Sadržaj

Uvod 1
Terminologija 4
Proces razvoja softvera 5

1. Inženjerstvo zahteva 7
Analiza i prikupljanje zahteva 9
Razumevanje potreba korisnika 9
Definisanje vizije projekta 15
Analiza i modelovanje poslovnog domena 16
Modelovanje poslovnih objekata 17
Modelovanje poslovnih slučajeva korišćenja 18
Modelovanje poslovnih procesa 19
Validacije poslovnih modela 21
Definisanje korisničkih zahteva 23
Zahtevi 23
Validacija zahteva 25
Tehnike dokumentovanja zahteva 27
Upravljanje zahtevima i promenama 35
Konfiguracija zahteva 35
Promene zahteva 36
Odobravanje promena 37
Sledljivost zahteva 39
Procene vremena i napora na osnovu zahteva 40
Zaključak 41

2. Arhitektura i dizajn 43
Inicijalni dizajn 45
Domenski model 46
Dizajn korisničke interakcije 49
Modelovanje podataka 56
Arhitektura softvera 57
Osnovni arhitekturni principi 59
Slojevita organizacija koda 66
MV arhitekturni stilovi 70
CQRS 77

Arhitekture zasnovane na porukama i događajima 79
Mikrokernel arhitektura 87
Arhitekture distribuiranih aplikacija 88
Detaljni dizajn 104
Dizajn interakcija softverskih komponenti 106
Dizajn softverskih komponenti 139
Fizički dizajn podataka 153
Zaključak 165

3. Razvoj programskog koda 167
Radno okruženje 168
Razvoj prototipa 170
Funkcionalni prototip 170
Arhitekturni prototip 171
Kodiranje 172
Standardi kodiranja 173
Konkurentno i distribuirano programiranje 177
Asinhrono programiranje 180
Programiranje tokova podataka 184
Deklarativno programiranje 188
Refaktorisanje 189
Analiza i pregled koda 191
Jedinično testiranje 195
Proces razvoja koda 200
Planiranje iteracije 200
Implementacija funkcionalnosti 202
Definicija kraja posla 206
Operacioni razvoj (DevOps) 207
Otkrivanje defekata 210
Kontrola i praćenje verzija koda 212
Osnove GIT sistema 213
Grananja 215
Distribuirana kontrola koda 220
Integracija koda 223
Kontinualna integracija 224
Isporučivanje koda 225
Podešavanje okruženja 226
Kontinualna isporuka 232
Zaključak 233

4. Testiranje 235
Test infrastruktura 237
Test podaci 239
Funkcionalno testiranje 239
Identifikacija test slučajeva 240
Dizajn test slučajeva 241
Implementacija automatskih testova 247
Izvršavanje test slučajeva 248
Testiranje sigurnosti 252
Testiranje performansi 255
Test opterećenja 256
Stres test 257
Razvoj zasnovan na testovima 259
Manuelno i istraživačko testiranje 262
Upravljanje defektima 265
Trijaža 266
Stanja problema 266
Zaključak 268

5. Upravljanje projektom 269
Planiranje 271
Identifikacija projektnih zadataka 271
Procene vremena i budžeta 272
Definisanje procesa razvoja softvera 273
Kreiranje plana razvoja 287
Praćenje i kontrola 288
Taktičko praćenje aktivnosti 288
Strateško praćenje aktivnosti 289
Upravljanje procesom isporuke 291
Kvantitativno praćenje napretka 292
Upravljanje rizicima 293
Identifikacija i analiza rizika 293
Prevencija rizika 294
Mitigacija rizika 295
Zaključak 296

Zaključak 299

Indeks pojmova 301

Literatura 304



Uvod

Moderno softversko inženjerstvo je složeni proces za koji je potrebno naučiti veliki broj principa, alata i tehnika kako bismo bili uspešni u svom poslu. Stručnost u softverskom inženjerstvu (kao i u ostalim oblastima) se ne stiče učenjem velikog broja tehnologija, alata i jezika ili praćenjem trendova, nego temeljnim poznavanjem suštine i fundamentalnih principa softverskog inženjerstva. Ako stvarno dobro razumete fundamentalne principe, lako ćete primeniti to znanje da naučite bilo koje tehnike ili tehnologije koje su vam potrebne, međutim ako vam nisu potpuno jasne osnove, možete se namučiti da shvatite bilo koju novu tehnologiju jer ona uvodi nešto što bi za vas verovatno bili potpuno novi principi.

Brzina promena koje se dešavaju u softverskom inženjerstvu nas „primorava" da stalno učimo nove tehnologije i metodologije razvoja, kao i da pratimo promene u postojećim. U toj brzini se nekada često previde osnovni principi koje svako mora da zna kako ne bi odabrao neku novu tehnologiju koja izgleda primamljivo, ali nije primenljiva u konkretnom slučaju. Zbog toga je nekada potrebno zaustaviti se na trenutak i utvrditi osnovne principe softverskog inženjerstva kako bi se dobila osnova za učenje novih stvari. Ova knjiga je namenjena svima koji žele da nauče šta sve čini softversko inženjerstvo uz najbitnije elemente i pojmove iz ove oblasti na koje ćete naići tokom karijere softverskog inženjera.

Kada sam počeo da pišem ovu knjigu, početna namera mi je bila da bude namenjena svima koji su tek počeli da se bave softverom. Međutim, shvatio sam da, bez obzira da li smo tek završili fakultet ili imamo već značajnog iskustva, mi uvek počinjemo da učimo neke nove stvari u oblasti softverskog inženjerstva. Kako je vreme odmicalo, u razgovoru sa profesionalcima koji već dugo rade u softverskoj industriji,

shvatio sam da svako na drugačiji način razume koncepte u softverskom inženjerstvu koji zavise od tehnologije, alata, pa čak i strane na koju ih je odveo Google kada su učili o određenom konceptu. Ako ste ikad čuli frazu „a šta tačno podrazumevaš pod...” znaćete da razgovaraju osobe koje imaju potpuno različiti pogled na koncept za koji su verovali da je isti. Proverite da li vaš kolega razume na isti način kao i vi neke „osnovne i dobro poznate” pojmove kao što su SOLID, GRASP, monadi, CI/CD ili DevOps, i možda ćete se iznenaditi ako je shvatio neke od tih pojmova potpuno drugačije od vas. Pored toga, u ovoj knjizi ćete verovatno naći objašnjenja nekih tema koje se predaju na dobrim softverskim konferencijama ili predavanjima, tako da će vam informacije koje dobijete u ovoj knjizi pomoći da spremniji odete na ovakva napredna predavanja i bolje razumete teme o kojima se govori. Zbog toga sam organizovao ovu knjigu tako da bude korisna širokom spektru čitalaca, od studenata i početnika do dugogodišnjih IT profesionalaca, pošto je svima potrebno da se, s vremena na vreme, podsete osnovnih principa. Iako je fokus ove knjige na osnovnim principima, to ne znači da će se objašnjavati samo lake i osnovne stvari. U ovoj knjizi ćete naći objašnjenja velikog broja pojmova i tehnika o kojima slušate na modernim konferencijama i predavanjima.

Ova knjiga je najvrednija početnicima zato što predstavlja prečicu u svet softverskog inženjerstva. Danas se obrazuje sve više ljudi koji će se baviti softverom i to ne samo na fakultetima, već i na kraćim specijalizovanim kursevima. Bez obzira da li ste tek prošli fakultet i naučili veliki broj programskih jezika, naprednih algoritama, objektno-orijentisanih koncepata ili završili kurs specijalizovan za neki poseban programski jezik, tehnologiju ili programiranje veb ili mobilnih aplikacija posle koga ste spremni da počnete da pravite aplikacije, da biste stvarno počeli da radite u softverskoj industriji, morate da steknete dodatna znanja koja čine jednog softverskog inženjera.

Činjenica je da posao softverskog inženjera ne predstavljaju samo klase, algoritmi i tehnologije. Veći deo posla softverskog inženjera čine razumevanje zahteva, refaktorisanje, pisanje testova, pronalaženje defekata, kontinualna integracija koda, procesi razvoja projekta i slično. Ovakvi koncepti su delimično pokriveni u nekim predmetima i kursevima, a postoje i predmeti i kursevi koji su isključivo specijalizovani za neke od ovih oblasti. Međutim, da biste radili kao uspešan softverski inženjer, morate sve ove koncepte uklopiti u celinu i primeniti ih u projektima od početka do kraja.

Cilj ove knjige je da vam da sliku o tome šta su uobičajene aktivnosti koje se svakodnevno obavljaju u softverskoj industriji. Teme iz ove knjige će vas provesti kroz sve najbitnije koncepte kroz koje se prolazi tokom životnog ciklusa softverskog

projekta i kroz ono što bi trebalo da naučite kako biste postali uspešni softverski inženjeri. Sadržaj ove knjige predstavlja skup osnova i najboljih preporuka koje se mogu primenjivati u softverskim projektima.

Mnoge sekcije iz ove knjige obrađuju teme koje je moguće opisati posebnim knjigama, tako da će, osim polazne osnove, ova knjiga pružiti i reference na preporučenu literaturu koju možete da pogledate kada bude bilo potrebno da proširite znanja u nekoj od oblasti koja vas interesuje.

Za vas koji već imate iskustva u softverskom inženjerstvu, ova knjiga može da posluži da učvrstite svoje znanje i prođete kroz oblasti u kojima se osećate manje lagodno. Ova knjiga vam posebno može koristiti prilikom traženja novog posla zato što pokriva znanja koja se očekuje da imate ako već godinama radite u softverskoj industriji. Ako možete da budete sigurni da možete da odgovorite na bilo koje pitanje iz ove knjige, onda ne morate da brinete o neugodnim fundamentalnim pitanjima na intervjuima.

Takođe, ova knjiga može da vam bude veoma korisna ako ste vođa tima u softverskoj kompaniji koji želi da standardizuje znanja svog tima. Ako mislite da neki članovi tima nisu dobro upoznati sa nekom od oblasti u ovoj knjizi, možete da uočite prilike i potrebe za edukacijom. Knjiga je tako koncipirana da predstavlja standardni skup znanja koje bi svaki softverski inženjer u vašim timovima morao da ima. Zbog toga može da bude korisna i kao podsetnik koji bi trebalo da koristite kada proveravate nove ljude na intervjuima.

Sadržaj ove knjige je organizovan po sledećim celinama:

- **Inženjerstvo zahteva** je prva oblast koja obuhvata tehnike razumevanja i dokumentovanja zahteva kako bi se utvrdilo šta je potrebno da se uradi da bi se implementirao softverski sistem. U ovom poglavlju ćete videti kako se prikupljaju i dokumentuju zahtevi u vidu korisničkih priča i slučajeva korišćenja, ali i kako se prate promene zahteva koje traže korisnici.
- **Arhitektura i dizajn** je oblast u kojoj se definiše kako će se implementirati sistem i obuhvata izbor tehnologija i načina za implementiranje sistema u skladu sa zahtevima. U ovom poglavlju ćete videti šta su česte opcije prilikom biranja arhitekture i dizajna, osnovne principe objektnog[1] i funkcionalnog programiranja, principe kao što su GRASP, SOLID, CQRS, kao i neke reprezentativne dizajn šablone.

[1] Iako će biti objašnjeni neki osnovni objektno-orijentisani koncepti, pretpostavka je da znate osnove objektno orijentisanog programiranja.

- **Razvoj programskog koda** predstavlja skup koraka kada se piše programski kôd, kojim se implementira softverski sistem kao i prateće aktivnosti jediničnog testiranja, integracije koda i slično. Pored generalnih pravila kodiranja i jediničnog testiranja, upoznaćete se sa nekim metodama organizacije tokom faze kodiranja, kao što su Kanban i ekstremno programiranje, osnove GIT sistema za kontrolu koda, kontinualne integracije, itd.
- **Testiranje** je proces kojim se osigurava da su implementirane funkcionalnosti ono što je potrebno korisnicima i da se mogu koristiti sa zadovoljavajućim performansama i bez sigurnosnih rizika. U ovom poglavlju ćete se upoznati sa različitim tehnikama testiranja od verifikacije korisničkih zahteva, jediničnog testiranja koda i mokovanja, do stalne verifikacije tokom kontinualnog procesa integracije koda (engl. *Continuous Integration*).
- **Upravljanje projektom** je poslednje poglavlje gde će biti objašnjeno na koji način se mogu organizovati aktivnosti iz ostalih poglavlja u kontinualnu celinu kojom se od zahteva dobija potpuno funkcionalni sistem. U ovom poglavlju će biti objašnjene razlike između formalnih i agilnih metodologija upravljanja softverskim projektima, kao i preporučene tehnike koje se mogu koristiti radi efikasnog upravljanja softverskim projektom.

Bez obzira koliko dugo radite i koliko imate iskustva u izradi softvera, kada završite sa ovom knjigom, imaćete fundamentalno znanje koje je potrebno svakom softverskom inženjeru.

Terminologija

Najpre je potrebno da se upoznate sa terminologijom koja će biti korišćena u knjizi. Iako možda znate dosta termina koji se koriste u softverskoj industriji, bilo bi dobro da se upoznate sa terminima koji će se koristiti. Terminologija će biti predstavljena kroz opis softverskog procesa.

Korisnici sistema su osobe koje imaju neki problem ili potrebu i postoji mogućnost da će im neka **softverska aplikacija** pomoći. Korisnici mogu biti individualne osobe koje koriste softver zato što imaju neke želje ili potrebe koje se mogu zadovoljiti pomoću aplikacije, ili zaposleni u kompanijama gde obavljaju neki posao i gde bi im softverski sistem olakšao posao, povećao efikasnost ili omogućio da urade stvari koje do sad nisu mogli da urade. Bez obzira ko su korisnici, treba uvek imati na umu da se sistem pravi zbog njih. **Domen** je specifična oblast u kojoj rade korisnici i to može da bude osiguranje, finansije, zdravstvo, prodaja i slično. Opis domena je predstavljanje koncepata, pojava i procesa u domenu za koji se piše aplikacija, a koji služi da softverski tim razume šta je potrebno da se uradi. Iako se mogu napraviti

različiti softverski sistemi za iste domene, neka generalna pravila i načini rada u okviru domena se često ne menjaju.

Softverski projekat je skup aktivnosti koje vrši **projektni tim** (ili **razvojni tim**) kako bi napravio softverski sistem na osnovu opisa domena, koji je potreban korisnicima i koji finansira sponzor projekta. Aktivnosti u softverskom projektu se mogu kategorizovati po disciplinama. **Discipline** su kategorije srodnih aktivnosti koje zajedno predstavljaju celinu kao na primer **dokumentovanje zahteva**, definisanje **arhitekture softvera** gde se odlučuje kako će se praviti softver, **kodiranje** tokom kog se piše kôd u nekom programskom jeziku koji predstavlja softverski sistem, **testiranje** gde se proverava da li softver radi u skladu sa očekivanjima, **upravljanje projektom** kojim se aktivnosti u disciplinama uklapaju u celinu, i slično.

Sponzor projekta je jedna ili više osoba ili organizacija koje finansiraju izradu softverskog sistema. Sponzor očekuje neku korist od softverskog sistema kao na primer da će novi sistem povećati efikasnost zaposlenih ili da će slobodni korisnici kupovati ili koristiti softver kako bi rešili svoje probleme. Svaki projekat ima svoju cenu koja predstavlja novac koji će biti uložen na plate, opremu koja je potrebna da bi se sistem razvio itd. Sponzor plaća cenu projekta i očekuje finansijsku dobit od njega. Projekat ima rokove koji određuju kada će se kompletan sistem ili neki njegovi delovi isporučiti korisnicima.

Stejkholderi su osobe koje imaju potrebu za projektom i interes da se on uspešno završi. Korisnici, sponzori, vlasnici softverskih firmi koje implementiraju sistem, pa čak i sam razvojni tim su stejkholderi projekta. Zajedničko svim ovim ulogama je da žele da projekat uspe kako bi nešto dobili.

U nastavku ćete naići na još dodatnih termina koji će biti objašnjeni u sklopu svojih poglavlja.

Proces razvoja softvera

Verovatno najbolji uvod u knjigu može da bude sažet opis poglavlja kroz koja ćete proći. U ovoj sekciji će biti ukratko opisan proces razvoja softvera.

Softverski projekat počinje procesom prikupljanja, analize, dokumentovanja i upravljanja zahtevima. Ovaj proces se često naziva inženjerstvo zahteva. Tokom ovog procesa je potrebno identifikovati buduće korisnike sistema i razgovarati sa njima kako bi se utvrdilo šta je potrebno uraditi za njih. Kao rezultat se dobija skup dokumentovanih zahteva u obliku lista zahteva, korisničkih priča ili slučajeva korišćenja koji softverski tim koristi kako bi počeo da razvija softver.

Dizajn softvera je proces kojim se na osnovu zahteva odlučuje kako će se implementirati softver. Dizajn se obično može podeliti na dve faze – inicijalni i detaljni dizajn. Tokom inicijalnog dizajna se modeluje sistem tako što se na osnovu zahteva pravi skup skica, prikaza korisničkog interfejsa i drugih dokumenata koji pomažu softverskom timu da razume šta i kako treba da uradi, ali i korisnicima da potvrde da softverski tim ispravno razume problem koji treba da reši. Detaljni dizajn predstavlja razvoj tehničke dokumentacije sa detaljima kao što su klasni dijagrami, dizajn šabloni, opisi protokola komunikacije i slično. Neposredno pred razvoj detaljnog dizajna je potrebno definisati arhitekturu sistema kojom se definišu generalni principi koji se moraju poštovati prilikom organizacije koda, struktura i algoritama u projektu.

Nakon što se napravi dizajn, može se početi sa razvojem programskog koda. Implementacija koda nije izolovana aktivnost i često je tesno povezana sa aktivnostima dizajna i testiranja tako da se ove tri aktivnosti često vrše uporedo. Implementacija, pored pisanja koda, uključuje i aktivnosti pregleda programskog koda, spajanja verzija koda koje napišu različiti članovi tima, kao i isporuku koda korisnicima.

Testiranje predstavlja aktivnost kojom se potvrđuje da softverski sistem radi kao što korisnici očekuju. Jedna od najvećih zabluda je da je testiranje aktivnost koja dolazi na kraju projekta i služi da pronađe probleme u kodu. Testiranje je kontinualni proces koji počinje skoro kada i počne projekat. Svaki dokument i aktivnost u procesu razvoja softvera je predmet testiranja od validacije da su prikupljeni zahtevi relevantni, dobro shvaćeni i dokumentovani, da je dizajn u skladu sa zahtevima, pa sve do konkretnog testiranja implementiranih funkcionalnosti. U slučaju da radite na projektu gde se očekuje da se na kraju projekta samo istestira gotov kôd, koji nije bio predmet ranijih provera, možete da očekujete samo veliki broj problema.

Paralelno sa ovim aktivnostima se odvija proces upravljanja projektom kojim se planiraju i nadgledaju aktivnosti u procesu razvoja. Ove aktivnosti služe da se utvrdi koje aktivnosti će se raditi u okviru projekta, koliki tim je potreban i koliko je vremena i novca potrebno kako bi se uspešno završio projekat. Pored planiranja, upravljanje projektom obuhvata i praćenje da li sve aktivnosti idu po planu, procene da li će se projekat uspešno završiti u okviru predviđenih vremenskih rokova i budžeta, kao i praćenje potencijalnih rizika koji mogu ugroziti projekat.

Opisani proces razvoja softvera će detaljnije biti prikazan u nastavku knjige.



1

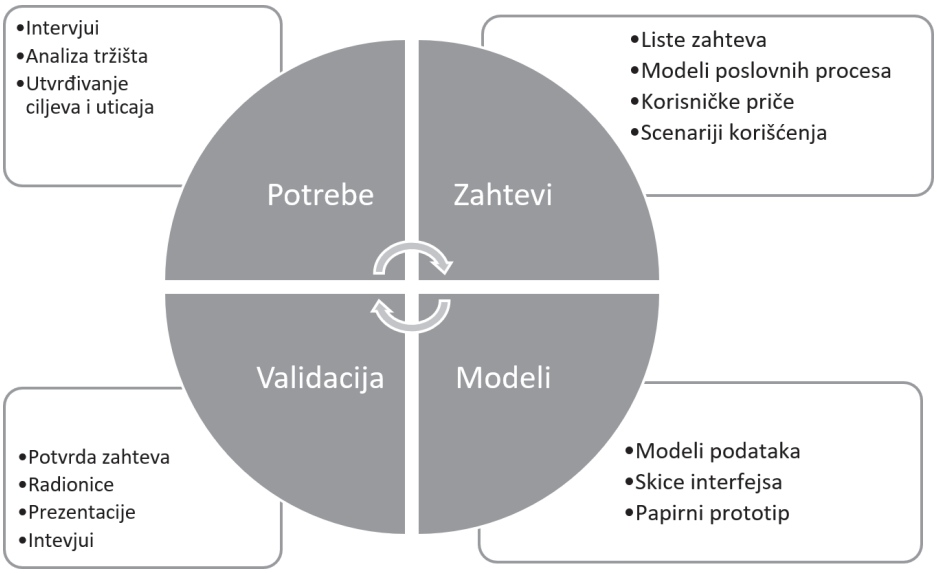
Inženjerstvo zahteva

Zahtevi u softverskom inženjerstvu su početna tačka svakog softverskog projekta. Zahtevi su definicije onoga što korisnici žele tj. onoga što treba da bude urađeno. Inženjerstvo zahteva predstavlja skup aktivnosti kojima se zahtevi prikupljaju od korisnika sistema, analiziraju, uklanjaju nekonzistentnosti i dokumentuju tako da se na osnovu njih može implementirati softverski sistem. Osnovne aktivnosti u ovoj disciplini su (1):

1. Analiza i prikupljanje zahteva tokom koje se razgovorima sa korisnicima utvrđuje šta im je potrebno i koje ciljeve žele da dostignu.
2. Analiza poslovnih procesa kojim se posmatra kako korisnici rade, sa ciljem da se stekne slika o procesima koje obavljaju i tokom koje se opisuju i modeluju ti procesi.
3. Definisanje korisničkih zahteva kojima se dokumentuju korisničke potrebe, zahtevi i scenariji korišćenja na način koji korisnici mogu da shvate i verifikuju ga.
4. Upravljanje zahtevima kojim se reaguje na promene zahteva.

Posebno je potrebno naglasiti da svaka od ovih grupa aktivnosti uključuje validaciju zato što je bitno biti siguran da su informacije koje se prikupljaju validne pre nego što nanesu štetu kasnijim aktivnostima.

Bitno je shvatiti da analiza i prikupljanje zahteva nisu „faza" u procesu razvoja softvera, već iterativni proces koji se može prikazati na sledećoj slici:



Slika 1.1. Iterativni proces analize i prikupljanja zahteva

Trenutno ne postoji jedinstvena metodologija ili tehnika kojom se prikupljaju sva potrebna znanja od korisnika, kako bi se na osnovu tih informacija napravio softverski sistem. Zahtevi se prikupljaju intervjuisanjem korisnika, utvrđivanjem njihovih ciljeva i potreba, kao i analizom tržišta. Potom se informacije dokumentuju i opisuju tekstualno, grafički, pa čak i prototipovima koji simuliraju način kako će se odvijati interakcija između korisnika i sistema dok pokušavaju da dođu do svojih ciljeva. Ove informacije se dele sa korisnicima kako bi ih validirali putem prezentacija, radionica (engl. *workshops*) ili ponovnih intervjua tokom kojih se demonstriraju prototipovi i prikazuju dokumenti i modeli. Pored validacije, osnovni rezultat ove aktivnosti su novootkrivene potrebe, scenariji i ideje koje se dobijaju od korisnika i koji se koriste kako bi se dokumentovali i modelovali novi zahtevi i nastavila sledeća iteracija.

> U softveru se često primenjuje poseban proces definisanja i dizajna korisničkog iskustva (engl. *UX– user experience design*) kojim se na ovakav iterativni način prikupljaju zahtevi i definiše interakcija korisnika sa sistemom. Ovaj proces obuhvata deo aktivnosti opisanih u ovom poglavlju i predstavlja efikasan način za identifikaciju zahteva i validnog korisničkog interfejsa.

U narednim sekcijama će detaljnije biti objašnjene ove aktivnosti. Treba imati u vidu da su neke od aktivnosti opisane u procesu na prethodnoj slici deo discipline dizajna koje će biti predstavljene u narednom poglavlju. Ovo pokazuje da se u modernim softverskim procesima stalno prepleću aktivnosti iz različitih disciplina softverskog inženjerstva.

Analiza i prikupljanje zahteva

Prva faza u bilo kom poslu je razumevanje **šta** je potrebno da se uradi bez ulaženja u detalje **kako** će se uraditi posao. Da bi se shvatilo šta je potrebno uraditi u nekom softverskom sistemu potrebno je shvatiti šta rade budući korisnici sistema, šta je potrebno uraditi kako bi im se pomoglo i dokumentovati dovoljnom količinom detalja njihove potrebe i zahteve. Tokom analize i prikupljanja zahteva se uobičajeno vrši više aktivnosti:

1. **Razumevanje potreba korisnika** kojima je potrebno uočiti šta tačno korisnici sistema žele, zašto im je to potrebno, kao i da li im je određeni zahtev zaista potreban.
2. **Kreiranje vizije projekta** koji predstavlja početnu tačku projekta gde se definiše šta je cilj projekta i koliki je opseg posla.
3. **Analiza poslovnih procesa** predstavlja skup aktivnosti kojima se posmatra šta korisnici trenutno rade i kako se ti procesi mogu preslikati u softverski sistem koji će se implementirati.
4. **Analiza i definisanje zahteva** je glavna aktivnost u ovom procesu tokom koje se sakupljaju i precizno dokumentuju zahtevi koje je potrebno implementirati.
5. **Upravljanje zahtevima i promenama** predstavlja kontinualni proces kojim se definiše šta treba uraditi u slučaju promena originalnih zahteva.

U softverskim timovima postoje posebne uloge koje se bave ovakvim poslovima sa različitim nazivima uloga (engl. *System Analyst, Program Manager, Product Owner, Requirement Engineer*) i njihova glavna uloga je komunikacija sa korisnicima, dokumentovanje i validacija njihovih potreba, procesa i zahteva.

Razumevanje potreba korisnika

Kao što je već rečeno, prvi korak u svakom projektu je razumevanje potreba osoba koji će koristiti softverski proizvod kao i potreba svih osoba koji imaju neki interes da se projekat implementira – ove osobe se nazivaju stejkholderi (engl. *stakeholder*). Bez razumevanja potreba, softverski sistem koji se pravi je samo realizacija nečije pretpostavke ili ideje kako bi nešto trebalo da radi, sa verovatnoćom da te pretpostavke nemaju veze sa realnim potrebama korisnika kojima je namenjen. Ovakav proces je često osuđen na neuspeh.

Korisnici su nekada unapred poznati, a u drugim slučajevima ih je potrebno identifikovati. Korisnici mogu da budu predstavljeni kao uloge (npr. student, profesor), a nekada se predstavljaju kao tzv. persone (engl. *Personas*) – osobe sa imenima,

slikama i opisima koje predstavljaju zamišljenog korisnika. Identifikacija korisnika je prvi korak u procesu razumevanja potreba.

Pošto se identifikuju korisnici i stejkholderi, potrebno je identifikovati koji su njihovi **ciljevi** i kako im se može pomoći da dođu do tih ciljeva. Proces razumevanja potreba uvek mora da bude orijentisan ka ciljevima kojima korisnici teže kako bi im se pomoglo da dostignu te ciljeve. Postoje dve vrste ciljeva:

1. **Poslovni ciljevi** predstavljaju ciljeve organizacija u svrhu zarade novca, smanjenja troškova, poboljšanja performansi i slično. U ovom slučaju korisnici su često zaposleni na obezbeđenju usluga kojima se dolazi do ciljeva, a stejkholderi su menadžment ili vlasnici organizacija koji očekuju da će zaposleni pomoću softvera brže, lakše i efikasnije doći do poslovnih ciljeva. Iako ciljevi mogu biti da se ubrza i optimizuje rad zaposlenih, stvarni cilj je ostvarenje poslovnog cilja.
2. **Korisnički ciljevi** predstavljaju lične ciljeve korisnika sistema kojima im se olakšava rad ili obezbeđuje neka vrednost. U ovom slučaju se softverski proizvod pravi zato što će korisnici koji dostignu ciljeve doneti neki prihod kupovinom softvera. Personalne aplikacije za rad sa podacima (Microsoft Word, Excel, PowerPoint), zabavu ili učenje, kao i veliki broj sajtova koji obezbeđuju informacije, korisnicima omogućavaju da ostvare svoje lične ciljeve.

Bitno pitanje na koje treba odgovoriti je kako navesti korisnike da dođu do svojih ciljeva, pošto se ne može pretpostaviti da će prihvatiti bilo koji način da dođu do njih. Veza između korisnika i ciljeva su tzv. **uticaji** (engl. *impacts*) koji predstavljaju razloge zbog kojih će korisnici svojevoljno promeniti trenutno ponašanje kako bi se došlo do ciljeva. Na korisnike je moguće uticati tako što se identifikuju i implementiraju funkcionalnosti koje će ih voditi do željenog cilja.

Osnovni metod prikupljanja informacija je razgovor sa korisnicima u vidu intervjua. Rezultat intervjua je identifikacija načina na koje će se pozitivno uticati na korisnike da dođu do cilja. U procesu razgovora sa korisnicima bitno je identifikovati korisnike koje treba intervjuisati, definisati šta je potrebno naučiti od njih i na koji način se mogu organizovati intervjui i sesije na kojima bi se prikupili verodostojni podaci (2). Intervjui se vrše postavljanjem pitanja koja mogu da budu:

1. **Zatvorenog tipa –** kojima se očekuje konkretan odgovor. Tipični odgovori na ova pitanja su „Da", „Ne" ili neki broj ili vrednost koja opisuje odgovor, uz eventualno prateće objašnjenje. Cilj ovakvih pitanja je potvrditi ili opovrgnuti pretpostavke ili dobiti konkretne metrike kojima će se opisati sistem. Ovakva pitanja obično počinju frazama „Da li..." „Koliko ..." i slično, a teže da prikupe veoma precizne odgovore.

2. **Otvorenog tipa –** u kojima se očekuje opisni odgovor koji se ne može unapred predvideti. Cilj je prikupiti znanje, informacije i ideje kojima će se kasnije definisati zahtevi. Pitanja obično počinju frazama „Kako bi..." „Šta bi trebalo da..." i slično, a navode korisnika da otvoreno priča o problemima. Često je tišina vrsta pitanja otvorenog tipa, naročito u intervjuima putem „konferencijskih poziva" (Skype, telefon i slično) zato što korisnici koji su odgovorili na pitanje, a ne dobiju potvrdu da je intervjuer shvatio, teže da nastave da pričaju i daju nove informacije.

Intervju treba početi pitanjima otvorenog tipa koja navode korisnika da dâ što je moguće više informacija, a pitanja zatvorenog tipa treba koristiti samo kao potvrdu kada neka izjava nije jasna. U inicijalnim intervjuima se mogu prikupiti osnovne informacije sledećim pitanjima:

1. Ko će da koristi sistem i zašto? Ko ima koristi od sistema koji će se napraviti?
2. Šta su ciljevi koje korisnici žele da postignu?
3. Na koji način korisnici trenutno dolaze do ciljeva (tj. trenutni poslovni procesi) i kako bi u idealnom slučaju želeli da dođu do cilja (možda korišćenjem nekog drugog postojećeg softvera)?
4. Da li postoje neki eksterni faktori koji utiču na korisnike u vidu zakona ili pravila?

Ovo su pitanja otvorenog tipa, njima se prikupljaju različite informacije pomoću kojih se dalje formulišu pitanja kojima će se detaljnije shvatiti potrebe korisnika. Odgovorima na ova pitanja se dobijaju osnovne informacije o sistemu koje predstavljaju osnovu za dalju analizu. Postoji više načina na koje članovi softverskog tima mogu da formulišu pitanja kojima se može shvatiti kako uticati na korisnike da dođu do ciljeva:

1. Počevši od ciljeva, identifikuju se načini kako se može uticati na korisnike putem funkcionalnosti koje bi se implementirale u softverskoj aplikaciji. Potom se bira minimalni skup funkcionalnosti koje je potrebno implementirati kako bi se došlo do cilja. Proces kojim se od ciljeva putem korisnika i uticaja identifikuju funkcionalnosti se naziva mapiranje uticaja (engl. *impact mapping*) (3).

Primer

U slučaju da je cilj smanjiti vreme koje nastavno osoblje fakulteta troši tokom ispitnog roka može se uticati na njih da automatski pripreme ispitne rokove tako što im se obezbede funkcionalnosti kojima lako unose zadatke, predaju ih studentima elektronski kada ispit počne i omogući se da se automatski pregledaju odgovori studenata.

2. Analiza trenutnih procesa i načina kako se rade određene aktivnosti se vrši razgovorom sa korisnicima i posmatranjem njihovog načina rada. Ukoliko korisnici već imaju neke načine da dođu do ciljeva, verovatno je već izvršen neki uticaj na njih, tako da je korisno identifikovati te uticaje. Pored toga, bitno je utvrditi kako se postojeći procesi mogu optimizovati. Optimizacija je najbolji način da se korisnici navedu da pređu sa starog načina rada na novi zato što u novom načinu rada vide dodatne vrednosti ili olakšice u radu.

Primer

U slučaju da se razvija sistem koji omogućava studentima da elektronski prijavljuju i polažu ispite, može se posmatrati kako studenti trenutno prijavljuju ispite, koje podatke unose, kako plaćaju upis, koji zakoni i pravila važe u ovom procesu. Analizom ovih aktivnosti može se doći do informacije kako treba implementirati softver. Ako se još i identifikuje način kojim će studenti lakše prijavljivati ispite koji bi se implementirao u novom softverskom sistemu, to bi bila značajna motivacija trenutnim korisnicima da pređu sa starog na novi sistem.

3. Analiza zakona i pravila po kojima se vrše procesi, pošto oni primoravaju korisnike da rade na određeni način. Pored ustaljenih procesa korisnika, postoje i eksterna ograničenja u vidu pravila, zakona i regulativa koji se moraju ispoštovati. Uvek postoji verovatnoća da korisnici nisu svesni ograničenja koja utiču na njihove aktivnosti.
4. Analiza postojećih ili konkurentskih softverskih sistema koji implementiraju slične procese. Velika je verovatnoća da su konkurentski proizvodi izvršili analizu načina na koje mogu da utiču na korisnika, tako da utisci koji se mogu dobiti od njihovih korisnika mogu da budu bitna informacija o uticajima koje aplikacija koja se implementira treba da izvrši na nove korisnike.

Čak i ako se zahtevi i funkcionalnosti identifikuju mapiranjem uticaja, analizom konkurencije ili nekom drugom metodom, neophodno je prikupiti i validirati zahteve intervjuisanjem i razgovorom sa korisnicima. Na primer, može se pretpostaviti da će automatski sistem za ocenjivanje studenata olakšati nastavnom osoblju da pregleda zadatke, ali je neophodno validirati tu pretpostavku i otkloniti rizik da postoji neki razlog zbog koga korisnici neće želeti da koriste pretpostavljeno rešenje. Ove pretpostavke se mogu validirati tako što se intervjuišu korisnici i predstavi im se idejno rešenje za koje se pretpostavlja da će im olakšati posao (u vidu prezentacije ili prototipa), a onda se kroz razgovor utvrđuje da li će prihvatiti takvo rešenje ili ih nešto sprečava da ga koriste. Ako postoji neka prepreka, potrebno je ili naći drugačije rešenje ili se fokusirati na otklanjanje prepreke.

Tokom intervjua, potrebno je pustiti korisnike da pričaju o načinima kako rade i svojim problemima, bez uplitanja i sugestija (2). Sugestije koje korisnici čuju mogu uticati na način kako razmišljaju i navesti ih da pričaju o hipotetičkim problemima i rešenjima koja u stvari nisu najprioritetnija u trenutnom poslovanju. Iako određeni zahtevi neće biti implementirani, a neki čak ni nemaju smisla, bitno je sakupiti sve moguće informacije kako bi se kasnije odlučilo u kom pravcu će se ići.

Jedna od najvećih opasnosti u procesu razumevanja potreba su korisnici ili softverski timovi koji umesto komunikacije, slušanja i prihvatanja tuđeg mišljenja pokušavaju da eksplicitno definišu drugoj strani šta bi trebalo da se uradi. Softverski timovi koji imaju iskustva u domenu nekada imaju potrebu da umesto slušanja potreba korisnika predlažu postojeća rešenja, često kako bi iskoristili postojeći programski kôd ili smanjili troškove. Na ovaj način oni pokušavaju da prilagode poslovni proces koji obavljaju korisnici svojim potrebama ili jednostavno pretpostavljaju da na osnovu svog iskustva u radu sa velikim brojem sličnih klijenata bolje znaju šta je potrebno novom klijentu. Sugestijama softverski tim može da nametne svoje rešenje koje možda nije rešenje problema korisnika.

> **Hajzenbergov princip neodređenosti** je pravilo u nuklearnoj fizici koje govori da što više energije uložite kako biste odredili gde se tačno nalazi neka nuklearna čestica, manja je verovatnoća da se ta čestica tu stvarno i nalazi. Sličan princip važi u procesu razumevanja zahteva – što više energije uložite da predložite rešenje, manja je verovatnoća da je to ono što je stvarno potrebno korisniku.

Pored softverskog tima koji može da nametne pogrešno rešenje i sami korisnici nekada traže rešenja za koja nekada nisu ni svesni da su loša. Zbog toga ne treba prihvatiti svaku informaciju koja se sazna od korisnika, nego je potrebno razumeti zašto korisnicima treba neki zahtev, da li im uopšte to treba ili su se samo navikli na neke procese pa veruju da je postojeća akcija neophodna, kao i da li postoje neke potrebe ili alternativni načini rada kojih još nisu svesni.

Primer

Zahtev koji opisuje kako vanredni student prijavljuje ispit uključuje složeni protokol uplate ispita sa komplikovanim integracijama sa eksternim platnim sistemom. Međutim, može se postaviti pitanje da li je to stvarno neophodno. Kao alternativa, može se predložiti da studenti uplate prijavljene ispite naknadno kao preduslov za upis sledećeg semestra ili prijavljivanje ispita u sledećem roku. Fakultet ima uvid o prethodnim uplatama, student može da uplati ispit kad mu odgovara u definisanom roku. Ako je mali rizik da će student potencijalno izvršiti prevaru tako što će napustiti fakultet i ugasiti svoj prijavljeni račun kako bi

izbegao plaćanje jednog ispita iz prethodnog semestra uz rizik tužbe, promena zahteva olakšava implementaciju i vreme izrade na obostranu korist.

Poseban rizik nametanja pogrešnog rešenja predstavljaju korisnici koji „previše dobro" znaju kako bi softverski sistem trebalo da radi. Iako nisu deo softverskog tima, često pokušavaju da „programiraju" softverski tim koji bi po njihovim uputstvima implementirao softver. Takvi korisnici ne govore o svojim potrebama nego pokušavaju da dizajniraju rešenje kako misle da treba. Međutim, ako nisu svesni tehničkih ograničenja dizajna, ovakav način komunikacije može da dovede do problema u kasnijem stvarnom dizajnu softvera.

Često je potrebno sagledati širu sliku poslovnih procesa i utvrditi kako se pojedinačni zahtevi uklapaju u ostatak procesa, da li su protivrečni drugim zahtevima i koliku korist donose svim korisnicima.

Pored konkretnih korisnika sistema, postoje i stejkholderi koji neće svakodnevno koristiti sistem, ali će imati neku korist od njega. Ta korist može biti finansijska za vlasnike preduzeća koja naručuju softverski sistem, manji broj problema i brži rad za menadžere korisnika sistema, lakše održavanje za administratore sistema, mogućnost nastavka projekta za vlasnike softverskih firmi koje implementiraju softver. Razumevanje potreba stejkholdera se može razlikovati od razumevanja potreba korisnika zato što je potrebno shvatiti zašto stejkholderi žele da se razvije softver i razviti ga u skladu sa njihovim potrebama.

> Potrebe stejkholdera se nekada zanemaruju i timovi se fokusiraju na "prave korisnike" zato što je njima najbitniji softver i oni deluju kao najočigledniji izvor informacija. Međutim, ovo može da bude kobno po projekat pošto stejkholderi, a ne korisnici, odlučuju o nastavku projekta. Na primer, ako se zanemari potreba menadžmenta organizacije koja naručuje softver u vidu profita koji treba ostvariti softverom, projekat se može otkazati bez obzira na vrednost koju donosi korisnicima.

Neizostavni deo procesa razumevanja zahteva je konstantna validacija i utvrđivanje uticaja svakog zahteva na korisnike i poslovne aktivnosti (engl. *impact mapping*) (3). Validacijom zahteva se odbacuju zahtevi koji nisu potrebni, dok se analizom uticaja dodeljuju prioriteti zahtevima. Ovaj proces je bitan zato što omogućava timu da iz velikog skupa zahteva koji dolaze od korisnika odabere one za koje veruje da su bitni korisnicima i da će se koristiti. Na ovaj način, tim se može prvo fokusirati na najbitnije zahteve i početi da ih dokumentuje i opisuje sa više detalja pomoću korisničkih priča, scenarija i sličnog, dok manje prioritetne zahteve odlaže za kasnije ili čak zanemaruje.

Cilj ovih aktivnosti je da softverski tim razume šta treba da uradi, zašto pravi softverski sistem, koji su tačno problemi koje je potrebno rešiti, ograničenja kojih treba biti svestan i, na kraju, na koji način će novi softverski proizvod pomoći korisnicima i stejkholderima. U ovoj fazi nije neophodno detaljno shvatiti sve procese i aktivnosti, ali je potrebno identifikovati šta je potrebno da se uradi kako bi se implementirao sistem koji će pomoći korisnicima.

Definisanje vizije projekta

U projektima je često potreban jedan kratak dokument koji će opisati šta su ciljevi projekta. Ovo je polazna tačka za svakog ko se uključuje u projekat kao i za korisnike i sponzore koji bi trebalo da sagledaju šta će se raditi u projektu (4). Ovakav dokument se često naziva „Vizija projekta" (5) i sadrži neke od sledećih celina:

1. Definicija problema – kratak pasus koji opisuje šta je tačno problem korisnika koji treba da se reši. Primer može da bude suviše komplikovan i birokratski proces prijavljivanja i pregledanja ispita.
2. Vizija – obično rečenica koja svima jasno govori šta je cilj projekta i kako će promeniti trenutno stanje. Primer može da bude „Cilj je da se bar 5 puta smanji vreme koje studenti i profesori troše tokom ispitnog roka".
3. Korisnici sistema – spisak svih uloga ili persona koji će koristiti sistem (npr. student, profesor), kao i generalni ciljevi (engl. *high-level goals*) koje oni žele da postignu korišćenjem sistema (npr. polaganje ispita, upisivanje naredne godine). Pritom, u ovu listu ne treba uključiti specifične ciljeve (npr. prijavljivanje ili uplata ispita) koji su rezultati nekih specifičnih aktivnosti i kojima se samo dolazi do glavnih ciljeva.
4. Funkcionalnosti – spisak funkcionalnosti i procesa koji će biti implementirani u sistemu uz kratak opis. Ovde ne treba navesti sve moguće funkcionalnosti koje su tražili korisnici, nego samo najbitnije funkcionalnosti kojima će korisnici doći do ciljeva. Ovaj skup predstavlja okvir funkcionalnosti (engl. *Scope*) koji bi trebalo da bude implementiran.
5. Tehnologije (opciono) – u nekim slučajevima je potrebno na početku naglasiti kakve tehnologije će se koristiti npr. da će biti napravljena i veb i mobilna aplikacija, da će sistem biti projektovan za Azure oblak, koja baza će se koristiti i slično. Često se ovakve odluke ne moraju doneti odmah i mogu se doneti u kasnijim fazama projekta, ali nekada klijenti imaju zahteve u pogledu tehnologija.

Vizija projekta je početna tačka u softverskim procesima kao što su UP (engl. *Unified Process*) (5) ili MSF (engl. *Microsoft Solution Framework*) (6). Pored vizije projekta, često je potrebno napraviti i tzv. poslovnu viziju koja je više fokusirana ka

stejkholderima i prikazuje im kako će se njihove potrebe zadovoljiti i generalno unaprediti poslovanje (smanjenjem troškova, povećanjem profita i slično). Ovo može da bude poseban dokument namenjen isključivo stejkholderima ili predstavljati nekoliko dodatnih sekcija u dokumentu vizije koji opisuju poslovnu opravdanost projekta. Ove sekcije mogu biti:

1. Poslovno opravdanje – sekcija koja opisuje zašto je projekat bitan stejkholderima za njihovo poslovanje i kakvu korist mogu da očekuju.
2. Analiza tržišta – analiza koja pokazuje kako će sistem uticati na tržište, kakve će promene doneti, koliki je potencijal da će se koristiti i u drugim slučajevima i slično.
3. Konkurencija i postojeća rešenja – sekcija koja opisuje slična rešenja koja postoje na tržištu, razlike, prednosti i mane u odnosu na ostala rešenja. Ova sekcija bi trebalo da objasni zašto se ne koriste druga rešenja umesto pravljenja novog softvera i kako će softver promeniti način poslovanja u odnosu na konkurenciju.

Ove sekcije moraju da osiguraju da poslovni stejkholderi i klijenti veruju da će im projekat doneti neku korist kako bi odobrili njegov nastavak.

Analiza i modelovanje poslovnog domena

Poslovni domen je opis načina rada korisnika sistema ili poslovanja organizacija i obuhvata terminologiju i opise podataka i procesa koje treba implementirati.

Pošto se shvate ciljevi projekta, osnovne potrebe korisnika i stejkholdera i dobije odobrenje za početak projekta, potrebno je shvatiti poslovne procese u kojima će softverski proizvod raditi. Cilj razvoja većine softverskih sistema je automatizacija postojećih procesa i zamena postojećih procesa novim procesima. Za neke projekte je potrebno shvatiti kompletan način poslovanja organizacije za koju se radi (npr. univerziteta), što uključuje načine kako se ostvaruju prihodi, sa kojim eksternim organizacijama se radi i slično.

Modelovanje je proces kojim se analiziraju postojeći procesi, podaci i načini rada i predstavljaju grafički kako bi ih članovi tima lakše shvatili, a korisnici validirali.

> Jedna od velikih zabluda je da je modelovanje proces koji zahteva da neko otvori poseban program ili alat za modelovanje i nacrta dijagrame ili UML modele kojima će opisati procese. Modelovanje procesa se često najefikasnije vrši na tabli ili čak na papiru gde se istovremeno crtaju i diskutuju modeli procesa, i po potrebi slikaju, kako bi se ručno nacrtani model ubacio u neki dokument. U modelovanju dajte prednost brzini, a ne formalnim

procesima modelovanja zato što je modelovanje često samo prelazna aktivnost koja ubrzava proces shvatanja zahteva. Jedini izuzetak su posebne metodologije razvoja kao što su razvoj zasnovan na modelima (engl. *Model driven development*) gde se generiše programski kôd na osnovu modela gde je ipak potrebno implementirati modele kao integralni deo procesa razvoja.

Tehnike kao što je BMC (engl. *Business model canvas*) omogućavaju da se sagleda i shvati šira slika poslovanja organizacije i na osnovu toga identifikuju poslovni ciljevi koje je potrebno dostići ili optimizovati (a samim tim i učesnici, uticaji i funkcionalnosti kojima se na optimalniji način dolazi do ciljeva). UML i njegove varijacije je široko prihvaćena metoda za grafičko opisivanje detalja objekata (npr. relacija među objektima, najbitnijih informacija) i procesa za specifične poslovne domene (uplate, finansije, rad sa studentima).

U nastavku će biti objašnjene neke osnovne tehnike modelovanja poslovnog domena, kao što su:

1. Modelovanje poslovnih objekata kojima se identifikuju koncepti i podaci sa kojima će se raditi u aplikaciji kao i relacije među njima.
2. Modelovanje poslovnih slučajeva korišćenja kojima se identifikuju procesi koji se obavljaju u sistemu i korisnici koji učestvuju u procesu.
3. Modelovanje poslovnih procesa kojima se opisuju detalji procesa u vidu dijagrama aktivnosti.

Bez obzira koliko je formalan ili dokumentovan proces razvoja koji koristite, uvek je potrebno uložiti bar neko minimalno vreme da se naprave ovakvi modeli kako bi se shvatio kontekst u kome rade korisnici.

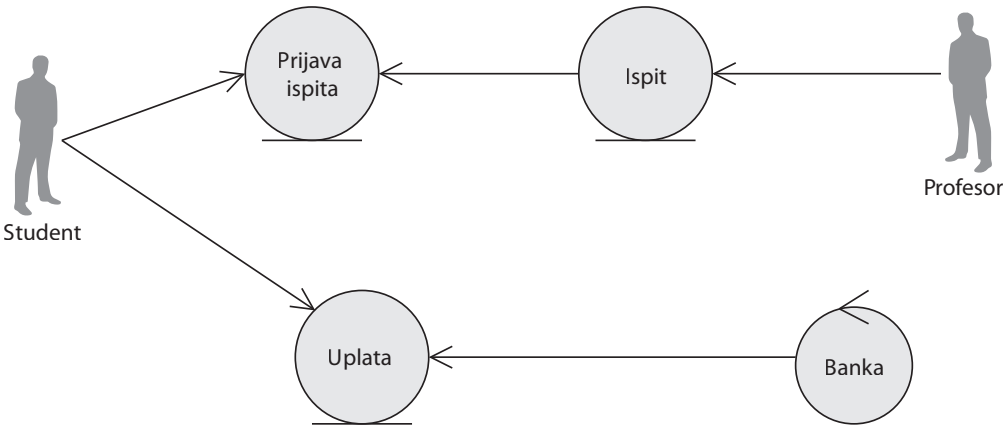
Modelovanje poslovnih objekata

Kao prvi korak u razumevanju poslovanja, potreno je identifikovati koncepte i terminologiju koja se koristi u poslovnim procesima. Ovo uključuje tzv. poslovni rečnik (engl. *business glossary*) kojim se obezbeđuje da članovi softverskog tima mogu lako da shvate terminologiju poslovnog domena i model poslovnih objekata koji predstavlja skup koncepata sa kojima se radi, kao i veze među njima. Na primer, student, profesor, predmet, ispit, prijava ispita i služba za plaćanje su primeri koncepata koji se pojavljuju u domenu univerziteta. Postoji nekoliko vrsta poslovnih objekata:

1. Poslovni učesnici koji predstavljaju uloge korisnika u procesima, kao na primer profesor i student.

2. Poslovni objekti koji predstavljaju koncepte sa kojima se radi, kao na primer predmet, ispit i prijava.
3. Poslovni servisi koji predstavljaju procese koji obavljaju neke funkcije koje su bitne za funkcionisanje sistema kao što su „banka", „služba za plaćanje" i „sistem za pregled ocena".
4. Poslovni interfejsi koji predstavljaju objekte ili servise kojima poslovni korisnici komuniciraju sa sistemom. To mogu biti delovi sajta, bankomati, šalteri i slično.

Poslovni objekti se često prikazuju vizuelno u obliku modela poslovnih objekata. U ovakvom modelu se prikazuju koncepti kao i veze među njima.

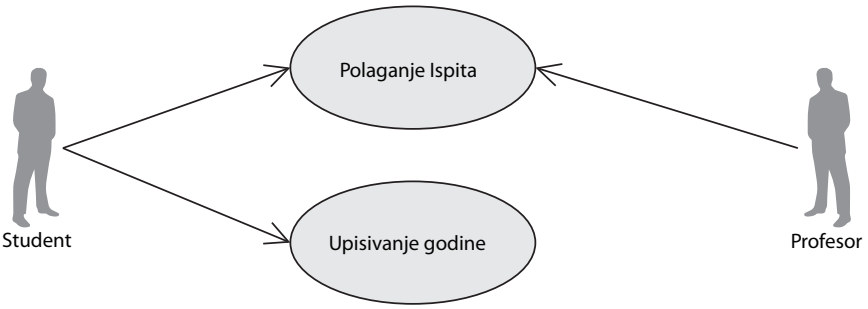


Slika 1.2. Model poslovnih objekata Univerziteta

Bitna karakteristika poslovnih objekata je da ih svi korisnici sistema mogu lako shvatiti i prepoznati. Ovi objekti nikada ne treba da budu specifične komponente ili funkcije koje shvataju samo članovi softverskog tima. Model poslovnih objekata se kreira kako bi korisnici potpuno shvatili kakvu sliku ima softverski tim o trenutnim procesima i verifikovali da je tim ispravno shvatio procese. Koncepti koji eventualno neće biti jasni korisnicima su suvišni u ovakvim modelima.

Modelovanje poslovnih slučajeva korišćenja

Poslovni slučajevi korišćenja (engl. *Business Use Cases*) (5) ili epovi (engl. *Epics*) kako se nekad nazivaju, predstavljaju opise generalnih funkcionalnosti ili procesa koje korisnici moraju da obavljaju u svakodnevnom poslu kako bi došli do nekih ciljeva. Ovo su generičke funkcionalnosti koje često nemaju neke konkretne detalje i akcije koje se vrše, nego generalno opisuju neke procese.



Slika 1.3. Model poslovnih slučajeva korišćenja

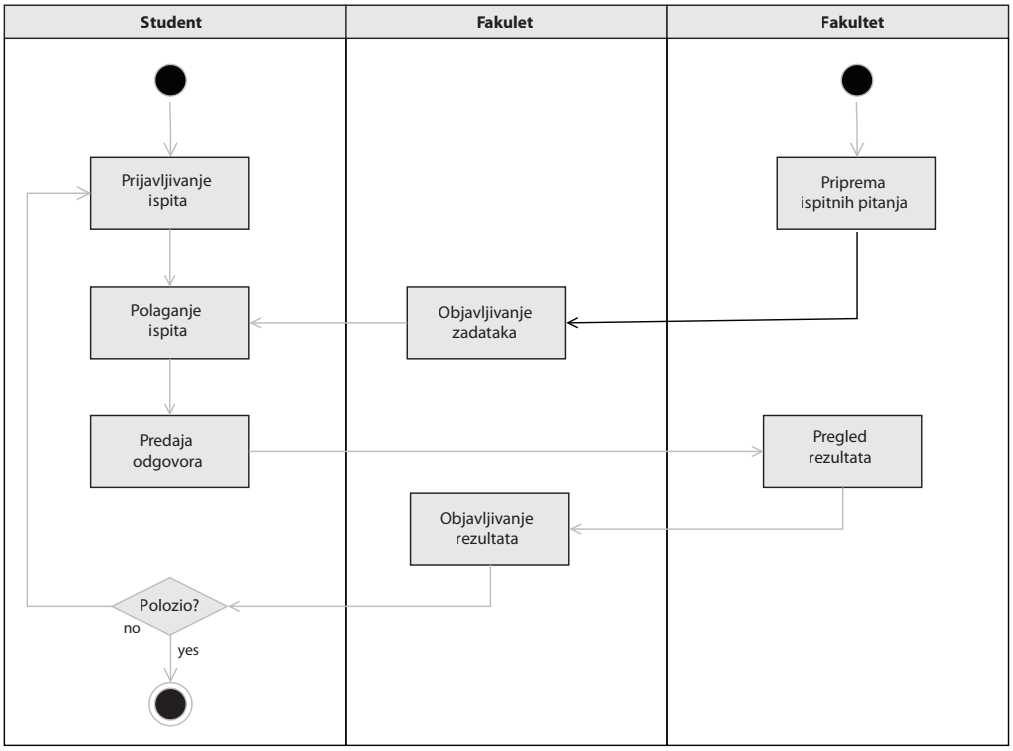
U slučaju univerziteta ove funkcionalnosti mogu da budu „Polaganje ispita", „Upisivanje godine" i slično. Ove funkcionalnosti se ukratko opisuju i prikazuju na modelima poslovnih slučajeva korišćenja gde se može videti koji poslovni korisnici učestvuju u određenim poslovnim aktivnostima.

Često je bitno nabrojati i prikazati ove funkcionalnosti (obično u dokumentu vizije projekta) kako bi svima bilo jasno šta će se raditi, bez ulaženja u detalje na koji način se vrše ovi procesi (tj. da li se plaćanje vrši na šalteru, elektronski, mobilnom aplikacijom) i koji sve poslovni korisnici učestvuju u procesima. Kao i u slučaju modela poslovnih objekata, ne treba praviti veliki broj dijagrama sa ogromnim brojem detalja pošto je cilj da se sa samo nekoliko dijagrama predstave osnovni koncepti u sistemu.

Modelovanje poslovnih procesa

Poslovni slučajevi korišćenja i procesi se mogu detaljnije modelovati i opisati kako bi se korisnici uverili da je softverski tim ispravno shvatio kako korisnici žele da koriste sistem. Modelovanje procesa predstavlja (najčešće) vizuelno opisivanje aktivnosti u procesu, kao i tokova kako se aktivnosti izvršavaju ili odluka koje je potrebno doneti kako bi se neke aktivnosti izvršile.

Dijagrami poslovnih aktivnosti u kojima aktivnosti izvršavaju različiti korisnici ili sistemi se mogu podeliti po linijama ili vertikalama (engl. *swim lanes*) tako što se po jedna vertikala dodeli svakom korisniku ili sistemu koji učestvuje u poslovnom slučaju korišćenja, a aktivnosti koje on izvršava se postavljaju u njegovu vertikalu. Ovakav način modelovanja nije potreban u slučaju kad poslovni proces obavlja jedan korisnik koji ima interakciju direktno sa softverskim sistemom. Linije su dodeljene poslovnim korisnicima ili sistemima koji su opisani u viziji projekta i ne bi trebalo da budu specifični sistemi i komponente kao što su štampač ili telefon.



Slika 1.4. Dijagram procesa polaganja ispita

Aktivnosti u modelima poslovnih procesa su poslovne logičke celine koje su prepoznatljive korisnicima i ne predstavljaju detaljne operacione aktivnosti i događaje kao na primer „prikazuje se dijalog koji potvrđuje da je prijava ispita snimljena". U slučaju da je neophodno prikazati neku specifičnu aktivnost, treba težiti da se ona što apstraktnije opiše (npr. „Student se obaveštava da su rezultati objavljeni", umesto konkretnih „Šalje se imejl studentima sa sadržajem..." ili „Student se obaveštava SMS porukom").

> Apstrakcijom i uopštavanjem zahteva i opisa aktivnosti se odlažu konkretne odluke i omogućava se korisnicima da daju svoje mišljene o zahtevima. U slučaju da je aktivnost bila opisana sa: "šalje se imejl studentima sa sadržajem...", čitaocu možda ne bi palo na pamet da bi možda SMS bio bolje rešenje, ali apstraktniji opis aktivnosti navodi čitaoce da razmišljaju šta bi im više odgovaralo.

Optimizacija poslovnih procesa

Izrada softverskog proizvoda ne predstavlja samo automatizaciju trenutnih poslovnih aktivnosti, već često i potpunu promenu procesa rada. Retko će se dogoditi da

korisnici žele da novi softver bude identična kopija nekog postojećeg sistema, samo u novoj tehnologiji. Pored toga, nove tehnologije mogu da otklone ograničenja koja su ometala/usporavala postojeće procese i primoravala korisnike da na neefikasniji način rade svoj posao. Uvek treba iskoristiti priliku da se shvati trenutni proces i identifikuju optimizacije.

> Optimizacije su bitne i za softverski tim zato što nema potrebe trošiti vreme na implementaciju nepotrebnih funkcionalnosti nasleđenih iz starih sistema ili načina razmišljanja.

Korisnici nekada nisu ni svesni da im nova tehnologija ili nova implementacija funkcionalnosti omogućava da rade posao na efikasniji način, a nekada su se toliko navikli na postojeće procese da odbijaju promene. Međutim, optimizacija procesa je bitna, kako za korisnike, tako i za stejkholdere koji finansiraju projekat, a koji su često direktno nadređeni korisnicima, tako da se preko stejkholdera utiče na korisnike. U slučaju da se pravi softver za široku populaciju korisnika gde stejkholderi nemaju uticaj na njih, potrebno je uveriti stejkholdere da bi trebalo uraditi neke optimizacije i predočiti koje su očekivane koristi od tih optimizacija. U ovom slučaju se prihvata rizik da optimizacije i promene ne utiču na korisnike.

Validacije poslovnih modela

Pošto se završe poslovni modeli, potrebno ih je prikazati korisnicima i skejkholderima kako bi se dobila povratna informacija i potvrda da se može nastaviti sa projektom. Tokom ovih aktivnosti potrebno je izvršiti sledeće validacije:

1. Validirati da je softverski tim ispravno shvatio postojeći poslovni proces i da nije propustio neke bitne korisnike ili poslovne aktivnosti.
2. Validirati ispravnost poslovnog modela i procesa koji će biti implementiran. Često se dešava da modeli pokazuju probleme u postojećim poslovnim procesima i načinima rada kao što su suvišne aktivnosti, aktivnosti koje bi trebalo ukloniti ili optimizovati i slično. Veliki broj ovakvih problema je uzrokovan postojećim softverskim sistemima ili pravilima i zakonima koje treba zameniti, a nekada su posledica činjenice da su korisnici navikli da rade na određeni način i ne primećuju da je loš. Implementacija novog softverskog sistema nije automatizacija i kopiranje trenutnog stanja sistema, nego inovacija i poboljšanje i zbog toga je neophodno osigurati da sve loše i nepotrebne aktivnosti neće ući u novi sistem. Treba imati na umu da poslovni procesi nikada nisu savršeni i da će implementacija postojećih loših poslovnih aktivnosti prouzrokovati probleme u budućnosti.

3. Potvrditi da korisnici shvataju šta će biti urađeno u projektu, a šta neće. Ovo je izuzetno bitno zato što različiti korisnici imaju različite potrebe i očekivanja i može se desiti da očekuju da će se implementirati neka funkcionalnost koja nije planirana. Svaki projekat ima granice u vidu skupa funkcionalnosti koje će biti implementirane i potrebno je osigurati da svi shvataju koje su to granice, inače se može desiti da softverski tim dobije veliki broj zahteva koji nisu relevantni za trenutni softverski sistem.

Definisanje granica ne podrazumeva da će u ranoj fazi biti tačno definisano šta će biti implementirano i da će biti odbijen svaki zahtev za promenu ili sugestija[2]. Moderne agilne metodologije razvoja softvera u kojima se rado prihvataju predlozi za promene i dodatne informacije koje dolaze od korisnika sistema upravo zahtevaju od softverskih timova da stalno prikazuju trenutni sistem korisnicima i traže smernice za dalji razvoj. Međutim, čak i u takvim metodologijama razvoja je potrebno postaviti neke granice koje govore koje vrste promena su prihvatljive, a koje zahtevaju potpuno novi sistem.

Primer

U slučaju projekta za automatizaciju ispita na univerzitetu se u planirane procese polaganja ispita i kolokvijuma mogu uključiti dodatne funkcionalnosti kao što su automatizacija polaganja prijemnih ispita, automatizacija pregledanja ispitnih zadataka i slično. Međutim, obračun naknada za dežurstva, praćenje dodatnih radnih sati na ispitima i bonusa nastavnog osoblja ne mora da bude implementiran u sklopu sistema. Ove poslovne aktivnosti mogu biti kritične za univerzitet i neki stejkholderi i korisnici verovatno smatraju da se bez takvih funkcionalnosti ne može koristiti sistem. Implementacija bilo kojih poslovnih zahteva koji nisu definisani u viziji projekta mogu skrenuti fokus tima u više pravaca i „razvodniti" kvalitet funkcionalnosti trenutnog projekta.

Kent Bek je definisao YAGNI pravilo (engl. *You Ain't Gonna Need It*) (7) u svojim principima ekstremnog programiranja (XP) kojim se preporučuje da pretpostavite da neki zahtev neće biti potreban, sem ako ne postoji dokaz koji ga opravdava. Zahtevi u viziji projekta su veoma verovatno neophodni za projekat, inače ne bi bili u prvom bitnom dokumentu koji ga opisuje.

[2] Ovakav način rada je odlika najstriktnijih formalnih metodologija u kojima je unapred definisano šta će se raditi i gde bilo kakva promena uzrokuje promene ugovora i cena. Ovakve striktne metodologije često loše utiču na razvoj projekta i dovode do softverskih sistema koji ne odgovaraju korisnicima.

Pored validacije samih procesa, potrebno je izvršiti i validaciju prioriteta procesa. Kao što postoje procesi koji su suvišni i štetni, tako i među procesima koji su korisni može postojati razlika u vidu činjenice da su neki procesi ipak bitniji od drugih. Prioritet procesa daje timu smernice koje govore koje procese je potrebno ranije implementirati i prikazati korisnicima.

Definisanje korisničkih zahteva

Pošto se kompletira poslovni model i verifikuje da su sve strane u procesu svesne šta će biti urađeno, potrebno je detaljno analizirati, prioritizirati i dokumentovati korisničke zahteve u poslovnim procesima, kako bi ih softverski tim pretočio u softverski proizvod koji radi na osnovu tih zahteva.

Postoje dva glavna načina kako se vrši analiza zahteva:

1. Formalne metodologije – detaljna analiza zahteva se može izvršiti pre početka bilo kojih drugih aktivnosti. Ovakav način rada je dobar za male projekte ili projekte gde je potrebno što je moguće ranije razrešiti nedoumice u vidu načina implementacije ili cene i trajanja projekta, koje mogu biti ključne za nastavak projekta.
2. Agilne metodologije – detaljna analiza zahteva se može vršiti kontinualno tokom celog projekta tako što se u svakom trenutku detaljno analizira samo određeni poslovni proces, kako bi softverski tim što pre mogao da počne da radi na njemu.

Zahtevi

Zahtev je jedna ili više rečenica koje precizno opisuju šta sistem treba da uradi i pod kojim uslovima. U slučaju da se rečenice razlikuju ili opisuju podslučajeve, potrebno ih je podeliti na posebne zahteve. Često je potrebno imati nedeljivu, odnosno atomičnu specifikaciju zahteva u kojoj nema podslučajeva koji se mogu dodavati ili uklanjati zato što je onda teško ispratiti šta je tačno potrebno uraditi kako bi se implementirao zahtev. Atomični zahtevi su nedeljivi i fokusirani na specifično ponašanje sistema i mogu se ili implementirati ili ukloniti iz sistema. Zbog toga je bitno da zahtevi budu što je moguće koncizniji kako niko ne bi tražio njihovu izmenu.

U praksi se često koristi SMART kriterijum za kvalitet zahteva. Zahtev je pametno napisan ako zadovoljava sledeće osobine:

- S (engl. *Specific*) – zahtev mora da bude veoma precizan i da opisuje tačno ono što softverski tim treba da uradi i ono što korisnik očekuje. Svi opisi i