Suvremeni trendovi razvoja informacijskih sustava

Farkaš, Albert

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

Albert Farkaš

SUVREMENI TRENDOVI RAZVOJA INFORMACIJSKIH SUSTAVA

Diplomski rad

Pula, 2015.

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

Albert Farkaš

SUVREMENI TRENDOVI RAZVOJA INFORMACIJSKIH SUSTAVA

Diplomski rad

Broj indeksa: 288-ED, redovni student
Studijski smjer: Poslovna informatika

Predmet: Razvoj informacijskog sustava
Mentor: prof. dr. sc. Vanja Bevanda

Pula, lipanj 2015.

IZJAVA O AKADEMSKOJ ČESTITOSTI

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

Student:

U Puli, 08. 06. 2015. _____

SADRŽAJ

1. UVOD

Informacijski sustavi su u današnje vrijeme sveprisutni i određuju svaki aspekt ljudskog (suvremenog) života. Ovi sustavi nalaze primjenu u svim segmentima društva koji se pojavom sve modernijih i pristupačnijih platformi (poput pametnih telefona i tableta) progresivno povećavaju, pa se tako osim poslovnih informacijskih sustava, sve više koriste sustavi namijenjeni za zabavu. U oba slučaja riječ je o zahtjevnim korisnicima, ali sa drugačijom vrstom zahtjeva. U središnjem smo se dijelu rada, radi obujma primjene informacijskih sustava, koncentrirali na razvoj web i mobilnih aplikacija, budući da su to aktualni i sve popularniji predstavnici suvremenih informacijskih sustava.

Cilj rada je dati uvid u razne metodologije i alate koji se koriste pri razvoju informacijskih sustava, te usporediti tradicionalne i suvremene metode razvoja informacijskih sustava.

Hipoteza ovog diplomskog rada je da razne suvremene metodologije i postojanje raznih alata za različite potrebe nisu garancija za uspješniji razvoj informacijskih sustava u kojima se korisnički zahtjevi neprestano mijenjaju.

Rad je podijeljen na pet poglavlja. Nakon uvoda, u drugome se dijelu rada definiraju pojmovi vezani uz informacijske sustave. Također se daje pregled faza i modela razvoja informacijskih sustava, koji će nam poslužiti za usporedbu klasičnih (tradicionalnih) metoda i suvremenih metoda, ali i kao kratak uvid u evoluciju i obim raznih metoda koje se koriste pri razvoju informacijskih sustava. Treće poglavlje objašnjava sam proces razvoja informacijskih sustava i njegovu složenost. Posebno se pridaje važnost modeliranju, budući da je utvrđivanje zahtjeva ključno polazište ka razvoju kvalitetnog i zadovoljavajućeg informacijskog sustava. Na kraju poglavlja, opisane su dvije metodologije (JAD i RAD) koje su, na neki način, nadogradnja tradicionalnih metoda, ali se isto tako ne mogu sasvim klasificirati u „suvremene“ metode – iako je, kao što ćemo vidjeti u radu, teško odrediti razliku između klasičnih i suvremenih pristupa razvoju informacijskih sustava. U centralnom dijelu rada, odnosno četvrtom poglavlju, obrađuju se suvremeni trendovi razvoja informacijskih sustava. Najprije se navode i opisuju agilne metode razvoja, budući da su našle najveću primjenu u razvoju web i mobilnih aplikacija. Nakon toga se dotičemo trendova razvoja web i mobilnih aplikacija, gdje možemo vidjeti neke aktualne statističke podatke i predviđanja tehnoloških trendova koja će obilježiti naredne godine u IT svijetu. Budući da smo se u radu osvrnuli na razvoj web

aplikacija, koje na neki način predstavljaju suvremene informacijske sustave, definiraju se određeni pojmovi web inženjerstva, kao i tehnologije potrebne za izradu tih istih aplikacija. Poglavlje završavamo sa modelima razvoja u web inženjerstvu koji nam pokazuju koliko se zapravo razvoj web aplikacija razlikuje od razvoja standardnih informacijskih sustava. U petom poglavlju, odnosno zaključku, iznose se zaključna razmatranja o obrađenoj temi.

Pri izradi diplomskog rada korištena je strana i domaća literatura raznih autora, u vidu knjiga, znanstvenih članaka i raznih stručnih materijala objavljenih na fakultetima. Također su kao izvor (aktualnih) informacija korištene različite Internet stranice. U radu su primijenjene različite znanstveno-istraživačke metode, poput induktivne i deduktivne metode, metode analize, metode sinteze, metode deskripcije i metode kompilacije.

2. DEFINIRANJE INFORMACIJSKIH SUSTAVA

2.1. Osnovni pojmovi i definicije

Informacijski sustav je sastavni dio svakog poslovnog sustava, dok je razvoj informacijskog sustava kontinuiran i zahtjevan posao. Informacijski sustav je prema Pavliću [8] skup povezanih dijelova kojima je cilj pribaviti i prenijeti informacije i podatke za funkcioniranje, planiranje, odlučivanje te upravljanje poslovnom organizacijom. U današnjim vremenima internetizacije i sve veće ovisnosti o tehnologiji, poslovne organizacije (bilo da je riječ o malim ili velikim) ne mogu opstati bez informacijskih sustava koji se baziraju na informacijsko komunikacijskoj tehnologiji. No, treba napomenuti kako se informacijski sustav ne mora nužno koristiti informacijskom tehnologijom. Informacijsku tehnologiju možemo promatrati kao sredstvo, a ne kao svrhu postojanja informacijskog sustava. Zadatak informacijskog sustava je prikupiti, pohraniti, čuvati, obraditi te naposljetku isporučiti informacije koje su ključan resurs za uspješno odvijanje poslovnih procesa.

Pojam informacijski sustav sastoji se od dvaju pojmova i potrebno je definirati dva termina: „sustav“ i „informacija“. Sustav je skup dijelova (elemenata), veza između dijelova te osobina dijelova svrsishodno organiziranih za neki proces. Za informaciju su vezani pojmovi: podatak, signal, kanal, znanje.

Varga definira podatak [5] kao skup znakova zapisanih na određenom mediju, npr. papiru (knjiga, notes), filmu (fotografija, filmski zapis), magnetskom mediju (datoteka na disketi ili disku u računalu, videozapis na videokaseti) itd. Podatak je zapravo skup znakova odnosno diskretna vrijednost neke fizičke veličine. Primjerice, broj „1988“ je podatak koji se sastoji od četiri znaka. Znajući vrijednost navedenog broja, a ne znajući kojoj vrsti stvari u sustavu pripada i što znači, kažemo da imamo neki podatak. Podaci su nositelji informacija i mogu se zapisivati na različite medije. Drugim riječima, podaci su sirove činjenice.

Informacija je protumačeni podatak, odnosno činjenica s određenim značenjem. Ona donosi novost, o nečemu obavještava, otklanja neizvjesnost i služi kao podloga za odlučivanje. Vrijednost informacija određuje primatelj. Ukoliko primjerice za podatak „1988“ napišemo da se odnosi na godinu rođenja, tada imamo informaciju. Ta zapisana činjenica je informacija za svaku osobu koja će je znati protumačiti iz svojega konteksta znanja o stvarnosti. Za osobu koja

ne zna pročitati rečenicu na tom jeziku, ta činjenica je podatak. Dakle, ovisno o kontekstu, informacija za nekoga može biti podatak i obrnuto.

Znanje je uređen skup informacija o nekom području koji je ljudski razumljiv i koristan za bilo koje ljudsko djelovanje. Svako područje organizira svoje znanje utvrđivanjem prikladnih koncepata, njihovih međusobnih odnosa i ograničenja. Znanje nam je potrebno da bismo ovladali nekim područjem ljudskog djelovanja i u njemu radili. Znanje je skup informacija s operacijama koje se mogu provesti nad njima.

Informacijski sustav je skup organiziranih i povezanih informacija koje predstavljaju sustav i dio su toga istog sustava. Informacijski sustav je skup dobro definiranih pravila, običaja i postupaka pomoću kojih ljudi, oprema ili jedno i drugo, rade na određenom inputu sa svrhom da dobiju informacije koje će zadovoljiti potrebe određenih pojedinaca u određenoj poslovnoj situaciji. [12]

Informacijska tehnologija predstavlja najvažniju tehnologiju obrade podataka, ali postoji niz drugih elemenata koji mogu izvoditi procese nad podacima i informacijama ili na njih utjecati, kao što su ljudski resursi, sustavi kvalitete, norme i državna regulativa, organizacija, komunikacijska tehnologija itd.

Današnji zahtjevni informacijski sustavi nastali su kao evolucija sljedećih vrsta sustava:

- elektronska obrada podataka (engl. *Electronic Data Processing - EDP*)
- transakcijski informacijski sustav (engl. *Online Transaction Processing - OLTP*)
- upravljački informacijski sustav (engl. *Management Information System - MIS*)
- sustav za potporu odlučivanju (engl. *Decision Support System - DSS*)
- sustav za automatizaciju ureda (engl. *Office Automation System - OAS*)
- sustav baziran na znanju (engl. *Knowledge Based System - KBS*)

Pojedinačni navedeni pristupi razvoju informacijskog sustava zanemaruju pitanje integracije, pa samim time ne mogu biti dobro rješenje za poslovnu organizaciju. Integracijom tih vrsta informacijskih sustava u cjeline, pokušava se pomoći procesima u organizaciji koristeći informacijske tehnologije i pripadajuće alate, jer najbolji alati ugrađeni u DSS neće dati potreban rezultat ukoliko nisu povezani s poslovnim procesima.

Pavlić [8] definira temeljne funkcije informacijskog sustava:

- Prikupljanje i upis podataka u bazu podataka: ostvaruje se programima za prihvat podataka, pri čemu se podaci s ulaznih i izlaznih dokumenata upisuju u bazu podataka.
- Obrada (procesiranje) podataka: ostvaruje se programima za automatizaciju procesa sustava na način da se uzimaju podaci iz baze podataka i novi podaci iz sustava te generiraju podaci za bazu i sam sustav.
- Prikaz i ispostavljanje podataka iz baze podataka: ostvaruje se mnogobrojnim programima za izradu izvješća.
- Čuvanje (dokumentiranje, trajno pohranjivanje) podataka: omogućuje dostupnost i sigurnost povijesnih podataka za buduće potrebe sustava (bilo u obliku papirnate ili digitalne dokumentacije).

Cilj informacijskog sustava je dostaviti u organizaciju pravu informaciju, na pravo mjesto, u pravo vrijeme, te uz minimalne troškove. Šehanović i Žugaj [12] funkcije informacijskog sustava dijele na: prikupljanje podataka, obradu podataka, pohranjivanje podataka i informacija, te dostavu podataka i informacija korisnicima, dok njegove elemente čine:

- Hardver (engl. *Hardware*): fizičke komponente.
- Softver (engl. *Software*): programska rješenja na kojima se temelji primjena hardvera.
- Lifver (engl. *Lifeware*): ljudi koji rade s informacijskom tehnologijom, bilo kao informatički inženjeri ili pak kao korisnici sustava.
- Orgver (engl. *Orgware*): organizacijski postupci, metode i načini vezanja prethodne tri komponente u skladnu funkcionalnu cjelinu.
- Netver (engl. Netware): koncepcija i realizacija komunikacijskog povezivanja svih elemenata sustava u skladnu cjelinu.

2.2. Klasifikacija informacijskih sustava

Informacijski sustavi mogu biti svrstani u različite klase. S obzirom na specijaliziranost funkcionalnosti informacijskih sustava, oni mogu biti: opći (generički) ili specijalizirani. S obzirom na menadžment i količinu ugrađenog znanja, informacijski sustavi mogu biti: transakcijski informacijski sustavi, sustavi potpore odlučivanju, ekspertni sustavi i sustavi automatizacije ureda.

Informacijski sustavi u svom razvoju prolaze kroz razne faze. Šehanović i Žugaj [12] navode sveukupno pet faza, koje nisu strogo odijeljene, već se viša faza oslanja na nižu. To su:

1. **Obrada podataka** (engl. *Data Procesing - DP*): je faza obrade podataka. Glavna ideja ovog sustava je dobivanje elementarne informacije.
2. **Upravljački informacijski sustav** (engl. *Managament Information System - MIS*): osigurava uvide u sve informacije potrebne menadžmentu za upravljanje organizacijom. Faza koja pokušava rješavati probleme operacijskog upravljanja u poduzeću uporabom distribuiranog unosa podataka na mjestu njihova nastanka i fleksibilnim izvršavanjem. Tu se javljaju slijedeći podsustavi:
    - marketing informacijski sustav,
    - financijski informacijski sustav,
    - informacijski sustav praćenja proizvodnje,
    - kadrovski informacijski sustav i drugi.
3. **Sustav za potporu odlučivanju** (engl. *Decision Support System - DSS*): je vrsta računalnih informacijskih sustava čiji je cilj pomoći menadžmentu u donošenju važnih odluka. Taj sustav ima mehanizme za organizaciju informacija, identifikaciju i dohvat informacija, analizu i transformaciju informacija, izbor modela odlučivanja i analizu dobivenih rezultata. Njihov cilj je potpora upravljanju i odlučivanju na svim razinama. Primjenjuju se u dinamičkim gospodarskim granama. Dobro poslovno odlučivanje temelji se na odgovarajućoj informacijskoj podršci, a ovakvi sustavi nude i modele na temelju kojih se podaci transformiraju u upravljačke informacije.
4. **Ekspertni sustav** (engl. *Expert System - ES*): je vrsta računalnih informacijskih sustava zasnovanih na upravljanju znanjem, te može oponašati ulogu stručnjaka na nekom specifičnom polju i to tako da ima sposobnosti zaključivanja i korištenja znanja iz baze znanja. Ekspertni sustavi oponašaju znanje eksperta. Ideja sustava sastoji se u integriranju znanja određenog broja eksperata za pojedine oblasti i njihovom povezivanju u bazu znanja poduprtu veoma dobrim bazama podataka.
5. **Izvršni informacijski sustav** (engl. *Executive Information System - EIS*): podupire izvršavanje poslovnih odluka. Predstavlja vezu između najboljih mogućnosti DSS-a i početnih pogodnosti ES-a. Podrazumijeva se nazočnost potpune podrške svih elemenata računala, pratećih uređaja i posebno komuniciranja. Obično se smatra specijaliziranim oblikom DSS-a.

Informacijske sustave s obzirom na vrstu softvera koji koriste možemo podijeliti na: **generičke** i **naručene**.

Generički informacijski sustavi ili opći informacijski sustavi su informacijski sustavi koji koriste generički softverski proizvod (ne nužno od istog proizvođača) u organizacijama različitih vrsta (banke, tvornice, trgovačka društva) za opće funkcionalnosti potrebne različitim organizacijama. Te funkcionalnosti uključuju primjerice izdavanje faktura, obračun plaća, plaćanje računa, mrežno planiranje projekata, podrške odlučivanju itd. Iako su organizacije potpuno različite, svaka od njih ima veći broj istih općih funkcija, poput planiranja, pa može koristiti isti generički softverski proizvod. Ako se generički softverski proizvod prilagođava kupcu, tada govorimo o naručenom informacijskom sustavu.

Naručen informacijski sustav ili specijalizirani informacijski sustav je informacijski sustav za specijalizirane funkcionalnosti jednog tipa organizacijskih sustava, a koristi specijalizirani softverski proizvod (softver po narudžbi). To su primjerice sustav za vođenje knjižnice, sustav za maloprodaju, bankarski informacijski sustav, sustav za rezerviranje zrakoplovnih karata itd.

Generički softver je nezavisni programski proizvod koji se prodaje na tržištu. Generički softver specificira i razvija proizvođač softvera, a ne korisnik, dok naručeni softver odnosno tzv. softver po narudžbi specificira korisnik, a razvija proizvođač softvera. Iako granica između generičkog i naručenog softvera (i informacijskog sustava) nije stroga, bitna je razlika u cijeni, vremenu uvođenja, održavanju i mogućnostima.

Generički softver ima niz početnih prednosti: nižu cijenu, ugrađene funkcionalnosti koje su definirali eksperti s tog polja, nije potrebna analiza korisničkih zahtjeva, te imamo unaprijed gotov proizvod (softver). Nedostatak je što proizvođač softvera neće ugraditi funkcionalnosti prema specijalnim zahtjevima jednog krajnjeg korisnika, već radije gleda potrebe cjelokupnog tržišta.

Neke od prednosti naručenih softvera: postojanje organizacije baze podataka i programskih modula tako da krajnji korisnik i organizacija imaju alate koji njima najbolje odgovaraju, praćenje razvoja organizacije i prilagođavanje alata novim potrebama, mogućnost integracije i povezivanja različitih informacijskih podsustava u jednu cjelinu, imaju ljude na raspolaganju samo za potrebe korisnika. Nedostaci tog pristupa: cijena je eksponencijalno viša, moguće su pogreške jer ne postoji bezgrešan razvoj, potrebno je vrijeme za izgradnju jer ne postoji gotovo rješenje.

2.3. Modeli i faze razvitka informacijskih sustava

Za razvoj informacijskog sustava koriste se metode za njegovo oblikovanje i alati za njegovu izgradnju. Zbog promjene i novih mogućnosti informacijskih tehnologija pokreće se projekt razvoja informacijskog sustava, pri čemu se postojeći informacijski sustav unapređuje i održava.

2.3.1. Faze životnog ciklusa

Ideja životnog ciklusa sustava ideja je grupiranja poslova i izvršavanja pojedinih skupina poslova potpuno ili djelomično prije drugih, u fazama. Izgradnja informacijskog sustava odvija se u fazama i ima svoj životni ciklus. Proces razvoja informacijskog sustava sastoji se od mnogobrojnih aktivnosti koje se mogu grupirati u klase sličnih aktivnosti pod nazivom faze.

Varga navodi sveukupno šest faza razvoja informacijskog sustava [5]:

1. **Planiranje informacijskog sustava**: Naziva se i fazom izrade strategije informacijskog sustava. Za manje informacijske sustave nije nužno obaviti ovu fazu, dok kod većih sustava može biti od presudne važnosti za daljnji razvoj projekta budući da menadžeri odlučuju o daljnjem nastavku projekta upravo na temelju rezultata dobivenih iz ove faze. Za početak treba ustanoviti opseg izgradnje informacijskog sustava, tako što ćemo identificirati korisnike sustava i nedostatke postojećeg sustava te ustanoviti ciljeve novog informacijskog sustava. Ciljevi se ustanovljuju početnom analizom funkcija poslovnog sustava kojoj se može pristupiti na više načina. Analiza kritičnih čimbenika bavi se pronalaženjem i analizom kritičnih čimbenika uspjeha poslovnog sustava. Analiza procesa promatra poslovne funkcije, odnosno poslovne procese koji upravljaju resursima poslovnog sustava. Ukoliko je informacijski sustav velik, razrađuje se njegova raščlamba na informacijske podsustave s pripadnim procesima, osnovnom strukturom podataka, potrebnim izvršiteljima itd. Faza završava ocjenom izvedivosti informacijskog sustava i razrađenim planom daljnjeg razvoja kroz sljedeće faze, procjenom potrebnih financijskih sredstava i drugih resursa, procjenom mogućih rizika daljnje izgradnje itd.

2. **Analiza poslovnog sustava:** U fazi analize potrebno je detaljno utvrditi korisničke zahtjeve, odnosno što informacijski sustav treba raditi da bi ispunio očekivanja korisnika. Sistem analitičari primjenjuju razne tehnike otkrivanja zahtjeva poput intervjuiranja korisnika, scenarija, slučajeva uporabe, rada na radnim sjednicama, „oluje mozgova“ (engl. *brainstorming*), proučavanja ulazno izlaznih dokumenata koji kolaju u poslovnom sustavu; da bi u suradnji s korisnicima (ili njihovim predstavnicima) definirali zahtjeve korisnika, te ih naposljetku formalno specificirali različitim modelima informacijskog sustava. Za model možemo reći da je apstrakcija, odnosno pojednostavnjena reprezentacija stvarnog sustava. Utvrđivanjem poslovnih procesa, objekata i događaja utvrđeni su korisnički zahtjevi iz kojih sistem analitičar kao i korisnik sustava mogu dobiti točan uvid u ono što će novi sustav raditi. Za precizno definiranje informacijskih zahtjeva odgovorni su korisnici jer oni najbolje znaju kako poslovni sustav funkcionira. Na temelju njihovih zahtjeva sistem analitičari izrađuju specifikacije informacijskih zahtjeva, odnosno modele koji se prikazuju različitim dijagramima.
3. **Oblikovanje informacijskog sustava:** U fazi oblikovanja razrađuje se kako će informacijski sustav raditi. Struktura podataka informacijskog sustava, prikazana dijagramima entiteta i veza, oblikuje se u bazu podataka koja će se nalaziti na računalu. Baza podataka na računalu može se sastojati od skupa relacija u relacijskoj bazi podataka. Također se moraju utvrditi tehnološki kao i organizacijski uvjeti za rad sustava, kao što su: računalo i ostala informatička oprema, potrebni programi (softveri) te organizacijski i kadrovski preduvjeti za rad informacijskog sustava (promjene u organizaciji poduzeća, dodatna izobrazba korisnika).
4. **Izrada informacijskog sustava:** U fazi izrade programiraju se prethodno definirani procesi, a kao rezultat dobivamo funkcionalne programe (softvere). Programiranje se može obavljati različitim programskim jezicima koji se razlikuju po svojoj namjeni. Klasični su proceduralni programski jezici COBOL i C, moderniji su objektni jezici C++ ili Java, a specijalizirani su npr. PHP (za izradu web aplikacija). Ova faza završava izradom, provjerom i dokumentiranjem programskog dijela informacijskog sustava.
5. **Uvođenje novog sustava u rad:** Implementacija novog informacijskog sustava u rad predstavlja fazu u kojoj treba staviti novi informacijski sustav u funkciju. Tijekom izobrazbe i pripreme za rad s novim informacijskim sustavom korisnici će se upoznati s njegovom svrhom, naučiti nove procedure obavljanja poslova te način korištenja računalne opreme na svojim radnim mjestima. Završni korak ove faze je provjera

odnosno testiranje cijelog sustava. Radi se o cjelovitoj provjeri novog rješenja uz pomoć stvarnih podataka, novih procedura i simuliranja normalnog rada novog informacijskog sustava.

6. **Održavanje informacijskog sustava:** Kao i svaki proizvod, informacijski sustav treba održavati iz dva razloga: da se isprave uočene pogreške i nedostaci otkriveni tijekom rada, te da se sustav prilagodi promjenama poslovnog sustava nastalim nakon njegova uvođenja. Ukoliko se kroz godine korištenja informacijski sustav slabo održava, sve će manje udovoljavati zahtjevima korisnika. To može dovesti do opravdanog uvođenja novog informacijskog sustava, što znači da ponovno treba krenuti u novi ciklus izgradnje informacijskog sustava kroz sve prethodno opisane faze razvoja.

2.3.2. Modeli razvoja

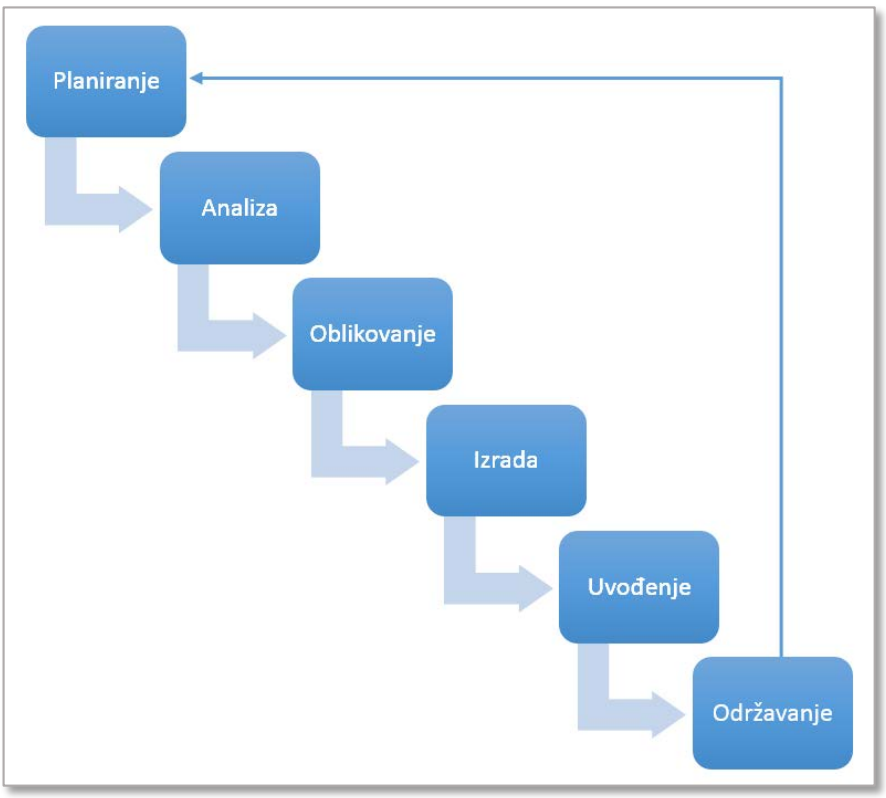
Informacijski sustav se razvija tako da se sukcesivno izvode aktivnosti i primjenjuju metode za izradu modela. Svaka faza koristi ulazni skup informacija i pomoću metode proizvodi model. Model je rezultat rada aktivnosti unutar faze. Ne postoji jedna jedinstvena metodologija za projektiranje svih vrsta informacijskih sustava, već postoji više odgovarajućih skupina metodologija, gdje su neke metodologije definirane samo za određene faze. Modeli razvoja dobri su i za definiranje faza softverskog inženjerstva. Softversko inženjerstvo je inženjerska disciplina čiji je cilj naći metodologije proizvodnje softvera i bavi se svim aspektima proizvodnje softvera. Prema Mangeru [20], softversko inženjerstvo bavi se modelima, metodama i alatima koji su nam potrebni da bi na što jeftiniji način mogli proizvoditi što kvalitetnije softverske proizvode.

Model je idealizirani prikaz sustava kojim se određuje poželjni način odvijanja i međusobnog povezivanja osnovnih aktivnosti. Metoda razvoja je profinjenje i konkretizacija odabranog modela. Metoda uvodi konkretan način dokumentiranja rezultata podaktivnosti te daje naputke vezane uz organizaciju rada, stil oblikovanja, stil programiranja itd. Modeli razvoja su opće klase ideja kako se može razvijati informacijski sustav, a koje mogu poslužiti za izgradnju strukture metodologije. Model razvoja je prijedlog semantike nizanja faza razvoja i njihovih veza.

U nastavku ćemo navesti i pobliže objasniti neke od najpoznatijih modela razvoja informacijskih sustava:

- **Model vodopada ili kaskadni model (engl. *Waterfall model*):**

Slika 1: Model vodopada



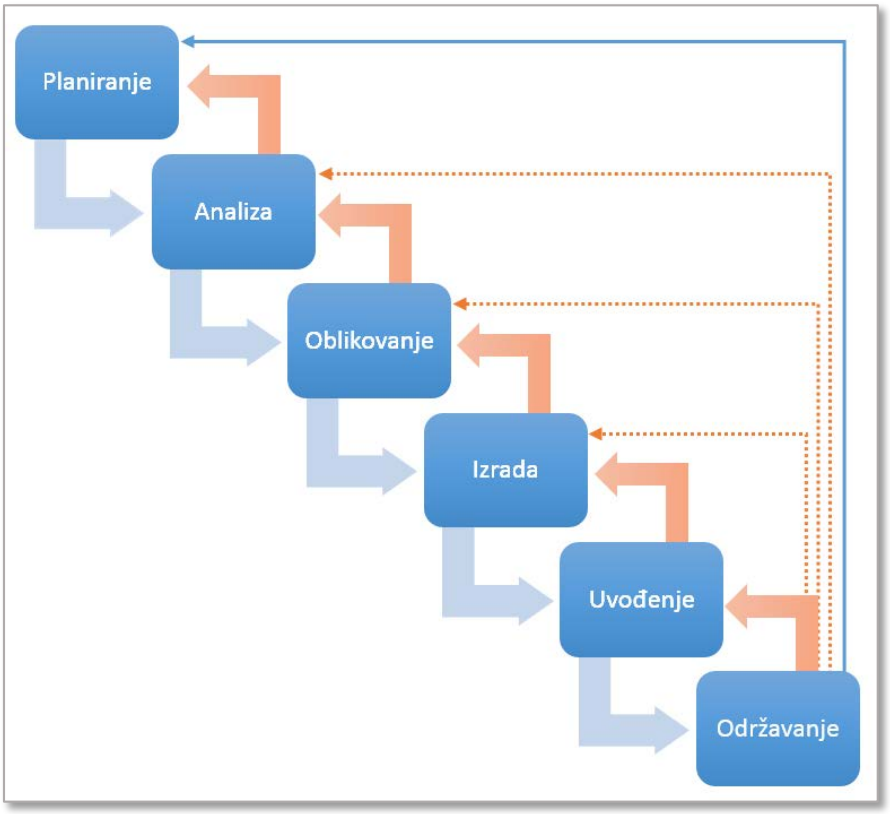
Izvor: vlastita izrada prema Pavlić, M. (2009): Informacijski sustavi, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

Kod modela vodopada, informacijski sustav je građen kao niz vremenski odvojenih aktivnosti. Kao što to možemo vidjeti na Slici 1, model je dobio ime radi oblika dijagrama koji podsjeća na vodopad, gdje se aktivnosti odvijaju kao faze sekvencijalno jedna iza druge. Svaka faza polučuje određeni rezultat koji nastavlja svoj put po vodopadu i predstavlja polazište za iduću fazu. Treba napomenuti kako u ovom modelu nisu dozvoljene naknadne promjene rezultata prethodnih faza. U iznimnim slučajevima (prilikom naknadnog otkrivanja greške) postoji mogućnost povratka u raniju fazu. Ta mogućnost se vrlo rijetko koristi jer bi poremetio normalni tijek procesa, pa bi zbog toga cijeli proces kasnio. Prednosti ovog modela: jednostavan je i lak za uporabu, prakticira se dugi niz godina (pa ljudi imaju iskustva s njim), jednostavan je za upravljanje, te pogodan je za manje projekte sa jednostavnim i jasnim zahtjevima. Nedostaci: zahtjevi moraju biti unaprijed i detaljno definirani, nema povratne informacije korisnika u procesu izrade sustava, značajni problemi sustava se otkrivaju u kasnijim fazama, nema paralelnog rada (među timovima) kod pojedinih faza što u konačnici predstavlja

neefikasnu upotrebu resursa. Iako ovaj model ima puno nedostataka, i dan danas je u širokoj upotrebi te predstavlja temelj za ostale djelotvornije postupke odnosno modele.

- **Pseudostrukturni model:**

Slika 2: Pseudostrukturni model

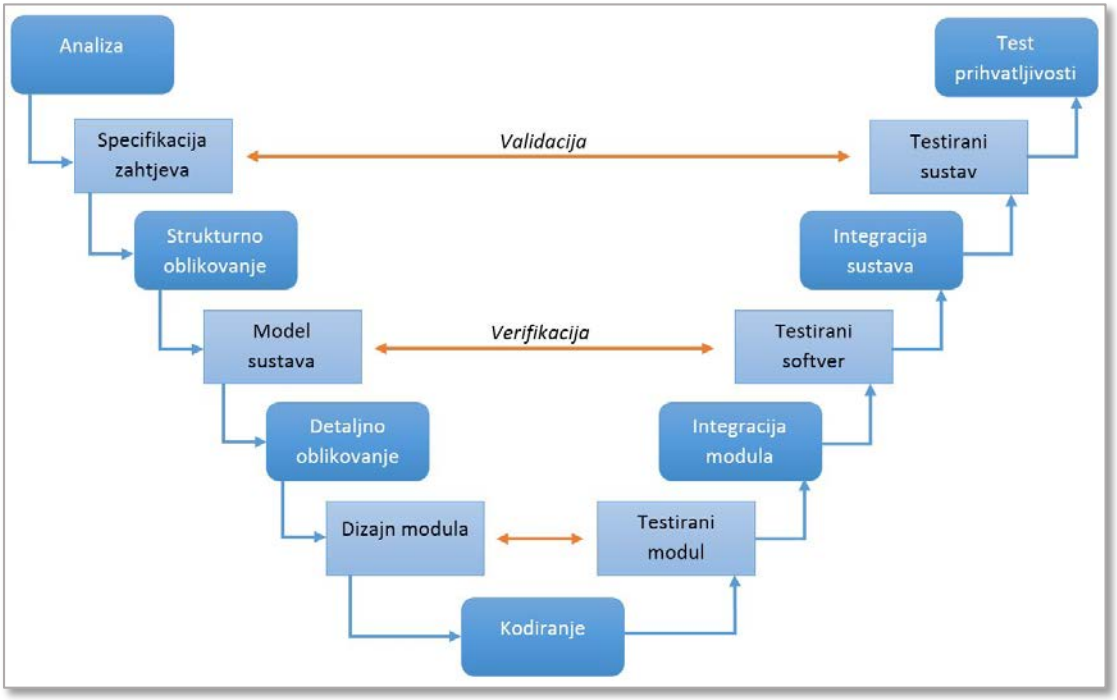


Izvor: vlastita izrada prema Pavlić, M. (2009): Informacijski sustavi, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

Pseudostrukturni model razvoja je kaskadni model s ugrađenim mehanizmom povratne veze na ranije faze. To omogućuje promjenu rezultata prethodnih faza i daljnji nastavak u prihvaćenim izmjenama. Output jedne faze koristi se kao input za slijedeću fazu, s tim da slijedeća faza neće započeti sve dok se prethodna ne izvrši. U ovom slučaju, napušta se strogo sekvencijalno odvijanje aktivnosti faza, te se sve više dopuštaju određene promjene rezultata prethodnih faza razvoja. Uvodi se određena povratna veza (Slika 2) iz kasnijih prema ranijim fazama, pa se pogreške ili nedostaci uočeni u kasnijoj fazi ispravljaju povratkom u jednu od prethodnih faza.

- **V-model:**

Slika 3: V-model



Izvor: vlastita izrada prema Pavlić, M. (2009): Informacijski sustavi, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

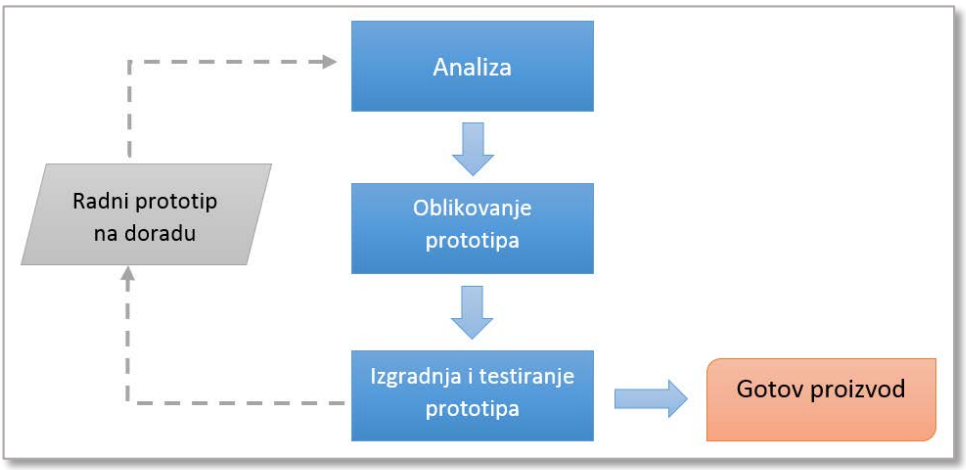
V-model nastao je na osnovi činjenice da se sustav i proizvod sastoje od dijelova, koji se dalje mogu sastojati od dijelova i da je tako moguće više razina hijerarhije. Smisao tog modela je da se najprije logički razrađuje sustav od cjeline prema detaljima sve do logičkog modula koji će kodiranjem postati programski modul (faze prikazane na lijevoj strani Slike 3). Zatim se od najmanjih programskih modula grade sve veće cjeline i tako do cjelovitog sustava (faze prikazane na desnoj strani Slike 3). Ovaj model je pogodan za manje softverske proizvode i programske jezike treće i nižih generacija.

- **Prototipski model:**

Prototipski model zasnovan je na zamisli da se izgradi prototip proizvoda koji nije gotov proizvod, te koji bi se provjeravao i poboljšavao dok ne bi zadovoljio zahtjeve korisnika. Izrada prototipa se koristi u slučajevima kada korisnici budućeg sustava ne uspiju u potpunosti izraziti svoje zahtjeve. Prototip je zapravo jednostavan i brzo razvijen program koji oponaša budući sustav. Njega je lako mijenjati i nadograditi, pa samim time *developeri* mogu isprobati razne ideje, opcije i koncepcije. Prototipski pristup

pogodan je za softverske proizvode ili dijelove informacijskog sustava. Ovaj model podržava iterativan pristup prema kojem se konačnom cilju približavamo mehanizmom povratne veze (Slika 4). Prije izrade prvog prototipa potrebno je najprije analizirati korisnikove zahtjeve i dizajnirati tzv. radni model (engl. *Working model*) koji se daje korisniku na uvid kako bi poboljšali ili izradili novi radni model. Ukoliko konačni proizvod zadovoljava i on postane dio sustava, onda takav pristup nazivamo „brzo prototipiranje“. Nasuprot tome, u „Ograničenom prototipskom modelu“ (Slika 5) konačan se prototip ne mora koristiti kao rješenje u sustavu, već se kao konačno rješenje gradi novi proizvod alatima koji ne omogućuju brze izmjene, ali dovode do kvalitetnih proizvoda, na način i u obliku kako je to potvrđeno prototipom.

Slika 4: Model brzog prototipiranja

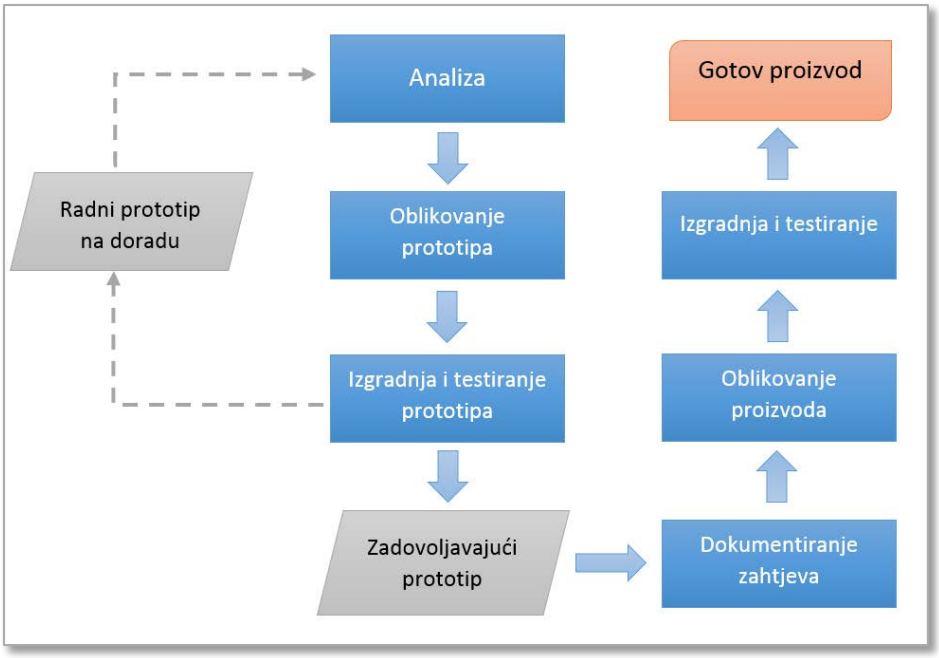


Izvor: vlastita izrada prema Pavlić, M. (2009): Informacijski sustavi, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

Brzo prototipiranje (engl. *Rapid Prototyping*) je brza iterativna izgradnja funkcionalnog proizvoda (prototipa) koji se ugrađuje kao rješenje u radni sustav. Taj pristup je uobičajen za male projekte.

Ograničeno prototipiranje (engl. *Constrained Prototyping*) je izrada nefunkcionalnoga programskog proizvoda kao sredstva za utvrđivanje korisničkih zahtjeva i prikaz konačnog izgleda proizvoda. U određenom se trenutku prototipiranje zaustavlja i izrađuje se novi konačni programski proizvod.

Slika 5: Model ograničenog prototipiranja



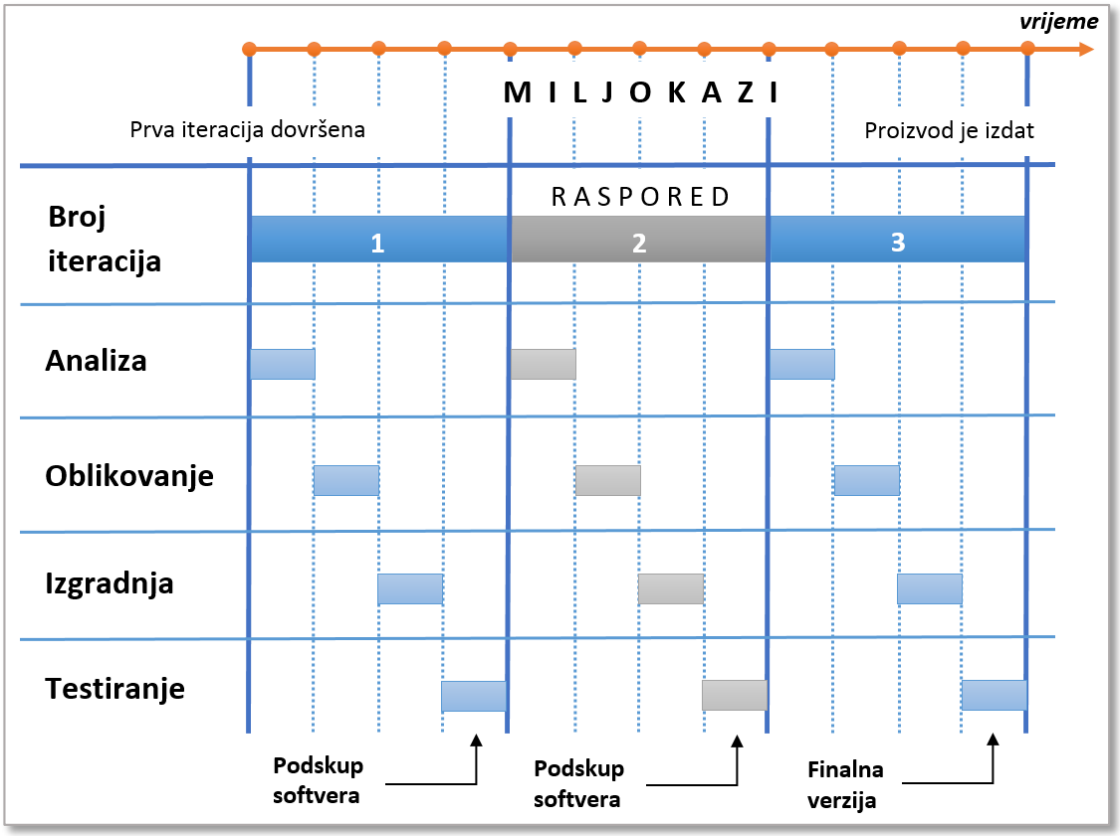
Izvor: vlastita izrada prema Pavlić, M. (2009): Informacijski sustavi, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

- **Iterativni i inkrementalni model:**

  Karakteristika modela vodopada je sekvencijalno odvijanje aktivnosti preko određenih faza. No, kao što smo naveli, on je praktičan jedino za manje projekte. Kako se sustav razvija i dorađuje, sve se više javlja potreba za izmjenom pojedine faze. Softver je prirodnije razvijati ciklički, tako što se dijelovi sustava razvijaju i testiraju, skupljaju se povratne informacije, te se na temelju toga sustav dalje razvija i usavršuje. Iterativni proces prihvaća taj ciklički razvoj, te je on tipiziran ponavljajućim izvršenjem faza modela vodopada. Iterativni[1] proces je inkrementalan[2] ukoliko je pojedina iteracija relativno mala. Inkrementalni razvoj je planirana i izvođena strategija koja omogućuje dijelovima sustava da se razvijaju u različitim vremenskim intervalima i da budu integrirani nakon što se dovrše. To implicira da iteracije ne trebaju biti izvršene serijski, već je mogu razvijati paralelno zasebni timovi (*developera*).

[1] iteracija (lat.): ponavljanje, obnavljanje
[2] inkrement (lat.): prirast, porast

Slika 6: Iterativni razvoj IS-a (sa 3 iteracije)



Izvor: vlastita izrada prema Braude, E. J. i Bernstein, M. E. (2010): Software Engineering - Modern Approaches (2nd Edition), John Wiley and Sons, New York NY, USA

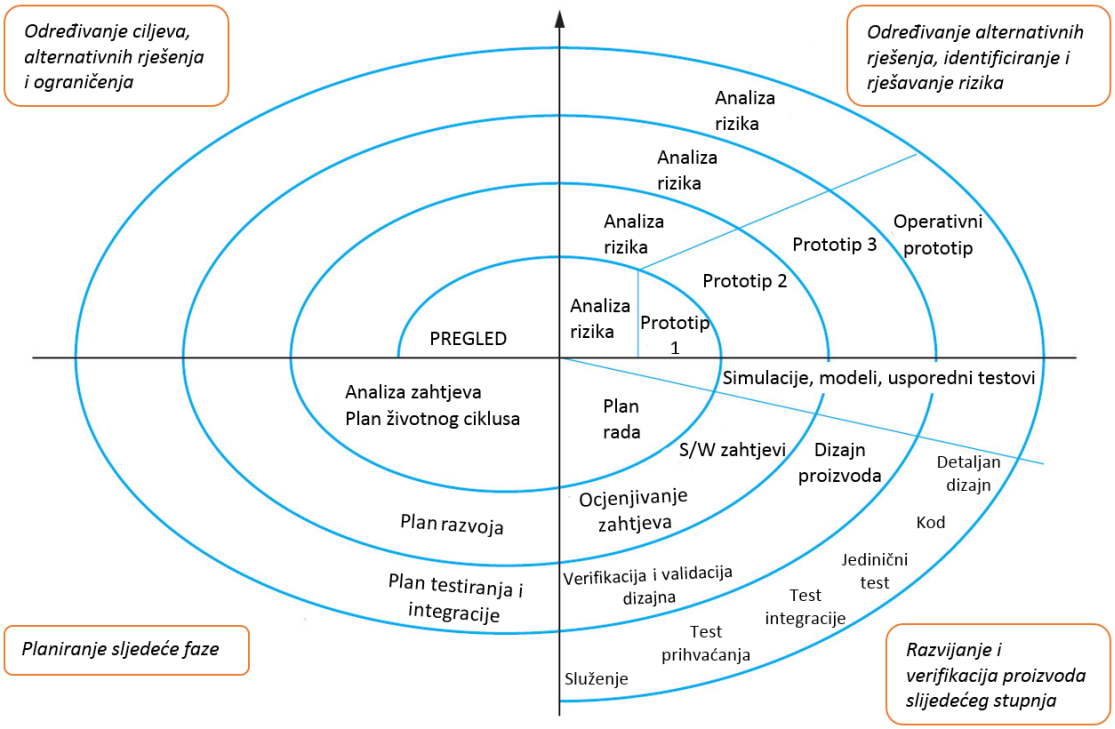
Slika 6 nam prikazuje koncept projekta gdje imamo tri iteracije. Treba napomenuti kako je output svake testirane faze zapravo podskup u izradi finalnog proizvoda koji inkrementalno sadrži više funkcionalnosti. Svaka iteracija je u stvari zaseban projekt sa svojim vlastitim skupom aktivnosti, plana, ciljeva i mjerljivih kriterija. Sustav se dijeli u niz manjih cjelina koja se svaka pojedinačno može dizajnirati, izgraditi i uvesti u sustav. Sustav se izrađuje u malim inkrementalnim koracima. Evolucijski pristup je niz paralelnih sljedova aktivnosti, takvih da svaki pojedini slijed vodi k proizvodu koji se isporučuje korisniku.

- **Spiralni model:**

  Jedan od najranijih i najpoznatijih iterativnih procesa je spiralni model Barrya Bohma. Bohm je konceptualizirao razvoj iteracija kao vanjsku spiralu, kao što vidimo na Slici 7. Spiralni model je prijedlog odvijanja faza jedne za drugom u krug. Jedan krug

predstavlja jednu fazu. Krug je izrezan u isječke, a svaki dio kruga predstavlja jednu apstraktnu vrstu posla koja se izvodi u svakoj fazi.

Slika 7: Bohmov spiralni model



Izvor: prilagođeno prema Sommerville, I. (2010): Software Engineering (9th Edition), Pearson Education Inc, Boston MA, USA

Projekt započinje u samom centru spirale, dok svaka spirala odnosno svaki njen ciklus predstavlja jednu iteraciju. Cilj je svakog ciklusa da poveća stupanj definicije sustava i implementacije uz smanjenje stupnja rizika. Nakon što se značajniji rizici definiraju i ublaže, projekt prelazi u model vodopada kao što vidimo na vanjskoj spirali. Glavna razlika između spiralnog modela i ostalih spomenutih modela razvoja je njegova eksplicitna identifikacija rizika.

2.4. Ključne uloge i sudionici

Prema Whittenu i Bentleyu, *stakeholderi* odnosno sudionici u informacijskom sustavu mogu se podijeliti u pet grupa [14]:

- Vlasnik sustava (engl. *System Owner*)
- Korisnik sustava (engl. *System User*)

- Dizajner (projektant) sustava (engl. *System Designer*)
- Graditelj sustava (engl. *System Builder*)
- Sistem analitičar (engl. *System Analyst*)

Sistem analitičar ima jedinstvenu ulogu u informacijskom sustavu. On služi kao voditelj ili podučavatelj kako bi premostio dva različita svijeta - vlasnike i korisnike sustava (koji nemaju tehničko znanje) s jedne strane, te projektante i graditelje sustava s druge strane. Treba napomenuti kako svaka grupa ima određenu ulogu u razvoju informacijskog sustava, ali isto tako svaki pojedinac može imati više od jedne uloge.

Vlasnici sustava obično dolaze iz menadžmenta, a odgovorni su za financiranje razvoja i održavanja informacijskog sustava. Njima je bitno koliko će informacijski sustav koštati, te koliko će vrijednosti ili koristi donijeti u poslovanje s takvim sustavom.

Za razliku od vlasnika, **krajnji korisnici sustava** ne brinu o vrijednosti ili koristima sustava, već o funkcionalnosti i kompleksnosti samog sustava. Korisnici predstavljaju klijente koji koriste sustav redovito radi obavljanja svog posla. Postoje dva tipa korisnika sustava: interni i eksterni. Interni korisnici su zaposlenici za čiji je posao izrađena većina informacijskih sustava, kao što su razni inženjeri, statističari, financijski službenici, odvjetnici itd. Eksterni korisnici sačinjavaju velik udio korisnika modernih informacijskih sustava. Tu ubrajamo online kupce, dobavljače ili pak zaposlenike koji rade od kuće ili kao terenski radnici.

Dizajner (projektant) sustava brine o izboru informacijske tehnologije kao i o dizajnu sustava koji koristi odabranu tehnologiju. Zadaća projektanta je prevesti korisnikove zahtjeve u tehnička rješenja. Mora dizajnirati bazu podataka, razne inpute i outpute, arhitekturu mreže i softver koje će ispuniti korisnikove želje.

Graditelj sustava izrađuje informacijski sustav i komponente koje se temelje na specifikacijama dobivenih od projektanta sustava. U manjim organizacijama, dizajneri i graditelji sustava često su iste osobe, no u većim organizacijama te su uloge odvojene. Graditelji sustava mogu biti: aplikacijski programeri, programeri sustava, programeri baze podataka, mrežni administratori, sigurnosni administratori, webmasteri itd.

Kao što smo prije naveli, glavna zadaća **sistem analitičara** je premostiti jaz između sudionika sa tehničkim znanjem i onih bez tog znanja. On mora identificirati probleme i potrebe određene organizacije kako bi odredio kako ljudi, podaci, procesi i informacijska tehnologija mogu na najbolji način poboljšati poslovanje organizacije. Kroz interakciju s ostalim sudionicima, on

zapravo olakšava sam razvoj informacijskog sustava. Sistem analitičar mora razumjeti i poslovni i tehnološki aspekt informacijskog sustava. Njegova uloga je od ključne važnosti za poslovanje, budući da sa novim sustavom mogu izmijeniti kompletnu organizaciju.

3. PROCES RAZVOJA INFORMACIJSKIH SUSTAVA

3.1. Utvrđivanje zahtjeva

Utvrđivanje zahtjeva je proces pronalaženja, analiziranja i dokumentiranja zahtjeva za budući informacijski sustav. Zahtjevi su opisi onoga što bi sustav trebao raditi, te koji odražavaju korisnikove potrebe.

Pri utvrđivanju zahtjeva treba razlikovati korisničke zahtjeve i sistemske zahtjeve. Korisnički zahtjevi su izjave u prirodnom jeziku popraćene dijagramima koji opisuju što bi sustav trebao raditi. Sistemski zahtjevi su detaljniji opisi funkcija, usluga i ograničenja informacijskog sustava.

Sistemski zahtjevi su uobičajeno klasificirani kao:

- Funkcionalni zahtjevi: specificiraju usluge koje bi aplikacija morala osigurati, navode kako bi sustav trebao reagirati na određene situacije, te ponekad izričito navode što sustav ne bi trebao raditi.
- Nefunkcionalni zahtjevi: su svi oni zahtjevi koji ne specificiraju funkcionalnosti određene aplikacije. Oni predstavljaju ograničenja na funkcije sustava, te uključuju vremenska ograničenja, ograničenja u procesu razvoja i ograničenja radi određenih standarda.

Specifikacija zahtjeva je proces zapisivanja korisničkih i sistemskih zahtjeva u dokument o zahtjevima. Navedeni zahtjevi bi trebali biti jasni, lako razumljivi, nedvosmisleni i konzistentni. Korisnički zahtjevi trebaju opisati funkcionalne i nefunkcionalne zahtjeve i to na način da ga mogu razumjeti korisnici sustava koji nemaju stručno tehničko znanje. U tom slučaju, trebalo bi izbjegavati stručne izraze i žargone, a radije opisivati zahtjeve putem jednostavnih tablica i dijagrama. S druge strane, sistemske zahtjeve koriste sudionici sa određenim tehničkim znanjem, kao početnu točku dizajniranja sustava.

Postoje slijedeći načini dokumentiranja specifikacije zahtjeva:

- Prirodni jezik: zahtjevi su pisani u numeriranim rečenicama u prirodnom jeziku.
- Strukturirani prirodni jezik: zahtjevi su pisani u prirodnom jeziku na standardnom obrascu ili predlošku.
- Jezik dizajna: zahtjevi su pisani u jeziku koji je sličan programskim jezicima, ali s više apstraktnih značenja.

- Grafičke notacije: koriste se grafički modeli, nadopunjeni sa komentarima, koji određuju funkcionalne zahtjeve sustava. Uobičajeno se koristi UML grafička notacija koja će biti objašnjena u poglavlju 3.4.
- Matematičke specifikacije: ove notacije se temelje na matematičkim konceptima, te su nerazumljive većini korisnika.

Analizom zahtjeva bavi se tim sastavljen od svih sudionika sustava, od *developera* do budućih korisnika. To je iterativan proces sa stalnim povratnim informacijama od svake aktivnosti. Proces se razvija ciklički i započinje sa otkrivanjem zahtjeva, a završava sa izradom dokumentacije zahtjeva. Aktivnosti procesa uključuju:

- Otkrivanje zahtjeva: proces koji uključuje interakciju sudionika sustava radi otkrivanja njihovih zahtjeva.
- Klasifikacija i organizacija zahtjeva: aktivnost koja služi za grupiranje i organizaciju povezanih zahtjeva.
- Pregovaranje i prioritizacija zahtjeva: aktivnost koja se bavi prioritiziranjem zahtjeva i pronalaženjem rješenja pri sukobljenim zahtjevima tijekom pregovora.
- Specifikacija zahtjeva: zahtjevi se dokumentiraju i to u obliku formalnih ili neformalnih zahtjeva dokumenata.

3.2. Otkrivanje zahtjeva

Otkrivanje zahtjeva je proces prikupljanja informacija o budućem i postojećem sustavu, kao i odvajanje korisničkih i sistemskih zahtjeva iz prikupljenih informacija. Interakcija sa sudionicima vrši se pomoću intervjua i opažanjem, a mogu se koristiti razni scenariji i prototipi za lakše razumijevanje budućeg sustava.

Najčešći način otkrivanja zahtjeva obavlja se putem formalnih i neformalnih intervjua sa sudionicima. U intervjuima, inženjerski tim postavlja pitanja korisnicima o postojećem sustavu koji koriste, ali i o sustavu koji žele razviti. Odgovaranjem na postavljena pitanja, izvode se zahtjevi. Ti intervjui mogu biti otvorenog i zatvorenog tipa. U zatvorenim intervjuima, korisnici odgovaraju na već predefinirana pitanja, dok u otvorenim intervjuima nema unaprijed definiranih pitanja. U praksi su intervjui najčešće mješavina obje vrste intervjua. Intervjui su dobri za dobivanje okvirne slike onoga što korisnici rade sa sustavom te s kakvim se poteškoćama susreću u svakodnevnom radu. No, intervjui nisu od pomoći ako

želimo razumjeti zahtjeve iz aplikacijske domene. Koristeći samo intervjuiranje, moglo bi doći do propusta bitnih informacija, pa ga je najbolje koristiti u kombinaciji s drugim tehnikama otkrivanja zahtjeva.

Prosječnim ljudima je teško shvatiti apstraktne opise, pa se radi toga pribjegava primjerima iz „stvarnog“ života, a tome mogu poslužiti scenariji. Scenariji su opisi interakcije sustava i korisnika. Svaki scenarij obično obuhvaća jedan ili mali broj mogućih interakcija.

Slučajevi uporabe (engl. *use cases*) su također jedna od tehnika otkrivanje zahtjeva koji su postali jedno od temeljnih svojstava UML-a. U svom najjednostavnijem obliku slučajevi uporabe identificiraju aktere koji su uključeni u interakciju te određuju vrstu interakcije. Interakcija se nadopunjuje dodatnim informacijama koje mogu biti u tekstualnom obliku ili u jednom ili više grafičkih modela. Slučajevi uporabe i općenito UML biti će detaljnije objašnjeni u poglavlju 3.4.

Prema etnografskom pristupu, informacijske sustave trebamo promatrati u socijalnom i organizacijskom kontekstu. Etnografija je tehnika koja se koristi za razumijevanje operacijskih procesa i kao pomoć za otkrivanje zahtjeva iz tih procesa. Budući da, u onom izvornom smislu, etnografija proučava kulturu nekog naroda, tako i sa informatičkog aspekta sistem analitičar mora preuzeti ulogu etnografa, dok korisnici sustava predstavljaju narod. Sistem analitičar će stoga, boraviti u radnom okruženju u kojem će se sustav koristiti, kako bi vodio bilješke o korisnikovim stvarnim zadacima. Na taj se način može stvoriti realna slika onoga što korisnici rade. Budući da se fokusira na krajnjeg korisnika, ovaj pristup nije uvijek prikladan za otkrivanje zahtjeva s organizacijskog aspekta. Radi toga, etnografija bi trebala biti korištena kao dopuna drugim pristupima.

3.3. Modeliranje

Modeliranje je proces razvoja apstraktnih modela sustava, a prikazuje sustav koristeći neku vrstu grafičke notacije. Modeliranje se koristi za vrijeme utvrđivanja zahtjeva kako bi uspjeli otkriti zahtjeve sustava, zatim za vrijeme dizajniranja procesa kako bi opisali sustav inženjerima koji implementiraju sustav, te naposljetku za dokumentiranje strukture i funkcioniranja samog sustava.

3.3.1. Modeliranje podataka

Informacijski sustav ujedno je i model poslovnog sustava kojem pripada. Općenito, model je apstrakcija odnosno pojednostavnjena reprezentacija promatranog stvarnog svijeta, u ovom slučaju promatranog poslovnog svijeta. Pritom se elementi stvarnog svijeta preslikavaju na odgovarajuće elemente modela, tj. model se sastoji od skupa elemenata koji prikazuju dijelove stvarnog svijeta. [13]

Modeliranje podataka možemo definirati kao postupak pronalaženja kategorija podataka i odnosa među njima. Kao rezultat različitih formalnih modela podataka javljaju se opisi podataka koji se mogu razlikovati po interpretaciji podataka i razini apstrakcije. Razlikujemo tri vrste modela odnosno opisa podataka: konceptualni, logički (implementacijski) i fizički model podataka.

Modeliranje podataka je specifično jer se obavlja kroz sve faze razvoja informacijskog sustava, pa tako u fazi planiranja imamo grubo modeliranje, dok se detaljnije modeliranje nastavlja u fazi analize. Modeliranje je zapravo proces koji počinje analizom zahtjeva informacijskog sustav, te završava izgradnjom baze podataka. Ciljevi dobrog modeliranja prema Vargi [13] su: dokumentiranje informacijskih zahtjeva, izgradnja baze podataka, povećanje vrijednosti podatkovnih resursa.

Model podataka definira što su podaci informacijskog sustava. Podacima se uvijek opisuju objekti odnosno stvari poslovnog sustava koji su relevantni za informacijski sustav, kao i njihova međusobna struktura.

3.3.1.1. Konceptualno modeliranje podataka

Konceptualno modeliranje polazi od specifikacija informacijskih zahtjeva, koje čine zahtjevi za strukturu podataka i zahtjevi za korištenjem podataka, a rezultira izrađenim konceptualnim modelom podataka. Konceptualni model podataka je cjelovit, konzistentan i neredundantan[3] opis podataka informacijskog sustava. [13] On predstavlja ključ razumijevanja informacijskog resursa organizacije budući da opisuje strukturu podataka čitavog informacijskog sustava. Izrađuju ga projektanti sustava zajedno sa sistem analitičarima i u suradnji s korisnicima, koji uostalom najbolje znaju svoje potrebe za informacijama.

[3] redundancija ili zalihost: pohranjivanje podataka na više mjesta

Konceptualni model podataka je neovisan kako logičkoj, tako i o fizičkoj implementaciji. Jasnoći i razumijevanju znatno pridonose dijagrami za prikaz modela podataka.

Model entiteti-veze (engl. *Entity-Relationship Model - ERM*) služi se slijedećim konceptima za opis konceptualnih modela podataka:

- entiteti: stvari, osobe, pojave, događaji, stanje
- veze: odnosi među entitetima
- atributi: svojstva entiteta ili veza.

Entitet je stvaran ili apstraktan predmet ili događaj o kojemu se u informacijskom sustavu prikupljaju podaci. Entitet je nešto o čemu želimo čuvati podatke u bazi podataka, nešto što može postojati ili ne postojati, ali se može identificirati. Entitet ima jasnoću te posjeduje značajke radi kojeg se može razlučiti od svoje okoline.

Veza predstavlja agregaciju dvaju ili više entiteta u novi entitet-vezu. Veza se uvijek definira na razini tipova entiteta, no realizira se povezivanjem pojedinih primjeraka entiteta dotičnih tipova. [22]

Svaki tip entiteta opisan je skupom atributa odnosno određenim obilježjima ili svojstvima.

Tablica 1: Vrste funkcionalnosti za vezu

NAZIV	OZNAKA	OPIS
jedan-prema-jedan	1:1	Jedan *entitet E1* može biti povezan najviše s jednim *entitetom E2*. Isto tako, jedan *entitet E2* može biti povezan najviše s jednim *entitetom E1*.
jedan-prema-više	1:M	Jedan *entitet E1* može biti povezan s više *entiteta E2*. No, jedan *entitet E2* može biti povezan najviše s jednim *entitetom E1*.
više-prema-jedan	M:1	Jedan *entitet E1* može biti povezan najviše s jednim *entitetom E2*. No, jedan *entitet E2* može biti povezan s više *entiteta E1*.
više-prema-više	M:M	Jedan *entitet E1* može biti povezan s više *entiteta E2*. Isto tako, jedan *entitet E2* može biti povezan s više *entiteta E1*.

Izvor: prilagođeno prema Manger, R. (2011): Baze podataka - Skripta (drugo izdanje), PMF, Zagreb

U Tablici 1 opisane su četiri vrste funkcionalnosti za vezu. Ista veza može se promatrati u dva smjera, od entiteta *E1* do entiteta *E2* i obratno. Prema Mangeru [19] funkcionalnost veze je

svojstvo koje kaže je li za odabrani primjerak entiteta jednog tipa moguće jednoznačno odrediti povezani primjerak entiteta drugog tipa. Drugim riječima, funkcionalnost je svojstvo koje kaže može li se veza interpretirati kao preslikavanje (funkcija) iz skupa primjeraka entiteta jednog tipa u skup primjeraka entiteta drugog tipa.

Nadalje, dolazimo do pojma kardinalnosti. Kardinalnost je svojstvo kojim se mogu izraziti svojstva funkcionalnosti i obaveznosti članstva. Promatrajući vezu između tipova entiteta *E1* i *E2* primjećujemo kako se kardinalnost veze (u smjeru od entiteta *E1* do entiteta *E2*) definira kao broj primjeraka od entiteta *E2* koji istovremeno mogu biti povezani s odabranim primjerkom entiteta *E1*. Kardinalnost se u suprotnom smjeru definira analogno. U Tablici 2 imamo opisane uobičajene vrste kardinalnosti. No, kardinalnost nije moguće u potpunosti točno izraziti, pa umjesto točnog broja navodimo interval u obliku donje i gornje granice koje se označavaju oznakama „1“ i „M“.

Tablica 2: Vrste kardinalnosti veze od tipa E1 do tipa E2

OZNAKA	OPIS
1:1	Jedan *entitet E1* može biti povezan s nijednim ili najviše s jednim *entitetom E2*.
1:M	Jedan *entitet E1* mora biti povezan s točno jednim *entitetom E2*.
M:1	Jedan *entitet E1* može biti povezan s nijednim, s jednim ili s više *entiteta E2*.
M:M	Jedan *entitet E1* mora biti povezan s najmanje jednim, no možda i s više *entiteta E2*.

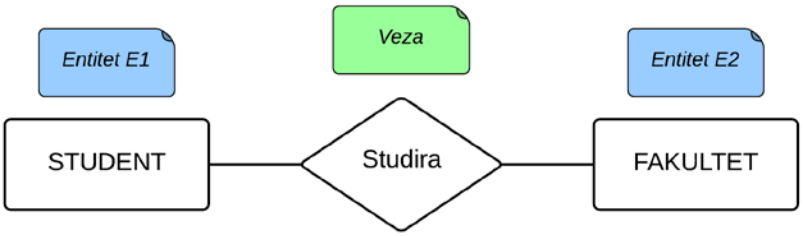
Izvor: prilagođeno prema Manger, R. (2011): Baze podataka - Skripta (drugo izdanje), PMF, Zagreb

Za identifikaciju pojava entiteta koristi se koncept ključa. Pojave entiteta međusobno se razlikuju po vrijednostima svojih atributa, pa je dovoljno pronaći takav skup atributa čije vrijednosti omogućuju jednoznačnu identifikaciju svake pojave entiteta u skupu entiteta. [13]
Ne postoje dva entiteta s posve istim vrijednostima atributa. Identifikatori ili ključevi skupa entiteta nazivamo skupove atributa čije vrijednosti jednoznačno određuju entitet u promatranom skupu entiteta.

Jedna od pozitivnih strana modela entiteti-veze je mogućnost kvalitetnog grafičkog prikaza. Navedeni dijagrami znatno pridonose lakšem razumijevanju modela podataka. U praksi se najviše koriste tzv. Chenov i Martinov grafički prikaz dijagrama entiteti-veze. Chenova

notacija[4] podrazumijeva navođenje naziva veza, dok se uloge entiteta ispuštaju. Kao što možemo vidjeti na Slici 8, naziv veze stavlja se unutar romba, koji predstavlja vezu, dok se entitet grafički prikazuje pravokutnikom unutar kojeg se nalazi ime entiteta.

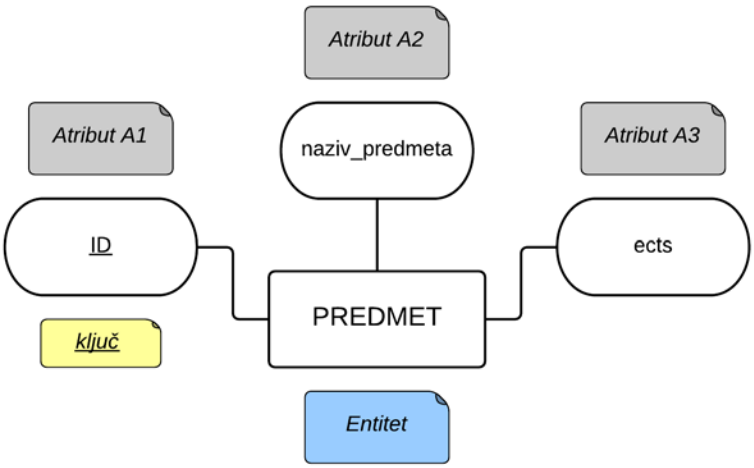
Slika 8: Primjer Chenovog dijagrama entiteti-veze



Izvor: vlastita izrada

Nadalje, atributi se crtaju unutar elipse ili kruga, unutar kojeg se upisuje ime atributa. Ključni atributi odnosno atributi s primarnim ključem uglavnom se podcrtavaju (Slika 9). U praksi se atributi ne crtaju često, budući da veliki broj atributa može učiniti dijagram nepreglednim.

Slika 9: Primjer Chenovog dijagrama entiteti-veze sa više atributa



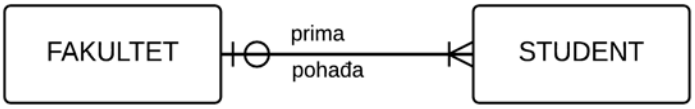
Izvor: vlastita izrada

Martinova notacija se razlikuje u tome što se u toj vrsti grafičkog prikaza koriste samo binarne veze. Veze se crtaju obično linijom koja povezuje dva entiteta. Veza se ne imenuje, nego se navode uloge entiteta koji u njoj sudjeluju, što najbolje možemo vidjeti na Slici 10. Tekst iznad linije (u vodoravno nacrtanoj vezi) imenuje ulogu entiteta na lijevoj strani, a ispod linije ulogu

[4] notacija (lat. notatio): Bilježenje, konvencionalan način koji služi za prikazivanje pojmova, veličina i odnosa u različitim disciplinama, kao i sustav znakova i simbola koji se za takav prikaz rabe.

entiteta na desnoj strani. Veza na Slici čita se ovako: „*FAKULTET* prima jednog ili više *STUDENATA*, a *STUDENT* pohađa ni jedan ili jedan *FAKULTET*“.

Slika 10: Primjer Martinovog dijagrama entiteti-veze



Izvor: vlastita izrada

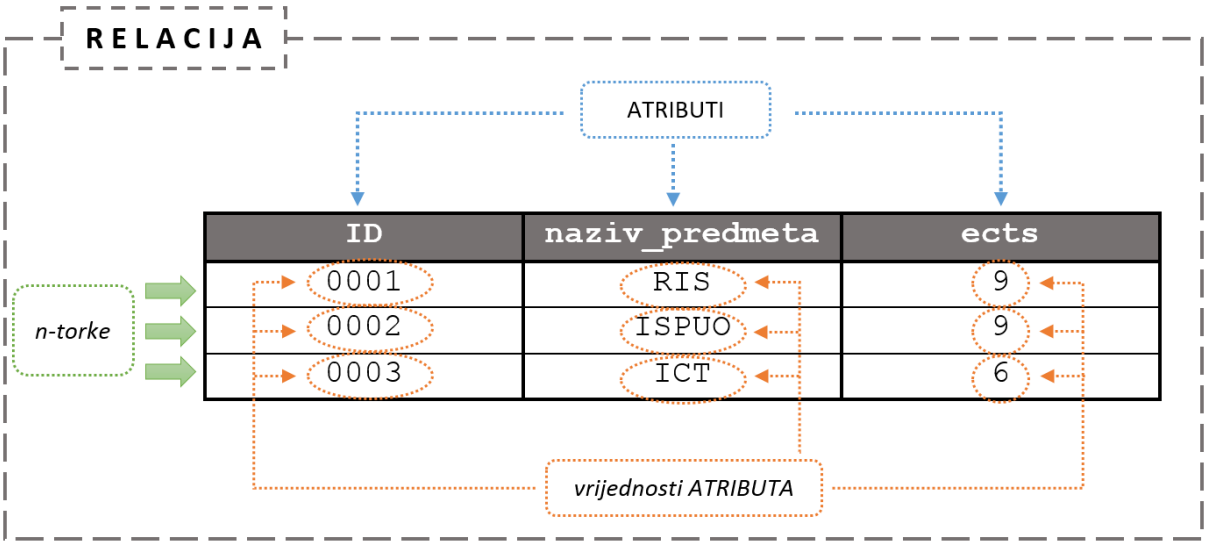
3.3.1.2. Logičko modeliranje podataka

Na temelju prethodno sastavljenog konceptualnog modela podataka te zahtjeva za korištenjem podataka, izrađuje se logički ili implementacijski model podataka. Ovaj model se naziva logičkim jer ne rezultira razradom konačne fizičke strukture podataka. Zahtjevi za korištenjem podataka služe za optimalnu transformaciju konceptualnog u radni logički model. Logički model se opisuje u obliku sheme baze podataka, a gdje se upotrebljavaju slijedeći pojmovi: relacija (datoteka), n-torka, stupac (polje) itd. Postoji više načina opisivanja logičkog modela podataka: relacijski, mrežni, hijerarhijski, objektni ili datotečni model. No, 90-ih godina relacijski model postaje najzastupljenijim modelom.

Relacijski model (engl. *Relational Data Model*) teorijski je razradio Codd. Njegov model odlikuje jednostavnost, kako u razumijevanju tako i u praktičnom radu. Relacijski model ne bavi se fizičkim smještanjem podataka u bazi podataka, već opisuje isključivo logičke aspekte podataka. Osnovne je koncepte preuzeo iz matematičke teorije skupova, a to su: domena, relacija i atribut. Domena je imenovani skup vrijednosti. Relacijom definiramo elemente skupa objekata u relacijskom modelu te se opisuje skupom tzv. n-torki. N-torka je redak relacije, dok je atribut je imenovani stupac relacije. Atribut je funkcija koja preslikava iz skupa entiteta ili skupa veza u skup vrijednosti.

Kao što možemo vidjeti na Slici 11, relacija se uobičajeno prikazuje dvodimenzialnom tablicom u kojoj redak odgovara jednoj n-torki, a stupac jednoj domeni. Relacijski model zahtijeva da se baza podataka sastoji od skupa pravokutnih tablica - relacija.

Slika 11: Primjer relacijskog modela



Izvor: vlastita izrada

U bazi podataka, relaciju ćemo od ostalih razlikovati na način da joj dodijelimo neki naziv, primjerice *PREDMET*. Svaki atribut, kojeg poistovjećujemo sa jednim stupcem relacije, također ima svoj naziv kako bismo ga razlikovali od ostalih atributa u istoj relaciji. U primjeru sa Slike 11, imamo slijedeće atribute: *ID, naziv_predmeta* i *ects*. Vrijednosti jednog atributa su podaci iste vrste. U ovom slučaju (kod atributa *ID*) to su brojevi *0001, 0002* i *0003*. Dakle, definiran je tip ili skup dozvoljenih vrijednosti za atribut, koji se također zove domena atributa. Vrijednost atributa ne smije se ponavljati i ne može se rastaviti na dijelove. Jedan redak relacije, kojeg nazivamo n-torkom, obično predstavlja jedan primjerak entiteta ili bilježi vezu između dva ili više primjeraka. Treba napomenuti kako jedna relacija ne može imati dvije jednake n-torke, budući da relaciju tumačimo kao skup n-torki. Spomenimo još kako se broj atributa zove stupanj relacije, a broj n-torki je kardinalnost relacije.

Građu relacije možemo opisati tzv. relacijskom shemom. **Relacijska shema** je redak koji se sastoji od naziva relacije i konačnog skupa naziva atributa odnosno stupaca (koji su odvojeni zarezima i smješteni u zagrade). Primarni atributi se mogu podcrtati. Primjerice, za relacijski model *PREDMET* sa Slike 11, shema bi izgledala ovako (Slika 12):

Slika 12: Primjer relacijske sheme

```
PREDMET (ID, naziv_predmeta, ects)
```

Izvor: vlastita izrada

Za rad s relacijskom bazom podataka postoji više jezika. Općenito, jezici za rad s relacijskom bazom podataka imaju tri grupe naredbi:

- naredbe za četiri osnovne operacije (pronalaženje, upis, brisanje ili izbacivanje i promjena);
- naredbe za definiranje relacijske sheme baze podataka, opis ograničenja u bazi podataka i definiranje pogleda na bazu podataka;
- naredbe za upravljanje bazom podataka.

Tijekom vremena se kao standardni jezik za rad s relacijskom bazom podataka izdvojio SQL. Danas je SQL najrašireniji jezik za rad s relacijskom bazom podataka. SQL je kratica za *Structured Query Language*. Jezik je nastao u 70-tim godinama unutar IBM-a. Najvažnija funkcija SQL-a je postavljanje upita u relacijskim bazama podataka. No, postoje i druge funkcije koje obavlja SQL, poput naredbi za stvaranje relacija, unos i mijenjanje podataka, upravljanje transakcijama, davanje i oduzimanje ovlaštenja korisnicima, kao i brojne funkcije za računanje s podacima.

3.3.1.3. Fizičko modeliranje podataka

Fizičko modeliranje polazi od logičkog modela i rezultira izrađenim fizičkim modelom podataka. Prema Vargi [13] fizički model je opis stvarne fizičke organizacije podataka, točnije baze podataka realizirane na medijima za memoriranje podataka. Model ima oblik unutarnje sheme baze podataka, primjeren korištenom sustavu za upravljanje bazom podataka i karakteristikama korištenog računala. Fizička shema zapravo je tekst sastavljen od naredbi u SQL-u.

3.3.2. Modeliranje procesa

Model procesa definira kako se obrađuju, prikupljaju i distribuiraju podaci informacijskog sustava, odnosno kako funkcioniraju objekti poslovnog sustava. Ovaj model opisuje dinamiku podataka informacijskog sustava. Također služi kao opis skupa procesa odnosno funkcija kojima se mijenjaju podaci informacijskog sustava.

3.3.2.1. Dijagram toka podataka (DTP)

Kada želimo opisati sustav, pojavljuju se problemi smještaja velikog broja informacija u projektnu dokumentaciju. Običan zapis u tekstualnom obliku ima niz mana, poput nemogućnosti održavanja i pretraživanja informacija. Rješenje problema prikaza velikog broja povezanih informacija leži u grafičkom prikazu informacija na više razina apstrakcije, odnosno u metodi za modeliranje procesa.

Dijagram toka podataka - DTP (engl. *Dataflow Diagram - DFD*) je grafičko sredstvo za modeliranje i prezentaciju procesa sustava. DTP je uveo DeMarco 1978. godine.

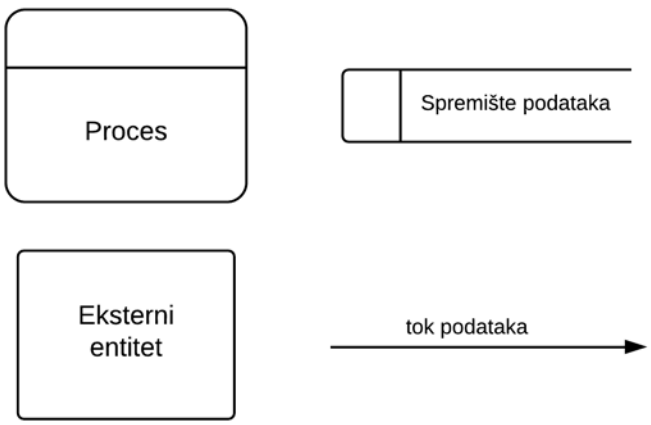
Odlike DTP-a prema Pavliću [8] su:

- DTP je grafički prikaz procesa,
- Korisnik i analitičar zajedno dolaze do modela,
- Precizno se definiraju zahtjevi korisnika,
- DTP je jezik za komunikaciju korisnika i analitičara,
- Broj različitih koncepata za DTP je mali (4 osnovna simbola).

DTP specificira što će sustav raditi na logičkoj razini. Osnovni cilj procesne analize sustava je utvrđivanje logičkog toka podataka kroz sustav. DTP pokazuje kako se podaci kreću kroz informacijski sustav, ali ne pokazuje programsku logiku kao ni korake obrade. DTP pruža logički model koji pokazuje što sustav čini, ali ne i kako to čini.

DTP koristi četiri osnovna simbola koji predstavljaju procese, tokove podataka, spremišta podataka i (eksterne) entitete, kao što vidimo na Slici 13.

Slika 13: Simboli DTP-a



Izvor: vlastita izrada

Proces prima ulazne podatke (inpute) te proizvodi outpute koji imaju različite sadržaje ili oblike. Proces može biti jako jednostavan, ali i dosta kompleksan. Simbol procesa je pravokutnik sa zaobljenim kutovima. Naziv samog procesa pojavljuje se unutar pravokutnika. Naziv procesa identificira specifičnu funkciju te se sastoji od glagola (i pridjeva ukoliko je potrebno) te imenice u jednini.

Kod DTP-a, procesni simbol možemo promatrati i kao „crnu kutiju“, zato što su inputi, outputi i glavne funkcije procesa poznati, dok su neke temeljne pojedinosti i logika procesa skriveni. Prikazujući procese kao crne kutije, analitičar može stvoriti DTP-ove koji pokazuju kako sustav funkcionira, ali također može izbjeći nepotrebne detalje. Kad analitičar želi pokazati dodatne razine detalja, on može „zumirati“ simbol procesa i stvoriti dublju odnosnu nižu razinu DTP-a. Niža razina prikazuje interno djelovanje procesa, koji bi pak mogao otkriti još više procesa, tokova podataka i spremišta podataka. Na taj način, informacijski sustav može se modelirati kao niz sve detaljnijih slika.

Tok podataka je put za podatke kako bi se preselili iz jednog dijela informacijskog sustava u drugi. Tok podataka u DTP-u predstavlja jednu ili više stavki podataka.

Simbol za tok podataka je linija sa jednom strelicom. Naziv toka podataka pojavljuje se iznad, ispod ili pored linije. Naziv toka podataka se sastoji od imenice u jednini i pridjeva, ukoliko je potrebno. Treba napomenuti, budući da proces podacima mijenja sadržaj ili oblik, najmanje jedan tok podataka mora ući i najmanje jedan tok podataka mora izaći iz svakog procesa. Također, simbol procesa može imati više od jednog „izlazećeg“ toka podataka.

Pri modeliranju procesa, moraju se izbjeći slijedeće kombinacije procesa i tokova podataka:

- Spontano generiranje: Situacija u kojoj proces proizvodi outpute, a nema ulazne tokove podataka. Budući da nema inputa, proces se naziva spontano generiranje procesa.
- „Crna rupa“: Situacija u kojoj proces ima inpute, a nema nikakvog outputa.
- „Siva rupa“: Situacija u kojoj postoji barem jedan ulazni i izlazni tok podataka, no input je nedovoljan za generiranje izlaznog toka podataka.

Spremište podataka se koristi u DTP-u kako bi predstavili podatke koje sustav pohranjuje, jer će kasnije jedan ili više procesa trebati koristiti te iste podatke. Nije važno koliko dugo vremena su podaci pohranjeni, bitno je da će pohranjeni podaci biti dostupni kada to sustavu bude potrebno.

Simbol za spremište podataka je pravokutnik koji je otvoren na desnoj strani i zatvoren na lijevoj strani. Naziv spremišta podataka pojavljuje se unutar pravokutnika i identificira podatke koje on sadrži. Naziv spremišta podataka sastoji se od imenice u množini i pridjeva, ako je potrebno.

Spremište podataka mora biti povezano s procesom i to s pomoću toka podataka. Spremište podataka mora imati najmanje jedan ulazni i izlazni tok podataka koji je povezan sa simbolom procesa. Također, dva spremišta podataka ne smiju biti međusobno povezana ukoliko između njih ne postoji proces. U iznimnim slučajevima, spremište podataka ne treba imati ulazni tok podataka, no to se dešava samo u primjerima gdje takva spremišta sadrže fiksne referentne podatke koje nisu ažurirane od strane sustava.

Simbol (eksternog) entiteta je pravokutnik gdje se naziv entiteta pojavljuje unutar pravokutnika. DTP prikazuje samo eksterne entitete koje odašilju ili primaju podatke u informacijskom sustavu. DTP zapravo prikazuje granice sustava i kako se sam sustav sučeljava s vanjskim svijetom. Svaki entitet mora biti povezan preko toka podataka sa procesom i ne smije biti direktno povezan ni sa spremištem podataka ni sa drugim entitetom.

Koraci pri izradi DTP-a:

1. **Konstruiranje kontekstualnog dijagrama**
   Prvi korak u konstruiranju seta DTP-ova je izrada kontekstualnog dijagrama (vidjeti Prilog 1). Kontekstualni dijagram je prikaz informacijskog sustava na najvišoj razini koji prikazuje granice sustava i njegov djelokrug. Pri izradi ovog dijagrama započinjemo sa smještanjem jedinstvenog simbola procesa na sredini stranice. Taj proces predstavlja cijeli informacijski sustav te ga identificiramo kao proces *0* (nulte razine). Entiteti se postavljaju oko tog procesa koji su povezani sa tokovima podataka prema središnjem procesu. Spremišta podataka nisu prikazana u kontekstualnom dijagramu, budući da se nalaze unutar sustava.
2. **Izrada DTP-a prve razine**
   Iz prethodnog koraka možemo zaključiti kako je tekstualni dijagram zapravo poput „crne kutije“ budući da je prikazan samo jedan simbola procesa. Kako bi se detaljnije prikazao informacijski sustav, treba kreirati DTP prve razine (vidjeti Prilog 2) koji „zumira“ u sustav te pokazuje glavne interne procese, tokove podataka i spremišta podataka. Treba napomenuti da, iako proširujemo kontekstualni dijagram, broj tokova podataka koji ulazi i izlazi iz središnjeg procesa (nulte razine) mora biti isti.

3. **Izrada DTP-ova druge (niže) razine**

   U zadnjem koraku trebamo kreirati tzv. DTP-ove niže razine (vidjeti Priloge 3 i 4) gdje se izrađuje niz sve detaljnijih dijagrama sve dok se funkcije ne svedu na primitivnu razinu, a to se postiže dekompozicijom. Dekompozicija ili particioniranje je detaljizacija tj. postupak detaljnijeg prikaza sustava pomoću niza detaljnih DTP-ova. Također treba paziti na konzistenciju odnosno da se ulazni i izlazni tokovi podataka ispravno usklade.

3.3.3. Modeliranje objekata

U informacijskim sustavima gdje su podaci uglavnom jednostavnog tipa, koriste se prethodno opisani modeli entiteti-veze i relacijski model podataka. No, u tim modelima ne postoje bogatiji tipovi podataka, pa se pojavom nestandardnih aplikacija (poput uredskog poslovanja, projektiranja podržano računalom, ekspertnih sustava, baze znanja), počela javljati potreba za kompleksnijim modelima. Mnogi od njih pripadaju objektno-orijentiranim modelima podataka.

Objektno-orijentirana metodologija je popularna jer se jednostavno integrira s objektno-orijentiranim programskim jezicima kao što su Java, Smalltalk, C++, Perl itd. Programeri također vole objektno-orijentirano programiranje zato što je modularno[5], moguće ga je ponovno koristiti i jednostavan je za održavanje. Objektno-orijentirana analiza opisuje informacijski sustav identificirajući objekte. Objekt je osnovni pojam objektnih modela te predstavlja stvarnu osobu, mjesto, događaj ili transakciju. On je zapravo vrlo sličan entitetu modela entiteti-veze. Objektno-orijentirana analiza je popularni pristup koja prikazuje sustav sa stajališta samih objekata pri određenoj interakciji. Krajnji proizvod objektno-orijentirane analize je objektni model, koji predstavlja informacijski sustav u pogledu objekata i objektno-orijentiranih koncepata.

Razvoj objektne paradigme započinje 60-ih godina koja vuče svoje korijene iz objektnog programiranja i objektno-orijentiranih programskih jezika, bogatih tipovima podataka (SIMULA, Smalltalk). U objektnim modelima podaci se definiraju kroz objekte, dok se sam model implementira kroz objektnu bazu podataka. Važno je naglasiti kako je u samom objektu opisano i njegovo ponašanje. Drugim riječima, sve što je vezano uz objekt opisano je

[5] Modularno programiranje: raspodjela programskih funkcija na manje neovisne module (dijelove).

u objektu. Objektnih modela ima više, a u ovom će radu detaljnije biti opisan UML (u slijedećem poglavlju).

3.4. UML

Ujedinjeni jezik modeliranja (engl. *Unified Modeling Language* - *UML*) proizašao je iz rada 90-ih godina na objektno-orijentiranom modeliranju gdje su slične objektno-orijentirane notacije bile integrirane radi stvaranja UML-a. UML je univerzalno prihvaćen kao standardni pristup za razvoj modela softverskih sustava. Cilj UML-a je bio pružiti zajedničku terminologiju objektno-orijentiranih uvjeta i tehnika pri modeliranju dijagrama kako bi mogli modelirati bilo koji projekt razvijanja sustava od analize do implementacije. U studenom 1997. godine *Object Management Group*[6] (OMG) službeno prihvaća UML kao standard za sve objektne *developere*. Tijekom sljedećih godina, UML je prošao višestruke manje izmjene. Velika revizija (UML 2.0) je dovršena 2004. godine. Trenutna službena verzija UML-a, verzija *2.4.1*, objavljena je od strane OMG u kolovozu 2011. U trenutku pisanja rada postoji još novija verzija *2.5 Beta 2*[7], koja je objavljena u prosincu 2013., ali je još uvijek u beta fazi, pa zbog toga nije uzeta u obzir.

UML omogućuje opisivanje sustava sa riječima i slikama. Može se koristiti za modeliranje različitih sustava: softverskih sustava, poslovnih sustava, ili bilo kojeg drugog sustava. Posebno se ističu različiti grafički dijagrami, kao što su *use case* dijagrami (dijagrami slučajeva uporabe) sa svojim figurama ili naširoko korišteni klasni (*class*) dijagrami. Dok sami dijagrami nisu fundamentalno novi, UML sa ujedinjenjem jezika modeliranja na svjetskoj razini, predstavlja novinu, koji je standardiziran od strane OMG-a.

Postoji nekoliko razloga za korištenjem UML-a kao jezika za modeliranje [35]:

- Jedinstvena terminologija i standardizacija notacija dovele su do značajno lakše komunikacije za sve uključene strane.
- UML raste kako rastu i zahtjevi za modeliranjem.
- UML nadograđuje naširoko korištene i dokazane pristupe.
- UML ima rasprostranjenu podršku.

[6] OMG je međunarodna udruga koja promiče otvorene standarde za objektno-orijentirane aplikacije.
[7] Službene verzije UML-a: http://www.omg.org/spec/UML/, [15. ožujka 2015.]

- Ponude za softverske sustave koje su bazirane na UML-u mogu se puno lakše usporediti.

UML nam pomaže da odredimo, vizualiziramo i dokumentiramo modele informacijskih sustava, uključujući i njihove strukture i dizajn. Korištenjem bilo kojeg od velikog broja UML alata, možemo analizirati zahtjeve buduće aplikacije i dizajn rješenja te predstaviti rezultate koristeći četrnaest standardnih tipova dijagrama UML 2.4.1.

Modelirati se može gotovo bilo koji tip aplikacije, koji radi na bilo kojem tipu i kombinaciji hardvera, operativnom sustavu, programskom jeziku i mreži. Fleksibilnost UML-a omogućuje modeliranje distribuiranih aplikacija koje koriste bilo koji middleware na tržištu. Izgrađen na osnovnom objektno-orijentiranom konceptu, uključujući klasu i operaciju, prirodno se uklapa u objektno-orijentirane jezike i okruženja, kao što su C++, Java, i noviji C#, ali može se koristiti i za modeliranje aplikacija koje nisu pisane u objektno-orijentiranim jezicima, kao npr. Fortran, VB ili COBOL.

Jedna od glavnih karakteristika UML-a je ta da je metodologija neovisna. Bez obzira koju metodologiju koristili za obavljanje analize i dizajna, možemo koristiti i UML za izražavanje rezultata. Također postoji mogućnost prijenosa UML modela iz jednog alata u skladište, ili u neki drugi alat ili pak u slijedeću fazu u razvojnom procesu. To su prednosti standardizacije.

UML 2.4.1 definira četrnaest tipova dijagrama [24] koje se koriste za modeliranje sustava. Dijagrami su podijeljeni u dvije glavne kategorije: jednu za modeliranje strukture sustava i drugu za modeliranje ponašanja, pa tako imamo:

- **Strukturni dijagrami (engl. *Structure Diagrams*):**
    1. *Class diagram* – dijagram klasa
    2. *Object diagram* – dijagram objekata
    3. *Package diagram* – dijagram paketa
    4. *Composite structure diagram* – dijagram kompozitne strukture
    5. *Component diagram* – dijagram komponenti
    6. *Deployment diagram* – dijagram razvrstavanja
    7. *Profile diagram* – dijagram profila
- **Dijagrami ponašanja (engl. *Behavior Diagrams):***
    8. *Use case diagram* – dijagram slučajeva uporabe (korištenja)
    9. *State (Machine) diagram* – dijagram stanja
    10. *Activity diagram* – dijagram aktivnosti
    - Dijagrami interakcije (engl. *Interaction diagrams*):

11. *Sequence diagram* – dijagram slijeda
12. *Communication diagram* – dijagram komunikacije
13. *Timing diagram* – dijagram vremenskog slijeda
14. *Interaction overview diagram* – dijagram pregleda interakcije

Strukturni dijagrami pružaju način predstavljanja podataka i statičkih odnosa u informacijskom sustavu. Oni pokazuju statičke strukture sustava i njegove dijelove na različitim razinama apstrakcije i implementacije te kako su međusobno povezani. Elementi u strukturnim dijagramima predstavljaju značajne koncepte sustava, a mogu uključivati apstraktni ili stvarni svijet i implementacijske koncepte.

Dijagrami ponašanja pružaju analitičaru način opisivanja dinamičkih odnosa između instanci ili objekata koji predstavljaju poslovni informacijski sustav. Oni također omogućuju modeliranje dinamičkih ponašanja pojedinih objekata tijekom njihovog cijelog životnog vijeka. Dinamičko ponašanje objekata u sustavu može se opisati kao niz promjena u sustavu tijekom vremena. Dijagrami ponašanja služe kao podrška analitičaru u modeliranju funkcionalnih zahtjeva pri razvijanju informacijskih sustava.

UML ima mnogo tipova dijagrama, pa zato i podupire stvaranje puno različitih tipova modela sustava. Prema istraživanju iz 2007. godine (Erickson i Siau) pokazalo se da velik broj korisnika UML-a dijeli mišljenje kako sveukupno pet tipa dijagrama mogu predstaviti suštinu sustava, a to su [11]:

- *Use Case diagrams*: koji pokazuju interakciju između sustava i njegove okoline.
- *Class diagrams*: koji pokazuju klase objekata u sustavu te povezanost između tih klasa.
- *State diagrams*: koji pokazuju kako sustav reagira na unutarnje i vanjske događaje.
- *Activity diagrams*: koji pokazuju aktivnosti uključene u proces ili obradu podataka.
- *Sequence diagrams*: koji pokazuju interakciju između aktera i sustava te između komponenti sustava.

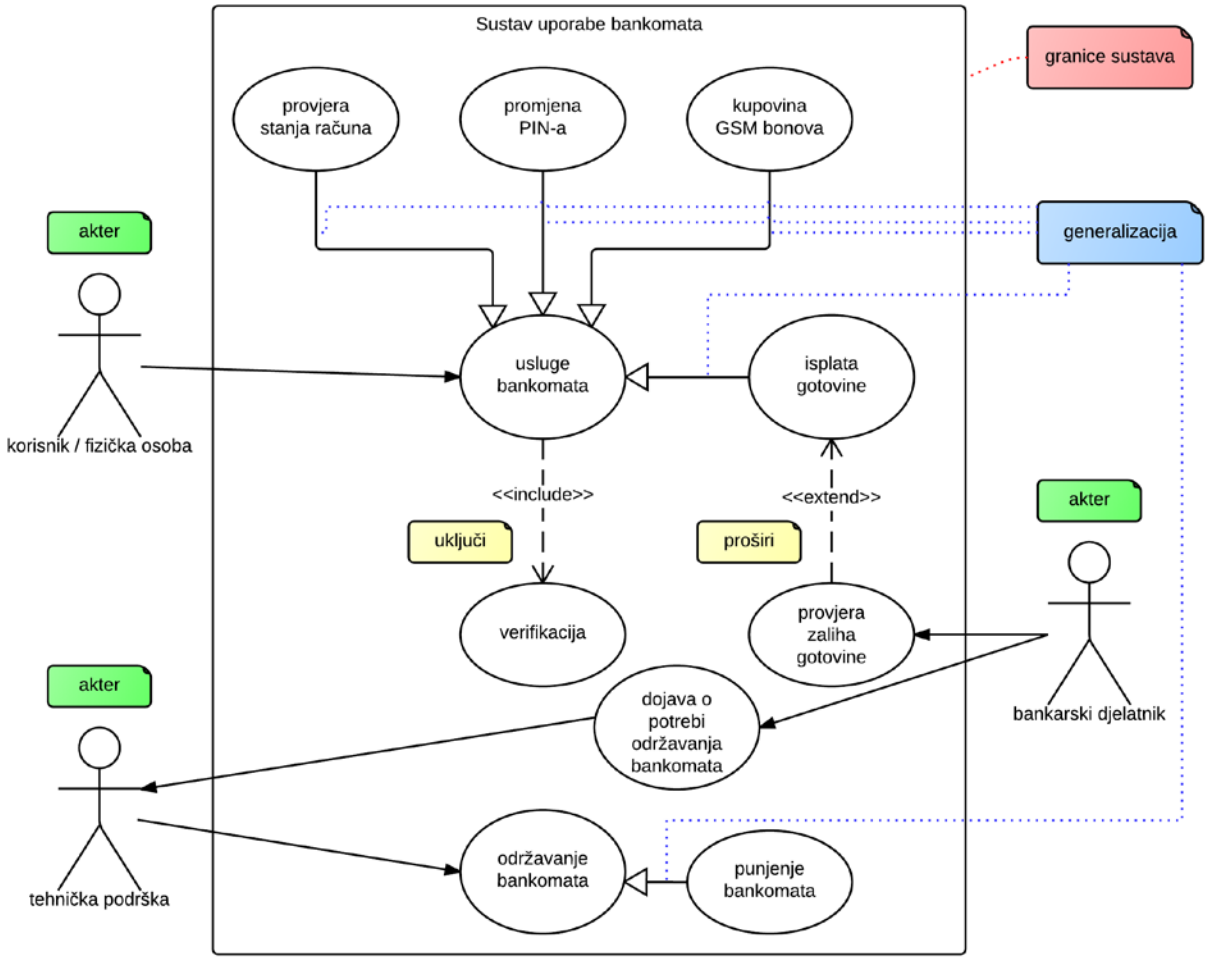
3.4.1. Dijagram slučajeva uporabe (Use Case Diagram)

Use case (slučaj uporabe) može se promatrati kao jednostavan scenarij koji opisuje ono što korisnik očekuje od sustava. Svaki slučaj uporabe predstavlja zaseban zadatak koji uključuje vanjsku interakciju sa sustavom. U svom najjednostavnijem obliku, slučaj uporabe je prikazan kao elipsa s akterima (engl. *actors*) koji su uključeni u slučaj uporabe te su prikazani kao figure.

Use case dijagrami omogućuju analitičaru modeliranje interakcije informacijskog sustava i njegove okoline. Okruženje informacijskog sustava uključuje krajnjeg korisnika i vanjski sustav koji surađuje s informacijskim sustavom. Primarna svrha use *case* dijagrama je osigurati sredstva za dokument i razumjeti zahtjeve informacijskog sustava u razvoju. *Use case* dijagrami su jedni od najvažnijih alata koji se koriste u objektno-orijentiranim sustavima analize i dizajna.

Elementi *use case* dijagrama uključuju: aktere (engl. *actors*), slučajeve uporabe (engl. *use case*), granice sustava (engl. *subject boundaries*) te skup odnosa među akterima i slučajevima uporabe. Ovi odnosi se sastoje od slijedećih tipova veza: *association* (asocijacija), *include* (uključi), *extend* (proširi) i *generalization* (generalizacija).

Slika 14: Primjer use case dijagrama (za sustav uporabe bankomata)



Izvor: vlastita izrada

Na Slici 14 vidimo primjer *use case* dijagrama za Sustav uporabe bankomata. Imamo tri figure u dijagramu koje predstavljaju aktere. Akter nije specifičan korisnik već uloga koju bi korisnik

mogao imati pri interakciji sa sustavom. Akteri, kao što vidimo, mogu unositi inpute u sustav, ali i primati outpute iz sustava. Slučajevi uporabe su povezani sa akterima preko asocijacija. Te veze prikazuju s kojim su slučajevima uporabe akteri u interakciji. Linija koja povezuje aktera sa slučajem uporabe predstavlja asocijaciju. Ona obično predstavlja dvosmjernu komunikaciju između aktera i slučaja uporabe, no možemo koristiti i strelicu kako bi odredili smjer protoka informacija. Slučaj uporabe prikazan ovalom je glavni proces koji sustav obavlja i od kojeg akteri imaju koristi. Postoje slučajevi kada slučaj uporabe uključuje, proširuje ili generalizira funkcionalnost drugog slučaja uporabe. „*Include*“ veza predstavlja uključivanje funkcionalnosti jednog slučaja uporabe unutar drugog slučaja uporabe. „*Extend*“ veza predstavlja proširenje slučaja uporabe kako bi uključili neko dodatno ponašanje. Generalizaciju koristimo kako bismo pojednostavili pojedinačne slučajeve uporabe, odnosno kako bismo više sličnih slučajeva spojili u jedan generalizirani. Slučajevi uporabe omeđeni su granicama sustava predstavljene okvirom koji definira opseg sustava i jasno ocrtava koji su dijelovi dijagrama unutar ili izvan sustava. Glavna odlika *use case* dijagrama je da omogućuje korisniku jasan pregled poslovnih procesa.

3.4.2. Dijagram klasa (Class Diagram)

Primarna svrha dijagrama klasa je stvoriti rječnik koji će koristiti i analitičar i korisnik. Dijagrami klasa obično predstavljaju stvari, ideje ili koncepte koji se nalaze u aplikaciji. Primjerice, ako se izrađuje aplikacija za plaću, dijagram klasa će vjerojatno sadržavati klase koje predstavljaju stvari, kao što su zaposlenici, doznake, te platne liste. Dijagram klasa također će prikazati odnose među klasama. Klasa je opis jednog ili više objekata koji imaju isti skup atributa i jednak opis ponašanja. Atribut klase predstavlja dio informacije koji je bitan za opis klase unutar domene aplikacije čiji problem istražujemo. Atribut sadrži informaciju koju analitičar ili korisnik misli da bi sustav trebao pohraniti.

Dijagrami klasa se koriste pri izradi objektno-orijentiranih modela sustava kako bi prikazale klase sustava u sustavu i veze između tih klasa. Postoje različite vrste veza koje možemo definirati, ali sve se mogu svrstati u tri osnovne kategorije: asocijacija (engl. *association*), generalizacija (engl. *generalization*) i agregacija (engl. *aggregation*).

Asocijacija predstavlja vezu između klasa, te je ona predstavljena običnom linijom iznad koje se nalazi naziv same veze. Asocijacija također sadrži informaciju o kardinalnosti, koja je već

detaljnije objašnjena u poglavlju 3.3.1.1. Brojevi su smješteni na liniji asocijacije kako bi obilježili minimalni i maksimalni broj instanci, koji su uostalom navedeni u Tablici 3.

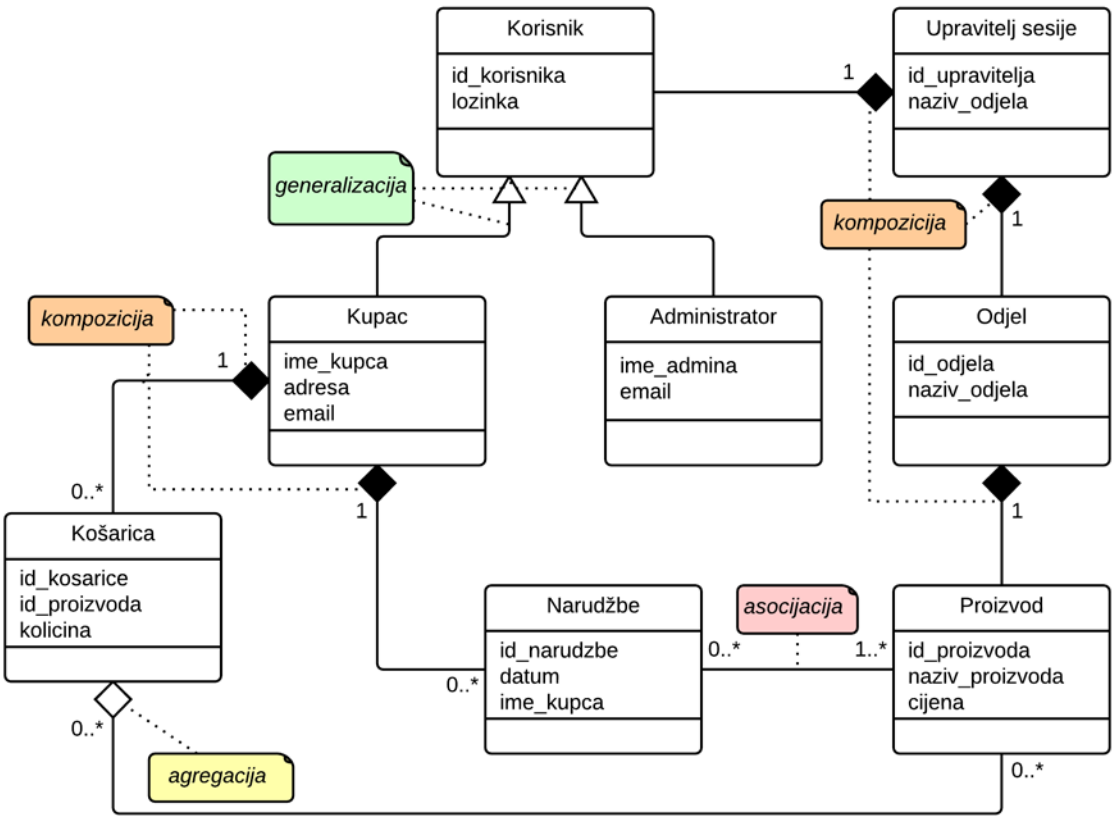
Tablica 3: Broj instanci

OZNAKA	OPIS
1	Točno jedan
0..*	Ni jedan ili više
1..*	Jedan ili više
0..1	Nijedan ili jedan

Izvor: vlastita izrada

Generalizacija nam pokazuje da jedna klasa (podklasa) nasljeđuje od druge (više) klase, što znači da svojstva i operacije više klase također vrijede i za objekte podklase.

Slika 15: Primjer dijagrama klase (za online naručivanje)



Izvor: vlastita izrada

Agregacija je specijalna vrsta asocijacije, te se koristi kada jedna klasa sadrži drugu klasu. Također postoji i kompozicija (engl. *composition*) koja je specijalna vrsta agregacije. Kompozicija označava jaku ovisnost između klase i komponirane (pod)klase, odnosno da komponirana klasa sama za sebe nema smisla. Kod agregacije viša klasa ima veću ulogu od njezine podklase, no te dvije klase nisu ovisne jedna o drugoj.

Na Slici 15 vidimo primjer *class* dijagrama za online naručivanje. Glavni elementi dijagrama klasa su klase, koji pohranjuju i upravljaju informacijama u sustavu. Klasa je prikazana u obliku trodijelnog pravokutnika, gdje se na vrhu nalazi naziv same klase, dok se u sredini smještaju njezini atributi, a u donjem dijelu se mogu po potrebi navesti operacije. Generalizaciju primjenjujemo kada se pri definiranju veze postavlja pitanje je li klasa podvrsta druge klase. Kao što vidimo na Slici, klase „*Kupac*“ i „*Administrator*“ su podvrste klase „*Korisnik*“ odnosno nasljeđuju svojstva navedene klase. Kod agregacije i kompozicije postavlja se pitanje da li klasa sadrži drugu klasu, kao što npr. klasa „*Kupac*“ sadrži klasu „*Košarica*“, a klasa „*Košarica*“ sadrži klasu „*Proizvod*“. Jedina razlika između tih dviju veza je u tome što komponirana klasa ne može postojati bez više klase, u ovom slučaju „*Košarica*“ ne može postojati bez klase „*Kupac*“, no klasa „*Proizvod*“ postoji neovisno o klasi „*Košarica*“ te se shodno tome ta veza prikazuje agregacijom.

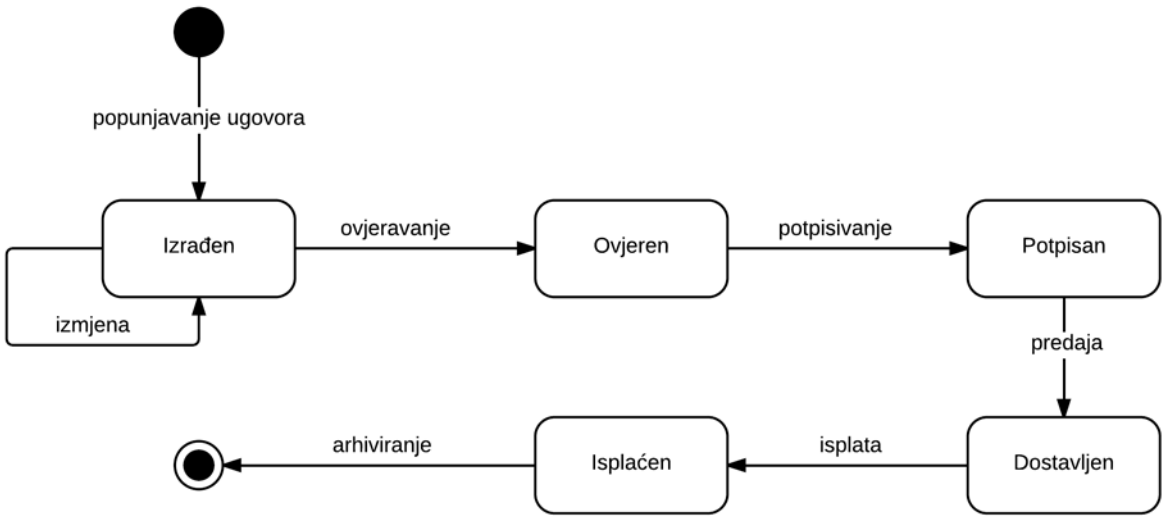
3.4.3. Dijagram stanja (State Machine Diagram)

Neke klase u dijagramima klasa predstavljaju skup objekata koji su vrlo dinamični iz razloga što prolaze kroz razna stanja tijekom svog postojanja. Dijagram stanja je pak dinamični model koji prikazuje različita stanja kroz koji jedan objekt prolazi tijekom svog životnog ciklusa. Dijagrami stanja ne koriste se za sve objekte, već da bismo dodatno definirali složene objekte kako bismo na taj način pojednostavili dizajn algoritama. Ovi dijagrami prikazuju različita stanja objekta, te koji događaji uzrokuju promjenu stanja objekta.

Stanje (engl. *state*) je skup vrijednosti koji služi za opisivanje objekta u određenom trenutku te predstavlja trenutak u životu objekta u kojem zadovoljava određene uvjete, obavlja određene radnje ili čeka da se nešto dogodi. Događaj (engl. *event*) je nešto što se dešava u određenom trenutku i mijenja vrijednost koja opisuje objekt, koji pak zauzvrat mijenja stanje samog objekta. Prijelaz (engl. *transition*) je veza koja prikazuje kretnju objekta iz jednog stanja u

drugo. Objekt se obično kreće iz jednog stanja u drugo na temelju ishoda radnje koji je potaknuo događaj.

Slika 16: Primjer state machine dijagrama (za isplatu naknade u Student servisu)



Izvor: vlastita izrada

Na Slici 16 vidimo primjer *state machine* dijagrama za isplatu naknade u Student servisu. Stanja su prikazana pravokutnicima sa zaobljenim rubovima, u kojima su nazivi samih stanja, poput „*Izrađen*“, „*Ovjeren*“, „*Potpisan*“ itd. Prijelaz je prikazan kao strelica koja izlazi iz jednog stanja, a ulazi u drugo, na kojoj je imenovan naziv događaja. Inicijalno ili početno stanje (engl. *initial state*) označeno je ispunjenim krugom koje predstavlja trenutak u kojem objekt započinje svoje postojanje; a završno stanje (engl. *final state*) ispunjenim krugom unutar drugog kruga koje predstavlja završetak aktivnosti.

3.4.4. Dijagram aktivnosti (Activity Diagram)

Dijagrami aktivnosti pružaju analitičaru sposobnost modeliranja procesa u informacijskom sustavu. Dijagrami aktivnosti mogu se koristiti za modeliranje tijeka rada, pojedinačne slučajeve uporabe ili logiku odlučivanja sadržane unutar pojedinačne metode. Oni također pružaju pristup za modeliranje paralelnih procesa.

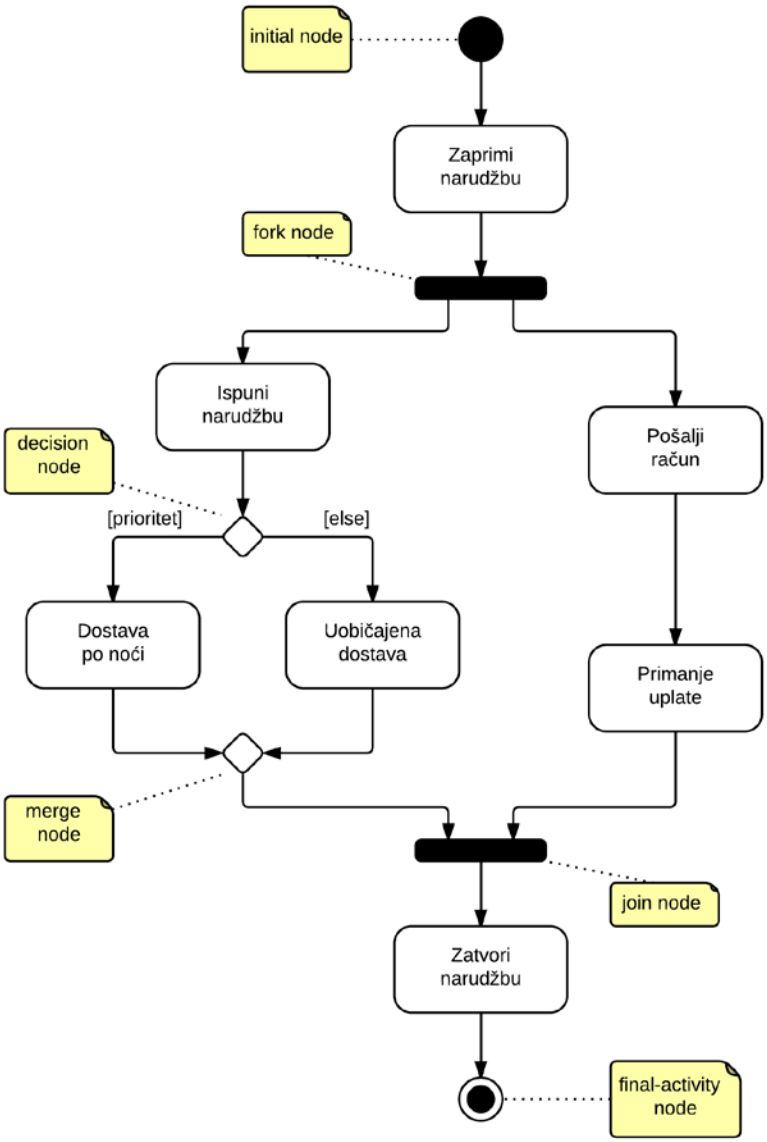
Dijagrami aktivnosti koriste se za modeliranje ponašanja u poslovnom procesu, neovisno o objektima. Oni se zapravo mogu promatrati kao sofisticirani dijagrami toka podataka koji se koriste u sprezi sa strukturnom analizom. Međutim, za razliku od dijagrama toka podataka,

dijagrami aktivnosti uključuju notacije koje se bave modeliranjem paralelnih, istovremenih aktivnosti i složenih procesa odlučivanja.

Dijagrami aktivnosti imaju za cilj pokazati aktivnosti koje čine proces sustava i tok kontrole od jedne aktivnosti do druge. Početak procesa tj. inicijalni čvor (engl. *initial node*) označen je ispunjenim krugom; a završetak procesa odnosno čvor završne aktivnosti (engl. *final-activity node*) ispunjenim krugom unutar drugog kruga. Pravokutnici sa zaobljenim kutovima predstavljaju aktivnosti odnosno određene pod-procese koji se moraju provesti. Strelice predstavljaju tijek rada od jedne aktivnosti do druge. Čvor račvanja (engl. *fork node*) koristi se za označavanje koordinacije aktivnosti. Kada tok iz više od jedne aktivnosti vodi do čvora račvanja, tada se sve te aktivnosti moraju izvršiti prije mogućeg daljnjeg napretka. Kada tok iz čvora račvanja vodi do niza aktivnosti, oni se mogu izvršiti paralelno. S druge strane, čvor spajanja (engl. *join node*) koristi se kako bi vratili skup paralelnih ili istovremenih tokova aktivnosti u zajednički tok. Čvor odluke (engl. *decision node*), predstavljen rombom, koristi se za ispitivanje stanja kako bi se osiguralo da tok protječe samo na jednoj stazi. Nasuprot tome, čvor spajanja (engl. *merge node*), također predstavljen rombom - ali bez uvjeta, koristi se kako bi vratili nazad različite odluke na istu stazu, koje su bile podijeljene pomoću čvora odluke.

Na Slici 17 vidimo primjer *activity* dijagrama za jednostavan proces narudžbe. Inicijalni čvor (engl. *initial node*) nam pokazuje gdje započinje aktivnost procesa. Možemo vidjeti kako čvor račvanja (engl. *fork node*) ima jedan ulazni tok aktivnosti, te više istovremenih izlaznih tokova. Nakon tog čvora vidimo kako se aktivnosti „*Ispuni narudžbu*“, „*Pošalji račun*“ i njihove slijedeće aktivnosti odvijaju paralelno i u isto vrijeme. Čvor odluke (engl. *decision node*) ima uvjetno izvođenje gdje odluka može biti *true* ili *false* (prikazano uglatim zagradama). Aktivnost u tom slučaju može odabrati samo jedan izlazni tok. Čvor spajanja (engl. *merge node*) označava kraj uvjetnog izvođenja započeto odlukom, te se tokovi spajaju u jedinstveni tok. Nakon što svi ulazni tokovi stignu do čvora spajanja (engl. *join node*), izlazni tok može nastaviti svoj put iz spomenutog čvora prema čvoru završne aktivnosti (engl. *final-activity node*) odnosno završetku aktivnosti procesa. Možemo zaključiti kako je ovaj dijagram vrlo sličan DTP-u, ali je glavna razlika ta da notacija podržava paralelno odvijanje aktivnosti.

Slika 17: Primjer activity dijagrama (za proces narudžbe)



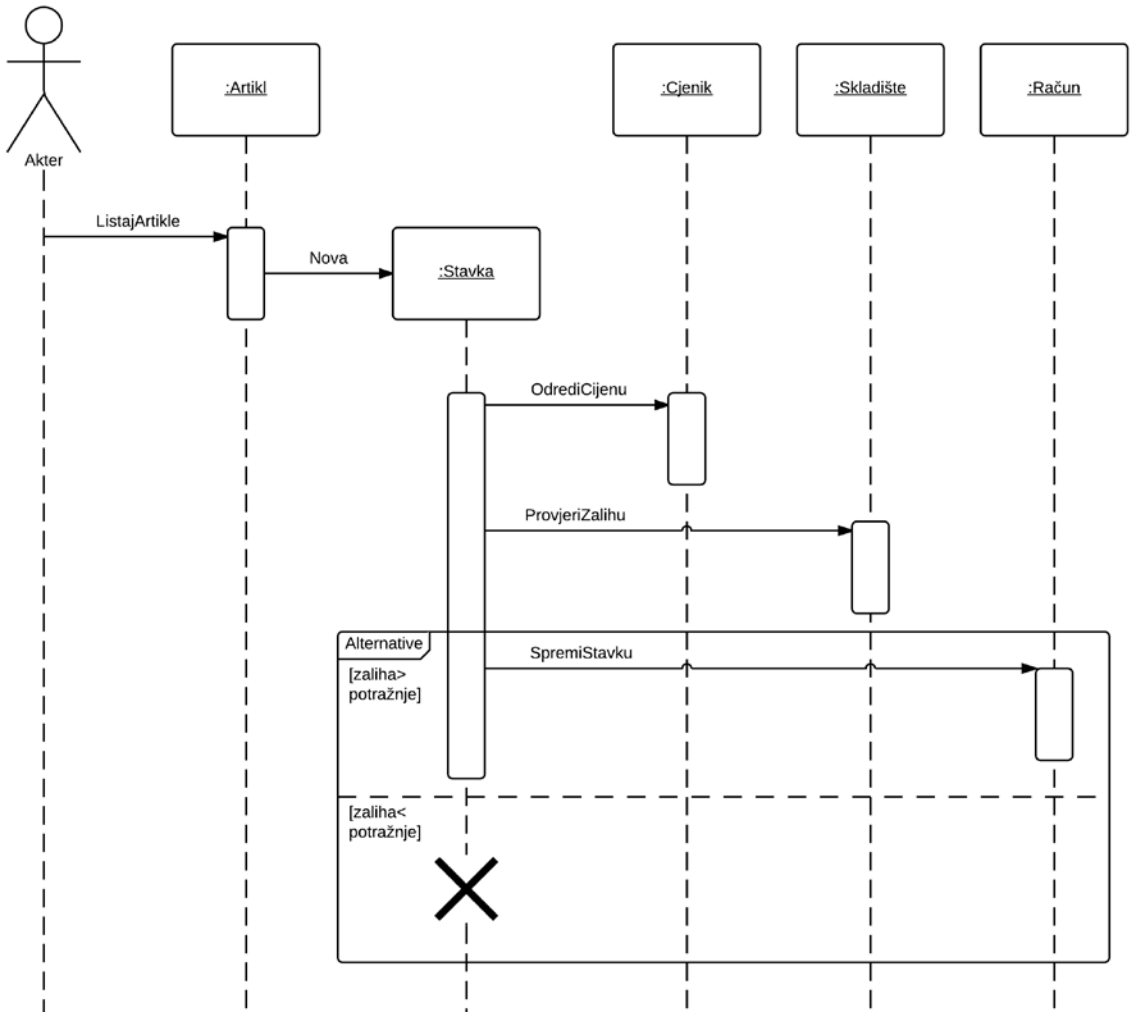
Izvor: vlastita izrada

3.4.5. Dijagram slijeda (Sequence Diagram)

Dijagrami slijeda omogućuju analitičaru prikazati dinamičke interakcije među objektima u informacijskom sustavu. Dijagrami slijeda su najčešći oblik dijagrama interakcije koji se koriste u objektno-orijentiranom modeliranju. Oni naglašavaju naručivanje aktivnosti na vremenskoj bazi koje se odvija sa skupom objekata koji međusobno surađuju. Oni su vrlo korisni u pomaganju analitičaru kako bi razumio specifikacije u realnom vremenu i složenije slučajeve uporabe. Dok *use case* dijagram samo imenuje interakcije (i ne prikazuje kako se zapravo odvijaju), *sequence* dijagram nam služi za detaljniji opis pojedinog slučaja uporabe.

Ovi dijagrami mogu se koristiti za opisivanje kako logičkih, tako i fizičkih interakcija između objekata. Kao takvi, oni su korisni i u dizajnu i u analizi.

Slika 18: Primjer sequence dijagrama (za upit dodavanja artikla u izradu računa)



Izvor: vlastita izrada

Na Slici 18 imamo primjer *sequence* dijagrama za upit dodavanja artikla u izradu računa. Objekti i akteri uključeni u dijagram pojavljuju se uz sam vrh dijagrama i nanizani su u vodoravnom smjeru, te iz njih okomito izlazi isprekidana linija koja označava tijek vremena. Interakcije između tih objekata prikazuju se strelicama. Pravokutnici koji se nalaze na vertikalnim linijama predstavljaju period aktivnosti dotičnih objekata ili aktera te označavaju kada objekti šalju ili primaju poruke. Slijed interakcija se iščitava od gore prema dolje. Vodoravne strelice označavaju vremenski redoslijed poruka koje akteri ili objekti međusobno razmjenjuju. Zapravo je riječ o pozivima operacija objekata od kojih svaka ima svoj naziv koji je smješten iznad strelice. Okvir „*Alternative*“ tj. uvjetno izvršavanje, određuje koja će se

poruka izvršiti ovisno o ispunjenju navedenog uvjeta. Ukoliko količina zaliha zadovoljava potražnju, izvršit će se „*SpremiStavku*“ u objektu „*:Račun*“, no ako nema zaliha stavka se briše. Brisanje stavke je prikazano tako što se stavlja „*X*“ na kraju vremenskog tijeka objekta. Objekt se na taj način deinstancira tj. briše iz memorije.

3.5. Zajednički razvoj aplikacije i Brzi razvoj aplikacije

Tradicionalni način modeliranja za razvoj informacijskih sustava bili su jednostavni IT odjeli koji bi koristili strukturirane analize, te bi se konzultirali sa korisnicima samo kad bi im bilo potrebno odobrenje ili nekakav input. Tijekom vremena, mnoge tvrtke su došle do zaključka kako timovi sastavljeni od IT osoblja, korisnika i menadžera, puno brže dovršavaju svoj posao te daju puno bolje rezultate. Ubrzo su dvije metodologije postale popularne: Zajednički razvoj aplikacije (JAD) i Brzi razvoj aplikacije (RAD).

Obje metodologije koriste timove sastavljenih od korisnika, *developera* i menadžera. Razlika je u tome što se JAD fokusira na utvrđivanje činjenica koje pronalaze pomoću tima, a predstavlja samo jednu fazu u razvojnom procesu, dok RAD možemo promatrati kao kompresiranu verziju cjelokupnog procesa.

3.5.1. JAD

Zajednički razvoj aplikacije (engl. *Joint Application Development - JAD*) je popularna tehnika bazirana na pronalaženju činjenica koja uključuje korisnike kao aktivne sudionike u razvojnom procesu. Nekada su, tijekom razvoja informacijskog sustava, *developeri* skupljali informacije od korisnika, utvrdili sistemske zahtjeve i na temelju toga razvili novi sustav. U današnje vrijeme, korisnici imaju puno veću ulogu u razvoju sustava. *Developeri* sada uviđaju važnost sudjelovanja korisnika u razvojnom procesu odnosno da će sustavi biti puno uspješniji ukoliko su usmjereni na korisnika. JAD tim se obično sastaje u periodu od nekoliko dana ili tjedana u sobi za sastanke ili nekoj drugoj lokaciji. Cilj je analizirati postojeći sustav, utvrditi očekivanja korisnika i inpute, te kreirati dokument sa zahtjevima korisnika za novi sustav.

JAD tim se sastoji od [10]:

- Voditelja projekta: koji vodi samu JAD sjednicu.

- Top menadžmenta: koji pruža ovlaštenje i podršku za projekt.
- Menadžera: koji pruža podršku na razini odjela.
- Korisnika: koji pruža inpute za željene izmjene, zahtjeve, probleme oko sučelja.
- Sistem analitičara: koji pruža tehničku pomoć, sudjeluje u raspravama, postavlja pitanja, zapisuje bilješke i pruža podršku timu.
- Zapisničara: koji radi sa sistem analitičarem i dokumentira rezultate sjednice.

JAD omogućuje svojim korisnicima da učinkovito sudjeluju u modeliranju procesa. Ukoliko se koristi na pravilan način, JAD može puno točnije utvrditi sistemske zahtjeve i pridonijeti puno boljem razumijevanju zajedničkih ciljeva. Iako JAD ima svoje pozitivne strane, treba napomenuti kako je JAD, za razliku od tradicionalnih metoda, skuplji te može biti nezgrapan ukoliko je JAD grupa prevelika u odnosu na veličinu samog projekta.

3.5.2. RAD

Brzi razvoj aplikacije (engl. *Rapid Application Development - RAD*) je tehnika koja je također bazirana na timu, ali koja ubrzava sam razvoj informacijskih sustava te stvara funkcionalan informacijski sustav. Dok je kod JAD-a krajnji proizvod bio model zahtjeva, krajnji proizvod RAD-a je novi informacijski sustav. Mnogi koriste RAD kako bi smanjili troškove i uštedili vrijeme utrošeno na razvoj, te kako bi povećali vjerojatnost uspjeha. RAD se dosta oslanja na prototipiranje i uključenost korisnika. Proces omogućuje korisnicima da dobiju uvid u razvojni model u ranoj fazi razvoja kako bi utvrdili da li ispunjava njihove zahtjeve. Prototip se zatim, na temelju njihove povratne informacije, mijenja i taj se interaktivni proces ponavlja sve dok sustav ne bude u potpunosti razvijen. Projektni tim koristi tzv. CASE alate kako bi razvili prototipe i pripadajuću dokumentaciju.

Softversko inženjerstvo podržano računalom (engl. *Computer Aided Software Engineering - CASE*) stavlja informatičarima na raspolaganje računalne alate za razvoj informacijskog sustava. Alat CASE je programski sustav koji omogućuje da se na računalu oblikuju i pohrane različiti modeli ili opisi sustava nastali u pojedinim fazama razvoja (poput modela entiteti-veze i relacijskog modela podataka). Alat CASE može pomoću ugrađenih pravila generirati dijelove informacijskog sustava. CASE alati su zapravo softverski paketi koji daju automatiziranu podršku za pojedine aktivnosti unutar softverskog procesa.

RAD model se sastoji od četiri faza, kao što možemo vidjeti na Slici 19.

Slika 19: Faze RAD modela

Izvor: vlastita izrada prema Shelly, G.B. i Rosenblatt, H. J. (2012): System Analysis and Design (9th Edition), Cengage Learning, Boston MA, USA

Pri **planiranju zahtjeva** korisnici, menadžeri i članovi IT osoblja razgovaraju o poslovnim potrebama, opsegu projekta, ograničenjima i sistemskim zahtjevima. Ova faza će završiti kada tim bude suglasan o ključnim pitanjima i dobije odobrenje za daljnji rad od strane menadžmenta.

U fazi **dizajna korisnika**, korisnik je u interakciji sa sistem analitičarem te se razvijaju modeli i prototipovi koji predstavljaju sve procese sustava, outpute i inpute. RAD timovi obično koriste kombinaciju JAD tehnike i CASE alata kako bi pretvorili potrebe korisnika u razvojne modele.

U fazi **izgradnje** razvija se sam informacijski sustav, a korisnici i dalje sudjeluju te mogu predlagati izmjene i poboljšanja u sustavu. Treba primijetiti kako su druga i treća faza u kontinuiranoj interakciji.

Faza **rezanja** je završna faza koja bi u tradicionalnim metodama uključivala implementaciju sustava, testiranje i obuku korisnika. Kod RAD-a su sve te aktivnosti kompresirane u jednu fazu. Kao rezultat toga, izgrađen je novi sustav koji se odmah isporučuje i puno brže stavlja u rad.

Glavni cilj RAD-a je smanjiti troškove i vrijeme razvoja uključivanjem korisnika u svaku fazu razvoja sustava. Budući da je to kontinuirani proces, razvojnom timu je omogućeno da brzo

izvrše potrebne izmjene. Prednost ovakvog način razvoja informacijskog sustava je, kao što smo naveli, brzina i smanjivanje troškova. No, RAD također donosi i određene rizike, budući da sustav može raditi dobro u kratkom roku, ali možda neće biti ispunjeni dugoročni ciljevi za sustav. Također, samo ubrzavanje procesa može rezultirati manjom kvalitetom, nedosljednošću i neispunjenjem određenih standarda. RAD prije svega stavlja samu mehaniku sustava u prvi plan, a ne strateške poslovne potrebe tvrtke za koje se sustav razvija.

4. SUVREMENI TRENDOVI RAZVOJA INFORMACIJSKIH SUSTAVA

4.1. Agilne metode razvoja

U 90-tim godinama prošlog stoljeća, agilni razvoj je postao alternativa postojećim klasičnim metodama koje su bile percipirane kao suviše „teške“ u procesima. Te klasične metode naglašavaju potrebu planiranja projekta unaprijed, zapisivanja zahtjeva, dokumentiranja dizajna koji zadovoljavaju zahtjeve, zapisivanja programskih kodova baziranih na tim dizajnima, i testiranja rezultata. Upravo zbog navedenih koraka, mnogi projekti nailaze na velike probleme. Najveći razlog je taj da *stakeholderi* (zainteresirane strane) obično ne znaju na samom početku projekta što žele u potpunosti. Drugim riječima, glavni cilj agilnog razvoja je bio ubrzati razvoj softvera i izbjeći nepotrebnu administraciju.

Prema istraživanjima koje navodi Manger [20], agilne metode imaju svoje prednosti kad je riječ o malim i kratkoročnim projektima u brzo promjenjivom poslovnom okruženju. No, još se nije dokazalo da se agilne metode mogu uspješno primijeniti na velike projekte. Zato se danas veliki softverski sustavi i dalje razvijaju klasičnim i to objektno-orijentiranim metodama. Agilne metode uglavnom se primjenjuju u razvoju web i mobilnih aplikacija (o kojima će više biti riječi u poglavlju 4.3.).

Tijekom 90-ih godina, razne iterativne metodologije su počele dobivati na popularnosti. Te metodologije su imale razne kombinacije starih i novih ideja, ali su sve dijelile slijedeće karakteristike:

- Bliska suradnja između programera i poslovnih stručnjaka,
- Komunikacija licem u lice,
- Česta isporuka funkcionalnog softvera,
- Samoorganizirani timovi.

2001. godine, grupa stručnjaka, koja je bila razočarana sa uobičajenim praksama softverskog inženjerstva, okupila se kako bi razmotrila načine unaprjeđenja razvoja softvera. Njihov cilj je bio proizvesti skup vrijednosti i načela kako bi ubrzali sam razvoj i učinkovito se prilagodili promjenama. Grupa je samu sebe zvala *Agile Alliance* (Agilni savez) kako bi prikazala svoju suštinu koja je imala za cilj razviti metodologiju koja bi bila efikasna i prilagodljiva. Rezultat njihovog rada je tzv. Manifest za agilni razvoj softvera (engl. *Manifesto for Agile Software*

Develpment) poznat kao i Agilni manifest (engl. *Agile Manifesto*), koji sadrži vrijednosti i načela koje su definirali.

Četiri točke Agilnog manifesta koje formiraju temelj agilnog razvoja govore da autori više cijene:

1. **Ljude i njihove međusobne odnose nego procese i alate**
   Agilna metoda naglašava važnost vještina kvalificiranih pojedinaca i poboljšanja vještina koje proizlaze iz njihovih interakcija. Iako su proces i alati važni, trebalo bi dopustiti vještim pojedincima prilagođavanje procesa i izmjenu alata kako bi obavili svoj posao što efikasnije. Kreativnost se ističe kao glavno sredstvo za rješavanje problema. To je u suprotnosti sa strožim klasičnim procesima koji zahtijevaju od članova tima da se prilagođavaju predodređenim pravilima.
2. **Upotrebljiv softver nego iscrpnu dokumentaciju**
   Upotrebljiv softver smatra se najboljim pokazateljem napretka projekta odnosno jesu li ciljevi ispunjeni. Timovi bi teoretski mogli sastaviti veliku količinu dokumentacije i navodno poštivati zakazani rok, no to su zapravo samo puka obećanja onoga što žele proizvesti. Stoga, agilna metoda naglašava važnost pisanja programskog koda na samim počecima, nakon kojeg se funkcionalnost softvera povećava dodavanjem malih inkrementa kako bi softver radio poput operativnog sustava. Na taj način i *developeri* i *stakeholderi* znaju kako sustav funkcionira u realnom vremenu.
3. **Suradnju s korisnicima nego pregovaranje oko ugovora**
   Budući da su korisnici najbitniji dio projekta (jer su oni ti koji će naposljetku kupiti i koristiti taj proizvod), bitno je surađivati što više s njima. Velik broj programera je isključen iz te suradnje zbog organizacije i posrednika. Svi *stakeholderi* bi stoga, trebali raditi zajedno i biti dio istog tima. Na taj način je omogućeno obavljati brze izmjene na softveru i proizvesti upravo ono što je i zahtijevano.
4. **Reagiranje na promjenu nego ustrajanje na planu**
   Ustrajanje na planu ograničava članove tima na način da razmišljaju unutar zadanog okvira. Budući da su izmjene na softveru neizbježne, pobornici agilne metode smatraju da moraju biti spremni na promjene. Kako se dešavaju izmjene u planovima, timovi ne bi smjeli biti fokusirani na zastarjeli plan već se moraju nositi s promjenama tako što će prilagoditi plan.

Autori Agilnog manifesta su, uz prije spomenute vrijednosti, definirali i načela koja podržavaju manifest, a prikazani su u slijedećoj Tablici 4:

Tablica 4: Načela agilnog razvoja

1	Najvažnije nam je zadovoljstvo naručitelja koje postižemo ranom i neprekinutom isporukom softvera koji nosi vrijednost.	**7**	Upotrebljiv softver je osnovno mjerilo napretka.
2	Spremno prihvaćamo promjene zahtjeva, čak i u kasnoj fazi razvoja. Agilni procesi upređu promjene da naručitelju stvore kompetitivnu prednost.	**8**	Agilni procesi potiču i podržavaju održivi razvoj. Pokrovitelji, razvojni inženjeri i korisnici trebali bi moći neograničeno dugo zadržati jednak tempo rada.
3	Često isporučujemo upotrebljiv softver, u razmacima od nekoliko tjedana do nekoliko mjeseci, nastojeći da razmak bude čim kraći.	**9**	Neprekinuti naglasak na tehničkoj izvrsnosti i dobar dizajn pospješuju agilnost.
4	Poslovni ljudi i razvojni inženjeri moraju svakodnevno zajedno raditi, tijekom cjelokupnog trajanja projekta.	**10**	Jednostavnost - vještina povećanja količine posla kojeg ne treba raditi - je od suštinske važnosti.
5	Projekte ostvarujemo oslanjajući se na motivirane pojedince. Pružamo im okruženje i podršku koja im je potrebna, i prepuštamo im posao s povjerenjem.	**11**	Najbolje arhitekture, projektne zahtjeve i dizajn, stvaraju samo-organizirajući timovi.
6	Razgovor uživo je najučinkovitiji način prijenosa informacija razvojnom timu i unutar tima.	**12**	Tim u redovitim razmacima razmatra načine da postane učinkovitiji, zatim usklađuje i prilagođava svoje ponašanje.

Izvor: vlastita izrada prema http://www.agilemanifesto.org/iso/hr/principles.html, [03. travnja 2015.]

Agilni proces je vrlo iterativan i inkrementalan. On ima slijedeće karakteristike:

- Mali, usko vezani timovi ljudi.
- Redoviti i disciplinirani sastanci o zahtjevima kupaca odnosno korisnika.
- Pristup koji u središnjicu stavlja programski kod, dok se dokumentacija sastavlja po potrebi.
- Predstavnici korisnika usko surađuju sa timom.
- Upotreba korisničkih priča (engl. *user stories*) kao temelj za zahtjeve korisnika.
- **Refaktoriranje (engl. *Refactoring*):** proces mijenjanja oblika tj. reorganizacija programskog koda, ne mijenjajući pritom svoju funkcionalnost. Programer zapravo pojednostavljuje i premještava dijelove programskog koda kada uoči da se struktura programa narušila. Upravo je *refactoring* velikim dijelom zaslužan za postojanje agilne metode.
- **Programiranje u paru:** Oblik neprekidne inspekcije gdje si dva člana tima međusobno ispravljaju greške. Na taj način zadatak obavljaju dvojica programera koji pišu isti programski kod te lakše dolaze do ispravnog rješenja. Parovi nisu fiksirani nego se mijenjaju ovisno o projektu.

- Kontinuirani jedinični testovi (engl. *unit testing*) i testovi prihvatljivosti (engl. *acceptance tests*) kao sredstva za ispunjenje korisnikovih očekivanja.

Prednosti agilne metode:

- Projekt uvijek ima demonstrirajuće rezultate - završna verzija svake iteracije je upotrebljivi softver.
- *Developeri* su više motivirani - više im odgovara proizvesti upotrebljiva rješenja, a ne vole sastavljati dokumentacije.
- Korisnici (kupci) u mogućnosti su pružiti bolje zahtjeve budući da mogu prisustvovati samom razvoju proizvoda.

Nedostaci:

- Problematične su za veće aplikacije. Agilne metode se više koriste za manje projekte. Postoje rasprave jesu li one korisne za veće projekte.
- Dokumentacija je upitna. Budući da je dokumentacija u drugom planu, postavlja se pitanje da li će se potrebna dokumentacija uopće sastaviti. To predstavlja velik problem, budući da je dokumentacija sastavni dio onoga što *developeri* proizvode. Naime, ukoliko se originalnim razvojem softvera bavio jedan tim (koji je koristio agilni razvoj), a održavanje je prepušteno nekom drugom timu (koji se bavi klasičnim metodama razvoja), tada može doći do velikih komplikacija.

Iako ne postoji potpuna suglasnost o tome, na što se zapravo pojam „agilan“ odnosi, on je ostvario veliki interes kod programera, a u posljednje vrijeme i u akademskoj zajednici. Uvođenje ekstremnog programiranja (koje je detaljnije objašnjeno u poglavlju 4.1.2.) smatra se polazištem za različite pristupe agilnog razvoja softvera. Postoje i brojne druge metode koje se razvijaju i nanovo otkrivaju te koje se čine da pripadaju istoj obitelji metodologija. Neke od tih metodologija su *Crystal Method, Feature Driven Development, Adaptive Software Development* itd., koje ćemo detaljnije objasniti u narednim poglavljima.

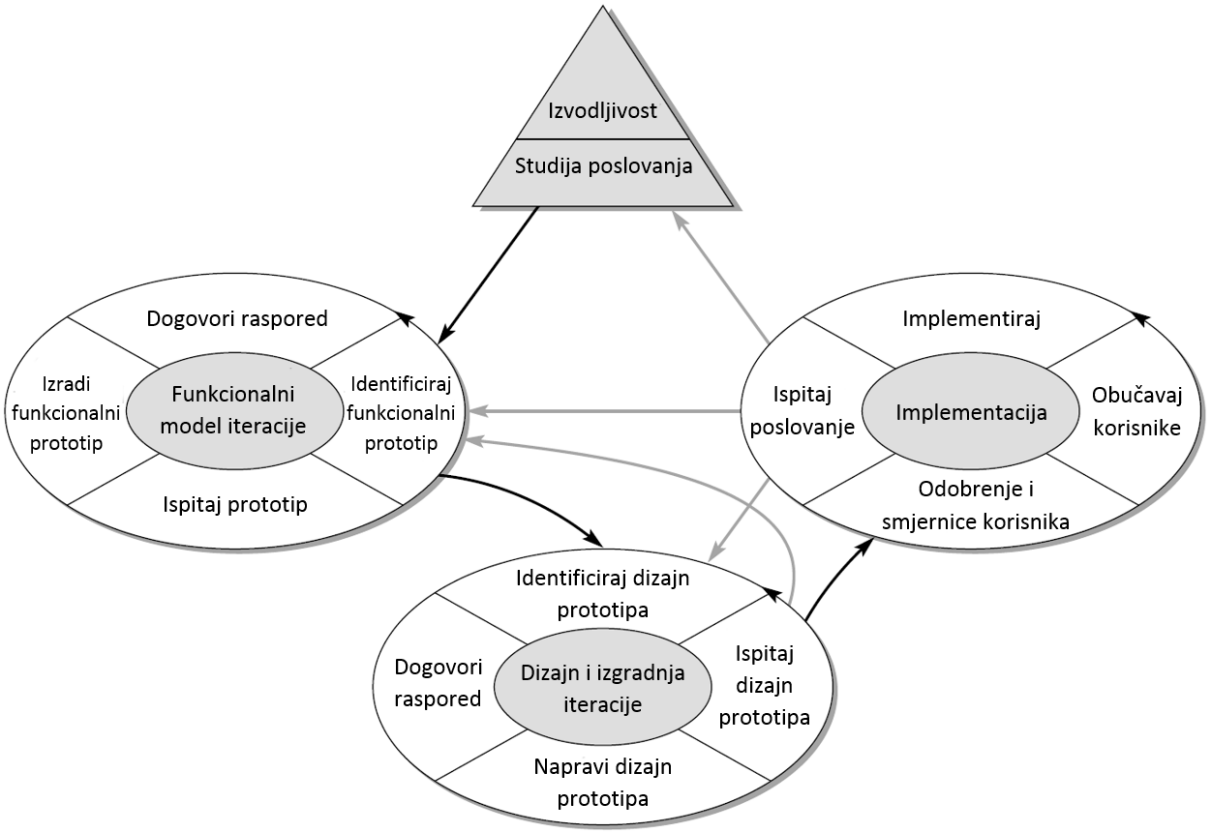
Unatoč velikom interesu, nije postignut jasni sporazum kako razlikovati agilni razvoj od tradicionalnih pristupa. Granice, ukoliko one uopće postoje, nisu jasno definirane.

4.1.1. Dynamic Systems Development Method (DSDM)

Metoda razvoja dinamičkih sustava (engl. *Dynamic Systems Development Method - DSDM*) služi za upravljanje robusnim agilnim projektima i kao razvojni okvir (engl. *framework*) koji donosi pravo rješenje u pravo vrijeme. DSDM je neprofitan i ne-vlasnički razvojni okvir kojeg održava tzv. DSDM konzorcij (engl. *DSDM Consortium*). Još od svoga osnutka 1994. godine, DSDM je postupno postao glavni razvojni okvir za *Rapid Application Development (RAD)* u Velikoj Britaniji. RAD je pokrenut s namjerom da se razlikuje od modela vodopada za razvoj aplikacija. No, RAD se razvijao na vrlo nestrukturiran način, nije postojao dogovor oko definicije njegovog procesa, a mnogi su poduzetnici i konzultanti imali vlastita tumačenja i pristupe. Na tržištu se pojavljivao sve veći broj alata za RAD, te se iz potrebe da se utvrde standardni razvojni okviri za RAD osnovao spomenuti konzorcij.

Filozofija DSDM-a se temelji na tome da svaki projekt mora biti usklađen sa jasno definiranim strateškim ciljevima i usredotočen na rane isporuke sa realnim koristima u poslovanju.

Slika 20: Faze DSDM procesa



Izvor: prilagođeno prema Abrahamsson, P., Salo, O. i Ronkainen, J. (2002): Agile software development methods - Review and analysis, VTT Electronics, University of Oulu, Finland

Temeljna ideja DSDM-a je da umjesto da se prvo ispravljaju količine funkcionalnosti u proizvodu, a tek onda vrijeme i resursi za postizanje tih funkcionalnosti; kao prioritet su postavljeni vrijeme i resursi, a tek onda prilagođavanje funkcionalnosti.

Kao što vidimo na Slici 20, DSDM se sastoji od pet faza: studija izvodljivosti, studija poslovanja, funkcionalni model iteracije, dizajn i izgradnja iteracije, te implementacija. Prve dvije faze su sekvencijalne i izvršavaju se samo jednom, dok su ostale tri faze iterativne i inkrementalne. DSDM pristupa iteracijama kao „vremenskim kutijama“ (engl. *timebox*). Vremenska kutija traje unaprijed određeni vremenski period, a iteracija mora završiti unutar tog razdoblja koja može trajati od nekoliko dana do nekoliko tjedana.

U **studiji izvodljivosti** (engl. *feasibility study*) procjenjuje se prikladnost DSDM-a za određeni projekt. Ovisno o tipu projekta i ponajviše o organizacijskim i ljudskim problemima, donosi se odluka da li treba koristiti DSDM. Studija izvodljivosti se također bavi tehničkim mogućnostima obavljanja samog projekta i njegovim rizicima.

Studija poslovanja je faza u kojoj se analiziraju tehnologija i osnovne karakteristike poslovanja. Preporučuje se organizirati radionice gdje bi se okupio dovoljan broj stručnjaka (od strane kupaca) kako bi mogli uzeti u obzir sve relevantne aspekte sustava i kako bi se mogli dogovoriti o razvojnim prioritetima.

Funkcionalni model iteracije je prva iterativna i inkrementalna faza. U svakoj su iteraciji planirani sadržaji i pristupi iteracije, a rezultati su analizirani radi daljnjih iteracija. U ovoj fazi se obavljaju analiza i programiranje, izrađuju se prototipovi, a stečena iskustva služe za poboljšavanje modela analize.

Dizajn i izgradnja iteracije je faza u kojoj se sustav uglavnom izrađuje. Output je testirani sustav koji ispunjava barem minimum dogovorenog skupa zahtjeva. Dizajn i funkcionalni prototipovi su ispitani od strane korisnika, a daljnji razvoj se temelji na njihovim komentarima.

U završnoj fazi - fazi **implementacije**, sustav se prenosi iz razvojnog okruženja u stvarno proizvodno okruženje. Korisnici se obučavaju, te im se predaje sustav. Ukoliko sustav ispunjava sve uvjete, daljnji rad je nepotreban. No, ukoliko se desi da nije ispunjena značajna količina uvjeta, proces se može pokrenuti ponovno, od početka do kraja. Ukoliko je izostavljena manje kritična funkcionalnost, proces se može pokrenuti od funkcionalnog modela iteracije. Ukoliko se neki tehnički problemi ne mogu riješiti radi vremenskih ograničenja, mogu se izvršiti ponovnom iteracijom, počevši od dizajna i izgradnje iteracije.

Treba napomenuti kako DSDM timovi mogu imati između dva i šest članova te može biti više timova u jednom projektu. Potrebno je minimalno dvije osobe u timu kako bi svaki tim imao barem jednog korisnika i jednog *developera*.

4.1.2. Ekstremno programiranje (XP)

1996. godine Kent Beck sa svojim kolegama započeo je projekt koristeći pristup razvoju softvera koji se činio puno jednostavnijim i ujedno efikasnijim. Metodologiju koju je razvio i počeo primjenjivati postala je poznata kao Ekstremno programiranje (engl. *Extreme programming - XP*).

Beck navodi četiri vrijednosti koje vode ekstremno programiranje [15]:

- Komunikacija,
- Jednostavnost,
- Povratna informacija (engl. *feedback*),
- Hrabrost.

XP programeri neprekidno komuniciraju sa svojim korisnicima i kolegama programerima. Njihovi dizajni su jednostavni i čisti. Testirajući svoj softver dobivaju povratnu informaciju, počevši od prvog dana. Ispostavljaju dijelove sustava korisnicima, ako su u mogućnosti što prije, te implementiraju preporučene promjene. Na temelju navedenog, XP programeri su sposobni hrabro reagirati na promjene u zahtjevima i tehnologiji. XP je stvoren kako bi se učinkovitije rješavali projekti čiji se zahtjevi mijenjaju. XP i očekuje da se zahtjevi modificiraju i nadograđuju.

XP projekti su podijeljeni na iteracije koje traju jedan do tri tjedna. Svaka iteracija proizvodi softver koji je potpuno testiran. Na početku svake iteracije održavaju se sastanci kako bi se odredio sadržaj svake pojedine iteracije.

XP prepoznaje problematiku u sve češćem raspadu odnosa između *developera* i korisnika, gdje svaka strana ima svoju ideju kako bi projekt trebao funkcionirati. Potrebno je da *developeri* uzmu u obzir kako je programski kod zajedničko vlasništvo, odnosno da ne pripada samo njima.

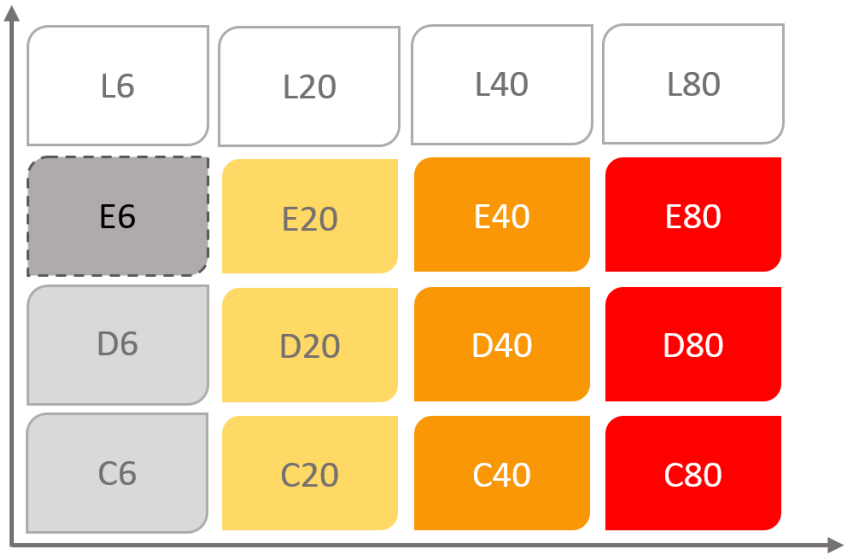
Prema Braudeu i Bernsteinu [2] XP koristi 12 praksi koje govore kako bi programeri trebali rješavati svoju svakodnevnu zadaću:

1. **Proces planiranja**: Zahtjeve, obično u formi korisničkih priča, definiraju korisnici i prioritiziraju se na temelju procjene troškova XP tima.
2. **Male distribucije**: Jednostavan sustav je izgrađen i pušten rano u proizvodnju koji uključuje minimalni skup korisnih značajka. Sustav se često i inkrementalno ažurira tijekom cijelog razvojnog procesa.
3. **Razvoj baziran na testovima**: Jedinični testovi su pisani za testiranje funkcionalnosti, prije nego što je sam kod za implementiranje funkcionalnosti zapravo napisan.
4. **Refactoring**
5. **Jednostavni dizajni**: Dizajni su kreirani kako bi riješili sadašnje (poznate), a ne buduće zahtjeve.
6. **Programiranje u paru**
7. **Kolektivno vlasništvo programskog koda**: Programski kod sustava je u vlasništvu cjelokupnog tima. Programeri mogu izmijeniti bilo koji dio sustava potreban za dovršenje određenih svojstva.
8. **Standardno programiranje**: Programeri moraju slijediti zajednički standard za programiranje. To će osigurati cijelom timu lakše razumijevanje programskog koda, neovisno o tome tko ga je napisao.
9. **Neprekidna integracija**: Programski kod se provjerava i integrira odmah nakon što se dovrši i testira. To se može desiti i po nekoliko puta na dan. Taj način osigurava proizvodu da bude što kvalitetniji.
10. **Raspoloživost korisnika**: Predstavnik korisnika tj. kupca je uvijek dostupan kako bi odredili zahtjeve, postavili prioritete i odgovorili na pitanja koja bi programeri mogli imati. Radi toga se komunikacija poboljšava i potrebna je sve manja količina dokumentacije.
11. **Održivi tempo**: XP timovi su produktivniji i rjeđe griješe ukoliko nisu umorni. Zbog toga im je cilj ne raditi prekovremeno i da se osjećaju svježi, zdravi i efikasni.
12. **Metafora**: XP timovi dijele zajedničku viziju sustava na način da definiraju i koriste zajednički sustav koji opisuje artefakte u projektu.

4.1.3. Crystal

Kristalna obitelj metodologija (engl. *Crystal Method*) razvijena je od strane Alistaira Cockburna. Crystal metode su orijentirane prema ljudima, smanjuju administraciju, te se prilagođavaju problemima i okolini. Cockburn navodi [4] kako je nazvao metodu Crystal jer je u svojoj knjizi „Crystal Clear" stvorio protagonista pod imenom Crystal kako bi personificirao metodologiju i raspravljao o dizajnu, te kako bi pružio metaforu za projektnu mrežu odnosnu rešetku koja je prikazana na Slici 21. Metode su dobile naziv prema geološkim kristalima: čisti (engl. clear), žuti, narančasti, crveni, itd., a najpoznatije metode su: *Crystal Clear*, *Crystal Yellow*, *Crystal Orange* i *Crystal Orange Web*.

Slika 21: Crystal metoda



Izvor: vlastita izrada prema Cockburn, A. (2000): Agile Software Development, Cockburn Highsmith Series Editors

Svakoj Crystal metodi dodjeljujemo boju, počevši od (kristalno) čiste za manje timove koja napreduje prema narančastoj, crvenoj itd., kako se broj ljudi u timovima povećava. Krećući se s lijeva na desno znači da koordiniramo više ljudi, što znači da nam je teže upravljati procesima, a samim time potrebna nam je veća količina dokumentacije i alata. Tako je „kristalno čista" (engl. *Crystal Clear*) metoda usmjerena za manje timove od oko 6 ljudi, žuta (engl. *Crystal Yellow*) za timove od 10 do 20 ljudi, narančasta (engl. *Crystal Orange*) za timove od 20 do 40 ljudi itd. Tako se primjerice *Crystal Orange* metoda koristi za D40 projekte, gdje imamo tim od 20-40 programera koji rade na izgradnji projekta, čiji bi nedostaci mogli uzrokovati gubitak diskrecijskog novca. Vertikalna koordinata definira kritičnost projekta gdje **L** označava „gubitak života" (engl. *loss of **L**ife*), **E** označava „gubitak značajnog novca" (engl. *loss of*

***E**ssential money*), **D** je „gubitak diskrecijskog novca“ (engl. *loss of **D**iscretionary money*), a **C** je „gubitak udobnosti“ (engl. *loss of **C**omfort*). Krećući se prema gore zapravo dolazi do veće potencijalne štete iz sustava, odnosno prema kristalnoj metafori kretnja prema gore uzrokuje povećanje „čvrstoće“ (kristala). S obzirom na veličinu tima i kritičnost projekta, odabiremo odgovarajuću Crystal metodu. Svaka metoda ima skup preporučenih praksi, tehnika i notacija. Treba napomenuti da red koji označava gubitak života nije osjenčan nikakvom bojom iz razloga što Cockburn nije imao iskustva sa projektima sa „životno-kritičnim“ sustavom u vrijeme kad je izradio dijagram, pa stoga ne postoji dovoljno informacija kako se ova metoda ponaša u tim kritičnim situacijama. Crystal Clear također ne podržava eksplicitno ni **E6** kvadrat, no tim bi mogao po potrebi proširiti Crystal Clear kako bi se prilagodio takvoj situaciji.

Svaka Crystal metoda naglašava važnost ljudi u razvijanju softvera. Crystal se usredotočuje na ljude, njihovu interakciju, talente, komunikaciju i vještine koje posjeduju. Proces je pri tome važan, ali se stavlja u drugi plan. Crystal projekti imaju sedam ključnih principa: česta isporuka, kontinuirana povratna informacija, bliska komunikacija, osobna sigurnost, fokus, jednostavan pristup stručnim korisnicima, te tehnička okolina sa automatskim testiranjem i čestom integracijom. Poput drugih agilnih metodologija, Crystal promiče rane i česte isporuke funkcionalnog softvera, visoku uključenost korisnika, prilagodljivost te smanjivanje odnosno potpuno izbjegavanje administracije. Treba napomenuti kako ne postoji sveobuhvatan opis svih pojavnih oblika Crystal metode, pa je zbog toga teže shvatiti način na koji funkcioniraju ove metode.

4.1.4. Scrum

Scrum je agilna metodologija koje je razvijena početkom 90-ih godina. Scrum je proces koji slijedi „organizirani kaos“. Temelji se na ideji da je proces razvoja kompliciran i nepredvidljiv te da može biti definiran jedino „labavim“ skupom aktivnosti.

Projekt je podijeljen na timove odnosno scrum-ove koji broje 6-9 članova. Svaki se tim fokusira na samostalna područja rada. Scrum majstor (engl. *scrum master*) je postavljen i odgovoran za vođenje dnevnih scrum sastanaka, mjerenje napretka, donošenje odluka te rješavanje prepreka koje stoje na putu napretka tima. Dnevni scrum sastanak ne bi trebao trajati dulje od 15 minuta. Tijekom sastanka scrum majstoru je dozvoljeno upitati članove tima samo 3 pitanja:

1. Koje stavke su dovršene od zadnjeg scrum sastanka?

2. Koji problemi su otkriveni koji moraju biti riješeni?
3. Koje nove zadatke ima smisla dovršiti od strane tima do slijedećeg scrum sastanka?

Na početku projekta, potrebno je kreirati listu želja i potreba kupaca tj. korisnika pod nazivom „*backlog*“. Scrum metodologiju nastavlja 30-dnevni agilni ciklus pod nazivom „*sprints*“. Svaki sprint uzima skupinu značajki iz „*backlog-a*“ radi razvoja. Tijekom sprinta, timu je dana potpuna kontrola kako bi uspješno završio sprint. Na kraju sprinta provodi se javno izlaganje za kupca, kako bi se:

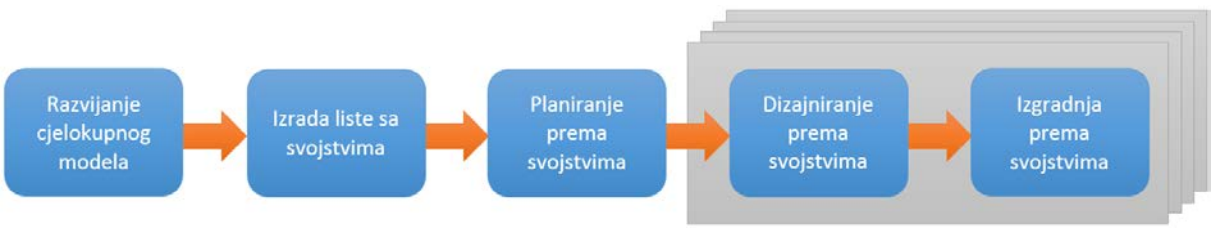
- Demonstriralo kupcu što se postiglo.
- Dalo *developerima* osjećaj ispunjenja.
- Osiguralo da je softver propisno integriran i testiran.
- Osiguralo da je ostvaren stvaran napredak na projektu.

4.1.5. Feature Driven Development (FDD)

Razvoj temeljen na svojstvima (engl. *Feature Driven Development - FDD*) je agilan i prilagodljiv pristup za razvoj sustava. FDD je prvi put predstavljen javnosti 1999. godine u knjizi „*Java Modeling In Color with UML*“, dok je prvi put primijenjen 1997. godine na projektu Singapurske banke.

FDD ne pokriva cijeli proces razvoja softvera, već se fokusira na faze dizajna i izgradnje. FDD se sastoji od pet sekvencijalnih procesa i pruža metode, tehnike i smjernice koje su potrebne za sudionike projekta kako bi dovršili sustav. Nadalje, FDD uključuje uloge, ciljeve i rokove potrebne za projekt. Za razliku od drugih agilnih metoda, FDD tvrdi kako je pogodan za razvoj kritičnih sustava. Kao što sam naziv govori, svojstva (engl. *features*) su bitan aspekt FDD-a. Svojstvo je mala funkcija usmjerena prema klijentima. Svojstva su zapravo, primarni izvor zahtjeva i primarni input u planiranju.

Slika 22: Procesi FDD-a



Izvor: vlastita izrada prema http://www.agilemodeling.com/essays/fdd.htm, [6. travnja 2015.]

Kao što vidimo na Slici 22, FDD se sastoji od pet glavnih aktivnosti koje se izvode iterativno. Radi iterativnosti, procesi se razvijaju sa brzim prilagodbama na iznenadne promjene u zahtjevima i poslovnim potrebama. U nastavku su objašnjene aktivnosti.

Pri **razvijanju cjelokupnog modela**, eksperti domene (engl. *domain experts*) su već svjesni opsega, konteksta i zahtjeva sustava koji će se razvijati. Ekspert područja može biti korisnik, klijent, sponzor ili poslovni analitičar, a zadatak mu je da stekne znanje o tome kako bi se trebali različiti zahtjevi izvršiti. Na početku projekta cilj je identificirati i razumjeti osnove područja sustava. Iako FDD ne naglašava potrebu prikupljanja i upravljanja zahtjeva, dokumentirani zahtjevi se, poput *use case-a*, pojavljuju u ovoj fazi. Eksperti predstavljaju tzv. prohod (engl. *walkthrough*) gdje se članovi tima informiraju o višim razinama sustava.

Drugi korak je **izrada liste sa svojstvima**, grupirajući ih u srodne skupove. Prohodi i modeli objekta iz prethodne faze služe kao baza pri izradi liste. U listi, razvojni tim predstavlja svakom klijentu vrijednosti funkcija uključene u sustav. Lista je pregledana od strane korisnika i sponzora kako bi odredili njenu valjanost i cjelovitost.

Planiranje prema svojstvima uključuje stvaranje plana visoke razine gdje su skupovi sa svojstvima postavljeni sekvencijalno u odnosu na njihov prioritet te se dodjeljuje glavnim programerima. Nadalje, klase identificirane u prvom koraku dodjeljuju se pojedinim programerima, odnosno vlasnicima klasa (engl. *class owners*). Zadaci vlasnika klasa su dizajniranje, programiranje, testiranje i dokumentiranje, koji je i odgovoran za razvoj klasa.

Glavnina FDD projekta sastoji se u četvrtom (**dizajniranje prema svojstvima**) i petom (**izgradnja prema svojstvima**) koraku. Ovi koraci uključuju detaljno modeliranje, programiranje, testiranje i dovršavanje sustava. Dizajniranje i izgradnja su iterativne procedure, a jedna iteracija bi trebala trajati nekoliko dana do maksimalno dva tjedna. Može postojati istodobno više timova koji dizajniraju i izgrađuju vlastiti skup svojstava.

FDD je pogodan za nove projekte, jačanje projekata i nadogradnju postojećeg koda, te za projekte čija je zadaća stvoriti drugu verziju postojeće aplikacije.

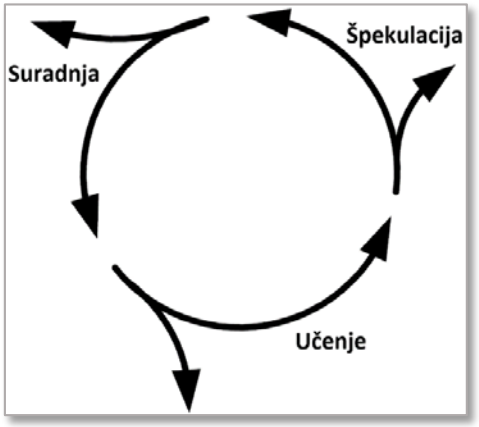
4.1.6. Adaptive Software Development (ASD)

Prilagodljiv razvoj softvera (engl. *Adaptive Software Development - ASD*) je razvijen od strane Jamesa A. Highsmitha III 2000. godine. ASD se uglavnom fokusira na probleme pri

razvoju kompleksnih i velikih sustava. Metoda snažno potiče inkrementalni i iterativni razvoj sa stalnim prototipiranjem. Cilj mu je pružiti razvojno okuženje sa dovoljno smjernica kako bi spasio projekte od kaosa, pazeći pritom da ne suzbije kreativnost.

ASD projekt se provodi u tri faze ciklusa (Slika 23). Te faze su slijedeće: špekulacija, suradnja i učenje.

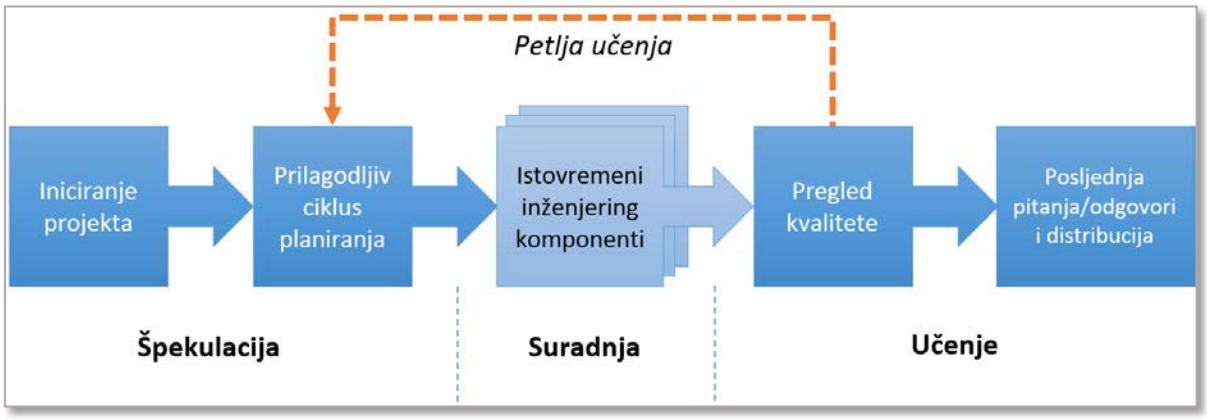
Slika 23: Faze ASD projekta



Izvor: Abrahamsson, P., Salo, O. i Ronkainen, J. (2002): Agile software development methods - Review and analysis, VTT Electronics, University of Oulu, Finland

Faza „**Špekulacija**“ se koristi umjesto Planiranja, budući da se plan percipira kao nešto neizvjesno i čija odstupanja ukazuju na neuspjeh. „**Suradnja**“ naglašava važnost timskog rada pri razvijanju promjenjivih sustava. „**Učenje**“ naglašava potrebu prepoznavanja i reagiranja na pogreške kao i činjenicu da se zahtjevi mogu mijenjati tijekom razvoja sustava.

Slika 24: Faze ASD ciklusa



Izvor: vlastita izrada prema Abrahamsson, P., Salo, O. i Ronkainen, J. (2002): Agile software development methods - Review and analysis, VTT Electronics, University of Oulu, Finland

Slika 24 daje nam detaljniju ilustraciju ASD ciklusa. „**Iniciranje projekta**“ određuje uporište projekta i započinje definiranjem misije projekta. Misija je ono za čim treba težiti završni proizvod, pa je shodno tome razvoj usmjeren na način kako bi se ostvarila misija. Ova faza također ispravlja raspored i ciljeve za razvojni ciklus.

ASD je isključivo orijentiran na komponente, a ne na zadatke. Komponente su veće i zatvorenije cjeline od objekata. Kad sustav treba neku uslugu, on poziva komponentu preko njezinog javnog sučelja, ne brinući se o tome gdje i kako se ona izvršava. [20] Dakle, naglasak se stavlja na rezultate i njihovu kvalitetu, a ne na same zadatke ili procese koji služe za dobivanje rezultata. Način na koji to ASD rješava je putem prilagodljivih razvojnih ciklusa koji sadrže fazu „**Suradnja**“ gdje nekoliko komponenti mogu biti pod istovremenim razvojem.

Osnovu za daljnje cikluse (**Petlja učenja**) dobivamo kroz ponavljajuće preglede kvalitete koji se fokusiraju na funkcionalnosti softvera razvijene tijekom ciklusa. Pregledi se obavljaju uz prisutnost kupaca. No, budući da su kvalitetni pregledi vrlo rijetki, prisutnost kupaca podupiru JAD sesije gdje se *developeri* i predstavnici kupaca nalaze kako bi razgovarali o željenim značajkama proizvoda te kako bi se poboljšala komunikacija.

Kod završna faze, ASD ne uzima u obzir kako bi se ona trebala provesti, ali naglašava važnost naučene lekcije.

Najznačajniji problem kod ASD-a je taj da je njegove postupke teško identificirati te ostavlja puno detalja otvorenima. ASD možemo promatrati kao način izgradnje prilagodljive organizacijske kulture koja se ne bavi specifičnostima.

4.2. Open-source

Softver otvorenog koda (engl. *Open-source software*) je razvijan i održavan od grupe volontera. Svi artefakti projekta, od dokumentacija zahtjeva pa do izvornog programskog koda, su dostupni svakome.

Open-source projekt obično započinje nečijim razvojem aplikacije te njegovim postavljanjem na web stranicu. Praktički bilo tko može predložiti nove zahtjeve i budući da je izvorni kod svima dostupan besplatno, svatko može implementirati zahtjeve koji nisu dio službene polazne vrijednosti. Postoji proces u kojemu se prijedlozi i implementacije prioritiziraju, te proces za prihvaćanje novih programskih kodova i sposobnosti za polazne vrijednosti projekta. Ukoliko

se pojave nedostaci, prijavljuju se, te ostali (volonteri) mogu raditi na njima kako bih ih ispravili. Ovaj proces se ponavlja, te se na taj način aplikacija usavršava glede sposobnosti i stabilnosti. Na taj način, *open-source* projekti se zapravo razvijaju na iterativan način. Još k tome, *open-source* zajednica utječe na golem proces testiranja budući da se aplikacija usavršuje mnogo puta.

Jedan od osnovnih razloga zašto se projekt razvija kao *open-source* je kako bi se iskoristila velika količina resursa koja inače ne bi bila dostupna. Neki vjeruju kako upravo *open-source* može postati dominantnim procesom pri proizvodnji tj. izradi softvera.

Ostali razlozi za izradu *open-source* projekta:

- Osjećaj profesionalnog zadovoljstva, *developeri* su motiviraniji budući da sami biraju na čemu će raditi.
- Omogućavanje nadograđivanja i integracije.
- Akademski i istraživački razlozi.
- Održavanje veće stabilnosti.
- Narušavanje izgleda konkurentskog proizvoda.
- Stjecanje većeg tržišnog znanja.

Nepovoljni razlozi za izradu:

- Dokumentacija je loša i nedosljedna. *Open-source* projekti posebno imaju problema pri sinkronizaciji dizajna sustava i programskog koda, pa dokumentacija zna biti nekonzistentna.
- Ne postoji garancija da će se *developeri* uopće pojaviti.
- Nema kontrole upravljanja.
- Ne postoji kontrola oko zahtjeva.
- Sve je dostupno svakome, pa i konkurentima.

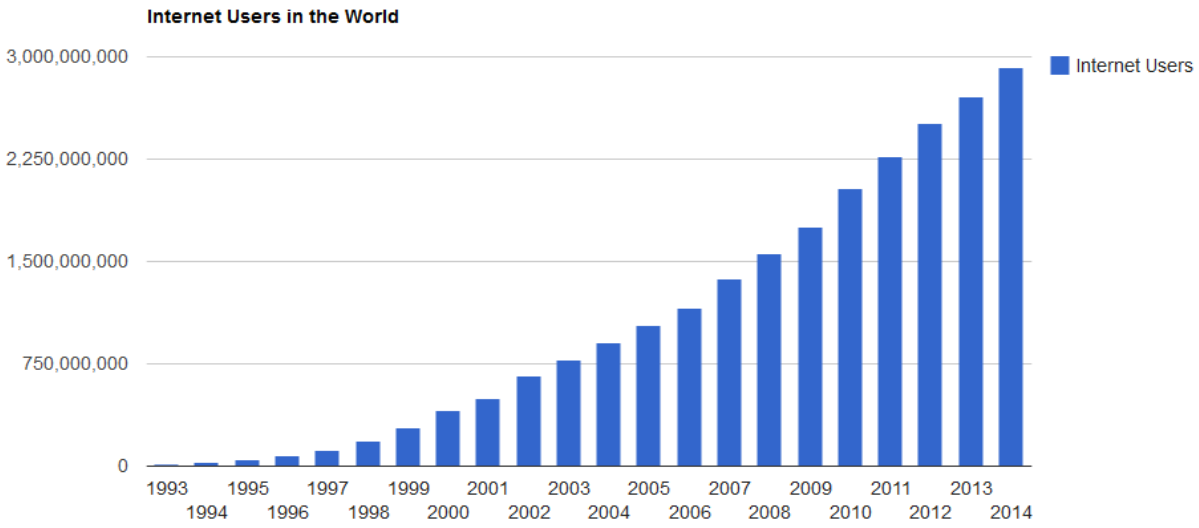
4.3. Trendovi razvoja web i mobilnih aplikacija

U ovom poglavlju ćemo se bazirati na trendovima razvoja web i mobilnih web aplikacija, budući da ove platforme dominiraju u našim svakodnevnim životima, te se primjenjuju u svim područjima ljudskog djelovanja (posao, edukacija, industrija, turizam, zabava).

4.3.1. Tehnološki trendovi

World Wide Web i Internet[8] predstavljaju najznačajnije tehnološke promjene, ne samo u računalnoj povijesti, već i povijesti civilizacije. Web je postao najmoćniji komunikacijski medij, te ga prema stranici *internetlivestats.com* [32] trenutačno koristi oko 40% svjetske populacije (Graf 1 i Tablica 5).

Graf 1: Broj Internet korisnika u svijetu



Izvor: http://www.internetlivestats.com/internet-users/, [3. svibnja 2015.]

Iz Tablice 5 možemo vidjeti kako se broj korisnika kroz godine višestruko povećava, gdje je npr. samo prije 5 godina broj korisnika iznosio svega 25% populacije.

[8] Internet je javno dostupna globalna paketna podatkovna mreža koja zajedno povezuje računala i računalne mreže, dok Web predstavlja uslugu odnosno način pristupanja informacijama preko Interneta.

Tablica 5: Broj Internet korisnika u odnosu na svjetsku populaciju

Year (July 1)	Internet Users	World Population	Penetration (% of Pop. with Internet)
2014*	**2,925,249,355**	7,243,784,121	**40.4%**
2009	**1,752,333,178**	6,834,721,930	**25.6%**
2004	**910,060,180**	6,435,705,600	**14.1%**
1999	**280,866,670**	6,051,478,010	**4.6%**

Izvor: prilagođeno prema http://www.internetlivestats.com/internet-users/, [3. svibnja 2015.]

Na istoj stranici [32] možemo pronaći zanimljive statističke podatke o važnosti i sveukupnom rastu informacijsko komunikacijske tehnologije i njenih popratnih uređaja, pa tako imamo:

- oko 970 milijuna web stranica,
- oko 200 milijardi poslanih e-mailova dnevno,
- oko 150 milijuna Skype poziva dnevno,
- oko 480 milijuna prodanih pametnih telefona u jednoj godini,
- oko 90 milijuna prodanih tablet računala u jednoj godini,
- oko 75 milijuna prodanih prijenosnih računala u jednoj godini.

Treba naglasiti kako mobilni uređaji postaju sve popularniji u načinu pristupanja web aplikacijama odnosno Internetu u cjelini. Naime, u siječnju 2014. godine u SAD-u je prvi put zabilježen veći pristup Internetu putem mobilnih web aplikacija u odnosu na osobna računala. Prema podacima *comScore-a* [25] u navedenom je razdoblju udio posjećenosti Interneta putem mobilnih uređaja iznosio 55%, od čega se 47% odnosilo na mobilne aplikacije, a 8% na pristup putem mobilnih web preglednika. Udio posjećenosti putem osobnih računala je iznosio 45%.

Agencija za istraživanje tržišta *GfK* objavila je podatke o posjedovanju mobilnih uređaja u Hrvatskoj. Podaci se odnose na 2012. godinu. Navodi se [37] kako je 90% kućanstva posjedovalo barem jedan mobitel, te se prosječan broj mobitela po kućanstvu kretao oko 2,5 komada. U 16% kućanstava nalazi se barem jedan pametni telefon, dok u kućanstvima preko 11 tisuća kuna mjesečnih prihoda posjedovanje pametnih telefona iznosi 57%. Osobna i prijenosna računala su prisutna u 70% kućanstva.

Američka tvrtka *Gartner*, koja je vodeća svjetska informatička tvrtka za istraživanje informatičke tehnologije i savjetovanje, na svom simpoziju ističe 10 najboljih tehnologija i trendova koji će biti od strateške važnosti za većinu organizacija u narednoj, ali i u slijedećih

nekoliko godina. Osvrnut ćemo se na predviđanja [27] [28] [29] [30] najznačajnijih tehnoloških trendova od 2012. do 2015. godine, koje možemo vidjeti u Tablici 6.

Tablica 6: Tehnološki trendovi 2012-2015

Top 10 Strategic Technology Trends For 2012	Top 10 Strategic Technology Trends For 2013	Top 10 Strategic Technology Trends For 2014	Top 10 Strategic Technology Trends For 2015
•Media Tablets and Beyond	•Mobile device battles	•Mobile Device Diversity and Management	•Computing Everywhere
•Mobile-Centric Applications and Interfaces	•Mobile applications & HTML5	•Mobile Apps and Applications	•The Internet of Things
•Contextual and Social User Experience	•Personal Cloud	•The Internet of Everything	•3D Printing
•Internet of Things	•Internet of Things	•Hybrid Cloud and IT as Service Broker	•Advanced, Pervasive and Invisible Analytics
•App Stores and Marketplaces	•Hybrid IT and Cloud Computing	•Cloud/Client Architecture	•Context-Rich Systems
•Next-Generation Analytics	•Strategic Big Data	•The Era of Personal Cloud	•Smart Machines
•Big Data	•Actionable Analytics	•Software Defined Anything	•Cloud/Client Computing
•In-Memory Computing	•Mainstream In-Memory Computing	•Web-Scale IT	•Software-Defined Applications and Infrastructure
•Extreme Low-Energy Servers	•Integrated Ecosystems	•Smart Machines	•Web-Scale IT
•Cloud Computing	•Enterprise App Stores	•3D Printing	•Risk-Based Security and Self-Protection

Izvor: vlastita izrada prema http://www.gartner.com/newsroom/id/1826214, http://www.gartner.com/newsroom/id/2209615, http://www.gartner.com/newsroom/id/2603623, http://www.gartner.com/newsroom/id/2867917, [3. svibnja 2015.]

Iz prethodne Tablice možemo zaključiti kako je u zadnje četiri godine Gartner uvelike bazirao predviđanja na tehnologijama vezane uz web i mobilne tehnologije, te možemo vidjeti kako se Računarstvo u oblaku (engl. *Cloud computing*) sve češće pojavljuje kao tehnologija koji bi mogla uzeti primat u IT svijetu.

Nadalje, prema izvještaju koju je izradila američka tvrtka *DANA Inc*. [23], objavljeni su slijedeći investicijski prioriteti u IT-u za 2015. godinu:

- Razvoj mobilnih aplikacija,
- Servisno orijentirana arhitektura (engl. *Service Oriented Architecture - SOA*),
- Službe za podršku,
- Infrastruktura,
- Testiranje aplikacija i osiguravanje kvalitete.

4.3.2. Definiranje web aplikacija

World Wide Web prožima svaki dio modernog društva, od ekonomije, industrije, zdravstva, pa sve to edukacije i zabave. Evolucijom Weba okupile su se naoko nepovezane discipline, poput medija, informacijske znanosti, informacijsko komunikacijske tehnologije itd. Zahvaljujući računalima i prijenosnim računalima te pojavom pametnih telefona i tableta, koji su postali dio svakodnevice, Web je postao dominantna platforma za razvoj poslovnih, društvenih i organizacijskih informacijskih sustava. Web je također postao univerzalno korisničko sučelje za razne poslovne aplikacije i informacijske sustave. Kako se naša ovisnost o web aplikacijama sve više povećava, od presudne je važnosti da web aplikacije budu pouzdane i kvalitetne. Radi toga, razvoj i dizajn takvih aplikacija je postao puno kompleksniji kroz godine. No, mnogi korisnici i *developeri* imaju krivu sliku o razvoju web aplikacija kao o nečemu vrlo jednostavnim.

Moderne web aplikacije su kompleksni informacijski sustavi. Radi toga je potreban inženjerski pristup razvoju takvih aplikacija. **Web inženjerstvo**, koje je temeljeno na softverskom inženjerstvu, obuhvaća uporabu sustavnih i mjerljivih pristupa kako bi ostvarili specifikacije, implementaciju, rad i održavanje visokokvalitetnih web aplikacija. Glavna posebnost web aplikacija u odnosu na tradicionalne softverske aplikacije je način na koji se koristi Web. Naime, Web tehnologije i standardi se u isto vrijeme koriste i kao razvojna platforma i kao korisnička platforma. Web aplikaciju, stoga, možemo definirati kao softverski sustav koji je baziran na tehnologijama i standardima tzv. W3C konzorcija[9] (engl. *World Wide Web Consortium*) koji pruža specifične web resurse putem korisničkog sučelja tj. web preglednika. Iz definicije slijedi kako su tehnologija i korisnička interakcija ključni čimbenici, te kako sama tehnologija ne čini web aplikaciju, kao što ni web stranice bez odgovarajućih softverskih komponenti ne predstavljaju web aplikacije.

Sa gledišta softverskog inženjerstva, razvoj web aplikacija spada u novu aplikacijsku domenu. Unatoč nekim sličnostima sa tradicionalnim aplikacijama, posebne karakteristike web aplikacije zahtijevaju prilagodbu mnogih pristupa softverskog inženjerstva ili pak razvoj potpuno novih pristupa. Prema Kappel, Proll, Reich i Retschitzegger [7] osnovni principi web inženjerstva, koji su zapravo slični principima softverskog inženjerstva, mogu se definirati na slijedeći način:

[9] W3C je glavna međunarodna organizacija za World Wide Web standarde.

- Jasno definirani ciljevi i zahtjevi,
- Sistematičan razvoj web aplikacije u fazama,
- Pažljivo planiranje tih faza,
- Kontinuirana revizija cjelokupnog razvojnog procesa.

Web aplikacije se razlikuju u odnosu na tradicionalne aplikacije u raznim značajkama. O samom tipu web aplikacije ovisi jesu li određene karakteristike prisutne te u kojoj mjeri. Te karakteristike su razlog zbog čega se mnogi koncepti, metode, tehnike i alati softverskog inženjerstva moraju prilagoditi potrebama web inženjerstva ili su pak potpuno neadekvatne. Spomenute karakteristike mogu se grupirati u tri dimenzije: proizvod, upotreba i razvoj.

Karakteristike koje se odnose na proizvod čine glavnu okosnicu web aplikacije, a sastoji se od sadržaja, navigacijske strukture i korisničkog sučelja. Aktivnosti poput generiranja sadržaja, integracije i ažuriranja su jednako bitne kao i sam razvoj web aplikacije. *Developeri* se, stoga, moraju staviti u ulogu autora, a ne isključivo programera. Sama kvaliteta sadržaja od velike je važnosti za proizvod, te se ovisno o strukturi, sadržaj može prikazati u obliku tablice, teksta, animacije, te audio i video formata. Jedna od specifičnosti web aplikacija je i njezina nelinearnost hipertekstualnih dokumenata[10]. Nadalje, korisničko sučelje ima dvije glavne značajke, a to su estetika i samorazumljivost. Za razliku od tradicionalnih aplikacija, estetika igra veliku ulogu radi sve veće konkurencije. Također je bitno da su te web aplikacije razumljive same po sebi, odnosno da se mogu koristiti bez dodatne dokumentacije.

Za razliku od tradicionalnih aplikacija, upotreba web aplikacija je vrlo raznolika. Tako se korisnici razlikuju u kulturnoj pozadini (socijalni kontekst), uređaji mogu biti različitih hardverskih i softverskih karakteristika (tehnički kontekst), a vrijeme i lokacija odakle se pristupa aplikaciji ne može se predvidjeti (prirodni kontekst). **Socijalni kontekst** se odnosi na specifične korisničke aspekte koje karakteriziraju spontanost i multikulturalnost. Korisnici pristupaju web aplikaciji kad god to oni požele, te se od njih ne može računati na vjernost određenom davatelju usluga. Web aplikacije su razvijane za različite grupe korisnika. Kako bi se omogućile odgovarajuće prilagodbe, moraju se stvoriti razne pretpostavke o kontekstu korisnika pri razvoju aplikacije. Često se zahtijeva od samih korisnika da definiraju svoje postavke pri personalizaciji. **Tehnički kontekst** se odnosi na svojstva koja se odnose na mrežni priključak, te na hardver i softver uređaja koji se koristi za pristup web aplikaciji. Web

[10] Hipertekstualni dokumenti se od običnih dokumenata razlikuju po tome što sadrže hiperveze (hiperlinkove) kojima su povezani s drugim (hipertekstualnim) dokumentima.

aplikacije, naime, moraju biti prilagođene svim vrstama uređaja, pogotovo sve rasprostranjenijim mobilnim uređajima sa raznim specifikacijama. Također treba voditi računa o raznim web preglednicima (i njihovim verzijama), budući da imaju razne funkcionalnosti i ograničenja. **Prirodni kontekst** uključuje aspekte pristupa korisnika s obzirom na vrijeme i lokaciju. Budući da se aplikaciji može pristupiti iz različitih dijelova svijeta, treba voditi računa o regionalnim, kulturnim i jezičnim razlikama. S obzirom da se aplikaciji može pristupiti u bilo koje vrijeme, povećava se važnost stabilnosti web aplikacije.

Karakteristike razvoja web aplikacija čine potrebni resursi, kao što su razvojni tim i tehnička infrastruktura, sam razvojni proces, te integracija postojećih rješenja. Na razvoj web aplikacija snažno utječe činjenica da su razvojni timovi multidisciplinarni i uglavnom dosta mladi. Budući da su web aplikacije kombinacije marketinga, informatike, umjetnosti i tehnologije, njen razvoj treba promatrati kao multidisciplinarni pristup koji zahtijeva znanje i stručnost iz raznih područja. Web aplikacije uglavnom razvijaju mlađi *developeri* koji samim time imaju manje iskustva. Tehničku infrastrukturu karakterizira nehomogenost i nezrelost. Nehomogenost se očituje u tome što ne postoji način da se utječe na web preglednike korisnika te njihove individualne postavke. Komponente koje su korištene u aplikacijama često su nezrele, odnosno pojavljuju se greške (engl. *bug*) ili postoji nedostatak željene funkcionalnosti.

Sam razvojni proces je osnova za sve karakteristike povezane sa razvojem, te je pod utjecajem fleksibilnosti i paralelizma. Od presudne je važnosti fleksibilno reagirati na promjene uvjeta, budući da je nemoguće pridržavati se unaprijed određenom planu projekta. Web aplikacije uglavnom paralelno razvijaju više grupa razvojnog tima. Posebna karakteristika mnogih web aplikacija se javlja u potrebi interne i eksterne integracije. Integracija se ne odnosi samo na tehnički aspekt, već i na organizacijski aspekt i sadržaj.

4.3.2.1. Klasifikacija Web sustava

Prema Murugesanu [9] Web sustave i aplikacije možemo klasificirati na temelju njihovih ključnih značajki i tehnologija koje se koriste za njihovu izradu i to na slijedeći način:

- Web 1.0
  - Statični Web (engl. *Static Web*)
  - Dinamički Web (engl. *Dynamic Web*)
- Web 2.0

- Mobilni Web (engl. *Mobile Web*)
- Semantički Web (engl. *Semantic Web*)

Statični Web je zapravo kolekcija statičnih HTML web stranica koje pružaju neke najosnovnije informacije npr. o proizvodima i uslugama. U počecima, većina web stranica je zapravo bila statična, da bi nakon nekog vremena Web postao **dinamički**. Drugim riječima, dinamičke web stranice su se mogle razvijati na temelju sadržaja pohranjenih u bazama podataka kako bi omogućili korisnicima pružanje prilagođenih informacija. No, ovakve su stranice pružale jednosmjernu interakciju i ograničenu interaktivnost korisnika. Danas se statični i dinamički Web objedinjuje u Web 1.0 klasifikaciju.

Druga generacija Weba, **Web 2.0**, omogućuje korisnicima suradnju i dijeljenje informacija online na nove načine. Primjeri takvih web stranica su društvene mreže poput Facebook-a, stranice za dijeljenje video sadržaja poput YouTube-a i kolaboracijske stranice poput Wikipedije. Web 2.0 je skup tehnologija, poslovnih strategija i društvenih trendova, koji nudi korisnicima pametno korisničko sučelje kako bi mogli generirati i uređivati različite sadržaje na Webu. Ugradnjom novih tehnologija poput AJAX-a, Web postaje sve dinamičniji i vrlo interaktivan te sve više dolazi do izražaja korisnički doprinos.

Sve veći napredci u mobilnoj i mrežnoj tehnologiji, kao i sve veća pristupačnost mobilnih uređaja, poput pametnih telefona i tablet računala, doprinijeli su porastu sve većeg broja korisnika koji pristupaju Webu mobilnim putem. Mobilni uređaji će zasigurno postati vodeća platforma za pristup Internetu, a time i web aplikacijama. Sve se više aplikacija zapravo prilagođava takvom načinu pristupanja Webu, pa je stoga pokrenuta nova inicijativa pod nazivom **Mobilni Web**. Mobilne web aplikacije u odnosu na tradicionalne (*desktop*) aplikacije nude dodatne mogućnosti, poput personalizacije i pružanja servisa ovisno o lokaciji.

Semantički Web je proširenje sadašnjeg Weba u kojemu se informacijama daju dobro definirana značenja, te se omogućuje bolja suradnja računala i ljudi. U sadašnjim web aplikacijama, informacija je predstavljena na način da ju ljudi mogu lako razumjeti. No, računala ne mogu smisleno rukovati s takvim informacijama, pa je upravo zadatak semantičkog Weba prevladati tu zapreku. Cilj semantičkog Weba je olakšati globalnu razmjenu informacija pomoću računala kako bi se informacije na Webu mogle koristiti što učinkovitije, te da bi se postigla automatizacija i integracija u raznim aplikacijama.

4.3.3. Tehnologije za izradu web stranica i aplikacija

Prve Web stranice sastojale su se od zbirke dokumenata kodiranih u HTML-u. Prikazani sadržaji bili su statični, odnosno bili su određeni u HTML dokumentima i oblikovani pomoću specijalnih HTML elemenata koji su definirali prezentacijska svojstva. Kroz godine, pojavilo se nekoliko novih tehnologija, koje ćemo obraditi u nastavku.

4.3.3.1. HyperText Transfer Protocol (HTTP)

HTTP se zasniva na arhitekturi klijent-poslužitelj koji definira standardni format za određivanje zahtjeva određenih resursa na Webu. Korisnik preko HTTP-a, pomoću korisničke aplikacije (web preglednika), može poslati zahtjev za određenim resursima (najčešće HTML stranica) dostupnih na poslužitelju (web server). Kako bi locirao i pristupio resursima potreban je identifikacijski mehanizam pod nazivom *Uniform Resource Locator (URL)*[11], naziv ili IP[12] adresa uređaja koji nudi resurse, broj porta, te naziv dokumenta i lokaciju u sustavu poslužitelja. Nakon zaprimljenog zahtjeva, poslužitelj locira resurs i šalje odgovor klijentu.

4.3.3.2. HyperText Markup Language (HTML)

HTML je *markup*[13] jezik za opisivanje Web dokumenta, te se koristi za vizualno oblikovanje Web stranica. HTML dokumenti su opisani pomoću tzv. HTML oznaka (engl. *tags*). Svaka HTML oznaka opisuje drugačiji kontekst dokumenta. Web preglednik naposljetku prima te sadržaje, kodirane u *markup* jeziku, te ih transformira u čitljivi dokument interpretirajući značenja HTML oznaka.

Od svojih početaka, HTML je imao više verzija koje su prikazane u Tablici 7. Prva verzija HTML-a se koristila za postavljanje jednostavnih web stranica između 1989. i 1995. godine. 1995. godine IETF[14] (*Internet Engineering Task Force*) je standardizirao HTML i nazvao ga HTML 2.0. Nakon toga je 1997. godine W3C predstavio HTML 3.2, kojeg su naslijedili HTML

[11] URL je globalna adresa dokumenata i drugih resursa na World Wide Webu.
[12] Internet Protocol (IP): skup pravila koja definiraju jedan vid komunikacije, određeno standardom.
[13] Markup jezici su dizajnirani za obradu, definiranje i prezentaciju teksta. Jezik specificira kod za oblikovanje izgleda i stila unutar tekstualne datoteke.
[14] IETF je organizacija za Internet standarde.

4.0 i HTML 4.01. W3C se zatim odlučio fokusirati na tzv. proširivi HTML (XHTML), iako su preporučili *developerima* korištenje HTML 4.01 standarda. Najnoviji standard je izašao 2014. godine pod nazivom HTML5, iako još nema široku primjenu kao prethodne verzije.

Tablica 7: HTML verzije

HTML verzija	Godina
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.0	1998
HTML 4.01	1999
XHTML	2000
HTML5	2014

Izvor: vlastita izrada prema Casteleyn, S., Daniel, F., Dolog, P. i Matera, M. (2009): Engineering Web Applications, Springer-Verlag, Berlin

4.3.3.3. Cascading Style Sheets (CSS)

CSS određuje kako će HTML elementi biti prikazani. CSS omogućuje dizajnerima da definiraju izgled i dojam web stranice i to odvojeno od *markup* stranice i sadržaja pisanih u HTML-u. Zajedno sa HTML-om i JavaScript-om, CSS predstavlja osnovnu tehnologiju za izradu vizualno privlačnih web stranica, kao i korisničkih sučelja za web i mobilne aplikacije. Razvijen je radi omogućavanja odvajanja sadržaja dokumenata od prezentacije dokumenata, uključujući elemente poput izgleda stranice, boje i fonta. CSS je također imao nekoliko verzija, a to su: CSS 1, CSS 2, CSS 2.1, te najnoviji CSS3.

4.3.3.4. eXtensible Markup Language (XML)

Budući da HTML nije proširiv te ima svoja ograničenja, koja sprječavaju dizajnere u ispunjavanju različitih vrsta zahtjeva u specifičnim aplikacijskim domenama, razvijen je novi jezik pod nazivom XML. Dokumenti bazirani na XML-u upotrebljavaju se kako bi mogli opisati i prenositi podatke, dok HTML služi za prikazivanje tih podataka. Ovaj jezik ne služi kao zamjena za HTML, već se radi o tome da XML prenosi informacije. Prednost XML-a je što pruža način izražavanja sadržaja stranice, bez da se isprepleću svojstva sadržaja i prezentacije.

4.3.3.5. Skriptni jezici

Skripte su programi koji su interpretirani i izvršeni pomoću drugog programa (web preglednika) kada se stranica učita ili kad korisnik poziva određene događaje preko HTML elemenata stranice. Skriptni jezici predstavljaju aktivni dio u dinamičkom HTML pristupu, dok *markup* jezik predstavlja statičan dio koji je podložan dinamičkim izmjenama. Najčešće se koriste za dodavanje funkcionalnosti web stranici, poput različitih stilova izbornika ili kako bi poslužili za dinamičke reklame.

Skriptni jezici tipično se koriste za povezivanje više samostalnih programa u cjelinu „lijepljenjem“ (engl. *glue language*). Obično imaju ugrađenu podršku za intenzivnu obradu teksta, a najčešće se koriste za pisanje programa usmjerenih specifičnim potrebama korisnika. [18] Postoji više skriptnih jezika, a neki od njih su: JavaScript, JScript, VBScript, Perl, Python, Ruby, Tcl, itd. **JavaScript** podržava većina današnjih web preglednika, dok je JScript podržan samo od strane Microsoftovog Internet Explorera. Radi toga, JavaScript je postao tako reći standardni skriptni jezik u web inženjerstvu.

Postoje dvije vrste skriptnih jezika, gdje *client-side scripting* može biti ugrađen unutar HTML-a te se najčešće koristi za dodavanje funkcionalnosti web stranici. Druga vrsta skriptnih jezika je *server-side scripting*, koji manipuliraju podacima i to obično u bazi podataka - na poslužitelju. Danas postoji nekoliko vrsta *server-side* skriptnih jezika, a najistaknutiji su slijedeća tri:

- **Hypertext Preprocessor (PHP)**: je jedan od najkorištenijih skriptnih jezika. Svoju popularnost zasigurno duguje tome što je besplatan, otvorenog je koda, dostupan na svim većim platformama te ga podržavaju svi značajniji web poslužitelji.
- **Active Server Pages (ASP)**: je Microsoftova tehnologija zatvorenog koda. Za razliku od PHP-a, nije vezan za određeni skriptni jezik. ASP koristi VBScript kao zadani jezik, ali se i drugi mogu koristiti.
- **Perl**: je poput PHP-a vrlo popularan skriptni jezik otvorenog koda. Perl se uglavnom koristi za grafičko programiranje, administraciju sustava, mrežno programiranje, u financijama, bioinformatici i ostalim aplikacijama.

4.3.3.6. Asynchronous JavaScript and XML (AJAX)

AJAX je skup tehnika u razvoju web aplikacija kako bi proizveli asinkrone[15] web aplikacije, te omogućuje *developerima* korištenje dinamičkog HTML-a na najbolji mogući način. AJAX nije neki novi programski jezik, već se radi o novom načinu korištenja postojećih standarda. Omogućuje razmjenu podataka sa poslužiteljem, kao i ažuriranje dijelova web stranica, bez ponovnog pokretanja cijele stranice. Web stranice koje ne koriste AJAX, moraju ponovno pokrenuti čitavu stranicu ako se sadržaj promijeni. Korištenje AJAX-a dovodi do puno responzivnijih i ergonomskih aplikacija. Primjeri aplikacija koje koriste AJAX su: Gmail, Youtube, Facebook i Google Maps.

AJAX koristi kombinaciju slijedećih standarda:

- XMLHttpRequest object: služi za asinkronu razmjenu podataka s poslužiteljem.
- JavaScript/DOM: služi za prikaz/interakciju s informacijama.
- CSS: služi za stiliziranje podataka.
- XML: često se koristi kao format za prijenos podataka.

4.3.3.7. Service-oriented architecture (SOA)

Web servisi omogućuju aplikaciji integraciju i ponovnu upotrebu aplikacijske logike koja radi na udaljenom poslužitelju. Situacija je slična onoj kada korisnik doziva web aplikaciju koja radi na web poslužitelju, tako i web aplikacija može dozvati web servis koji radi na udaljenom poslužitelju. Prema Mangeru [20] Web servisi predstavljaju nadogradnju klasične Web tehnologije. Njime sadržaji s weba postaju dostupni i drugim programima. Točnije rečeno, Web servis je resurs na Webu, koji je prikazan na standardizirani način i koji se može koristiti iz nekog drugog programa.

Njihova svrha je pružiti određene funkcionalnosti koje će putem Interneta koristiti razne aplikacije. Aplikacije šalju Web servisima ulazne podatke i kao rezultat od Web servisa dobivaju izlazne podatke. Bitno je naglasiti da su izlazni podaci funkcija ulaznih podataka. To znači da izlazni podaci ovise isključivo o ulaznim podacima. [17]

[15] Aktivnosti se izvode u različitim vremenskim intervalima.

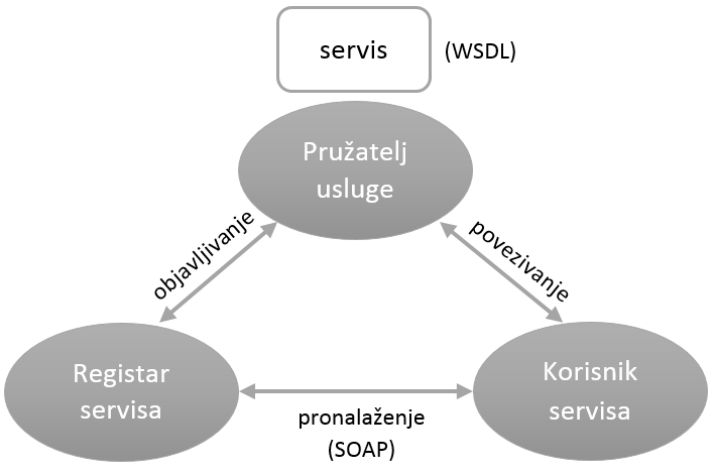
Web servisi koriste XML za kodiranje i dekodiranje podataka, dok SOAP (vrsta protokola) služi za povezivanje tih podataka. Pomoću web servisa, omogućena je razmjena podataka između različitih aplikacija kao i različitih platformi.

Servisno orijentirana arhitektura (engl. *Service-oriented architecture - SOA*) je način razvoja distribuiranih sustava gdje sistemske komponente djeluju kao samostalni servisi, te se izvršavaju na udaljenim računalima. Treba napomenuti kako SOA nije isto što i Web servis, iako su izgrađene na sličnim principima. Web servisi predstavljaju skup tehnologija koji omogućuju vezu između korisnika (aplikacije) i pružatelja usluge, dok se SOA koristi za izgradnju arhitekture koja će najbolje raditi u okruženju web servisa.

Protokoli Web servisa obuhvaćaju sve aspekte SOA-e, od temeljnih mehanizama za razmjenu informacija servisa (putem SOAP-a) do standarda programskih jezika (WS-BPEL). Ključni standardi koji se temelje na XML-u su slijedeći:

- SOAP: protokol koji služi za komunikaciju među web servisima.
- WSDL: standard za opisivanje sučelja web servisa.
- WS-BPEL: standard za programski jezik za pisanje aplikacije koja poziva Web servise i kombinira njihove ulaze i izlaze.

Slika 25: SOA arhitektura



Izvor: vlastita izrada prema Sommerville, I. (2010): Software Engineering (9th Edition), Pearson Education Inc, Boston MA, USA

Slika 25 opisuje tipičnu Servisnu orijentiranu arhitekturu. Pružatelj usluga implementira Web servis te ga opisuje putem WSDL-a. WSDL je XML jezik koji omogućuje apstraktni opis servisa u vidu vrste podataka, poruka, operacija i vrste portova (koji pak opisuju strukturu

sučelja servisa). Kako bi potencijalni korisnik servisa pronašao dotične WSDL opise servisa, opisi se objavljuju u registru servisa kojeg može pretražiti korisnik servisa. Ukoliko pronađe servis sa traženom funkcionalnošću, korisnik može povezati svoju aplikaciju na taj specifičan servis i izvršiti komunikaciju, koristeći standardne protokole servisa.

4.3.4. Razvoj web aplikacija

Razvoj web aplikacija uključuje nekoliko bitnih izazova koji podrazumijevaju usvajanje odgovarajućih tehnologija (opisane u prethodnom poglavlju 4.3.3.) i metodologija. Potrebna je konzistentna i sveobuhvatna metodologija koja će se nositi sa složenošću web aplikacija te osigurati njenu kvalitetu. *Developer* mora biti u stanju analizirati zahtjeve, prevesti ih u odgovarajući dizajn aplikacije, implementirati aplikaciju odabirom prave tehnologije i alata, testirati i provjeriti rezultat, te održavati i mijenjati aplikaciju po potrebi.

Web aplikacije su uobičajeno podijeljene u tri sloja:

- Podatkovni sloj (engl. *Data layer*)
- Aplikacijski sloj (engl. *Application layer*)
- Prezentacijski sloj (engl. *Presentation layer*)

Kod **podatkovnog sloja**, *developer* mora shvatiti kako najbolje strukturirati podatak, koji sustav za upravljanje bazama podataka treba koristiti i treba li koristiti eksterne izvore podataka.

U **aplikacijskom sloju** *developer* mora odlučiti koje će programske jezike, modele, protokole i aplikacijske arhitekture koristiti.

Developer se u **prezentacijskom sloju** fokusira na eksterne probleme, kao što su estetika i dizajn same aplikacije, HTML predlošci i stil.

Razvoj web aplikacija je kreativan proces koji vodi ka inovativnom proizvodu odnosno informacijskom sustavu. Taj proces se može podijeliti u više razvojnih faza kao što smo to naveli u poglavlju 2.3.1. Prema Casteleynu, Danielu, Dologi i Matera [3], osnovne faze razvoja aplikacija su:

- **Utvrđivanje zahtjeva**: ima za cilj razumjeti problem.
- **Dizajn**: cilj je planiranje rješenja problema.
- **Implementacija**: plan se prevodi u aplikacijski kod.

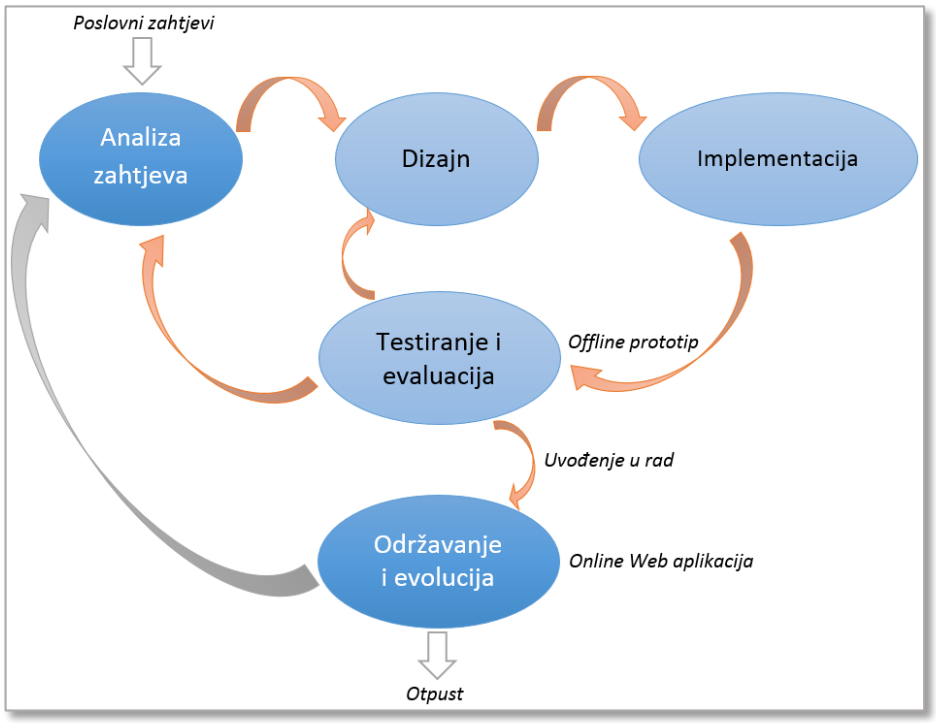
- **Testiranje i evaluacija**: cilj je identificiranje pogreške pri programiranju ili nedosljednosti između prikupljenih zahtjeva i njihove implementacije.
- **Objavljivanje**: donosi se rješenje za kupce (korisnike).
- **Održavanje**: cilj je praćenje (engl. *monitoring*) i održavanje sustava.
- **Evolucija**: ima za cilj poboljšati razvijeno rješenje pružajući nove zahtjeve.

4.3.5. Modeli razvoja u web inženjerstvu

4.3.5.1. Online Evolution Model

Online Evolution Model sastoji se od pet glavnih aktivnosti, a to su: analiza zahtjeva, dizajn, implementacija, testiranje i evaluacija, te održavanje i evolucija. Također se pojavljuje sedam prijelaznih stanja među aktivnostima, kao što možemo vidjeti na Slici 26.

Slika 26: Online Evolution Model



Izvor: vlastita izrada prema Casteleyn, S., Daniel, F., Dolog, P. i Matera, M. (2009): Engineering Web Applications, Springer-Verlag, Berlin

Prvi ciklus odnosno ciklus izgradnje i testiranja obuhvaća dizajn, implementaciju, te testiranje i evaluaciju, dok drugi ciklus povezuje održavanje i evoluciju sa analizom zahtjeva. Upravo je

taj drugi ciklus, koji je poznat i kao ciklus evolucije, karakterističan za ovaj model. Model predlaže izričitu vezu aktivnosti održavanja i evolucije sa aktivnošću analize zahtjeva. Navedena dva ciklusa zapravo odgovaraju dvjema fazama koje su svojstvene za moderne web aplikacije, a to su *offline* razvoj i *online* evolucija. Ciklusi se odvijaju različitim brzinama: prvi se ciklus odvija brzo, dok se drugi odvija sporije. Drugim riječima, aplikacije se moraju razvijati što je brže moguće, dok su zahtjevi podložni promjenama tijekom razvojne faze. Pri razvijanju web aplikacija, sve se više prakticira zamjena dokumentacija sa razvojnim prototipovima. Za razliku od tradicionalnih metoda, web aplikacije se plasiraju online što je ranije moguće, čak i ako nisu ispunjeni svi zahtjevi. Ciklus evolucije dobiva sve veći značaj, budući da se povratne informacije korisnika sve više uzimaju u obzir, pa na taj način evolucija konačno postaje prilika za poboljšanje aplikacije, a ne samo potreba za prilagođavanje aplikacije.

4.3.5.2. WebML i IFML

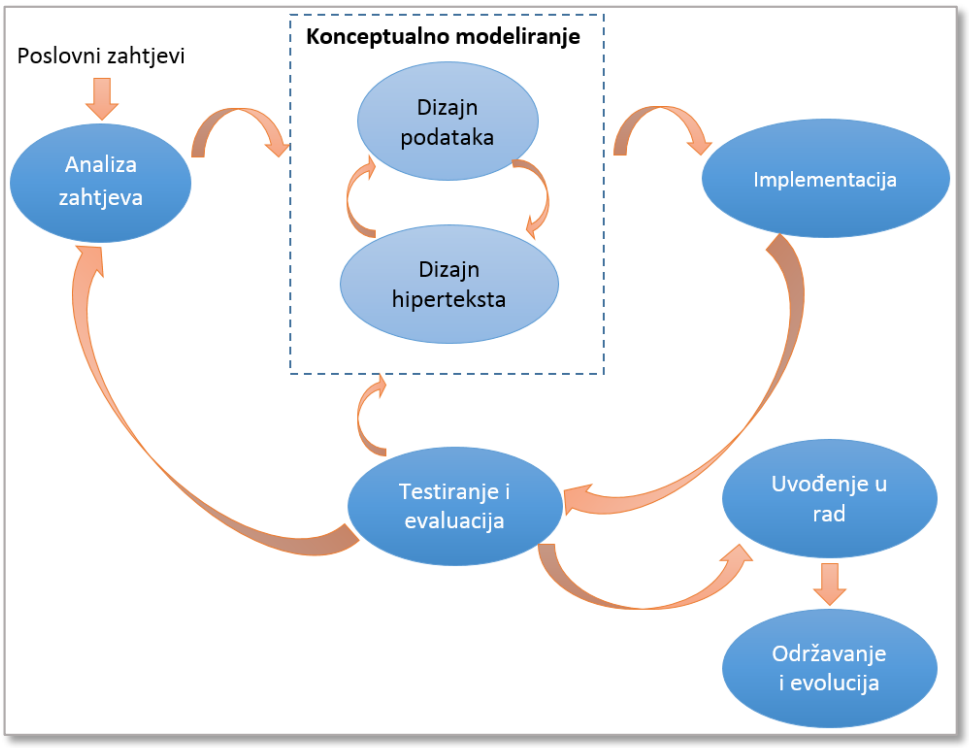
W3I3 Project (financiran od strane EU-a) proizveo je 1998. godine novi jezik za web modeliranje, zvan **Web Modeling Language - WebML**, te podržavajuće CASE okruženje koje nosi ime Toriisoft. Toriisoft je paket alata za dizajn koji obuhvaća cijeli životni ciklus web aplikacija te slijedi pristup web dizajna temeljen na modelima. WebML omogućuje visoku razinu opisa web stranice po određenim dimenzijama: sadržaju podataka (strukturalni model), stranicama koje ga tvore (kompozicijski model), topologije veza između stranica (navigacijski model), izgledu i grafičkim zahtjevima za prikazivanje stranice (prezentacijski model), te mogućnostima prilagodbe za isporuku sadržaja (personalizacijski model).

WebML je ponovno iskoristio postojeće konceptualne modele podataka i predložio novu notaciju za izražavanje navigacijskih i kompozicijskih značajki hipertekstualnih sučelja. Razvijeno je sveukupno četiri osnovnih verzija WebML-a: WebML 1, WebML 2, WebML 3 i WebML 4.

WebML jamči pristup temeljen na modelu pri izradi web stranica, što je ključni faktor za definiranje nove generacije CASE alata za izgradnju složenih web stranica, koje podržavaju naprednije značajke kao primjerice pristup sa više uređaja, personalizaciju te evolucijski menadžment. WebML omogućuje dizajnerima da izraze temeljne značajke jedne stranice na visokoj razini, ne obazirući se na detaljnu arhitekturu. WebML također podržava XML

sintaksu, koja se može učitati u softverske generatore za automatsku implementaciju web stranica.

Slika 27: WebML model



Izvor: vlastita izrada prema Rossi, G., Pastor, O., Scwabe, D. i Olsina, L., eds. (2008): Web Engineering - Modelling and Implementing Web Applications, Springer-Verlag, London

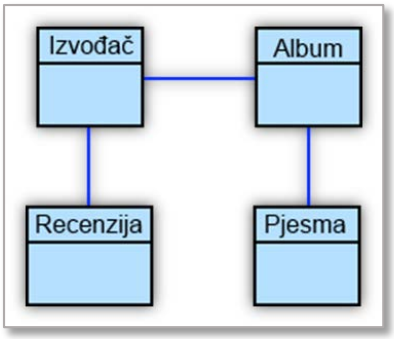
Kao što vidimo na Slici 27, WebML pristup razvoju web aplikacija sastoji se od različitih faza. Primjenjuje se na iterativan i inkrementalan način gdje se različite faze ponavljaju i poboljšavaju sve dok rezultati ne ispune aplikacijske zahtjeve. Životni ciklus proizvoda prolazi kroz nekoliko ciklusa gdje svaki ciklus proizvede prototip ili djelomičnu verziju aplikacije. Trenutna verzija aplikacije se testira i procjenjuje prilikom svake iteracije, pa se modificira radi usklađenja sa prikupljenim zahtjevima. Takav iterativan i inkrementalan razvoj je pogodan u kontekstu Weba gdje aplikaciju treba uvesti u rad što prije i gdje se zahtjevi konstantno mijenjaju tijekom razvojnog procesa.

Prema Ceriju, Fraternaliju i Bongi [16] specifikacija web stranice u WebML-u sastoji se od četiri perspektive:

- **Strukturalni model:** Definira sadržaj podataka stranice, u smislu relevantnih subjekata i odnosa. WebML nije tek još jedan jezik za modeliranje podataka, već je kompatibilan sa klasičnim notacijama.
- **Hipertekstualni model:** Opisuje jedan ili više hipertekstova koji mogu biti objavljeni na stranici. Svaki hipertekst definira pogled stranice koji se sastoji od dva pod-modela:
    - Kompozicijski model: određuje koje stranice tvore hipertekst, te koje sadržajne jedinice čine stranicu. Šest tipova sadržajnih jedinica mogu se koristiti za sastavljanje stranice: podatkovna jedinica, višepodatkovna jedinica, indeksna jedinica, skrolna jedinica, jedinica filtriranja, usmjerivačka jedinica. Podatkovne jedinice koriste se za objavljivanje informacija jednog objekta, dok preostale vrste jedinica predstavljaju alternativne mogućnosti za pregledavanje skupa objekata.
    - Navigacijski model: definira na koji su način stranice i sadržajne jedinice povezane pri formiranju hiperteksta. Za to služe usmjerene veze (engl. *links*) koje mogu biti ne-kontekstne, kada povezuju semantički neovisne stranice, ili kontekstne, kada sadržaj odredišnih jedinica usmjerenih veza ovisi o sadržaju izvornih jedinica.
- **Prezentacijski model:** Definira izgled stranica, neovisno o izlaznom uređaju i jeziku koji se koristio. Prezentacijske specifikacije su ili specifične ili generičke. U prvom slučaju one diktiraju prezentaciju određene stranice i uključuju eksplicitne reference na sadržaj stranice; dok se u drugom slučaju temelje na predefiniranim modelima neovisno o određenom sadržaju stranice.
- **Personalizacijski model:** Modelira korisnike i grupe korisnika u strukturnoj shemi u obliku predefiniranih entiteta koji se zovu Korisnik (engl. *User*) i Grupa (engl. *Group*). Značajke tih entiteta mogu se koristiti za pohranu određenog sadržaja specifičnih grupa ili pojedinca, kao npr. prijedlozi za kupnju, popis favorita, i sredstva za grafičku prilagodbu.

Osnovni elementi WebML **strukturalnog modela** su entiteti, koji predstavljaju spremnike podatkovnih elemenata; te odnosi (relacije), koje omogućuju semantičko povezivanje entiteta. Slika 28 prikazuje jednostavnu strukturnu shemu za prikaz informacija o albumu i izvođaču. Izvođači objavljuju albume sa pjesmama te imaju biografiju i recenziju njihova rada. Strukturna shema se sastoji od četiri entiteta (*Izvođač, Album, Recenzija, Pjesma*) i tri relacije (*Recenzija-Izvođač, Izvođač-Album, Album-Pjesma*).

Slika 28: Primjer strukturne sheme



Izvor: prilagođeno prema Ceri, S., Fraternali, P. i Bongio, A. (2000): Web Modeling Language (WebML): a modeling language for designing Web sites; Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy

Svrha **kompozicijskog modeliranja** je definiranje čvorova koji tvore hipertekst sadržan u web stranici. Točnije, kompozicijsko modeliranje specificira sadržaj jedinice.

WebML podržava šest vrsta jedinica za izradu hiperteksta:

- podatkovna jedinica: pokazuje informaciju o jednom objektu,
- višepodatkovna jedinica: pokazuje informaciju o skupu objekata,
- indeksna jedinica: pokazuje popis objekata, bez predstavljanja detaljnih informacija pojedinog objekta,
- skrolna jedinica: pokazuje naredbe za pristupanje elementima u poredanom skupu objekata,
- jedinica filtriranja: pokazuju polja za izmjenu za unos vrijednosti koje se koriste se za pretraživanje unutar skupa objekata koji zadovoljavaju uvjet,
- usmjerivačka jedinica: ne prikazuje informacije, ali se koristi za definiranje veze prema jednom objektu koji je semantički vezan na drugi objekt.

Jedinice i stranice ne postoje zasebno već se moraju spojiti kako bi oformile strukturu hiperteksta. Svrha **navigacijskog modeliranja** je specificiranje načina na koji se jedinice i stranice povezuju u hipertekst cjelinu. Tome služi pojam veze (engl. *link*), koje mogu biti:

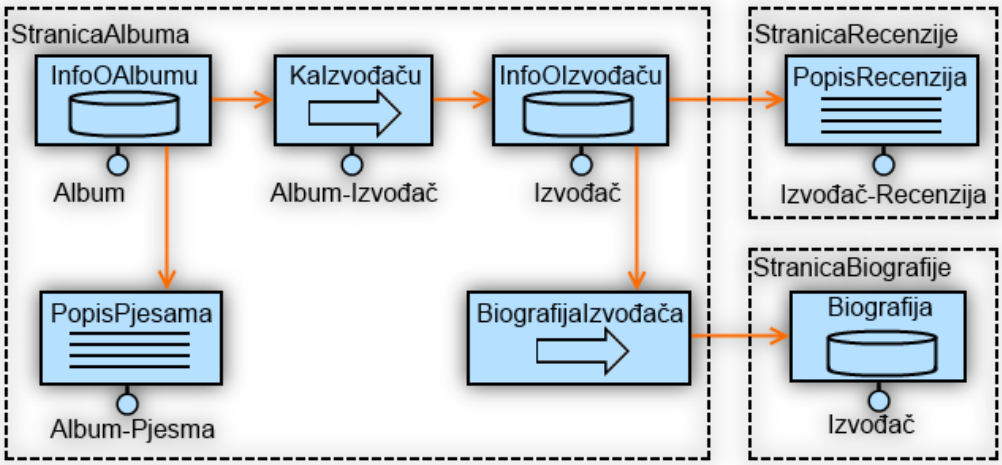
- Kontekstualne veze,
- Nekontekstualne veze.

Kontekstualne veze povezuju jedinice na način koherentan semantici koja je izražena strukturnom shemom aplikacije. Kontekstualna veza prenosi dio informacije (tzv. kontekst) od

izvorišne do odredište jedinice, kako bi se pomoću tog konteksta utvrdilo koji će se objekt ili skup objekata prikazati na odredišnoj jedinici.

Nekontekstualne veze povezuju stranice na potpuno slobodan način, tj. neovisno o jedinicama koje one sadrže i neovisno o semantičkim relacijama između strukturalnih koncepata sadržanih u tim jedinicama.

Slika 29: Primjer hipertekstualnog modela WebML-a



Izvor: prilagođeno prema Ceri, S., Fraternali, P. i Bongio, A. (2000): Web Modeling Language (WebML): a modeling language for designing Web sites; Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy

Slika 29 prikazuje dio specifikacije pogleda stranice (engl. *site view*), koristeći WebML. Izvođači objavljuju albume sa pjesmama te imaju biografije i recenzije njihova rada. Prilikom publikacije takvog sadržaja u obliku hiperteksta na Web, potrebno je specificirati kriterije za kompoziciju i navigaciju, tj. definirati pogled stranice. Hipertekst se sastoji od tri stranice (isprekidani pravokutnici). Svaka stranica sadrži skup jedinica (puni pravokutnici sa ikonama) koji će se zajedno prikazati na stranici. Primjerice, *StranicaAlbuma* prikuplja informacije o albumu i njegovom izvođaču. Ona sadrži podatkovnu jedinicu (*InfoOAlbumu*) koja prikazuje informacije o tom albumu, indeksnu jedinicu (*PopisPjesama*) sa popisom pjesama u albumu, te još jednu podatkovnu jedinicu (*InfoOIzvođaču*) sa osnovnim informacijama o izvođaču albuma. Jedinica *InfoOAlbumu* je povezana sa jedinicom *InfoOIzvođaču* pomoću posredne direktne jedinice (*KaIzvođaču*) što znači da se *InfoOAlbumu* odnosi na određenog izvođača koji je komponirao album prikazan na stranici. Jedinica *InfoOIzvođaču* sadrži jednu izlaznu vezu koja vodi na zasebnu stranicu koja sadrži popis recenzija, te jednu vezu na direktnu jedinicu koja pokazuje na biografiju izvođača, koja je pak prikazana na trećoj zasebnoj stranici.

WebML je naposljetku konvergiran u novi standard pod nazivom **Interaction Flow Modeling Language - IFML** kojeg je 2014. godine usvojila grupa OMG. IFML je razvijen 2013. godine te je ožujku 2015. godine [31] objavljena službena verzija 1.0.

IFML je neovisan o platformi odnosno podržava opis grafičkog korisničkog sučelja za aplikacije kojima se može pristupiti sa različitih sustava, poput osobnih računala, prijenosnih računala, pametnih telefona i tableta. Usmjeren je na strukturu i ponašanje aplikacije sa aspekta korisnika. IFML obuhvaća različite aspekte korisničkog sučelja:

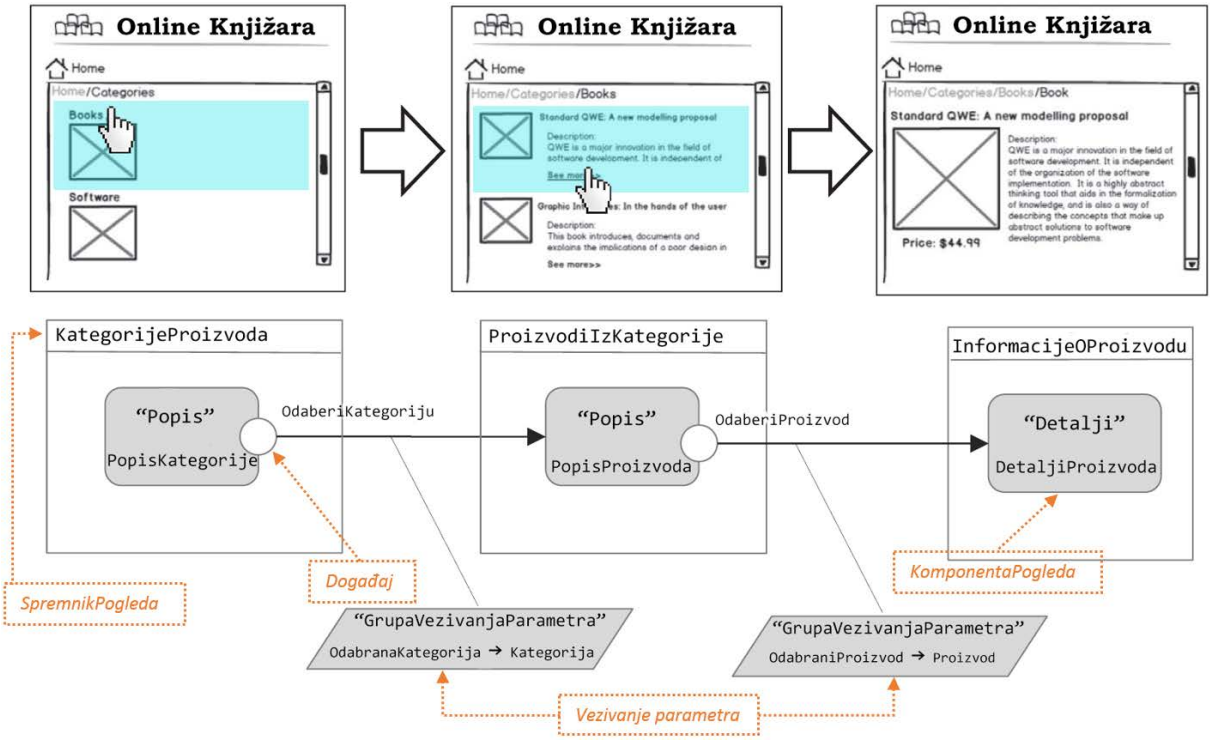
- **Struktura pogleda (engl. *View structure*):** obuhvaća opću organizaciju sučelja uz pomoć tzv. *SpremnikaPogleda* (engl. *ViewContainers*). *SpremniciPogleda* (engl. *ViewContainers*) su elementi sučelja koji obuhvaćaju komponente za prikaz sadržaja i podrške interakcije.
- **Sadržaj pogleda (engl. *View content*):** određuje što zapravo *SpremniciPogleda* sadrže s aspekta *KomponentaPogleda* (engl. *ViewComponents*). *KomponentePogleda* (engl. *ViewComponents*) su elementi za prikaz sadržaja i unos podataka.
- **Događaji (engl. *Events*):** predstavljaju pojave, izazvane od strane korisnika ili same aplikacije, koje utječu na stanje korisničkog sučelja.
- **Prijelazi događaja (engl. *Event transitions*):** određuju posljedice događaja na korisničkom sučelju.
- **Vezivanje parametra (engl. *Parameter binding*):** pojašnjava ovisnost inputa i outputa između *SpremnikaPogleda, KomponentiPogleda* i određenih radnji.

Za razliku od UML-a, koji se oslanja na više vrsta dijagrama, IFML sažima sve prethodne navedene aspekte unutar jedne vrste dijagrama pod nazivom **Interaction Flow Diagram**.

Na Slici 30 vidimo jednostavan primjer IFML modela sučelja za online knjižaru. Struktura pogleda (engl. *View structure*) sastoji se od tri *SpremnikaPogleda*: *KategorijeProizvoda, ProizvodiIzKategorije* i *InformacijeOProizvodu*. Model također prikazuje sadržaj svakog *SpremnikaPogleda.* Primjerice, spremnik *KategorijeProizvoda* sadrži jednu *KomponentuPogleda* pod nazivom *PopisKategorije*. Nadalje, događaji su prikazani kao obični krugovi. U slučaju događaja *OdaberiKategoriju,* on pokazuje kako je komponenta *PopisKategorije* interaktivna. Drugim riječima, korisnik može (u web pregledniku) odabrati jednu od kategorija kako bi pristupio popisu proizvoda u izabranoj kategoriji. Ovisnost inputa i outputa između komponenti *PopisKategorije* i *PopisProizvoda* je predstavljena putem

vezivanja parametra. Vrijednost parametra *OdabranaKategorija* povezana je sa vrijednošću inputa parametra *Kategorija* koja je zahtijevana radi izračuna komponente *PopisProizvoda*.

Slika 30: Primjer IFML modela sučelja (za online knjižaru)



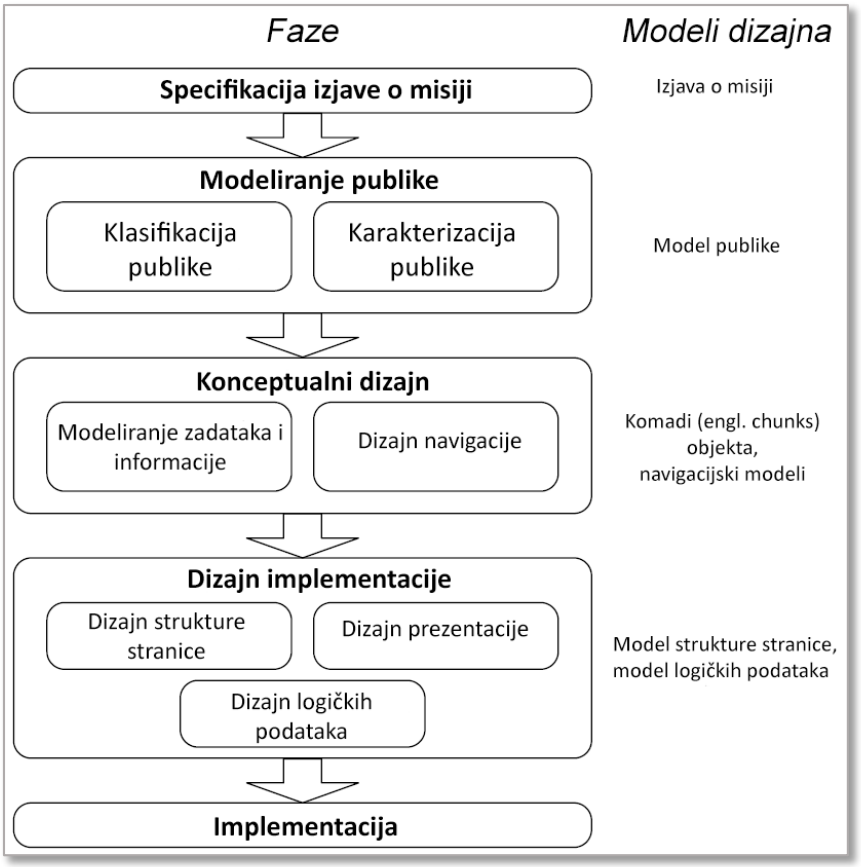
Izvor: prilagođeno prema Brambilla, M. i Fraternali, P. (2015): Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML, Morgan Kaufmann, Waltham MA, USA

4.3.5.3. WSDM

Metoda dizajna web stranice (engl. *Web Site Design Method - WSDM*) predstavlja jednu od prvih metoda web dizajna. U međuvremenu, metoda se uvelike razvila od svojih početaka, pa tako omogućuje razvoj kako tradicionalnih web aplikacija, tako i semantičkih web aplikacija, te je metoda danas poznata kao *Web Semantics Design Methods*.

WSDM je metodologija koja omogućuje *developerima* konstruiranje modela za opisivanje web stranica odnosno aplikacija iz različitih perspektiva, te pruža semantički način razvijanja web aplikacija. Također prepoznaje važnost korisnika (tzv. publike), pa pribjegava analizi različitih vrsta korisnika te njihovih zahtjeva i karakteristika. WSDM ima pet faza dizajna sa pripadajućim modelima dizajna kao što možemo vidjeti na slijedećoj Slici 31.

Slika 31: WSDM model



Izvor: prilagođeno prema Casteleyn, S., Daniel, F., Dolog, P. i Matera, M. (2009): Engineering Web Applications, Springer-Verlag, Berlin

U prvoj fazi tj. u **specifikaciji izjave o misiji** daje se izjava o misiji za web stranicu. U izjavi se navodi svrha, predmet i ciljani korisnici web stranice. U daljnjim fazama omogućit će se odlučivanje koje informacije ili funkcionalnosti uključiti, kako strukturirati i naposljetku prezentirati informaciju. Nakon procesa dizajna, izjava će poslužiti kao provjera jesu li navedeni ciljevi ispunjeni. Izjava se piše u prirodnom jeziku.

Tijekom druge faze, **modeliranja publike**, WSDM uzima u obzir činjenicu kako različiti korisnici mogu imati različite potrebe i zahtjeve, pa je samim time potrebna posebna podrška prilagođena njihovim zahtjevima. Radi toga su ciljani korisnici podijeljeni u tzv. klase publike koje izvršavaju dvije pod-faze: *Klasifikacija publike* i *Karakterizacija publike*. Tijekom Klasifikacije publike različite vrste korisnika se identificiraju i grupiraju u korisnike s istim funkcionalnim i informacijskim zahtjevima. Za svaku klasu publike navedeni su i zahtjevi navigacije i upotrebljivosti, a to se izvršava tijekom Karakterizacije publike.

Treća faza odnosno **konceptualni dizajn** koristi se za određivanje informacija, funkcionalnosti i strukture web sustava na konceptualnoj razini. Konceptualni dizajn je podijeljen u dvije pod-faze. Tijekom *modeliranja zadataka i informacije*, dizajner modelira zadatke koje moraju obaviti različite klase publike. Svrha ove pod-faze je detaljno modeliranje različitih zadataka kako bi ih svaki član klase publike mogao izvršiti, te formalno opisivanje podataka i funkcionalnosti. Zadaci, koje članovi klasa moraju biti u stanju obaviti, temelje se na zahtjevima koji su navedeni u prethodnoj fazi tijekom Klasifikacije publike. Svaki model zadatka sastoji se od dekompozicije zadatka kako bi se ispunio određeni zahtjev. Nakon što je završena dekompozicija, svaki se osnovni zadatak modelira pomoću tzv. komada (engl. *chunks*) objekta. Komad objekta formalno opisuje informaciju kako bi se izvršio zadatak. Cilj *dizajna navigacije* je definiranje konceptualne strukture web sustava i modeliranje načina kako se članovi različitih klasa publike mogu kretati kroz Web sustav i obavljati svoje zadatke.

U četvrtoj fazi, **dizajnu implementacije**, konceptualni modeli se nadopunjuju sa svim potrebnim detaljima radi pripreme za stvarnu implementaciju. U prvoj pod-fazi konceptualna navigacijska struktura se preslikava na stvarne web stranice. Tijekom druge pod-faze, definira se opći izgled i dojam web stranice. Dizajner je zaslužan za pozicioniranje konkretnih elemenata sučelja. To sve rezultira modelima stranica (engl. *page models*). U trećoj pod-fazi specificiraju se izvori podataka (engl. *data source*) te mapiranje između konceptualnih modela podataka i izvora podataka.

Posljednja faza je faza **implementacije** koja može biti izvedena i automatski pomoću informacija prikupljene tijekom prethodnih faza.

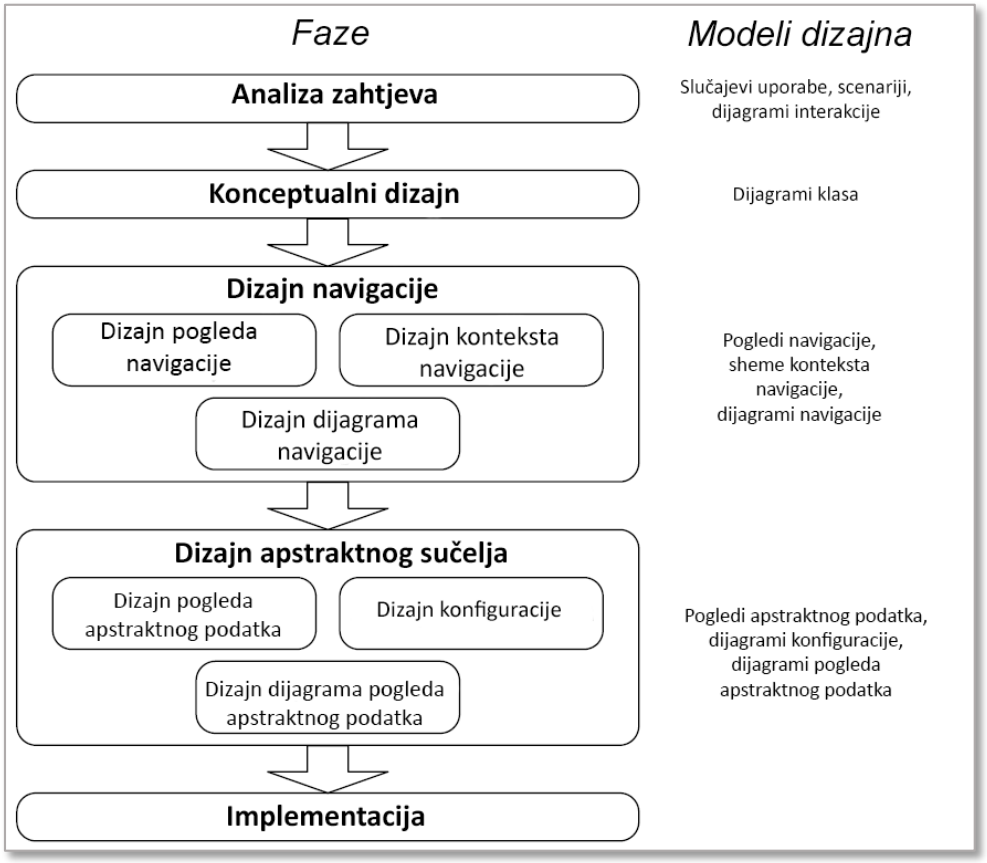
4.3.5.4. OOHDM

Object-Oriented Hypermedia Design Method - OOHDM je jedna od prvih usvojenih metoda za razvoj web aplikacija. Kao što sam naziv govori svoje korijene vuče iz hipermedijske domene te se usredotočuje na pomaganje razvoja aplikacija koji uključuju hipertekstualne i hipermedijske[16] paradigme. OOHDM metoda sadrži objektno-orijentirane apstrakcije za analizu i dizajn informacijsko intenzivnih web aplikacija. Poput WebML-a i WSDM-a, također

[16] Hipermedija je nadogradnja hiperteksta koji podržava povezivanje grafičkih, zvučnih i video sadržaja sa već postojećim hipertekstom.

pruža metodologiju koja vodi *developera* kroz različite aktivnosti u razvoju web aplikacija. Na Slici 32 možemo vidjeti faze OOHDM-a zajedno sa modelima dizajna proizašle iz njih.

Slika 32: OOHDM model



Izvor: prilagođeno prema Casteleyn, S., Daniel, F., Dolog, P. i Matera, M. (2009): Engineering Web Applications, Springer-Verlag, Berlin

Glavni cilj prve faze tj. faze **analize zahtjeva** je otkrivanje i razumijevanje funkcionalnih i nefunkcionalnih zahtjeva web aplikacije. Funkcionalni zahtjevi se otkrivaju uz pomoć slučajeva uporabe i zainteresiranih strana web aplikacije. Slučajevi uporabe se dodatno proširuju scenarijima. OOHDM također koristi dijagrame interakcije koje služe za interakciju korisnika i aplikacije pri izvršavanju određenih slučajeva uporabe.

Druga faza se bavi dizajnom strukture informacija kako bi se prikazao sadržaj koji je zastupljen u web aplikaciji. Tijekom ove faze upotrebljavaju se objektno-orijentirani principi, a kao rezultat je konstruiran dijagram klasa. **Konceptualni dizajn** je odvojen od ostalih aktivnosti i bavi se samo klasama aplikacijske domene bez povezivanja daljnjih aplikacijskih rješenja za pregledavanje i organiziranje sadržaja.

Treća se faza (**dizajn navigacije**) bavi strukturama navigacije koja podržava informacije koje otkriva korisnik u web aplikaciji. Daje se podrška korisnicima kako bi se mogli orijentirati u informacijskom „hiperprostoru“. Pogledi struktura informacija mogu biti različiti za različite korisnike. To se najbolje odražava u shemama konteksta navigacije gdje su zajednički pogledi grupirani u jedan kontekst. Modeli navigacije su povezani sa konceptualnim modelima te koriste temeljne koncepte iz konceptualnih modela radi dobivanja pravih perspektiva informacijske strukture. Iz jednog konceptualnog modela moguće je izgraditi različite poglede navigacije i sheme za različite svrhe ili korisnike.

Dizajn apstraktnog sučelja slijedi objektno-orijentirane principe i usredotočuje se na objekte te definira kako bi poglede (engl. *site view*) navigacije trebali prikazati i proširiti s daljnjim elementima interakcije (npr. linkovi). Pogledi apstraktnih podataka slijede isti princip poput pogleda navigacije. Također sadrže interaktivni aspekt, pa su radi toga pogodne za opisivanje modernih interaktivnih web aplikacija.

U završnoj fazi razvojni tim mora odlučiti kako će transformirati rezultate prethodnih faza u **implementaciju**. Razvojni tim donosi odluku o arhitekturi, sustavu za upravljanjem bazom podataka i web poslužitelju.

5. ZAKLJUČAK

Iako se softversko inženjerstvo kao znanstvena disciplina jako razvilo, u kojem imamo za birati pregršt metodologija i alata, ovisno o svrsi za koju koristimo, ipak nije polučilo ekstreman razvoj kakav bi *developeri* i ostali IT stručnjaci voljeli imati. Korištenjem suvremenih trendova razvoja aplikacija, poput agilnih metoda, ne dolazi nužno do poboljšavanja funkcija i usluga koje bi *developeri* trebali pružiti korisnicima. Agilne metode su donijele novinu u vidu bržeg razvoja aplikacija odnosno softvera, te smanjivanja ili potpunog nestanka administracije.

Budući da agilne metode imaju svoje nedostatke i propuste, tradicionalne metode razvoja još su i dan danas u primjeni ili kao zasebna metodologija ili kao nadopuna suvremenim metodologijama. Pri razvoju manjih projekata, kao što je to slučaj sa web aplikacijama, agilne metode se mogu dosta efikasno primjenjivati. No, za veće projekte, poput kompleksnih poslovnih informacijskih sustava (npr. u zdravstvu i financijama), tradicionalne metode su puno sigurniji izbor ukoliko se želi razviti stabilan i kvalitetan sustav sa dobrom popratnom dokumentacijom.

Jedan od najvećih problema koji se javlja i kod tradicionalnih i kod suvremenih trendova je naoko bezazlena aktivnost – a to je definiranje korisničkih zahtjeva. Korisnici vrlo često mijenjaju svoje zahtjeve i često ne znaju izraziti svoje potrebe sistem analitičaru, pa to dovodi do raznih komplikacija. U tom su pogledu suvremene metode u prednosti, budući da se mogu brzo prilagoditi promjenama i proširivanju zahtjeva. U tradicionalnim metodama vrlo je teško vraćati se u ranije faze razvoja radi promjene zahtjeva. No, za sada, ni jedna metodologija nije dala definitivno rješenje za taj gorući problem u softverskom inženjerstvu, što rezultira malim brojem uspješnih projekata. Stoga, ni ne čudi podatak, gdje je prema jednom istraživanju iz 2013. godine [36] 58% ispitanika izjavilo kako se softverska rješenja isporučuju na vrijeme, 36% ih je procijenilo da ne prelaze budžet, a 14% je procijenilo da su softveri izrađeni prema specifikaciji. Neovisno o svemu, nesumnjivo je da će se metode razvoja informacijskih sustava i samo softversko inženjerstvo i dalje poboljšavati i nadograđivati, što dokazuje i činjenica kako određene međunarodne udruge (poput OMG-a) neprestano izbacuju nove verzije alata, standarda i metodologija.

Diplomski rad je, između ostalog, uz pojašnjenje i usporedbu raznih metodologija, pokušao dati sažet i jasan opis dugotrajnog i kompleksnog procesa razvoja informacijskih

sustava, koji su popraćeni raznim grafičkim primjerima (npr. UML) radi lakšeg razumijevanja opsežne teme. Prikazan je zapravo cijeli proces prikupljanja zahtjeva i modeliranja, odnosno sve one aktivnosti u kojima sudjeluju sistem analitičari do onog trenutka kad programeri započnu sa svojim dijelom posla, a to je izgradnja i implementacija samog softvera. Rad je također i vrlo aktualan, budući da je obuhvatio neke nove trendove razvoja u web inženjerstvu, te su se radi lakšeg razumijevanja najprije obradili osnovni koncepti web aplikacija i njezine popratne tehnologije. Kao aktualni modeli razvoja posebno se ističu WebML tj. današnji IFML koji predstavljaju nove jezike web modeliranja.

LITERATURA

Knjige:

[1] Brambilla, M. i Fraternali, P. (2015): *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*, Morgan Kaufmann, Waltham MA, USA

[2] Braude, E. J. i Bernstein, M. E. (2010): *Software Engineering - Modern Approaches (2nd Edition),* John Wiley and Sons, New York NY, USA

[3] Casteleyn, S., Daniel, F., Dolog, P. i Matera, M. (2009): *Engineering Web Applications*, Springer-Verlag, Berlin

[4] Cockburn, A. (2000): *Agile Software Development*, Cockburn Highsmith Series Editors

[5] Čerić, V. i Varga, M., ur. (2004): *Informacijska tehnologija u poslovanju*, Element, Zagreb

[6] Dennis, A., Wixom, B. H. i Tegarden, D. (2012): *Systems Analysis and Design with UML Version 2.0 (4th Edition)*, John Wiley and Sons, New York NY, USA

[7] Kappel, G., Proll, B., Reich, S. i Retschitzegger, W., eds. (2006): *Web Engineering - The Discipline of Systematic Development of Web Applications*, John Wiley and Sons, New York NY, USA

[8] Pavlić, M. (2009): *Informacijski sustavi*, Odjel za informatiku Sveučilišta u Rijeci, Rijeka

[9] Rossi, G., Pastor, O., Scwabe, D. i Olsina, L., eds. (2008): *Web Engineering - Modelling and Implementing Web Applications*, Springer-Verlag, London

[10] Shelly, G.B. i Rosenblatt, H. J. (2012): *System Analysis and Design (9th Edition)*, Cengage Learning, Boston MA, USA

[11] Sommerville, I. (2010): *Software Engineering (9th Edition)*, Pearson Education Inc, Boston MA, USA

[12] Šehanović, J., Hutinski, Ž. i Žugaj, M. (2002): *Informatika za ekonomiste*, Sveučilište u Rijeci - FET dr. Mijo Mirković u Puli, Pula

[13] Varga, M. (1994): *Baze podataka - Konceptualno, logičko i fizičko modeliranje*, Društvo za razvoj informacijske pismenosti, Zagreb

[14] Whitten, J. L. i Bentley, L. D. (2007): *Systems Analysis and Design Methods (7th Edition)*, McGraw-Hill/Irwin, New York NY, USA

Ostalo:

[15] Abrahamsson, P., Salo, O. i Ronkainen, J. (2002): *Agile software development methods - Review and analysis*, VTT Electronics, University of Oulu, Finland

[16] Ceri, S., Fraternali, P. i Bongio, A. (2000): *Web Modeling Language (WebML): a modeling language for designing Web sites*, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy

[17] Ferdelja, I. (2006): *Web servisi - Seminar*, FER, Zagreb

[18] Kalafatić, Z. (2012): *Skriptni jezici - Materijali za predavanja*, FER, Zagreb

[19] Manger, R. (2011): *Baze podataka - Skripta (drugo izdanje)*, PMF, Zagreb

[20] Manger, R. (2013): *Softversko inženjerstvo - Skripta (nadopunjeno drugo izdanje)*, PMF, Zagreb

Internet:

[21] Agile Software Development: A gentle introduction, <http://www.agile-process.org/>, [27. ožujka 2015.]

[22] Baranović, M. i Zakošek, S. (2012): *Baze podataka - Predavanja*, FER, Zagreb, <https://www.fer.unizg.hr/_download/repository/Baze_podataka_Preddiplomski_Predavanja.pdf>, [2. veljače 2015.]

[23] Baseline - IT's 2015 Top Investment Priorities, <http://www.baselinemag.com/it-management/slideshows/its-2015-top-investment-priorities.html>, [3. svibnja 2015.]

[24] Classification of UML 2.4 Diagrams, <http://www.uml-diagrams.org/uml-24-diagrams.html>, [15. ožujka 2015.]

[25] CNN Money - Mobile apps overtake PC Internet usage in U.S., <http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet/>, [3. svibnja 2015.]

[26] Feature Driven Development (FDD) and Agile Modeling, <http://www.agilemodeling.com/essays/fdd.htm>, [6. travnja 2015.]

[27] Gartner Identifies the Top 10 Strategic Technologies for 2012, <http://www.gartner.com/newsroom/id/1826214>, [3. svibnja 2015.]

[28] Gartner Identifies the Top 10 Strategic Technology Trends for 2013, <http://www.gartner.com/newsroom/id/2209615>, [3. svibnja 2015.]

[29] Gartner Identifies the Top 10 Strategic Technology Trends for 2014, <http://www.gartner.com/newsroom/id/2603623>, [3. svibnja 2015.]

[30] Gartner Identifies the Top 10 Strategic Technology Trends for 2015, <http://www.gartner.com/newsroom/id/2867917>, [3. svibnja 2015.]

[31] IFML: The Interaction Flow Modeling Language, <http://www.ifml.org/>, [27. svibnja 2015.]

[32] Internet Live Stats - Internet Users, <http://www.internetlivestats.com/internet-users/>, [3. svibnja 2015.]

[33] Načela na kojima se zasniva proglas o agilnom razvoju softvera, <http://www.agilemanifesto.org/iso/hr/principles.html>, [3. travnja 2015.]

[34] OMG - OMG Formal Versions of UML, <http://www.omg.org/spec/UML/>, [19. ožujka 2015.]

[35] SourceMaking, UML - Introduction, <https://sourcemaking.com/uml/introduction>, [15. ožujka 2015.]

[36] There is No Common Definition of Software Development Success, <http://scottambler.com/no-common-definition-of-success.html>, [17. lipnja 2015.]

[37] Vidi.hr - Hrvatske statistike posjedovanja mobitela, računala, automobila, pasa..., <http://www.vidi.hr/Lifestyle/Business-3.0/Hrvatske-statistike-posjedovanja-mobitela-racunala-automobila-pasa>, [3. svibnja 2015.]

[38] W3Schools, <http://www.w3schools.com>, [24. travnja 2015.]

Popis slika, tablica i grafikona

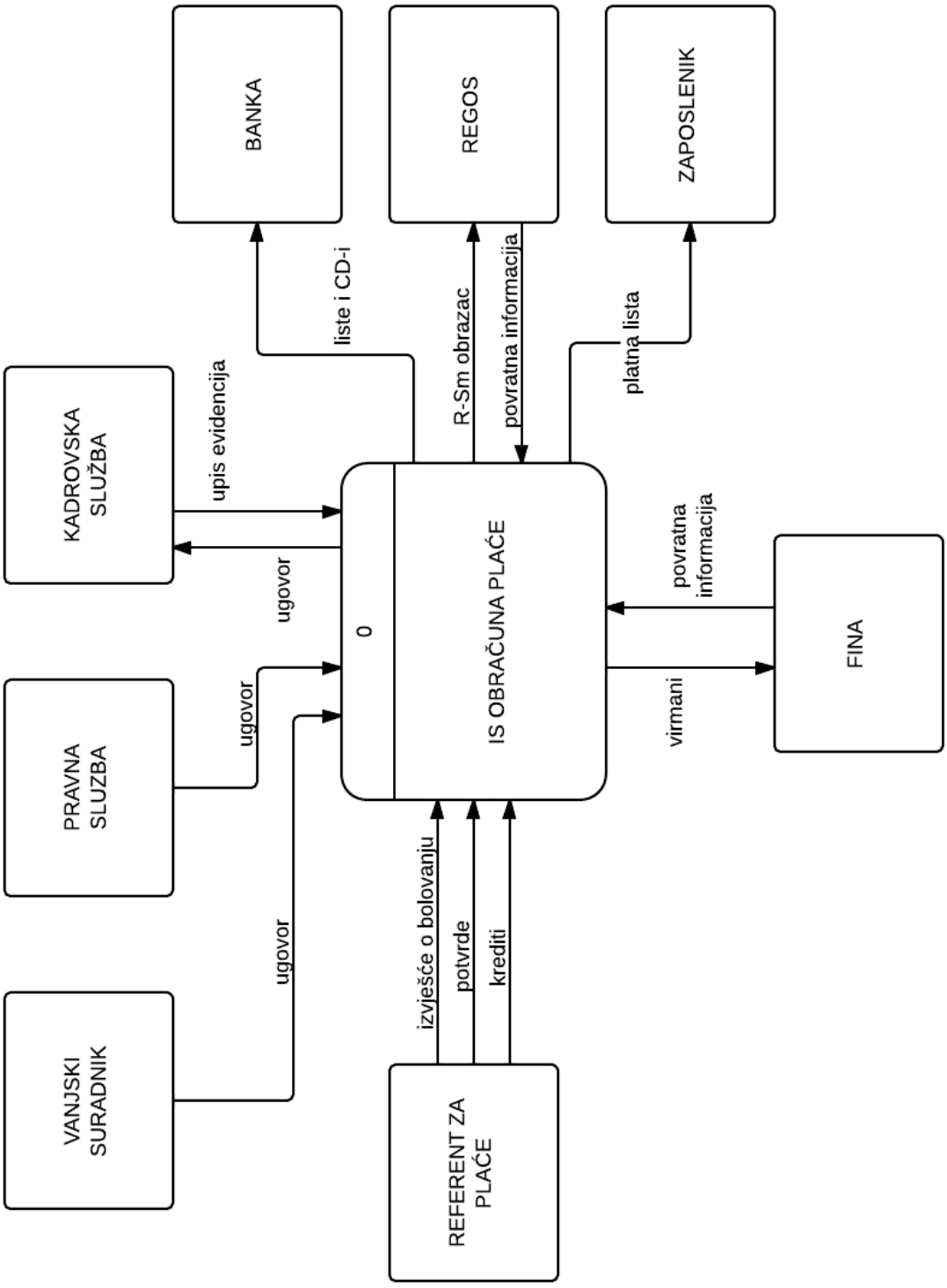
Slike:

Tablice:

Grafovi:

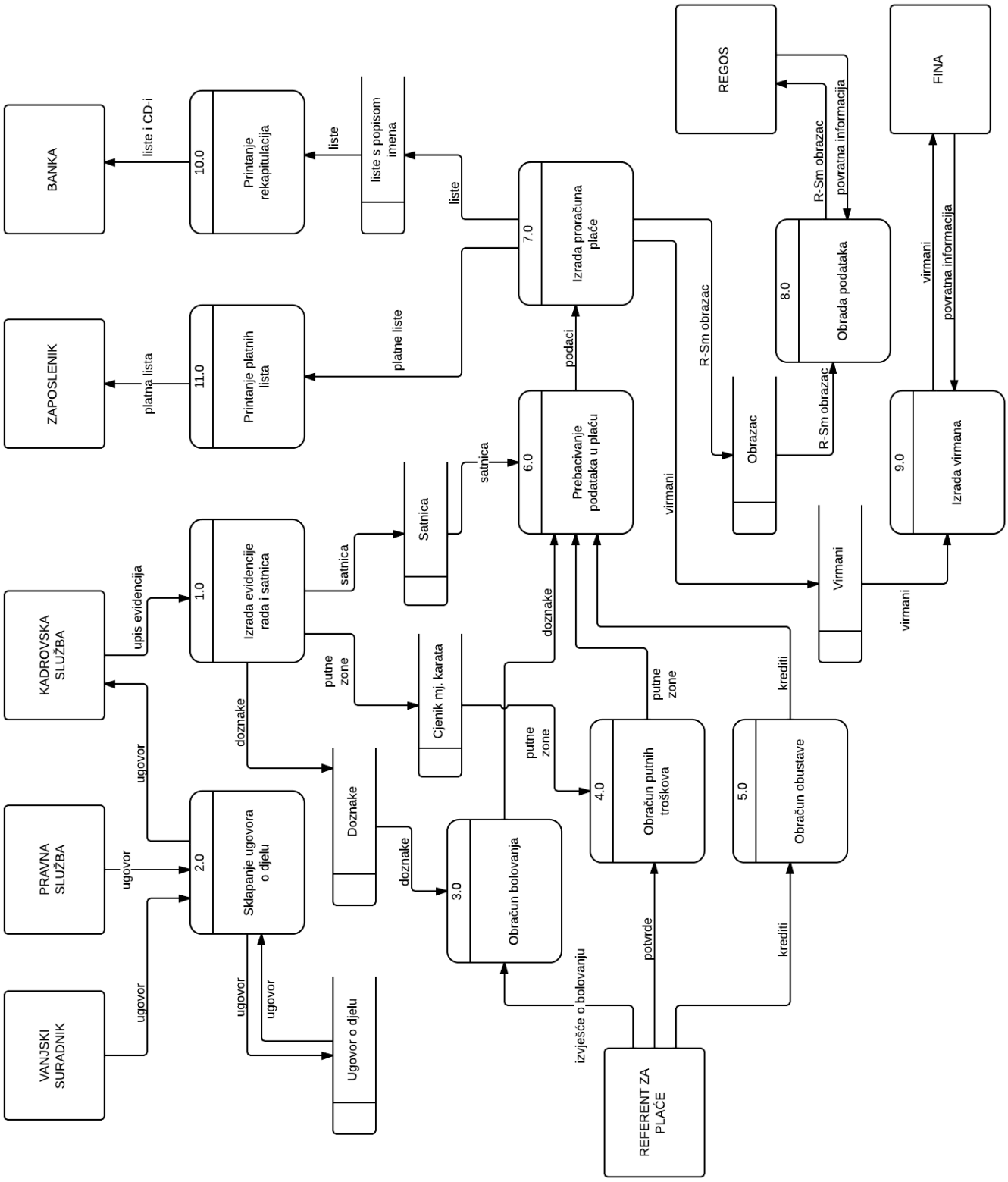
Prilozi

Prilog 1: IS obračuna plaće - Kontekstualni dijagram



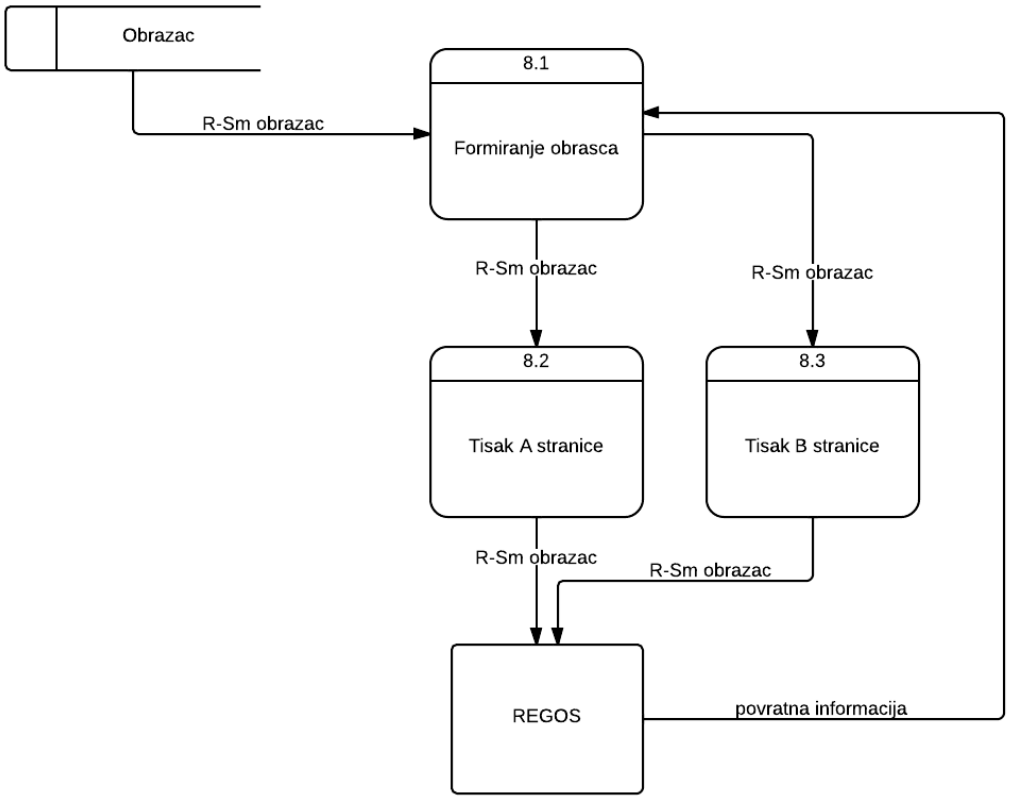
Izvor: vlastita izrada

Prilog 2: IS obračuna plaće - DTP prve razine



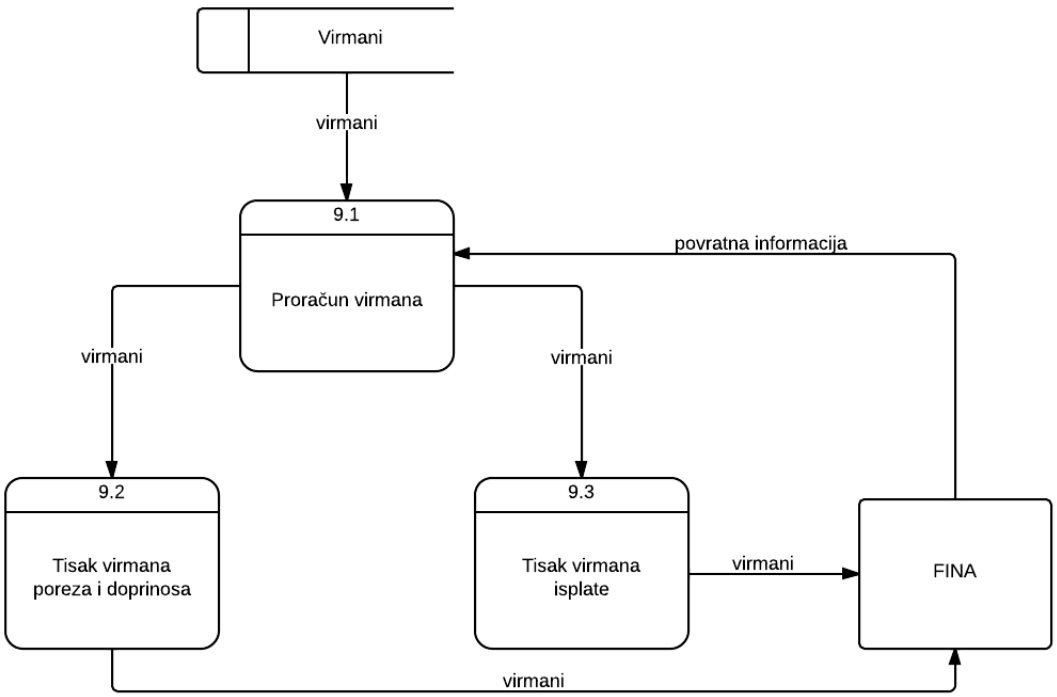
Izvor: vlastita izrada

Prilog 3: IS obračuna plaće - DTP druge razine (Proces 8: Obrada podataka)



Izvor: vlastita izrada

Prilog 4: IS obračuna plaće - DTP druge razine (Proces 9: Izrada virmana)



Izvor: vlastita izrada

Sažetak

Suvremeni trendovi razvoja informacijskih sustava

Zbog ubrzanog načina života, teško je zamisliti život bez informacijskih sustava u bilo kojem segmentu ljudskog djelovanja. Razvoj informacijskih sustava je vrlo složen i dugotrajan posao. Tijekom njihovog razvoja koriste se različite metode i alati za oblikovanje i izgradnju funkcionalnih sustava. U odabiru između raznih metodologija pomaže nam softversko inženjerstvo, koji ima za cilj odabrati najbolje metode i alate s kojima ćemo biti u mogućnosti izraditi što bolji i kvalitetniji softver, koji će odgovarati zahtjevima korisnika. Izgradnja informacijskog sustava, bilo da se radi o tradicionalnom ili suvremenom pristupu, odvija se u određenim fazama razvoja.

Kao jedan od najpopularnijih pristupa (objektno-orijentiranom) modeliranju, javlja se UML pomoću kojeg možemo modelirati projekte od samih početaka (faze analize) pa sve do implementacije sustava.

Agilni razvoj informacijskih sustava pojavio se iz potrebe da se ubrza dugotrajan proces razvoja softvera i kako bi se izbjegla (nepotrebna) dokumentacija. Pojavom novih i suvremenih informacijskih sustava, pojavio se i novi inženjerski pristup razvoju aplikacija pod nazivom web inženjerstvo, koji je specifičan po tome što se Web koristi i kao razvojna i kao korisnička platforma.

Zbog povećane potrebe za razvojem web aplikacija, pojavili su se i novi jezici specifični za web modeliranje, od kojih se najviše ističu WebML i njegov nasljednik IFML.

Ključne riječi: *razvoj informacijskih sustava, razvoj web aplikacija, modeliranje, UML, agilni razvoj, web i mobilne aplikacije, softversko inženjerstvo, zahtjevi*

Summary

Information Systems Development Trends

Because of today's rapid lifestyle, it is hard to imagine life without information systems in any segment of human activity. The development of information systems is a very complex and time-consuming task. During their development, various methods and tools are used to design and build a functional system. Software engineering helps us in choosing between the various methodologies, which goal is to select the best methods and tools with which we will be able to make a better and high quality software, to suit user requirements. Building an information system, be it a traditional or a modern approach, takes place at certain phases of development.

UML is one of the most popular (object-oriented) modeling approach, with which we can model projects from the very beginning (analysis phase) to the implementation of the system.

Agile information systems development emerged from the need to speed up the time-consuming process of software development and in order to avoid (unnecessary) documentation. With the appearance of new and modern information systems, there is also the new engineering approach for developing applications called Web Engineering, which is specific in that it uses the Web as a development platform and as a user platform.

Due to the increased need for the development of Web applications, new Web modeling languages have appeared, of which the most prominent are WebML and his successor IFML.

Keywords: *information system development, web application development, modeling, UML, agile development, web and mobile applications, software engineering, requirements*