SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Karmen Klarin

Programsko inženjerstvo

skripta

Split, 2012.

Autor:
Mr.sc. Karmen Klarin, predavač

Recenzenti:
Mr.sc. Ivica Ružić, viši predavač
Toma Rončević, predavač

Lektor:
Ivanka Kuić, prof.

Izdavač:
Sveučilište u Splitu, Sveučilišni odjel za stručne studije
Kopilica 5, Split

Odobreno odlukom povjerenstva za izdavačku djelatnost.

SADRŽAJ

1. Uvod u programsko inženjerstvo

Informacijski sustav i programsko inženjerstvo
Složenost programskog inženjerstva
Vrste programskih proizvoda

1.1. Informacijski sustav i programsko inženjerstvo

Informacijski sustav (IS, eng. *Information System*) je dio svakog poslovnog sustava čija je funkcija neprekidna opskrba svih razina upravljanja, odlučivanja i svakodnevnog poslovanja potrebnim podacima/ informacijama [11].

Informacijski sustav se razvija za realni poslovni sustav pa su temeljne aktivnosti informacijskoga sustava prikupljanje/ razvrstavanje, obrada, čuvanje i raspoređivanje (distribucija) informacija/ podataka svim radnim razinama poslovnog sustava.

IS je redovito kompleksan sustav pa je rad na projektiranju i razvoju programske potpore poslovnom sustavu nekog poduzeća složen organizacijski i tehnološki proces u kome sudjeluje veći broj ljudi. Znanja i vještine oblikovanja i projektiranja programske potpore moraju biti potpomognuti dobro definiranim sredstvima za projektiranje, kao i organizacijom procesa odnosno metodologijom projektiranja.

Računarstvo je prije svega inženjerska struka, te je primjena sistematiziranog, discipliniranog i kvalitetnog pristupa razvoju, izradi i održavanju programske potpore definirana kao **programsko inženjerstvo** (eng. *Software Engineering*) [9]. Programsko inženjerstvo pokušava objediniti računalnu znanost i inženjerske principe koji su prisutni već stoljećima u rješavanju opipljivih problema s izrazito materijalnim osobinama.

> Primjerice, izgradnja mosta je posao u kome je precizno definirano što je potrebno izgraditi i kako se to radi. Posao izgradnje mosta temelji se na prokušanoj i ustaljenoj inženjerskoj disciplini *mostogradnja*. Posao se sastoji od zadataka koje treba obaviti, ljudi koji će to napraviti i roka do kada most treba biti gotov. U praksi, prilikom konstrukcije, može doći do manjih problema koji se uglavnom bezbolno rješavaju.

Iz ovog primjera je vidljivo da je izgradnja mosta većinom rutinska, a manje kreativna vještina. Međutim, programsko rješenje nekog poslovnog problema je nematerijalne prirode, a problem koji treba riješiti često je nepoznat informatičarima. Poznate metodologije razvoja IS-a podržavaju inženjerski pristup razvoju, te je često njihova primjena nužna jer pospješuje rješavanje problema. Ipak, kako je oblikovanje programskog rješenja kreativan posao koji se ne zasniva na standardima to je više reakcionarna disciplina jer se do cilja dolazi pokušajima i promašajima.

Moglo bi se reći da je u klasičnim inženjerskim disciplinama sve poznato, za razliku od programskog inženjerstva u kome je dosta toga nepoznato. Može se zaključiti da, ma koliko disciplinirani bili u inženjerskom pristupu razvoju programske potpore, ipak nam treba dosta dodatnih vještina i kreativnosti da bi svladali i naučili probleme o kojima u početku ne znamo mnogo; a bez poznavanja problema ne možemo pristupiti njihovu rješavanju.

Bez obzira na to što postoji neizvjesnost usvajanja znanja o poslovanju kod informatičara, može se reći da programsko inženjerstvo obuhvaća [4]:

- poslove i zadatke kojima se oblikuje programsko rješenje;
- sustavnu primjenu alata i tehnika na cjelokupni proces razvoja programskih rješenja;
- ciklus razvoja koji obuhvaća: analizu i specifikaciju postupaka koje treba programirati, izradu samih programa, načine testiranja, pisanje projektne i programske dokumentacije te postupke uvođenja u rad kod korisnika.

1.1.1. Programsko rješenje kao proizvod

Svaki projekt razvoja **programskog proizvoda** (u širem smislu se govori o razvoju informacijskog sustava[1]) nastaje kao rezultat potreba nekog poslovanja; bilo da je riječ o potrebi popravka/ nadgradnje postojećih programa, o potrebi da se gotovo programsko rješenje uvede u poslovanje, ili potrebi da se izradi potpuno novi programski proizvod. Na početku projekta poslovne potrebe su izražene u neformalnom obliku i rezultat su preliminarnih jednostavnih poslovnih razgovora među (ruko)voditeljima zainteresiranih strana. U takvim razgovorima programi i razvoj cjelokupnog IS-a jedva se i spominju, iako znamo da su se sve zainteresirane strane sastale kako bi se pronašlo rješenje korisnikovih problema.

Sa stanovišta informatičara (programskog inženjera, eng. *software engineer*) proizvodi su programskog inženjerstva programi, baze podataka i pripadajuća programska dokumentacija. Međutim, sa stanovišta korisnika programskog rješenja programsko inženjerstvo daje proizvod čiji rezultati su informacije koje poboljšavaju i unapređuju nečiji posao.

Danas je tehnologija računalnih aplikacija jedna od najvažnijih na svijetu kojoj se još uvijek ne mogu sagledati krajnje granice. Sigurno nitko 50-tih godina prošlog stoljeća nije mogao ni slutiti da će informacijska tehnologija, naglasak je na programima, biti nezaobilazna u poslovanju, znanosti, inženjerstvu. Osim primarnog doprinosa, informacijska tehnologija je omogućila stvaranje nekih novih tehnologija kao što je genetsko inženjerstvo, zatim je potaknula proširenje postojeće tehnologije, primjerice

[1] Kada se naglašava složenost i cjelovitost IT podrške poslovanju, koristi se pojam informacijski sustav (IS, eng, *Information System*). U ovoj skripti se većinom naglašava i opisuje rad na izradi programa kao proizvoda, odnosno kao rezultata rada informatičara. Dakle, ova skripta se bavi proučavanjem inženjerskog pristupa razvoju programskog proizvoda, te će ovisno o konkretnosti ili općenitosti biti riječi o programskoj potpori, odnosno o informacijskom sustavu.

telekomunikacije, dok je, s druge strane, ukinula ili marginalizirala neke stare tehnologije, kao što je štamparska industrija.

Tih 50-tih godina prošlog stoljeća nitko nije mogao ni zamisliti da će programski proizvodi biti sastavni dio djelatnosti kao što su medicina, vojska, industrija, promet, telekomunikacije, zabava, nabrojati bi se moglo mnogo toga. Uz sve to, danas je još i internet kao široko rasprostranjeni virtualni prostor omogućio promjene i napredak svega, od pretraživanja literature do trgovanja i kupnje putem interneta.

Posljedica je tog brzog, i sada već dugotrajnog širenja računalnih programa u sve segmente našeg života, da danas imamo mnoštvo programa koje treba prilagođavati, održavati i popravljati. Stoga, što vrijeme više prolazi, to je izgledno potreban veliki broj informatičara koji će te programe održavati, možda veći nego što ih je sveukupno trebalo da ih izrade i uvedu u rad.

Kako je važnost programskih proizvoda rasla, to je informatička zajednica nastojala razviti tehnologije koje će olakšati, ubrzati i pojeftiniti izradu i održavanje računalnih programa. Cilj pojedinih tehnologija bila su specifična područja primjene (primjerice dizajn i implementacija web stranica). Druge tehnologije su se posvetile području tehnologije (primjerice objektno-orijentirano programiranje), dok su neke namijenjene širokoj uporabi (primjerice operacijski sustav Linux).

Danas velika industrija programskih proizvoda postaje važan (dominantan) čimbenik u ekonomiji industrijaliziranog svijeta. Tako su prijašnji programeri sada zamijenjeni timovima informatičkih specijalista od kojih je svaki zadužen za jedan dio tehnologije; a svi zajedno moraju ispuniti zahtjeve složenog sustava. Ipak, cijelo to vrijeme razvoja programskih rješenja informatičaru se postavljaju stalno ista pitanja, a neka od karakterističnih su:

- Zašto je potrebno toliko dugo vremena da se završi program?
- Zašto je razvoj programa tako skup?
- Zašto se ne mogu pronaći i ispraviti sve pogreške programa prije nego što se isporuči korisniku?
- Zašto informatičari troše toliko mnogo vremena i napora za održavanje postojećeg programa?
- Zašto je teško izmjeriti napredak koji donosi uvođenje i korištenje programskog rješenja?

Ova i slična pitanja karakteriziraju proizvodnju programa i značenje njihova razvoja, a odgovore na njih može dati, između ostaloga, praktična primjena programskog inženjerstva.

1.1.2. Inženjerski pristup

Namjena računalom podržanog poslovnog sustava je da pruži odgovore na sva pitanja povezana sa zadanim poslovnim **područjem** (eng. *domain*). Tako područje poslovanja

odnosno djelatnost poduzeća određuje unutarnju strukturu programske podrške koja uključuje podatke predstavljene pomoću entiteta i veza zadanog područja, te programa koji određuje kako se koristiti i upravljati tim podacima.

Parcijalno gledano, program se sastoji od mnoštva jednostavnih dijelova (koda) koji se mogu (bolje reći, moraju moći) povezati na mnogo različitih načina kojima se rješavaju različiti složeni problemi. Stoga promatranje i shvaćanje pojedinog jednostavnog dijela programske podrške ne mora izgledati zahtjevno. Ako promatramo pojedini jednostavni dio, teško je reći zašto je cjelina tako kompleksna. Međutim, programska potpora poslovanju mora se nositi sa stvarnim složenim problemima poslovnog područja za koje je namijenjena.

Programsko inženjerstvo je proces koji nastoji riješiti složene probleme tako da ih raščlani na jednostavnije dijelove koji se mogu analizirati, optimizirati, oblikovati (eng. *design*) prema računalnim postavkama i pravilima i na kraju izgraditi kao gradbeni blokovi (eng. *building block*) novog složenog informacijskog sustava.

***PRIMJER*:** Slikoviti prikaz složenog posla koji se sastoji od jednostavnih elementarnih zadataka može se naći u mnogim praktičnim granama poslovanja. Ovaj primjer opisuje izradu drvene ograde oko kuće [16].

Izrada ograde se može podijeliti u četiri jednostavna zadatka: postavljanje kolaca, rezanje drvene građe, bojanje greda i spajanje greda (nisu posloženi po redoslijedu odvijanja). Poznato je da zadatak rezanja drva mora prethoditi koraku spajanja i koraku bojanja, a korak rezanja prethodi koraku spajanja. Ako se u problem uvede parametar vremena potrebnog za izradu pojedinog koraka, onda se mogu zadati još neka ograničenja. Primjerice, postavljanje kolaca zahtijeva 3 vremenske jedinice po komadu, rezanje greda traje 2 vremenske jedinice po komadu, bojanje 5 za neizrezano i 4 za izrezano drvo, te spajanje 2 za neobojano i 3 za obojano drvo.

Postavlja se pitanje kojim redoslijedom odraditi zadatke tako da se projekt završi u najkraćem mogućem vremenu? Naravno da za rješavanje problema treba poznavati i parametre veličine okućnice i količine drvene građe koju treba upotrijebiti.

Bit ovog primjera je uočiti da bez zabilježenih mogućih opcija rješavanja problema te njihove optimizacije ne možemo sagledati mogućnosti i rješenja. Inženjerski pristup uključuje postavljanje problema i prepoznate elementarne korake, te njihovu međusobnu ovisnost, što je već napravljeno i zabilježeno u prethodnim pasusima ovog primjera (i zove se analiza problema). Nadalje, postoje razne inženjerske metode optimizacije kojima se rješava ovaj problem. Može biti više rješenja te je na naručitelju ograde da odabere ono koje mu najviše odgovara. I na kraju, u realizaciji pojedinačnih zadataka može doći do određenih problema koje treba znati riješiti.

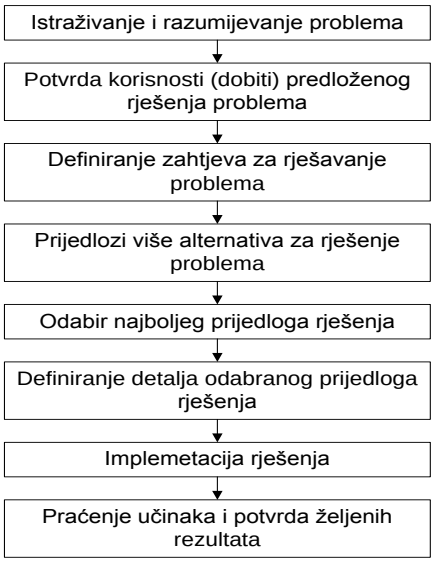
Razvoj programske potpore odnosno, šire, razvoj cjelokupnog informacijskog sustava je redovito složen (kompleksan) posao. Praksa pokazuje da većina teškoća i nesporazuma u analizi i projektiranju novog sustava proizlazi iz toga što korisnik često nije u mogućnosti precizno obrazložiti *što* i *zašto* radi (odnosno opisati postojeći sustav) te iznijeti što želi (odnosno precizne zahtjeve koji se postavljaju pred novi

informacijski sustav). Korisnik često poznaje samo svoj uski dio posla, ali nema uvida u način djelovanja i svrhu postojanja svih funkcija koje informacijski sustav zahvaća.

Stoga projektant mora (1) rekonstruirati postojeći poslovni sustav i definirati strukturu budućeg IS-a te (2) voditi računa o daljnjim potrebama i mogućnostima integracije svakog podsustava u cjelokupni informacijski sustav.

1.1.3. Analitički pristup rješavanju problema

Na početku se projektant fokusira na razumijevanje poslovnih problema te nastoji otkriti i skicirati način na koji će problemi biti riješeni (kao u prethodnom primjeru). Rješavanje prepoznatih problema korištenjem informacijske tehnologije je samo dio priče o razvoju IS-a. Cjelokupnost i složenost analitičkog pristupa **rješavanju problema** prikazana je na slici 1.1. i opisana u nastavku [21].



Slika 1.1. Analitički pristup rješavanju problema

Analitički pristup rješavanju problema opisan je u sljedećim koracima:

- Projektant prvo mora istražiti i razumjeti problem: tko od zaposlenih je uključen, koje poslovne procese sadrži te jesu li i kako ostali sustavi (vlastiti ili vanjska poduzeća, institucije i sl.) povezani s rješavanjem zadanog problema.
- Rješenje problema koje predloži projektant treba za rukovodstvo predstavljati dobit koja će biti veća od troška samog rješavanja. Događa se da je ponekad preskupo rješavanje nekog problema te je bolje odustati od takvog rješenja.

- Ako je rješenje problema izvedivo, projektant treba definirati zahtjeve koji opisuju što treba riješiti: koje specifične ciljeve treba ispuniti, koje podatke treba pohraniti u sustav i koji podaci se koriste iz sustava te pomoću kojih procesa se odvija tijek podataka. U ovom trenutku je važno definirati *što* treba uraditi. *Kako* će se to uraditi za sada nije od presudne važnosti.
- Za ovako detaljno definirane zahtjeve treba razviti nekoliko prijedloga rješenja. Svako od predloženih rješenja (alternative) treba pažljivo razmotriti, posebice jer su najčešće alternative rješenja razne varijante izbora fizičkih komponenti koje moraju pružiti odgovor na pitanja: koje su komponente potrebne i gdje će biti smještene, koja tehnologija će se koristiti i kako će komponente komunicirati preko mreže i slično.
- Odabir najboljeg prijedloga rješenja uglavnom se vodi preporukom najmanjeg rizika i najveće korisnosti. Uz to treba voditi računa i o strategiji poduzeća i ustanoviti doprinosi li odabrano rješenje osnovnim ciljevima tog poduzeća.
- Definiranje detalja odabranog rješenja predstavlja rad projektanta na detaljnoj specifikaciji tog rješenja oblikovanjem novog IS-a. Oblikovanje novog sustava obuhvaća baze podataka, korisnička sučelja, programske module i procedure te umrežavanje.
- Modeli detaljnog oblikovanja koriste se za izradu novog sustava, između ostaloga i za izradu programskog koda i implementaciju sustava kod korisnika.
- Uspješna implementacija označava početak rada na novom IS-u pa se nadalje prate učinci rada podržanog novim sustavom i, po potrebi, izmjene i nadopune postojećih rješenja.

Pristup rješavanju problema koji je gore naveden predstavlja uobičajeni inženjerski pristup rješavanju problema. Stoga je jasno zašto se još naziva i *programsko inženjerstvo*. Navedeni koraci su u osnovi zajednički bilo da se rješava problem gradnje mosta ili izrade IS-a.

1.2. Složenost programskog inženjerstva

Poznato je iz raznih izvora da uspješnost razvoja IS-a nema zavidnu razinu[2]. Zašto je programsko inženjerstvo tako složen posao?

S obzirom na ubrzani razvoj i sveprisutnost informacijske tehnologije u poslovnom svijetu, uporaba informacijskog sustava treba biti jedan od ključnih čimbenika u poslovanju poduzeća. Korištenje IS-a treba povećati međudjelovanje (eng. *interoperability*) sa drugim sustavima, jer se informacijski sustavi povezuju s klijentima, pružateljima usluga te ostalim subjektima preko telekomunikacijskih mreža. Što se tiče poslovne strategije poduzeća, IS postavlja temelj za razvoj novih proizvoda

[2] Prema http://www.standishgroup.com (21.12.2011.) neki od objavljenih rezultata su: samo 34% projekata razvoja IS-a je završeno uspješno a 17% je bio potpuni neuspjeh (2002.); prosječno prekoračenje rokova je ≈220%, prosječno prekoračenje troškova je ≈190% (1994.), itd.

i usluga te utječe na ostvarivanje konkurentskih prednosti. Također, IS pokreće reinženjering strateški važnih poslovnih procesa te zahtijeva usvajanje i primjenu novih stručnih znanja.

Očito je da se od programskih rješenja očekuje da podržavaju poslovanje, ali i da poboljšaju, unaprijede, ubrzaju i reorganiziraju[3] poslovanje u poduzeću. Informatičari kao inženjeri novog sustava trebaju primijeniti znanje i vještine svoje struke, ali isto tako trebaju savladati znanja *nepoznatih* područja i struka te prilagoditi potrebe korisnika postojećim tehnološkim mogućnostima. Tako problemi postaju složeni, a rješenja nisu uvijek oblikovana na najbolji mogući način.

Neka od gledišta u analizi složenosti razvoja i uspješnosti izrade i implementacije IS-a dana su u nastavku ovog potpoglavlja. Tako se razmatraju:

- složenost strukture sustava koju treba podržati
- problemi uočeni nakon zgotovljenog programskog proizvoda
- ograničenja i karakteristike kroz ciklus razvoja IS-a.

1.2.1. Složenost strukture sustava

Složenost strukture poslovnog sustava koju treba podržati novi IS može se podijeliti u nekoliko pogleda na sustav i njegov razvoj:

- **Funkcionalni** pogled na sustav prikazuje sustav kao skup entiteta koji obavljaju određene zadatke, opis tih zadataka te djelovanje entiteta međusobno i u interakciji s okolinom. Funkcionalni pogled se može razložiti na manje i jednostavnije dijelove te kao takav informatičaru predstavlja polazište za oblikovanje programskog rješenja.
- **Strukturni** pogled prikazuje strukturu komponenti sustava, povezanost sučelja i okoline, te protok informacija među njima. U idealnom slučaju strukturni pogled bi trebao predstavljati razradu funkcionalnog pogleda. Svaki entitet funkcionalnog pogleda treba se razložiti na niz elementarnih komponenti koje se mogu izraditi i provoditi zasebno. Pretvaranje funkcionalnog pogleda u strukturni pogled zadatak je koji se u razvoju IS-a radi u koraku oblikovanja (dizajna) novog sustava. Međutim, struktura novog sustava pod utjecajem je ograničenja resursa koji sprječavaju ili ometaju korištenje proizvoljno mnogo komponenti. Tako u modele oblikovanja novog sustava treba ugraditi ograničenja koja zahtijeva tehnološko rješenje.

[3] Preoblikovanje poslovnih procesa često se naziva i reinženjering poslovnih procesa (eng. *Bussiness Process Reengineering*, BPR). Preoblikovanje poslovnih procesa znači ponovno promišljanje postojećih poslovnih procesa, njihova nova organizacija i ponovno oblikovanje, čime se postižu bitne i kvalitativne promjene [7].

- Prikaz **ponašanja** sustava opisuje način na koji sustav reagira (odgovara): zadani i prihvatljivi ulazi, izlazi kao rezultat obrade ulaza, te rubni uvjeti koje trebaju ispuniti ulazi i izlazi. To uključuje i opis okoline iz koje dolaze ulazi (koja proizvodi ulaze) te koja preuzima i tumači izlaze. Također uključuje i ograničenja radnih svojstava (performansi) koja su nametnuta od strane okoline i funkcija u sustavu. Zahtjevi nad performansama su naročito izraženi kod sustava u realnom vremenu (eng. *real-time system*). Stoga u ponašanje sustava treba ugraditi definicije koje će obaviti potrebne zadatke i očekivane odgovore sustava na te zadatke a u modele kojima se opisuje ponašanje sustava treba ugraditi i zahtjeve nad performansama.

Modeli i metode razvoja programskih rješenja (poglavlje 2.) prepoznaju osnovne korake analize i oblikovanja sustava. U analizi područja poslovanja prepoznaju se funkcionalna područja, koja se onda detaljno razrađuju do elementarnih komponenti. U oblikovanju novog sustava važno je definirati ponašanje sustava i osigurati povezivanje elementarnih komponenti tako da korisnik bude u stanju riješiti sve svoje poslovne zadatke/ probleme korištenjem izrađenih programa.

1.2.2. Problemi gotovog programskog rješenja

Nakon završenog posla na razvoju i izradi programske podrške (kod uvođenja u rad i praćenja učinaka u poslovanju) prepoznati su sljedeći problemi:

- Iako su u programe ugrađene sve planirane funkcionalnosti u zadanom vremenu i za zadani proračun, ipak sustav predstavlja razočarenje za korisnika jer je njegova provedba neučinkovita i ne pridonosi poboljšanju i efikasnosti poslovanja.
- Unatoč nastojanju da se provedu promjene koje omogućava uvođenje nove tehnologije, nikakve se značajne promjene u poslovanju nisu dogodile; poduzeće radi na isti način s novom kao što je radilo i sa starom tehnologijom.

  Primjerice, mogu biti uzaludni napori informatičara da u oblikovanju i izradi novog IS-a ostvare maksimalnu integraciju podataka (i tako osiguraju veću učinkovitost, točnost i pouzdanost informacija). Korisnici novog sustava svoje poslovne zadatke obavljaju kao i prije uvođenja, ne koriste automatizme prijenosa podataka i informacija, nego ručno prenose podatke iz jednog dijela sustava u drugi.
- U nekim slučajevima usvojena nova tehnologija (zajedno sa IS-om) donosi manje djelotvoran radni sustav. Tako se može dogoditi da ERP[4] projekti koji stvaraju značajne prednosti poslovanja poduzeća, u lokalnim jedinicama (podružnice i/ili funkcionalni podsustavi) imaju manju učinkovitost nego prijašnji način rada. Razlog može biti u specifičnostima lokalnih procesa koje ne podržava novi sustav ili se može dogoditi da postavke novog sustava nisu dobro konfigurirane (prilagođene lokalnim zahtjevima i ograničenjima).

[4] ERP sustavi (eng. *Enterprise Resource Planning*) integriraju unutarnje i vanjske informacije o upravljanju kroz cijelo poduzeće. Obuhvaćajući financije/ računovodstvo, proizvodnju, prodaju i servis, upravljanje odnosima s klijentima i slično. ERP sustavi koji predstavljaju integrirana programska rješenja automatski objedinjuju ova

Ovih nekoliko karakterističnih problema slikovito prikazuje kako se informatičari mogu suočiti s problemom neodgovarajućeg proizvoda u koji je uloženo dosta rada i vremena. Dakle, inženjerski pristup rješavanju problema izrade programskih rješenja važan je uvjet da posao bude napravljen stručno, no postoji još cijeli niz uvjeta koji su više-manje karakteristični za razvoj IS-a (za razliku od drugih struka). Suradnja s korisnikom, učenje o korisnikovu poslovanju, pomaganje korisniku da promijeni način poslovanja uvođenjem IS-a te podrška korisniku u proširenju i unapređenju sustava kojim se koristi trebaju biti sastavni dio zadataka poslova programskog inženjera.

1.2.3. Ograničenja i karakteristike razvoja

Inženjeri koji razvijaju IS moraju se nositi s karakteristikama sustava koji će biti izgrađen, ograničenjima koja su realno prisutna u izradi programa i, što je izuzetno važno, oni moraju pratiti proces razvoja IS-a koji je definiralo poduzeće u kojem rade.

Važnija ograničenja i karakteristike u ciklusu razvoja IS-a:

- Ograničenja arhitekture hardvera koja nameće postojeća hardverska tehnologija i sredina u kojoj taj hardver treba raditi. Tako sustavi u realnom vremenu imaju važna ograničenja nad performansama poput pouzdanog rada u teškim uvjetima, te učinkovitog i usmjerenog rada prema vanjskom okruženju. Posljedica ovih ograničenja su hardverske platforme s posebnim karakteristikama kojima treba prilagoditi programsku podršku.
- Ograničenja arhitekture programskih rješenja nametnuta su potrebom za ekonomičnim razvojem i održavanjem IS-a koji mora djelovati pouzdano uz zadana hardverska ograničenja. Tako je implementacija programa često ograničena standardima postojećih operacijskih sustava, programskim jezicima i programskim komponentama. Ovi standardi su često nametnuti zbog smanjenja složenosti, vremena razvoja i cijene razvoja IS-a.
- Ograničenja integracije i testiranja sustava određena su zahtjevima nad performansama, vremenskim rokovima i ekonomskim parametrima uobičajenima u projektu razvoja IS-a. Važno je napomenuti da su zadaci integracije i testiranja novog sustava u stvarnom vremenu često zanemareni prilikom ugovaranja i planiranja projekta i zadataka razvoja IS-a. Integracija i testiranje su važni i teški zadaci zato što trebaju osigurati kvalitetu rada novog sustava u realnom vremenu i u realnim poslovnim uvjetima. Stoga bi integracija i testiranje trebali ispuniti sljedeće uvjete:
    - Informatičari u suradnji s korisnikom trebaju dogovoriti plan i zadatke testiranja, te odgovornosti i zaduženja povezana s tim zadatcima.
    - Dio testiranja informatičari će obaviti u testnom okruženju. No sustav nije u cijelosti testiran dokle god ga korisnici ne primijene u svim zadacima poslovanja čije funkcionalnosti IS treba podržati.

područja i njihove rezultate (informacije). Svrha ERP sustava je olakšati protok informacija između svih poslovnih funkcija u poduzeću i upravljanje odnosima sa vanjskim sudionicima.

  - Testovi bi trebali sadržavati generiranje velikog broja podataka i testiranje izvršavanja algoritama generiranja (prvenstveno provjere optimizacije procesa i brzine izvršenja).
  - Testovi bi također trebali osigurati analizu rezultata koja je usporediva sa stvarnim analizama u poslovanju.
- Ograničenja održavanja i prilagođavanja novog sustava uočavaju se tijekom dužeg vremenskog razdoblja kroz koje se javlja potreba za izmjenama i poboljšanjima. Poboljšanja mogu biti u obliku proširenja funkcionalnosti uzrokovane novim potrebama ili hardverskim platformama koje se često mijenjaju kako bi se iskoristile prednosti novih tehnologija (veća pouzdanost, smanjeni gabariti i slično).

Već je naglašeno da postoje i ograničenja procesa razvoja IS-a jer se programsko inženjerstvo javlja u kontekstu informatičkog poduzeća koje izrađuje programske proizvode zbog stjecanja vlastitog prihoda. Stoga to poduzeće prilikom izrade nekog IS-a treba osim troškova poslovanja uzeti u obzir i troškove održavanja projektiranog IS-a u duljem vremenskom periodu. Tako su voditelji projekata trajno u situaciji da informatičarima nameću ograničenja u razvoju programa, a isto tako i u kasnijem održavanju. Na taj se način može smanjiti rezultat napora informatičara da oblikuje optimalna rješenja, ili da prilagođava već postojeća rješenja novim spoznajama i mogućnostima koje nudi IT.

1.3. Vrste programskih proizvoda

Danas u mnogim industrijama poduzeća ovise o svojim informacijskim sustavima. Intenzivna i dinamična poslovanja kao što su bankarstvo, osiguranje, telekomunikacije, danas ne bi mogla opstati bez IS-a. I u tradicionalnim industrijama kao što su proizvodnja i maloprodaja, postoji rastuća ovisnost o IS-u.

U ovom poglavlju su opisane neke od podjela programskih rješenja ovisno o IT, i kako tu podjelu vide informatičari. S druge strane, postoje podjele kojima se može sagledati raznovrsnost poslovanja poduzeća, to jest kako informatizaciju vidi korisnik.

Podjele ovisno o IT:

- Prema [17] nekoliko suvremenih rasprostranjenih **kategorija računalnih programa**: sistemski, aplikativni, proizvodnih, web i slično (potpoglavlje 1.3.1.).
- Prema [13] važno je proučiti i načine na koje se može nabaviti IS rješenje. To su razne vrste razvoja IS-a koje bi se ukratko mogle sažeti u pitanju: **Izgraditi, kupiti ili unajmiti?**

Poduzećima je IS potreban za svaki dio njihova poslovanja: za svakodnevno poslovanje, za kontrolu i izvješćivanje, za strateško planiranje, te za održavanje poslovnih odnosa s dobavljačima i s kupcima.

Podjele ovisno o poslovanju su:

- Prema [13] konfiguracija IS-a u poduzeću po **funkcionalnim područjima** sastoji od tri velika sustava: sustava za upravljanje resursima, sustava za upravljanje lancem nabave i sustava za upravljanje odnosima s kupcima (potpoglavlje 1.3.3.).
- Prema [21] podjela po **razini i odgovornosti poslovnih zadataka** dijeli sustave na transakcijske, izvršne i sustave za potporu odlučivanju (potpoglavlje 1.3.4.)

1.3.1. Kategorije računalnih programa

Danas sedam kategorija računalnih programa predstavlja kontinuirani izazov za programske inženjere. To su:

- **Sistemski programi** (eng. *System software*): to su skupina programa čija je namjena servisiranje ostalih programa na računalu. Neki od sistemskih programa su *kompajleri, editori*, programi za upravljanje datotekama i slično. Ostali sistemski programi (komponente operacijskog sustava, *driveri*, mrežni programi i slično) obrađuju veliki broj neodređenih podataka. Također, sistemske programe karakterizira veliko međudjelovanje s hardverom, višekorisnički pristup i korištenje operacija koje zahtijevaju dijeljenje resursa, te sofisticirano upravljanje procesima, složenim strukturama podataka i višekorisničkim sučeljima.
- **Aplikativni programi** (eng. *Application software*): sastoje se od samostalnih programskih rješenja koja rješavaju specifične poslovne potrebe. Ovi programi obrađuju poslovne ili tehničke podatke tako da omogućavaju poslovne operacije ili upravljačko-tehničke odluke, tj. zadatke. Kao dodatak konvencionalnim programima za obradu podataka aplikativni programi se koriste za kontrolu poslovnih funkcionalnosti u realnom vremenu. Takvi programi su, primjerice, obrada transakcija na prodajnoj kasi (eng. *point of sale*, POS) ili kontrola procesa proizvodnje u realnom vremenu.
- **Inženjersko-znanstveni programi** (eng. *Engineering/ Scientific software*): sadrže složene algoritme, te inženjerske i znanstvene primjene problema kao što su, primjerice, astronomija, vulkanologija, molekularna biologija, pa sve do automatizirane i robotizirane proizvodnje. Danas su se inženjersko- znanstveni programi odmakli od konvencionalnih numeričkih algoritama. Računalom podržano oblikovanje (eng. *Computer-aided design*, CAD), simulacije sustava i drugi interaktivni programi već imaju karakteristike rada u realnom vremenu i neke druge karakteristike sistemskih programa.
- **Ugrađeni programi** (eng. *Embedded software*): imaju osobine proizvoda ili sustava i koriste se za implementaciju i kontrolu svojstava i funkcija namijenjenih krajnjem korisniku. Ugrađeni programi izvode ograničene funkcije (primjerice, kontrolna ploča za mikrovalne pećnice) ili omogućavaju važne funkcionalnosti i kontrolu sposobnosti (primjerice, digitalne funkcije u automobilu kao što su kontrola protoka goriva, sustava kočenja, ekrana kontrolne ploče i slično).

- **Programi proizvodne linije** (eng. *Product-line software*): oblikovani su tako da različitim korisnicima pruže specifične mogućnosti korištenja i fokusirani su na ograničeno tržište (primjerice, kontrola zaliha proizvoda) ili na masovno tržište (primjerice obrada teksta, proračunske tablice, računalna grafika, multimedija, zabava, upravljanje bazama podataka, ili programi za profesionalnu i poslovnu primjenu).
- **Web programi** (eng. *Web applications*) ili *WebApps:* obuhvaćaju široko područje programa. U najjednostavnijem obliku ovi programi sadrže manji skup povezanih *hipertekst* datoteka koje prikazuju informacije koristeći tekst i ograničenu grafiku. Međutim, kako je rasla važnost elektroničkog poslovanja (eng. *e-business*) i B2B modela (eng. *Business-to-Business*), *WebApps* postaju sofisticirana i složena računalna okruženja koja omogućavaju ne samo samostalne funkcionalnosti računala nego i integraciju s poslovnim bazama podataka i poslovnim programima.
- **Programi umjetne inteligencije** (eng. *Artificial Intelligence software*) su programi koji koriste nenumeričke algoritme za rješavanje složenih problema koji ne podliježu klasičnoj računskoj analizi. Programi ovog područja uključuju robotiku, ekspertne sustave, prepoznavanje uzoraka (primjerice slike ili glasa), umjetne neuronske mreže, dokaze teorema i računalne igrice.

Mnogi programski inženjeri diljem svijeta rade na projektima koji spadaju u neku od nabrojanih kategorija, bilo da grade nove sustave ili da se pomažu u popravljanju, nadogradnji i održavanju postojećih sustava.

Osim ove osnovne podjele, u posljednje vrijeme postoje još neke kategorije programskih proizvoda (uglavnom zahvaljujući rasprostranjenosti interneta i telekomunikacije). Neki od standardnih proizvoda su:

- **Sveprisutno računarstvo** (eng. *Ubiquitous computing*). Koristi se i pojam *opća informatizacija*. To je model interakcije računala i čovjeka u kojem je obrada informacija integrirana u svakodnevne aktivnosti i objekte. Opisuje se i kao "računarstvo prilagođeno ljudima, bez prisiljavanja ljudi da se uče koristiti programima". Ova kategorija računalnih programa je pravi izazov za programske inženjere koji su u mogućnosti razvijati sustave primjenjive za male uređaje, računala i poduzeća, te ostvariti međudjelovanje kroz cijelu široko rasprostranjenu mrežu komunikacija.
- ***Netsourcing*** su programi koji rade na webu. Kako preglednici omogućavaju univerzalni pristup web sadržajima i programima, ova vrsta programa se može izvoditi i od treće strane web poslužitelja (jednako lako kao što može i u okviru internog web servera - intraneta). Izazov za programske inženjere je izgradnja jednostavnih (primjerice, osobno financijsko planiranje) i profinjenih programa koji pružaju pogodnosti korisnicima širom svijeta.

- **Otvoreni kod** (eng. *open source*) predstavlja rastući trend distribuiranja izvornog koda koji je moguće slobodno doraditi tako da korisnici mogu izrađivati lokalne varijante programa. Izazov za programske inženjere je izrada izvornog koda koji je samoopisni (eng. *self-descriptive*), i što je još važnije, da razviju tehnike koje će klijentu i programeru omogućiti da znaju koje izmjene su napravljene i kako će one utjecati na vlastitu verziju programa.

1.3.2. Izgraditi, kupiti ili unajmiti?

Razvoj informacijskog sustava promatra se polazeći od poduzeća za koje se razvija programska podrška (naručitelj IS-a, korisnik), te poduzeća koje se bavi razvojem programske podrške (informatičko poduzeće, proizvođač).

Od koga dolazi inicijativa za uvođenjem novog IS-a (ili programa)? Može se reći da inicijativa dolazi od nekoga tko uoči problem ili nezadovoljavajuću situaciju u poslovanju te traga za boljim rješenjem.

***PRIMJER*:** Neki od problema i situacija koje potiču korisnika da potraži nova informatička rješenja kako bi poboljšali poslovanje su:

- Odjel marketinga smatra da poduzeće reagira presporo na zahtjeve klijenata za razliku od uspješnije konkurencije koja očito ima učinkovit sustav praćenja klijenata. Ispitivanja pokazuju da vlastiti sustav ne pruža konsolidirane preglede po proizvodima i prodajnim mjestima u realnom vremenu.
- Poslovni procesi se ne odvijaju glatko bez obzira na to što je novi sustav nedavno uveden u poduzeće. Izgleda da problem leži u neprilagodbi tijeka procesa poslovanja sa mogućnostima koje pruža postojeći IS.
- Postojeći sustav nema funkcionalnosti prilagođene novim tehnologijama. Tako za prodaju putem interneta IS treba proširiti programima koji omogućavaju takav oblik poslovanja.
- Proizvođači programske podrške ističu svoje nove proizvode i značajne pogodnosti njihove nabave ako ih korisnik nabavi odmah. Korisnici, u strahu da ne propuste nešto važno, prihvaćaju nova programska rješenja.

Postoji više načina kako izgraditi IS, od izrade sustava unutar *kuće*, to jest poduzeća (što je danas relativno rijetko), preko kupnje i prilagođavanja standardnih rješenja koja se prodaju na tržištu, do unajmljivanja informatičara koji će napraviti specifično rješenje.

Jedna je od značajki razvoja IS-a, promatrana od strane korisnika kao poduzeća, da se izrada novih sustava u zadnje vrijeme znatno smanjila. Korisnici se danas sve manje odlučuju za razvoj vlastitog, sebi prilagođenog, IS-a, a sve više pronalaze na tržištu gotova rješenja koja proizvođači i distributeri nakon toga prilagođavaju specifičnostima korisnikova poslovanja.

Postoji više mogućnosti za izradu ili kupnju programskog rješenja:

- prema vlastitim potrebama i poslovnoj tehnologiji **samostalno** razviti i izraditi IS;
- kupiti **gotovo** programsko rješenje;
- prepustiti realizaciju posla trećoj strani, takozvana **eksternalizacija** ili *outsourcing;*
- kupovati pojedine module sustava od raznih poznatih dobavljača za pojedina poslovna područja.

Svaka od navedenih mogućnosti ima prednosti i mane koje moraju biti poznate svim sudionicima u odlučivanju. Odluka o izboru najpovoljnije varijante donosi se na osnovu ekonomskih pokazatelja (ukupne cijene), te ocjene funkcionalnosti i kvalitete sustava.

Ukoliko se poduzeće odluči za **samostalno** razvijanje IS-a tada je izbor odgovarajuće tehnološke infrastrukture poput hardvera, mrežnog softvera, operativnog sustava i slično zahtjevan zadatak koji otežava donošenje ovakve odluke. Važno je naglasiti da je specifično rješenje koje se razvija na zahtjev poduzeća često značajno skuplje, a razvoj i uvođenje IS-a traje dulje nego uvođenje gotovih proizvoda koji se mogu naći na tržištu.

Ako poduzeće kupuje **gotovo** rješenje, u troškove kupnje i korištenja treba uračunati i održavanje, unapređivanje i dogradnju kupljenog sustava. Ovi zahvati mogu značajno povećati ukupnu cijenu gotovog rješenja. Važno je razlučiti tko je vlasnik ovakvih programskih rješenja jer nije svejedno je li program kupljen, je li kupljeno pravo korištenja (licenca) ili je program vlasništvo treće strane (za *outsourcing*). Kako su poduzeća često nemarna u unapređenju programa, gotovo rješenje može biti povoljnije. Još jedan važan čimbenik je i neovisnost poduzeća o jednom dobavljaču, jednom proizvođaču ili jednom voditelju projekta.

Za poduzeća kojima je cilj veća efikasnost, dobar izbor je **eksternalizacija,** jer je vanjski dobavljač vjerojatno stručniji i ima više iskustva u odnosu na interne zaposlenike. Stoga se aktivnosti u poslovnom sustavu lakše fokusiraju na osnovnu djelatnost i ne troše se resursi zaposlenika na razvoj programske podrške. U ovom slučaju je vjerojatno i kvaliteta usluge bolja jer se tehnološke novine brže uvode (budući da ozbiljan partner koji želi opstati na tržištu mora voditi u uvođenju i primjeni dostupnih tehnoloških novina).

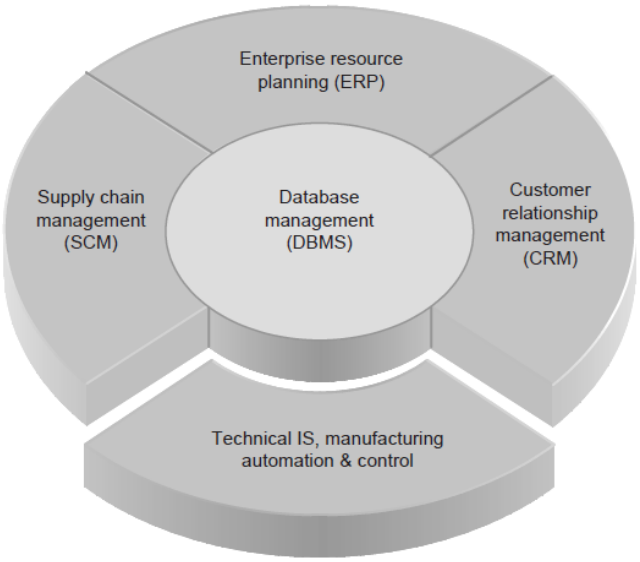
Jedan od problema eksternalizacije je demotiviranost i negativna selekcija informatičkog osoblja, zbog čega kvalitetni zaposlenici često odlaze u druge tvrtke, a ostaju oni nekvalitetni. Na taj način s vremenom dolazi do gubitka specifičnih kompetencija (eng. *know how*) jer se često izručuju vanjskim partnerima, koji zatim to stečeno znanje mogu preprodati konkurenciji. I u ovom slučaju mogu se dogoditi spore prilagodbe zahtjevima poslovanja i nedovoljna fleksibilnost dobavljača, što se uočava u poteškoćama prilikom vrednovanja postignute koristi primjenom ovakvog modela nabave programskog rješenja.

Čest je slučaj da kod kupnje gotovog rješenja poduzeće ima veću produktivnost i manje troškove u usporedbi s razvojem unutar kuće. Međutim, **kombiniranjem** kupnje gotovih rješenja s vlastitim razvojem ili kupnjom rješenja od različitih dobavljača često su javljaju problemi s integracijom aplikacija. U ovakvim slučajevima prisutna je i ograničena fleksibilnost proizvoda, pa u konačnici ukupni troškovi mogu biti znatno veći od procijenjenih.

1.3.3. Funkcionalna područja informacijskog sustava

Ako se izrađuje ili već postoji na tržištu cjeloviti IS koji će neko poduzeće implementirati u svoje poslovanje, onda se standardni moduli (sustavi) za ERP (eng. *Enterprise Resource Planning*, sustav za upravljanje resursima), SCM (eng. *Supply Chain Management*, sustav za upravljanje lancem nabave), CRM (eng. *Customer Relationship Management*, sustav za upravljanje odnosima s kupcima) prilagođavaju zahtjevima korisnika. Danas ti moduli teže integraciji, primjerice SCM modul ima pristup informacijama iz ERP modula. Stoga se može očekivati da će se svi koristiti logički integriranim bazama podataka (unutar sustava za upravljanje bazama podataka), slika 1.2. [13].

Ako se poduzeće bavi proizvodnjom, IS za takvu vrstu poslovanja treba sadržavati module koji obrađuju tehničke informacije, automatizaciju proizvodnje, kontrolu proizvoda i slično. Neki od tehničkih modula su CAD (eng. *Computer Aided Design*, računalno potpomognuto oblikovanje), CAP (eng. *Computer Aided Planning*, računalno potpomognuto planiranje) i CAM (eng. *Computer Aided Manufacturing*, računalno potpomognuta proizvodnja), koji će također biti dobro integrirani sa poslovnim sustavima koristeći se istim logičkim bazama podataka.



Slika 1.2. Osnovni sustavi u tipičnom poduzeću

Poduzeća koriste i više sustava nego je naznačeno na slici 1.2. Postoji veliki broj namjenskih sustava za razna specifična poslovna područja. Ipak, ovi sustavi koji su na slici 1.2. mogu se smatrati jezgrom svakog IS-a, tako da najčešće razvoj IS-a u poduzeću započinje razvojem ovih osnovnih modula. Razvoj i dodavanje specifičnih elemenata i njihovu integraciju u cjelokupni IS treba napraviti kvalitetno, tako da se osigura integracija podataka i funkcionalna povezanost svih dijelova IS-a.

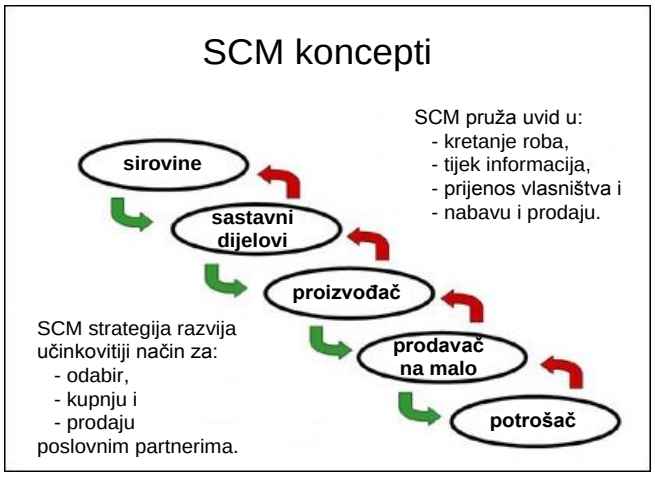
Slijedi detaljniji opis navedenih sustava:

- **Sustav za upravljanje resursima** (ERP) je u većini poduzeća temeljni sustav. To je sveobuhvatni sustav koji pokriva sve glavne poslovne funkcije i procese te prikuplja, obrađuje i daje informacije o svim dijelovima poduzeća. Tako u ERP spadaju funkcionalna područja poput financija i računovodstva, proizvodnje, prodaje i servisa i slično (slika 1.3.). Iako je fokus ERP sustava podrška internim poslovnim procesima, poslovne aktivnosti ne završavaju na granicama (u odnosu na okolinu) poduzeća. Poslovanjem izvan granica poduzeća bavi se sustav za upravljanje lancem nabave.

Slika 1.3. Dijelovi ERP sustava[5]

- **Sustav za upravljanje lancem nabave** (SCM) podržava suradnju s opskrbnim lancima te stvaranje mreže dobavljača i kupaca. Učinkovita mreža dobavljača i kupaca je presudna za uspjeh poduzeća i o njoj uvelike ovise performanse poduzeća (slika 1.4.). Stoga SCM uključuje poslovne procese kretanja i skladištenja sirovina, praćenja inventara te gotove proizvode od nastajanja do prodaje. Definicija APICS[6] daje cjelovit opis SCM: „Oblikovanje, planiranje, provedba, nadzor i praćenje aktivnosti lanca nabave s ciljem stvaranja neto vrijednosti, izgradnje konkurentne infrastrukture, utjecaja na učinkovitost logistike i usklađenje ponude sa potražnjom tržišta“.

[5] http://www.ezdia.com/epad/erp-implementation-system-software/673/ (5.1.2012.)
[6] http://www.apics.org/resources/APICSDictionary.htm (9.1.2012.)



Slika 1.4. Dijelovi SCM sustava[7]

- **Sustav za upravljanje odnosa s kupcima** (CRM) je sustav u kome je integriran pristup prepoznavanja, stjecanja i zadržavanja kupaca. Mnogi smatraju da je dobar odnos s kupcima najveća vrijednost poduzeća. Danas rukovodstvo (eng. *management*) i marketing CRM-u pridaje veliku važnost (slika 1.5.). Potreba za CRM sustavima nastaje kada se uz tradicionalne kanale odnosa s kupcima pojavljuju i web trgovine (e-trgovine), e-mail narudžbe, mobilno trgovanje, kontakt centri (eng. *call center*) te takozvani *push servisi*[8].

Slika 1.5. Primjer web CRM sustava[9]

[7] http://www.youthmagz.com/artikel-56-Supply-Chain-Management-Software-Solutions.html (15.12.2011.)
[8] *Push tehnologija* opisuje način komunikacije putem interneta gdje zahtjev za neku transakciju pokreće izdavač ili centralni servis (za razliku od *Pull tehnologije* gdje je zahtjev pokrenut od strane primatelja ili klijenta).
[9] http://webbasedcrmsoftware.net/ (5.1.2012.)

1.3.4. Razina i odgovornost poslovnih zadataka

Kako u poduzeću postoje različiti tipovi aktivnosti s obzirom na organizacijsku strukturu, to se može govoriti i o različitim tipovima IS-a. Sustavi poslovanja najčešće se sastoje od:

- transakcijskog procesnog sustava,
- upravljačkog informacijskog sustava,
- izvršnog informacijskog sustava,
- sustava za potporu odlučivanju,
- sustava za potporu komunikacijama,
- sustava za potporu uredskom poslovanju.

Transakcijski procesni sustav prihvaća i zapisuje informacije o osnovnim transakcijama kao što su, primjerice, događaji povezani s prodajom, naručivanjem od dobavljača, naplatom zaostalih dugovanja i slično. Ovi sustavi su među prvima informatizirani i automatizirani. Noviji transakcijski sustavi koriste najnovija dostignuća IT-a. Takvi sustavi predstavljaju konkurentsku prednost za poduzeće i osiguravaju povrat uloženih sredstava u vlastitu informatizaciju. Danas su modeli *B2C* i *B2B*[10] elektroničkog poslovanja moderni predstavnici transakcijskih sustava.

Upravljački informacijski sustav preuzima podatke iz transakcijskog sustava i generira izvješća koja su potrebna rukovodstvu za planiranje i kontrolu poslovanja. Ovaj sustav je izgledan i koristan jer su informacije prikupljene u transakcijskom sustavu ujedno i pohranjene u transakcijskoj bazi podataka.

Izvršni informacijski sustav omogućava dostupnost informacijama važnim za donošenje poslovnih odluka nadzora i upravljanja poslovanjem. Neke informacije dolaze iz transakcijske baze podataka, a neke iz vanjskih izvora poput novosti o konkurenciji, izvješća s burzi, ekonomskih pokazatelja i slično.

Sustav za potporu odlučivanju omogućava korisniku da ispita utjecaj mogućnosti i odluka koje nudi sami sustav. Često se u ovakve sustave ugrađuje takozvana *što ako* (eng. *what if*) analiza koja pruža korisniku varijante za izbor prilikom donošenja poslovnih odluka. Primjerice, korisnik želi od sustava dobiti odgovor na pitanje: „Što ako prodaja kroz treći kvartal padne ispod 100 mil., a kamate narastu na 6,5%?“. Tako korisnik može istražiti financijske projekcije koje nudi ovaj sustav.

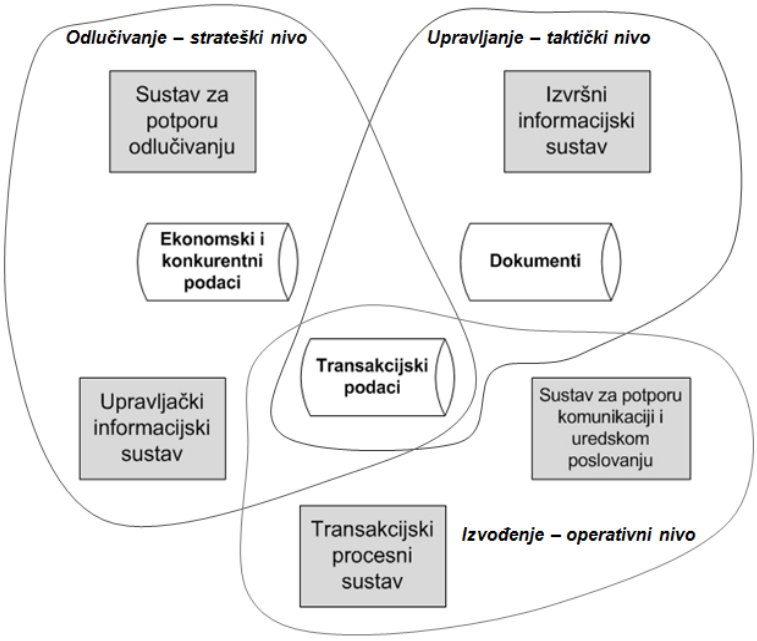
Sustav za potporu komunikaciji omogućava korisnicima međusobnu komunikaciju, a također i komunikaciju s vanjskim partnerima. Danas su to sustavi koji pružaju e-mail usluge, fax, pristup internetu, video konferencije i slično.

Sustav za potporu uredskom poslovanju pomaže korisnicima u izradi zajedničkih dokumenata kao što su izvješća, ponude, podsjetnici i slično. Ovaj sustav omogućava podršku informacijama o radnim zadacima i raspodjeli posla, te radnim sastancima i drugim planiranim obavezama.

[10] B2C (eng. *Business to Consumer*) – poslovanje poduzeća s krajnjim korisnikom, B2B (eng. *Business to Business*) – razmjena roba i usluga među poduzećima.

Na slici 1.6. prikazani su navedeni sustavi zajedno s jedinstvenom logičkom bazom podataka koja je na slici prikazana modelom koji ističe tri dijela:

- Transakcijske podatke – to su osnovni podaci koji se unose u sustav a unose ih *operativci*.
- Dokumenti – to su podaci i informacije koji su izvedeni iz osnovnih podataka a služe izvršnom sustavu i pomažu taktičkoj razini organizacije da upravlja poslovnim procesima.
- Ekonomski i konkurentni podaci koji služe sustavu za potporu odlučivanju i pomažu razinama odlučivanja da donose strateške odluke važne za razvoj poduzeća.



Slika 1.6. Tipovi IS-a obzirom na razinu poslovnih zadataka

S obzirom na vrste sustava prema razinama upravljanja i nadležnosti [11], ova podjela se može malo poopćiti, te se na slici 1.6. vidi da postoji operativni, taktički i strateški nivo s odgovarajućim dijelovima IS-a.

Pitanja za ponavljanje

1. *Objasnite što je informacijski sustav i koje su njegove temeljne aktivnosti.*
2. *Objasnite što je programsko inženjerstvo. Koja je razlika između rutinske i kreativne vještine?*
3. *Koje elemente obuhvaća programsko inženjerstvo?*
4. *Kako informatičar specificira proizvod programskog inženjerstva, a kako korisnik?*
5. *Usporedite prepoznavanje i rješavanje složenih problema nasuprot jednostavnim elementima od kojih se oni sastoje. Što je to gradbeni blok?*
6. *Nabrojite i opišite korake analitičkog pristupa rješavanju problema.*
7. *Objasnite funkcionalni pogled na poslovni sustav (također i strukturni).*
8. *Nabrojite neke od često uočenih problema nakon završenog posla na razvoju IS-a.*
9. *Objasnite ograničenje arhitekture hardvera u ciklusu razvoja IS-a.*
10. *Objasnite ograničenja kod integracije i testiranja sustava. Koje uvjete treba ispuniti da bi se osigurala njihova kvaliteta?*
11. *Nabrojite osnovne kategorije računalnih programa. Objasnite što su to aplikativni programi (ili neke druge od 7 opisanih kategorija).*
12. *Što je otvoreni kod?*
13. *Opišite karakteristike samostalnog razvoja IS-a.*
14. *O čemu poduzeće mora voditi računa ako kupuje gotovo programsko rješenje?*
15. *Što poduzeću pruža razvoj IS-a putem takozvane eksternalizacije? Koji problemi postoje u takvom razvoju?*
16. *Što je sustav za upravljanje resursima i koje su njegove karakteristike?*
17. *Objasnite pojam i namjenu sustava za upravljanje lancem nabave.*
18. *Objasnite pojam i namjenu sustava za upravljanje odnosima s kupcima.*
19. *S obzirom na razinu i odgovornost poslovnih zadataka u poduzeću koje sve informacijske sustave poznajete? Opišite svaki od njih.*

2. Razvoj programske potpore

Pristup razvoju
Sudionici u razvoju
Upravljanje projektom

2.1. Pristup razvoju

Razvijati programsko rješenje znači analizirati potrebe i oblikovati ih u rješenja koja se realiziraju u računalnim programima. Analiza, oblikovanje i realizacija su samo osnovni zadaci jer je razvoj programske potpore samo dio vođenja i upravljanja projektom (ukoliko cijeli zadatak razvoja bude tretiran kao projekt).

Bilo bi dobro da svaki razvoj programske potpore bude sastavni dio projekta kojem se zna početak i kraj, koji se prati kroz zadane faze izrade, za koji su osigurani resursi i koji ima određene mehanizme praćenja i kontrole obavljenog posla. Zadaci razvoja programske potpore su samo jedan dio upravljanja projektom, iako najvažniji.

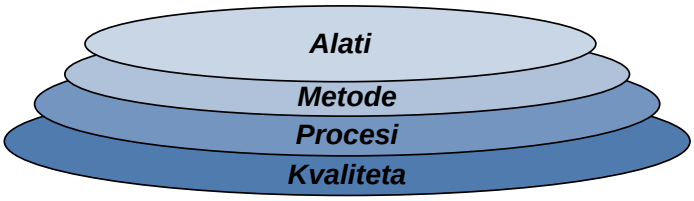
2.1.1. Slojeviti prikaz inženjerskog pristupa

Projektni tim na čelu s voditeljima projekta treba definirati kako će teći razvoj, tko će ga voditi, tko sudjeluje u njemu, što su zadaci članova tima i koji su međurezultati, te ne kraju i konačni rezultat projekta. Stoga bi se trebalo u svakom razvoju koristiti određenom **metodologijom**, pratiti faze i zadatke nekom **metodom** rada te koristiti se dogovorenim informatičkim ili nekim drugim modelima i **tehnikama**.

Metodologija razvoja objedinjuje odabranu metodu, tehnike za prikaz znanja te plan i realizaciju projekta kojemu je zadatak proizvesti informacijski sustav. Metoda i tehnika ima više, a koju će odabrati projektni tim ovisi o iskustvu, načinu na koji informatičko poduzeće izrađuje programe te odluci korisnika s obzirom na ponuđene opcije.

Osnovni zadatak metodologije projektiranja informacijskog sustava je izraditi rješenje koje će taj posao učiniti što je moguće više **formaliziranim** te tako smanjiti potrebu za *genijalnošću* pojedinaca.

Programsko inženjerstvo je proces koji se zasniva na slojevitoj tehnologiji. Prema slici 2.1. [17] inženjerski pristup se zasniva na osiguranju **kvalitete** kao osnovne pretpostavke. Postupci koji potiču upravljanje kvalitetom i promiču kulturu provođenja i poboljšanja kvalitete omogućavaju efikasnu primjenu programskog inženjerstva.



Slika 2.1. Slojeviti prikaz inženjerskog pristupa

Nadalje, temelj programskog inženjerstva je **procesni** pristup koji spaja tehnološke mogućnosti i omogućava racionalni i pravovremeni razvoj računalnih programa. Procesni pristup definira okvire (eng. *framework*) koje treba postaviti kako bi razvoj tekao po dogovorenom scenariju i kako bi se provodile kontrola i dogovorena metoda. To uključuje praćenje proizvoda (modeli, dokumenti, podaci, izvješća, forme i slično), rokova i kontrolnih točaka (eng. *milestones*). Također, procesnim pristupom osigurava se kvaliteta i mogućnost prilagodbi za vrijeme trajanja projekta.

Metode obuhvaćaju široku paletu zadataka koji uključuju komunikaciju, analizu zahtjeva, oblikovanje modela, izradu programa i njihovo testiranje te na kraju održavanje. Metode programskog inženjerstva zasnivaju se na osnovnim principima koji pokrivaju svako područje tehnologije i načina njenog opisivanja.

Alati programskog inženjerstva omogućavaju automatsku i poluautomatsku podršku procesima i metodama. Alati su osmišljeni tako da se mogu integrirati i razmjenjivati informacije koje stvaraju. Sustavi koji osiguravaju podršku razvoju IS-a nazivaju se CASE alati (eng. *Computer Aided Software Engineering*).

2.1.2. Modeli, metode, tehnike

Metodologija o projektiranju i izgradnji informacijskog sustava je znanost o njegovu razvoju. Metodologija razvoja nekog sustava pruža smjernice za organizaciju, praćenje i završetak svake aktivnosti životnog ciklusa razvoja sustava, uključujući i odabrane metode i tehnike.

Metoda je definirani postupak djelovanja za postizanje određenog cilja na nekom praktičnom ili teorijskom području. To je način istraživanja ili praktičnog postupanja i djelovanja kako bi se došlo do nekog rezultata.

Tehnika (sredstvo) se odnosi na jezik, alat i slično, čime se izvršava projektni zadatak, odnosno formalizira neka operacija. Ovdje se sredstvima nazivaju prvenstveno grafički jezici pomoću kojih se opisuju odgovarajući pogledi na promatrani sustav (primjerice DTP – dijagram tijeka podataka, MPT – matrica poslovne tehnologije, RADD – radni dijagram, E-R model podataka; UML-ov dijagram slučajeva korištenja, UML-ov dijagram klasa, ...)

Model je pojednostavljena slika stvarnosti u kojoj se ističu najvažnija svojstva te stvarnosti. Svaki put kada ljudi žele zabilježiti ili razmijeniti informacije o nečemu iz

realnog svijeta korisno je izraditi model. Takva je uloga i modela koji se izrađuju u razvoju IS-a.

Neki modeli su fizički vrlo slični gotovim proizvodima (primjerice model aviona). Neki modeli su grafičke reprezentacije nekih važnih detalja. Neki modeli su apstrakcije pomoću matematičkih simbola. Dakle, model može sadržavati različite tipove informacija.

Modeli kojima se prikazuju elementi IS-a nisu u potpunosti standardizirani ili precizni, kao primjerice model aviona. Međutim, ti modeli pomažu u razumijevanju i napredovanju u procesu razvoja IS-a.

Ovdje se promatraju dvije grupe modela:

- U prvoj su grupi modeli razvoja IS-a. Modeli razvoja opisuju kako teče proces razvoja IS-a od snimke stanja preko analize i oblikovanja do realizacije i implementacije. Modela razvoja koji će biti detaljnije opisani u poglavlju 3. su: vodopadni, iterativni, evolucijski i prilagodljivi te neke njihove kombinacije.
- U drugoj su grupi modeli pomoću kojih se opisuju poslovni sustav i/ili informacijski sustav. Tako su neki od važnijih modela sustava:
    - **Model procesa** prikazuje skup procesa koji mijenjaju stanje sustava i skupa procesa pomoću kojih se formiraju izlazi iz sustava. Model procesa je skup poslova nad skupovima podataka. Procesi na modelu jesu skupovi poslova koji stvaraju ili koriste informacije za svoje funkcioniranje.
    - **Model podataka** prikazuje stanje sustava preko skupa podataka. Podaci su u sustavu i u modelu u njihovom prirodnom odnosu na temelju kojih će proizići organizacija baze podataka na računalu.
    - **Model resursa** specificira tehnološku osnovicu. On prikazuje kadrove, organizacijske jedinice, opremu i slično, koji omogućuju smještanje i dinamiku podataka i procesa sustava. U modelu resursa su skriveni svi aspekti različiti od podataka i procesa.

Konačnu odluku o izboru metodologije razvoja informacijskog sustava uvijek donosi poslovodstvo tvrtke koje mora osigurati financijske, organizacijske, tehničke i kadrovske preduvjete za realizaciju.

PRIMJER: Neki od poznatih modela koji se koriste u razvoju IS-a su:

- Radni dijagram (eng. *Flowchart*)
- Dijagram toka podataka (eng. *Data flow diagram*, DFD)
- Dijagram Entiteti-Veze (eng. *Entity-relationship diagram*, ERD)
- Strukturni grafikon (eng. *Structure chart*)
- Dijagram slučajeva korištenja (eng. *Use case diagram*)
- Dijagram klasa (eng. Class diagram)
- Dijagram slijeda (eng. *Sequence dijagram*).

Neki od modela koji se koriste u upravljanju i vođenju projekta razvoja IS-a su:

- PERT dijagram
- Gantt dijagram

- Grafikon organizacijske hijerarhije (eng. *Organizational hierarchy diagram*)
- Modeli analize financija, primjerice ROI (eng. Return on Investment).

Neki alati koji se koriste u razvoju IS-a:

- Aplikacija za upravljanje projektom
- Aplikacija za crtanje/ grafiku
- Tekst procesor/ editor
- CASE alati
- Integrirana razvojna okolina (IDE)
- Aplikacije za upravljanje bazama podataka (DBMS)
- Alati za reverzno inženjerstvo
- Alati za generiranje koda.

2.1.3. Osnovne metodologije razvoja

Metodologije razvoja programske potpore mogu se podijeliti u četiri grupe [16], kako je kroz povijest nastajala potreba za razvojem programa za poslovanja poduzeća, a paralelno sa razvojem (informacijske) tehnologije:

- Strukturirana analiza i oblikovanje (dizajn), (eng. *Structured analysis and design*, SAD)
- Objektno-orijentirana analiza i oblikovanje (eng. *Object-oriented analysis and design*, OOAD)
- Brzi razvoj programa (eng. *Agile software development*, ASD)
- Aspektno orijentirani razvoj programa (eng. *Aspect-oriented software development*, AOSD).

Strukturna analiza i oblikovanje pojavljuju se 70-tih godina prošlog stoljeća a glavne karakteristike su modeliranje sustava alatima i tehnikama koje podržavaju funkcionalnu dekompoziciju i analizu toka podataka. Još se naziva i tradicionalni pristup razvoju.

Objektno-orijentirana analiza i oblikovanje pojavljuje se 80-tih godina prošlog stoljeća, a široko je usvojen sredinom 90-tih. Uvodi opisivanje poslovnih i sistemskih funkcionalnosti pomoću slučajeva korištenja (eng. *use case*). UML jezik (eng. *Unified Modeling Language*) se koristi kao osnovni alat za modeliranje raznih aspekata sustava.

Ideja o brzom razvoju programa pojavljuje se krajem 90-tih godina prošlog stoljeća i brzo stječe popularnost u programskoj industriji kao lagan (eng. *lightweight*) način za razvoj programa.

Aspektno orijentirani razvoj programa uvodi se u kasnim 90-tim godinama prošlog stoljeća i nije zamjena za bilo koju od drugih metodologija. Funkcionalna obilježja sustava mogu se podijeliti na (1) funkcionalnosti koje omogućuju krajnjem korisniku da postigne zadane poslovne ciljeve, i (2) funkcionalnosti podrške koje osiguravaju značajke poput prava pristupa, povezanosti, podudarnosti, sučelja i slično. Moderne tehnologije razvoja programske potpore imaju isprepletene (do određene mjere) ove

dvije vrste funkcionalnosti. Aspektno orijentirani razvoj programa pomaže u savladavanju tih problema na sustavan način.

U 3. poglavlju detaljno su opisana prva tri pristupa razvoju IS-a kroz opće modele tradicionalnog razvoja, preko specifičnosti unificiranog procesa do brzog razvoja programske potpore koja je sve više danas zastupljena u informatičkoj industriji.

2.2. Sudionici u razvoju

Sudionici (eng. *stakeholders*) su sve zainteresirane strane, odnosno svi koji sudjeluju i doprinose uspješnom razvoju i izradi IS-a. Općenito se sudionici mogu podijeliti na:

- korisnike koji će se svakodnevno koristiti sustavom;
- klijente koji plaćaju izradu/ nabavu IS-a;
- tehničko osoblje koje će osigurati da sustav radi u okviru zadane konfiguracije računala i odabranih sistemskih programa;
- vanjske entitete, primjerice kupci.

2.2.1. Vještine projektanta sustava

Programsko inženjerstvo je karakteristično upravo zbog vladanja različitim vještinama, naročito za analitičare poslovnog sustava i projektante (dizajnere) novog IS-a. Potrebne vještine i znanje mogli bi se podijeliti u tri grupe: tehničko znanje, poslovno znanje i znanje o ljudima (nešto između vještine komuniciranja i poznavanja psihologije).

Tehnička znanja i vještine su važni za analitičara jer je on uključen u zadatke izrade programa. Stoga je važno da razumije različite tipove tehnologija, čemu služe, kako funkcioniraju i kako su nastale. Nije potrebno da analitičar bude tehnološki ekspert, ali svakako treba razumjeti:

- Kako rade računala.
- Uređaje koji su povezani s računalom (ulazne/izlazne jedinice i memorijske uređaje).
- Komunikacijsku mrežu na koju su spojena računala.
- Baze podataka i sustave za upravljanje bazama podataka.
- Programske jezike.
- Operacijske sustave i uslužne programe.

Osim toga, analitičar sustava bi trebao znati mnogo toga o alatima i tehnikama koje se koriste za razvoj IS-a. Od alata bi trebao poznavati programske pakete poput MS

Access i *PowerBuilder*, *VisualStudio.Net*, *CASE* pomagala, generatore programskog koda, alate za testiranje, biblioteke za pohranu datoteka, dokumenata i slično.

Poslovna znanja i vještine su važni za analitičara kako bi bio u stanju razumjeti poslovanje poduzeća na općoj razini, jer su problemi koje rješava upravo problemi poslovanja. Stoga bi analitičar trebao znati:

- Koje poslovne funkcije se obavljaju u poduzeću?
- Kako je poduzeće strukturirano?
- Kako se upravlja poduzećem?
- Koje vrste poslovanja su zastupljene, primjerice, financijski poslovi, proizvodnja, marketing, klijentski servisi, i slično.

Bilo bi dobro da se za vrijeme školovanja informatičari susreću i sa znanjima povezanim s administrativnim poslovima poput računovodstva, marketinga ili upravljanja. Često su analitičari zaposleni u nekim specifičnim granama industrije ili poslovanja, te su njihova znanja od velike koristi u rješavanju složenih problema za takvu vrstu poslovanja. Ponekad, ako se pokaže potrebnim, projektni tim može angažirati vanjskog suradnika koji će igrati ulogu analitičara jer je njegovo znanje dragocjeno za kvalitetu programskog rješenja.

Što više analitičar zna o radu nekog poduzeća to će biti efikasniji, te bi bilo dobro da poznaje:

- Koje su specifičnosti promatranog poduzeća?
- Što poduzeće čini uspješnim?
- Koju strategiju i plan razvoja ima poduzeće?
- Koje tradicije i vrijednosti *njeguje* poduzeće?

Vještina poznavanja ljudi mora biti karakteristika za analitičara budući da radi u timu s više zaposlenika iz svoje organizacije, ali i iz drugih organizacija. Analitičar će provesti mnogo vremena u radu s ljudima i nastojati razumjeti njihovo viđenje problema koji rješava. Stoga je važno da analitičar razumije kako ljudi:

- Razmišljaju
- Uče
- Reagiraju na promjene,
- Komuniciraju
- Rade (varijacije posla i odgovornosti).

Analitičar mora razumjeti kako ljudi razmišljaju kako bi što bolje shvatio što ljudi žele od programa u smislu interakcije s računalom. Važno je saznanje, kod oblikovanja novog sustava i obuke korisnika, uočiti način na koji ljudi uče. Kada novi sustav bude implementiran, rad ljudi će se promijeniti i na takve nove situacije oni trebaju biti pripremljeni. Analitičaru stoga treba umijeće komuniciranja kako bi od korisnika

dobio što više informacija motivirajući ga za suradnju. Kako je u situaciji da razgovara s više ljudi u nekom okruženju, analitičar ima jedinstvenu priliku utjecati na cijelu organizaciju.

2.2.2. Naručitelj i korisnik

Koja je razlika između naručitelja (klijenta, (eng. *customer*) i krajnjeg korisnika (eng. *end-user*)?

Programski inženjeri komuniciraju s mnogo različitih interesnih grupa među kojima naručitelji i krajnji korisnici imaju značajan utjecaj na tehničku izvedbu programskog rješenja. U nekim slučajevima ove dvije grupe objedinjene su u jednu, no u mnogim projektima naručitelj i krajnji korisnik različiti su ljudi koji rade u različitim poslovnim organizacijama i imaju različito ulogu u rukovodstvu i hijerarhiji poduzeća.

Naručitelj je predstavnik onih koji:

- postavljaju zahtjeve za izgradnjom IS-a
- definiraju cjelokupan opseg poslovanja koji treba informatizirati
- određuju osnovne zahtjeve koje proizvod mora ispuniti
- koordiniraju financiranje projekta.

Korisnik je predstavnik marketinškog ili upravljačkog sektora poduzeća.

Krajnji korisnik je predstavnik grupe koja će:

- stvarno se koristiti programima koji su izrađeni u svrhu rada i postizanja nekih poslovnih ciljeva,
- definirati operativne detalje programske podrške tako da se postignu poslovni ciljevi.

Postoji još različitih varijacija sudionika koje je važno prepoznati u fazi analize poslovnog sustava i određivanja zahtjeva. Treba uzeti u obzir i blisko surađivati s predstavnicima svih kategorija zainteresiranih za novi sustav jer će jedino tako IS pružiti cjelovitu funkcionalnost i ispuniti sve zahtjeve. Informatičari, odnosno predstavnici izvođača IS-a, trebaju biti sigurni da će angažirani sudionici surađivati tijekom razvoja, te da su osobe koje su određene da sudjeluju ujedno i stručnjaci u svom području.

Primjer koji slijedi pokazuje nekoliko karakterističnih problema koji su poznati informatičarima, a odnose se na korisnike kao osnovni izvor informacija i znanja o poslovnom sustavu.

***PRIMJER*:** **Izazovi definiranja korisničkih zahtjeva**

Još dvije, može se reći, anegdote koje slikovito prikazuju razmišljanje i ponašanje korisnika na početku razvoja IS-a (ili jednog dijela programske podrške).

Oba primjera su smještena na početak razvoja, u analizu poslovanja i definiranje korisničkih zahtjeva.

a) U prvom se primjeru [12] može postaviti pitanje: "Kako otkriti što korisnik treba?".

   Odgovor bi, na prvi pogled, mogao biti: "Ako želiš znati što korisniku treba, idi i pitaj ga!?"

   Međutim, ako je riječ o korisniku IS-a, ovaj savjet *ne igra*. Korisnik ne zna što hoće jer je pitanje "Što Vi hoćete u Vašem novom IS-u?" presloženo i ne može se od njega očekivati odgovor na ovo pitanje.

   Pitanje bi prije trebalo formulirati ovakvo:

   Jednom kada se počnete koristiti novim IS-om
   i
   Vaš posao doživi izmjene
   i
   poslovanje Vašeg poduzeća se promijeni
   i
   naučite se koristiti aplikacijama na računalu
   kako želite da radi Vaš novi IS?

b) Drugi primjer pokazuje (općenito) znanje o ICT-u[11] i pristup korisnika s kojima treba surađivati i ostvariti kvalitetan programski proizvod.

   Steve McConnell, u knjizi Brzi razvoj (*Rapid Development* 1996.) daje razne načine na koje korisnik pokušava zaobići proces definiranja korisničkih zahtjeva:

   1. Korisnici ne razumiju što žele.
   2. Korisnici ne žele zaključiti listu zahtjeva.
   3. Korisnici inzistiraju na novim zahtjevima nakon što se dogovore troškovi i plan rada.
   4. Komunikacija sa korisnicima je spora.
   5. Korisnici često ne sudjeluju u redefiniranju zahtjeva.
   6. Korisnici su tehnički neobrazovani.
   7. Korisnici ne razumiju proces razvoja programa (IS-a).
   8. Naravno, ova lista ne završava s brojem 7.

Ova dva primjera pokazuju probleme s kojima se susreću informatičari u nastojanju da savladaju probleme i znanje s područja za koje treba izgraditi programski proizvod. Također, ova dva problema naglašavaju potrebu da se od strane korisnika odaberu stručne i pouzdane osobe koje će svojim sudjelovanjem u timovima zaduženima za razvoj IS-a doprinijeti njegovoj kvaliteti i uspjehu.

2.2.3. Izvođači

Ljudi koji se bave razvojem programskih rješenja rade na poslovima prepoznavanja i rješavanja problema te implementaciji sustava. Tako poslove analize i oblikovanja radi

[11] Informacijsko-komunikacijska tehnologija

nekoliko različitih uloga u svijetu informatičara. Mogu se ti poslovi zajedno s programiranjem dodijeliti jednoj osobi, ukoliko ona ima dosta iskustva i znanja.

U sljedećem primjeru pokazani su neki od poslova koje obavljaju ljudi zaposleni u poduzećima koja se bave izradom i/ili distribucijom programskih proizvoda [21]:

- Analitičar programer (eng. *Programmer analyst*)
- Analitičar poslovnog sustava (eng. *Business systems analyst*)
- Sistem analitičar (eng. *System liaison*)
- Analitičar krajnji korisnik (eng. *End-user analyst*)
- Poslovni konzultant (eng. *Business consultant*)
- Sistemski konzultant (eng. *Systems consultant*)
- Analitičar održavanja sustava (eng. *System support analyst*)
- Dizajner sustava (eng. *Systems designer*)
- Inženjer programa (eng. *Software engineer*)
- Arhitekt sustava (eng. *System architect*)
- Webmaster
- Web programer (eng. *Web developer*).

Potrebno je imati i osobu koja će u vremenu razvoja projekta imati ulogu voditelja projekta (eng. *Project manager*). U informatičkom se svijetu zasigurno mogu čuti razni nazivi slični navedenima ili kao kombinacije navedenih.

2.3. Upravljanje projektom

Razvoj programske potpore je zahtjevan zadatak za sve sudionike, a praksa je pokazala da je rezultat prilično neizvjestan. Unatoč nastojanju da se razvoj odvija po unaprijed zadanim pravilima ili barem po dogovoru, on u informatici često izmakne kontroli.

Ako projekt nije ispunio očekivane ciljeve, **razlozi neuspjeha** mogu biti u:

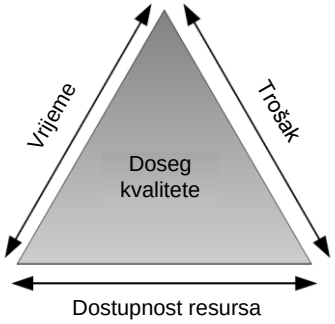
- slabom planiranju projekta i nejasnim ciljevima
- ograničenom sudjelovanju korisnika
- manjku ili nedovoljnoj stručnosti informatičara
- manjku resursa
- nepotpunim i nejasnim zahtjevima za novi sustav.

S druge se strane kao zaslužni za **uspjeh** nekog projekta navode:

- cjelokupan i detaljan plan projekta
- jasno definirani zahtjevi

- stalno i stvarno sudjelovanje korisnika
- realan plan rada uz podršku poslovodstva.

Projekt je (općenito) aktivnost koja se odvija u vremenski zadanom periodu, a cilj je proizvesti jedinstveni proizvod, uslugu ili rezultat [18]. Osim rokova (slika 2.2.) za projekt je važan tim ljudi koji će raditi na projektu. Kada projekt završi tim se raspušta i tako se ljudski resursi oslobađaju za druge zadatke. Rad na projektu zahtijeva i troškove koji mogu biti troškovi plaća članova tima, tehnike i drugih resursa potrebnih da se projekt uspješno razvija. Projekt treba imati i uračunanu cijenu proizvoda koji se razvija, prije svega zbog klijenta za kojega se proizvod razvija.



Slika 2.2. Osnovni parametri unutar projekta

Životni ciklus razvoja informacijskog sustava je samo jedan dio aktivnosti koje uključuje upravljanje projektom (eng. *Project Management*). Cjelokupni zadaci upravljanja projektom uključuju još i planiranje i organizaciju projekta, rješavanje problema, koordinaciju, kontrolu i raspodjelu posla, vremensko planiranje i slično.

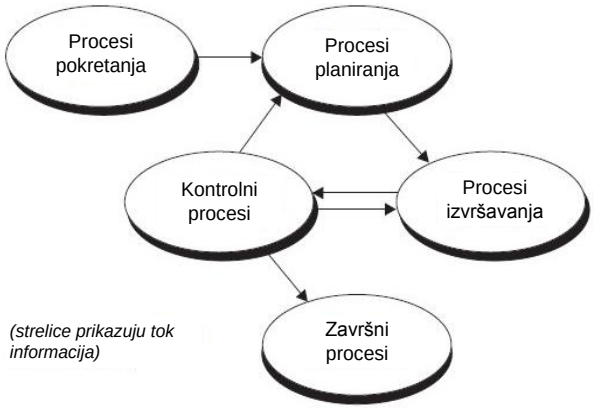
Upravljanje projektom je primjena znanja, vještina, alata i tehnika u projektnim aktivnostima da se ispune projektni zahtjevi. Za upravljanje projektima razvoja informacijskog sustava razvijeno je više različitih modela. Ovdje su ukratko prikazani neki elementi modela upravljanja projektom koji je definirala međunarodna udruga za upravljanje projektima PMI[12] (eng. *Project Management Institute*).

2.3.1. Procesi upravljanja projektom

Naglasak je na zadacima, tj. procesima, projekta koji prate razvoj IS-a. To su ipak ključni zadaci koji će na kraju proizvesti programe i baze podataka, te cjeloviti IS, nakon što budu integrirani i implementirani kod korisnika. Ipak, kako bi projekt započeo, tekao i bio završen treba ga pokrenuti (inicirati) i planirati, u skladu s pokazanim na slici 2.3. Nije to samo planiranje tijeka zadataka u vremenu, nego i

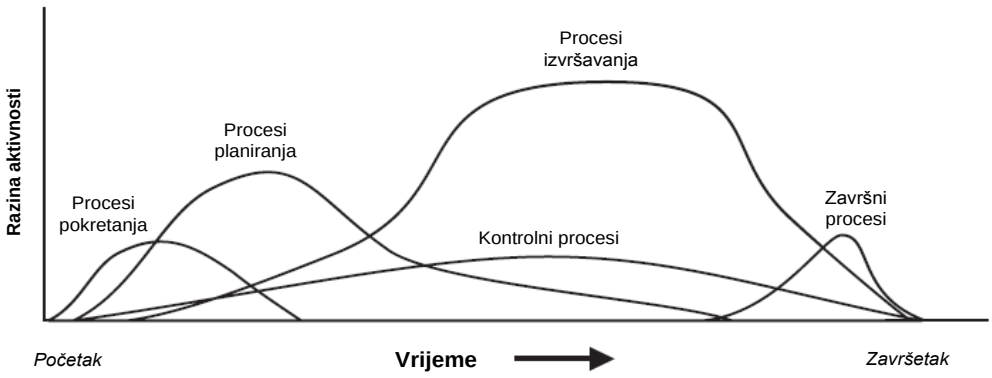
[12] PMI je međunarodna udruga za upravljanje projektima sa sjedištem u SAD-u i s ograncima po cijelom svijetu, pa tako i u Hrvatskoj. Zbog sjedišta u SAD-u, njezin je standard prihvaćen kao ANSI standard.

usklađivanje raspoloživih resursa i reagiranje na probleme, kojih ima uvijek i svugdje. Stoga se tijekom razvoja IS-a neprekidno kontrolira izvršavanja zadataka, i po potrebi korigira planiranje s obzirom na novonastale spoznaje u tijeku projekta. U skladu s izvršenim zadacima i zadovoljavajućim rezultatom kontrole, u trenutku kada je posao završen, projekt se zatvara.



Slika 2.3. Povezanost osnovnih grupa procesa projekta

Tijek procesa upravljanja projektom se preklapa i ima različite intenzitete tijekom projektiranja. Na slici 2.4. vidi se kako se spomenute grupe procesa preklapaju i variraju tijekom vremena. Razumljivo je da procesi pokretanja i planiranja imaju veći intenzitet na početku projekta, a završni procesi prethode zaključenju projekta kada je proizvod gotov i isporučen klijentu. Uočava se da procesi izvršavanja i kontrole usporedno imaju manje ili veće intenzitete, što je i razumljivo, jer se međusobno isprepliću. Kontrola se najviše i obavlja za vrijeme izvršavanja zadataka kada se provjerava i popravlja ono što ne teče prema planu ili ako se pojave novi neevidentirani problemi.



Slika 2.4. Preklapanje i intenzitet procesa u projektu

2.3.2. Područja upravljanja projektom

Procesi upravljanja projektom, opisani u prethodnom poglavlju, odvijaju se kroz nekoliko područja. Naime, upravljanje projektom zadire u mnoga područja znanja jer se razne komponente izrade proizvoda moraju pratiti kako bi se došlo do konačnog zadanog cilja. Tako treba pratiti vrijeme, troškove, kvalitetu, rizik, te ljude i komunikaciju. U tablici 2.1. [18] pokazana je veza između procesa i područja upravljanja projektom.

Tablica 2.1. Pregled procesa i područja aktivnosti upravljanja projektom

Grupe procesa / Područje znanja	Pokretanje	Planiranje	Izvršenje	Kontrola	Završetak
Upravljanje dosegom projekta	Pokretanje	Planiranje dosega Definiranje dosega		Provjera dosega Kontrola izmjena dosega	
Upravljanje vremenskim rasporedom projekta		Definiranje aktivnosti Određivanje redosljeda aktivnosti Procjena trajanja akt. Izrada vrem. rasporeda		Nadzor ispunjenja rokova	
Upravljanje troškovima projekta		Procjena troškova Upravljanje proračunom		Nadzor nad troškovima	
Upravljanje kvalitetom projekta		Planiranje kvalitete	Osiguranje kvalitete	Nadzor nad kvalitetom	
Upravljanje ljudskim resursima projekta		Planiranje ljudskih res. Prikupljanje projektnog tima	Razvoj projektnog tima		
Upravljanje razmjenom informacija u projektu		Planiranje komunikacije	Distribuiranje informacija	Izvješćivanje o provedbi	Zatvaranje poslova
Upravljanje rizicima projekta		Planiranje upravljanja rizicima Prepoznavanje rizika Kvalitativna analiza rizika Plan ublažavanja rizika		Praćenje i nadzor rizika	

U tablici 2.1. pokazan je jedan dio područja koje obrađuje PMI metodologija upravljanja projektom. Kratki opisi aktivnosti ovdje navedenih područja su:

- Upravljanje dosegom projekta treba osigurati da u projekt budu uključeni svi poslovi nužni za uspješno dovršenje projekta (ali ne više nego što je potrebno). Ovo područje treba posebnu pažnju posvetiti razlučivanju što jest, a što nije uključeno u projekt.
- Upravljanje vremenskim rasporedom projekta je područje odgovorno za planiranje i pravodobno zaključenje svake od aktivnosti i cjelokupnog projekta. Kako se projekt sastoji od velikog broja aktivnosti s različitim vezama i međusobnim ovisnostima, ovo područje vodi računa o ispunjavanju zadataka u okviru vremenskih obaveza i kontrolnih točaka.

- Upravljanje troškovima projekta je također važno područje jer se ukupni troškovi projekta ne znaju unaprijed, nego se uvijek rade procjene troškova. Stoga troškove treba razbiti na dijelove koji prate aktivnosti i takve pakete onda kontrolirati tijekom razvoja. Ako se otkriju odstupanja od proračuna, treba pokrenuti korektivne mjere i nastojati ostati u okvirima procijenjenih vrijednosti.
- Upravljanje kvalitetom projekta obuhvaća sve aktivnosti koje određuju kvalitetu, te ciljeve i standarde u postizanju kvalitete. To uključuje aktivnosti planiranja, osiguranja i kontrole, kako kvalitete rezultata projekta tako i kvalitete procesa upravljanja projektom.
- Upravljanje ljudskim resursima projekta je područje koje se bavi identificiranjem uloga i odgovornosti sudionika u timu, poboljšanjem stručnosti i sposobnosti, međusobnim odnosima članova tima, praćenjem rezultata članova tima te rješavanjem problema i usklađenjem promjena u timu.
- Upravljanje razmjenom informacija u projektu bavi se generiranjem, prikupljanjem, distribucijom, pohranom i korištenjem informacija o projektu pravovremeno i na odgovarajući način. Ove aktivnosti su ključne za uspjeh projekta, naročito kada se zna da su procesi i aktivnosti za učinkovitu razmjenu informacija i izvješćivanje uključeni u ovo područje.
- Upravljanje rizicima projekta također igra važnu ulogu u konačnom uspjehu projekta jer su oni izloženi značajnim rizicima. Ovo područje uključuje identifikaciju i analizu rizika, razvoj protumjera, praćenje i kontrolu rizika te vrednovanje mjera kao odgovor na rizike.

Pitanja za ponavljanje

1. *Objasnite procesni pristup kao temelj programskog inženjerstva.*
2. *Definirajte i objasnite pojmove metoda, tehnika, alat, metodologija.*
3. *Što je model? Zašto se koristimo tehniku modeliranja?*
4. *Objasnite pojmove model procesa, model podataka i model resursa.*
5. *Navedite neke od poznatih modela koji se koriste u razvoju IS-a. Skicirajte primjere za svaki od poznatih modela.*
6. *Koje osnovne grupe metodologija razvoja IS-a poznajete? Opišite svaku od njih.*
7. *Objasnite tehničke vještine i znanja koja mora imati analitičar sustava.*
8. *Zašto su važna poslovna znanja i vještine za analitičara i projektanta sustava?*
9. *Objasnite karakteristike vještina poznavanja ljudi koje su važne za razvoj IS-a?*
10. *Objasnite pojmove naručitelj, korisnik i krajnji korisnik.*
11. *Koje su uloge informatičara važne u procesu razvoja IS-a?*
12. *Koji su razlozi neuspjeha nekog projekta razvoja IS-a? Koji su razlozi uspjeha?*
13. *Objasnite dosege kvalitete projekta kao što su trošak, vrijeme i dostupnost resursa.*
14. *Koje grupe procesa upravljanja projektom poznajete? Kako se te grupe preklapaju i kakav je njihov intenzitet kroz vrijeme trajanja projekta razvoja?*
15. *Koja osnovna područja upravljanja projektom poznajete?*
16. *Objasnite upravljanje vremenskim rasporedom projekta (upravljanje troškovima, upravljanje kvalitetom, upravljanje rizicima)?*

3. Modeli razvoja

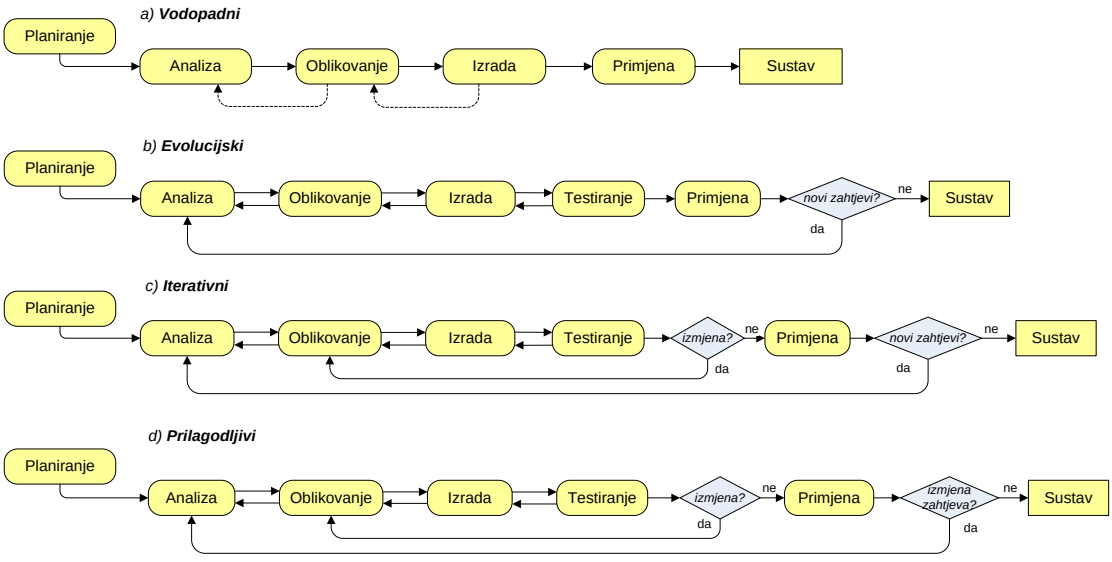
Strukturni modeli
Unificirani proces
Agilno programsko inženjerstvo

3.1. Strukturni modeli

Postoji nekoliko osnovnih (generičkih) modela razvoja IS-a na temelju kojih se, tijekom godina, razvio i veći broj varijanti. Neke se varijante zasnivaju na fazama razvoja IS-a, a neke druge na stupnjevima iteracija koje su dopuštene u pojedinoj metodi. Neke varijante su specifične s obzirom na organizaciju i razvojni tim, te tehnološko okruženje.

Svi modeli bi trebali imati nekoliko osnovnih faza razvoja programske potpore u koje se ubraja analiza poslovnog sustava, oblikovanje buduće programske podrške, izrada programske podrške te provjera ispravnosti (testiranje) i na kraju uvođenje programskog rješenja kod korisnika. Navedena podjela nije jednostavna, koraci se mogu preklapati, mogu se odvijati slijedno ili istovremeno, a rezultati nekog koraka mogu utjecati na jedan ili više drugih koraka.

Slijede ukratko opisani generički modeli razvoja programske potpore: vodopadni, evolucijski, iterativni i prilagodljivi model.



Slika 3.1. Modeli razvoja programske potpore

Vodopadni je model (slijedni, linearni, eng. *waterfall*) primjenjiv kada su korisnički zahtjevi dobro definirani. Na osnovu takvih zahtjeva slijedno se odvijaju modeliranje i izrada programa i na kraju njegova primjena, kako je pokazano na slici 3.1.a). [2]

Događa se da se kod primjene (ili ranije, u modeliranju i izradi) uoče nedostatci koji su posljedica nepotpunih ili krivih korisničkih zahtjeva. U vodopadnom modelu povratak na prethodne faze nije dopušten.

Varijanta takozvanog pseudostrukturnog modela (povratne strelice na slici 3.1.a) omogućava evidentiranje razlika i nedostataka te povratak i doradu programa. Ovaj bi popravak trebao ponovno proći ciklus vodopadnog modela od dopune korisničkih zahtjeva, ugradnje izmjena u modeliranju i izrade programa te ponovne primjene. Presudostrukturni model zahtijeva ograničeno i kontrolirano vraćanje na prethodne faze razvoja. On predstavlja prijelaz prema narednim modelima razvoja.

Evolucijski model (naziva se i inkrementalni) je primjenjiv kod dobro definiranih zahtjeva za jedan ciklus razvoja. To znači da je projekt podijeljen u više manjih cjelina u kojima programer nakon svake primjene traži povratni odgovor od korisnika, kako je pokazano na slici 3.1.b).

Iterativni model (razvojni, prototipiranje) je karakterističan po tome što se u zadanom vremenu planira, izrađuje, testira i primjenjuje manji dio zadane cjeline. Ako je potrebno, a sigurno jeste jer bi inače bio vodopadni model, nakon provjere se dodaju nove funkcionalnosti. Na taj se način vraća na postojeće izrađene dijelove koje treba proširiti (novi ciklus nakon primjene) ili poboljšati (nova iteracija nakon testiranja), kako pokazuje slika 3.1.c).

Prilagodljivi (adaptivni) model se primjenjuje kada se za početnu aktivnost odabranih zahtjeva potrebe stalno mijenjaju te je na kraju jednog ciklusa izrade potrebno prikupiti mišljenje korisnika koje se po potrebi preoblikuje i ponovno izrađuje (slika 3.1.d).

Ovaj model u sebi sadrži prototipiranje koje nastoji da izmjenama napravljenog programa dođe do zadovoljavajućeg rješenja. Uz to sadrži i mogućnost izmjene već postojećeg opsega posla kako bi bilo moguće utjecati na sadržaj analize postojećeg sustava, tj. na korisničke zahtjeve. Prilagodba modela je najviše i izražena u mogućnosti prilagođavanja opsega sustava potrebama korisnika.

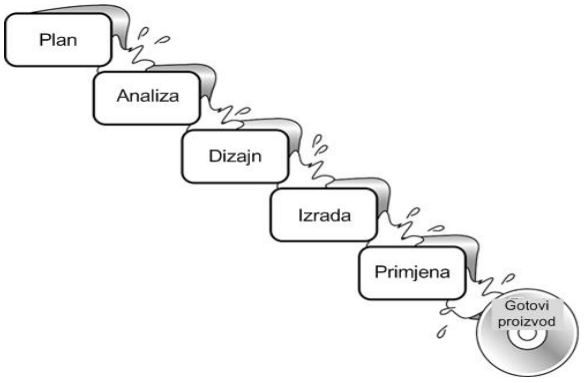
U nastavku će, u nekoliko poglavlja, biti detaljno opisan skup karakterističnih strukturnih modela koji se često primjenjuju u praksi [3]. Za početak je detaljno opisan vodopadni model u kojem su navedeni osnovni principi ovog modela, njegove dobre i loše strane te situacije u kojima je pogodan za korištenje. Nadalje su još od osnovnih modela opisani iterativni i evolucijski te spiralni model i brzi razvoj aplikacija kao složenije varijante i kombinacije osnovnih modela.

3.1.1. Vodopadni model

Ovaj se model pojavio na početku razvoja IS-a i predstavlja prvi model procesa u programskom inženjeru.

Osnovne karakteristike vodopadnog modela (slika 3.2.):

- Proces je podijeljen u faze slijedno, tako da je prihvatljivo preklapanje faza razvoja u manjoj mjeri. Primjerice, može se započeti s izradom programa dok još traje faza oblikovanja.
- Potrebno je odjednom za cijeli posao izraditi plan zadataka, raspored, odrediti rokove i troškove.
- Tijekom cijelog projekta važna je stroga kontrola i praćenje kroz brojnu dokumentaciju. Na kraju svake faze od sudionika u timu traži se mišljenje i suglasnost.



Slika 3.2. Vodopadni model

Dobre su strane vodopadnog modela:

- Podržavanje slijednog tijeka faza i koraka razvoja te stroga kontrola odgovarajuće dokumentacije osiguravaju kvalitetu, pouzdanost i mogućnost održavanja u procesu razvoja.
- Praćenje napretka razvoja sustava je mjerljivo.
- Predvidljiva je zauzetost resursa.
- Model je pogodan za korištenje u projektnom timu s manjkom iskustva ili u timu čiji se ljudi često mijenjaju.

Loše su strane vodopadnog modela:

- Ovaj proces razvoja je spor, skup i neefikasan, a razlog je robusna struktura i stroga kontrola. Model nije prilagođen brzom odgovoru na promjene. Promjene koje se javljaju u kasnijim fazama razvoja su skuplje i demotivirajuće.

- Razvoj ovisi o ranom prepoznavanju i specifikaciji zahtjeva. Međutim, korisnici u tim ranim fazama razvoja često nisu u stanju jasno definirati svoje potrebe.
- Nema prostora za primjenu iteracija pa se može smanjiti efikasnost upravljanja projektom.
- Često se javlja nedosljednost zahtjeva, manjak komponenti sustava i neočekivani zahtjevi u razvoju. Ovi nedostatci se uočavaju tek u fazi oblikovanja i izrade koda, a tada je već kasno.
- Problemi se često otkrivaju tek u fazi testiranja sustava. Također, pune performanse sustava se mogu testirati tek kada je sustav gotov. Ukoliko se pokaže problem nedostatka kapaciteta teško ga je ispraviti.
- Primjena ovog modela proizvodi prekomjernu dokumentaciju za koju treba odvojiti dosta vremena kako bi je poslije održali ažurnom. Osim toga, pisana specifikacija zahtjeva je korisnicima teška za čitanje.
- Model ima jasno definiranu podjelu odgovornosti, pa je u razvojnom timu slaba suradnja, što stvara jaz između korisnika i informatičara.

Situacije u kojima je vodopadni model primjenjiv:

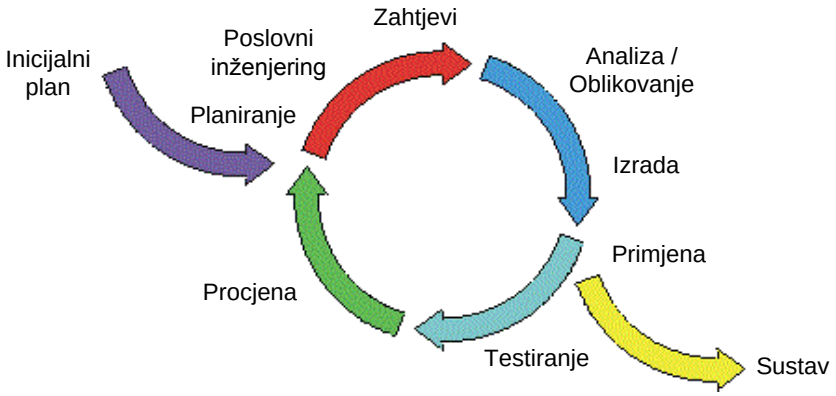
- Za sustave koji su predviđeni da rade u *client-server* okruženju.
- Projekt ima jasne ciljeve i rješenja. Uz to ne postoji pritisak za neposrednu provedbu. U zadanim kontrolnim točkama zahtjevi se isključivo formalno odobravaju.
- Projektni su zahtjevi naznačeni razumljivo i nedvosmisleno. Za vrijeme trajanja ciklusa razvoja IS-a zahtjevi su stabilni i nepromjenjivi.
- Korisnik je stručan u svom području i informatički obrazovan.
- Sastav razvojnog tima je neizvjestan i varira tijekom vremena. Dopušta se da članovi tima mogu biti i neiskusni.

Situacije u kojima vodopadni model nije primjenjiv:

- Za sustave u realnom vremenu i sustave koji su upravljani događajima.
- Veliki projekti u kojima zahtjevi nisu u cijelosti razumljivi ili se mogu promijeniti iz nekog od ovih razloga: vanjske promjene, promjene očekivanja, izmjene proračuna, brza promjena tehnologije i slično.
- IS za web, jer se takvi sustavi razvijaju pod pritiskom brze provedbe, zahtjevi se stalno mijenjaju, a za razvoj su potrebni ljudi s iskustvom.

3.1.2. Iterativni model

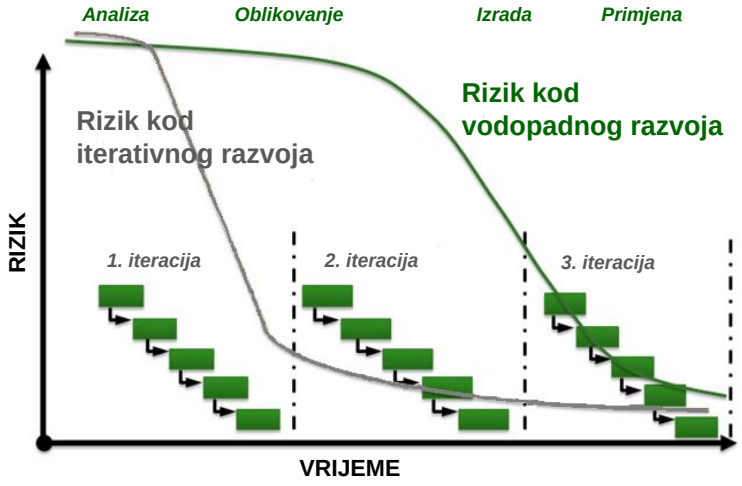
Iterativni model pruža mogućnost vraćanja na faze analize, oblikovanja te izrade i primjene ako postoje novi zahtjevi koje treba ugraditi u sustav (slika 3.3.).



Slika 3.3. Iterativni model

Jedna od važnih karakteristika iterativnog modela je da omogućava bolje upravljanje i prepoznavanje rizika razvoja. Kako se razvoj programske podrške bavi određenom vrstom inovacija, nije jednostavno predvidjeti takav razvoj, bez obzira na stručnost razvojnog tima. Ukoliko je moguće što ranije u životnom ciklusu prepoznati rizike, utoliko je vjerojatnije da će uspjeh projekta biti pozitivan.

Sa slike 3.4. se vidi usporedba prepoznavanja rizika za vodopadni i iterativni model razvoja.



Slika 3.4. Usporedba upravljanja rizicima u vodopadnom i iterativnom modelu

U vodopadnom modelu rizik se ne može izbjeći analizom i oblikovanjem jer se prepoznaje tek kasno u razvoju. Procjena programskog proizvoda se radi nakon ili za vrijeme izrade i testiranja, te je rizik da napravljeno ne odgovara zahtijevanom vrlo velik.

U iterativnom modelu se po iteracijama radi procjena dijela koji je izrađen. Stoga se rano u razvoju cijelog projekta može reagirati na prepoznate rizike i tako ih umanjiti i eliminirati na vrijeme.

Osnovne karakteristike iterativnog modela (slika 3.3.):

- Iterativni pristup je primjenjiv u slučaju kada je cjelina podijeljena na dijelove. U tom slučaju, kako je već pokazano, rizik projekta se umanjuje u ranijim fazama razvoja.
- Manji dijelovi sustava se razvijaju praćenjem procesa koji je iterativno promjenjiv dokle god prototip ne preraste u gotovi proizvod.
- Prototipiranjem se može dogoditi da neka verzija bude odbačena, no također se događa da verzija preraste u gotovo rješenje.
- Korisnik je bolje uključen u cjelokupan proces razvoja jer se svaka iteracija provjerava. Uz to, ako informatičari razumiju osnovne poslovne probleme, to jamči izbjegavanje rješavanja loše postavljenih problema.

Dobre strane iterativnog modela:

- Može se koristiti za realističan model važnih dijelova sustava i to za vrijeme svake faze tradicionalnog razvoja IS-a.
- Pogodan je za rješavanje nejasnih ciljeva, razvoj i provjeru korisničkih zahtjeva, eksperimentiranje i usporedbu različitih dizajnerskih rješenja, te istraživanje performansi i sučelja.
- Postojeći potencijal znanja stečenog u ranijim iteracijama iskoristiv je za unapređenje iteracija koje se poslije javljaju.
- Omogućava sudjelovanje korisnika u razvoju sustava i komunikaciju među sudionicima projekta.
- Pogodan je ako ima mnogo korisnika koji nisu u stanju specificirati svoje potrebe te informatičar isporučuje probni sustav u najkraćem mogućem vremenu.
- Pomaže da se lakše prepoznaju zbunjujuće, komplicirane ili nedostajuće funkcionalnosti sustava.
- Potiče inovativnost i fleksibilno oblikovanje.
- Omogućava brzu implementaciju nekompletnog, ali funkcionalnog programskog rješenja.

Loše strane iterativnog modela:

- Ne provodi se stoga kontrola i uvođenje procesa razvoja IS-a.
- Nepotpuna i neodgovarajuća analiza može rezultirati površnim i jednostavnim rješenjima koja se lako ugrađuju u sustav, ali su nepotpuna.
- Zahtjevi se mogu često i značajno mijenjati.
- Informatičari prebrzo izrađuju rješenja koja nemaju dovoljno fleksibilan dizajn i tako ograničavaju potencijale programa u budućnosti. Ako su dizajneri sustava neiskusni, izrađuje se „brz i prljav sustav“ bez zajedničkog dogovora o integraciji s ostalim komponentama sustava.

- Uzrokuje kriva očekivanja, jer kada korisnik vidi prototip pogrešno misli kako je sustav gotov, naročito kada sučelje izgleda dobro, ali funkcionalnost je još daleko od potpunosti.
- Iteracije mogu povećati proračun i vrijeme izrade projekta. Stoga treba uvijek ponovno vrednovati potencijalne dobiti koje će pružiti novi sustav.

Situacije u kojima je iterativni model primjenjiv:

- Projekt koji razvija *on-line* sustav i zahtijeva stalni dijalog s korisnikom, ili projekt koji ima veliki broj korisnika, integracije, funkcionalnosti, gdje uočeni rizik projekta treba povezati sa zahtjevima koje treba smanjiti.
- Ciljevi projekta nisu dovoljno jasni.
- Postoji pritisak da se što prije izradi i implementira *bilo što*.
- Funkcionalni zahtjevi će se možda mijenjati često i značajno.
- Korisnik nema cjelovito znanje o poslovnom području.
- Sastav projektnog tima je stabilan i članovi tima su iskusni.
- Voditelj projekta je iskusan.
- Ne postoje strogo određeni zahtjevi za koje treba suglasnost u zadanim kontrolnim točkama.
- Prije pokretanja projekta korisnik i analitičar definiraju poslovne probleme koje treba riješiti.
- Implementacija inovativnih i fleksibilnih rješenja koja će možda trebati ubuduće prilagođavati, ne smije biti kritična za poslovanje

Situacije u kojima iterativni model nije primjenjiv:

- Sustavi koji su većinom transakcijski ili sustavi elektroničkog poslovanja koji su web orijentirani.
- Nestabilan sastav projektnog tima.
- Očekuju se buduće nadogradnje programskih rješenja.
- Ciljevi projekta su vrlo jasni, a rizik projekta, s obzirom na definirane zahtjeve, nizak.

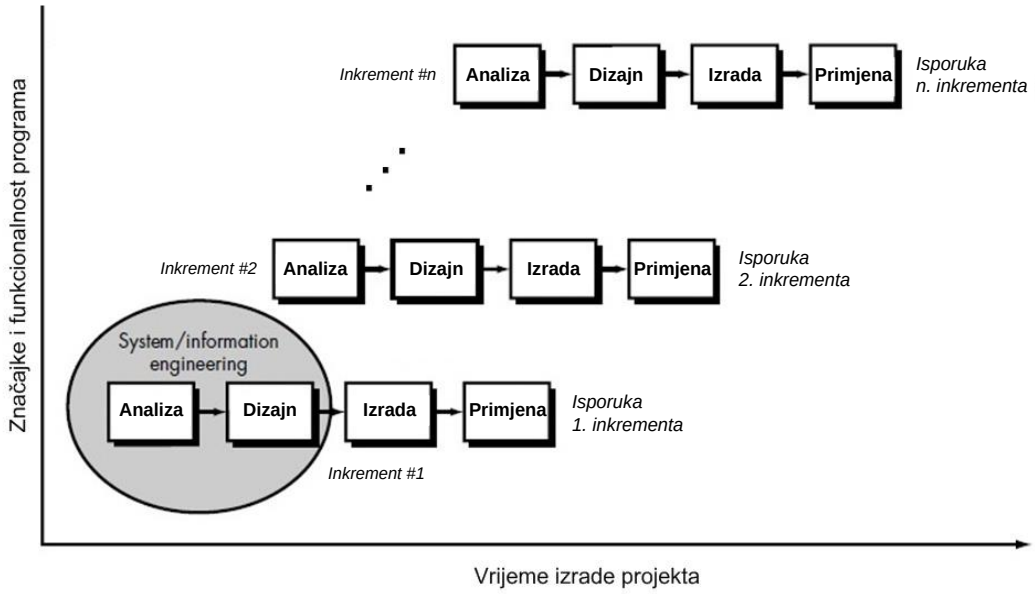
3.1.3. Evolucijski model

Evolucijski model bi se mogao opisati kao kombinacija vodopadnog i iterativnog modela.

Osnovne karakteristike evolucijskog modela (slika 3.5.):

- Izvodi se niz *mini-vodopadnih* ciklusa u kojima su za jedan dio sustava odrađene kompletno sve faze jednog ciklusa vodopadnog pristupa, nakon čega se pokreće novi ciklus.

- Cjelokupni zahtjevi su definirani prije pokretanja evolucijskog procesa koji će biti rješavan *mini-vodopadnim* iteracijama.
- Inicijalno planiranje, analiza zahtjeva te oblikovanje arhitekture i osnovnog sustava definirani su vodopadnim pristupom i korištenjem iterativnog prototipiranja, sve do konačne instalacije gotovog prototipa koji je ujedno i radni sustav.



Slika 3.5. Evolucijski model

Dobre strane evolucijskog modela:

- Znanje koje je stečeno u početnim inkrementima predstavlja potencijal za naknadne inkremente.
- Pisana dokumentacija, formalni pregled i odobrenje od korisnika osigurava umjerenu kontrolu u zadanim glavnim točkama kontrole tijekom trajanja projekta.
- Sudionici mogu pružiti konkretne dokaze o statusu projekta tijekom cijelog životnog ciklusa.
- Pomaže u ublažavanju rizika integracije i arhitekture ranije u projektu razvoja.
- Omogućuje isporuku niza implementacijskih rješenja koja su cjelovita i funkcionalno predstavljaju gotovi proizvod (u jednom inkrementu jedan njegov dio).
- Postupna provedba pojedinog inkrementa svaki put pruža mogućnost za praćenje učinaka. Tako se pitanja i problemi praćenja izoliraju samo na taj inkrement, koji se onda lakše usklađuje, te je minimiziran njegov negativan utjecaj na korisnika.

Loše strane evolucijskog modela:

- Primjena *mini-vodopadnog* pristupa za male dijelove sustava, a prije prelaska na idući inkrement, umanjuje pregled i razmatranje poslovnih problema i tehničke zahtjeve za cjelokupni sustav.
- Na samom početku razvoja treba dobro definirati sučelja jer se neki moduli završavaju prije drugih (koji mogu biti operativno kroz poslovanje usko povezani).
- Teški problemi se guraju za naknadno rješavanje, a sve s ciljem da se vrlo rano u razvoju prikaže uspjeh projekta.

Situacije u kojima je evolucijski model primjenjiv:

- Veliki projekti gdje zahtjevi nisu dobro shvaćeni ili su promjenjivi i ovise o vanjskim izmjenama, promijenjenim očekivanjima, promjeni proračuna ili brzoj promjeni tehnologije.
- Web IS ili sustavi upravljani događajima.
- Vodeće aplikacije.

Situacije u kojima evolucijski model nije primjenjiv:

- Vrlo mali, kratkotrajni projekti.
- Mali rizik integracije i arhitekture.
- Programi visoke interaktivnosti za koje već postoje podaci (u cijelosti ili djelomično). Projekt se uglavnom sastoji od analize i izvješćivanja o tim podacima.

3.1.4. Spiralni model

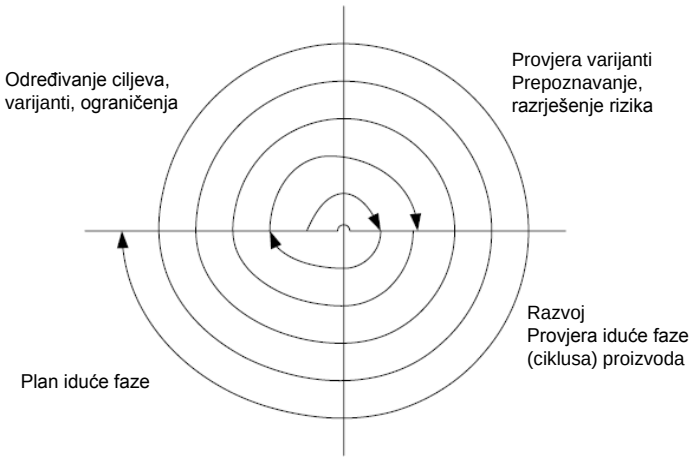
I spiralni model se može promatrati kao kombinacija vodopadnog i iterativnog modela. Već je istaknuto da je vodopadni model dobar kada su zahtjevi dobro definirani, a razvoj programske podrške detaljno ugovoren.

Međutim, vodopadni model nije dobar za, primjerice, interaktivne aplikacije za krajnjeg korisnika. U tom slučaju specifikacija zahtjeva je vrlo teška jer je dizajn sučelja subjektivan, a korisniku često nisu jasne objektivne potrebe ovakvih programskih rješenja. Stoga ovdje nije pogodno imati tromu razradu specifikacije zahtjeva te projektiranje i razvoj velikih količina neupotrebljivog koda (što je za očekivati od vodopadnog modela). Dodatni problem vodopadnog pristupa je slabo upravljanje rizicima u početnim fazama razvoja (slika 3.4.).

Stoga je 1988. Barry Boehm[13] predložio sveobuhvatan model životnog ciklusa razvoja IS-a nazvan spiralni model. Glavna karakteristika ovog modela je pristup razvoju vođen rizicima (eng. *risk-driven*), za razliku od dotadašnjeg često korištenog pristupa

[13] Barry W. Boehm (1935) je američki programski inženjer i profesor emeritus Sveučilišta Južne Kalifornije, a poznat je po značajnom doprinosu programskom inženjerstvu. Istraživao je modele procesa razvoja, inženjerstvo zahtjeva, arhitekturu programa, metriku i troškove programskih rješenja i programsko inženjerstvo zasovano na znanju (eng. *knowledge-based*). Njegovi poznatiji doprinosi su: COCOMO metoda, spiralni model i W pristup (*win-win*).

vođenog dokumentacijom ili programskim kodom (eng, *document-driven, code-driven*). Na slici 3.6. je osnovni koncept spiralnog modela na kojem se vidi da je prostor za razvoj sustava predviđen samo u donjem desnom kvadrantu. Spiralni model se čita *iznutra prema vani* što simbolizira njegovu iterativnu prirodu.



Slika 3.6. Osnovni koncepti spiralnog modela

Osnovne karakteristike spiralnog modela (slika 3.7.):

- Fokus je na procjeni rizika tako da se razbijanjem projekta na manje dijelove i lakšim omogućavanjem izmjena tijekom procesa razvoja minimizira rizik projekta. Model pruža mogućnost provjere rizika i prosuđivanja napretka projekta kroz životni ciklus razvoja.
- „Svaki ciklus uključuje napredovanje kroz isti niz koraka i to za svaki dio proizvoda, od ciljeva i zahtjeva, provjere rizika, do razvoja i kodiranja.“ (*Boehm, 1986.*)
- Svaki ciklus po spirali pokriva četiri kvadranta, i to: (1) određivanje ciljeva, varijanti i ograničenja, (2) provjera varijanti, te prepoznavanje i razrješenje rizika, (3) razvoj programskog rješenja i provjera idućeg ciklusa te (4) plan iduće faze razvoja. (*Boehm, 1986. i 1988.*)
- Svaki ciklus treba započeti s određivanjem sudionika i njihovih zadataka, a završava s pregledom i provjerom obavljenog posla. (*Boehm, 2000.*)

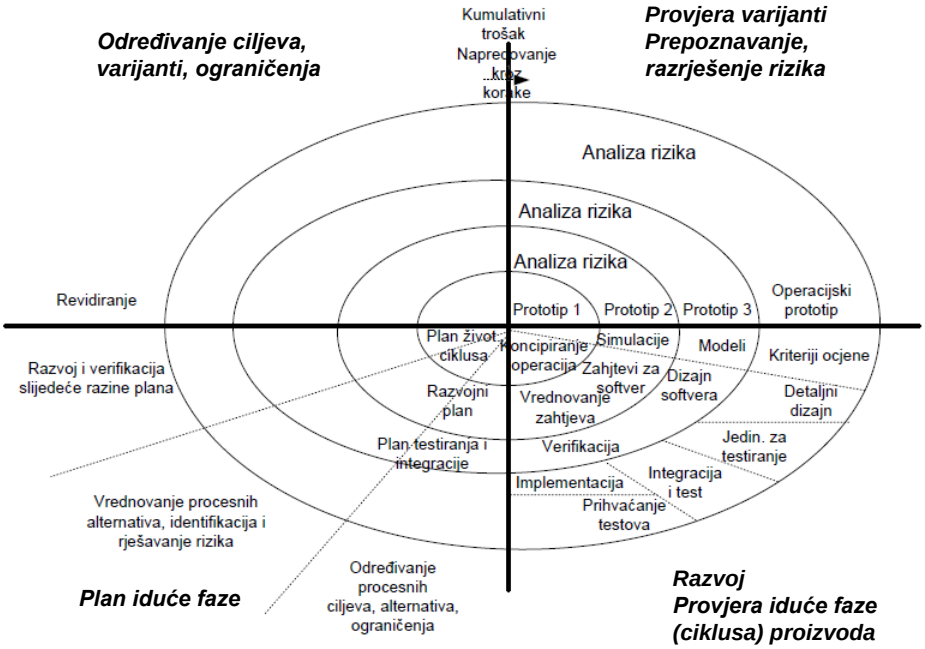
Dobre strane spiralnog modela:

- Poboljšava se izbjegavanje rizika projekta.
- Pomaže u odabiru najbolje metodologije razvoja koja uključuje iterativni pristup zasnovan na vrednovanju rizika projekta.

- Može povezati vodopadni, iterativni i/ili evolucijski model kao specijalni okvir koji pruža smjernice za najbolje karakteristike iterativnog razvoja uz vrednovanje projektnog rizika. Primjerice, projekt koji ima nizak rizik ispunjenja korisničkih zahtjeva, ali zato visok rizik prekoračenja proračuna ili ciljanih rokova, u osnovi će spiralnom metodom slijediti linearni vodopadni pristup za zadane iteracije razvoja. Obrnuto, ako je rizik nad zahtjevima visok a nad proračunom i rokovima nizak, spiralna metoda dopušta iterativni pristup prototipiranjem (slika 3.7.).

Loše strane spiralnog modela:

- Izazov je odrediti točnu strukturu metodologije razvoja koja će se koristiti za svaku pojedinu iteraciju u spirali.
- Visoka razina prilagođavanja svakom pojedinom projektu, što čini složenim pojedini projekt i umanjuje mogućnost ponovnog korištenja (eng. *reuse*) izrađenih komponenti.
- Znanje i vještina voditelja projekta treba biti na visini, tako da se zna postaviti prema svakom novom projektu.
- Nema uspostavljenih (zadanih) kontrola prilikom prelaska iz jednog ciklusa u drugi. A bez zadanih kontrola može se dogoditi da pojedini ciklus generira još više posla za idući ciklus.
- Ne postoje zadani rokovi. Ciklusi slijede bez jasno definiranih uvjeta prekida. Stoga je prisutan latentni rizik da se neće ispuniti zadani rokovi ili proračun.
- Postoji mogućnost da projekt završi provodeći okvir vodopadnog modela.



Slika 3.7. Detaljni pregled spiralnog modela[14]

[14] http://oliver.efos.hr/nastavnici/jmesaric/modeliranje/P2-MODELI%20I%20METODE.pdf (16.01.2012.)

Situacije u kojima je spiralni model primjenjiv:

- Sustavi u realnom vremenu ili sigurnosno-kritični sustavi.
- Projekt se može razvijati kombinacijom više metoda.
- Implementacija ima prioritet nad funkcionalnosti; ova zadnja će biti ugrađena u kasnijim ciklusima.
- Voditelji projekta imaju veliko iskustvo.
- Izbjegavanje rizika ima visoki prioritet.
- Minimiziranje korištenja resursa nema apsolutni prioritet.
- Za zahtjeve se provodi strogo odobravanje i kontrola dokumentacije.

Situacije u kojima spiralni model nije primjenjiv:

- Ne dozvoljava se izbjegavanje rizika.
- Visok stupanj točnosti sustava nije u prvom planu.
- Funkcionalnost ima prioritet nad provedbom.
- Apsolutni prioritet je minimiziranje resursa.

3.1.5. Brzi razvoj aplikacija

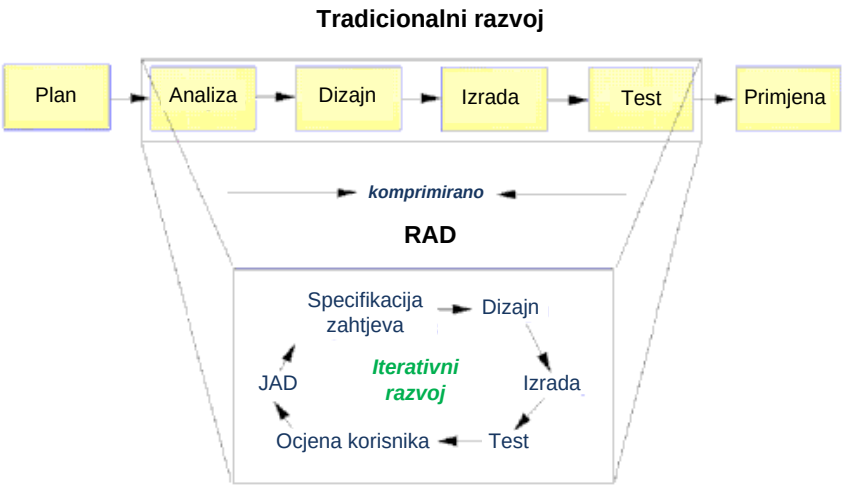
Brzi razvoj aplikacija (eng, *Rapid Application Development*, RAD) je prilagodljiva metoda koja se koristi minimum planiranja a potiče brzo prototipiranje do konačnog proizvoda. Plan i analiza su uklopljeni u izradu programa koji se onda izrađuju mnogo brže. Ovakav pristup ujedno omogućava i lakše izmjene zahtjeva. RAD uključuje model iterativnog razvoja i programskog prototipiranja. Tako se izrađuju djelomične verzije aplikacija koje mogu evoluirati do konačnog rješenja.

RAD predstavlja moderan pristup razvoju programske potpore a nastao je kao odgovor na tradicionalne modele i pripadajuće metode, primjerice na vodopadni i iterativni razvoj (slika 3.8.).

Osnovne karakteristike brzog razvoja aplikacija su:

- Glavni cilj je brzi razvoj i isporuka visoko kvalitetnog rješenja uz relativno niske troškove investiranja.
- Nastojanje da se smanje svojstveni rizici projekta tako da se projekt podijeli u manje segmente i tako lakše osiguraju potrebne izmjene na proizvodu.
- Pristup je usmjeren na brzu izradu sustava visoke kvalitete koristeći se iterativnim prototipiranjem (u bilo kojoj fazi razvoja), aktivnim uključivanjem korisnika i korištenjem alata za razvoj programa. Ti alati trebaju uključivati GUI *builder*, CASE alate, DBMS, jezike za programiranje 4G, generatore koda i objektno-orijentirane tehnike.

- Ključni naglasak je na ispunjenju poslovnih potreba, dok je tehnološka i inženjerska izvrsnost u drugom planu.
- Kontrola projekta uključuje razvoj prioriteta i određivanje kontrolnih točaka, takozvani *timeboxes* (na slici 3.9., to je 60-90 dana). Ako projekt počinje proklizavati treba ograničiti zahtjeve kako bi se obvezno ispunio *timebox*, a ne prolongirala kontrolna točka.
- Aktivno sudjelovanje korisnika je imperativ.
- Općenito uključuje i takozvani zajednički razvoj aplikacija (eng. *Joint Application Development*, JAD; vidi sliku 3.8.), u kojima su korisnici aktivno uključeni u oblikovanje sustava.
- Rezultati iteracija u konačnici proizvode program, za razliku od klasičnog prototipiranja koje se odbacuje (djelomično ili u cijelosti).
- Izrađuje se samo ona dokumentacija koja je nužna za budući razvoj i održavanje.
- Okvir ovakvog pristupa omogućava provođenje standardnih tehnika analize i oblikovanja.

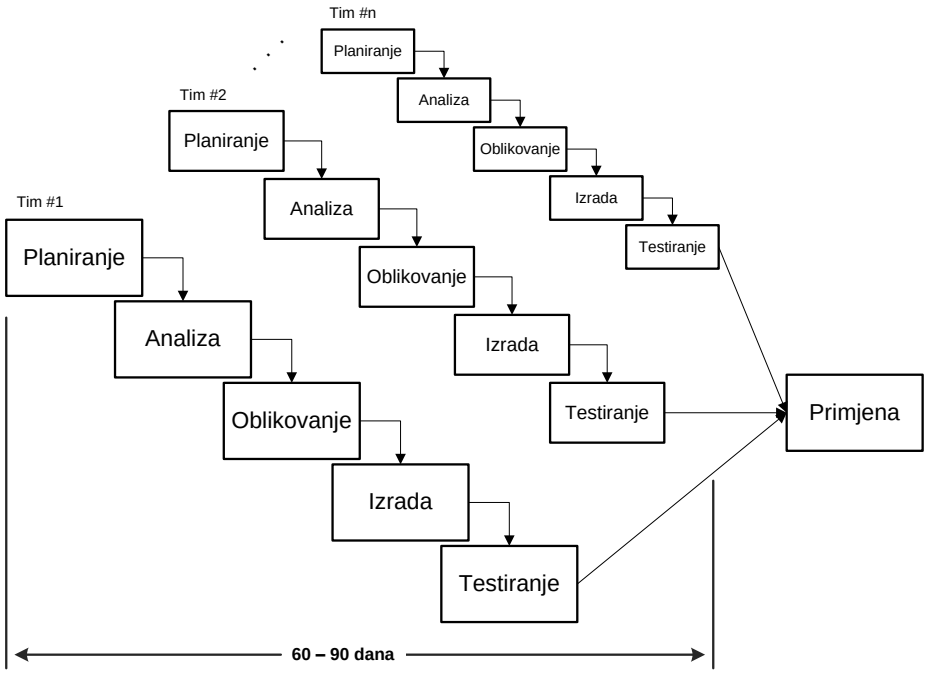


Slika 3.8. RAD i tradicionalni pristup

Dobre strane RAD pristupa:

- Uporabljiva verzija aplikacije je dostupna prije nego u vodopadnom, evolucijskom ili spiralnom pristupu.
- Kako RAD producira funkcionalni sustav mnogo prije, nastoji se da ta proizvodnja bude sa što manjim troškovima.
- Sudionici projekta angažirani su više nego u vodopadnom, evolucijskom ili spiralnom pristupu. Na taj način korisnici dobivaju sigurnost vlasništva nad sustavom, a informatičari zadovoljstvo brzom izradom uspješnog proizvoda.
- Usmjerenost na ključne elemente sustava kako ih vidi korisnik.

- Pruža mogućnost brze promjene oblikovanja sustava na zahtjev korisnika.
- Pruža čvršću poveznicu između korisničkih zahtjeva i specifikacije sustava.
- Općenito proizvodi dramatične uštede u vremenu, novcu i naporu sudionika.



Slika 3.9. Razvojni ciklusi metode RAD

Loše strane RAD pristupa:

- Brzina izrade sustava i niska cijena mogu općenito dovesti do niže kvalitete sustava.
- Opasnost od neusklađenosti razvijenog sustava s poslovanjem zbog nedostatka informacija. Mogućnost nedosljednog oblikovanja unutar sustava i među sustavima.
- Projekt može realizirati više zahtjeva nego što je potrebno.
- Mogućnost proklizavanja značajki (svojstava) sustava ako se sve više i više značajki dodaje, a da one nisu proizišle iz razvojnih aktivnosti.
- Mogućnost kršenja programskih standarda koji se odnose na nekonzistentnost konvencija imenovanja i nekonzistentnost dokumentacije.
- Poteškoće s izrađenim modulima sustava prilikom ponovnog korištenja u budućim projektima.
- Potencijalno manjak skalabilnosti (nadogradnje) u oblikovanju sustava.
- Potencijalno nedostatak pažnje koju treba posvetiti administriranju sustava nakon uvođenja u rad.

- Visoka cijena stručnog kadra korisnika koji sudjeluju u razvoju sustava.
- Formalne provjere i ispitivanja parcijalnih modula provode se teže nego kod cjelovitog sustava.
- Nastojanje da se rješavanje teških problema odgađa za buduće iteracije u razvoju projekta rano pokazuje neuspjeh upravljanja projektom.
- Odmah na početku razvoja obvezno je dobro definirati sučelja jer će kroz iteracije neki moduli biti gotovi prije drugih.

Situacije u kojima je RAD primjenjiv:

- Projekti male i srednje težine i kratkog trajanja (ne više od 6 čovjek/godina).
- Cilj projekta je jasan, a ciljevi poslovanja dobro definirani i ograničeni.
- Program je vrlo interaktivan, za jasno definiranu skupinu korisnika, ali računalno (programerski) nije složen.
- Funkcionalnosti sustava su jasno vidljive kroz korisničko sučelje.
- Korisnici dobro poznaju područje primjene sustava.
- Ukoliko se sustav razvija u okviru inovacija ili novih spoznaja onda nije moguće precizno, u zadanom vremenu, definirati korisničke zahtjeve.
- Članovi tima posjeduju vještine komunikacije i ophođenja s ljudima, jednako koliko su vješti i u svom poslu.
- Sastav tima je stabilan, a kontinuitet osnovnog dijela tima za razvoj treba održavati.
- Programeri posjeduju vještinu korištenja naprednih alata i tehnologija.
- Ključne tehničke komponente su poznate i definirane.
- Tehnički zahtjevi (vrijeme odgovora, veličina baze, protok podataka i slično) su razumljivi i primjenjivi u okviru mogućnosti korištene tehnologije.
- Razvojni tim je osposobljen da donosi odluke o dizajnu programa na dnevnoj razini bez potrebe da se savjetuje s nadređenima.

Situacije u kojima RAD nije primjenjiv:

- Infrastrukturno veliki projekti ili veći distribuirani IS sa rasprostranjenom bazom podataka.
- Sustavi u realnom vremenu ili sustavi kojima je kritična sigurnosna komponenta.
- Računalno složeni sustavi u kojima, unutar zadanog projekta, treba analizirati, oblikovati i izraditi opsežne i složene količine podataka.
- Zadan je opseg projekta dok su poslovni ciljevi nejasni.
- Razvoj koji ima u cijelosti specificirane sve funkcionalne zahtjeve, a nije započeta izrada programa.
- Veliki broj ljudi odlučuje o projektu, ali su uglavnom nedostupni ili su geografski dislocirani.

- Projektni tim je velik ili ima više projektnih timova koje treba koordinirati.
- Ukoliko postoji manjak sudionika kod korisnika ili nedostatak njihove predanosti projektu.
- Nema izvrsnih (stručnih) ljudi u timu koji će *nositi* projekt.
- U projekt će biti uvedene nove tehnologije ili je tehnološka arhitektura nejasna te će se za prvo vrijeme koristiti raznim tehnologijama.
- Tehnički zahtjevi (vrijeme odgovora, veličina baze, protok podataka i slično) su kompleksni i ne odgovaraju opremi koja se koristi.

3.2. Unificirani proces

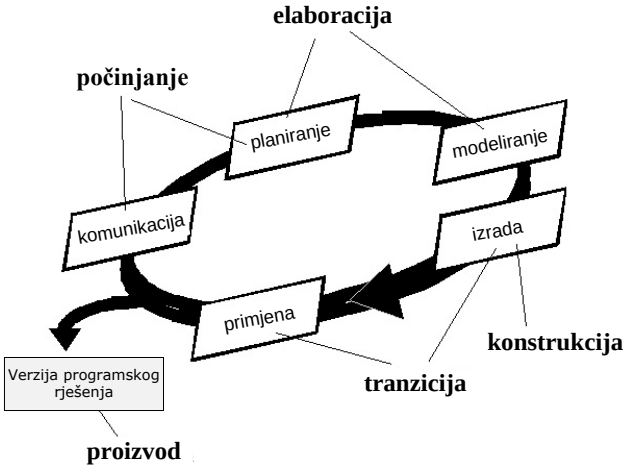
Unificirani proces (eng. *Unified Process*) predstavlja rezultat nastojanja da se iskoriste najbolje značajke i karakteristike konvencionalnih (tradicionalnih) modela razvoja IS-a u koji su implementirani mnogi dobri principi agilnog razvoja programa (eng. *Agile Software Development*; više u idućem poglavlju).

Unificirani proces prepoznaje važnost komunikacije s korisnikom i modernije metode za opisivanje pogleda na sustav onako kako ga vidi korisnik (primjerice dijagram slučaja korištenja; više u 4. poglavlju). Također naglašava važnu ulogu arhitekture softvera i pomaže da se arhitekti sustava fokusiraju na prave ciljeve kao što je razumljivost, povezanost s budućim izmjenama i ponovna iskoristivost (eng. *reuse*).

3.2.1. Faze unificiranog procesa

Kroz 80-te i 90-te godine prošlog stoljeća objektno-orijentirane (OO) metode i programski jezici naišli su na širok odaziv i zainteresiranost kod informatičke zajednice. Ova nova paradigma razvija objektni pristup analizi (OOA) i oblikovanju (dizajnu, OOD) te programiranju (OOP). Tako je uvedena općenita primjena objektno-orijentiranog modeliranja procesa, slična evolucijskom modelu pokazanom u poglavlju 3.1.3.

Za opis unificiranog procesa koriste se općenite faze razvoja (komunikacija i planiranje, modeliranje u analizi i oblikovanju, izrada i primjena). Na slici 3.10. [17, poglavlje 3.] pokazane su faze unificiranog procesa i njihova korelacija s općenitim modelom razvoja.



Slika 3.10. Unificirani proces

Faza *početak* objedinjuje komunikaciju s korisnikom i aktivnosti planiranja projekta i zadataka razvoja. Identificiraju se poslovni (funkcionalni) zahtjevi, predlaže se ugrubo arhitektura sustava te plan iteracija razvoja. Skup početnih slučajeva korištenja opisuje osnovne poslovne zahtjeve koji pomažu u prepoznavanju ciljeva projekta i pružaju osnovu za planiranje projekta.

Faza *elaboracije* objedinjuje komunikaciju s korisnikom i aktivnosti modeliranja sustava. Elaboracija proširuje i detaljizira početne slučajeve korištenja iz faze *početak*, te proširuje arhitekturu sustava prikazujući je kroz pet različitih modela (pogleda) na programska rješenja: model slučajeva korištenja (specifikacije zahtjeva), model analize, model oblikovanja, model implementacije (razvoja programa) i model primjene novog sustava.

Faza *konstrukcije* koristi se modelom arhitekture sustava kao ulaznim skupom informacija pomoću kojih se razvijaju programske komponente (moduli) koje će pojedine slučajeve korištenja učiniti operativnima za krajnjeg korisnika. Kako se pojedine komponente primjenjuju tako se izrađuju testovi sustava i provode testiranja nad tim komponentama. Slučajevi korištenja upotrebljavaju se kako bi se izveli odgovarajući testovi prihvatljivosti čiji rezultat se razmatra u idućoj fazi (tranzicija).

Faza *tranzicije* objedinjuje zadnje etape aktivnosti faze konstrukcije i prvi dio aktivnosti primjene novog sustava. Programi se daju korisniku kao beta verzija za koju će korisnik dati primjedbe o greškama i potrebnim izmjenama. Pročišćena verzija zajedno sa uputama za rad, rukovanje s problemima i procedurom instalacije predaje se korisniku kao radna verzija programa.

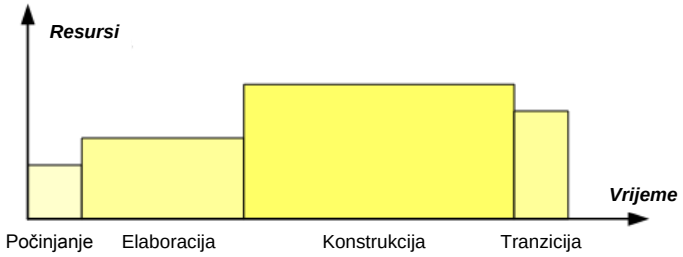
Faza *proizvoda* nastavak je primjene novog sustava u kojoj se prati rad programske potpore, održava operativna infrastruktura, te prikuplja i procjenjuje izvješće o greškama i zahtjevi za izmjenama.

Što unificirani proces predstavlja sa stanovišta tehnologije i objektnog pristupa pokazuje ovaj opis: *use-case driven, architecture-centic, iterative and incremental*.

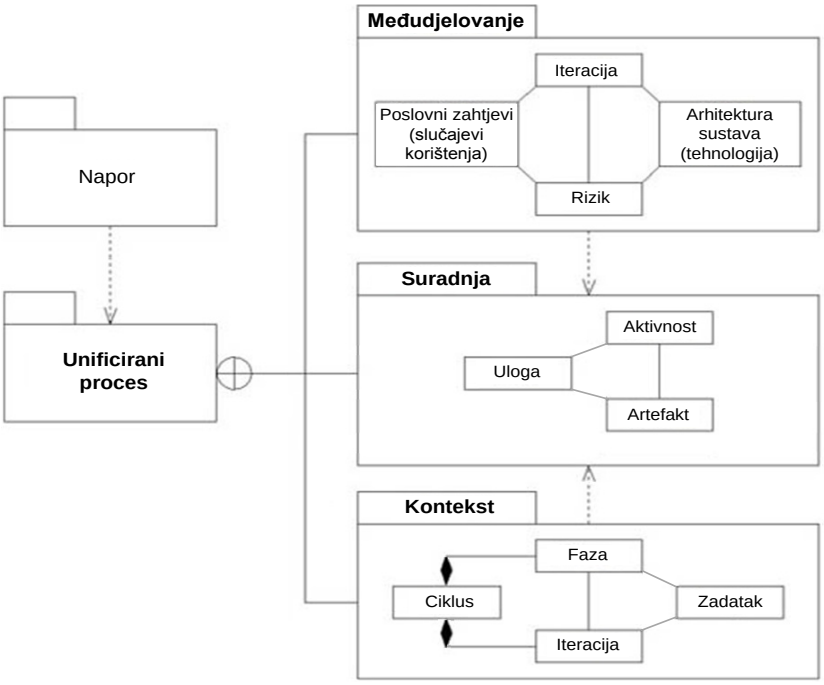
3.2.2. Karakteristike unificiranog procesa

Karakteristike unificiranog procesa:

- Široko je primjenjiv na različite vrste programskih rješenja, uključujući i male i velike projekte koji imaju različite stupnjeve upravljačke i tehnološke složenosti. Tako slika 3.11. pokazuje profil tipičnog projekta koji pokazuje relativni odnos četiri faze unificiranog procesa.
- Ideja koja pruža okvir za infrastrukturu i izvršavanje projekta, ali bez detalja, samo opis životnog ciklusa koji uključuje kontekst, suradnju i međudjelovanje sudionika i tehnologije (slika 3.12.).



Slika 3.11. Relativni odnos faza unificiranog procesa s obzirom na resurse



Slika 3.12. Kontekst, suradnja i međudjelovanje unificiranog procesa[15]

[15] http://www.methodsandtools.com/archive/archive.php?id=32 (18.01.2012.)

Dakle, unificirani proces je općeniti naziv za grupu modela procesa razvoja koji zadovoljavaju više kriterija kao što su: iterativni i inkrementalni pristup, upravljanje pomoću slučajeva korištenja i rano fokusiranje na prepoznavanje i rješavanje rizika projekta.

Takozvani *Rational Unified Process* (RUP) je usavršeni i detaljizirani unificirani proces koji je osmislila kompanija Rational Software (sada u vlasništvu IBM-a). RUP pristup upotrebljava niz programskih alata za oblikovanje okvira kojima se definira kako provesti aktivnosti upravljanja i razvoja projekta, zajedno s okvirom koji definira potrebe i zadatke projektnog rima. O RUP pristupu će detaljno biti riječi u 5. poglavlju.

3.3. Agilno programsko inženjerstvo

Relativno mlada metoda razvoja, poznata kao agilni (brzi, žustri) pristup, pokušava zamijeniti tradicionalnu procesno vođenu (eng. *process-driven*) dokumentaciju i obveznu detaljnu specifikaciju zahtjeva. Važno je spomenuti da agilni pristup pretpostavlja dovoljan broj iskusnih ljudi u razvojnom timu.

Glavni cilj agilnost je potaknuti razumijevanje sustava mnogo brže nego što to pruža tradicionalni pristup te omogućiti jeftiniji razvoj, a sve zasnovano na postojećem znanju. Filozofija agilnog pristupa formulirana je u takozvanom **manifestu**[16] (eng. *Manifesto for Agile Software Development*).

Formalni prijevod manifesta glasi[17]:

Proglas o metodi agilnog razvoja programa

Tražimo bolje načine razvoja programa
razvijajući programe i pomažući drugima pri njegovom razvoju.
Takvim radom smo naučili da više cijenimo:

Ljude i njihove međusobne odnose nego procese i oruđa
Upotrebljive programe nego iscrpnu dokumentaciju
Suradnju s korisnicima nego pregovaranje oko ugovora
Reagiranje na promjene nego ustrajanje na planu.

Drugim riječima, iako cijenimo vrijednosti na desnoj strani,
više vjerujemo u one na lijevoj.

u potpisu je 17 autora manifesta

Kako agilnost funkcionira u kontekstu programskog inženjerstva? Odgovor na ovo pitanje ponudio je Ivar Jacobson[18] [17]:

[16] http://agilemanifesto.org/ , potpisano 2001. od strane 17 vodećih programskih razvojnika.
[17] http://agilemanifesto.org/iso/hr/ (25.01.2012.)
[18] Švedski informatičar poznat po doprinosu u stvaranju UML, RUP, AOSD pristupa i *Objectory* metodologije.

> „Danas je agilnost općepoznat pojam kada se opisuje moderni pristup razvoju programa. Danas je sve agilno. Agilni tim je okretan tim sposoban brzo reagirati na promjene. A promjena u razvoju programa ima dosta. Promjene su prisutne u izgradnji programa, u sastavu projektnog tima, zbog novih tehnologija, te razne druge promjene koje utječu ili na programski proizvod ili na projekt razvoja programskog proizvoda. Podrška promjenama treba biti ugrađena u sve što je povezano s programima. Agilni tim je svjestan činjenice da programe razvijaju pojedinci koji rade u timu, te su vještine tih ljudi i njihova sposobnost da međusobno surađuju najvažniji za uspjeh projekta."

Po Jacobsonu, sveprisutnost promjena je osnovni pokretač agilnosti. Programski inženjeri trebaju biti agilni ako žele prilagoditi programski proizvod brzom pojavljivanju potreba za promjenama.

3.3.1. Principi agilnog razvoja

Prilikom razvoja programa informatičar razmišlja o planu razvoja, uglavnom sa stajališta metodologije kojom se koristi: što napraviti prvo, što zatim slijedi kao drugo, treće te koje zadatke i probleme riješiti. Već je opisano da prvo dolazi studija područja problema i pronalaženje rješenja problema pomoću postojeće tehnologije. Zatim dolazi razvoj programa kroz zadatke oblikovanja, izrade i implementacije sustava, a na kraju se sustav uvodi u rad i procjenjuju se njegovi učinci.

Ovakvo sekvencijalno razmišljanje o slijedu zadataka vodi ka modelu vodopadnog pristupa (ili nekog složenijeg općeg modela koji u sebi ima iteracije sa sekvencijalnim pristupom). Uz takav pristup nužno ide i opsežna dokumentacija.

Međutim, klijent ne vidi razvoj programske potpore na ovakav način. Korisniku nikad nije bilo jednostavno objasniti zašto se programi razvijaju tako sporo (a nije jednostavno ni danas, unatoč činjenici da su korisnici mnogo više informatički obrazovani i osviješteni, nego su to bili prije desetak i više godina). Korisnici bi radije da mogu što prije vidjeti neke osnovne funkcionalnosti sustava, a nakon toga se može program širiti i dorađivati.

Pobornici i korisnici agilnog pristupa ne pouzdaju se previše u vizualni prikaz spoznaja i modeliranje pomoću dijagrama i drugih grafičkih tehnika. Stoga nema mnogo primjera praktičnih modela agilnih projekata u literaturi. Ponekad taj pristup ide prilično ekstremno u drugu krajnost (bez ikakve dokumentacije) što ipak treba uzeti s velikom rezervom.

Istina je da ljudi s iskustvom mogu vrlo dobro čitati programski kod i snalaziti se u njemu. Međutim, takav pristup ne ostavlja dovoljno prostora za dijeljenje znanja u timu, za učinkovito održavanje sustava kada ga preuzme programer koji ne razmišlja na isti način ili ako se sustav *seli* na neku drugu tehnološku osnovu (drugi programski jezik i slično).

Dokumentacija je poželjna, ali ne smije biti sama sebi svrha. Čak i projektanti koji oblikuju dijagrame analize poslovanja i izgradnje novog sustava nakon nekog vremena izgube ideju (*nit vodilju*) iz vida. Tako i njima samima može biti problem rekonstruirati znanje kroz postojeću dokumentaciju i pripadajuće dijagrame.

Na temelju ovog malog pregleda o argumentima koji su za tradicionalni pristup ili protiv njega i o razmišljanju koje se suprotstavlja agilnom pristupu, postaju jasni i principi (načela) na kojima se zasniva agilni razvoj.

Formalni prijevod načela agilnog razvoja:[19]

Načela na kojima se zasniva proglas o agilnom razvoju programa

Rukovodimo se sljedećim načelima:

Najvažnije nam je zadovoljstvo korisnika koje postižemo pravovremenom i stalnom isporukom vrijednih programa.

Spremno prihvaćamo promjene zahtjeva, čak i u kasnoj fazi razvoja. Agilni razvoj koristi promjene da korisniku stvori tržišnu prednost.

Redovito isporučujemo upotrebljive programe, u razmacima od nekoliko tjedana do nekoliko mjeseci,nastojeći da to bude što češće.

Poslovni ljudi i informatičari moraju zajedno raditi svakodnevno, tijekom cjelokupnog trajanja projekta.

Projekte ostvarujemo oslanjajući se na motivirane pojedince. Pružamo im okruženje i podršku koja im je potrebna, i prepuštamo im posao s povjerenjem.

Razgovor uživo je najučinkovitiji način prijenosa informacija razvojnom timu i unutar tima.

Upotrebljivi programi su osnovno mjerilo napretka.

Agilni procesi potiču i podržavaju održivi razvoj. Pokrovitelji, informatičari i korisnici moraju biti u stanju stalno raditi, usklađenim tempom, nezavisno od trajanja projekta.

Neprekinuti naglasak na tehničkoj izvrsnosti i dobro oblikovanje pospješuju agilnost.

Jednostavnost – umješnost povećanja količine posla kojeg ne treba raditi – je od suštinske važnosti.

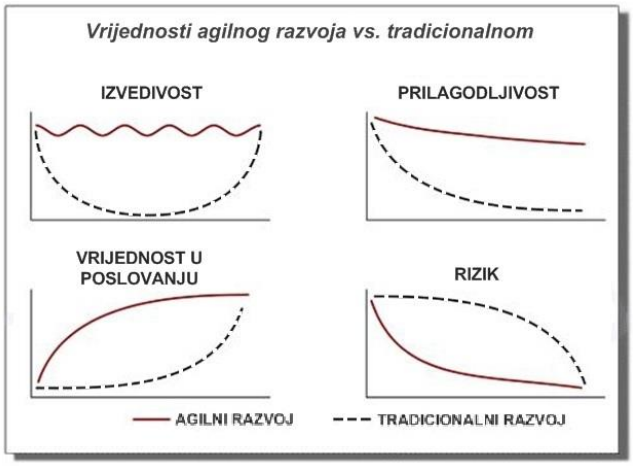
Najbolje arhitekture, projektne zahtjeve i oblikovanje, stvaraju samo–organizirajući timovi.

Timovi u redovitim razmacima razmatraju načine da postanu učinkovitiji, zatim se usklađuju i prilagođavaju svoje ponašanje.

3.3.2. Karakteristike agilnog razvoja

Agilni se razvoj temelji na iterativnom modelu, ali je temeljni pristup razvoju za agilni proces mnogo jednostavniji (lakši) i više orijentiran na ljude, odnosno sudionike u projektu, za razliku od tradicionalnog pristupa. Na slici 3.13. vidi se usporedba vrijednosti agilnog i tradicionalnog pristupa u vremenu (horizontalna os) i u težini promatranog parametra (vertikalna os).

[19] http://agilemanifesto.org/iso/hr/principles.html (26.01.2012.)



Slika 3.13. Usporedba vrijednosti uspjeha agilnog i tradicionalnog pristupa

Za usporedbu može poslužiti i analiza tablice 3.1. koja pruža pregled karakteristika agilnog i tradicionalnog pristupa. Dakle, prema cilju, veličini, korisnicima i ostalome što se navodi u tablici 3.1. za neki konkretni projekt može se donijeti odluka je li u projektiranju bolje pristupiti agilnom ili tradicionalnom razvoju.

Tablica 3.1. Razlike agilne i tradicionalne metode razvoja

	Projekt	
Karakteristike projekta	***Agilni***	***Tradicionalni***
Prvenstveni cilj	Brza korist	Visoka sigurnost
Zahtjevi	Hrpimice se evidentiraju, brzo se mijenjaju, često su nepoznati	Vrlo rano su poznati i većinom stabilni
Veličina	Manji timovi i manji projekti	Veći timovi i veći projekti
Arhitektura	Oblikuje se za trenutne zahtjeve	Oblikuje se za trenutne zahtjeve i one u doglednoj budućnosti
Planiranje i kontrola	Internacionalizacija plana, kontrola kvalitete	Dokumentirani plan, kvantitativna kontrola
Korisnici	Predani, obrazovani, spremni na suradnju, prikupljeni na licu mjesta	Oni koji su potrebni po ugovoru
Informatičari	Agilni, obrazovani, spremni na suradnju, prikupljeni po potrebi	Planski orijentirani, posjeduju vještine potrebne za područje razvoja
Refaktoriranje	Jeftino	Skupo

Unutar agilnog pristupa razvijeno je nekoliko metodologija razvoja, među kojima su: *Extreme Programming* (XP), *Scrum*, *Features Driven Development* (FDD), *Agile Unified Process* (AUP), *Dynamic Systems Development Method* (DSDM), i *Crystal Method*. Iako se ove metodologije zasnivaju na principu agilnog pristupa, svaka ima različite temeljne postavke, procese i zahtjeve razvoja.

U 6. poglavlju detaljnije će biti opisana *Scrum* metodologija agilnog pristupa.

Pitanja za ponavljanje

1. *Koje su osnovne faze razvoja svih modela?*
2. *Objasnite (ukratko) vodopadni (evolucijski, iterativni, prilagodljivi) model razvoja.*
3. *Koje su karakteristike vodopadnog modela razvoja?*
4. *Skicirajte vodopadni model i objasnite njegove dobre i loše strane.*
5. *U kojim situacijama je vodopadni model primjenjiv? U kojima nije?*
6. *Ponoviti pitanja od 3. do 5. za iterativni, evolucijski, spiralni i RAD model?*
7. *Koji od spomenutih modela pružaju pravovremeno prepoznavanje rizika razvoja? Skicirajte usporedbu upravljanja rizicima u vodopadnom i iterativnom modelu.*
8. *Usporedite tradicionalni pristup razvoju i RAD.*
9. *Objasnite što je to unificirani proces. Opišite fazu početak (elaboracija, konstrukcija, tranzicija).*
10. *Objasnite relativni odnos četiriju faza unificiranog procesa u odnosu na vrijeme i resurse.*
11. *Definirajte agilno programsko inženjerstvo.*
12. *Objasnite prednosti agilnog razvoja kroz četiri glavne karakteristike manifesta.*
13. *Koja je razlika između tradicionalnog pristupa razvoju i agilnog razvoja.*
14. *Nabrojite načela agilnog razvoja. Pomoću njih odgovorite na prethodno pitanje.*
15. *Kakvu uspješnost pruža agilni razvoj u odnosu na tradicionalni?*

4. Tehnike modeliranja

Odabir tehnike modeliranja
Tradicionalne tehnike
Unificirani jezik za modeliranje UML

4.1. Odabir tehnike modeliranja

Informacijski sustav koji se razvija prolazi kroz faze razvoja od kojih su analiza i oblikovanje obogaćeni modelima raznih pogleda na sustav. Općenito se može reći da pogledi na sustav koji se razvija uključuju [16]:

- Odabir **tehnika** modeliranja koje će se koristiti za specifikaciju zahtjeva.
- Razumijevanje problema i određivanje **opsega** sustava (te njegova detaljizacija kroz zahtjeve).
- **Provjeru** ispunjava li izrađeni sustav zahtjeve iz specifikacije.

U ovom će poglavlju biti detaljno opisane tehnike modeliranja koje se koriste kako bi se opisao sustav poslovanja. Tehnike modeliranja su potrebne i da se na logičkoj razini oblikuje novi IS.

U ovom će poglavlju bit opisano i kako se (pomoću tehnika modeliranja) nastoji razumjeti problem koji treba riješiti i kako rješenje uobličiti u programski proizvod.

U 9. poglavlju će biti riječi o pristupu testiranju sustava, načinu na koji se provjeravaju zahtjevi i o osiguranju kvalitete programskog proizvoda koji treba implementirati kod korisnika.

Razni parametri odlučuju o odabiru tehnike(a) modeliranja sustava. Zasigurno je važno iskustvo informatičara u korištenju njemu poznatih tehnika. Također je važno kojom se metodologijom koristi poduzeće koje se bavi razvojem IS-a. Pojedine metodologije, kao RUP pristup (vidi poglavlje 5.2.) idu zajedno sa UML jezikom jer su paralelno razvijani i nadopunjuju se.

Za kvalitetno modeliranje složenog sustava nedovoljan je jedan tip modela. Treba više različitih modela prikazanih u različitim tehnikama (jezicima). Već je rečeno da je za cjeloviti prikaz sustava potrebno izraditi modele procesa, podataka i resursa. Neke tehnike koje će biti opisane u nastavku odgovaraju isključivo modeliranju procesa, neke druge samo modeliranju podataka, dok postoje i tehnike kojima se mogu prikazati oba pogleda na sustav i njihovo međudjelovanje. Naravno da je važno pokazati kako procesi djeluju na podatke, te je takav model koristan za sve sudionike tima razvoja.

Često se razlikuju modeli sustava, ovisno o sudioniku kojemu su namijenjeni. Modeli općeg pogleda na sustav su korisni za rukovodeću strukturu, dok su detaljni modeli

koji prikazuju elementarne korake poslovnih procesa važni za suglasnosti s krajnjim korisnikom (operativna razina).

Prvi dio razmatranja tehnika modeliranja bavi se takozvanim tradicionalnim modelima koji su nastali 70-tih godina prošlog stoljeća i bili su podrška tradicionalnom pristupu razvoja IS-a.

U drugom dijelu su opisani neki od modela UML jezika koji je blizak objektno-orijentiranom pristupu razvoja IS-a.

Ipak, važno je naglasiti da, u određenoj mjeri, tehnike predstavljaju neovisne opise pogleda na sustav te se mogu koristiti u bilo kojoj metodologiji ili pristupu razvoju IS-a. Tako, primjerice, informatičari mogu odabrati metodologiju razvoja IS-a koja prati tradicionalni vodopadni model i upotrebljava UML jezik za modeliranje.

4.2. Tradicionalne tehnike

U okviru tradicionalnog pristupa razvoju IS-a bit će prikazani modeli za opis procesa i podataka sustava, poput matričnih dijagrama, lanca događaja i procesa, dijagrama toka podataka, modela entiteti-veze. Osim prikaza simbola i načina na koji se koriste kako bi se opisao dio sustava, bit će prikazano i nekoliko jednostavnih primjera kojima je ilustrirano korištenje pojedinog modela.

Grafika, to jest simboli tehnika koje se koriste u izradi dijagrama treba biti jednostavna radi komfornog komuniciranja s ostalim sudionicima, ljudskih ograničenja u usvajanju i snalaženja u kompliciranim prikazima.

Osnovni koncepti (simboli) pojedine tehnike nisu jedinstveni, razne literature i alati u koje su ugrađene mogućnosti tih tehnika često variraju s oznakama simbola. Ipak, najvažnije je da informacije i znanje koje treba prikazati nekom tehnikom, te model koji apstraktno opisuje to znanje budu ispravni. Nadalje, važno je da se o sadržaju modela slože svi sudionici kojima je on namijenjen.

4.2.1. Matrični dijagrami

Različite matrice veza kojima se prikazuju veze između raznovrsnih ili istovrsnih elemenata sustava izrađuju se u različitim fazama razvoja IS-a. Koriste se za provjeru cjelovitosti, ispravnosti i logike modela informacijskog sustava s obzirom na elemente od kojih je sastavljena.

Matrice veza sastoje se od niza redaka i stupaca u čijem presjeku je zapisana oznaka postojanja ili nepostojanja veze između elemenata matrice. Osim toga važno je dodatno naznačiti i prirodu veze.

Neka pomagala za projektiranje i razvoj programske podrške (CASE pomagala) podržavaju automatsku izradu raznih oblika matrica veza iz elemenata pohranjenih na računalu, jer je izrada matrice veza često dugačak i mukotrpan posao. Matrica može biti jako velika ukoliko opisuje složeni sustav a elementi matrice su detaljno prepoznati i popisani.

Matrice veza mogu biti korisne u svim fazama razvoja IS-a, ovisno o tome koji su elementi sustava povezani matricom i koji je cilj povezivanja tih elemenata.

U početku, tijekom **planiranja** projekta mogu se pomoću matrice veza povezati funkcionalna područja poslovanja (odjeli/cjeline unutar poduzeća) i postojeća programska rješenja koja poduzeće upotrebljava. U presjeku se mogu naznačiti kategorije koje pokazuju ispunjava li pojedino programsko rješenje očekivanja odjela i rješava li sve njihove probleme. Kako je uprava poduzeća odlučila da treba pristupiti razvoju novog IS-a, vrlo je vjerojatno da postojeća rješenja ne odgovaraju većini poslovnih potreba. Spomenuta matrica veza bi trebala pokazati gdje su slabe točke i tako pomoći u definiranju opsega novog projekta.

U **analizi** poslovanja matrica veza može poslužiti: (1) u početku analize za prepoznavanje osnovnih funkcionalnosti sustava i njihovu međusobnu povezanost, te (2) za prikaz detaljnih elemenata nekog dijela (ili cijelog) poslovanja.

Slijedeća dva primjera pokazuju prvi i drugi slučaj.

PRIMJER: Na slici 4.1. pokazane su četiri matrice veza za poslovnu funkciju obračuna plaća zaposlenicima poduzeća. Od matrice do matrice jasno se vidi da elementi sustava mogu biti u međudjelovanju na različite načine.

Primjer veze funkcije / odjeli

	Kadrovska služba	Financijska služba
Obračun plaća	Izrađuje platnu listu	
Isplata plaća		Izrađuje nalog za plaćanje

Primjer veze odjeli / dokumenti

	Platna lista	Nalog za plaćanje
Kadrovska služba	stvara	
Financijska služba	koristi	stvara

Primjer veze funkcije / rukovoditelji

	Anita	Stipe
Obračun plaća	ogovorna	
Isplata plaća		odgovoran

Primjer veze funkcije / dokumenti

	Platna lista	**Nalog za plaćanje**
Obračun plaća	**upisuje**	
Isplata plaća	**čita**	**upisuje**

Slika 4.1. Primjeri matrice veza za različite elemente sustava

***PRIMJER*:** U tablici 4.1. pokazana je matrica veza sa detaljnim popisom elemenata sustava studiranja. To su procesi poslovanja i entiteti koji predstavljaju skup podataka koje odgovarajući proces obrađuje. Može se primijetiti da prvih nekoliko procesa ne čini direktnu funkcionalnost sustava, nego su to događaji koji prethode upisu na neko visoko učilište.

Tablica 4.1. Primjer matrice veza za sustav studiranja

PROCESI \ ENTITETI	Natječaj	Dokumentacija za upis	Prijemni ispit	Index	Potvrda o upisu	Raspored sati	Evidencija održane nastave	Raspored polaganja ispita	Prijavnica	Prijava diplomskog rada	Diplomski rad	Potvrda o stručnoj spremi	Diploma
Objava natječaja	C												
Prikupljanje dokumentacije	R	C											
Predaja dokumentacije u referadu		R											
Polaganje prijemnog ispita		R	C										
Objava rezultata prijemog ispista			R										
Upis na prvu godinu			R	C									
Izdavanje potvrde o upisu			R		C								
Preuzimanje indeksa				R									
Izrada rasporeda sati						C							
Evidentiranje održane nastave				R		R	C						
Predaja izvješća o održanoj nastavi							R						
Potpisivanje indeksa				U			R						
Izrada rasporeda polaganja ispita								C					
Prijava ispita				R			R	R	C				
Odjava ispita									U				
Polagane ispita									U				
Upis ocjene				U					U				
Prijava izrade diplomskog rada				R						C			
Izrada diplomskog rada				R							C		
Obrana diplomskog rada				R					R	R	U		
Izdavanje potvrde o stručnoj spremi											R	C	
Uručivanje diplome											R		C

Matrica veza može prikazivati i povezanost istovrsnih elemenata sustava. Matrica primjera u nastavku povezuje poslovne podsustave i prikazuje integriranost podsustava poduzeća dokumentima koji *kolaju* unutar poduzeća, a isto tako i dokumentima kojima je poduzeće povezano s okolinom.

***PRIMJER*:** U tablici 4.2. prikazana je matrica strukture sustava s podsustavima kao elementima za koje se razmjenom dokumenata analizira međudjelovanje. . Podsustavi su: VP-veleprodaja, MP-maloprodaja, RIF-računovodstvo i financije i Mark-marketing.

Tehnika matričnih dijagrama ovdje ima varijantu koja unutar same matrice dodaje i podmatrice veze sustava s okruženjem. Tako se, osim analize međudjelovanja podsustava, ova matrica može primjenjivati i za definiranje opsega projekta. Iz ovako strukturirane matrice moguće je iščitati koji dokumenti ulaze u sustav iz okruženja i koje dokumente sustav generira i daje u okruženje.

Tablica 4.2. Primjer matrice strukture sustava i djelovanja s okolinom

	O	VP	MP	RIF	Mark
O		narudžbenica VP račun	MP račun	nalog za plaćanje PDV obrazac financijska izvješća	reklame oglasi
VP	otpremnica	/	međuskl. u MP	VP račun	prodajne akcije
MP		međuskl. u VP	/	MP račun	
RIF	izvod banke			/	
Mark					/

I u **oblikovanju** novog sustava matrica veza može pomoći u modeliranju optimalnog programskog rješenja. Primjerice, ako su osnovni elementi matrice veza podsustavi, onda optimalno strukturiran IS treba imati niz zaokruženih i međusobno relativno neovisnih podsustava. U tom će slučaju procesi koji su sadržani u pojedinom podsustavu biti skup zaokruženih i cjelovitih segmenta koji onda omogućava bolju funkcionalnost i efikasnost poslovnog sustava, bolju kontrolu i upravljanje, ali i podržava informatizaciju (izradu i implementaciju) korak po korak, modul po modul.

4.2.2. Lanac događaja i procesa

Lanac događaja i procesa (eng. *Event-driven Process Chain*, EPC[20]) je vrsta dijagrama toka kojim se opisuju poslovni procesi. Koristi se u mnogim poduzećima za modeliranje, analizu i redizajn poslovnih procesa. Upotrebljava se i prijevod *procesni lanac upravljan događajima*.

EPC dijagram prikazuje tijek poslovnih procesa u ovisnosti o događajima koji te procese pokreću. Tako se EPC dijagrami koriste grafičkim simbolima (tablica 4.3.) za prikaz strukture kontrole tijeka poslovnih procesa kao lanca događaja i funkcija poslovnog sustava.

Osnovna četiri tipa objekata koji se koriste u EPC modelu su procesi (funkcije), događaji, pravila i resursi.

Pri upotrebi EPC dijagrama treba poštivati sljedeća pravila:

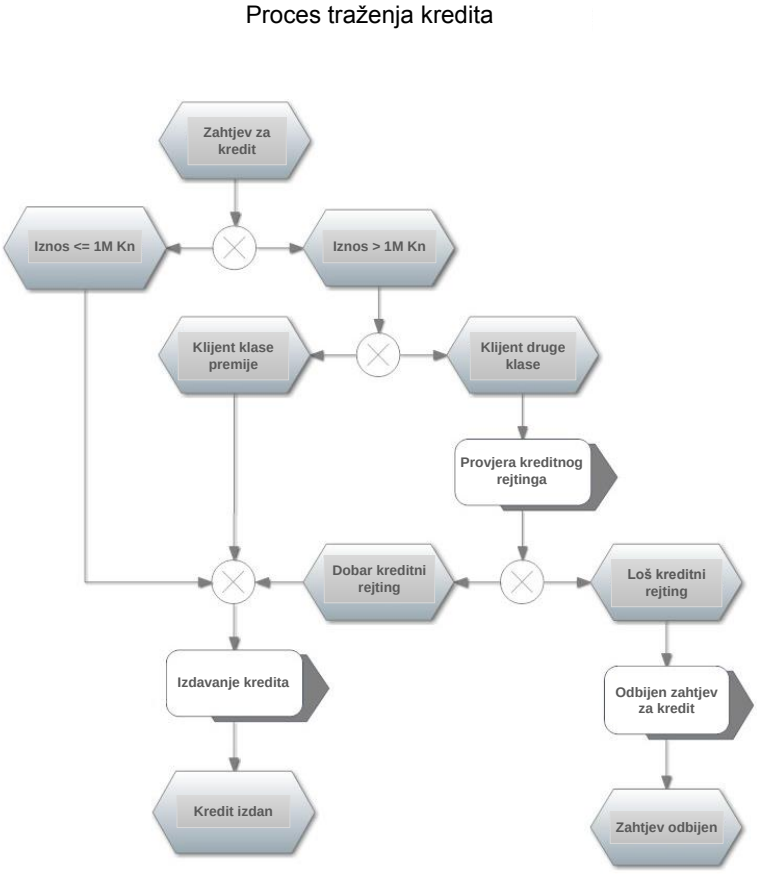
- svaki model mora imati najmanje jedan početni i jedan završni događaj;
- funkcije i događaji su uvijek naizmjenični;
- funkcije se nikad ne bi trebale spajati s funkcijama, a događaji s događajima;
- događaj i funkcija se ne mogu međusobno spojiti u oba smjera;
- organizacijska jedinica se ne može spojiti s događajem.

[20] EPC metoda je razvijena u okviru ARIS alata. Razvio ju je prof. Wilhelm-August Scheer na Ekonomskom institutu računalnih znanosti na Sveučilištu Saarland u ranim 90-im prošlog stoljeća.

Tablica 4.3. Simboli dijagrama EPC

Proces		Promjena ulaza i izlaze
Događaj		Postizanje određenog stanja
Kontrola toka		Privremena logička veza između procesa i događaja
Operatori	AND exclusive-OR inclusive-OR	Logička veza između procesa i događaja
Naglašeni proces		Veza sa prethodnikom ili nasljednikom procesa *

***PRIMJER*:** Na slici 4.2. pokazan je primjer lanca događaja i procesa za traženje kredita, koji ima naznačene događaje što uvjetuju pokretanje nekog od procesa ovog lanca. Ako čitamo krajnju lijevu granu ovog dijagrama možemo postaviti uvjete (događaje) koji moraju biti ispunjeni kako bi klijent dobio kredit: ako je iznos veći od 1M Kn te ako je običan klijent (*Klijent druge klase*) i ima loš rejting, zahtjev za kredit bit će mu odbijen. Konačni događaj koji je izlaz iz ovog lanca je *Zahtjev odbijen*.



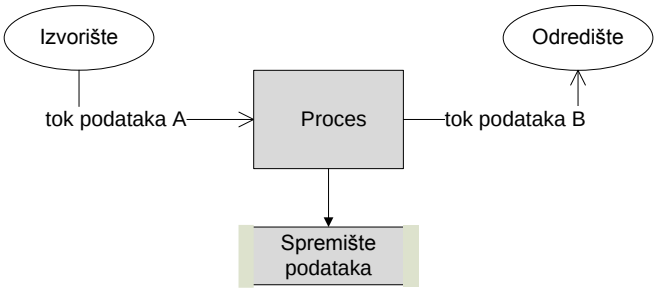
Slika 4.2. Lanac događaja i procesa za traženje kredita

4.2.3. Dijagram toka podataka

Jedna od poznatijih strukturnih tehnika za prikaz procesa sustava je dijagram toka podataka (DTP) (eng. *Data Flow Diagram – DFD*). Cilj je uočiti jedan ili nekoliko ulaza kojima se koristi jedan određen proces ili skup procesa, te uočiti jedan ili nekoliko izlaza iz tog procesa.

Osnovni simboli (koncepti) dijagrama toka podataka su (slika 4.3.):

- procesi sustava (programa, aktivnosti, podsustavi) koji transformiraju ulazne tokove u izlazne tokove podataka;
- ulazni i izlazni tokovi podataka (zaslona, dokumenata) koje sustav dobiva ili daje u okruženje;
- vanjski objekti - izvorišta i odredišta (organizacije, ljudi, drugi unutarnji ili vanjski sustavi) koji šalju prema ili primaju tokove podataka od sustava;
- spremišta podataka (baze podataka, datoteke) u kojima se čuvaju podaci potrebni za izvršenje procesa ili dobiveni kao rezultat rada procesa.



Slika 4.3. Dijagram toka podataka općeg procesa

Dijagram toka podataka sa slike 4.3. čita se na slijedeći način:

> *Tok podataka A* dolazi iz *Izvorišta* podataka i odvijanjem *Procesa* biva transformiran u *tok podataka B*. Da se transformacija efikasno izvede upisuju se i koriste podaci iz *Spremišta podataka*. Izlazni *tok podataka B* iz *Procesa* odlazi prema *Odredištu*.

Vanjski objekt je simbol koji predstavlja sustave koji su u vezi s promatranim sustavom, bilo da su izvori ili odredišta podataka. Ime vanjskog objekta je obično imenica, odnosno naziv sustava koji vanjski objekt predstavlja.

Tok podataka je putanja kojom teku informacije poznate strukture i sadržaja. Podaci se kreću od jednog dijela sustava prema drugom i pri tome bivaju obrađeni kroz jedan ili više procesa. Imena tokova podataka biraju se tako da jasno opisuju podatke i poznate elemente tih podataka. Nazivi tokova podataka su imenice, najčešće u jednini, ili kombinacije imenica i pridjeva.

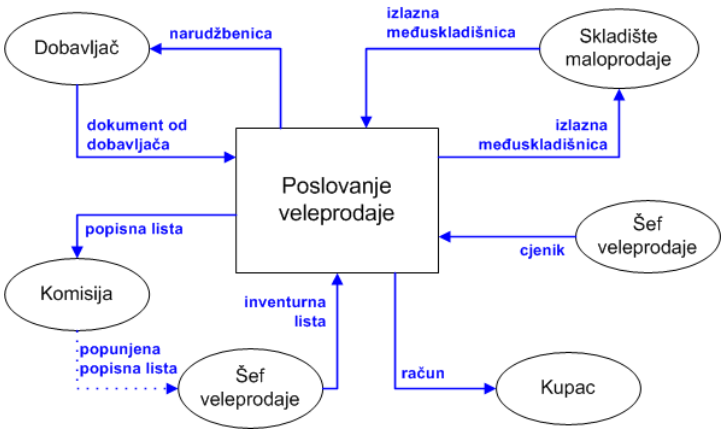
Spremište podataka je memorijsko mjesto gdje se nalaze podaci pohranjeni nekim procesom, a njima se dalje koriste neki drugi procesi u svojoj obradi, odnosno podaci preneseni tokom podataka.

Proces transformira ulazne tokove podataka u izlazne tokove podataka. Naziv procesa treba biti kratak opis značenja procesa (nije potrebno precizno specificirati njegovu unutarnju logiku). Za ime se bira glagol ili skup riječi koji opisuje vrstu posla. Proces se može imenovati i prema ulaznom ili izlaznom toku podataka, no češće se imenuje prema izlaznom toku, jer je izlazni tijek podataka cilj procesa.

Takozvani **kontekst dijagram** (ili dijagram konteksta) je dijagram toka podataka cijelog sustava na najvišoj razini promatranja. To je dijagram nulte razine koji sadrži samo jedan proces koji smatramo cjelinom, te sadrži ulaze i izlaze tog procesa.

Dijagram konteksta prikazuje granicu između sustava i okoline i tako definira poslovno područje koje se analizira, a tokovi podataka prikazuju veze sustava s okolinom.

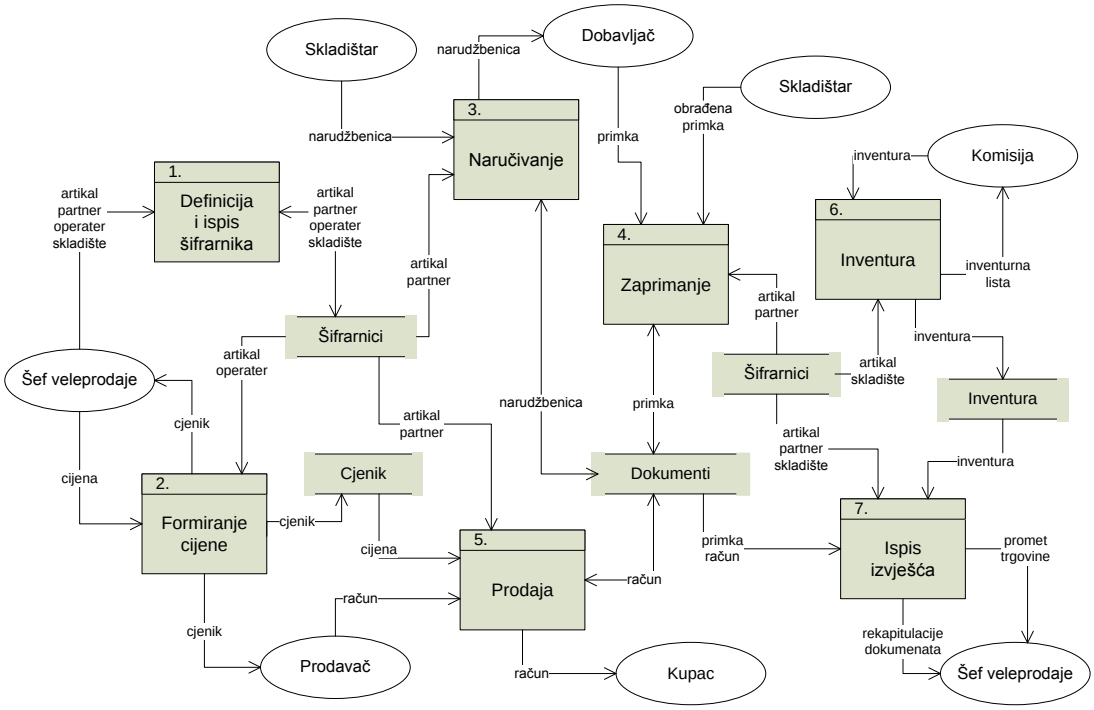
Na dijagramu konteksta prikazuju se osnovna imena izvorišta i odredišta, te osnovni dokumenti koji ulaze u sustav ili izlaze iz sustava (Slika 4.4.), ali se na njemu u pravilu ne prikazuju spremišta podataka i dokumenata.



Slika 4.4. Dijagram konteksta poslovanja veleprodaje

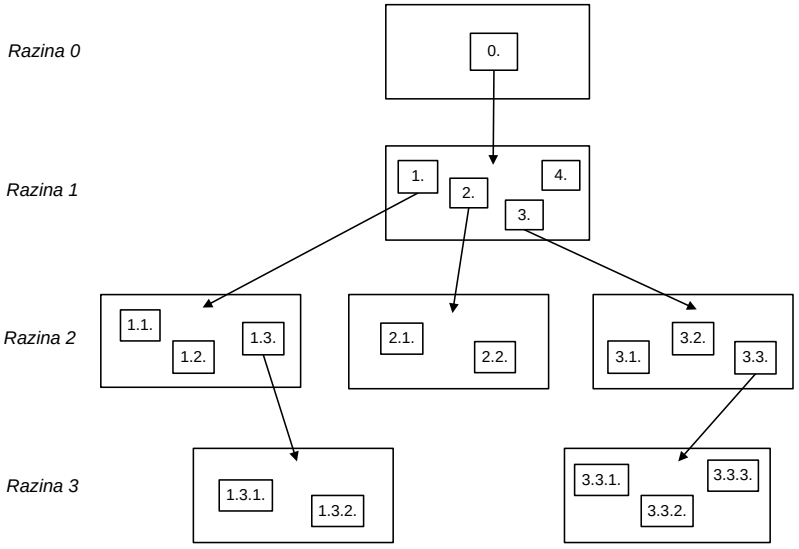
Dekompozicijom se proces u kontekst dijagramu raščlanjuje na više potprocesa razine 1, koji se svi nalaze na sljedećem dijagramu toka podataka (slika 4.5.).

Daljnjim raščlanjivanjem svakog od potprocesa formiraju se procesi razine 2. Međutim, njih se ne pokazuje na jednom dijagramu, nego se za svaki proces razine 1 radi poseban dijagram toka podataka razine 2. Taj postupak raščlanjivanja se može ponavljati dokle god postoji potreba da se detaljnije opiše pojedini proces.



Slika 4.5. Dijagram toka podataka (razina 1) poslovanja veleprodaje

Složeni procesi, koji se ne mogu prikazati na jednom dijagramu toka podataka, dijele se u potprocese (slika 4.6.). Tako se stvara hijerarhija, odnosno dekompozicija procesa. Osim procesa, dekomponirati se mogu: ulazni/izlazni tokovi podataka, izvorišta/odredišta podataka i spremišta podataka. Oni se dekomponiraju paralelno s procesima.



Slika 4.6. Model hijerarhije DTP-ova

S obzirom na mogućnost dekompozicije DTP-a procesi se mogu klasificirati prema složenosti s obzirom na položaj u modelu dekompozicije:

- proces dijagrama konteksta;
- procesi srednje razine - procesi na dijagramu toka podataka između konteksta i elementarnih procesa;
- primitivni (elementarni) procesi, koji na dijagramu toka podataka nemaju svoje potprocese (ne mogu se dalje dekomponirati).

Kako bi dijagram toka podataka prikazivao ispravan logički slijed procesa u sustavu potrebno ga je povezati s dijagramom dekompozicije (raščlanjenja) poslovnih procesa koji se često izrađuje pri analizi sustava.

4.2.4. Model entiteti-veze

Modeliranje podataka je proces koji počinje utvrđivanjem i analiziranjem potreba korisnika za informacijama, a završava izgradnjom stabilne, ali prilagodljive baze podataka.

Svaki model podataka sastoji se od strukture (skup entiteta i njihovih svojstava: atributa i veza), ograničenja (dopuštena i/ili zabranjena stanja) i operatora (omogućuju interpretaciju dinamičkih karakteristika). Struktura i ograničenja predstavljaju statička svojstva promatranog sustava, a operatori omogućuju izmjenu stanja podataka u skladu s promjenom stanja u sustavu.

Struktura modela podataka gradi se od temeljnih koncepata: **entitet**, **veza** i **atribut** (nastalih generalizacijom temeljnih pojmova promatranog sustava):

- **Entitet** je stvarni ili apstraktni predmet ili događaj o kojemu se u informacijskom sustavu pamte podaci. To je neka posebnost, suština, bitnost koja postoji u sustavu i jasno se razlikuje od drugih entiteta (događaja, činjenica).
- **Veza** (eng. *relationship*) povezuje entitete. Ona predstavlja neku interakciju među entitetima u sustavu, odnosno predstavlja znanje o njihovoj povezanosti. Veza je ovisna o entitetima sustava, ne može postojati sama za sebe, a predstavlja bilo koje znanje o vrsti povezanosti između dva ili više entiteta.
- **Atribut** (osobina entiteta) je neko kvalitativno ili kvantitativno svojstvo entiteta. Pridjeljujući pojedinom entitetu njegove pripadajuće atribute definira se entitet preko njegovih odgovarajućih obilježja.

Ograničenja koja se postavljaju nad koncepte strukture modela podataka su:

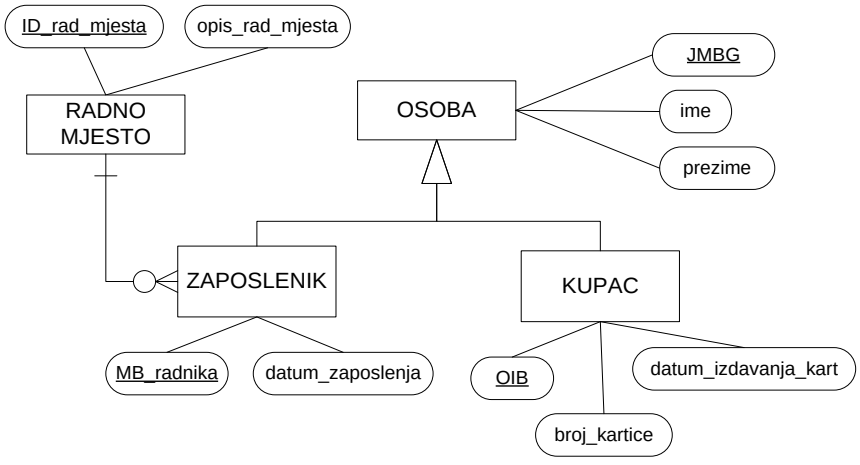
- Ograničenje na **vezu** između dvaju entiteta znači da jedan entitet jednog tipa može biti povezan s nijednim, jednim ili više entiteta drugog tipa i obrnuto.

- Ograničenje na **vrijednost** atributa entiteta znači da vrijednost atributa mora biti određena prema zadanim pravilima i u postavljenim granicama.
- Ograničenja na **domenu** znači da svi atributi koji su definirani nad tom domenom mogu imati samo vrijednosti zadane tom domenom.
- Ograničenje na **pridruživanje** atributa entitetu znači da promatrani entitet može imati nijedan, jedan ili više vrijednosti tog atributa.

Model entiteti – veze (E-V model, eng. *Entity-Relationship*, E-R ili) je model koji je se izrađuje logičkim modeliranjem podataka i ne ovisi niti o računalnoj opremi, niti o alatu (programskoj podršci) za upravljanje bazom podataka. Često se koristi i skraćenica ERA (eng. *Entity Relationship Attribute*) kako bi se naglasilo da je riječ o modelu koji predstavlja grafički prikaz znanja o objektima, vezama i svojstvima podataka.

***PRIMJER*:** Na slici 4.7. pokazan je jednostavni primjer nekoliko entiteta koji sudjeluju u općem procesu prodaje. U ovom primjeru su prikazane dvije vrste veza:

- Između zaposlenika i radnog mjesta gdje je naznačeno da na jednom radnom mjestu (misli se na vrstu, primjerice prodavač, skladištar, blagajnik, ne na fizičko radno mjesto) može biti nijedan, jedan ili više zaposlenika, dok jedan zaposlenik radi samo na jednom radnom mjestu.
- Između entiteta osobe koja je nadređena i dvaju entiteta (zaposlenik i kupac) koji su podređeni u ovom hijerarhijskom prikazu. Cilj ovakve vrste veze je izdvojiti atribute koji su zajednički u nadređeni entitet, a one koji opisuju svaki entitet posebno ostaviti unutar tog entiteta. Ova vrsta veze se zove generalizacija.



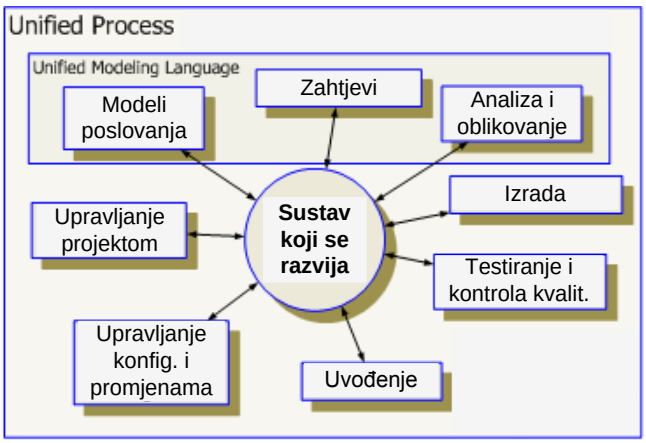
Slika 4.7. E-V model sudionika u procesu prodaje

4.3. Unificirani jezik za modeliranje UML

Modeliranje omogućava vizualizaciju problema, rukovanje i upravljanje složenim strukturama, te jednostavniju komunikaciju među sudionicima. UML (eng. *Unified Modeling Language*) je jezik za vizualnu specifikaciju, izradu i dokumentiranje elemenata sustava. Može se koristiti u svim fazama razvoja IS-a i u raznim metodologijama razvoja.

UML je odobren (OMG[21]) i standardiziran jezik za razvoj programske potpore i zajedno s RUP pristupom (detaljnije u poglavlju 5.2.) čini cjelinu metodologije razvoja objektno-orijentiranim pristupom.

Metoda razvoja korištenjem unificiranog procesa (slika 4.8.) najčešće je podržana UML-om, koji pruža mogućnost modeliranja kroz cijeli postupak projektiranja novog sustava.



Slika 4.8. UML u metodi unificiranog procesa[22]

Ovdje su navedene karakteristike i smisao nekoliko dijagrama UML-a, uglavnom onih koji su potrebni za analizu i oblikovanju funkcionalnosti IS-a. To su:

- Dijagram slučajeva korištenja
- Dijagram klasa
- Dijagrami slijeda
- Dijagram aktivnosti
- Dijagram paketa.

[21] Object Management Group – konzorcij koji se bavi standardima modeliranja (programa, sustava i poslovnih procesa)
[22] http://www.xops-online.com/content/development-process.html (25.01.2012.)

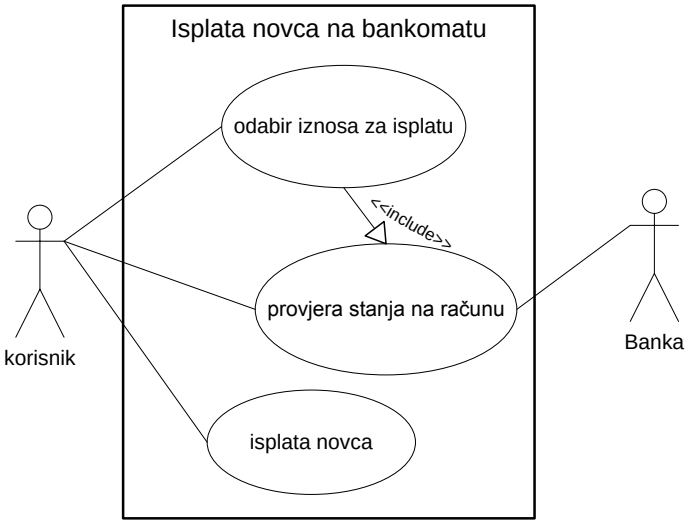
4.3.1. Dijagram slučajeva korištenja

Slučaj korištenja (eng. *Use Case*) predstavlja jednostavni prikaz veze poslovnog sustava i vanjskih elemenata (vanjskog svijeta, okoline). Stoga se dijagrami slučajeva korištenja crtaju prije svih ostalih dijagrama i osnova su za ostale dijagrame UML-a. Slučajeve korištenja treba pisati jezikom korisnika, izbjegavajući stručne (informatičke) izraze i detalje povezane s implementacijom.

Slučaj korištenja je pogodan za prikupljanje zahtjeva, ali isto tako i za cijeli ciklus razvoja softvera. Slučaj korištenja je tako jednostavan da predstavlja jezgru budućeg IS-a, a s druge strane tako moćan da podupire cijelu metodologiju razvoja IS-a.

Dijagram slučajeva korištenja (slika 4.9.) se sastoji od nekoliko osnovnih elemenata:

- **slučaja korištenja** kao skupa scenarija povezanih jednim ciljem (zahtjevom) korisnika;
- **sudionika** (vanjski entitet direktno povezan sa sustavom) koji je direktni pokretač neke aktivnosti sustava ili se na njega direktno odražava neka aktivnost sustava;
- **veza** između sudionika i slučaja korištenja, između dva slučaja korištenja ili između dva sudionika.



Slika 4.9. Dijagram slučajeva korištenja za funkcionalnost bankomata

Za svaki pojedini slučaj korištenja treba napraviti i pripadajući obrazac. U obrascu slučaja korištenja, između ostalog, opisuje se scenarij, tj. tijek događaja (koraka) od kojih se sastoji određeni slučaj korištenja (tablica 4.4.).

Opis svakog koraka mora biti jednostavan i jasno pokazivati tko je zadužen za njegovo izvršavanje. Iako su većina UML elemenata dijagrami, scenarij slučaja korištenja predstavlja stranicu-dvije teksta koji je prikazan odgovarajućom oznakom u dijagramu.

Tablica 4.4. Scenarij slučaja korištenja za odabir iznosa za isplatu na bankomatu

	Slučaj korištenja: odabir iznosa za isplatu
Autor:	KK
Datum:	17.01. 2012
Opis:	Nakon što je korisnik odabrao opciju isplate novca sustav traži upis iznosa za isplatu
Sudionici:	Korisnik, Sustav
Okidači:	-
Preduvjeti:	Korisnik posjeduje PIN
Osnovni scenarij:	1. Sustav na ekran ispisuje unaprijed zadane vrijednosti. 2. Korisnik odabire jednu od vrijednosti. 3. Sustav provjerava iznos u odnosu na: a) stanje novca u pričuvi bankomata b) stanje novca na računu. 4. Sustav šalje odgovarajuću poruku.
Izuzetci:	2. a) Ako nije ponuđena vrijednost koju korisnik želi, odabire opciju „druga vrijednost“. 2. b) Sustav na ekranu nudi polje za unos proizvoljne vrijednosti. 2. c) Korisnik unosi ručno proizvoljan iznos.

Scenarij je opis događaja kome je cilj opisati korake kroz koje prolazi sudionik i sustav. Sudionik uvijek napravi prvi korak, a onda sustav odgovara na njega. Ovaj povrat i nastavak međudjelovanja sudionika i sustava dok se ne postigne cilj predstavlja jedan scenarij.

4.3.2. Dijagram klasa

Dijagram klasa (eng. *Class Diagram*) opisuje tipove objekta u sustavu i različite vrste statičkih veza koje postoje između njih. Dijagram klasa također prikazuje svojstva i operacije klasa, kao i ograničenja načina povezivanja objekata. Klasa je kolekcija objekata sa zajedničkom strukturom, zajedničkim ponašanjem, zajedničkim vezama i zajedničkom semantikom.

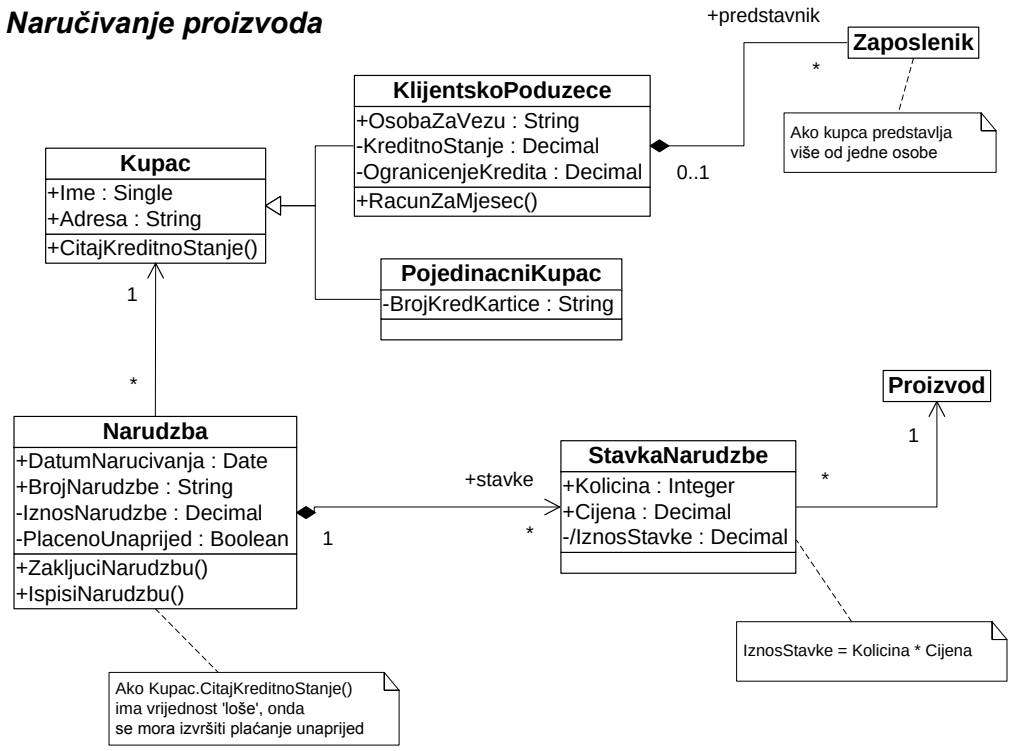
Tijekom procesa razvoja programske potpore, dijagrami klasa se detaljiziraju od konceptualnih dijagrama do dijagrama dovoljnih za izradu koda. Konceptualni dijagrami su korisni u analizi i oblikovanju poslovne tehnologije. Stoga u toj fazi razvoja IS-a treba izbjegavati pojmove iz područja računarstva i koristiti se jednostavnom notacijom.

Svojstva (eng. *properties*) predstavljaju strukturne karakteristike klase. Svojstvo je jedan pojam, a u prikazu dijagrama klasa pojavljuje se u dva različita oblika:

- atribut (eng. *attribute*) je svojstvo kojim se opisuju detalji klase;
- asocijacija (eng. *association*) se prikazuje kao veza među klasama.

U dijagramu klasa (slika 4.11.) postoje tri vrste relacija između klasa:

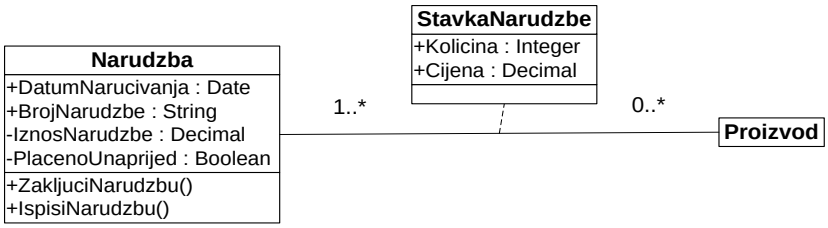
- **Asocijacija** je najopćenitiji oblik relacije, opisuje vezu među klasama čiji su objekti na neki način strukturno povezani (na slici 4.11. veza između kupca i narudžbe). Kardinalnost (eng. *multiplicity*) pokazuje na koliko objekata se odnosi neko svojstvo.
- **Kompozicija** je relacija u kojoj jednoj klasi pripada druga klasa, koja sadrži najmanje jedan objekt druge klase. Označava se rombom na liniji na strani nadređene klase (na slici 4.11. veza između klijentskog poduzeća i njihovih zaposlenika).
- **Generalizacija** je relacija kada je jedna klasa podklasa nadređene klase. Prikazuje se trokutastom strelicom prema nadklasi (na slici 4.11. kupac je nadklasa za klase klijentsko poduzeće i pojedinačni kupac).



Slika 4.11. Dijagram klasa za proces naručivanja proizvoda

Klasa asocijacija (eng. *association classes*) omogućava da se asocijaciji dodaju atributi, operacije i druge karakteristike. Klasa asocijacija može biti korisna kod nadopunjavanja asocijacije koja s obje strane ima višestruki kardinalitet.

Odnos između dokumenta narudžbe i proizvoda koji se naručuju može biti prikazan kao u već navedenom primjeru dijagrama klasa *Naručivanje proizvoda*. Taj odnos se može prikazati i na drukčiji način, pomoću klase asocijacija kako je pokazano na slici 4.12.



Slika 4.12. Primjer klase asocijacija

Ako se na jednoj narudžbi može u samo jednoj stavci nalaziti jedan konkretan proizvod, onda se može primijeniti i klasa asocijacija za stavku narudžbe. Međutim, ako se na jednoj narudžbi može više puta pojaviti stavka koja ima isti proizvod (nije isključeno u praksi), onda je klasa asocijacija neprimjerena te je bolje upotrijebiti novu klasu, kako je i pokazano na dijagramu *Naručivanje proizvoda***.**

Od svih UML dijagrama, dijagram klasa je najbogatiji simbolima i njihovim značenjem, pa je problem to mnoštvo elemenata savladati i ispravno upotrebljavati. Može se dati nekoliko preporuka (po pravilu *manje je više*) za crtanje preglednog dijagrama klasa:

- Ne treba se truditi koristiti se svim elementima koji su dostupni. Dovoljno je započeti od jednostavnih elemenata kao što su klasa, asocijacija, atribut, generalizacija.
- Konceptualni dijagrami su korisni u analizi i dizajnu poslovne tehnologije. Stoga u toj fazi razvoja IS-a treba izbjegavati softverske pojmove i upotrebljavati jednostavnu notaciju.
- Nije potrebno crtati model za cijelo područje koje se analizira. Bolje je imati nekoliko dijagrama i na svakom od njih pokazati neko područje koje u tom trenutku detaljno analiziramo. Poveznice među dijagramima mogu biti elementi koji su samo konceptualno navedeni (pravokutnik samo sa imenom klase).

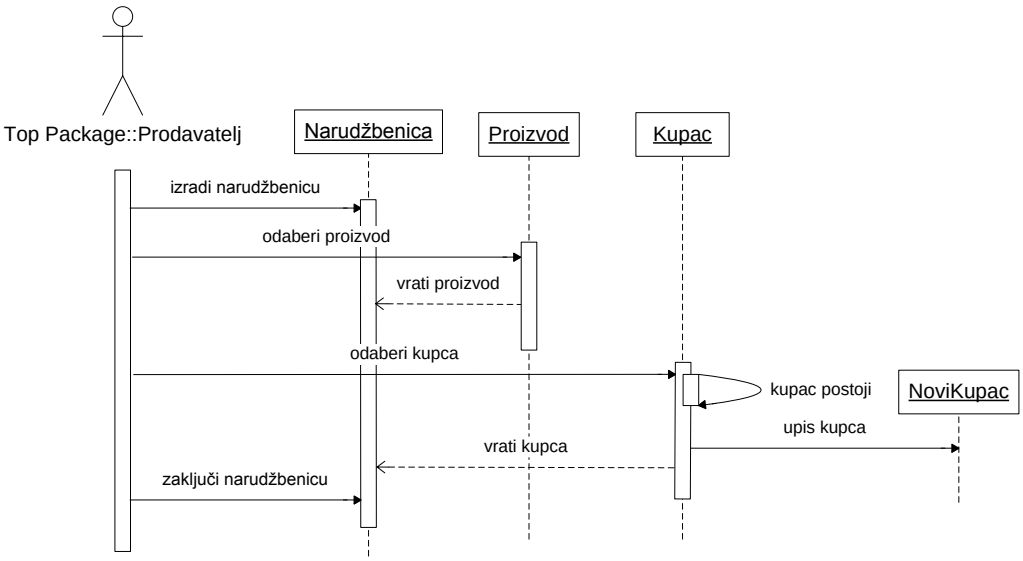
4.3.3. Dijagram slijeda

Za opis rada sustava treba prikazati objekte i njihovo međudjelovanje. UML dijagram slijeda (eng. *Sequence Diagram*) prikazuje vremensku dinamiku međudjelovanja. Međudjelovanje je ponašanje koje sadrži skup poruka koje se razmjenjuju između skupa objekata u nekom kontekstu s nekim ciljem. Poruka je specifikacija komunikacije između objekata koja prenosi informaciju. Prijem poruke izaziva akciju to jest izvršenje naredbe.

Dijagrami slijeda eksplicitno pokazuju slijed aktivnosti koje utječu na sustav da se izvrši željena operacija i dobije rezultat (slika 4.10.). Koristi se za aplikacije u realnom vremenu i za složene scenarije u kojima vrijeme igra važnu ulogu.

Ako su prethodno izrađeni dijagrami slučajeva korištenja, onda su dijagrami slijeda jedna od njihovih realizacija jer pokazuju redoslijed događaja i operacija iz pripadajućeg obrasca. Redoslijed događaja i operacija prikazan je u dvije dimenzije:

- dimenzija koja predstavlja vrijeme (vertikalna os);
- dimenzija koja predstavlja različite objekte koji sudjeluju u događanjima (horizontalna os).



Slika 4.10. Dijagram slijeda za slučaj korištenja izrade narudžbenice

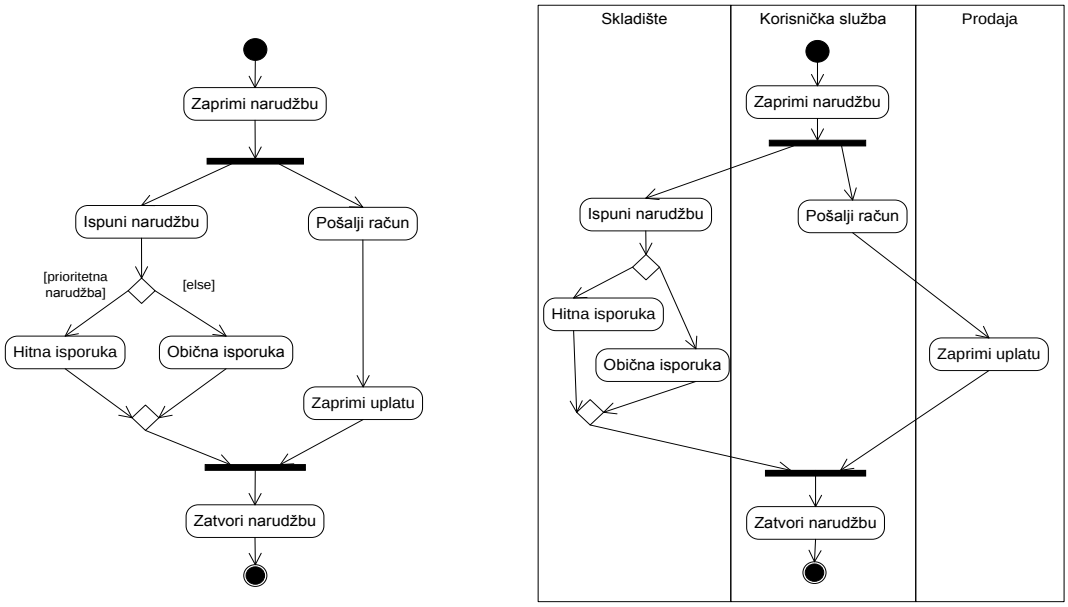
4.3.4. Dijagram aktivnosti

Dijagram aktivnosti (eng. *Activity Diagram*) služi za opisivanje logike procedura poslovnih postupaka i toka rada. Prikazuje tok rada (eng. *workflow*) između objekta nekog slučaja korištenja ili tok upravljanja za neku operaciju. Njegova uloga je slična ulozi dijagrama toka, tj. radnog dijagrama (eng. *flowchart*), ali je osnovna razlika u tome što se na dijagramu aktivnosti mogu pokazati istovremenosti procesa.

S obzirom na slučajeve korištenja, dijagramom aktivnosti može se dobro pokazati opći slijed akcija za nekoliko objekata i nekoliko slučajeva korištenja. U ovom dijagramu nije bitno dodijeliti aktivnosti pojedinim objektima (sudionici, dokumenti, ...), nego dobiti opću sliku ponašanja sustava i odrediti međuzavisnosti u ponašanju. Aktivnosti se vezuju s objektima u kasnijim fazama razvoja.

Dijagramom aktivnosti modelira se tijek posla (poslovni procesi), kako bi se bolje shvatio poslovni proces i omogućile izmjene i poboljšanja (lijeva strana slike 4.13.). kako bi modeliranje bilo uspješno treba uključiti osobe (korisnika) koje su upućene u poslovni proces. Kod opisivanja složenih sekvencijalnih algoritama, dijagram aktivnosti se koristi kao klasični radni dijagram.

Velika prednost dijagrama aktivnosti je u tome što podržavaju paralelno izvršavanje. Zato su oni odlično sredstvo za modeliranje tokova rada i procesa.



Slika 4.13. Dijagram aktivnosti za naručivanje artikala od kupca

U dijagramu aktivnosti može se prikazati i tko što radi, tako da se uvedu staze (eng. *swim lane*). Dijagram aktivnosti se proširuje prikazom po stazama ako je, osim prikaza što se događa s aktivnostima, potrebno prikazati i izvršioce pojedinih aktivnosti.

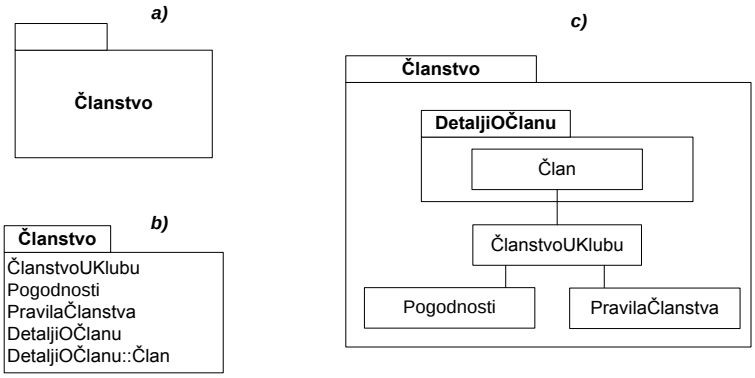
Na desnoj strani slike 4.13. pokazan je primjer dijagrama aktivnosti naručivanja artikala od kupaca podijeljen po organizacijskim jedinicama.

4.3.5. Dijagram paketa

UML dijagram paketa (eng. *package diagram*) služi za logičko grupiranje. To ne znači da ono što je grupirano u pakete predstavlja cjelinu u fizičkoj realizaciji. Paketima se mogu grupirati semantički srodni elementi. Tako je olakšano uočavanje granica sa okolinom. Grupiranjem elemenata u pakete stvaraju se dijelovi sustava na kojima se može paralelno raditi za vrijeme oblikovanja.

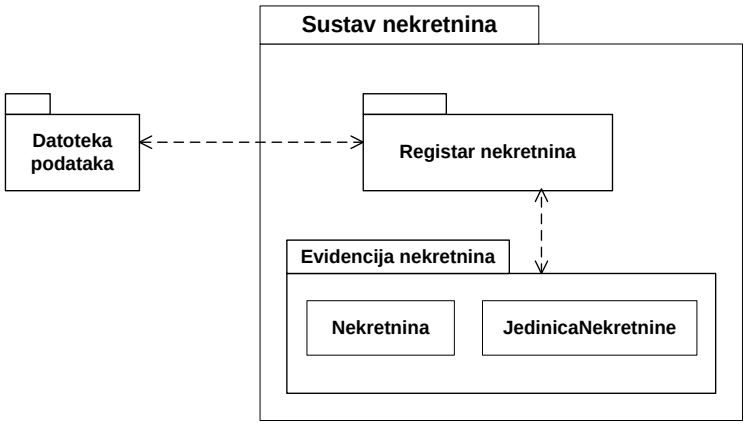
Pravilo je da svaki element može pripadati samo jednom paketu. Paketi mogu međusobno biti u hijerarhiji, a paket na vrhu obično se posebno označava (primjerice stereotipom <<toplevel>>). U analizi poslovanja u paketima se mogu grupirati slučajevi korištenja i prepoznate klase.

Ovisno o tome koliko razina detalja želimo pokazati dijagramom paketa, možemo naznačiti samo naziv (slika 4.14.a) ili unutarnju strukturu (slika 4.14.b) ili svojstva unutarnje strukture (slika 4.14.c).



Slika 4.14. Različiti načini prikaza istog paketa

Dijagrami paketa mogu biti ugniježđeni, kako je to naznačeno u primjeru na slici 4.15. Paket *Sustava nekretnina* unutar sebe ima dva paketa: *Registar nekretnina* i *Evidencija nekretnina*.



Slika 4.15. Različiti načini prikaza istog paketa

Pitanja za ponavljanje

1. *Objasnite zašto su razni tipovi modela i modeli različitih pogleda na sustav važni za razvoj i izgradnju IS-a.*
2. *Kako se izrađuje matrični dijagram i koja je njegova namjena?*
3. *Nabrojite nekoliko primjera matrica veza i objasnite čemu svaki od tih primjera služi.*
4. *Objasnite primjer matrice veza koja pruža pregled procesa i podataka sustava.*
5. *Objasnite matricu strukture sustava i njenu povezanost s okruženjem.*
6. *Koji su osnovni simboli lanca događaja i procesa?*
7. *Kojih pravila se treba pridržavati kod izrade lanca događaja i procesa?*
8. *Od kojih simbola se gradi dijagram toka podataka?Objasnite svaki od tih simbola.*
9. *Što je kontekst dijagram i kako se crta?*
10. *Objasnite dekompoziciju u modeliranju dijagramom toka podataka?*
11. *Čemu služi model entiteti-veze?*
12. *Koji simboli se koriste prilikom izrade modela entiteti-veze? Koja ograničenja nad konceptima se mogu prikazati u ovom modelu?*
13. *Što prikazuje dijagram slučajeva korištenja?*
14. *Objasnite kako se izrađuje scenarij slučaja korištenja?*
15. *Koje dijelovi sustava se mogu prikazati dijagramom sijeda?*
16. *Koji dijelovi sustava se mogu prikazati dijagramom klasa?*
17. *Kakve vrste relacija za povezivanje klasa u dijagramu klasa poznajete?*
18. *Nabrojite neke od preporuka za crtanje dijagrama klasa.*
19. *Što se od sustava može pokazati dijagramom aktivnosti?*
20. *Objasnite proširivanje dijagrama aktivnosti sa stazama?*
21. *Čemu služe dijagrami paketa?*

5. Metode razvoja

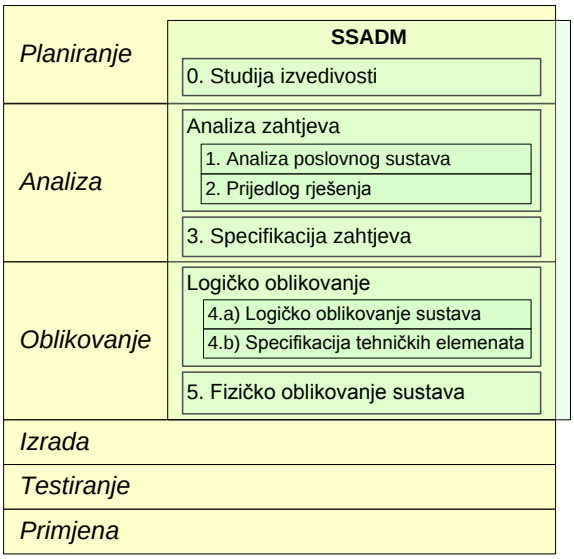
Metoda SSADM
Razvojni okvir RUP
Metoda Scrum

U ovom ćemo poglavlju obraditi nekoliko metodoloških pristupa razvoju IS-a. To su strukturirani pristup, objektno-orijentirani pristup i agilni pristup. Detaljno će biti opisani:

- za strukturni pristup metoda SSADM (eng. *Structured Systems Analysis and Design Methodology*);
- za objektno-orijentirani pristup RUP (eng. *Rational Unified Process*) metoda;
- za agilni pristup *Scrum* metoda.

5.1. Metoda SSADM

Metoda SSADM (eng. *Structured Systems Analysis and Design Methodology*) je tradicionalna strukturna metoda koja podržava faze analize i oblikovanja programskog rješenja. Ostale faze razvoja, kao što su izrada, testiranje i uvođenje u rad SSADM, uglavnom ne podržava (više u nastavku; vidi sliku 5.1.).



Slika 5.1. Primjena SSADM u modelu razvoja programske potpore

Ova strukturirana metoda razvijena je u Velikoj Britaniji. Razvila ju je CCTA (eng. *Central Computing and Telecommunications Agency*) za potrebe državne uprave te je 1983. godine postala standard za vladine projekte. Tijekom godina postala je i *open* standard koji primjenjuje veliki broj informatičkih poduzeća.

SSADM se koristi zbog discipliniranog inženjerskog pristupa koji u odgovarajućim, dobro uspostavljenim, okolnostima osigurava kvalitetu sustava koji se razvija. Uspjeh ove metode se zasniva na činjenici da se koristi trima neovisninim tehnikama koje prikazuju različite poglede na sustav.

U sebi ima cijeli niz preporučenih dijagrama kojima se mogu prikazati detalji poslovanja. U radu na analizi i oblikovanju programske potpore spomenuti dijagrami timovima pomažu da svladaju znanje iz područja poslovanja i da modeliraju novi poslovni sustav.

5.1.1. Tehnike modeliranja

SSADM upotrebljava ove tri osnovne tehnike modeliranja:

- Logičko modeliranje podataka (eng. *Logical Data Modelling*) - podržava aktivnosti prepoznavanja, modeliranja i dokumentiranja opsega podataka poslovnog sustava. Sastoji se od logičke strukture podataka (eng. *Logical Data Structure*, LDS) i pridružene dokumentacije. Logička struktura podataka prikazuje entitete i njihove međusobne veze te predstavlja SSADM inačicu modela entiteti-veze (MEV), odnosno dijagrama entiteti-veze (DEV).
- Modeliranje tijeka podataka (eng. *Data Flow Modelling*) - podržava aktivnosti prepoznavanja, modeliranja i dokumentiranja toka podataka, odnosno dokumenata, kroz poslovni sustav. Ova se tehnika sastoji od skupa integriranih dijagrama tijeka podataka (DTP, eng. *Data Flow Diagrams*) koji su podržani odgovarajućom dokumentacijom.
- Modeliranje stanja entiteta (eng. *Entity Event Modelling*) – to je proces koji prepoznaje, modelira i dokumentira poslovne događaje koji imaju utjecaj na svaki pojedini entitet te ishod tih utjecaja. Prikazuje se kao skup životnih ciklusa svakog pojedinog entiteta (eng. *Entity Life History*, ELH) i odgovarajuće dokumentacije.

5.1.2. Faze razvoja

SSADM se sastoji od faza (eng. *stage*), faze od koraka (eng. *step*), a koraci od zadataka (eng. *task*). Osnovni životni ciklus se sastoji od 5 (odnosno 6) faza (slika 5.1.):

1. Studija izvedivosti (faza 0)

U ovoj se fazi provodi početna analiza poslovanja u kojoj je potrebno ustanoviti ograničenja ljudskog potencijala, vremena i drugih značajnih resursa. Kreira se početna lista problema/zahtjeva (eng. *Problems/Requirements List*), DTP prve

razine postojećeg poslovnog sustava i pregledni DEV koji ne mora biti kompletan i koji može sadržavati veze više-na-više među prepoznatim entitetima. Korištenjem DTP-a prve razine identificiraju se glavna područja istraživanja i prikazuju korisnici i njihova međusobna povezanost.

Cilj je kreirati jedinstvenu i konzistentnu listu problema/zahtjeva, koja će zajedno s opisom logičkog sustava pomoću DTP-a i opisom podataka pomoću ELH dati odgovarajuću i točnu definiciju cjelokupnog područja koje se promatra.

2. Analiza poslovnog sustava (faza 1)

U fazi 1 analizom tokova podataka identificira se rad postojećeg poslovnog sustava, te uključuje kreiranje DTP-a postojećeg sustava na osnovu provedenih intervjua s naručiteljem i/ili korisnicima i proučavanja dokumenata. To je najzahtjevniji korak faze analize, pogotovo kada se malo zna o sustavu koji se razvija.

Nakon izvršene studije izvedivosti potrebno je dalje razraditi poslovni sustav pomoću DTP-a nižih razina. DTP-ovi se rade do razine koja je potrebna za razumijevanje što se u sustavu događa. Sada DEV treba prikazati detaljan logički model postojećeg sustava.

3. Prijedlog rješenja (faza 2)

U fazi 2 formira se od tri do šest mogućih opcija za rješavanje problema definiranih u fazi 0. Cilj je da korisnici koji su za to odgovorni odaberu opciju projekta, koja će se dalje razvijati. Na projektantima leži odgovornost za jasno razumijevanje ponuđenih opcija. Uvijek postoji mogućnost da niti jedna od opcija ne bude prihvatljiva, da se usvoji neko hibridno rješenje ili da se zaključi da razvoj sustava nije izvediv.

DTP i DEV izrađuju se tako da podrže ponuđene opcije, a nakon odabira se izrađuju varijante koje podržavaju odabranu opciju. Prijedlozi rješenja u pravilu su prijedlozi oblikovanja novog sustava u koje bi trebalo biti ugrađeno i preoblikovanje poslovnih procesa (eng. *Business Process Reengineering*, BPR).

Ova faza je najveći korak u nepoznato jer model postojećeg sustava predstavlja podlogu za oblikovanje novog sustava. Analitičari su se uvijek suočavali s problemom prijelaza od poznatog, postojećeg sustava na nepoznati novi sustav koji bi trebao riješiti sve prisutne probleme. Primjerice, sada se DTP i DEV preoblikuju i tako doprinose novim ili izmijenjenim problemima/zahtjevima u listi korisničkih zahtjeva.

4. Specifikacija zahtjeva (faza 3)

Sada dokumentacija prethodnih faza doprinosi detaljnom zapisivanju funkcionalnih i nefunkcionalnih zahtjeva. Uz to se definiraju zahtjevi sigurnosti, kontrole sustava i praćenja sustava.

DTP-ovi koji opisuju odabranu opciju poslovnog sustava sada se šire i detaljiziraju. Ako je potrebno, prilagođavaju se DTP-ovi najnižih razina

uključivanjem rješenja iz stavki navedenih u listi problema/zahtjeva i obrnuto, spomenuta lista se mora ažurirati sa detaljima iz prihvaćene opcije.

DEV se mijenja i poboljšava koristeći principe normalizacije. Kompletiraju se opisi entiteta tako da sadrže sve poznate atribute. Opis atributa na ovom stupnju ne mora sadržavati podatke o formatu.

Proučavanje detaljne logike sustava pomaže u modeliranju ELH. Cilj je analizirati životni ciklus svakog entiteta u sustavu kako bi se izvršila provjera jesu li procesi u DTP-u u skladu s DEV. Tako se oblikuje novi sustav te izrađuje matrica životnog ciklusa svih entiteta koja identificira entitete zahtijevanog DEV i inicijalni skup događaja koji pokreću procese zahtijevanih DTP-ova.

Na kraju još jednom treba pregledati liste problema/zahtjeva i predložiti rješenja za stavke koje nisu bile ranije riješene. Ukoliko se neki zahtjev neće uključiti u zahtijevani sustav to treba dokumentirati.

5. Specifikacija logičkog sustava (faza 4)

Ova faza se sastoji od specifikacije tehničkih elemenata budućeg sustava i logičkog oblikovanja tog sustava. Logičko oblikovanje se razrađuje do detaljnog opisa procesa, izrade upita i strukture podataka.

6. Fizičko oblikovanje (faza 5)

Logička specifikacija podataka i procesa iz faze 4 i tehnička specifikacija sustava koriste se za fizičko oblikovanje baze podataka i specifikaciju skupa procedura za obradu podataka (za kasnije programiranje).

Na ovakav način se u najranijoj fazi razvoja programske potpore (slika 5.1.) dolazi do detaljnih informacija od kojih se gradi sustav. To je potrebno prije svega zbog korisničkih zahtjeva. Naravno da se u fazama izrade, testiranja i primjene upotrebljava SSADM dokumentacija, no sama metoda za ove faze nije razrađena.

5.1.3. Osvrt na karakteristike metode SSADM

Specifikacija zahtjeva je postavljena kao osnovni korak SSADM-a iz kojeg kreće razvoj novog sustava, Kad je specifikacija usvojena više se ne bi smjela mijenjati.

Iskustvo je pokazalo da su standardni modeli i tehnike kojima se prikazuju analiza postojećeg sustava (DTP, MEV, ELH, dijagrami toka, čak i dijagrami mreža i arhitekture) strani korisniku. Međutim, činjenica je da ih projektant upotrebljavaju za prikaz detalja analize što znači da je to za njega temelj znanja o području koje se opisuje.

Spomenuti modeli koji se koriste predstavljaju dokumentaciju velikih gabarita koju je teško održavati, čak i sa CASE pomagalima (što je vrlo skupo i neisplativo u praksi).

Često nove verzije projekta ili novi projekti nastaju iz postojeće dokumentaciji, te se stvaraju duplikati. Tako nastaje redundancija informacija, što je suprotno nastojanjima

da se u informatici ponovno koriste (eng. *reuse*) znanje i elementi sustava koji su već negdje modelirani ili implementirani.

Lista zahtjeva se naslanja na modele koji često ne nastaju prije zahtjeva nego paralelno ili poslije (upravo zato što korisnik nije aktivni sudionik), te se događa da su dokumenti protkani propustima i pogreškama. Uz to, SSADM ima tehniku zapisivanja zahtjeva u obliku liste problema/zahtjeva gdje se bilježe opis zahtjeva u obliku pisanog teksta. Ako je taj tekst terminološki blizak korisniku, onda može u pozadini sadržavati mnoga znanja o poslovanju koje se podrazumijeva, ali ga projektant ne mora biti svjestan. S druge strane, projektant piše rješenja zahtjeva koja često sadrži tehničke elemente radi uvjerljivosti, pa korisnik treba vjerovati projektantu da je takvo rješenje dobro.

5.2. Razvojni okvir RUP

RUP (eng. *Rational Unified Process*) je objektno-orijentirani pristup koji nudi IBM Rational Software, a rezultat je nastojanja da se definira okvir procesa razvoja programske potpore, te da se koristi UML kao jezik za modeliranje sustava.

RUP je cjelovit (poput prave metodologije) jer osim faza razvoja definira i osnovne zadatke upravljanja projektom i razvoja projekta. Tako su unutar svake faze uključeni zadaci:

- modeliranje poslovanja
- modeliranje zahtjeva
- analiza i oblikovanje
- izrada
- testiranje
- primjena
- upravljanje konfiguracijom i promjenama
- vođenje projekta.

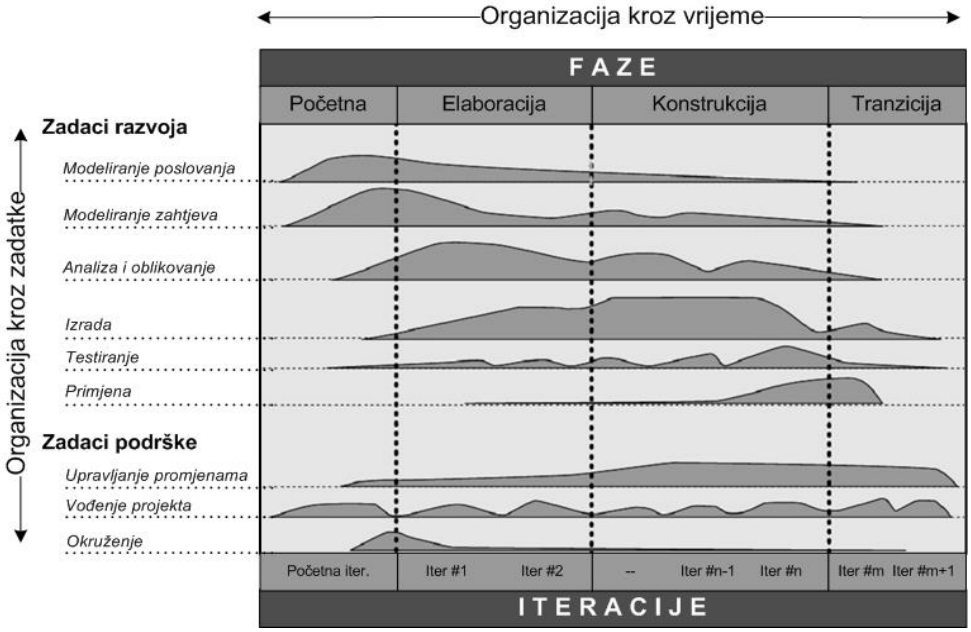
Stoga je RUP u stanju pružiti mehanizme za osiguranje izvedivosti procesa razvoja kao što su: vremenski relativno kratke iteracije sa dobro definiranim ciljevima i odluka da/ne (eng. *go/nogo decision point*) za nastavak nakon svake faze.

5.2.1. Faze i iteracije

Dvije osnovne dimenzije RUP-a su: **faze** koje prikazuju četiri osnovna stanja projekta kroz vrijeme i **zadaci** koji prikazuju logičke aktivnosti koje treba obaviti za vrijeme projekta.

Poznata i karakteristična shema s izrazitim takozvanim grbavim grafikonima (eng. *hump chart*) [24], slika 5.2. ilustrira dvije dimenzije razvoja programa:

- vrijeme razdijeljeno u faze, a faze u iteracije; horizontalni prikaz;
- zadatke; vertikalni prikaz s resursima koje slikovito prikazuju spomenute grbe na grafikonima. To je gruba procjena učešća odgovarajućeg zadatka u pojedinim fazama i iteracijama.



Slika 5.2. Faze i iteracije, te zadaci RUP pristupa

Za životni ciklus razvoja pomoću RUP-a može se kazati da je u cijelosti (generalno) serijski, a po dijelovima (detaljno) iterativan. Karakteristika slijednosti prisutna je kroz faze, a iterativnosti kroz zadatke.

Faze su podijeljene na iteracije, pa se može reći da je RUP prije svega iterativna metodologija. Broj iteracija unutar neke faze nije određen i ovisi o potrebama razvoja.

Četiri faze RUP-a se, generalno, poklapaju s osnovnim fazama razvoja programske potpore:

- početna faza koja definira opseg projekta;
- faza elaboracije koja definira zahtjeve i osnovni plan arhitekture sustava;
- faza konstrukcije u kojoj se izrađuje programska potpora;
- faza tranzicije u kojoj se novi sustav uvodi kod korisnika tako da se napravi obuka korisnika, instalacija aplikacija sustava i osigura podrška.

Na slici 5.2. mogu se vidjeti sljedeće zadaće svake od faza:

- Početak (eng. *Inception*)

Osnovni zadatak početne faze je postići dogovor svih sudionika projekta oko ciljeva koje treba ispuniti. U tu svrhu treba izraditi *high-level* model zahtjeva koji određuje opseg projekta i potencijalno predstavlja početak razvoja prototipa UI sučelja. Uspostavlja se radna okolina i organiziraju aktivnosti za radni tim, koji su posljedica *high-level* plana rada i izvršavanja projekta.

- Elaboracija (eng. *Elaboration*)

  U ovoj fazi zadatak je detaljna specifikacija zahtjeva i uspostavljanje stabilne arhitekture sustava. Dijelove sustava određene zadanom arhitekturom treba fizički realizirati i testirati. Tako se gradi kostur sustava koji se u fazi konstrukcije puni detaljnom funkcionalnošću. Posebnu pažnju treba posvetiti listi potencijalnih rizika. One rizike koji dovode u pitanje uspješnost projekta treba eliminirati do kraja ove faze.

- Konstrukcija (eng. *Construction*)

  U ovoj se fazi razvija novi sustav, a cilj je osposobiti ga za implementaciju. Najvažniji su zahtjevi, njihova analiza i specifikacija, oblikovanje rješenja koji zadovoljava te zahtjeve, te kodiranje i testiranje izrađenog programa. Ako je potrebno, izdaju se prve verzije sustava kako bi se dobila povratna informacija od korisnika.

- Tranzicija (eng. *Transition*)

  Ova je faza fokusirana na isporuku proizvoda i njegovo korištenje. Sustav testira izvođač i krajnji korisnik, dogovaraju se i nakon toga izvode potrebne popravke i izmjene. Treba obučiti krajnje korisnike i one koji će biti podrška sustavu.

Grbavi grafikon za, primjerice, zadatak modeliranja zahtjeva pokazuje veliki udio u početnoj fazi, ali i kasnije, u fazi konstrukcije, često sa javljaju dodatni zahtjevi koji traže i dodatni rad.

Grbavi grafikoni predstavljaju vizualni mehanizam koji omogućava grubu procjenu koliko pojedini zadatak ima udjela u svakoj od faza. Drži se prirodnim da se obuhvati jedan dio (podskup) zahtjeva i da se nad njima obavi analiza, da se eventualno vrati unatrag i preinače neki od zahtjeva, a da se nakon toga započne oblikovanje koje opet može ukazati na preinake zahtjeva i/ili analize, zatim se započne s kodiranjem koje može ukazati na preinake oblikovanja. Ovaj slijed predstavlja jedan inkrement u iterativnom pristupu, kakav podržava RUP.

5.2.2. Iterativni pristup

Faze RUP-a su podijeljene u više iteracija. Iteracije se odnose na jedan dio cjelokupnog sustava koji se u tom momentu razvija (za razliku od vodopadnog modela koji zahtijeva razvoj cjelokupnog sustava).

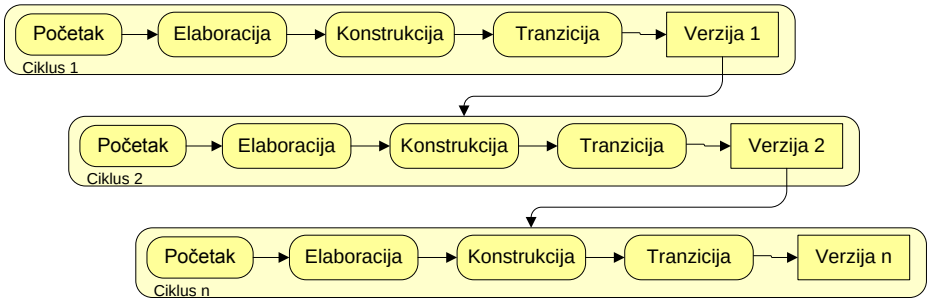
Svaka iteracija ima detaljni plan s jasno određenim ciljem. Pojedina iteracija se gradi na osnovu urađenog iz prethodnih iteracija i postepenim dodavanjem nekog dijela posla sve do konačne verzije sustava.

Iterativna priroda RUP-a vidi se u aktivnostima pojedinih disciplina (vertikalna dimenzija sa slike 5.2.) i njihovom značenju u fazama razvoja. Stoga je bit RUP-a u njegovim disciplinama, a ne u fazama[23]. U pojedinoj iteraciji naizmjenično se odvijaju i međusobno podupiru aktivnosti zadataka, stalno imajući na umu cilj te iteracije. Tijekom jedne iteracije jedan dio korisničkih zahtjeva se izdvoji (s obzirom na cilj iteracije) te se analizira, dizajnira, kodira, testira i integrira u proizvod koji je sveukupno rezultat tekuće i prethodnih iteracija.

5.2.3. Inkrementalni razvoj

U RUP-u jedna verzija proizvoda mora proći kroz sve četiri faze. Stoga za vrijeme ili nakon faze tranzicije započinje novi posao koji rješava preostale neriješene zahtjeve ili jedan njihov dio.

Kako se vidi na slici 5.3., jedan ciklus razvoja, koji uključuje odabrane slučajeve korištenja, provodi se kroz sve faze razvoja i na kraju se izdaje verzija programa za korisnika. Ciklusi se ponavljaju, svaki put s odabranim novim skupom slučajeva korištenja koje treba ugraditi u programsko rješenje. Unutar novog ciklusa dopuštene su i preinake na osnovu primjedbi koje je dao korisnik testirajući rezultate prethodnog ciklusa.



Slika 5.3. Inkrementalni razvoj RUP-a

5.2.4. Osvrt na karakteristike RUP pristupa

Prednost RUP pristupa je u prilagodljivosti i ne bi bilo dobro koristiti se njime po nekom ustaljenom obrascu. Prilagodljivost se ogleda u iterativnom pristupu

[23] U vodopadnom modelu su poslovi i priroda spoznaja (učenje o problemu) jedno, dok je u RUP-u to odijeljeno, čak se navodi kao karakteristika da se u početku malo zna te se i obuhvaćaju samo neki zahtjevi (UC) a u kasnijim iteracijama ostatak.

oblikovanju i izradi programskog rješenja. Razvoj i isporuka manjih dijelova cjelokupnog sustava na vrijeme će ukazati na kritične točke i tehničke rizike.

Prilagodljiv je i u određivanju prioriteta u odabiru skupa zahtjeva tako da se prioriteti po potrebi mogu i promijeniti. Prilagodljiv je i s obzirom na rizike u razvoju. Rizici višeg prioriteta rješavaju se u ranijim iteracijama za razliku od nižih rizika koji se mogu rješavati kasnije. To je ujedno i bit upravljanja rizika u razvoju programske potpore.

RUP kao iterativan postupak zahtijeva pažljivo vođenje projekta kako se ne bi dogodilo da projekt traje predugo. Primjerice, tvrdnja da se analiza odvija u svim fazama i u svim iteracijama, samo u početku većeg intenziteta, a poslije smanjeno, može biti protumačena na preslobodan način. To nije upitno, no što je to više, a što manje. Naime, grbavi grafikon samo simbolički objašnjava da je u pojedinim iteracijama neki od zadataka više ili manje zastupljen. S druge strane, RUP dopušta da se, koliko god je potrebno, vraća na svaki od zadataka, kako bi u novoj iteraciji rezultat bio bolji od prethodne iteracije. Rizici i odluka da ili ne su ključni i nikako ih ne treba ublažavati željom da se udovolji korisniku ili potrebom da se obvezno projekt odradi do kraja.

5.3. Metoda Scrum

Scrum se, kao okvir za proces razvoja programskih proizvoda, počeo koristiti 90-tih godina prošlog stoljeća. Unutar *Scrum* pristupa mogu se koristiti razni procesi i tehnike. *Scrum* omogućava međusobno efikasno upravljanje projektom i razvoj programa.

Scrum okvir se sastoji od timova kojima su pridružene uloge, te događaja, artefakata i pravila. Pravila povezuju uloge, događaje i artefakte, zadaju njihove odnose i međudjelovanje. Strategija korištenja *Scrum* pristupa, zadana na razini vođenja projekta razvoja, nije strogo određena, ona se prilagođava nekom zadanom projektu i njegovim specifičnostima.

Scrum je utemeljen na takozvanoj teoriji empirizma [22] koja tvrdi da znanje dolazi iz iskustva ili odlučivanja koja se temelje na nepoznatom.

Karakteristike *Scruma* su:

- Transparentnost koja zahtijeva da svi aspekti procesa moraju biti definirani kao zajednički standardi, tako da svi sudionici dijele zajedničko razumijevanje o radu i proizvodu.

- Kontrola koja se često provodi tijekom proces tako da se na vrijeme uoče neželjena odstupanja.
- Prilagodba koja osigurava da se odstupanja od prihvatljivih granica isprave. Ispravci se moraju provesti što je prije moguće radi minimiziranja budućih odstupanja.

5.3.1. Sudionici Scrum pristupa

Scrum timovi trebali bi imati sposobnost samoorganiziranja, što znači da sami biraju najbolji način za obavljanje nekog zadatka. Također, timovi bi trebali biti više-funkcijski tako da imaju sve potrebne kompetencije za obavljanje posla i ne ovise o nikome izvan tima. *Scrum* tim se sastoji od:

- Vlasnika proizvoda (eng. *Product Owner*)
- Razvojnog tima (eng. *Development Team*)
- *Scrum Mastera*.

Vlasnik proizvoda (jedna osoba u timu) je odgovoran za upravljanje *Product Backlogom* (to je neobavljeni posao, odnosno zaostatak). Kako bi vlasnik proizvoda bio uspješan cijeli tim mora poštovati njegove odluke koje uključuju:

- Objašnjavanje razvojnom timu vizije, ciljeva, i skupa zadataka.
- Određivanje redoslijeda zadataka.
- Osiguranje vrijednosti rada razvojnog tima.
- Osiguranje vidljivosti *Product Backloga* tako da bude svima transparentan i da pokazuje slijedeće zadatke na kojima će tim raditi.

Razvojni tim se sastoji od profesionalaca koji rade na potencijalno isporučivom inkrementu proizvoda na kraju svakog Sprinta. Razvojni timovi bi trebali imati sljedeće karakteristike:

Samoorganizirajući su, što znači da nitko izvan tima (čak ni *Scrum Master*) ne utječe na način realizacije *Product Backloga* u inkrement proizvoda.

Višefunkcijski su i posjeduju vještine potrebne da mogu izraditi gotovi inkrement proizvoda.

Svi bi trebali biti jedino i samo programeri, jer *Scrum* pristup ne poznaje druge specijalnosti. To ne znači da pojedinci nemaju specifična znanja, ali ona nisu od presudne važnosti u odgovornosti tima za obavljeni posao.

Razvojni tima bi trebao biti dovoljno mali da bude okretan, ali i dovoljno velik da obavi većinu određenih zadataka. Čini se da manje od tri člana tima nije prihvatljivo jer se može nedostajati vještina. Također se pokazalo da više od devet članova tima

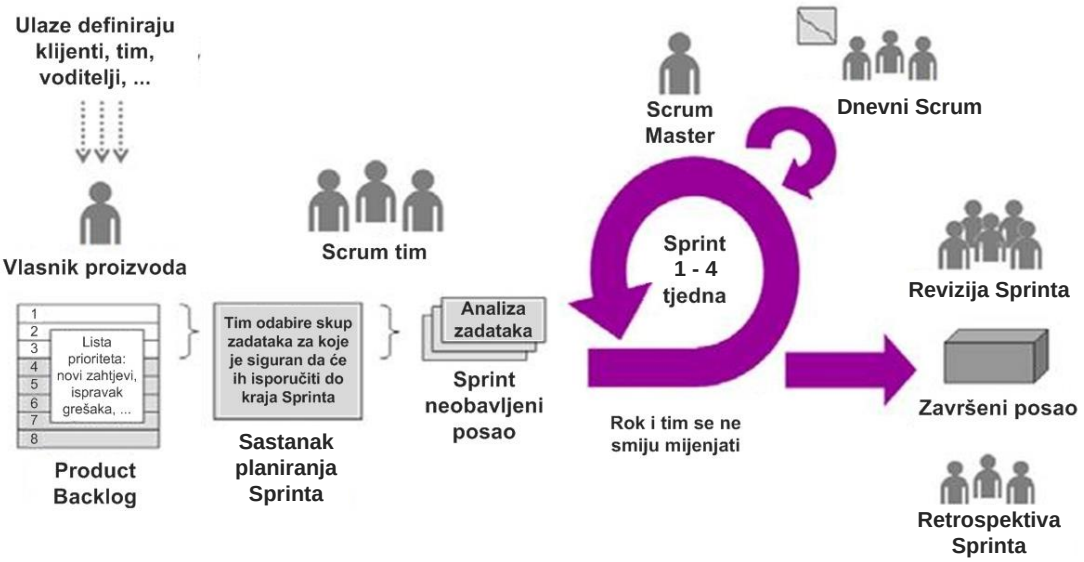
nije efikasno jer se ne mogu međusobno koordinirati i među njima nije moguće sprovesti spomenuti empirijski proces.

Scrum Master je odgovoran za provedbu *Scrum* projekta, njegovo razumijevanje i korištenje. On pomaže onima koji su izvan *Scrum* tima da razumiju njihove postupke. Svim sudionicima treba pomoći da promijene svoje ponašanje i tako doprinesu maksimizaciji vrijednosti proizvoda.

5.3.2. Scrum događaji

Scrum je kombinacija iterativnog i inkrementalnog pristupa s ciljem optimizacije predvidivosti i kontrole rizika. Četiri formalna *Scrum* događaja (slika 5.4.) koja su zastupljena u jednoj iteraciji (naziva se *Sprint*) su:

- Sastanak planiranja *Sprinta*
- Dnevni *Scrum*
- Revizija *Sprinta*
- Retrospektiva *Sprinta*.



Slika 5.4. Ciklus razvoja Scrum procesa[24]

Događaji u *Scrumu* su vremenski ograničeni sa zadanim maksimalnim trajanjem, a nazivaju se *Sprint*. Tijekom *Sprinta* treba proizvesti jedan inkrement proizvoda, a svaki *Sprint* bi trebao imati jednako vremensko trajanje tijekom razvoja proizvoda.

[24] http://www.sadhanbiswas.com/myblog/techsavvy/agile-methodology/ (23.01.2012)

Novi *Sprint* započinje neposredno nakon što završi prethodni i svaki *Sprint* je na neki način mini-projekt čije trajanje ne prelazi mjesec dana. Naime, za svaki *Sprint* se zna što treba napraviti, kako i na koji način, tko obavlja zadani posao i što se očekuje kao konačni proizvod (jedan inkrement). Kontrola i prilagodba je u okviru vremena trajanja, a time se i rizik troškova ograničava na jedan mjesec.

Samo vlasnik proizvoda ima pravo prekinuti *Sprint* (možda i na inicijativu vanjskih sudionika, tima ili *Scrum Mastera*), i to ako cilj izrade projekta nije važeći ili se, s obzirom na neke novonastale okolnosti, gubi smisao izrade zadanog inkrementa.

Na **sastanku planiranja *Sprinta*** dogovara se posao koji će biti obavljen u jednoj iteraciji, a dogovara ga cijeli tim. Sastanak je vremenski ograničen na osam sati za *Sprint* od mjesec dana (kraći *Spint* ima proporcionalno kraće vrijeme).

Na sastanku je potrebno dati odgovor za pitanja (1) koje zadatke treba izraditi u nadolazećem inkrementu *Sprinta* i (2) kako će taj posao biti obavljen.

Prvo je pitanje zapravo određivanje opsega zadataka i razumijevanje funkcionalnosti koje treba ugraditi u proizvod. Kako bi odgovor na ovo pitanje bio zadovoljavajući treba razmotriti što je sve preostalo u *Product Backlogu*, te koliki su kapaciteti i kakve su performanse tima. Tim određuje koji zadaci će ući u novi *Sprint* te se ovisno o tome definira krajnji cilj te iteracije.

Za odgovor na drugo pitanje također je važna odluka tima koji posao započinje oblikovanjem budućeg proizvoda. Ukupni posao može biti različite veličine i težine te ga je na ovom sastanku planiranja potrebno razložiti na dnevne obaveze. Tijekom iteracije tim se samoorganizira ovisno o zadacima koji su ostali u *Sprint Backlogu* (neobavljeni zadaci). Ako se nešto od plana (zadaci, tehnologija i slično) promijeni, tim bi trebao razgovarati s vlasnikom proizvoda o promjenama, opsegu *Sprinta* i ciljevima.

Dnevni *Scrum* bi trebao trajati 15-tak minuta, a služi samo za to da tim uskladi aktivnosti i donese plan za slijedećih 24 sata. Ovaj sastanak bi se trebao održavati uvijek u isto vrijeme uz pružene odgovore na pitanja:

- Što je napravljeno od prethodnog sastanka?
- Što će se učiniti do slijedećeg sastanka?
- Koje prepreke su prisutne u rješavanju?

Dnevni *Scrum* služi timu da procijeni trend napredovanja u izradi proizvoda te pomaže timu da optimizira vjerojatnost postizanja cilja.

Operativno *Scrum Master* organizira dnevni sastanak tima, no razvojni tim je odgovoran za održavanje i sadržaj dnevnog sastanka.

Dnevni *Scrum* poboljšava komunikaciju među članovima tima i eliminira potrebu za ostalim sastancima koji se često događaju prekasno. Također, na ovim sastancima se unapređuje znanje i potiče brže donošenje odluka. Sve zajedno predstavlja bolje sprovođenje kontrole i prilagođavanje manjim izmjenama.

Revizija *Sprinta* se radi na kraju svakog *Sprinta,* a predstavlja kontrolu izrađenog inkrementa proizvoda i evidentiranje eventualne potrebe za prilagodbama *Product Backloga*. Ovaj sastanak je informativne prirode i služi za prezentaciju inkrementa kako bi se potakle povratne informacije i poboljšala komunikacija sudionika projekta.

Na reviziji *Sprinta* treba razmotriti slijedeće elemente:

- Vlasnik proizvoda će pregledom inkrementa reći što jest, a što nije urađeno.
- Razvojni tim prezentira tijek razvoja inkrementa, probleme nastale u razvoju i kako su riješeni.
- Razvojni tip prezentira proizvod i odgovara na pitanja.
- Vlasnik proizvoda komentira zadatke koji su preostali u *Product Backlogu* i predlaže rokove za iduće inkremente.
- Cijela grupa raspravlja što dalje odraditi i rezultat ove rasprave je ulaz u događaj sastanka planiranja novog *Sprinta*.

Retrospektiva *Sprinta* služi kao kontrola članovima tima. Oni obavljaju svojevrsnu samoanalizu kroz ono što je urađeno i ono što treba napraviti u idućoj iteraciji. Ovaj događaj je nastavak revizije *Sprinta* a prethodi sastanku planiranja idućeg *Sprinta*. Svrha je retrospektive:

- Kontrola završenog *Sprinta*, njegova tijeka s obzirom na ljude, međuodnose, zadatke i alate.
- Prepoznavanje glavnih zadataka koji su dobro napravljeni i unapređenja na osnovu uočenoga.
- Planiranje primjene unapređenja za način na koji *Scrum* tim obavlja svoj posao.

Identificirana unapređenja bit će implementirana tijekom sljedeće iteracije, To je svojevrsna prilagodba poslova na osnovu kontrole samog razvojnog tima.

5.3.3. Osvrt na Scrum

Može se primijetiti da Scrum pristup isto kao i RUP u sebi sadrži i model pristupa rješavanju problema (događaji) i zadatke koje sudionici trebaju odraditi u svakoj fazi projekta. Tako se u ovim metodama isprepliću i projektni posao i vođenje projekta. To je i razumljivo budući da su to novije metode razvoja programskih proizvoda kojima je glavni cilj brzina i učinkovitost.

Zanimljiva je lista koja se naziva *Product Backlog* a sadrži sve mogućnosti, funkcionalnosti, zahtjeve, poboljšanja i popravke. Sve zajedno je skup zadataka koje treba napraviti u budućnosti. To znači da *Product Backlog* (teoretski) nikada nije konačan. U početku sadrži zahtjeve koji su prepoznati i jasni, no u nastavku kroz iteracije evoluira kako evoluira i inkrement proizvoda.

Iako je ideja o dinamičnosti liste Product Backloga prihvatljiva sa stanovišta korisnika zadovoljnog proizvodom, ona može biti stalni izvor promjena tako da proizvod „nikada ne bude gotov“. Dobra je strana ove liste što je posložena prema vrijednosti, riziku, prioritetu i nužnosti. Tako se uvijek rješavaju funkcionalnosti bitne za konačni proizvod.

Vrlo je vjerojatno da one stavke u listi koje su više rangirane iza sebe već imaju nekoliko razmatranja. Tako su one jasnije definirane i opisane sa više detalja jer su bile više puta razmatrane (kroz više iteracija) pa postoji više konsenzusa oko njihovog rješenja.

Kako više *Scrum* timova može zajedno raditi na jednom proizvodu treba *Product Backlog* obilježiti sa više atributa, tako da područja koja pojedini timovi rješavaju budu grupirana u cjeline.

Ako se proizvod želi proglasiti završenim, svi sudionici se oko toga trebaju složiti. To znači da ostatak (ako ga ima) u *Product Backlogu* više nije predmet projekta razvoja i (vrlo vjerojatno) za korisnika ne predstavlja bitne funkcionalnosti programskog proizvoda.

I za kraj [22]; Scrum postoji samo u svojoj cjelovitosti i funkcionira dobro kao kontejner za primijenjene tehnike, modele i stečena iskustva iz prakse.

Pitanja za ponavljanje

1. *Koje faze razvoja IS-a mogu biti podržane SSADM metodom?Koje faze SSADM metode su prisutne unutar faza razvoja IS-a?*
2. *Kojim se tehnikama modeliranja koristi metoda SSADM?*
3. *Objasnite fazu studije izvedivosti.*
4. *Objasnite fazu analize poslovnog sustava i kako na nju utječe faza studije izvedivosti?*
5. *Što sve treba napraviti u fazi prijedloga rješenja?*
6. *Kako faza prijedloga rješenja utječe na fazu specifikacije zahtjeva?*
7. *Objasnite specifikaciju logičkog sustava i fizičko oblikovanje u SSADM metodi.*
8. *Koje su dobre a koje loše strane liste korisničkih zahtjeva u SSADM metodi?*
9. *Što su faze a što iteracije u RUP pristupu?*
10. *Od koliko faza se sastoji RUP pristup? Objasnite svaku od njih.*
11. *Kako su podijeljeni zadaci RUP pristupa? Nabrojite te zadatke.*
12. *Objasnite karakteristiku iterativnog pristupa i inkrementalnog razvoja RUP pristupa.*
13. *Objasnite pojam i karakteristike Scrum okvira.*
14. *Tko su Scrum sudionici?Opišite ih.*
15. *Navedite četiri osnovna Scrum događaja. Objasnite gdje se u pojedinoj iteraciji Sprinta svaki od događaja nalazi i zašto?*
16. *Objasnite događaj sastanka planiranja Sprinta.*
17. *Objasnite čemu služi i tko provodi dnevni Scrum.*
18. *Čemu služi revizija a čemu retrospektiva Sprinta?*
19. *Što je to Product Backlog i čemu služi?*

6. Specifikacija korisničkih zahtjeva

Definicija zahtjeva
Inženjerstvo zahtjeva
Metode razvoja i aktivnosti zahtjeva

6.1. Definiranje zahtjeva

Specifikacija korisničkih zahtjeva događa se najvećim dijelom u fazi **analize** poslovnog sustava. Prije analize važno je definirati probleme i izraditi studiju izvedivosti (kako je rečeno u metodi SSADM). Rezultat ovih početnih faza razvoja je definiran opseg projekta na osnovu kojeg se radi **početna** lista zahtjeva, a do **konačne** liste korisničkih zahtjeva dolazi se detaljnom analizom.

Namjena specifikacije zahtjeva i zadaci informatičara su:

- Analiza i razumijevanje korisnikovih potreba prema novom sustavu.
- Specificiranje tih potreba u obliku korisničkih zahtjeva i uspostava konceptualnih osnova za oblikovanje novog sustava.
- Realizacija tih zahtjeva kroz informacijski sustav koji u potpunosti zadovoljava navedene potrebe.

Ove zadatke nije jednostavno realizirati i ukoliko se pogriješi rezultat će biti loše izveden informacijski sustav.

Prikupljanje i specifikacija zahtjeva je jedan od ključnih preduvjeta za uspješnost informacijskog sustava. Ovaj korak je početak razvoja informacijskog sustava, pa je važno da bude dobro obavljen i dokumentiran. Bez obzira na izbor metoda koje se koriste u razvoju informacijskog sustava, kao i na izbor tehnika za izradu dokumentacije, zahtjevi moraju biti jasno definirani, detaljno opisani i razumljivi svim zainteresiranim stranama u procesu razvoja informacijskog sustava.

Loša i nepotpuna specifikacija zahtjeva zasnovana je na analizi, oblikovanju i izvedbi koja se oslanja na klimave temelje ili, još gore, na krivo postavljene temelje. Još je zanimljivije da ispravna i potpuna specifikacija zahtjeva ne jamči uspješnu implementaciju sustava, nego je tek čini vjerojatnom i mogućom. Napori i trud u razvoju softvera propadaju mnogo češće nego što bi trebalo, jer se pokazalo da što je razvoj softvera opsežniji i složeniji, to je vjerojatnost neuspjeha veća.

***PRIMJER*:** Capers Jones, utemeljitelj istraživanja produktivnosti softvera i „guru“ za metriku[25] napravio je dosta interesantnih radova o neuspjelim projektima. Tablica 6.1.[26] pokazuje da veliki projekti završavaju neuspjehom u velikim postotcima, a mali projekti u malim postotcima. Funkcijske točke[27] pokazuju složenost projekta.

Tablica 6.1. Odnos težine projekta i realizacije projekta

Funkcijske točke	Postotak prekida prije završetka projekta
100	6%
1.000	17%
10.000	45%
100.000	80%

Način definiranja **zahtjeva** i **ciljeva** novog informacijskog sustava nije formalno definiran. Neovisno o primijenjenoj metodologiji treba ih dokumentirati u neformalnom, ali pismenom obliku. Sami ciljevi ne smiju biti idealizirani do te mjere da u određenim okolnostima budu neostvarivi. Dakle, ciljevi projekta ne smiju biti samo zbroj želja ili dalekih vizija, već trebaju biti **realno** ostvarivi.

Kao polazna osnova za definiranje ciljeva budućeg sustava mogu poslužiti **evidentne slabosti** (takozvana uska grla) u postojećem sustavu. Otklanjanje tih slabosti može biti osnovni i polazni cilj koji se postavlja pred novi informacijski sustav.

Često se u praksi ciljevi sustava oblikuju po uzoru na već postojeća programska rješenja iz bliže ili dalje okoline. Naime, rijetko se razvija potpuno novi informacijski sustav koji se ne ugleda na neke, već postojeće, proizvode. Iz ovih iskustava informatičari mogu doprinijeti kvaliteti izrade zahtjeva. Osim toga, stečena iskustva i znanje mogu ubrzati specifikaciju zahtjeva tako da informatičari što prije počinju s oblikovanjem i izradom verzija programskog proizvoda.

6.2. Inženjerstvo zahtjeva

Zahtjevi obuhvaćaju aktivnosti koje uključuju:

- prikupljanje zahtjeva (utvrđivanje i razjašnjavanja zahtjeva, eng. *requirements elicitation*);
- analizu, procjenu i oblikovanje zahtjeva (eng. *requirements evaluation*);

[25] Funkcijski orijentirana metrika (eng. *function-oriented metrics*) bazira se na mjerenju funkcionalnosti isporučenog softverskog rješenja (proizvoda).
[26] Jones, C., Applied Software Measurement: Assuring Productivity and Quality, Second ed. McGraw Hill, 1996.
[27] Funkcijska točka (FT, eng. *function point*) je jedinica mjere pomoću koje se može izraziti količina poslovnih funkcionalnosti koje će korisniku pružiti informacijski sustav.

- specifikaciju i dokumentiranje zahtjeva (eng. *requirements specification*) kao rezultat prethodne radnje.

Prije rada na zahtjevima potrebno je provesti studiju izvedivosti (eng. *feasibility study*).

Aktivnosti prikupljanja i specifikacije zahtjeva započinju utvrđivanje osnovnih potreba za sustavom i određivanjem granica tog sustava. Analiza sustava sadrži aktivnosti upoznavanja s raznim detaljima poslovanja koji kroz modele sustava (model procesa i model podataka) predstavljaju specifikaciju i dokumentaciju zahtjeva. Ova bi dokumentacija trebala pridonijeti izradi kvalitetnog proizvoda.

6.2.1. Studija izvedivosti

Prilikom pokretanja projekta uvođenja informacijskog sustava u neko poduzeće:

- predstavnici **korisnika** (rukovodioci) definiraju poslovne potrebe i opseg projekta;
- **informatičari** nastoje:
  - razumjeti problem
  - upoznati korisnika i njegovu ideju o rješenju problema
  - sagledati potencijal i učinak zajedničkog udruživanja s ciljem rješavanja tog problema.

Studija izvedivosti trebala bi odgovoriti na nekoliko ključnih pitanja: organizacijsku izvedivost, tehničko-tehnološku izvedivost, vremensku izvedivost, ekonomsku izvedivost te na pitanje o spremnosti korisnika za prihvaćanje promjena.

Kako je studija izvedivosti jedan od početnih koraka, normalno je da se tako rano ne dogovaraju konkretni i detaljni parametri budućeg IS-a. Ovaj prvi korak dogovaraju visoko pozicionirane osobe (za obje zainteresirane strane) koje poslije sudionicima u projektu nisu operativno korisne i ne mogu im pomoći kad dođu do nepremostivih problema.

Primjerice, često su informatičari sudionici rada na projektu koji je dogovoren „naslijepo“ ili sa siromašnim opisom ugovorenog posla. Cilj i opseg takvog projekta su izraženi u par deklarativnih rečenica (npr. program za skladišno poslovanje).

Drugi primjer je pretjerano opsežna dokumentacija elaboriranja potreba i rješenja (tender[28]) koja sadrži previše detalja za koje nitko u ovoj ranoj fazi ne može biti siguran da su stvarno potrebni (npr. navođenje atributa i entiteta u prijedlogu rješenja buduće aplikacije za skladišno poslovanje).

[28] Tender je ponuda sudionika u natječaju (javno nadmetanje) koja se podnosi u zadanoj formi a sastoji se od komercijalnih uvjeta ponuđača, tehničke dokumentacije i bankarske garancije.

6.2.2. Prikupljanje zahtjeva

Utvrđivanje i razjašnjavanje zahtjeva često se naziva i prikupljanje zahtjeva. Utvrđivanje je možda prikladniji pojam jer često zahtjevi nisu eksplicitno prisutni u razmišljanju korisnika, niti su jasno predočeni informatičaru koji mora učiti i razumijevati područje budućeg sustava. Kod utvrđivanja zahtjeva važno je sagledati:

- granice sustava – utvrditi koje probleme budući sustav treba riješiti;
- uloge korisnika – utvrditi kategorije i znanje korisnika te njihove potrebe;
- ciljeve sustava – utvrditi globalno (eng. *high-level*) ciljeve koje sustav mora pružiti;
- poslovne zadatke – utvrditi kako korisnici obavljaju svoje zadatke i kako očekuju da će ih obavljati uvođenjem novog sustava te što time žele/trebaju postići.

U ovoj aktivnost je potrebno izraditi **početnu listu zahtjeva** koja treba konzistentno pratiti zadani opseg posla. Listu zahtjeva treba podržati i opisati modelima procesa i podataka visoke razine.

Svim metodama i tehnikama kojima se informatičari koriste da prikupe korisničke zahtjeve zajednička je bliska suradnja s korisnikom. Na samom početku prikupljanja zahtjeva informatičar ne zna ništa ili vrlo malo o svijetu za koji treba izraditi program. Stoga je podložan raznim utjecajima korisnika, jer se u dijagramima koji opisuju model poslovnog sustava nalaze i informacije upitne korisnosti i vjerodostojnosti. Uz često prisutnu proturječnost i zalihost u spomenutim modelima, informatičar još uvijek ne vlada dobro zadanim područjem te nije u stanju optimizirati prikupljene informacije.

Stoga je potrebno izraditi detaljnu analizu sustava rezultat koje će biti detaljno oblikovani korisnički zahtjevi.

6.2.3. Analiza i oblikovanje zahtjeva

Analiza i provjera valjanosti zahtjeva treba osigurati da korisnik na kraju dobije ono što mu je stvarno potrebno, a ne ono što želi/zahtijeva. U ovoj fazi su kritična pregovaranja s korisnikom oko ispravne definicije pojedinih zahtjeva i prepoznavanje logičkih cjelina u koje se zahtjevi mogu grupirati.

Glavne aktivnosti analize i oblikovanja zahtjeva:

- Prikupljanje informacija i izrada detaljne liste zahtjeva.
- Određivanje prioriteta među zahtjevima.
- Provjera valjanosti zahtjeva.

Prikupljanje informacija počinje podjelom odgovarajućih upitnika korisnicima te intervjuiranjem korisnika na jednom ili više radnih sastanaka. Upitnike uglavnom priprema projektant s ciljem da sazna što više o poslovanju.

Od korisnika se očekuje da pruži pregled odgovarajućih dokumenata pomoću kojih će opisati poslovne procedure. Analiziraju se i modeliraju tokovi podataka kroz sustav uz sveobuhvatno i optimizirano modeliranje podataka i procesa.

Projektantu sve zajedno služi za razumijevanje procedura, ograničenja i funkcionalnosti novog sustava, a rezultat bi trebao biti detaljan model novog sustava koji prati lista korisničkih zahtjeva.

Početnu listu zahtjeva iz aktivnosti utvrđivanja zahtjeva sada treba razraditi tako da sadrži:

- zahtjeve koji opisuju funkcionalnost sustava (funkcionalne zahtjeve);
- zahtjeve koji specificiraju oblikovanje novog sustava, kao što su razne metode optimizacije i algoritmi za rješavanje problema:
- nefunkcionalne zahtjeve poput arhitekture novog sustava, zahtjeva sigurnosti, kontrole i praćenja sustava.

Određivanje prioriteta među zahtjevima je aktivnost u kojoj treba odrediti koji zahtjevi su ključni za sustav i njima dati najveću prednost. Događa se da korisnik predlaže dodatne funkcionalnosti koje su poželjne, ali nisu neophodne. I korisnik i projektant trebaju se zapitati koje funkcije su stvarno važne, a koje nisu. Jedino projektant koji razumije organizaciju i njeno poslovanje može kvalitetno sudjelovati u određivanju prioriteta zahtjeva.

Praksa pokazuje da skoro uvijek postoje ograničenja u resursima i da je projektant osuđen na balansiranje između resursa i opsega (eng. *scope*) sustava. Ako to nije u stanju, zahtjevi, zahvaljujući korisniku, ekspandiraju i nastaje takozvani klizeći opseg (eng. *scope creep*).

Provjeru valjanosti zahtjeva provodi projektant zajedno s korisnikom na formalan način pregledom pisanih dokumenata (liste zahtjeva) i pratećim modelima sustava. Provjera valjanosti zahtjeva uglavnom je podržana izrađenim logičkim modelima sustava, izradom logičke strukture podataka i provjerom pomoću upita.

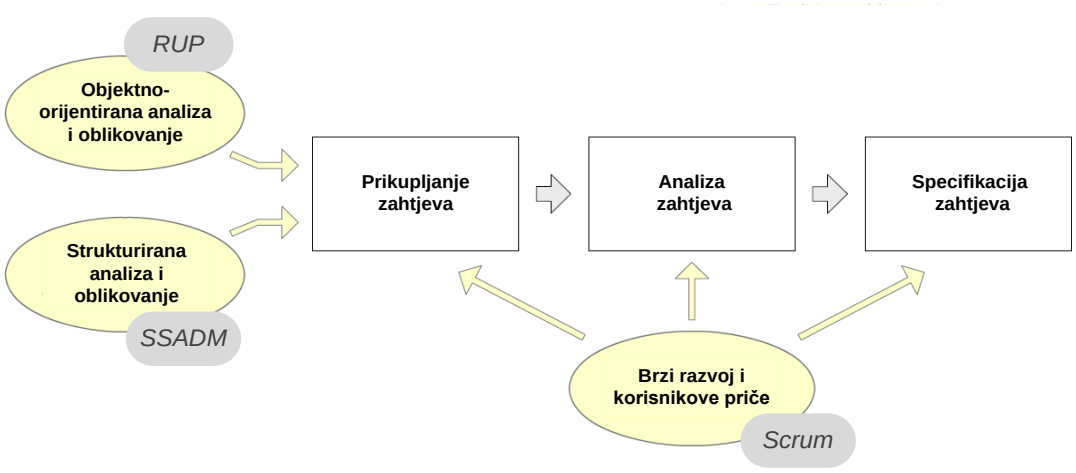
Problem je što oblikovanje i izrada novog sustava ovise o ispravnosti zahtjeva, a zasigurno je prekasno i preskupo provjeravati (testirati) ispravnost zahtjeva tek kada je završeno programiranje. Provjeru se može osigurati izradom modela koji opisuju zahtjeve za novi sustav. Često je dobro izraditi i *light* verziju prototipa kao svojevrsnu provjeru valjanosti zahtjeva i potvrdu informatičarima da su na dobrom putu.

6.3. Metode razvoja i aktivnosti zahtjeva

Opis korisničkih zahtjeva uvijek sadrži niz dokumenata koji su strukturirani kao tekstualne definicije te cijeli niz modela koji pomoću odgovarajućih tehnika modeliranja opisuju razne dijelove sustava i razna područja sustava.

6.3.1. Zahtjevi kroz modele i metode razvoja

Pristup modeliranju poslovnog sustava ili novog informacijskog sustava razlikuje se u tradicionalnim metodama, objektno-orijentiranim metodama i/ili nekim drugim, primjerice brzim (agilnim) metodama razvoja. Međutim, kako pokazuje slika 6.1., svaki od modela pristupa (pa tako i svaka od metoda; na slici je po jedan predstavnik za prikazane modele) treba proći kroz osnovne procese inženjerstva zahtjeva. To je prirodni ciklus savladavanja područja nekog znanja, a način na koji će to područje biti usvojeno i specificirano ovisi o metodi razvoja.



Slika 6.1. Zahtjevi kroz modele i metode razvoja

Strukturirana analiza i oblikovanje i objektno-orijentirana analiza i oblikovanje prolaze slijedno kroz faze inženjerstva zahtjeva. Za odgovarajuće metode (SSADM i RUP) bitno je naglasiti da su faze inženjerstva zahtjeva podržane modelima i specificiranom dokumentacijom koja prati svaku od metoda. Zato je i prirodno da se prelazi iz jedne faze inženjerstva zahtjeva u drugu, jer svaka prethodna faza ima kao rezultat niz modela koji se koriste za nastavak razvoja.

S druge strane, brzi razvoj programa ne drži previše do pisane dokumentacije pa je razumljivo da se nastoji što brže prikupiti zahtjeve i sagledati njihovo značenje i funkcionalnost koja će biti implementirana u programsko rješenje. U pregledu *Scrum* pristupa opisan je princip tog paralelnog sagledavanja svih faza inženjerstva zahtjeva. Pojednostavljeno, zahtjevi se nalaze u listi (*Product Backlog*) i dokle god nisu jasni, analizirani, specificirani, zajedno s procjenom rizika, oni se ne nalaze visoko u listi pa ne mogu doći na red u sljedećem radu na inkrementu proizvoda. U trenutku kada sve

bude jasno, zahtjev će postati kandidat za novu verziju proizvoda. U tom smislu su u brzom razvoju paralelno prisutne sve faze inženjerstva zahtjeva.

6.3.2. Tradicionalni i objektno-orijentirani pristup zahtjevima

Razlika između tradicionalnog i objektno-orijentiranog pristupa je u promatranju ponašanja sustava za vrijeme nekog događaja, odnosno kako sustav odgovara na taj događaj [21]. Tradicionalni pristup se razlikuje od objektno-orijentiranog pristupa u načinu na koji se sustav modelira i implementira.

Tradicionalni pristup gleda na sustav kao na skupinu procesa koje obavljaju ljudi ili računala. Opis procesa koje obavlja računalo vrlo je sličan uobičajenom načinu izrade računalnih programa koji sadrže naredbe i izvršavaju se sekvencijalno. Kada se izvršava pojedini proces on je u sprezi s podacima tako da čita podatke i nakon obrade nove vrijednosti upisuje natrag u bazu podataka. Proces može biti u međudjelovanju i s ljudima tako da traži od korisnika da unese podatke preko sučelja ili da prikaže korisniku informacije na sučelje.

Zna se da tradicionalni pristup opisa rada sustava uključuje procese, podatke, ulaze i izlaze. Kada se modelira kako sustav reagira na neki događaj i što sustav radi kako bi odgovorio na taj događaj, onda se izrađuju modeli procesa s naglaskom na njegove komponente.

SSADM metoda opisuje sustav na tradicionalan način modelirajući procese i podatke te tijek podataka kroz sustav (ulazne i izlaze).

S druge strane, objektno-orijentirani pristup vidi sustav kao skupinu objekata s uzajamnim djelovanjem. Objekti imaju sposobnost različitog ponašanja (tzv. metode) što im omogućava da uzajamno djeluju s ljudima koji se koriste sustavom ili međusobno. Jedan objekt traži od drugog objekta da nešto napravi tako što mu šalje poruku. Kada se modelira što sustav treba raditi kao odgovor na neki događaj, onda objektno-orijentirani pristup uključuje modele koji prikazuju objekte, njihovo ponašanje i međudjelovanje.

UML dijagrami uglavnom opisuju objekte sustava i njihovo međudjelovanje koje je posljedica poslanih poruka od jednog objekta/osobe ka drugom objektu/osobi, koji onda odgovara na te poruke.

U poglavlju 5. opisano je kako se inženjerstvo zahtjeva provodi kroz faze razvoja metoda SSADM i RUP. Zajedno s potrebnom dokumentacijom u svakom koraku te osvrtom na probleme koji prate analizirane aktivnosti izrađena je tablica 6.2. koja daje pregled svih tih elemenata.

Tablica 6.2. Inženjerstvo zahtjeva i problemi metoda razvoja

Inženjerstvo zahtjeva			SSADM		RUP		
Aktivnosti	**Područje**	**Modeli**	**faza**	**dokumenti/ proizvodi**	**faza**	**dokumenti/ proizvodi**	**Problem**
Studija izvedivosti	Poslovne potrebe Opseg projekta	Poslovanja	Studija izvedivosti	Početna lista problema/ zahtjeva	Početna faza (prva iteracija)	Dijagrami slučajeva korištenja poslovnog modela Sudionici s odgovornostima	Visoka razina modela s upitnom korisnosti za kasnije faze
Prikupljanje zahtjeva	Granice sustava Uloge korisnika Ciljevi poslovanja Poslovni zadaci	Postojećeg sustava	Studija izvedivosti Analiza poslovnog sustava	Razrađena lista problema/ zahtjeva DTP visoke razine Pregledni MEV	Početna faza (druga iteracija)	Većina dijagrama slučajeva korištenja Opis scenarija slučajeva korištenja Model objekata Dijagram slijeda	Nemogućnost razlučivanja bitnog od nebitnog Česti izostanak optimiziranja izrađenih modela
*Analiza i oblikovanje - **Prikupljanje informacija***	Odabrano područje poslovanja – postojeće stanje	Procesa Podataka Događaja	Analiza poslovnog sustava	Intervju s korisnikom DTP nižih razina Detaljni DEV Detaljni modeli procedura	Početna faza (druga iteracija) Elaboracija	*svi dijagrami iz prethodne ćelije* Popis rizika složen po prioritetima	Intervjui s korisnikom i dokumenti poslovanja mogu ugroziti dinamiku realizacije plana projekta
*Analiza i oblikovanje - **Određivanje prioriteta***	Odabrano područje poslovanja – novi sustav	Procesa	Specifikacija zahtjeva	Nadopunjena lista problema/ zahtjeva Nadopunjeni DTP nižih razina	Početna faza (druga iteracija) Elaboracija	Detaljno oblikovanje (*svi dijagrami iz prethodne ćelije*) odabranih zahtjeva za jednu iteraciju	Ograničenje resursa Korisnikove želje
*Analiza i oblikovanje - **Provjera valjanosti***	Odabrano područje poslovanja – novi sustav	Procesa Podataka Događaja	Specifikacija zahtjeva	Normalizirani DEV Cjeloviti DTP nižih razina Matrica ŽCE Upiti za provjeru	Početna faza (druga iteracija) Elaboracija Konstrukcija	Izgrađen kostur sustava Ugrađeni rizici najvišeg prioriteta	Od strane korisnika provjera prijedloga rješenja novog sustava

Mogući problemi, koji nastaju kada se pristupi slijednom i detaljnom specificiranju zahtjeva pomoću modela sustava i prateće dokumentacije, mogu se analizirati i pomoću ove tablice. Neke od problema moguće je minimizirati i/ili riješiti metodama brzog razvoja programa. Ispunjenjem preduvjeta, kao što je prikladan broj specijalista i njihova visoka stručnost, primjer *Scruma* može ublažiti probleme sudjelovanja i zadovoljstva korisnika, optimiziranja modela sustava i rješenja, i što je najvažnije, upuštanja u oblikovanje i izradu programskog proizvoda na osnovu loše specifikacije zahtjeva.

Pitanja za ponavljanje

1. *Kada se kroz ciklus razvoja IS-a odvija specifikacija zahtjeva i koja je njena namjena?*
2. *Je li nužno dobra specifikacija zahtjeva garancija za uspjeh IS-a? Obrazložite odgovor.*
3. *Nabrojite aktivnosti kroz koje prolazi inženjerstvo zahtjeva?*
4. *Objasnite aktivnosti i rezultate studije izvedivosti.*
5. *Objasnite aktivnosti i rezultate prikupljanja zahtjeva.*
6. *Koje su glavne aktivnosti faze prikupljanja zahtjeva?*
7. *Objasnite korak prikupljanja informacija od korisnika. Kako se provodi taj korak i koji je njegov rezultat?*
8. *Objasnite korak određivanja prioriteta među zahtjevima.*
9. *Objasnite korak provjere valjanosti zahtjeva.*
10. *Kako metode RUP, SSADM i Scrum pristupaju procesu inženjerstva zahtjeva? Objasnite sličnosti i razlike.*

7. Analiza sustava

Funkcionalnost sustava
Podaci i informacije
Razvojna okolina

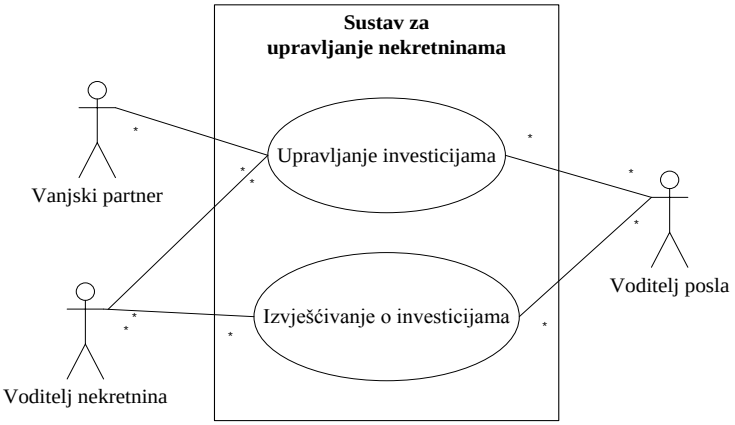
7.1. Funkcionalnost sustava

Analizirajući funkcionalne zahtjeve IS-a koristimo se specifičnim modelima kako bi opisali spoznaje o zahtjevima i zapisali informacije o sadržaju (struktura i ponašanje) zahtjeva. Virtualno gledano, analitički pristup na početku razvoja sustava započinje modeliranjem procesa.

Modeliranje procesa sustava započinje s konceptima i događajima (eng. *event*). Događaji predstavljaju opis procesa tako da su naznačene specifičnosti vremena i mjesta odvijanja procesa. Događaji pokreću ili iniciraju svaki proces, te aktivnosti odnosno stanja unutar procesa.

Kod definiranja zahtjeva treba analizirati koji događaji imaju utjecaj na sustav koji se istražuje, odnosno koji su događaji rezultat odgovora sustava. Događaji koji utječu na ponašanje sustava donose i informacije (podatke) koje sustav obrađuje. Događaji koji su rezultat odgovora sustava nose izlazne podatke i informacije.

***PRIMJER*:** Razmatra se sustav za upravljanje nekretninama gdje je početak analize opisan dokumentom u kome su naglašeni namjena, opseg i cilj projekta razvoja IS-a koji će podržati ovaj sustav.



Slika 7.1. Opseg sustava za upravljanje nekretninama

Pretpostavlja se da je iz razgovora s korisnikom, odnosno iz strateškog plana proizašao ovaj dokument (tablica 7.1.), a uz njega ide i grubi model sustava (slučaj korištenja visoke razine, slika 7.1.)

Tablica 7.1. Namjena, opseg i cilj projekta

DOKUMENT SPECIFIKACIJE SUSTAVA		
Područje:	Upravljanje nekretninama	
Datum:		
Odgovorna osoba:	KK	
Namjena:	Sustav za upravljanje nekretninama korisniku treba omogućiti evidentiranje podataka o nekretninama: kupnju, prodaju, unajmljivanje i prestanak najma. Iz sustava upravljanja nekretninama korisnik očekuje izvješća o poslovanju i izvješća o vođenju nekretnina. Korisnik mora biti u mogućnosti upravljati sredstvima u smislu osiguranog portfelja (visine zarade, tj. dohotka). Da bi se u odlučivanju i upravljanju izbjegle razne proračunske tablice investitor očekuje od sustava da automatski bilježi i obavještava o svim promjenama povezanim s određenom investicijom. Tako bi opseg sustava uključio mogućnost predbilježbe i dodjele lokacije, proizašli novčani tijek povezan s određenom nekretninom i izvorište novčanog tijeka te mogućnost izračuna povrata uložene investicije.	
Opseg:	Matični podaci:	registar nekretnina, grad/općina, poslovni partner, najmoprimac
	Funkcionalnosti:	import (Učitavanje) podataka, iznajmljivanje nekretnina, Izrada plana otplate
	Izvješća:	Izvješće o isteku najma, Izvješće o korištenju imovine, Izvješće o očekivanom povratu novca, Izvješće o deset najisplativijih nekretnina, Izračun trenutne rate
Izvori informacija:	Korisnik, agencije za prodaju i najam nekretnina, zakoni i pravilnici, algoritam za obračun kamata, ostala literatura.	

Važnost prepoznavanja događaja koji podržavaju i prate procese poslovanja prepoznaje se u činjenici da IS postaju sve više interaktivni, a korisnici očekuju da na akciju dobiju odgovor u realnom vremenu.

Može se primijetiti da se i tradicionalni pristup (uključujući strukturnu analizu i informacijski inženjering) koristi konceptom događaja, dok je u okviru objektno-orijentiranog pristupa događaj osnovna pretpostavka funkcioniranja sustava.

7.1.1. Tipovi događaja

Ovdje se razmatraju tri tipa događaja. To su vanjski događaji, vremenski događaji i događaji stanja. Analiza poslovanja započinje nastojanjem da se prepozna i popiše što više ovih događaja.

Vanjski događaji pojavljuju se izvan promatranog sustava i najčešće su pokrenuti od strane vanjskih agenata (eng. *agent*) ili sudionika (eng. *actor*). Vanjski agenti (sudionici) su osobe ili organizacije koje opskrbljuju sustav podacima ili iz sustava dobivaju podatke.

> Primjer vanjskog agenta je kupac koji želi naručiti neki proizvod. Ovaj vanjski događaj je osnova za pokretanje procesa naručivanja.
>
> Drugi primjer je kada ljudi ili organizacije unutar poduzeća zahtijevaju nešto od sustava. Primjerice, kada u procesu naručivanja voditelj prodaje traži provjeru statusa narudžbenica jer želi izaći u susret ključnim kupcima.

Vremenski događaji nastaju u nekom zadanom vremenu. Veliki broj informacijskih sustava proizvodi izlazne informacije u nekim definiranim intervalima.

> Primjerice, danas, ovisno o dogovoru s klijentom, banke i druge uslužne institucije poput telekomunikacija, svojim klijentima mjesečno šalju račune elektroničkim putem, poput e-mail poruke s računom u privitku.

U ovom slučaju nema vanjskog agenta (sudionika) koji na zahtjev od sustava dobiva izlazne informacije, nego se to događa mimo njihove *volje*.

O vremenskim događajima analitičar saznaje ukoliko postavi pitanja o specifičnim rokovima (eng. *deadline*) koje sustav treba ispuniti. Također je važno saznati koje izlazne informacije i ostale aktivnosti se očekuju u zadanim rokovima. O ovoj vrsti događaja može se saznati i ispitivanjem što sustav mora proizvesti u zadanom vremenu.

> Primjerice, ukoliko je prihvaćena narudžba od kupca, treba u periodu od 7 dana isporučiti robu.

Događaji stanja su prisutni u situacijama kada se nešto zbiva unutar sustava te je potreban okidač (eng. *trigger*) da se proces pokrene.

> Primjerice, ako je količina artikala u skladištu pala ispod minimalno zadane, treba pokrenuti proces nabave tog artikla.

Ponekad je logika ovog tipa događaja slična vremenskom događaju, samo u ovom slučaju vrijeme nije unaprijed zadano.

7.1.2. Prepoznavanje događaja

Nije uvijek lako prepoznati događaje koji utječu na sustav. Neke od smjernica kojih se analitičar može držati razmišljajući o procesu poslovanja dane su u nastavku.

Događaji u odnosu na uvjete i odgovore: često nije jednostavno razlučiti korake (sekvence) stanja i uvjeta koji prethode nekom događaju. Primjer u nastavku pokazuje kako se u analizi nekog procesa može evidentirati cijeli niz koraka koji prethode događaju od važnosti za sustav.

***PRIMJER*:** Osoba (Ana) želi kupiti neki odjevni predmet, primjerice hlače. Niz stanja u kojima će se naći osoba i njena konačna odluka (događaj) koja pokreće proces kupnje opisani su u sljedećem slučaju:

- Ana razmišlja o kupnji hlača
- Ana je sjela u automobil i odvezla se u trgovinu A
- U trgovini A Ana je isprobala hlače, no nije ih kupila jer su joj premale
- Ana je otišla u trgovinu B
- U trgovini B Ana je isprobala hlače
- Ana je odlučila kupiti hlače u trgovini B.

Samo zadnji korak prikazanog popisa stanja koja prate Aninu želju za kupnjom hlača predstavlja događaj koji pokreće kupnju (izdavanje računa i robe) u **sustavu prodaje** trgovine B.

Događaj iz primjera predstavlja vanjski događaj u odnosu na cjelinu, to jest, funkcionalnost procesa prodaje.

Ako i dalje analiziramo proces prodaje, onda se u ovom primjeru unutar procesa prodaje mogu pojaviti i dodatni događaji koji će utjecati na konačni rezultat procesa.

> Primjerice, ako osoba plaća kreditnom karticom ili čekom, trgovina će knjižiti stvarni tijek novca tek onda kada od banke naplati karticu ili ček. Ako osoba plaća gotovinom, knjiženje novca se događa odmah po izdavanju računa.

U nastavku na ovom primjeru mogu se uočiti događaji stanja koji uz ispunjene uvjete (kartica ili ček ili novac) pokreću skup aktivnosti unutar procesa. Ukoliko se naplate od banke događaju u nekom zadanom vremenu ili u vremenskim intervalima, utoliko se u tom slučaju može govoriti o vremenskim događajima.

Redoslijed događaja: prati životni ciklus promatranog procesa poslovanja. Radi prepoznavanju događaja često je korisno pratiti korake događaja. Primjer u nastavku prikazuje korake koje izvodi vanjski agent (kupac) prilikom kupnje artikala putem kataloške prodaje.

***PRIMJER*:** Kupac kupuje artikle iz kataloga, a koraci te transakcije su slijedeći:

- Kupac zatraži prodajni katalog od robne kuće
- Kupac u katalogu pronalazi nekoliko artikala koje želi kupiti.
- Kupac predaje narudžbu sa popisom artikala.

- Kupac se predomišlja oko naručenih artikala.
- Kupac mijenja sadržaj narudžbe ili otkazuje narudžbu.
- Kupac se informira o statusu narudžbe (zaprimljeno, zapakovano, poslano, kada će doći?).
- Kupac vraća jedan od artikala zbog greške.

Napomena: neki od navedenih koraka mogu se odvijati ručno, telefonski ili web prodajom, što opet dodatno komplicira analizu procesa poslovanja i prepoznatih događaja.

Dakle, pokazani proces prodaje artikala kataloškim putem aktiviran je događajem izrade narudžbe artikala, no uz taj događaj postoje i prethodni događaji (naručivanje i isporuka kataloga) te cijeli niz događaja unutar samog procesa prodaje.

Iz primjera je važno uočiti da odvijanju procesa poslovanja prethodi ispunjenje jednog ili više događaja. Zatim se unutar procesa pojavljuje cijeli niz događaja koji onda pokreću razne dijelove procesa.

Proces kao cjelina treba imati ugrađene razne varijante pokretanja i izvršenja procesa. Stoga je važno prilikom oblikovanja novog sustava i izrade programske podrške nekom poslovnom procesu uzeti u obzir i ugraditi sve varijante procesa, praćeno do njegova završetka kada će završni događaji rezultirati različitim izlaznim informacijama.

Tehnološki zavisni događaji i sistemske kontrole: analitičar treba razmotriti i događaje koji su važni za sustav, ali se izravno ne odnose na poslovne transakcije. Vjerojatno je da će tijekom analize takvi događaji biti zanemareni jer je tada važno proučiti funkcionalne zahtjeve.

Takvi događaji su oni koji se odnose na sistemske kontrole koje postaju važne kod implementacije sustava kao što su, primjerice, sigurnosne kontrole, prijava (*logiranje*) na sustav, zaštita i integritet baze podataka (svakodnevni *backup*) te prava pristupa pojedinih vrsta korisnika i slično.

Kada se analizira sustav, onda se polazi od idealnih tehnoloških pretpostavki da će sustav biti u mogućnosti odgovoriti na događaje, da se neće događati prekidi rada, prekidi komunikacije, problemi u kapacitetu resursa i, na koncu, da ljudi koji će raditi sa sustavom neće nikada pogriješiti. Primjer u nastavku pokazuje niz događaja na koje ne treba obraćati pažnju u analizi, ali ih svakako treba uzeti u obzir i u oblikovanju novog sustava.

PRIMJER: O ovim događajima treba voditi računa i uključiti ih (te riješiti problematične situacije) prilikom dizajna sučelja, konfiguracije, baze podataka, komunikacije i slično:

- Korisnik se želi prijaviti na sustav.
- Korisnik želi promijeniti lozinku.

- Vrijeme odziva korisnika kod promjene lozinke.
- Pad sustava zahtijeva oporavak (eng. *recovery*) baze podataka.
- Vrijeme za rezervnu kopiju (eng. *backup*) baze podataka.
- Korisnik želi promijeniti zadane postavke sustava.

7.2. Podaci i informacije

U analizi i definiranju zahtjeva osim događaja važnu ulogu imaju i koncepti poput dokumenata, odgovornih osoba, organizacije, lokacija, uređaja i mnoštva drugih opipljivih i neopipljivih stvari za koje treba voditi podatke. To su prije svega koncepti važni za tijek poslovnih procesa, primjerice, ako za sustav materijalnog poslovanja korisnik vodi podatke o klijentima i artiklima, onda je važno da analitičar sustava prepozna i evidentira sve potrebne informacije o njima.

U tradicionalnom pristupu ovi koncepti su sastavni dio baze podataka u koju se pohranjuju podaci. Tip podatka, ograničenja i međusobna povezanost podataka koje treba pohraniti ključni su elementi zahtjeva sustava.

U objektno-orijentiranom pristupu spomenuti koncepti su objekti koji su u međudjelovanju (eng. *interact*) sa sustavom ili međusobno.

Bez obzira na to kojim pristupom se razvija IS, potrebno je prepoznati i razumjeti koncepte koji su sastavni dio odvijanja poslovnih procesa i događaja nekog sustava.

7.2.1. Prepoznavanje koncepata

Analitičari će u postupku istraživanja zahtjeva sustava pregledati mnoge izvore informacija da, za početak, sastavi listu koncepata koje sustav treba kako bi čuvao i koristio se potrebnim podacima i informacijama.

Često se kaže da u početku treba pažljivo sagledati sve *imenice* koje korisnik spominje kada opisuje kako sustav radi. Pri tome treba uzeti u obzir (prije spomenute) događaje, aktivnosti, vanjske agente i sudionike, te okidače događaja i odgovore koji su rezultat događaja, kako se ne bi propustila pojavljivanja važnih koncepata.

Kada se na osnovu pregledanih područja poslovnih procesa dođe do liste koncepata, potrebno je tu listu pročistiti i odlučiti što će ostati u njoj, a što je nepotrebno. Slijedeće smjernice (pitanja i odgovori) mogu pomoći u ovoj odluci.

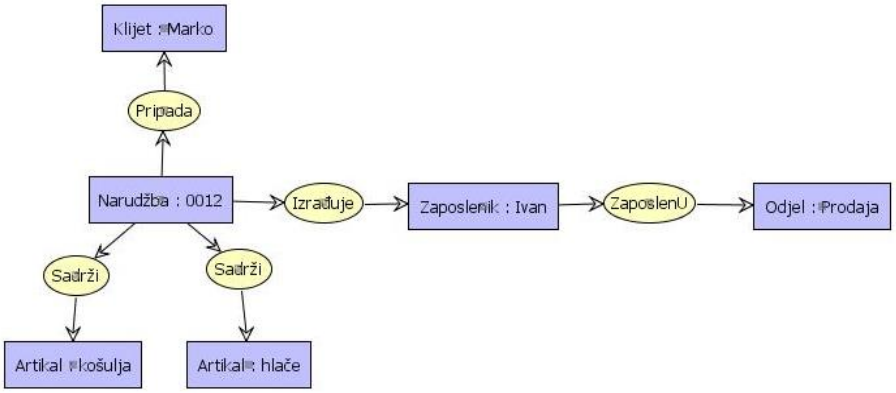
- Pitanja na koja potvrdan odgovor znači da te koncepte treba uključiti u sustav:
  - Je li koncept jedinstven u sustavu?
  - Je li korištenje koncepta u zadanom okviru promatranog sustava?
  - Treba li sustav pamtiti više pojavljivanja (stavki) pojedinog koncepta?

- Pitanja na koja potvrdan odgovor znači da te koncepte ne treba uključiti:
  - Je li prepoznati koncept sinonim za neki koji je već uključen u listu?
  - Je li prepoznati koncept samo skup izlaznih informacija proizveden iz postojećih informacija i podataka koji su već na listi koncepata?
  - Sadrži li prepoznati koncept ulazne podatke i informacije koje se već nalaze u sustavu?
- Pitanja na koja potvrdan odgovor znači da te koncepte treba dodatno istražiti:
  - Je li vjerojatno da je prepoznati specifični podatak (atribut) o nekom konceptu već u sustavu?
  - Ako se neke pretpostavke zahtjeva sustava promijene, je li potrebno još informacija osim onih koje su evidentirane?

7.2.2. Veze među konceptima

U okviru poslovnih procesa veza među konceptima je prirodno pridruživanje koje nastaje kao posljedica odvijanja aktivnosti pojedine transakcije i potrebe da se podaci i informacije o tijeku i rezultatu transakcije povežu i tako povezane zabilježe.

***PRIMJER*:** Proces naručivanja artikala od strane klijenta zabilježen pomoću konceptualnog grafa[29] prikazan je na slici 7.2.



Slika 7.2. Konceptualni graf za proces naručivanja

Primjer pokazuje koncepte i njihove veze (relacije) procesa naručivanja za klijenta Marka. Proces započinje događajem naručivanja košulje i hlača. Informacija o ovom događaju zabilježena je u narudžbenici broj 0012 koja predstavlja vezu između Marka i artikala. Također, ta narudžbenica povezuje i zaposlenika Ivana, koji radi u odjelu prodaje i zaprimio je narudžbu od klijenta.

[29] Konceptualni grafovi (eng. *Conceptual Graphs*) su tehnika prikaza znanja koju prvi put koristi John F. Sowa još 1976. godine za prikazivanje konceptualne sheme baze podataka.

Važno je uočiti da pretpostavljeni koncepti i njihove veze vrlo vjerojatno predstavljaju i osnovne elemente baze podataka programskog rješenja ovog problema.

Iz prethodnog primjera treba dodatno analizirati neke važne momente povezane s prepoznavanjem podataka i informacija, a povezano je i s pitanjima iz prethodnog potpoglavlja koja postavlja analitičar.

Pojavnost u vezi je važna budući da istraživanje postojanja koncepta u vezi određuje njegovu brojnost. Potrebno je čitati vezu u oba smjera.

Primjerice, veza između klijenta i narudžbenice nije jednostavna kako se čini na prvi pogled. Za narudžbenicu je potrebno obvezno evidentirati klijenta, pa je je veza u smjeru *narudžbenica* → *klijent* obvezna. Međutim, klijenti koji su evidentirani u sustavu ne moraju imati niti jednu narudžbenicu. Naime, ako su to klijenti s kojima poduzeće obavlja i neke druge poslove (a prema pitanjima prethodnog potpoglavlja koncept klijenta bi trebao biti evidentiran samo na jednom mjestu), onda gotovo sigurno postoje klijenti koji nemaju niti jednu narudžbenicu. Stoga veza u smjeru *klijent* → *narudžbenica* nije obvezna.

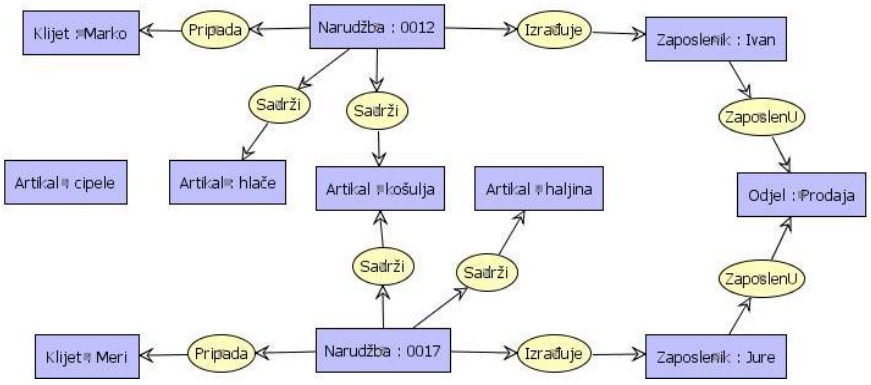
Brojnost (eng. *cardinality*) koncepta u vezi označava broj pojavljivanje svakog od koncepata. Iz teorije baza podataka poznato je da brojnost može biti: *jedan prema jedan*, *jedan prema više* i *više prema više*.

U primjeru na slici 7.2. prisutne su neke od ovih veza. Može se pročitati da, primjerice, jedna narudžba pripada jednom klijentu, jedan zaposlenik je zaposlen u jednom odjelu, ali isto tako i da jedna narudžba ima jedan, dva ili više artikala.

Primjer 7.2. nije dovoljan da bi se uočile zakonitosti veza među konceptima te pojavnost i brojnost koncepata u vezama među njima. Zato analitičar treba ispitati više događaja pojedinog procesa poslovanja i iz njih izvesti opća pravila i zaključke.

Primjer u nastavku prikazuje nekoliko događaja naručivanja koji mogu dati obrazac pojavnosti koncepata, njihovih veza i brojnost koncepata u tim vezama.

***PRIMJER*:** Na slici 7.3. naznačena su dva događaja za dvije narudžbe, iz čega se vidi raznolikost veza među konceptima.



Slika 7.3. Konceptualni graf za više narudžbi

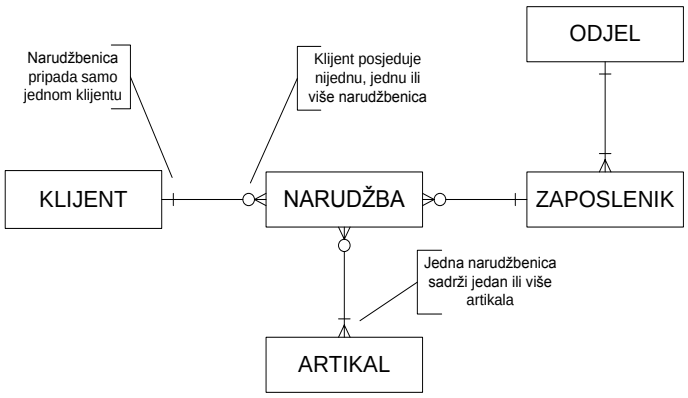
Iz primjera se još (uz prethodne zakonitosti sa slike 7.2.) može vidjeti da: u pojedinom odjelu radi jedan ili više zaposlenika; jedan artikl se može nalaziti na nijednoj, jednoj ili više narudžbenica.

Ako bismo na slici 7.3. dodali još saznanja za dodatne primjere događaja naručivanja artikala, tada bismo mogli zaključiti i da: pojedinom klijentu pripada nijedna, jedna ili više narudžbi; pojedini zaposlenik je izradio nijednu, jednu ili više narudžbi.

7.2.3. Entiteti i atributi

Entitet je skup objekata iz realnog svijeta koji imaju neka zajednička svojstva. Svojstva entiteta se nazivaju atributima.

Analitičari, kao i mnogi informatičari, nakon što izvrše analizu i prepoznaju obrazac ponašanja u procesu naručivanja artikala, mogu izraditi logički model podataka, primjerice modelom E-V (vidi potpoglavlje 4.2.4.). Na slici 7.4. je prijedlog E-V modela za koncepte iz prethodna dva primjera.

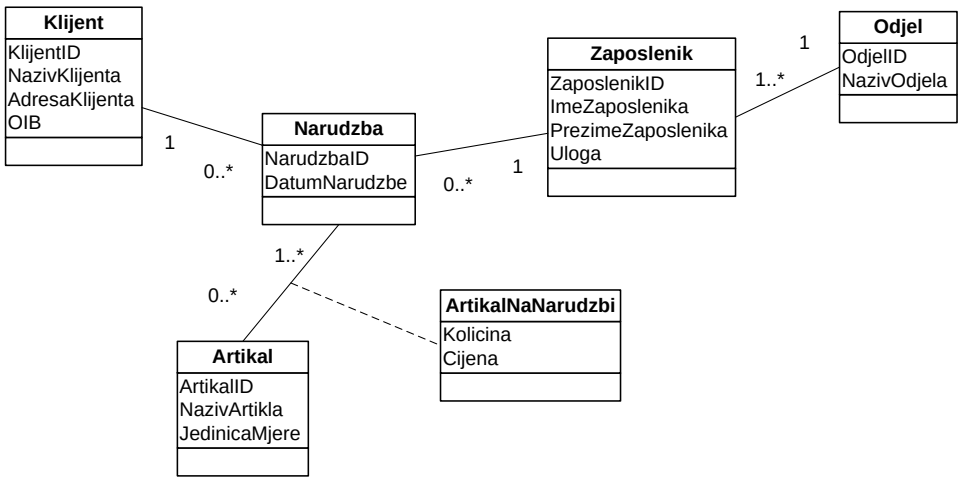


Slika 7.4. Model E-V za proces naručivanja robe

Za svaki entitet je potrebno evidentirati jedan ili više pripadajućih atributa. Prateći detaljno događaje procesa naručivanja artikala uočio bi se veliki broj atributa. Neki su i naznačeni na slikama 7.2. i 7.4. Vjerojatno za klijenta treba voditi podatke o nazivu, adresi, OIB-u, žiro-računu, za narudžbenicu je važan broj i datum, a recimo za artikl treba voditi podatke o šifri, nazivu i jedinici mjere.

E-V model se uglavnom koristi u tradicionalnom pristupu razvoju (pa tako i analizi) programske podrške. Navedeni primjeri atributa, te još dodatni potrebni za naručivanje, prikazani su u slijedećem primjeru, ovaj put objektno-orijentiranim pristupom pomoću dijagrama klasa.

***PRIMJER*:** Na slici 7.5. prikazan je dijagram klasa za proces naručivanja artikala, zajedno sa pripadajućim atributima.



Slika 7.5. Dijagram klasa za proces naručivanja robe

7.3. Razvojna okolina

Osim definiranja funkcionalnih zahtjeva na način da se modeliraju koncepti poslovnog sustava i tako specificira opseg i namjena, treba analizirati još neke tehničke elemente koji će biti sastavni dio zahtjeva a time i budućeg IS-a.

Potrebno je razmotriti računalnu opremu, operacijske sustave, mrežne mogućnosti i slično. Novi sustav će biti kompleksan skup programa i baza podataka, možda sa velikim brojem *batch* transakcija (primjerice, obrade u bankama). Treba odgovoriti i na pitanja koliko korisnika će imati novi sustav, na koliko lokacija će biti instalirana računalna oprema, koliko su te lokacije udaljene, koja prava pristupa će se definirati s obzirom na pristup računalima i na pristup dijelovima IS-a s kojima će korisnik raditi.

Odgovori na ova, i na još niz važnih pitanja, vrlo vjerojatno će odrediti okolinu u kojoj će se programi razvijati. Stoga je važno odrediti i detaljno definirati okolinu, a taj dio specifikacije se još nazivaju i **nefunkcionalni** zahtjevi (za razliku od takozvanih funkcionalnih zahtjeva koji opisuju područje poslovanja). Bilo da je riječ o velikim ili o malim sustavima, moguće su različite konfiguracije: centralizirani sustavi, distribuirani sustavi, internet, intranet, ekstranet. Osim toga, treba odabrati standardne programe i alate koji će podržati IS.

Okolina sistemskih programa uključuje operacijske sustave, mrežne protokole, sustave za upravljanje bazama podataka, i slično. U razvoju nekih sustava izbor okoline može biti proizvoljan, dok je u drugima potrebno prilagoditi se postojećim sustavima koji su u funkciji u poduzeću.

Za analitičara je u fazi analize važno odrediti komponente okoline koje će podržati i kontrolirati sustav koji se razvija. Neki od tih komponenti opisane su u nastavku:

- **Programski jezik razvoja**

  Sigurno da će informatička kuća zagovarati programske jezike koji se već koriste i za koje ima stručne ljude. Ipak, kako napreduje tehnologija tako nove programske platforme nude elemente za bolju produktivnost, iako su sve složenije za korištenje. Ipak, ako je potrebno da novi sustav ima i mogućnost elektroničkog poslovanja i potrebu za Internet programima, onda se informatičari trebaju obučiti za programske jezike koji omogućavaju te funkcionalnosti.

- **CASE alati i metodologije**

  Ako informatičko poduzeće ima CASE alat, onda je razvoj sustava podređen korištenju metodologije razvoja koju pruža taj alat. Neki CASE alati imaju mogućnosti generiranja programskog koda iz modela korištene metodologije. U tom slučaju, analitičar treba voditi računa o mogućnostima i ograničenjima takve okoline.

- **Zahtijevano sučelje prema drugim sustavima**

  Rijetko se događa da poduzeće zamjenjuje cijeli postojeći sustav novim IS-om. Samo u tom slučaju bi bilo svejedno kakva programska i hardverska okolina je specificirana za novi sustav. Međutim, najčešće novi sustav treba primati i davati informacije dijelovima već postojećeg sustava, te je stoga važno osigurati nesmetanu komunikaciju među sustavima. Često se zahtijeva da ovaj proces integracije novog s postojećim sustavom bude tretiran kao novi mini-projekt i vođen neovisno o vođenju projekta razvoja IS-a.

- **Okolina operacijskog sustava**

  I okolina operacijskog sustava ima važan utjecaj na oblikovanje i implementaciju novog IS-a. Često u strategiji razvoja novog sustav postoji potreba za mijenjanjem okoline ili kombiniranjem više okolina sa složenim zahtjevima prema komunikaciji i sučeljima.

- **Okolina sustava za upravljanje bazama podataka**

  Ovaj dio je također važan za analitičara, i to ne samo zbog analize i specifikacije koja će pripremiti detalje sustava za buduće oblikovanje i izradu, nego i zbog drugih detalja. Ponekad se poduzeće opredijeli za jednog dobavljača DBMS-a. Ako informatičko poduzeće već razvija slične sustave kao što traži i naručitelj, analitičar treba biti spreman prepoznati i uskladiti moguće razlike između DBMS sustavima i osigurati ispravnu strukturu, ograničenja i mogućnosti komunikacije korisnikovih baza podataka i baza podataka novog sustava.

Pitanja za ponavljanje

1. *Koje elemente sustava je potrebno uočiti kada se započinje analiza poslovanja?*
2. *Skicirajte namjenu, opseg posla i ciljeve sustava za prodaju.*
3. *Objasnite pojam događaja u odnosu na uvjete i odgovore.*
4. *Zašto je za analizu značajno prepoznavanje redoslijeda događaja?*
5. *Kako tehnološki zavisni događaji i sistemske kontrole utječu na izgradnju novog sustava?*
6. *Koje skupine podataka je važno evidentirati na početku analize poslovanja?*
7. *Na koja pitanja treba odgovoriti kada se pročišćava lista koncepata sastavljena u analizi poslovanja?*
8. *Objasnite pojavnost i brojnost koncepata u vezi.*
9. *Što se prikazuje modelom entiteti-veze?*
10. *Skicirajte model entiteti-veze za sustav prodaje.*
11. *Skicirajte model klasa za sustav prodaje.*

8. Oblikovanje sustava

Glavne komponente i razina oblikovanja sustava
Oblikovanje programa
Oblikovanje baza podataka

8.1. Glavne komponente i razina oblikovanja sustava

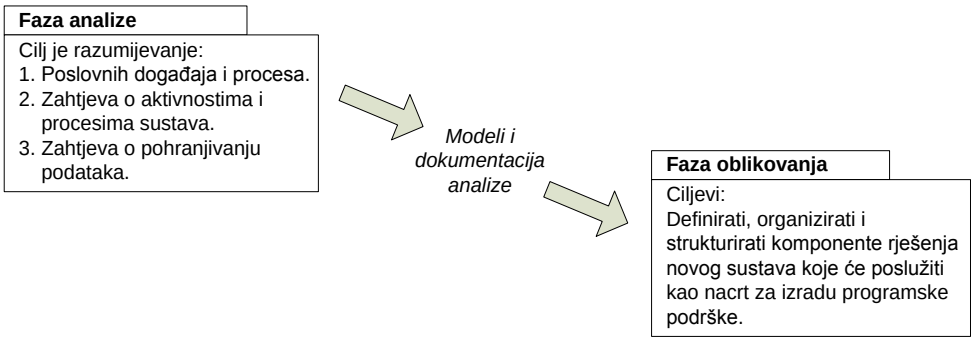
Dok se analiza fokusira na zahtjeve koji opisuju **što** bi sustav trebao raditi, oblikovanje je usmjereno ka nastojanju da se opiše **kako** će sustav raditi. Za opis rada sustava informatičari primjenjuju strukturni pristup definiranju i oblikovanju komponenti sustava.

Elementi definiranja programske razvojne okoline započeti su u fazi analize (potpoglavlje 7.3.) i dalje se nastavljaju kompletirati i detaljizirati u fazi oblikovanja novog sustava.

8.1.1. Elementi i modeli oblikovanja sustava

Procesi oblikovanja uključuju opise, organizaciju i strukturiranje komponenti sustava kroz razine arhitekture i detalja od kojih će se konstruirati predloženi novi sustav (slika 8.1.).

> Primjerice, oblikovanje u razvoju IS-a je konceptualno slično izradi nacrta za gradnju zgrade. U nacrtima je detaljno opisana unutrašnja struktura zgrade, prostori, sobe, opisi za zidove, prozore, razvođenje struje, vode i ostali detalji. U oblikovanju IS-a rade se isti postupci samo su ovdje to komponente programske podrške.



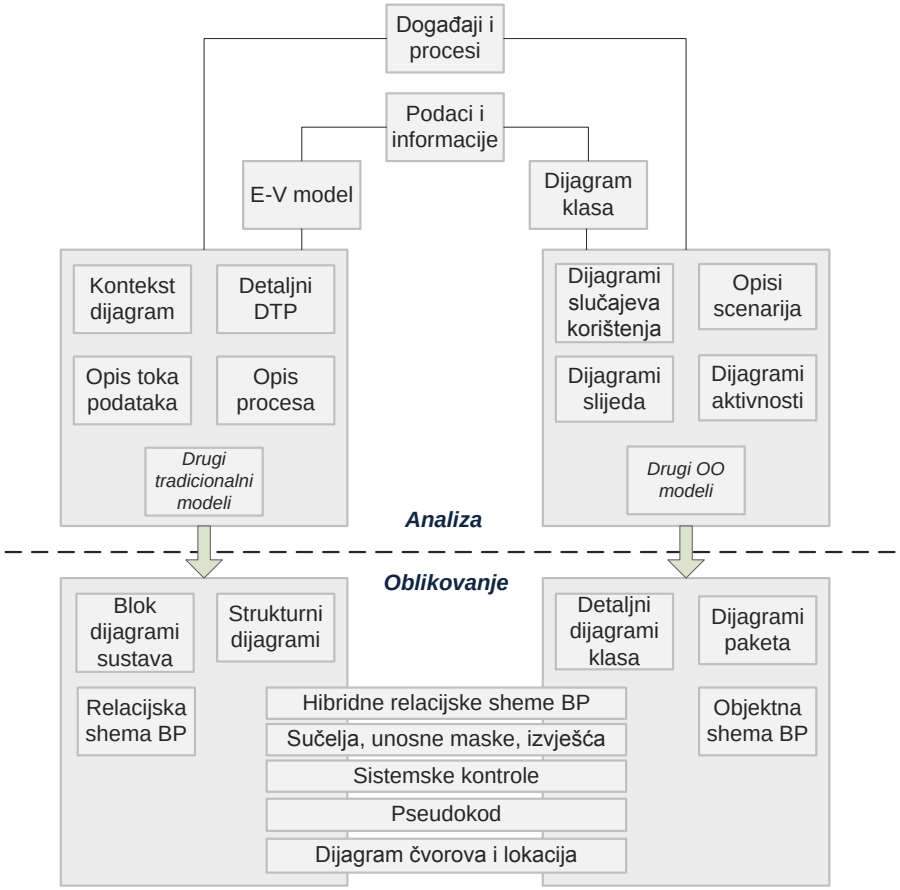
Slika 8.1. Od ciljeva analize k ciljevima oblikovanja

Oblikovanje je orijentirano k tehničkim detaljima te je u ovoj fazi korisnik manje uključen, a više su uključeni analitičari i drugi informatičari.

Kako bi analitičari razumjeli različite elemente oblikovanja treba razmotriti tri pitanja:

- Od kojih komponenti će se graditi novi sustav?
- Koji su ulazi i izlazi procesa oblikovanja novog sustava?
- Kako se radi oblikovanje sustava?

Oblikovanje i analiza su potpomognuti modelima i dijagramima koji predstavljaju detaljno opisane različite situacije funkcionalnosti novog sustava. Na slici 8.2. [21] pokazan je pregled dijagrama (a time i područja sustava koji je opisan u dijagramu) za dva pristupa koja su do sada već u nekoliko navrata bila razmatrana: tradicionalni pristup i objektno-orijentirani pristup.



Slika 8.2. Modeli tradicionalnog i objektno-orijentiranog pristupa u analizi i oblikovanju

Na slici 8.2. je vidljivo da modeli poslovnog sustava, izrađeni u fazi analize, služe kao podloga za oblikovanje modela novog IS-a. Modeli analize su svojevrsna riznica znanja o poslovanju koju će projektanti novog sustava prilagoditi karakteristikama

razvojnih alata. Za oblikovanje baza podataka najčešće se koriste relacijski modeli. Za objektno-orijentiranu strukturu može se izraditi ili relacijska ili objektna baza podataka (ovisi o razvojnim alatima).

Zanimljivo je osvrnuti se na pitanje mogu li se kombinirati strukturne i objektno-orijentirane tehnike. Može li se dogoditi da se u razvoju IS-a napravi strukturna analiza poslovanja, a nakon toga da se pristupi objektno-orijentiranom oblikovanju i razvoju programa (ili obrnuto)?

Ponekad je moguće izraditi tradicionalnu analizu jer ta faza predstavlja usvajanje i specifikaciju znanja nekog područja poslovanja. Pripadajući tradicionalni modeli su dovoljno općeniti, te se mogu koristiti u nastavku razvoja. Nastavak može uslijediti s OO oblikovanjem i izradom jer je oblikovanje stvaranje novog sustava. Informatičari s iskustvom u stanju su znanje usvojeno na temelju analize pretočiti u sučelja i podršku korištenog OO alata za razvoj programa. Ipak, ovakvo miješanje nije preporučljivo jer se tradicionalni pristup zasniva na procesima sustava, a OO tehnike su zasnovane na međudjelovanju objekata.

8.1.2. Aktivnosti faze oblikovanja

Faza oblikovanja novog sustava spaja potrebne funkcionalnosti iz specifikacije zahtjeva i korištene tehnologije te je potrebno provesti slijedeće aktivnosti:

- Oblikovanje i integracija mreže – detaljno specificirati kako različiti dijelovi sustava međusobno komuniciraju.
- Oblikovanje arhitekture programa – detaljno specificirati kako aktivnosti sustava obavljaju ljudi ili računalo.
- Oblikovanje korisničkog sučelja – specificiranje međudjelovanja korisnika i sustava.
- Oblikovanje sistemskog sučelja - detaljno specificirati kako će sustav raditi zajedno sa drugim postojećim sustavima u i/ili van poduzeća.
- Oblikovanje i integracija baza podataka – specificirati kako i gdje će sustav pohranjivati podatke i informacije potrebne poduzeću.
- Prototipiranje detalja oblikovanja – izraditi prototipove koji će potvrditi sve odluke donesene u oblikovanju sustava.
- Oblikovanje i integracija sistemskih kontrola – detaljno specificirati provjere ispravnog funkcioniranja sustava i održavanja podataka, te sigurnosti i zaštite sustava.

Oblikovanje i integracija mreže mogu biti za potpuno novu umreženost računalne podrške ili uklapanje u već postojeću mrežu. U prvom slučaju treba dizajnirati cijelu mrežu, dok u drugom slučaju treba razmotriti mogućnosti integracije postojeće i nove mreže. U oba slučaja, a naročito u drugom, treba voditi računa o pouzdanosti,

sigurnosti, propusnosti i sinkronizaciji. Najvažnije pitanje na koje umreženost treba dati pozitivan odgovor je: jesu li detaljno specificirani svi različiti dijelovi sustava koji trebaju međusobno komunicirati?

Oblikovanje arhitekture programa treba definirati kako će se izvršavati aktivnosti novog sustava. To su aktivnosti koje su detaljno specificirane u modelima sustava za vrijeme faze analize kada se nije vodilo računa o tehnologiji. U trenutku kada se odabere tehnološko rješenje treba specificirati detalje računalne obrade i fizičke realizacije prepoznatih aktivnosti sustava. Tako se neće jednako oblikovati arhitektura sustava koji počiva na *Cobol* jeziku za izradu programa ili na *Visual Studio* platformi. Također, neće biti svejedno je li riječ o centraliziranom sustavu ili *client-server* arhitekturi. Neke aktivnosti će i dalje ostati u domeni ljudske odluke i izrade te neće biti direktno podržane računalom. Najvažnije pitanje na koje oblikovanje arhitekture programa treba dati odgovor je: je li detaljno specificirano kako aktivnosti sustava obavljaju ljudi i računala?

Oblikovanje korisničkog sučelja definira kako korisnik komunicira s računalom. Za većinu korisnika sučelja ona predstavljaju grafička korisnička sučelja sa prozorima i upitima uz korištenje miša (može biti uključen i zvuk, video ili glasovne naredbe). Uz to, sposobnosti i potrebe ljudi su različite, te ako je IS većinom interaktivan i dostupan, onda korisničko sučelje predstavlja veliki dio tog sustava. Već kod specifikacije zahtjeva javlja se potreba za definiranjem korisničkog sučelja ali tada se mogu samo navesti alternative koje ispunjavaju zadane funkcionalne potrebe. U fazi oblikovanja treba odabrati i detaljno razraditi zahtijevana korisnička sučelja. Najvažnije pitanje na koje oblikovanje korisničkog sučelja treba dati odgovor je: je li detaljno specificirano međudjelovanje korisnika i sustava?

Oblikovanje sistemskog sučelja uzima u obzir i ostale sustave koji su već u uporabi i s kojima novi sustav treba integrirati. Sustavi će međusobno razmjenjivati podatke a komponente koje to omogućuju su sistemska sučelja koja je također potrebno detaljno oblikovati. U nekim će slučajevima biti potrebno novi sustav integrirati sa sustavom izvan poduzeća, primjerice, ako je potrebno osigurati sučelja za razmjenu podataka među trgovačkim poduzećima (takozvano *B2B* elektroničko poslovanje). Kako je danas u uporabi veliki broj specijaliziranih tehnoloških mogućnosti, sistemska sučelja mogu biti vrlo kompleksna. Najvažnije pitanje povezano s oblikovanjem sistemskog sučelja je: je li detaljno specificirano kako će sustav raditi zajedno sa drugim postojećim sustavima u i/ili van poduzeća?

Oblikovanje i integracija baza podataka je još jedna ključna aktivnost faze oblikovanja. Logički model podataka koji je izrađen za vrijeme analize sada treba prevesti u odgovarajući fizički model baze podataka. Najčešće su to relacijske baze podataka. Projektant treba voditi računa o mnogim postavljenim zahtjevima kada oblikuje fizičku bazu podataka, između ostaloga i o brzini vraćanja rezultata upita nad podacima. Ako već postoje sustavi i pripadajuće baze podataka, treba oblikovati kvalitetnu integraciju podataka među bazama. Najvažnije pitanje povezano s

oblikovanjem i integracijom baza podataka je: je li specificirano kako i gdje će sustav pohranjivati podatke i informacije potrebne poduzeću?

Prototipiranje detalja oblikovanja je često potrebno kako bi se potvrdio odabir tehničkih elemenata sustava poput baze podataka, arhitekture mreže, postavljenih kontrola ili programske okoline. Tako analitičar treba razmisliti kako oblikovati prototip sustava (ili dijela sustava) kako bi mu pomogao u odlukama povezanim s konačnim programskim rješenjem. U nekim metodama razvoja (vidi poglavlje 3., iterativni modeli i metode prototipiranja) prototip će biti jedna od među-verzija gotovog sustava. Najvažnije pitanje povezano s prototipiranjem detalja oblikovanja je: potvrđuje li izrađeni prototip sve odluke donesene u oblikovanju sustava?

Oblikovanje i integracija sistemskih kontrola treba osigurati odgovarajuću zaštitu svih resursa poduzeća. To su sistemske kontrole koje se specificiraju ovisno o odabranom korisničkom sučelju, sistemskom sučelju, arhitekturi programa, baza podataka i mreže. Sistemske kontrole ograničavaju pristup korisnika sustavu, štite bazu podataka od neovlaštenog pristupa podacima, te osiguravaju resurse u slučaju sistemskih grešaka i pada sustava. Uz to, sistemske kontrole osiguravaju i komunikaciju preko mreže. Najvažnije pitanje povezano s oblikovanjem i integracijom sistemskih kontrola je: jesu li detaljno specificirane provjere ispravnog funkcioniranja sustava i održavanja podataka, te sigurnosti i zaštite sustava?

8.1.3. Kvaliteta procesa oblikovanja

Za vrijeme oblikovanja IS-a treba voditi računa o kvaliteti rezultata tog procesa jer oblikovanje predstavlja visoku razinu apstrakcije koja se treba moći direktno preslikati u specifične sustave podržane različitim tehnologijama. Za dobro i kvalitetno oblikovanje trebalo bi se pridržavati nekoliko općenitih smjernica povezanih s dobro oblikovanim sustavom i atributima kvalitete koji potvrđuju tu kvalitetu.

Smjernice za primjenu tehničkih kriterija dobrog oblikovanja sustava su:

- Modeli oblikovanja trebaju prikazati arhitekturu sustava prema poznatim obrascima, dobro usklađenima, tako da se mogu izrađivati u iteracijama, a isto tako i testirati i implementirati.
- Oblikovanje treba biti modularno, što znači da sustav treba razložiti u podsustave, te nadalje u logičke elementarne dijelove.
- Modeli oblikovanja sadrže jasne prikaze podataka, arhitekture, sučelja i komponenti.
- Modeli moraju sadržavati strukture podataka koje su prepoznatljive s obzirom na mogućnosti i obrasce implementacije.
- Modeli moraju pokazivati komponente sustava koje se mogu izvesti neovisno o funkcionalnim karakteristikama područja koji je opisan.

- Modeli moraju pomoći u izradi sučelja koje će smanjiti složenost među komponentama sustava.
- Modele oblikovanja treba prikazati jasnom i poznatom notacijom s mogućnosti učinkovite komunikacije među informatičarima.

Atributi koji će osigurati kvalitetno oblikovanje su takozvani FURPS (eng. *functionality, usability, reliability, performance, supportability*) [6]:

- **Funkcionalnost** se postiže procjenom osobina i mogućnosti programa, općenitošću funkcija koje se isporuče i sigurnošću cijelog sustava.
- **Uporabljivost** se postiže stalnom prisutnošću ljudskog faktora, nasuprot estetike, dosljednosti i dokumentacije.
- **Pouzdanost** se procjenjuje mjerenjem učestalosti i težine grešaka, preciznosti izlaznih rezultata, sposobnosti sustava da se oporavi nakon grešaka, te predvidivosti programa.
- **Performanse** se mjere brzinom obrade, vremenom odgovora, zauzetošću resursa, propusnom moći i učinkovitosti.
- Sposobnost **podrške** predstavlja kombinaciju mogućnosti proširenja, prilagodbe i servisiranja programa, i uz to još mogućnosti dodatne konfiguracije, kompatibilnost i mogućnost testiranja.

Nisu svi navedeni atributi kvalitete jednako važni za pojedini razvoj programa. Neki sustavi će dati važnost sigurnosti, drugi performansama s obzirom na opseg i protok informacija, dok se treći, primjerice, mogu fokusirati na pouzdanost.

8.2. Oblikovanje programa

Arhitektura programa mora biti u skladu s oblikovanjem baza podataka i korisničkog sučelja. **Tradicionalne** tehnike i prijašnji jezici treće generacije (*Visual Basic*, *Cobol*, *Pascal*) su (bili) organizirani u module koji su formirali hijerarhijsko stablo. Glavni modul je vrlo vjerojatno glavni izbornik, moduli srednje razine su kontrolni moduli, a krajnji moduli sadrže većinu algoritama i logike, operativnog rada unosa i pregleda podataka (slika 8.3.)

***PRIMJER*:** Na slici 8.3. pokazan je jednostavni izbornik za program *Veleprodaja*.

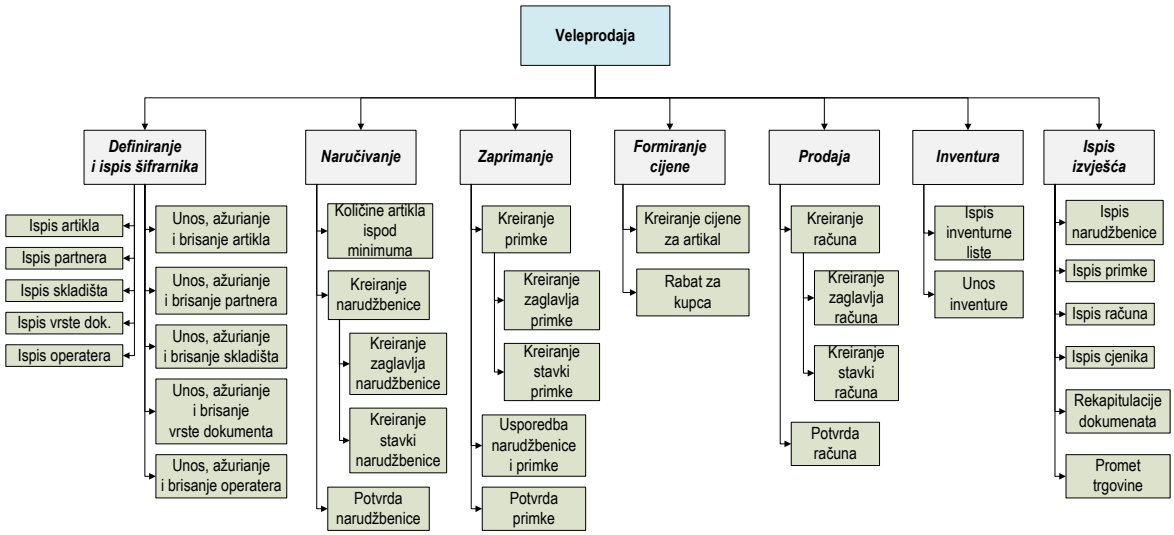
Glavni modul je prvi izbornik programa u kome se biraju mogućnosti: *Unos/ispis šifrarnika*, *Naručivanje*, *Zaprimanje*, *Formiranje cijene*, *Prodaja*, *Inventura* i *Izvješća*.

Moduli srednje razine su funkcionalnosti pojedinih područja, primjerice, *Naručivanje*. To može biti modul koji ima nekoliko kontrola:

- Kontrola pristupa koja provjerava ima li operater pravo pristupiti nižim modulima: provjera količine, potvrda ili samo unos narudžbenice.

- Kontrole pristupa podacima koja primjerice za količine ispod minimuma provjerava može li operater pristupiti samo svom skladištu ili i svim skladištima u regiji.
- Ako pristupa svim skladištima, aktiviraju se dodatne sistemske kontrole jer svi nemaju istu programsku podršku za sustav prodaje.

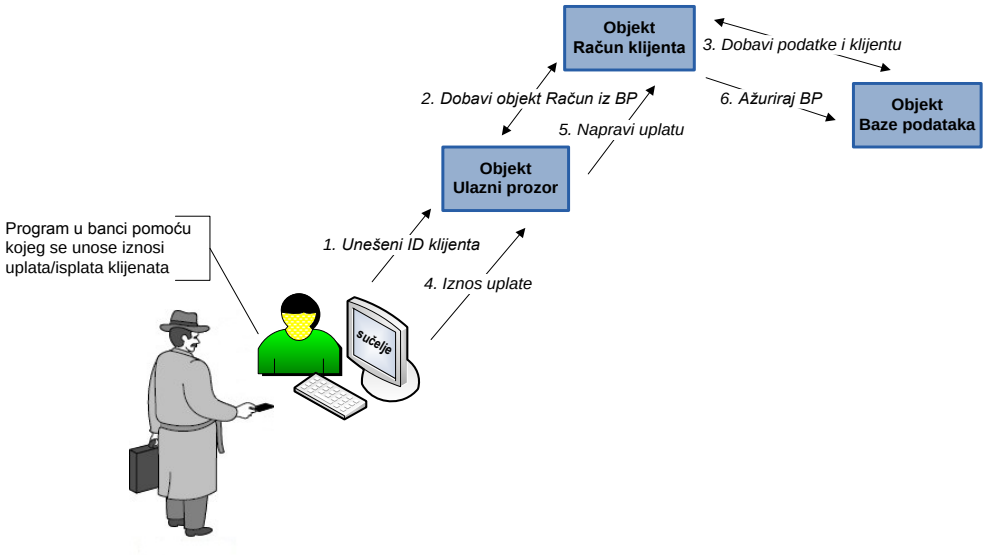
Moduli najniže razine su operativni moduli sučelja za unos i izvješćivanje. U nekima kao *Količine ispod minimuma* i *Usporedba primke i narudžbenice* nalaze se i dodatni algoritmi za pretraživanje dokumenata i artikala u bazi podataka, te sužavanje informacija dodatnim ograničenjima.



Slika 8.3. Hijerarhija modula programa za veleprodaju

S druge strane, **objektno-orijentirani** programi se sastoje od skupa programskih objekata koji surađuju kako bi dali rezultat. Svaki programski objekt unutar sebe sadrži i programsku logiku (eng. *encapsulate*) i sve potrebne atribute kako bi bio neovisna jedinica. Ovi objekti rade tako da jedni drugima šalju poruke i primaju ih , te rade usklađeno podržavajući funkcije glavnog programa. U stvari, u OO programima nema glavnih modula nego analitičar definira strukturu programske logike i podataka definirajući klase koje opisuju privremenu strukturu objekata koji se u tom trenutku izvršavaju.

***PRIMJER*:** Na slici 8.4. pokazan je OO program koji sadrži objekt *Ulazni prozor* koji prikazuje unosnu masku za ID klijenta i ostale podatke o klijentu. Nakon što je unesen ID klijenta objekt *Ulazni prozor* šalje poruku (br.2.) klasi *Račun klijenta* koja kaže da treba kreirati novi objekt *Račun klijenta*. Ta instanca objekta s informacijama o klijentu otići će k bazi podataka, pokupiti na sebe podatke o klijentu za zadani ID klijenta i vratiti u objekt *Račun klijenta* (br.3.). Tada će taj novi objekt *Račun klijenta* poslati podatke o klijentu natrag u objekt *Ulazni prozor* koji će ih prikazati na ekranu. Službenik će tada unijeti iznos pologa (br.4.), te će nova poruka biti poslana sa objekta *Ulazni prozor* prema objektu *Račun klijenta* u programu (br.5.), a ovaj će ju poslati u bazu podataka (br.6.).



Slika 8.4. Događajima pokretani tok programa u OO pristupu

U nastavku ovog potpoglavlja o oblikovanju programa detaljno su obrađeni tradicionalni pristup oblikovanju i objektno-orijentirani pristup oblikovanju programa.

8.2.1. Tradicionalni pristup oblikovanju programa

U strukturnim programskim jezicima program je skup modula koji prilikom prevođenja postaju jedan izvršni entitet. Oblikovanje ovakvih programa može se podržati modelima visoke razine koji opisuju skup programa i potprograma, komunikaciju s bazama podataka i međusobnu interakciju.

Jedna od odgovarajućih tehnika je i dijagram toka podataka (DTP, vidi potpoglavlje 4.2.3.) kojim se može prikazati potrebna razina opisa programa i pripadajućih modula. Detaljniji opis odnosa događaja i procesa može podržati tehnika **radnog dijagrama** (eng. *system flowchart*), koji je vrlo sličan dijagramu aktivnosti iz UML-a (vidi potpoglavlje 4.3.4.; DA je i nastao na osnovu koncepata i strukture starijeg „brata“ – radnog dijagrama).

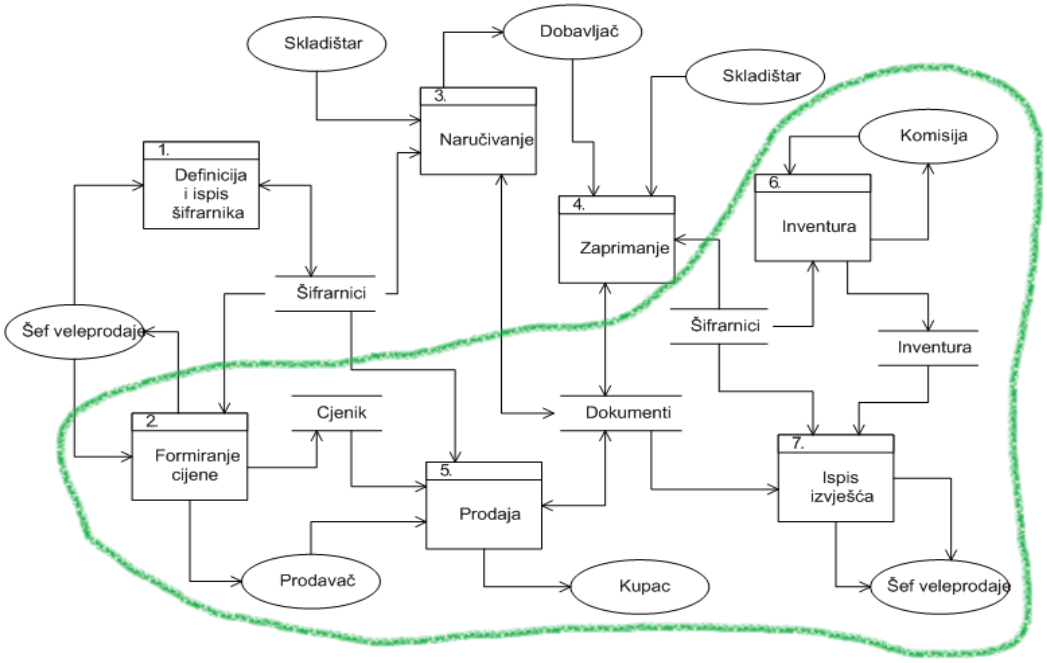
Spomenuti dijagrami su modeli poslovanja nastali u fazi analize i na osnovu njih se može nastaviti (1) modeliranje novog sustava tako da se pristupi detaljiziranju i organizaciji elemenata programa, ili (2) izmjena modela analize jer arhitektura i tehnologija novog sustava zahtijevaju drugačiju organizaciju elemenata sustava.

Jedna od važnih odluka prilikom izrade modula programa je njihovo grupiranje. Ono može biti diktirano formiranjem poslovnih podsustava po područjima poslovanja, alatima i tehnologijom koja će bi korištena za pojedine podsustave, arhitekturom baza podataka, mreže i slično.

Kada se odlučimo za skupine modula koje predstavljaju funkcionalne cjeline treba iz dijagrama razmotriti kakve su veze tih cjelina sa okruženjem. Te veze mogu biti unutar cijelog sustava koji se razvija i tada treba prije svega paziti na integraciju s ostalim podsustavima i bazama podataka.

Druga vrsta je veza s okruženjem izvan sustava poduzeća. U tom slučaju treba posvetiti pažnju i mrežnoj integraciji te sistemskim kontrolama u razmjeni informacija i podataka. Ako je veza s vanjskim okruženjem "papirnata" treba osigurati ispis odgovarajućih izvješća iz podsustava.

***PRIMJER*:** Za hijerarhiju modula sa slike 8.3. prikazan je jednostavni izbornik za program *Veleprodaja*. Pripadajući kontekst dijagram sa slike 4.4. po principu izrade DTP-a dalje se razlaže na niže dijagrame, pa prva razina prepoznatih modula srednje razine izgleda kao na slici 8.5. Sa slike se vidi da je odlučeno da će u opsegu programa za trgovinu biti funkcionalnosti povezane s prodajnim aktivnostima i izradom cjenika te aktivnost inventure.



Slika 8.5. Odabrani skup funkcionalnosti za program u trgovini

Ako se analiziraju granice zadanog programa (podsustava), može se uočiti integracija s bazom podataka osnovnih šifrarnika te bazom podataka skladišnog poslovanja (nabava i zaprimanje). Ako ima više trgovina i jedno centralno mjesto za zajedničke podatke, te *client-server* arhitektura, vrlo je vjerojatno da će se glavna BP za šifrarnike nalaziti na centralnom mjestu. Stoga treba posvetiti pažnju oblikovanju i integraciji baza podataka šifrarnika.

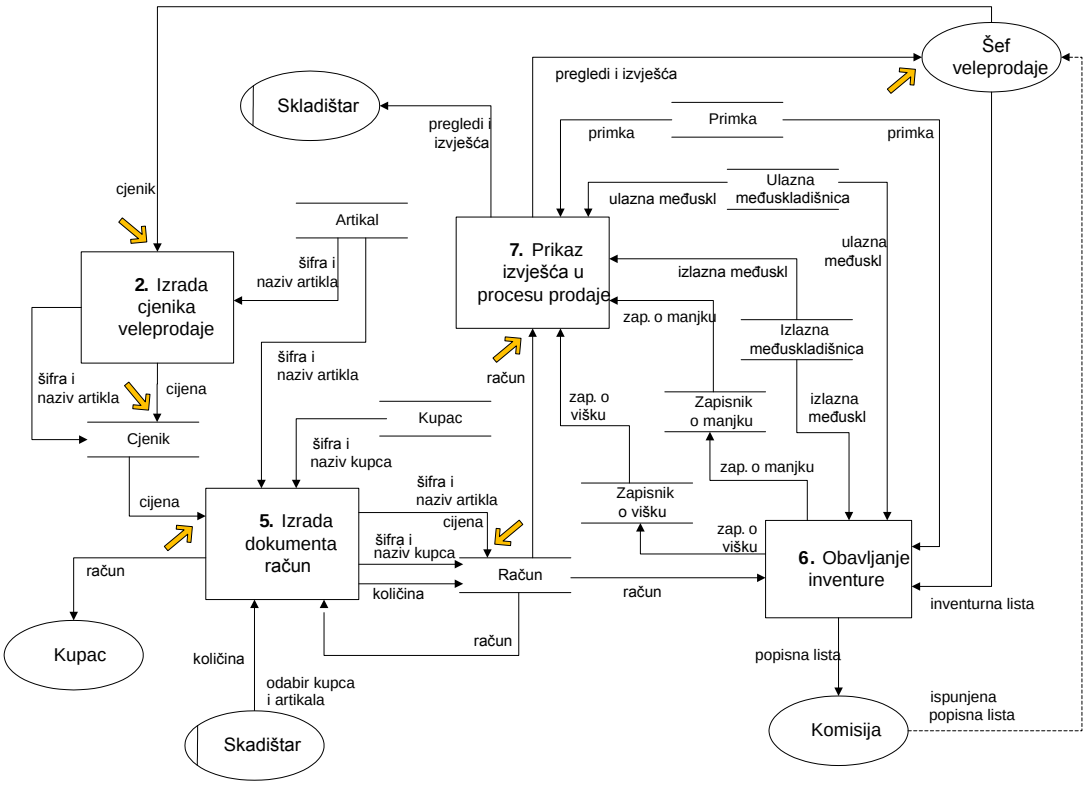
Kada se na visokoj razini modula prepoznaju cjeline (podsustavi) mogu se dalje analizirati moduli nižih razina, za što je također pogodan DTP. Sada dolazi do izražaja detaljan pregled tijeka podataka, što i jeste jedna od namjena modela koji prikazuju

zajedno procese i podatke. Ako se prati tijek podataka od ulaska u podsustav (ulazni tokovi iz okruženja) te kroz procese do izlaska (izlazni tokovi u okruženje), može se konstruirati detaljan pregled **životnog ciklusa entiteta** (vidi potpoglavlje 5.1.1.).

Za detaljno oblikovanje odnosa događaja koji pokreću procese, sučelja koja su interakcija programa s korisnikom i različitih situacija koje može poprimiti tijek poslovnih procesa pokrenutih događajima i odlukama, može se koristiti i radni dijagram (prije spomenuto) kojim se prikazuju odluke i grananje tijeka programa.

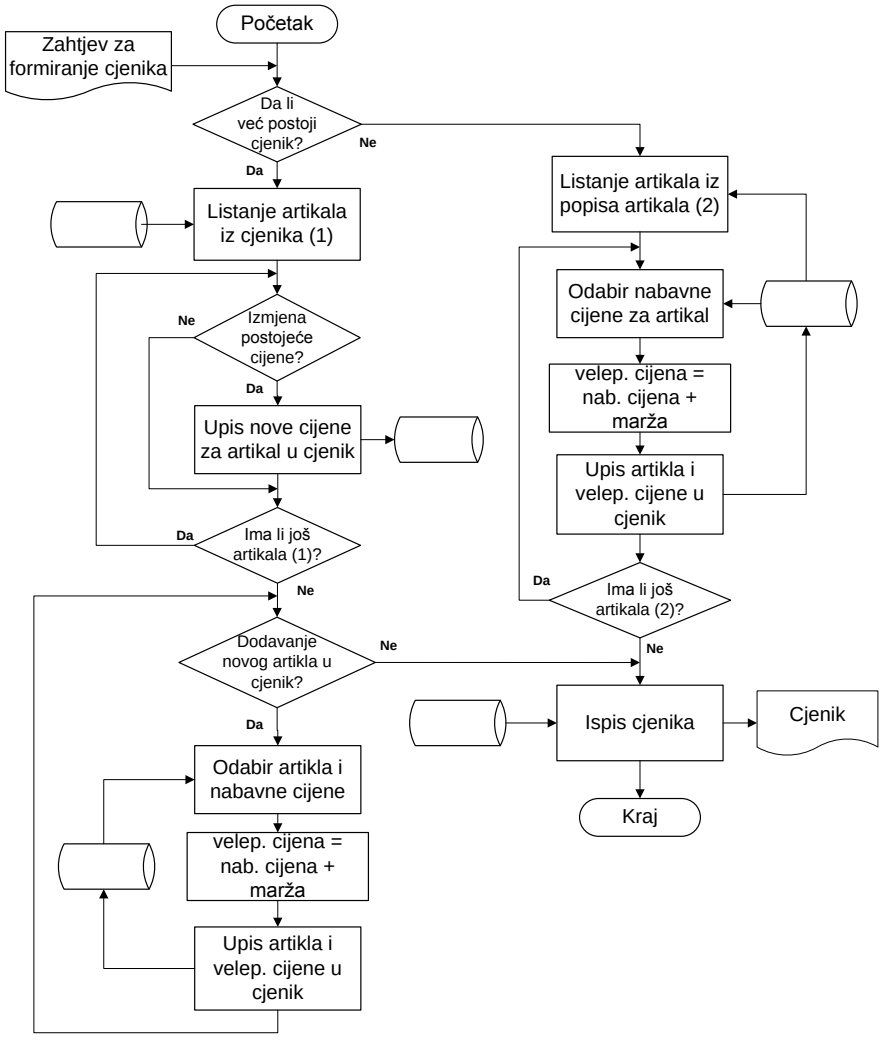
***PRIMJER*:** Odabrani procesi sa slike 8.5. detaljno su analizirani u DTP-u niže razine kako pokazuje slika 8.6. Na ovoj slici su prikazani svi dokumenti koji se koriste u programu prodaje. Sada DTP prikazuje detalje potrebne da se programiraju sučelja prema korisniku koja će obrađivati potrebne dokumente, te da se podaci sa sučelja upišu u BP.

Sa DTP-a se može iščitati i životni ciklus entiteta i podataka. U primjeru na slici 8.6. je strelicama prkazano kako nastaje dokument cjenika (ili samo izmjena cijene, pa je riječ o ažuriranju već postojećih podataka), kako se cijena iz cjenika koji je u BP čita i zapisuje na dokument račun, te kako se sa računa čita (zajedno sa drugim relevantnim podacima) kako bi se prikazala u obliku obrađenih informacija na raznim izvješćima potrebnim za operativni rad i odluke u poslovanju trgovine.



Slika 8.6. Skup modula funkcionalnosti inventure

Iz DTP-a sa slike 8.6. još uvijek nije jasno kako programirati detalje procesa i aktivnosti poslovanja, te je na slici 8.7. prikazan jedan radni dijagram koji opisuje funkciju *Izrada cjenika veleprodaje*.



Slika 8.7. Radni dijagram za izradi cjenika

Odmah na početku radnog dijagrama je događaj koji pokreće izradu ili doradu cjenika, a to je zahtjev za formiranjem novih cijena u trgovini. Ovisno o odluci radi li se novi cjenik ili mijenja postojeći, program se dijeli u dvije grane koje se opet mogu sastati u funkciji ispisa cjenika.

Važno je primijetiti da postoje dijelovi algoritma koji se ponavljaju (na slici 8.7. je to namjerno napravljeno); no trebalo bi promijeniti i optimizirati radni dijagram tako da ponavljajući procesi budu samo jednom naznačeni). Prilikom programiranja ovaj slučaj se svakako optimizira tako da se formiraju funkcije i procedure koje će se onda pozivati na raznim mjestima u programu (vidi idući primjer).

Odluka je li riječ o novom cjeniku ili samo o doradi postojećeg je poslovna odlika šefa trgovine (vjerojatno ne samo njega nego i nekoga iz rukovodeće strukture).

Za oblikovanje unutarnje logike pojedinih modula, algoritama i procedura obično se koristi *pseudokod*. Originalno se prikazuje strukturiranim engleskim jezikom, no može biti i proizvoljan do određene mjere. Na raspolaganju bi trebao biti ograničeni broj riječi (posebno onih koje jesu simboli naredbi nekog programskog jezika). Izbor riječi i značenja u pseudokodu može biti proizvoljan, važno je da se projektni tim složi oko termina, oznaka i njihova značenja (u protivnom se neće moći jasno i jednoznačno tumačiti).

***PRIMJER*:** Za funkcionalnost *Izrada dokumenta račun* sa slike 8.5. može se izraditi pseudokod relativno visoke razine (općenitosti), kako je pokazano na slici 8.8. Ovaj primjer pseudokoda naglašava postojanje uvjeta u programskom kodu, petlji koje u sebi sadrže skup naredbi koje se ponavljaju te akcije koje su relativno jednostavno naznačene, ali nisu elementarne naredbe nekog programskog jezika.

```
* Kreiranje računa
  Odredi prodajno mjesto (skladište) za izdavanje računa
  Otvori novi račun i dodijeli mu broj računa
    IF Veleprodajni račun
    Dodaj kupca na račun
  Dodaj datum računa
  Dodaj operatera na račun
    FOR ALL Stavka računa
      Otvori novu stavku računa
      Dodaj artikal u stavku (šifra i naziv artikla, jedinica mjere, cijena)
      Dodaj količinu u stavku
      Izračunaj iznos stavke (količina * cijena)
  Izračunaj ukupan iznos računa
  Izračunaj iznos PDV-a
```

Slika 8.8. Pseudokod za izradu računa u sustavu trgovine

U drugom dijelu je opisano oblikovanje programa objektno-orijentiranim pristupom. Punu korisnost ovakvo će oblikovanje dati tek kada je i analiza poslovanja i prethodno planiranje projekta izrađeno objektno-orijentiranim pristupom. U tom će slučaju modeli i odgovarajući dijagrami sistematizirati područje koje se obrađuje, te će nastavak oblikovanja programa i ostalih tehnoloških mogućnosti doći do potpunog izražaja.

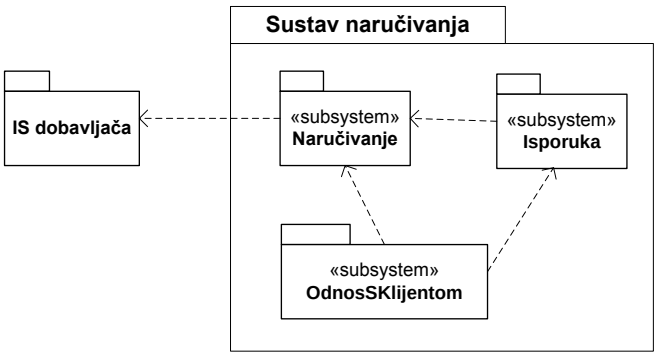
Ukoliko se strategija razvoja IS-a odvija potpomognuta objektno-orijentiranom tehnologijom, utoliko je prirodno uzeti odgovarajuće metode i tehnike, a slučajevi korištenja (vidi potpoglavlje 4.3.1.) nikako se ne bi smjeli izostaviti jer na osnovu njihove priče idu sva daljnja modeliranja.

8.2.2. Objektno-orijentirani pristup oblikovanju programa

Objektno-orijentirano oblikovanje programa i ostalih elemenata IS-a je složen proces koji je bogat tehnikama i modelima, te pravilima koja pružaju današnje metodologije za izgradnju takozvanih događajima pokretanih sustava (eng. *event-driven system*). Kako bi se uspješno koristili svi modeli i tehnike koje su na raspolaganju i pravila metodologije razvoja, trebalo bi da projektanti i programeri vrlo dobro vladaju i OO paradigmom i svim pratećim sadržajima.

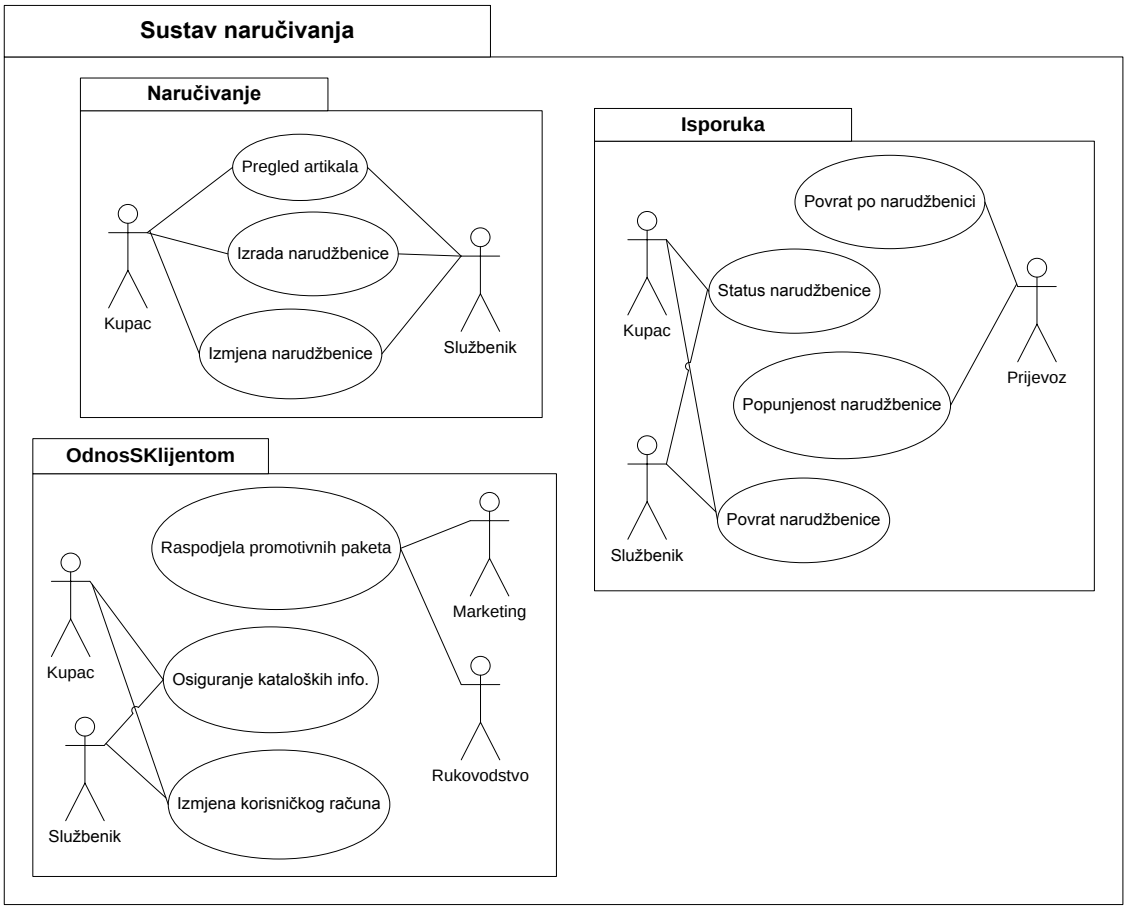
U analizi poslovanja prevladavaju dijagrami slučajeva korištenja i njihovi scenariji kojima se opisuje ponašanje sustava onako kako ga vidi korisnik. Scenarij je detaljan opis jednog slučaja korištenja, a nekoliko slučajeva korištenja može se grupirati na više načina. Jedan je pomoću paketa. Tako se u slučaju OO pristupa visoka razina podjele sustava na podsustave može prikazati paketima, a kada se spuštamo na niže razine onda drugim dijagramima.

***PRIMJER*:** Na slici 8.9. je dijagram paketa visoke razine koji pokazuje osnovne podsustave sustava naručivanja. Naglašena je i povezanost s vanjskim sustavom dobavljača (možemo pretpostaviti da je riječ o integraciji i automatskoj razmjeni podataka).

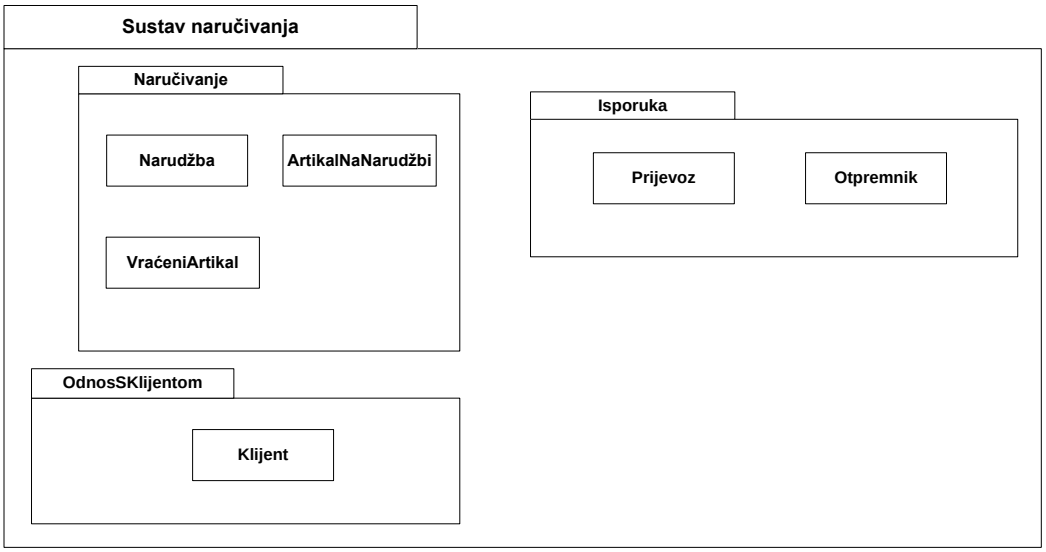


Slika 8.9. Dijagram paketa za sustav naručivanja

Dijagram paketa ovog primjera izrađen je u namjeri da se na najbolji način grupiraju (oblikuju) dijelovi budućeg sustava. Kako bi se došlo do ovakvog grupiranja, analitičar je razmotrio prikupljene slučajeve korištenja i pripadajuće početne dijagrame klasa. Budući da svaki element paketa treba pripadati samo jednom paketu, jasna je podjela na slici 8.10. i na slici 8.11.



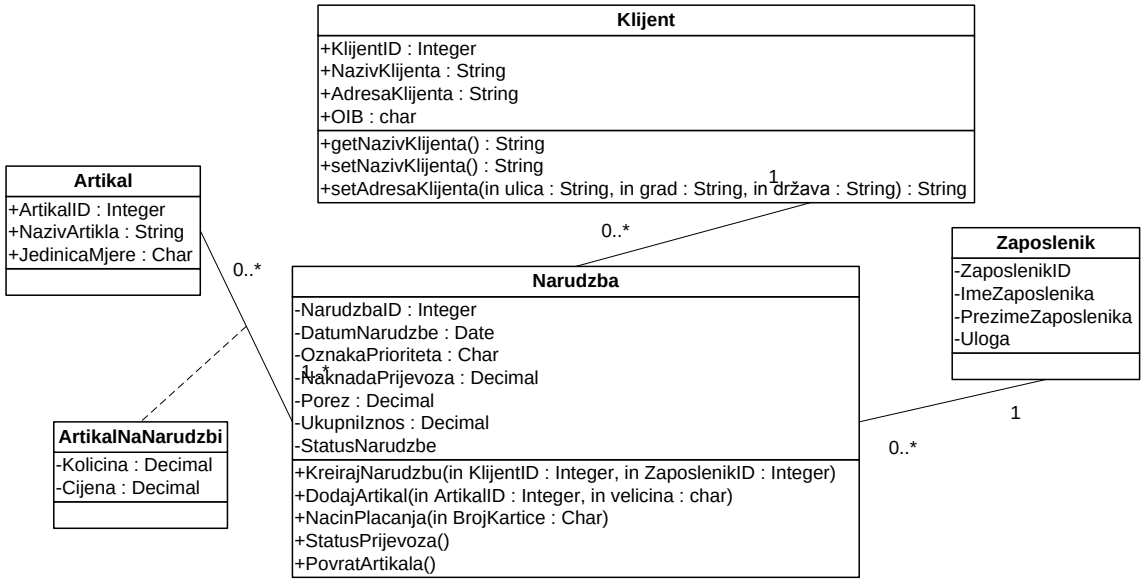
Slika 8.10. Dijagram paketa sa slučajevima korištenja



Slika 8.11. Dijagram paketa s pripadajućim klasama

Sada bi bilo važno povezati slučajeve korištenja i klase tako da se aktivnosti i procedure, prepoznate u koracima scenarija slučajeva korištenja, ugrade u klase. Naime, za vrijeme analize ne vodi se toliko računa o svim detaljima klasa, kao što su atributi, tipovi atributa, ograničenja. Uz to, klasa ima sposobnost da u sebi čuva i metode povezane s elementima klase, pa je sada u oblikovanju trenutak da se te metode definiraju i povežu s detaljnom specifikacijom atributa i ostalih objekata u sustavu.

PRIMJER: Na slici 8.12. naznačeni su detalji samo nekih klasa iz dijagrama klasa koji je prepoznat u analizi poslovanja (slika 7.4.).



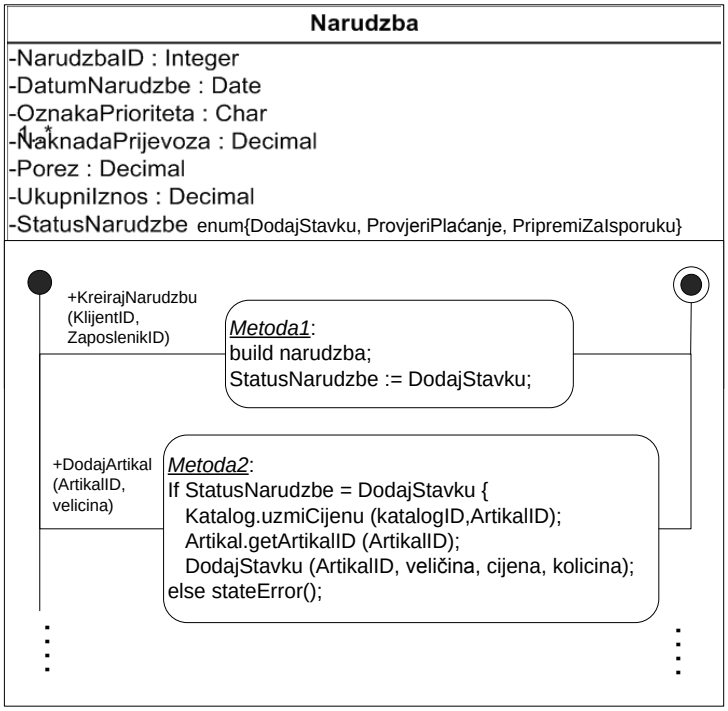
Slika 8.12. Dijagram klasa s detaljima atributa i metoda

Na dijagramu klasa za narudžbu vide se detalji specifikacije metoda. Primijetimo da se ovaj dio može usporediti sa prepoznavanjem životnog ciklusa entiteta sa slike 8.6. i 8.7. koji su bili logičko oblikovanje i priprema za programiranje strukturnim pristupom. Objektno-orijentirani pristup je cjelovit jer su sada svi događaji povezani s klasom (entitet) narudžbe okupljeni u narudžbi, a nisu raspršeni po programskom kodu.

Za stvarni opis detalja unutar metoda, dakle dijelova programskog koda ili pseudokoda, potrebno je klasu prikazati na jedan složeniji način. U primjeru na slici 8.12. klase su prikazane s detaljima atributa, a metode su samo naznačene, slično kao funkcije i procedure, gdje se samo prikazuje njihov naziv te ulazni i izlazni parametri. Ipak, tijelo metode (naredbe koje su sadržaj metode) još uvijek nisu specificirane. U slijedećem primjeru prikazan je jedan od načina kako specificirati detalje metoda.

PRIMJER: Metode koje su na slici 8.12. zapisane u klasi narudžbe sada su detaljno razrađene do razine kodiranja. Sa slike 8.13. može se protumačiti druga metoda koja dodaje artikl na narudžbu. Za tu se metodu, u slučaju da se na narudžbu može dodavati

artikl, pregledava katalog artikala (cjenik). Kada se pronađe artikal zajedno sa odgovarajućom veličinom, prenese se njegov ID i veličina u stavku narudžbe, zajedno sa unesenom količinom.



Slika 8.13. Dijagram klasa s razrađenim kodom metoda

Očigledno je da faza analize donosi puno informacija prikazanih modelom, ali je ipak u oblikovanju znatno više detalja. To je i razumljivo, jer detalji se ne mogu pretjerano navoditi vrlo rano u razvoji IS-a. Takav pristup bi bio kontraproduktivan, jer bi se trošilo mnogo vremena i napora, a izgledno je da bi veliki dio razrade bio upitne kvalitete. Taj princip je uključen u brzi razvoj, pa je to jedan od razloga zašto metode brzog razvoja IS-a sve više uspijevaju na tržištu informatičke tehnologije.

8.3. Oblikovanje baza podataka

Baza podataka predstavlja integrirano mjesto na kojem se prikupljaju podaci, dok cijeli proces unošenja podataka i rukovanja postojećim podacima podržava i kontrolira sustav za upravljanje bazama podataka, SUBP (eng. *Database Management System*, DBMS).

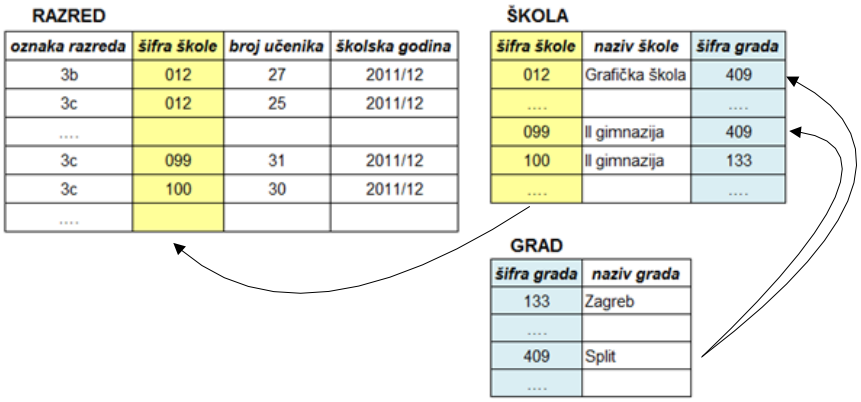
Baza podataka je u SUBP zapisana u obliku takozvane sheme baze podataka, odnosno zapisana je njena struktura. Shema baze podataka sadrži opise informacija o podacima pohranjenima na fizičkom mediju za pohranu, a sastoji se od:

- Kontrole pristupa - dopuštenih vrijednosti pojedinih elemenata podataka, vrijednosti koje ovise o nekim drugim elementima baze te liste korisnika kojima je dopušten pristup bazi podataka i njenim podacima.
- Veza među elementima baze i grupama elemenata.
- Detalja o fizičkoj organizaciji podataka kao što su tip i duljina elemenata, lokacije pojedinih elemenata te indeksi i ključni elementi za brži i sigurniji pristup podacima.

8.3.1. Relacijske baze podataka

Strukturu relacijskog modela podataka grade relacije. Relacija u relacijskom modelu podataka je isto što i relacija u matematici s tom razlikom da su relacije u relacijskom modelu podataka vremenski promjenjive. U relacijskoj bazi podaci su strukturirani u tablice, odnosno entitete, koje predstavljaju dvodimenzionalno spremište podataka. Stupci predstavljaju polja, odnosno atribute, a svaki je redak, odnosno *record*, pojavljivanje jedne jedinke zadanog entiteta s vrijednostima, odnosno podacima, za svaki od pripadajućih atributa (napomena; ne mora svaki atribut biti obvezan za unos).

PRIMJER: Tablice sa slike 8.14. su jednostavni predstavnici nekoliko relacija baze podataka koja, između ostaloga, skladišti podatke o školama, vjerojatno na nivou regije ili države. Glavni princip ispravno organizirane baze podataka vidljiv je iz ovog primjera. Tako je ponavljanje istovjetnih podataka eliminirano grupiranjem u entitete, a povezanost je ostvarena preko jedinstvenih šifri.



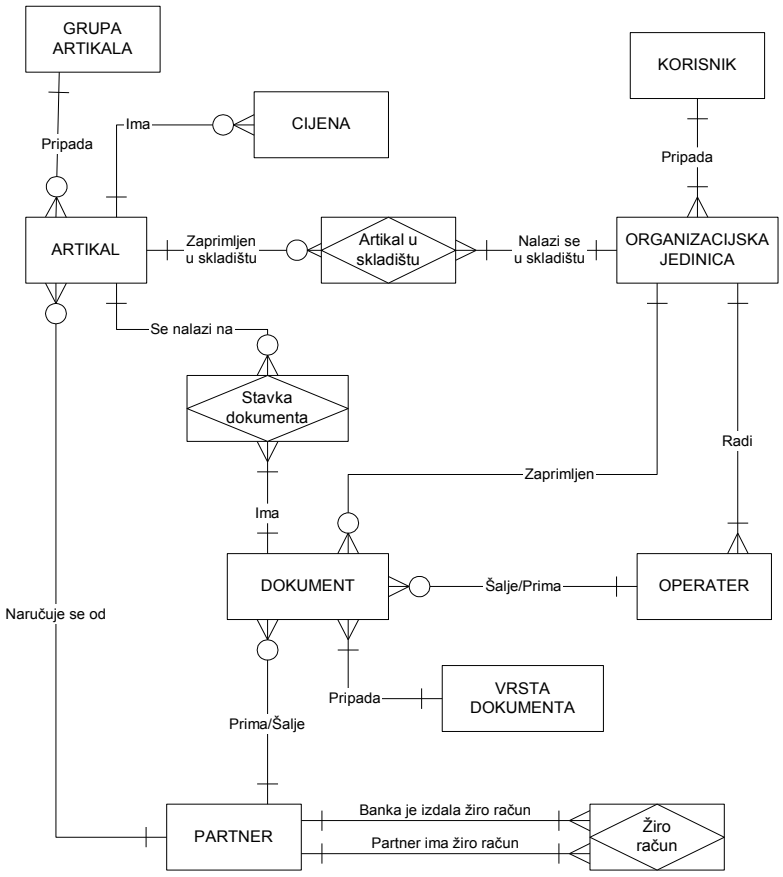
Slika 8.14. Primjer tablica relacijske baze podataka

Kako bi formirali relacijsku bazu podataka potrebno je učiniti nekoliko koraka koji su vrlo vjerojatno raspodijeljeni u nekom dužem vremenskom periodu. Naime, izgradnja dobre baze podataka je evolucijski proces gdje se početne informacije o podacima strukturiraju preko jednostavnih modela podataka (E-V model) do složenijih, koji se

sa svim detaljima preslikavaju u fizičku shemu relacijske baze podataka. Za izgradnju relacijske baze podataka iz E-V dijagrama potrebni je:

- Kreirati tablicu za svaki entitet.
- Odabrati glavni ključ za svaki entitet (ako ne postoji među atributima treba ga umjetno dodati).
- Dodati strane ključeve u entitete na strani *više* u vezama *jedan naprema više*.
- Dodati nove tablice za veze *više naprema više*.
- Definirati ograničenja referencijalnog integriteta.
- Provjeriti kvalitetu sheme i sprovesti potrebna poboljšanja.
- Za svaki atribut odabrati odgovarajući tip podataka i eventualna ograničenja.

***PRIMJER*:** Model E-V sa slike 8.15. sadrži entitete potrebne da programi za naručivanje i zaprimanje te prodaju artikala mogu funkcionirati.

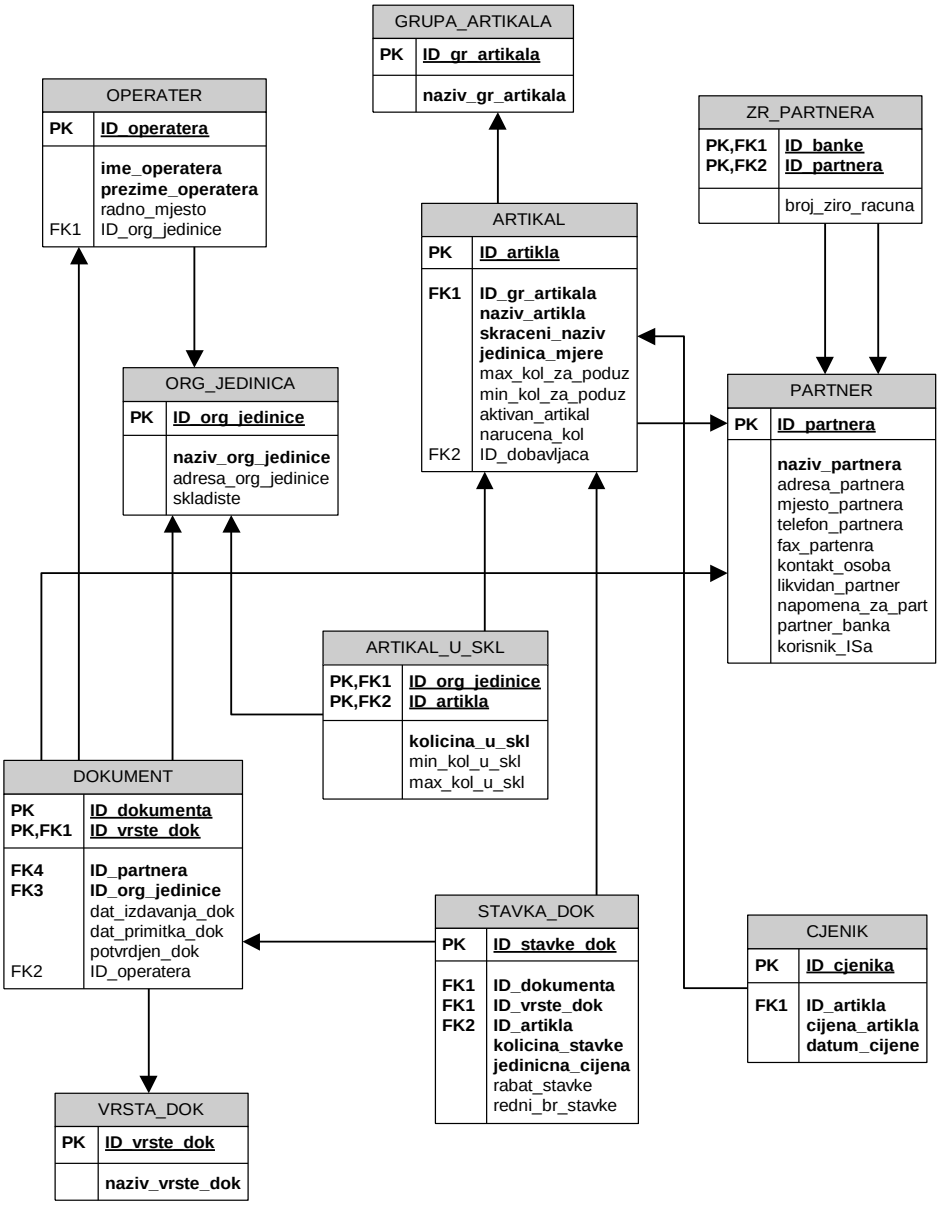


Slika 8.15. Logički E-V model za sustav trgovine

Primijetimo da je ovaj model optimiziran, što znači da je i u fazi oblikovanja dobro koristiti se E-V modelom za izradu što kvalitetnije strukture baze podataka. U modelu na slici 8.15. svi dokumenti iz poslovanja trgovine su smješteni u jednu tablicu. Također, podaci o kupcu i podaci o dobavljaču su istovjetni, te je prirodno

organizirati ih u jedan entitet poslovnih partnera (odnosno poslovnih subjekata). Veza s bankom u kojoj partner ima žiro račun ostvarena je preko dodatne tablice žiro račun. Kako su i banke poslovni subjekti, prema tablici žiro računa, definirane su dvije veze: jedna je šifra poslovnog partnera, a druga je šifra banke u kojoj partner ima žiro račun.

Nakon primjene prije navedenih koraka za izgradnju relacijske baze podataka, dobiven je relacijski model kao na slici 8.16.



Slika 8.16. Relacijski model baze podataka za sustav trgovine

8.3.2. Objektno-orijentirane baze podataka

Objektni sustav za upravljanje bazama podataka (ODBMS od eng. *Object Database Management System*) dizajniran je specijalno za pohranu objekata te kao sučelje za komunikaciju s objektno-orijentiranim programskim jezicima. Iako se objekti mogu spremati i u relacijske baze podataka, ODBMS je posebno dizajniran tako da može direktno podržati spremanje metoda, nasljeđa, ugniježđenih objekata, specifičnih veza među objektima, te tipova podataka koje samostalno kreira programer.

Kako bi se iz dijagrama klasa izgradila shema objektne baze podataka potrebno je :

- Prepoznati koje klase predstavljaju skladišta podataka.
- Definirati trajne klase.
- Prikazati veze među trajnim klasama.
- Za svaki atribut odabrati odgovarajući tip podataka i eventualna ograničenja.

Definiranje trajne klase je važno jer se takav objekt neće poništiti nakon što program prestane raditi. Primjer opisa klase za klijenta sa slike 8.12. može izgledati ovako:

```
Class Klijent {
  attribute integer KlijentID
  attribute string NazivKlijenta
  attribute string AdresaKlijenta
  attribute char(11) oib
}
```

Izgraditi vezu među klasama znači ugnijezditi identifikator jedne klase u drugu klasu u vezi. Ovdje su opisani načini ugnježđivanja za tipove veza:

- jedan naprema više
- jedan naprema više
- više naprema više
- generalizacija.

Za početak evo primjer veze ***jedan naprema jedan*** između klasa zaposlenik i računalo:

```
Class Zaposlenik {
  attribute string ImeZaposlenika
  attribute string PrezimeZaposlenika
  attribute integer IznosPlace
  relationship Racunalo Koristi inverse Racunalo::Pripada
}
Class Racunalo {
  attribute string Proizvodjac
  attribute string SerijskiBroj
  relationship Zaposlenik Pripada inverse Zaposlenik::Koristi
}
```

Rezervirana riječ *relationship* se koristi da bi se deklarirala veza među klasama. Nazivi veza su: za klasu *Zaposlenik* veza prema klasi *Računalo* naziva se *Koristi*, a za klasu *Računalo* veza prema klasi *Zaposlenik* naziva se *Pripada*.

Na slici 8.12. veza između klasa *Klijent* i *Narudzba* je veza ***jedan naprema više***. Takva vrsta veze može se u objektnoj bazi podataka definirati na sljedeći način:

```
Class Klijent {
  attribute integer KlijentID
  attribute string NazivKlijenta
  attribute string AdresaKlijenta
  attribute char(11) oib
  relationship set<Narudzba> Izradjuje inverse
  Narudzba::JeNapravljena
}
Class Narudzba {
  attribute integer NarudzbaID
  attribute date DatumNarudzbe
  attribute char(2) OznakaPrioriteta
  attribute decimal(15,2) Porez
  attribute decimal(15,2) UkupniIznos
  attribute StatusNarudzbe
  relationship Narudzba JeNapravljen inverse
  Klijent::Izradjuje
}
```

Veza *Izradjuje* deklarirana je između jednog *Klijenta* i skupa (jedan ili više) objekata *Narudzbe*. Definiranjem veze kao skupa (eng. *set*) daje se do znanja objektnoj bazi podataka da za klijenta treba rezervirati (eng. *allocate*) onoliko objekata narudžbe koliko je potrebno da bi se ostvarila veza i prikazali svi potrebni podaci. U ovom slučaju sama ODBMS dodaje ili briše objekte koji skladište atribute instanci u vezi.

Za vezu ***više naprema više*** sa slike 8.12. za klase *Narudzba* i *Artikal* može se definirati sljedeća shema:

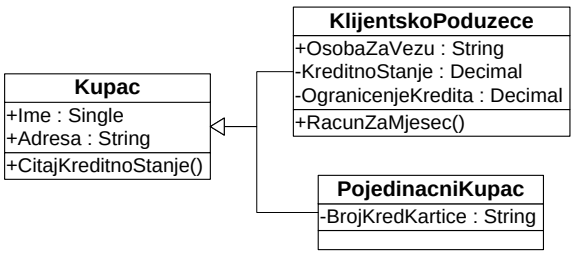
```
Class Narudzba {
  attribute integer NarudzbaID
  attribute date DatumNarudzbe
  attribute char(2) OznakaPrioriteta
  attribute decimal(15,2) Porez
  attribute decimal(15,2) UkupniIznos
  attribute StatusNarudzbe
  relationship set<Artikal> Sadrzi inverse Artikal::NalaziSe
}
Class Artikal {
  attribute integer ArtikalID
  attribute string NazivArtikla
  attribute char(10) JedinicaMjere
  relationship set<Narudzba> NalaziSe inverse Narudzba::Sadrzi
}
```

Definiranjem u obje klase veze kao skupa (eng. *set*) daje se do znanja objektnoj bazi podataka da i za jednu i za drugu klasu treba rezervirati onoliko objekata prema klasi u vezi koliko je potrebno da se prikažu svi potrebni podaci.

Može se očekivati da se veza *više naprema više* može prikazati kao dvije veze *jedan naprema više* pod uvjetom da se doda još jedna klasa kao u relacijskoj bazi podataka. U slučaju da na toj klasi veze ima i atributa, kao što je to slučaj na slici 8.12., onda je nova klasa nužna.

```
Class Narudzba {
  attribute integer NarudzbaID
  attribute date DatumNarudzbe
  attribute char(2) OznakaPrioriteta
  attribute decimal(15,2) Porez
  attribute decimal(15,2) UkupniIznos
  attribute StatusNarudzbe
  relationship set<ArtikalNaNarudzbi> Sadrzi1 inverse
  ArtikalNaNarudzbi::NalaziSe1
}
Class Artikal {
  attribute integer ArtikalID
  attribute string NazivArtikla
  attribute char(10) JedinicaMjere
  relationship set<ArtikalNaNarudzbi> NalaziSe2 inverse
  ArtikalNaNarudzbi::Sadrzi2
}
Class ArtikalNaNarudzbi {
  attribute decimal(15,2) Kolicina
  attribute decimal(15,2) Cijena
  relationship Narudzba NalaziSe1 inverse Narudzba::Sadrzi1
  relationship Artikal NalaziSe2 inverse Artikal::Sadrzi2
}
```

Još jedna česta veza među klasama u objektnoj bazi podataka je **generalizacija**. U 4. poglavlju je u prikazu tehnike UML-a za primjer klase naveden jedan dijagram klasa koji u sebi ima i generalizaciju. Dio tog primjera pokazan je na slici 8.17.



Slika 8.17. Primjer generalizacije u dijagramu klasa

U slučaju generalizacije za ovaj primjer potrebno je napisati shemu objektne baze podataka na slijedeći način:

```
Class Klijent {
  attribute string Ime
```

```
  attribute string Adresa
}
Class KlijentskoPoduzece extends Klijent{
  attribute string OsobaZaVezu
  attribute decimal(15,2) KreditnoStanje
  attribute decimal(15,2) OgranicenjeKredita
}
Class PojedinacniKupac extends Klijent{
  attribute string BrojKredKartice
}
```

Pitanja za ponavljanje

1. *Što je važno u analizi poslovnog sustava a prenosi se u daljnje faze, prvenstveno u oblikovanje?*
2. *Nabrojite i objasnite modele razvoja u fazama analize i oblikovanja (za tradicionalni i za objektni pristup). Objasnite eventualnu međusobnu povezanost.*
3. *Koje važne aktivnosti treba sprovesti u fazi oblikovanja novog sustava?*
4. *Objasnite aktivnost oblikovanje i integracija mreže.*
5. *Objasnite aktivnost oblikovanje programa i korisničkog sučelja.*
6. *Objasnite aktivnost oblikovanje sistemskog sučelja.*
7. *Objasnite aktivnost oblikovanje i integracija baza podataka.*
8. *Objasnite aktivnost prototipiranja.*
9. *Objasnite aktivnost oblikovanje i integracija sistemskih kontrola.*
10. *Navedite smjernice za primjenu tehničkih kriterija dobrog oblikovanja sustava.*
11. *Objasnite karakteristike tradicionalnog pristupa razvoju programske podrške.*
12. *Objasnite karakteristike objektnog pristupa razvoju programske podrške.*
13. *Objasnite vezu između radnog dijagrama i dijagrama toka podataka.*
14. *Kako se u dijagramu toka podataka može pratiti životni ciklus entiteta?*
15. *Kako iz radnog dijagrama generirati pseudokod?*
16. *Kako su u dijagramu paketa povezani slučajevi korištenja i dijagrami klasa?*
17. *Što su metode u klasama?*
18. *Što je to relacijska baza podataka?*
19. *Koje korake treba proći da bi se iz E-V modela izgradila relacijska baza podataka?*
20. *Što je to objektna baza podataka?*
21. *Koje korake treba proći da bi se iz dijagrama klasa izgradila objektna baza podataka?*
22. *Navedite po jedan primjer sheme objektne baze podataka za klase koje su u vezama: jedan naprema više, više naprema više i za generalizaciju.*

9. Testiranje i održavanje sustava

Poznato je da implementacija nedovoljno testiranog novog sustava i otklanjanje grešaka *u hodu* kod korisnika stvara nepovjerenje u sustav i podatke koje sustav obrađuje. Nepouzdani podaci uzrok su nesigurnosti korisnika pa korisnik često ne želi prihvatiti novi sustav, a uz to i odbija rad s njim. Ovakav stav korisnika nije moguće promijeniti niti mjerama prisile. Stoga je važno dobro testirati sustav koji se izgrađuje, ne samo na kraju verzije programskog rješenja nego i tijekom razvoja.

U praksi je često usvojeno pogrešno mišljenje da je razvoj i izgradnja informacijskog sustava značajno kompliciraniji i teži posao od održavanja dovršenog sustava.

To međutim nije točno, ovisno o tome kojom metodom je razvijan novi sustav te kako i kada sprovesti korake testiranja.

Ako je programsko rješenje razvijeno metodom prototipa, nema posebnog testiranja i uvođenja u produkciju, jer su sve aktivnosti provedene tijekom razvoja.

Ako je programsko rješenje razvijeno primjenom linearnog pristupa, nakon završetka razvoja sustava, a prije njegova produktivnog korištenja, treba testirati program. U tom slučaju testiranje mora provesti korisnik (informatičari su svoj dio testiranja proveli tijekom rada na razvoju) i tek nakon njegove potvrde da programi ispravno rade može se stvarno početi primjenjivati novi sustav.

Tijekom razvoja informacijskog sustava aktivni bi sudionici trebali biti i korisnici, tako da je njihovo uvođenje u rad s novim sustavom relativno jednostavno. Kvalitetna korisnička dokumentacija i *online* pomoć značajno pomažu pri otklanjaju problema u radu novih korisnika sustava. Poželjno je da razdoblje testiranja sustava i njegova uvođenja u rad traje od jedan do tri mjeseca, kada korisnici uz stari sustav paralelno koriste i novi.

Proces održavanja počinje onog trenutka kada je informacijski sustav pušten u produktivan rad. Održavanje se može odnositi na:

- popravak grešaka koje su utvrđene nakon što se sustav počeo koristiti;
- unapređenje i doradu postojećih funkcionalnosti programskog rješenja;
- dodavanje novih funkcionalnosti i/ili novih podsustava postojećem rješenju.

Poznavanje logike programa najlakše se postiže pridržavanjem standardnih metoda i tehnika razvoja informacijskog sustava, pri čemu je posebno povoljno ako je osoba koja radi na održavanju sudjelovala u razvoju u nekom od njegovih segmenata.

Izmjena postojećih programa najosjetljiviji je dio održavanja. Provodi se u tri koraka:

1. Korak čini oblikovanje promjene programa, što često uključuje ponovno modeliranje procesa te doradu modela podataka i resursa.
2. Korak čini izrada programskog koda koji mijenja postojeći programski kod. Ponekad se taj posao obavlja relativno brzo i jednostavno, kada se koriste generatori aplikacija. Ako, međutim, ne postoji kvalitetna dokumentacija programskog rješenja koje treba mijenjati, ovaj posao može biti mukotrpan i slijedi uglavnom nakon povratnog reinženjeringa.
3. Korak je provjera ispravnosti rada novog programa, te umanjivanje takozvanog *bočnog efekta* promjene na druge programe ili na druge funkcionalnosti sustava. Može se dogoditi da tražena nova funkcionalnost programa ukida neku od postojećih opcija, ili da im je suprotstavljena. U tom slučaju korisnik se mora jasno očitovati o tome da je upoznat s problemom i odlučiti koju verziju programa zapravo želi. Slučajne posljedice promjene programa moraju se utvrditi i otkloniti u ovom koraku.

Ponovna validacija programa znači provesti sustavno testiranje starih i novih funkcionalnosti. Taj posao ponovno radi korisnik.

Procedura prijave grešaka različita je u različitim poslovnim sustavima. Ona ovisi i o tomu jesu li informacijski sustav razvijali vlastiti informatičara ili je kupljen od neke informatičke tvrtke. U tu svrhu svako poduzeće treba postaviti vlastita pravila, ili usvojiti formalna pravila koja propisuje vanjski partner. To mogu biti :

- način prijave greške ili zahtjeva
- određivanje vrste održavanja
- određivanje prioriteta zahtjeva
- rok za provođenje promjene
- kalkulacija cijene promjene i slično.

Ponekad se za prijavu grešaka popunjavaju propisani obrasci na papiru, intranetu ili Internetu. Zahtjev za promjenom postojećeg informacijskog sustava uvijek mora biti dokumentiran i potpisan od strane ovlaštene osobe.

Literatura

[1] Boehm B., Turner R., Balancing Agility and Discipline, Addison Wesley, 2004.

[2] Chang M., Agile and Crystal Clear with Library IT Innovation, VALA2010, 15th Conference and Exibition, Melbourne, 2010.

[3] CMS, Selecting a Development Approach, Department of Health and Human Services, USA, 2008.

[4] Fertalj K., Kalpić D., Projektiranje informacijskih sustava, FER-ZPM-GRZ, power-point prezentacija, akademska godina, 2004/05.

[5] Fowler M., Scott K., UML Distilled Second Edition, Addison Wesley, 1999.

[6] Grady R. B., Caswell D. L., Software metrics: Establishing a Company-Wide Program, Prentice-Hall, 1987.

[7] IBM Rational Unified Process (RUP), http://www-01.ibm.com/software/awdtools/rup/, (rujan, 2010.)

[8] http://www-01.ibm.com/software/rational/, (studeni, 2010.)

[9] IEEE Standard for Developing Software Life Cycle Processes, IEEE Std 1074-1997, (Revision of IEEE Std 1074-1995; Replaces IEEE Std 1074.1-1995)

[10] Klarin, K., Klasić, K., Projektiranje informacijskih sustava (skripta), Veleučilište u Splitu, Split, 2003.

[11] Klasić K., Klarin. K., Informacijski sustavi – načela i praksa, Intus informatika d.o.o., Zagreb, 2009.

[12] Kulak D., Guiney E., Use Cases: Requirements in Context, Second Edition, Addison Wesley, 2003.

[13] Kurbel E.K., The Making of Information Systems, Software Engineering and Management in Globalized World, Springer-Vrelag, 2008.

[14] Louridas P., Software Development Processes, Department of Management Science and Technology, Athens University of Economics and Business, 2005.

[15] Manger R., Baze podataka, skripta, PMF – Matematički odsjek, Zagreb, 2011.

[16] Marsic I., Software Engineering, Rutgers: The State University of New Jersey, 2011., *http://www.ece.rutgers.edu/~marsic/books/SE/* (lipanj, 2011.)

[17] Pressman S.Roger, *Software engineering, A Practioner's Approach*, sixth edition, McGrow Hill, 2005.

[18] Project Management Body of Knowledge, PMBOK Guide, Project Management Institute, Pennsylvania, USA, 2000.

[19] Robertson E., Runing the River: A Survival Guide for the Waterfall Model, 2007.

[20] Rosenberg D., Scott K., Applying Use Case Driven Object Modeling with UML, Addison-Wesley, 2001.

[21] Satzinger J.W., Jackson R.B., Burd S.D., Systems Analysis and Design in a Changing World, Course Technology, Thomson Learning, 2002.

[22] Schwaber K., Sutherland J., Vodič za Scrum: pravila igre, scrum.org, listpad, 2011.

[23] Scott K., UML explained, Addison-Wesley, 2001.

[24] Scott W. A., A Manager's Introduction to The Rational Unified Process (RUP), Ambysoft Inc., 2005., www.ambysoft.com/, (studeni, 2010.)

[25] Sommerville I., Software Engineering, Eighth Edition, Addison-Wesley, Harlow, UK, 2007.

Popis slika

Popis tablica