisano dokumentiraju te se predstavlja ulaz u sljedeću iteraciju. Zahtjevi se zapisuju u formalnim ili neformalnim dokumentima, a najčešće grafičkom notacijom u vidu UML dijagrama studije slučaja i UML sekvencijskih dijagrama. Treba spomenuti kako pri izlučivanju zahtjeva različiti sudionici ili grupe sudionika imaju različiti fokus i perspektivu na zahtjeve sustava.

Da bi se utvrdili zahtjevi potrebno je proučavati tokove informacija. Dakle, proučavaju se dokumenti koji su u optjecaju, prate se radni procesi, razgovara se s korisnicima te se proučavaju postojeći softveri i uočavaju se podaci koje treba pohranjivati i veze među njima. U većim organizacijama postoje razne grupe korisnika te se iz tog razloga mogu pojaviti razna tumačenja značenja i svrhe pojedinih podataka te razni načini njihove upotrebe. Analiza zahtjeva treba uskladiti razlike na način da eliminira redundanciju i nekonzistentnost. Primjerice, u raznim nazivima podataka treba prepoznati sinonime i homonime te ih uskladiti u jedinstvenu terminologiju. Važno je procijeniti opseg i frekvenciju pojedinih transakcija te zahtjeve na performanse. Zbog navedenog razloga, analiza zahtjeva treba obuhvatiti i analizu transakcija (operacija, postupaka) koje će obavljati s podacima, kako to redovno ima utjecaja na sadržaj i konačni oblik softvera.

Rezultat utvrđivanja i analize zahtjeva je dokument koji je najčešće pisan neformalno u prirodnom jeziku, a naziva se specifikacija. Taj dokument rijetko se odnosi samo na

podatke jer on ujedno definira i najvažnije transakcije s podacima, a često i cijele aplikacije.

Cilj oblikovanja je u skladu sa specifikacijom oblikovati građu samog sustava, tj. softvera. Analiza zahtjeva određuje vrste podataka koji se trebaju koristiti i što se s njima mora raditi. S druge strane, oblikovanje predlaže način kako da se podaci na pogodan način grupiraju, strukturiraju te međusobno povežu. Glavni rezultat oblikovanja treba biti shema sustava, građena u skladu s pravilima rabljenog modela podataka te zapisana na način je već rabljeni sustav može razumjeti i realizirati.

9.1. Pogledi

Pogledi (eng. viewpoint) su način strukturiranja zahtjeva na način da oslikavaju perspektivu i fokus različitih grupa sudionika. Svaki pogled uključuje skup zahtjeva sustava. (Jović i sur., 2015., str.29).

Razlikujemo sljedeće poglede (Jović i sur., 2015., str. 29):

- Pogledi interakcije
- Neizravni pogledi
- Pogledi domene primjene

Pod pogledima interakcije podrazumijevaju se ljudi i drugi sudionici koji izravno komuniciraju sa sustavom. Neizravni pogledi obuhvaćaju sudionike koji utječu na zahtjeve, ali ne koriste sustav izravno. Pogledi domene primjene su karakteristike domene i ograničenja na sustav.

Perspektive na sustav nisu posve nezavisne, nego se katkad i poklapaju. Pogledi se često koriste kako bi se strukturirale aktivnosti otkrivanja i specifikacije zahtjeva.

Aktivnost otkrivanja zahtjeva teška je iz više razloga, a oni su sljedeći (Jović i sur., 2015., str. 29-30):

- Sudionici ne znaju što stvarno žele. Naime, oni teško artikuliraju zahtjeve ili su zahtjevi nerealni s obzirom na izvedivost ili cijenu.

- Sudionici izražavaju zahtjeve na različite, njima specifične načine te posjeduju implicitno znanje o svojem radu, tj. domeni.
- Različiti sudionici mogu imati konfliktne zahtjeve i izražene na različite načine.
- Organizacijski i politički faktori mogu utjecati na zahtjeve. Primjerice, rukovoditelj traži da uvedeni sustav ima značajke koje povećavaju njegov utjecaj u organizaciji.
- Zbog dinamike ekonomskog i poslovnog okruženja zahtjevi se mijenjaju za vrijeme procesa analize.
- Uz promjenu poslovnog okruženja, pojavljuju se novi sudionici s novim, specifičnim zahtjevima.

9.2. Intervju

Intervjuiranje, kao metoda izlučivanja zahtjeva, jedan je od najčešće korištenih i često nezaobilaznih pristupa dobivanju korisničkih zahtjeva. Razlikuju se formalni i neformalni intervjui. Kod formalnih intervjua, klijent se prethodno obavještava da će biti intervjuiran vezano uz zahtjeve budućeg programskog sustava. S druge strane, navedena obavijest se ne zahtijeva kod neformalnog intervjua. No, i u formalnom i u neformalnom intervjuiranju tim zadužen za inženjerstvo zahtjeva ispituje sudionike o sustavu koji trenutno koriste te o novom predloženom sustavu.

Prema tipu intervjua, razlikuju se zatvoreni i otvoreni intervjui. Kod zatvorenog tipa intervjua, klijent odgovara na skup već prije definiranih pitanja koja zanimaju razvojni tim. S druge strane, kod otvorenog tipa intervjua ne postoje unaprijed definirana pitanja, nego se niz pitanja otvara i raspravlja sa sudionicima. Može se reći da su u praksi najčešće intervjui kombinacija oba tipa. Najčešće se krene s nekim definiranim pitanjima, a zatim se diskusija razvija u određenom smjeru. Potrebno je naglasiti da potpuno otvoreni tip intervjua rijetko kad rezultira preciznim informacijama jer diskusije krenu u posve nepotpunim pravcima.

Intervjui su korisni za dobivanje globalne slike o tome što pojedinci rade i koja im je uloga te kako će se oni integrirati s novo razvijenim sustavom. Također, intervjui su dobri i za otkrivanje poteškoća s trenutačnim stavovima budući da klijenti najčešće vole pričati o svojem poslu, a često se vole i žaliti na trenutno stanje u tvrtci.

Međutim, intervju nije toliko koristan za razumijevanje zahtjeva u domeni primjene budući da inženjeri zahtjeva ne razumiju specifičnu terminologiju domene, a eksperti domene toliko poznaju te zahtjeve da ih ni ne artikuliraju, tj. misle da su svima razumljivi sami po sebi. Intervjui nisu dobri za dobivanje uvida u organizacijske zahtjeve i ograničenja jer unutar tvrtke često postoje suptilni odnosi moći koji sprečavaju inženjera da dobije ispravnu sliku o organizaciji tvrtke. Osobe koje su zadužene za provođenje intervjua su učinkovite ako su otvorenog uma, bez unaprijed osmišljenih zahtjeva, posebice u vidu prototipa sustava. U pravilu nije dobro pristupiti klijentu na način da se traži točno što klijentu treba. Odnos se uvijek treba postepeno graditi i potrebno je biti susretljiv prema klijentu.

Klijenti često puno lakše shvate programski sustav na temelju stvarnih primjera nego kroz apstraktne opise. Upravo iz toga razloga, scenariji su popularna, ali samim time i uspješna metoda izlučivanja zahtjeva.

9.3. Scenariji

Scenariji su pažljivo osmišljeni primjeri o stvarnom načinu korištenja sustava. Oni opisuju stanje sustava na početku scenarija, tok događaja u scenariju, koje su moguće neodgovarajuće situacije, što se u takvim situacijama poduzima te stanje sustava na završetku scenarija. Može se reći da je scenariji neformalni opis zamišljene interakcije korisnika i sustava. No, potreban je veći broj scenarija da bi se dobila cjelovita slika o zahtjevima na sustav.

Tijekom izlučivanja zahtjeva, sudionici diskutiraju i kritiziraju scenariji. Većini korisnika je teško raspravljati o apstraktnim i sveobuhvatnim specifikacijama te im je lakše diskutirati o konkretnijim scenarijima koji opisuju neke od mogućih načina rada sa sustavom. Također, svaki scenariji pokriva jedan ili nekoliko mogućih interakcija korisnika sa sustavom. Scenariji najčešće započinje s opisom interakcije. Tijekom procesa izlučivanja zahtjeva, dodaju se razni detalji opisu interakcije sve dok scenariji nije završen. Iz navedenih diskusija softverski inženjeri sakupljaju informacije potrebne za formuliranje stvarnih zahtjeva na sustav. Scenariji može biti napisan u obliku teksta uz potporu dijagrama, slika ekrana (eng. screenshot) i sl. te se mogu koristiti više ili manje strukturirani oblici scenarija.

Svaki scenariji trebao bi sadržavati sljedeće elemente (Jovići sur., 2015., str. 31):

- Opis početne situacije
- Opis normalnog (standardnog) tijeka događanja
- Opis što se eventualno može dogoditi krivo
- Informaciju o paralelnim aktivnostima
- Opis stanja gdje scenariji završava

Primjer scenarija za prikupljanje povijesti bolesti u sustavu MHC-PMS prikazan je na slici 9.

Slika 9.: Primjer scenarija za prikupljanje povijesti bolesti

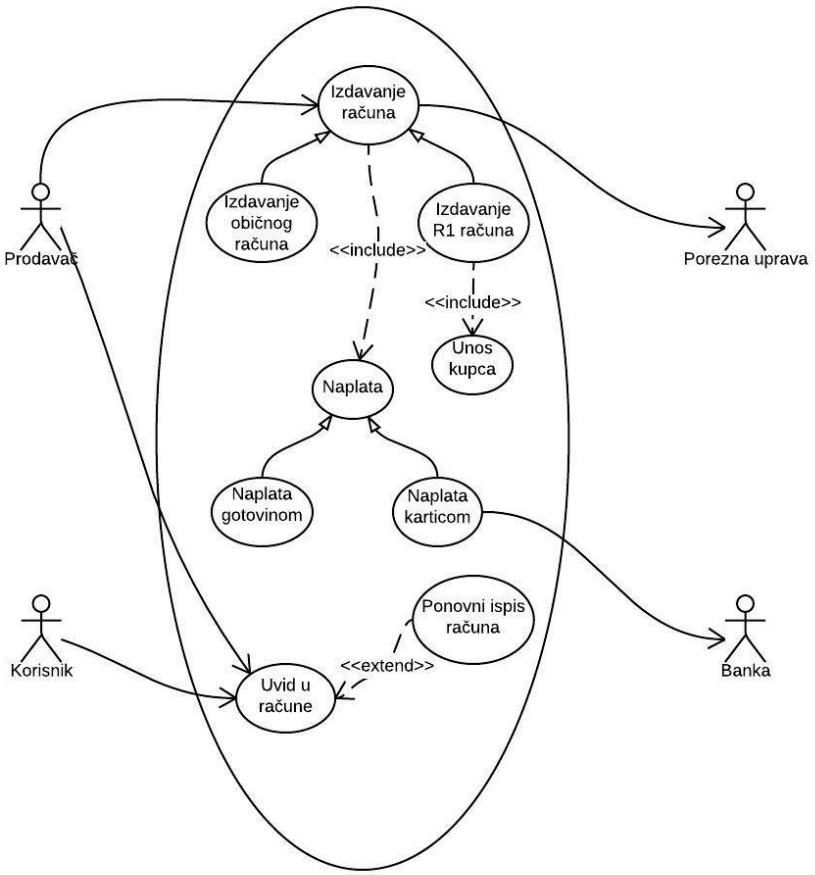
Početna pretpostavka:
Pacijent se našao s medicinskom sestrom na recepciji koja mu je otvorila zapis u sustavu i prikupila osobne informacije. Druga medicinska sestra se prijavila u sustav i prikuplja povijest bolesti.
Normalni tijek događaja:
Sestra pretražuje pacijenta po prezimenu. Ako ima više pacijenata s istim prezimenom, pacijenta se dodatno identificira s imenom i datumom rođenja.
Sestra odabire opciju u izborniku za dodavanje povijesti bolesti.
Sestra slijedi niz upita kako bi unijela informacije o: mentalnom zdravlju (tekst), postojećim fizičkim bolestima (odabir iz izbornika), lijekovima (odabir iz izbornika) i alergijama (tekst).
Što može poći po krivu:
Pacijentov zapis ne postoji ili ga se ne može pronaći. Sestra bi trebala napraviti novi zapis.
Pacijentove bolesti ili lijekovi nisu odabrani iz izbornika. Sestra bi trebala izabrati opciju "ostalo" i unijeti tekstualni opis bolesti/lijeka.
Pacijent ne može ili ne želi dati povijest bolesti. Sestra bi trebala unijeti tekstualni opis o tome da pacijent ne može ili ne želi dati informaciju. Sustav treba ispisati uobičajeni, potpisani obrazac na kojem piše da nedostatak informacija znači da će liječenje biti ograničeno ili odgođeno. Sestra treba obrazac predati pacijentu.
Ostale aktivnosti:
Pacijentov zapis se može pregledavati ali ne i mijenjati od strane ostalih zaposlenika dok ga sestra mijenja.
Završno stanje:
Sestra je prijavljena u sustav. Pacijentov zapis koji sadrži povijest bolesti je unešen u sustav. Dnevnik sustava pokazuje početak i kraj sjednice i ime i prezime sestre koja je provela upis podataka.

Izvor: Jović i sur., 2015., str. 32

Kod scenarija je potrebno spomenuti i slučajeve upotrebe, tj.studiju slučaja (eng. use cases). Studije slučaja pojavljuju se u grafičkom jeziku za modeliranje UML.

Najjednostavniji oblik je oblik gdje jedna studija slučaja bilježi jednu interakciju između sustava i njegove okoline. Može se uočiti iz navedenog da studija slučaja određuje aktere koji su uključeni u određenu interakciju. Skup od više studija slučaja dokumentira se jednim dijagramom studije slučaja u kojem su akteri prikazani kao čovječuljci, a interakcije kao ovali. Kasnije se svaki slučaj prikazuje, tj. opisuje strukturiranim tekstom ili sequence dijagramom. Slika 10. prikazuje primjer dijagrama studije slučaja. Prikazana slika prikazuje sustav za izdavanje računa. Sustav je usklađen sa zakonom o fiskalizaciji te je spojen s poreznom upravom. Stoga, svaki račun se šalje u sustav porezne uprave. Račun može biti običan ili R1. Ukoliko je račun R1, obavezno se unose podaci kupca. Također, sustav omogućava naplatu gotovinom, ali i kreditnom karticom za što se koristi sustav banke partnera. Prodavač, uz izdavanje računa, ima uvid i u svaki izdani račun, što ima i vlasnik, tj. kupac. Također, moguće je ponovno ispisati račun.

Slika 10: Primjer dijagrama studije slučaja



Izvor: Samostalni rad studenta

Treba napomenuti da ne postoji jasna razlika između scenarija i studije slučaja. Riječ je o sličnim produktima koje neovisno upotrebljavaju razni autori. Može se reći da je svaka upotreba dijagrama studije slučaja zapravo jedan scenarij. S druge strane, određeni autori smatraju da jedan slučaj upotrebe obuhvaća više scenarija jer može uključivati više mogućih tokova kontrole. Može se zaključiti da scenariji i studija slučaja predstavljaju efektivnu tehniku otkrivanja zahtjeva koji se tiču neposredne interakcije korisnika sa sustavom. No, tom tehnikom se ne mogu otkriti specifični zahtjevi koji su vezani za aplikacijsku domenu te ne mogu obuhvatiti nefunkcionalne zahtjeve.

9.4. Etnografija

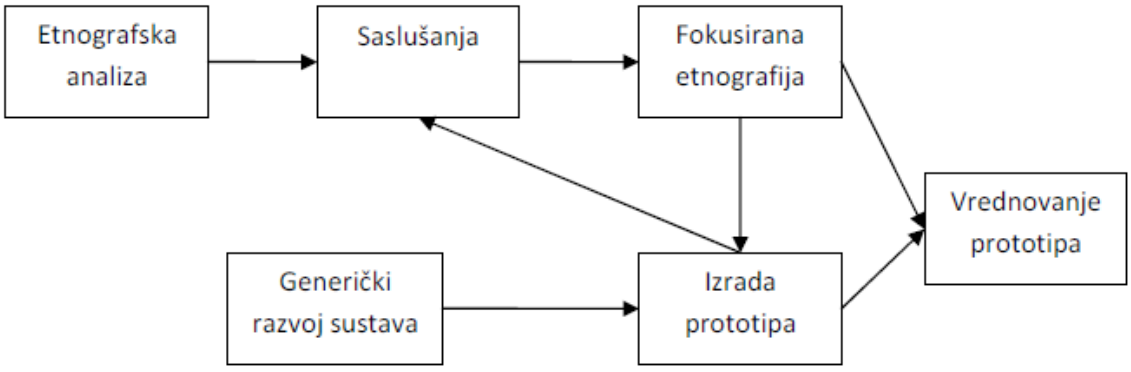
Etnografija je tehnika opažanja koja se koristi kako bi se bolje razumjeli procesi kod klijenata i kako bi se pomoglo otkriti što je moguće više ispravnih i korisnih zahtjeva. Ona podrazumijeva dolazak jednog ili više ljudi iz razvojnog tima u organizaciju gdje će se sustav primjenjivati. Naime, ona podrazumijeva i uključivanje navedenih inženjera (tzv. etnografa) u svakodnevne aktivnosti u tom okruženju. (Jović i sur., 2015., str. 31).

Etnografija je znanstvena disciplina (metoda) gdje nepristrani promatrači promatraju i analiziraju kako ljudi stvarno rade. Etnografske studije pokazuje da je rad obično dinamičniji i kompleksniji nego što to sugeriraju jednostavni modeli sustava. Etnografija je znanost koja proučava i opisuje materijalnu, društvenu i duhovnu kulturu pojedinih naroda gdje se etnograf, da bi proučavao neko pleme, uključuje u život tog plemena. U to vrijeme, etnograf bilježi sve ono što vidi i čuje oko sebe (primjerice odjeću, običaje, pjesme i sl.). Etnografija otkriva dva tipa zahtjeva, a to su zahtjevi koji proizlaze iz načina na koji ljudi stvarno rade, a ne kako proces definira da bi ljudi trebali raditi te zahtjevi koji proizlaze iz suradnje i drugih ljudskih aktivnosti. Iz navedenog se može zaključiti da je etnografija u kontekstu softverskog inženjerstva metoda za otkrivanje zahtjeva gdje softverski inženjer preuzima ulogu etnografa, a korisnici ulogu plemena. Softverski inženjer boravi na radnim mjestima korisnika te promatra i bilježi njihove radne postupke. Prati se svakodnevni rad korisnika i zapisuju se stvarni zadaci koje korisnici obavljaju te se na taj način

otkrivaju implicitni zahtjevi sustava koji pokazuju stvarni način kako ljudi rade, a ne formalne procese koje definira organizacija.

Uza sve navedeno, etnografija se može kombinirati s izradom prototipova u tzv. fokusiranoj etnografiji, što prikazuje slika 11. Ideja je da etnograf informira ostale inženjere tijekom razvoja prototipa, čime se smanjuje broj ciklusa poboljšavanja prototipa. Nadalje, izrada prototipa fokusira etnografiju tako što otkriva probleme i pitanja koja se potom mogu prodiskutirati s etnografom na koje on može potražiti odgovore kod korisnika u sljedećoj fazi razvoja.

Slika 11.: Fokusirana etnografija



Izvor: Jović i sur., 2015., str. 33

Mora se naglasiti kako etnografija nije pogodna za uočavanje novih značajki koje treba razviti, a i teško je pomoću nje uočiti organizacijske i domenske zahtjeve budući da se fokusira na krajnjeg korisnika. Ipak, vrlo je korisna za uočavanje kritičkih detalja u procesu koji su možda izbjegli ostalim tehnikama izlučivanja i analize zahtjeva te uvijek se koristi u kombinaciji s ostalim tehnikama.

10. Upotreba prototipova

Korisnici budućeg sustava često ne uspijevaju jasno izraziti svoje zahtjeve i očekivanja. Naime, oni ne mogu predvidjeti kako će taj sustav utjecati na njihove radne navike, u kakvoj će interakciji biti s drugim sustavima te u kojoj mjeri može automatizirati radne procese i zbog toga se ide u realizaciju prototipa. Riječ je o

jednostavnom i brzo razvijenom programu koji oponaša budući sustav. Prednost prototipa je to što se lako mijenja i nadograđuje te omogućuje isprobavanje različitih ideja i opcija. Prototip podržava dvije podaktivnosti unutar aktivnosti specifikacije, a to su otkrivanje zahtjeva i validacija zahtjeva. Otkrivanje zahtjeva podrazumijeva da korisnici eksperimentiraju te na taj način otkrivaju svoje potrebe, a dobivaju i nove ideje o tome što bi sustav trebao raditi. U validaciji zahtjeva uočavaju se pogreške i propusti u polaznim zahtjevima (npr. nepotpuni ili nekonzistentni zahtjevi). Uza sve navedeno, prototip može donijeti dodatne koristi poput otkrivanja nesporazuma između razvijača softvera i korisnika, lakša identifikacija složene funkcije koje zahtijevaju pozornost, menadžmentu se pokazuje izvedivost i korisnost sustava i sl.

Prototipiranje je vrlo pogodna metoda za specificiranje korisničkog sučelja. No, zahtjeve na sučelje teško je opisati riječima, a pogotovo kad je riječ o grafičkom sučelju te se do pogodnog sučelja dolazi se eksperimentiranjem. Pogodni alati su jezici četvrte generacije, Visual Basic, TCL/Tk, alati za internetske stranice i sl.

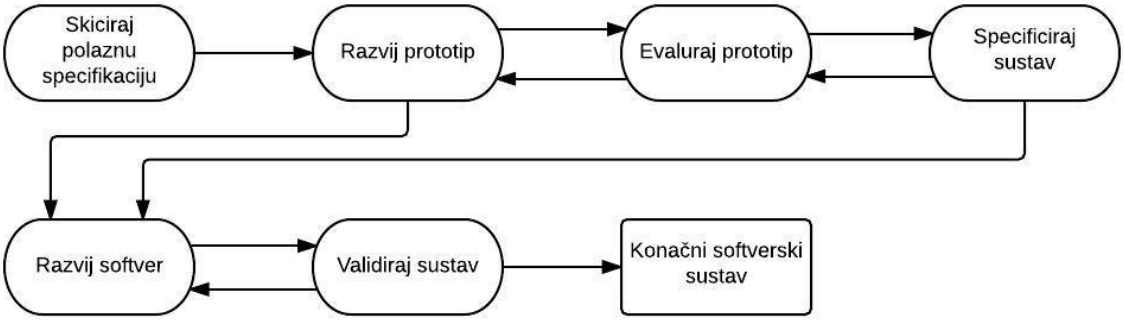
10.1. Mjesto prototipiranja unutar softverskog procesa

U praksi, pojam „prototipiranje“ često se miješa s pojmom „istraživačko programiranje[3]“. No, razlike između tih dvaju pojmova postoje. Cilj prototipiranja je otkrivanje i validacija zahtjeva. Prioritet kod implementiranja zahtjeva imaju oni koji su nejasni i koje treba dodatno istražiti. Nakon utvrđivanja zahtjeva, prototip se odbacuje, a sustav se dalje oblikuje i implementira na konvencionalan način. S druge strane, cilj istraživačkog programiranja je postupno dotjerivanje samog prototipa u skladu sa zahtjevima koji se otkrivaju, sve dok se dotični prototip ne pretvori u konačni sustav. Za razliku od prototipiranja, prioritet kod implementiranja zahtjeva u istraživačkom programiranju imaju oni koji su najjasniji i koji su korisnicima najpotrebniji.

Sljedeća slika prikazuje slučaj prototipiranja, tj. kako izgleda cjelokupni softverski proces. Treba napomenuti da treba biti oprezan u izboru dijelova iz prototipa jer su oni često loše strukturirani i dokumentirani, imaju slabe performanse i pouzdanost te ne zadovoljavaju zahtjeve sigurnosti.

[3]Razvoj softvera po modelu evolucijskog razvoja.

Slika 12.: Softverski proces s uključenim prototipiranjem



Izvor: Manger i Mauher, 2011., str. 25

10.2. Tehnike za prototipiranje

Kako bi prototipiranje imalo smisla, sam prototip se mora stvoriti brzo, jeftino i fleksibilno. Iz navedenog razloga su potrebne odgovarajuće razvojne tehnike poput (Manger i Mauher, 2011., str. 25-26):

- Razvoj u dinamičkim jezicima visoke razine
- Programiranje za baze podataka
- Povezivanje gotovih dijelova ili aplikacija

Kod razvoja u dinamičkim jezicima visoke razine, riječ je o programskim jezicima koji su velikim dijelom interpretirani i zahtijevaju veliku run-time okolinu. Također, programi se pišu jednostavno i brzo te se mijenjaju na dinamičan način, ali njihove performanse obično slabe zbog interpretiranja i run-time podrške.

Kod programiranja za baze podataka, svi komercijalni sustavi su opskrbljeni vlastitim alatima za izradu aplikacija koje obavljaju radnje s podacima poput ažuriranja, pretraživanja i izrade izvještaja. Navedeni alati se nazivaju i „jezici četvrte generacije“ (eng. kratica 4GL). Što se tiče primjera 4GL-a, oni se mogu naći unutar baza Oracle, Informix ili DB2. Izrada aplikacija, odnosno prototipova, pomoću 4GL-a brza je i jednostavna te ne zahtijeva puno programiranja. No, dobivena aplikacija je poprilično spora i rastrošna, te jedna od mana 4GL-a je da su nestandardizirani i neprenosivi s jedne baze na drugu.

Kod povezivanja gotovih dijelova ili aplikacija, prototip se slaže od već gotovih dijelova. U tu svrhu, treba imati na raspolaganju dovoljan broj upotrebljivih i gotovih dijelova te okvir ili mehanizam za povezivanje svih tih dijelova. Također, treba biti spreman na određene kompromise u specifikaciji samog prototipa.

11. Validacija zahtjeva

Validacija zahtjeva je proces provjere definiraju li zaista zahtjevi koji su dobiveni od klijenata sustav koji korisnik želi. To je iznimno važan korak u inženjerstvu zahtjeva jer pogrešno specificirani zahtjevi mogu uzrokovati visoke troškove. Validacija se isprepliće s analizom zahtjeva budući da se bavi pronalaskom pogrešaka u zahtjevima. Naime, naknadno ispravljanje pogreške u zahtjevima može biti mnogo puta skuplje od jednostavnog ispravljanja pogreške u implementaciji. Tehnike validacije uključuju (Jović i sur., 2015., str. 32):

- Recenziju zahtjeva
- Izradu prototipa
- Generiranje ispitnih slučaja

Recenzija zahtjeva je detaljna, ručna analiza zahtjeva od strane zajedničkog tima. Izrada prototipa je zapravo provjera zahtjeva na izvedenom sustavu, a generiranje ispitnih slučaja je razvoj ispitnih sekvenci za provjeru zahtjeva.

U zahtjevima se provjerava više svojstava, a to uključuje (Jović i sur., 2015., str. 33):

- Valjanost (Ostvaruje li sustav funkcionalnost koja podupire većina sudionika?)
- Konzistentnost (Postoji li konflikt u zahtjevima?)
- Kompletnost (Uključuje li sustav sve funkcije koje je korisnik tražio?)
- Realnost (Mogu li se sve funkcije implementirati uz danu tehnologiju i proračun?)
- Provjerljivost (Mogu li se svi zahtjevi provjeriti?)
- Razumljivost (Je li dokument zahtjeva jasno napisan?)
- Sljednost (Je li naveden izvor dokumenta u slučaju više povezanih dokumenata?)
- Prilagodljivost (Mogu li se zahtjevi mijenjati bez utjecaja na druge zahtjeve?)

Treba napomenuti da nije dobro podcijeniti probleme koji se tiču validacije zahtjeva. U konačnici, teško je pokazati da je neki skup zahtjeva točno onaj koji odgovara klijentskim potrebama. Stoga, potrebno je imati dobro razvijenu sposobnost zamišljanja programskog sustava u izvođenju, za što je potrebno imati iskustva.

12. Testiranje i održavanje

Testiranje se svodi na to da korisnici pokusno rade sa sustavom i provjeravaju udovoljava li sustav njihovim potrebama. Pokreću se najvažnije transakcije s podacima, prati se njihov učinak te se mjere performanse i nastoji se utvrditi stabilnost i pouzdanost rada pod opterećenjem. Glavni je cilj testiranja otkrivanje i popravak pogrešaka koje su se potkrale u svakoj od prethodnih aktivnosti (u analizi zahtjeva, oblikovanju te implementaciji). Pogreške u prijašnjim aktivnostima imaju teže posljedice jer se provlače i kroz sljedeće aktivnosti, stoga zahtijevaju više truda da se poprave. Primjerice, pogreška u analizi može uzrokovati da u specifikaciji nedostaje neki važni podatak, a to onda znači da tog podatka neće biti niti kasnije u dokumentaciji te se popravak treba izvršiti u svim dokumentima.

Tijekom testiranja mjerenjem performansi nastoji se utvrditi jesu li zadovoljeni zahtjevi povezani s performansima. Ukoliko performanse nisu dovoljno dobre, administrator sustava može ih pokušati ispraviti podešavanjem određenih parametara fizičke organizacije (npr. dodavanje pomoćnih struktura podataka). No, loše performanse mogu biti posljedica i pogrešaka u oblikovanja, npr. posljedica neuočavanja važnih veza između određenih vrsta podataka. U navedenom slučaju slijedi opet popravak pogrešaka.

Održavanje se odvija u vrijeme kada je sustav, tj. softver već ušao u redovnu upotrebu. Riječ je o kontinuiranom procesu gdje su softver i njegovi prateći dokumenti podvrgnuti stalnim promjenama.

Potrebno je naglasiti da je održavanje sustava nužnost na koju treba biti svjestan već tijekom razvoja sustava te ju uzimati u obzir tijekom cijelog njegovog života. Također, treba naglasiti da sustav koji se ne mijenja, vrlo brzo postaje neupotrebljiv. Da bi se promjene obavljale što lakše i bezbolnije, iznimno je važno da sustav od početka ima zdravu građu koja odražava inherentnu logiku i povezanost samih podataka.

13. Formalna specifikacija

Formalna specifikacija omogućuje da se zahtjevi zapišu precizno i nedvosmisleno. Umjesto prirodnog jezika, koristi se jezik čiji su rječnik, sintaksa i semantika formalno definirani. Taj jezik je zasnovan na matematičkim pojmovima, obično iz teorije skupova, algebre ili logike.

13.1. Uloga, prednosti i mane formalne specifikacije

Formalna specifikacija je nužna kada se koristi model formalnog razvoja softvera. Kod drugih modela softverskog procesa ona nije nužna, ali može koristiti ako se želi razviti preciznija vrstu opisa zahtjeva, a to je specifikacija softverskog dizajna. U tom slučaju, formalna specifikacija je na granici sa specifikacijom i oblikovanjem. Sam rezultat formalne specifikacije namijenjen je ponajprije razvijačima softvera kao polazište za daljnje oblikovanje, implementaciju te verifikaciju.

Prednosti formalne specifikacije su (Manger i Mauher, 2011., str. 27):

1) Bolji uvid u zahtjeve, otklanjanje nesporazuma, smanjenje mogućnosti grešaka (što sve pridonosi pouzdanosti softvera)
2) Mogućnost analiziranja zahtjeva matematičkim metodama (potpunost, konzistentnost)
3) Može služiti kao podloga za formalnu verifikaciju implementiranog sustava
4) Mogućnost „animacije“ specifikacije u svrhu prototipiranja

Mane formalne specifikacije su (Manger i Mauher, 2011., str. 27):

1) Nerazumljiva je korisnicima i menadžmentu
2) Zahtijeva posebno osposobljene softverske inženjere
3) Ne da se dobro „skalirati“, odnosno za veći sustav količina posla je prevelika

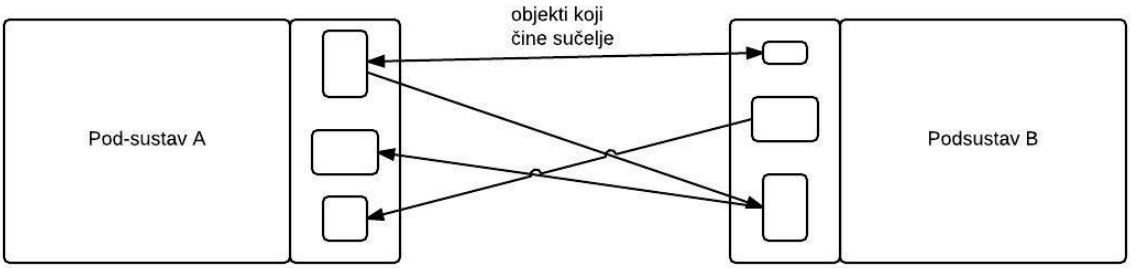
U skladu s navedenim prednostima i manama, formalne metode se koriste jedino za razvoj manjih dijelova pojedinih sustava. Konkretnije, koriste se tamo gdje se zahtjeva iznimno velika pouzdanost i sigurnost.

13.2. Formalna specifikacija sučelja

Veliki sustavi se sastoje od podsustava koji su u međusobnoj interakciji, a važan dio specifikacije sustava je definiranje sučelja pojedinih podsustava. Nakon što su sučelja utvrđena, svaki se podsustav može dalje razvijati neovisno o drugom. (Manger i Mauher, 2011., str. 28).

Sučelje podsustava najčešće se definira kao skup apstraktnih tipova podataka ili objekata kao što je prikazano slikom 13.

Slika 13.: Sučelje podsustava prikazano pomoću objekata



Izvor: Manger i Mauher, 2011., str. 28

Dakle, definiraju se podaci i operacije kojima se može pristupiti izvana te se specifikacija sučelja svodi na specifikaciju svakog od korištenih apstraktnih tipova podataka.

Zaključak

U računarstvu je moguće primijeniti metode i tehnike koje su razvijene u samim informacijsko-komunikacijskim znanostima, ali one nisu dovoljne. Formalizacija podrazumijeva niz neprihvatljivih pojednostavljenja, kao što su gradnja sustava od početka, ograničavanje budžeta, zamrzavanje tehničkih zahtjeva u jednom trenutku i sl. No, takav pristup nije realan te nije prihvatljiv na tržištu, a onda ni organizacijama koje se sa softverskim rješenjima na njemu pojavljuju. Pritisak tržišta često otvara nove probleme u razvoju softvera te se do rješenja ili proizvoda nastoji doći što prije. U prvi plan stavlja se fokus na proizvod za tržište pa se s raspoloživim metodama, tehnikama i pomagalima započinje razvoj. U tom se slučaju zaboravljaju nove metode, tehnike i pomagala te takav pristup rezultira problemima pa je potrebno zastati, promotriti problem istraživanja i razvoja softvera iznova i sveobuhvatno, ne zaobilazeći organizacijske aspekte.

Može se zaključiti da je softver najčešće kritičan element nekog sustava, a razvoj softvera mnogo je više od programiranja. Softverski proizvodi su softverski sustavi koji su različite veličine i kompleksnosti te koji su isporučeni korisniku zajedno s dokumentacijom koja opisuje npr. kako ga instalirati, koristiti, održavati i sl. No, na žalost, zahtjevi za brzinom, kvalitetom i kompleksnošću softverskog razvoja rastu brže nego što softversko inženjerstvo može ponuditi odgovore.

U ovom završnom radu fokus nije na softverskom inženjerstvu u cijelosti. U većini slučajeva, softverskom inženjerstvu se pristupa s tehničkog aspekta te se naglašava da je softverski razvoj potrebno kontrolirati i njime upravljati. Ukratko, ovaj se rad fokusira na aspektima stvaranja i vođenja istraživačke i komunikacijske tehnologije kako bi se sami zahtjevi utvrdili kako treba te kasnije i sam gotovi softver. Kako bi se osigurala kompetitivnost, ne smije se dozvoliti da jedan istraživački projekt započne da bi zatim zbog važnijeg bio zaustavljen. Također, u ovom radu su opisani postupci utvrđivanja i analize zahtjeva poput pogleda, intervjua, scenarija te prototipiranja koji se trebaju kombinirati u samoj specifikaciji kako bi se otkrilo što veći broj pogodnih informacija koje će se kasnije iskoristiti za stvaranje što boljeg softvera. S obzirom na sveprisutnost i stalni rast različitih softvera, potrebno je odmah na početku shvatiti potrebe korisnika kako bi softver kasnije bio što bolji i kvalitetnije te što bolje koristio korisnicima.

Literatura

1) Carić, A. (2003.) *Istraživanje i razvoj u informacijskoj i komunikacijskoj tehnologiji* Element, 1. izdanje, Zagreb
2) IEEE (1998.)*IEEE Recommended Practice for Software Requirements Specifications*. IEEE, United States of America: Institute of Electrical and Electronics Engineers, Inc
3) Jović, A., Horvat, M. i Ivošević, D. (2015.) *Procesi programskog inženjerstva: Oblikovanje programske potpore*, 2.0. verzija, Zagreb: Sveučilište u Zagrebu
4) Khosrow-Pour, M. *Encyclopedia of Information Science and Technology*(2009.) Second Edition, USA: Information Resources Management Association;
5) Manger, R. (2015.) *Softversko inženjerstvo; Vježbe 3: Utvrđivanje i modeliranje zahtjeva*. Zagreb: Sveučilište u Zagrebu; dostupno na: http://web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-03.pdf, pristupljeno: 1.05.2016.
6) Manger, R. i Mauher, M. (2011.) *Programsko inženjerstvo*. Zagreb:Algebra d.o.o.
7) Požgaj, Ž., *Procesi karakteristični za inženjerstvo zahtjeva*. Dostupno na: http://web.efzg.hr/dok//inf/pozgaj/pisani%20materijali/T04%20Softverski%20zahtjevi.pdf, pristupljeno: 1.07.2016.
8) Sommerville, I.(2011.)*Software Engineering,*9th edition, Addison-Wesley, Harlow, England
9) Sommerville, I. (2015) *Software Engineering,* 10th edition. Pearson Education, Boston

Popis slika

Popis tablica