3. Rational Unified Process

Rational Unified Process (u nastavku RUP) predstavlja objektno orijentisani proces za izradu softvera. Ovaj proces je zasnovan na pristupu baziranom na disciplinama, u okviru kojih se vrši raspodela poslova i odgovornosti na članove razvojnog tima i ostale učesnike u procesu razvoja softvera. Osnovna namena RUP-a jeste proizvodnja visoko kvalitetnog softvera, kao krajnjeg rezultata projekta, koji zadovoljava potrebe korisnika, a u okviru planiranog budžeta i vremena.

RUP je framework (radni okvir) koji je moguće prilagođavati potrebama organizacije koja ga usvaja. On sadrži skup dobro definisanih preporuka i uputstava za uspešan razvoj softverskog proizvoda. Upravo radi toga se kod navođenja definicije RUP-a izbegava izraz metodologija, koji predstavlja znatno čvršći set pravila od navedenog izraza framework. Međutim pojedinačnim podešavanjem RUP-a organizacija može da kreira svoj metodološki okvir, a u okviru njega i čvrsta pravila izgrađena na osnovu datih preporuka i uputstava.

U izgradnji informacionog sistema, RUP preporučuje korišćenje raznih metoda i tehnika, ali su svakako najdominantnije tehnike modelovanja korišćenjem Unified Modeling Language-a (u nastavku UML) tj. jedinstvenog jezika za modelovanje. Za RUP se čak može reći da predstavlja uputstvo za korišćenje UML-a.

RUP je inicijalno zamišljen za razvoj velikih projekata, ali je svoju primenu našao i u srednjim i malim softverskim projektima. Softverski timovi, naročito veliki, znatno unapređuju svoju produktivnost korišćenjem RUP-a. RUP omogućuje svakom članu razvojnog tima lak uvid u bazu znanja, zasnovanu na uputstvima, templejtima i uputstvima za korišćenje alata, što predstavlja podršku u svim kritičnim razvojnim aktivnostima. To doprinosi da svi članovi razvojnog tima, bez obzira na aktivnosti koje obavljaju, koriste zajedničku metodologiju, jezik i pogled na to kako treba razvijati softverski proizvod.

3.1. Principi na kojima je zasnovan RUP

Principi na kojima je zasnovan RUP, omogućuju razvoj kvalitetnih softverskih proizvoda i postavljanje metodoloških koraka kojima će se realizovati kompletne preporuke i uputstva zasnovane na dole navedenim principima.

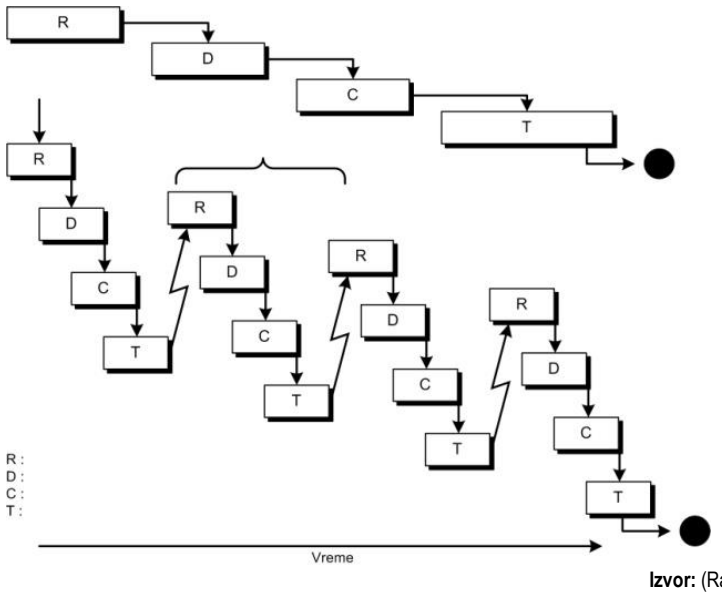
3.1.1. Iterativan razvoj

Klasični proces razvoja softvera se zasniva na modelu vodopada, koji je ranije opisan. Alternativa modelu vodopada jeste iterativno-inkrementalni model razvoja softvera. Korišćenjem ovoga modela dolazi se do relativno brze realizacije početne verzije softvera koja se razvija dodatnim iteracijama. Takođe, integrisanje i testiranje softvera se obavlja češće, te se na taj način postiže smanjenje rizika. Što je razlika između trenutka otkrivanja greške i trenutka nastanka greške veća, to je njeno otkrivanje i otklanjanje teže i skuplje. Kao što se vidi na slici 3.1., iterativni proces se sastoji iz više sekvencijalnih procesa zasnovanih na modelu vodopada. Time je vremenska dimenzija izmeštena u odnosu na sadržajnu dimenziju, pa je vreme između potencijalnog nastanka i otkrivanja grešaka značajno skraćeno.

Kao prednosti iterativnog razvoja mogu se navesti sledeće:

- Greške učinjene u razvoju se ranije identifikuju i na njih je moguće uspešnije reagovati;
- Omogućava se korisnički feedback;
- Razvojni tim je primoran da se fokusira na ona pitanja koja su najvažnija za projekat;
- Kontinuirano i iterativno testiranje pruža objektivan uvid u status razvoja;
- Nepravilnosti učinjene tokom analize zahteva, dizajna i implementacije otkrivaju se ranije;
- Tim za testiranje je uključen tokom celog životnog ciklusa projekta;
- Tim usavršava naučene lekcije i samim tim može neprekidno da unapređuje proces razvoja;

Slika 3.1.
Iterativan razvoj



Izvor: (Rational Team, 2005)

3.1.2. Upravljanje zahtevima

Prateći rezultate Standish grupe dolazi se do spoznaje, da je u 37% slučajeva razlog za neuspeh projekata vezan za informacione zahteve. Nedostatak korisničkih zahteva predstavlja razlog neuspeha projekata u 13% slučajeva, nepotpuni zahtevi i specifikacije u 12% slučajeva, a promena zahteva i specifikacija takođe u 12% slučajeva. Dodajući navedenim podacima činjenicu, da su troškovi otklanjanja grešaka nastalih u oblasti zahteva izuzetno visoki, iako ih iterativan pristup značajno umanjuje, nameće se zaključak da se zahtevima ne posvećuje dovoljna pažnja, u procesu razvoja informacionih sistema.

Informacioni zahtevi predstavljaju inpute za dizajn sistema, testiranje sistema, kao i izradu korisničke dokumentacije. Greške u fazi analize zahteva su pogubne po uspeh projekta razvoja. Da bi se gore navedeni procenti smanjili potrebno je posvetiti značajnu pažnju pronalaženju, organizovanju, dokumentovanju i upravljanju zahtevima. RUP upravo to i radi kroz disciplinu zahteva čiji je cilj da opiše šta sistem treba da radi. Taj opis treba da bude prihvaćen, kako od strane korisnika, tako i od strane razvojnih inžinjera.

3.1.3. Upotreba arhitekture zasnovane na komponentama

Vizuelizacija, specifikacija, konstrukcija i dokumentovanje softverskog sistema zahteva da sistem bude sagledan iz brojnih perspektiva. Svaki stejkholder (analitičari, integratori sistema, testeri, projektni menadžeri, dizajneri, ...) donosi različitu sliku projekta, i svaki od njih gleda sistem na različite načine, mnogo puta tokom života projekta. Arhitektura sistema je možda najvažnija podela, koja se može koristiti da bi se upravljalo ovim različitim gledištima i na taj način kontrolisao iterativni i inkrementalni razvoj sistema, tokom njegovog životnog ciklusa.

Arhitektura je prevashodno bitna za razvojni tim koji realizuje projekat, ali indirektno utiče i na zadovoljstvo krajnjeg korisnika. Arhitektura sistema obuhvata grupu značajnih odluka o:

- Organizaciji softverskog sistema
- Izboru gradivnih elemenata i njihovih interfejsa, od kojih je sistem sastavljen
- Ponašanju gradivnih elemenata, određenom upravo njihovom saradnjom
- Kompoziciji strukturalnih i bihevioralnih elemenata u veće rastuće podsisteme
- Arhitektonskom stilu, koji objedinjuje gore navedene odluke u jednu celinu.

Razvoj baziran na komponentama (CBD - Component-Based Development) je značajan pristup softverskoj arhitekturi jer omogućava ponovnu upotrebu, kao i specijalizaciju komponenti. Pored sopstvene izgradnje komponenti za ponovnu upotrebu moguća je upotreba komponenti i iz velikog broja komercijalno dostupnih izvora (Microsoft Component Object Model - COM i Microsoft Foundation Classes - MFC, Common Object Request Broker Architecture - CORBA, Enterprise JavaBeans - EJB samo su neke od široko podržanih platformi na kojima je arhitektura zasnovana na komponentama ostvariva).

Kombinujući iterativan razvoj sa arhitekturom zasnovanom na komponentama svaki korak proizvodi izvršnu arhitekturu sistema, koja je merljiva, pogodna za testiranje i vrednovanje u odnosu na zahteve sistema. Na ovaj način se otvara mogućnost za konstantno napadanje ključnih rizika projekta.

3.1.4. Vizuelno modelovanje

Model predstavlja pojednostavljenu sliku realnosti, koja kompletno opisuje sistem iz pojedinačnih perspektiva. U softverskom inžinjeringu modeli se izgrađuju da bi se bolje razumeo sistem koji se modeluje. Kod izrade velikih sistema modelovanje pomaže njihovom razumevanju u potpunosti. Slobodno se može konstatovati da bi bez modelovanja automatizacija takvih sistema bila nemoguća.

Činjenica da jedna slika govori više nego hiljadu reči je poznata odavno. Međutim, ono što je nedostajalo jeste standardizacija. UML kao jedinstveni jezik za modelovanje je prihvaćen od strane vodećih svetskih IT kompanija i nametnut kao standard u softverskoj industriji. On služi za vizuelizaciju, specifikaciju, konstrukciju i dokumentaciju sistema koji se izgrađuje. UML je omogućio svim članovima razvojnog tima da razgovaraju istim jezikom. Standardizacija je dovela do toga da se UML izučava u školama i fakultetima širom sveta, pa je time zamenjivost bilo kojeg člana razvojnog tima značajno olakšana. RUP i UML predstavljaju nerazdvojivu celinu.

Vizuelno modelovanje podiže kvalitet procesa razvoja softverskog proizvoda, što je lako uočljivo preko sledećih karakteristika:

- Use Case-ovi i scenarija nedvosmisleno definišu ponašanje sistema,
- Dizajn je jasan i nedvosmislen,
- Nemodularne i nefleksibilne arhitekture su lako uočljive na nivou modela,
- Različite perspektive i različiti nivoi detaljnosti se mogu koristiti u različiti situacijama,

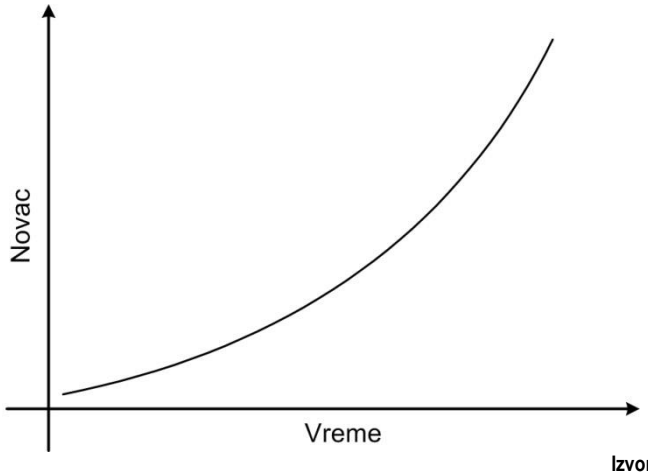
- Otklanjanje nekonzistentnosti na nivou dizajna, značajno olakšava implementaciju,
- Use Case model i dizajn model stvaraju jasne inpute za testiranje,
- Korišćenje UML-a nameće standardizaciju u softverskoj industriji.

3.1.5. Kontinuirana potvrda kvaliteta softvera

Kao što slika br. 3.2. ilustruje, softverski problemi su 100 do 1000 puta skuplji kada se pronalaze i otklanjaju nakon uvođenja nego pre toga. Upravo zato, važno je kontinuirano proveravati kvalitet sistema sa posebnim akcentom na njegovu funkcionalnost, pouzdanost, aplikativne performanse i sistemske performanse.

Proveravanje funkcionalnosti sistema uključuje kreiranje testova, za svaki ključni scenario koji predstavlja neki aspekt željenog ponašanja sistema. Prilikom proveravanja funkcionalnosti sistema potrebno je evidentirati svaki scenario koji nije zadovoljio zahteve i pristupati njegovom osposobljavanju. Poštujući itearativan razvoj softvera tako je potrebno pristupati i testiranju, testirajući svaku iteraciju.

Slika 3.2. Prikaz cene rešavanja grešaka spram vremena otkrivanja



Izvor: (Rational Team, 2005)

Provera kvaliteta softvera pruža brojna rešenja ključnim uzrocima problema razvoja softvera:

- Procena statusa razvojnog projekta se izrađuje objektivno, ne subjektivno, jer se vrednuju rezultati testa
- Objektivno procenjivanje otkriva nekonzistentnosti u zahtevima, dizajnu i implementaciji
- Testiranje i verifikacija su usmereni na oblasti najvišeg rizika, te se na taj način uvećavaju kvalitet i efektivnost tih oblasti
- Greške se otkrivaju rano, radikalno snižavajući troškove njihovog ispravljanja

3.1.6. Kontrola promena softvera

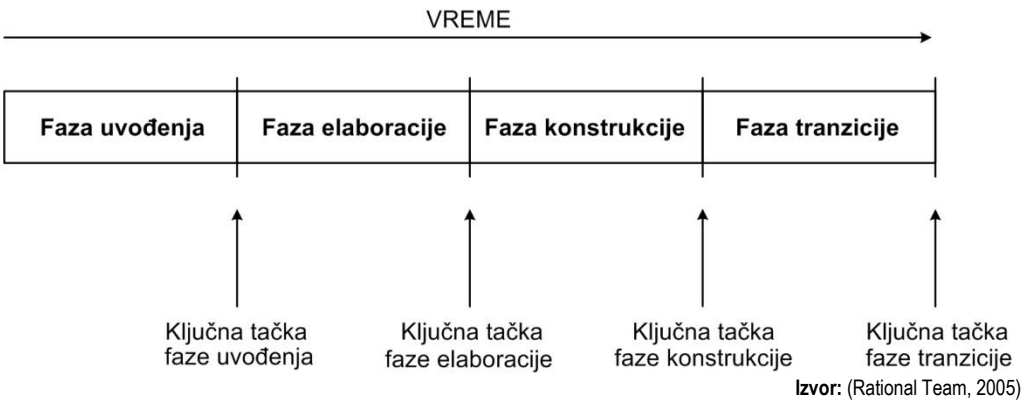
Ključni izazov prilikom razvoja softverskih sistema je usklađivanje rada razvojnih inžinjera, organizovanih u razvojne timove koji zajedno rade na više iteracija stvarajući softverski proizvod. Tokom razvoja novonastajući sistem se menja i razvija, a kontrola tih razvojnih promena je nužna za sprečavanje haosa u procesu razvoja.

Usklađivanje aktivnosti, artifakta i uloga predstavlja ponovljivi workflow koji prati razvoj softvera kroz promene u procesu razvoja. Standardizacija workflow-a, sastavljenih od aktivnosti, artifakta i uloga, u procesu razvoja omogućuje praćenje promena, rano otkrivanje problema i brzo reagovanje u cilju njihovog otklanjanja.

RUP razvoj informacionih sistema postavlja između dve dimenzije, vremenske i sadržajne. Vremenska dimenzija je podeljena u četiri faze: uvođenje, elaboracija, konstrukcija i tranzicija. Sadržajna dimenzija je podeljena u šest osnovnih i tri pomoćne discipline. U osnovne spadaju disciplina poslovnog modelovanja, disciplina zahteva, disciplina analize i dizajna, disciplina implementacije, disciplina testiranja i disciplina raspoređivanja. Pomoćne discipline su konfigurisanje i upravljanje promenama, upravljanje projektom i okruženje. Svaki element matrice RUP-a predstavlja kombinaciju elemenata statičke strukture RUP-a i vremenskog rasporeda istih.

RUP je moguće posmatrati kroz dve dimenzije, kroz dinamičku u kojoj se proces opisuje kroz životni ciklus razvoja proizvoda i statičku u kojoj se opisuju aktivnosti i rezultati aktivnosti podeljeni na uloge.

Slika 3.3.
Faze u životnom ciklusu po RUP-u



Izvor: (Rational Team, 2005)

RUP razvojni proces predstavlja kroz ciklus od četiri faze. Kao krajnji proizvod ovog ciklusa dobija se gotov softverski proizvod.

3.2. Faza Uvođenja

Uvođenje predstavlja prvu fazu RUP-a. Krajnja svrha ove faze jeste stvaranje pretpostavki za potpunu saradnju svih stejkholdera budućeg projekta, kao i presretanje i stavljanje pod kontrolu većine rizika pre započinjanja razvojnih aktivnosti. U skladu sa navedenom svrhom kao najznačajniji ciljevi se nameću:

1. Razumevanje šta se gradi. Pod ovim se podrazumeva uspostavljanje obima projekta, utvrđivanje graničnih kriterijuma i definisanje šta će biti deo projekta, a šta ne.
2. Identifikovanje ključnih funkcionalnosti sistema. U toku ove aktivnosti identifikuju se najznačajniji Use Case-ovi sistema. Najčešće se za sistem od 20-tak Use Case-ova identifikuje 4-5 ključnih, čijim opisom se u stvari opisuju ključne funkcionalnosti koje budući sistem treba da pruža.
3. Definisanje, a ukoliko je moguće i demonstriranje najmanje jednog mogućeg arhitekturalnog rešenja. Ovom aktivnošću se utvrđuje da li postoji odgovarajuća arhitektura sistema kojom se mogu zadovoljiti tražene funkcionalnosti. Ukoliko se uoči više arhitekturalnih solucija potrebno ih je sve predstaviti, a u kasnijim fazama će se doneti odluka izbora optimalne arhitekture.
4. Identifikovanje troškova, utvrđivanje rasporeda izvršenja projekta. Ovo aktivnost predstavlja poslovnu procenu razvoja i održavanja budućeg sistema. Na osnovu nje se obezbeđuje odgovor na pitanje da li su projekat i rezultati projekta održivi ili ne.
5. Uočavanja potencijalnih rizika, kako poslovnih tako i onih vezanih za zahteve korisnika.
6. Priprema podrške i definisanje okruženja za razvoj projekta. Ova aktivnost podrazumeva razradu procesa razvoja i određivanje alata koji će se koristiti. Neophodno je da svi članovi razvojnog tima dele isti pogled na to kako će se softver razvijati. U cilju toga se vrši razrada procesa razvoja i određuju se alati pomoću kojih će se razvijati softverski proizvod.

U ključnoj tački faze Uvođenja, a nakon ispunjenja postvaljenih ciljeva, potrebno je znati odgovore na sledeća pitanja:

- Da li je projekat izvodljiv?
- Da li je nivo rizika prihvatljiv?
- Da li je projekat finansijski isplativ?

Za dostizanje postavljenih ciljeva izvršavaju se aktivnosti od strane različitih uloga u razvojnom projektu. Pri tome se koriste i kreiraju odgovarajući artifakti koji predstavljaju konkretizaciju projektnih aktivnosti.

U prethodnom pasusu uvedeno je nekoliko novih pojmova koji će u kasnijem izlaganju biti često upotrebljavani. To su sledeći pojmovi:

- Uloga– Uloga predstavlja centralni koncept u procesu. Ona defiše ponašanje i odgovornost osobe ili grupe odnosno tima koji radi zajedno. Ponašanje predstavlja aktivnost koje uloge izvršavaju, a svaka uloga je zadužena za određeni set aktivnosti.
- Artifakt – Artifakt je informacija koju modifikuje ili koristi neka aktivnost. Oni se koriste kao inputi od strane uloga kako bi uloge mogle da izvrše neku aktivnost, a takođe su i izlazni rezultat neke aktivnosti.
- Aktivnost – Aktivnost je deo procesa rada koju neka uloga izvršava. Aktivnosti se odnose na kreiranje, modifikovanje artifakata kao što su modeli, klase i planovi. Svaka aktivnost je sastavni deo jedne uloge.

Ovo su ključni pojmovi pri izvršavanju projektnih aktivnosti. U nastavku je prikazan workflow Uvođenja faze po Rational Unified Process-u. Svaki od prikazanih koraka se sprovodi od strane uloga u projektu, koje realizuju aktivnosti i pri tome koriste i grade artifakte, te na takav način grade softverski projekat.

U tabeli 3.1. opisano je stanje ključnih artifakta u prvoj ključnoj tački na kraju faze Uvođenja životnog ciklusa razvoja proizvoda po RUP-u.

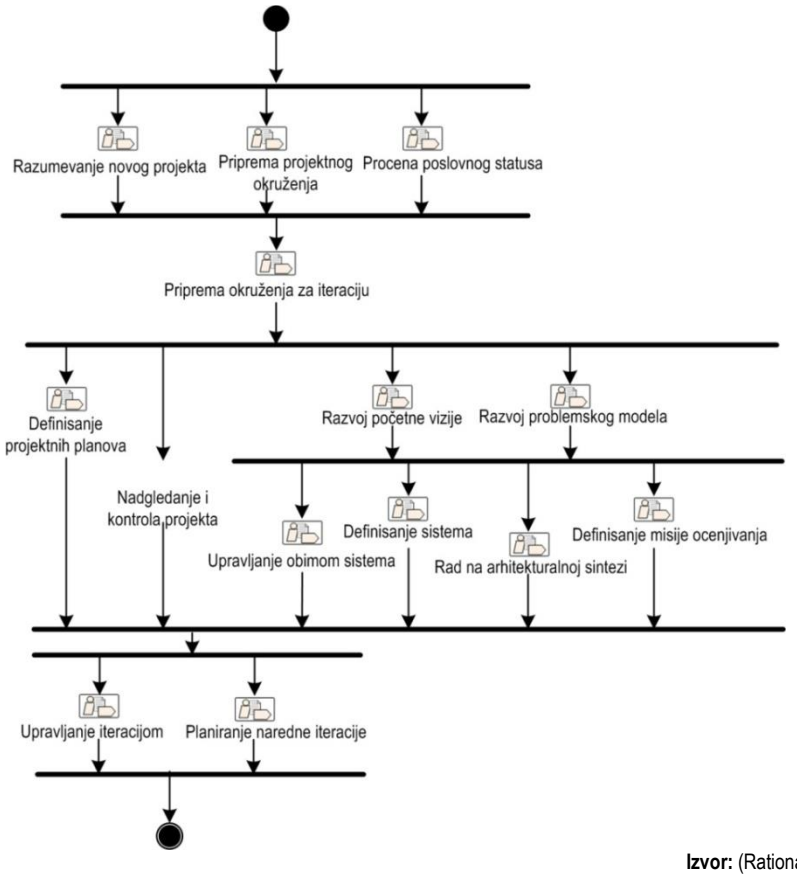
Tabela 3.1. Stanje najznačajnijih artifakta u ključnoj tački faze Uvođenja

Ključni artifakti poredani po važnosti	**Stanje artifakta u ključnoj tački**
Dokument vizije	Identifikovani osnovni zahtevi i ključne funkcionalnosti, te dokumentovana najznačajnija ograničenja
Dokument poslovnog slučaja	Definisan i odobren
Lista rizika	Inicijalna lista rizika identifikovana
Razvojni plan projekta	Razvojne faze i njihovo trajanje identifikovano. Definisani ciljevi. Preračunati su utrošci resursa (vreme, osoblje, troškovi vezani za okruženje i sl.) koji moraju biti konzistentni sa dokumentom poslovnog slučaja (Troškovi mogu biti proračunati ili za čitav projekat ili samo za fazu elaboracije, kao narednu fazu u životnom ciklusu razvoja proizvoda. Ukoliko se radi o kompletnom proračunu nije realno očekivati preteranu tačnost. Tu se obično radi o grubom proračunu koji se dopunjuje pre svake faze i/ili itearacije.).
Plan iteracija	Plan iteracija u ovoj tački sadrži u potpunosti razrađenu i odobrenu prvu iteraciju za fazu Elaboracije.
Razvojni proces	Adaptacije u proširenja RUP-a za konkretan razvojni postupak u ovo fazi su završena i odobrena
Razvojna infrastruktura	Alati za rad u toku projekta su izabrani, a alati potrebni za rad u fazi Uvođenja i početnoj iteraciji faze Elaboracije su instalirani.
Rečnik	Važni pojmovi su definisani. Rečnik je pregledan i odobren.
Use Case model	Većina aktera i slučajeva upotrebe je identifikovana, a dijagrami aktivnosti su razrađeni za najznačajnije slučajeve upotrebe.

Izvor: (Rational Team, 2005)

3.2.1 Koraci u fazi uvođenja

Slika 3.4.
Rational Unified Process – Workflow Uvođenja faze



Izvor: (Rational Team, 2005)

Koraci koji se sprovode u fazi uvođenja su:

1) Razumevanje novog projekta;
2) Priprema projektnog okruženja;
3) Procena poslovnog statusa;
4) Priprema okruženja za iteraciju;
5) Definisanje projektnih planova;
6) Nadgledanje i kontrola projekta;
7) Razvoj početne vizije;
8) Definisanje sistema;
9) Definisanje misije ocenjivanja;
10) Upravljanje obimom sistema;
11) Rad na arhitekturalnoj sintezi;
12) Razvoj problemskog modela;
13) Upravljanje iteracijama i
14) Planiranje naredne iteracije.

3.3. Faza Elaboracije

Faza Elaboracije predstavlja drugu fazu životnog ciklusa razvoja proizvoda po RUP-u. Krajnja svrha faze Elaboracije jeste obezbeđivanje takve arhitekturalne osnove koja će obezbediti stabilno i neometano sprovođenje aktivnosti u fazama Konstrukcije i Tranzicije. Na ovakav način se vrši značajna mitigacija rizika u tehničko tehnološkom domenu razvojnog projekta. Do stabilnosti arhitekture se dolazi kroz ocenjivanje jedne ili više arhitekturalnih alternativa, uvažavajući pri tome ključne zahteve, čije zadovoljenje zavisi od arhitekturalne osnove budućeg rešenja. Do postizanja krajnje svrhe faze Elaboracije se dolazi kroz dostizanje primarnih ciljeva:

- Obezbeđenje takve usklađenosti arhitekture, zahteva i planova koja će obezbediti mogućnost predviđanja troškova i rasporeda izvršenja aktivnosti do samog završetka projekta.
- Identifikovanje svih arhitekturalno značajnih rizika u projektu.
- Uspostavljanje osnovne arhitekture na osnovu analize svih arhitekturalno značajnih scenarija za izvršenje funkcionalnosti budućeg sistema.
- Izrada razvojnog prototipa sačinjenog od razvojnih komponenti, koje će biti upotrebljene u procesu razvoja, a eventualno i u nekom kasnijem razvojnom procesu. Na takav način se postiže ublažavanje rizika vezano za ponovnu upotrebu komponenti, kao i za demonstriranje izvodljivosti pred investitorima i/ili krajnjim korisnicima.
- Demonstriranje takve osnovne arhitekture koja će obezbediti zadovoljenje zahteva u razumnom vremenskom roku i u okviru razumnog budžeta.

U ključnoj tački faze Elaboracije potrebno je postići sledeće:

- Dokument vizije i zahtevi treba da su čvrsto definisani.
- Arhitektura treba da je čvrsto definisana.
- Ključni pristupi koji će biti korišćeni u testiranju i ocenjivanju treba da su odobreni.
- Kroz testiranje i ocenjivanje izvršnog prototipa potrebno je prikazati da su ključni rizici identifikovani i da postoji značajna verovatnoća da će biti rešeni.
- Plan iteracija za fazu Konstrukcije treba da je detaljno razrađen da se sa sigurnošću može nastaviti proces razvoja.
- Saglasnost ključnih stejkholdera da je definisanu Viziju moguće postići ukoliko postojeći planovi budu uspešno sprovedeni, posmatrano u kontekstu razvijene arhitekture.
- Stvarni utrošak resursa postavljen u odnos sa planiranim utroškom resursa treba da je prihvatljiv.

Stanje najznačajnijih artifakta u ključnoj tački faze Elaboracije je detaljno prikazano u tabeli 3.2. Posmatrano kroz artifakte koji se proizvode i dorađuju u fazi Elaboracije moguće je sprovesti proveru dostignutosti navedenih postavki.

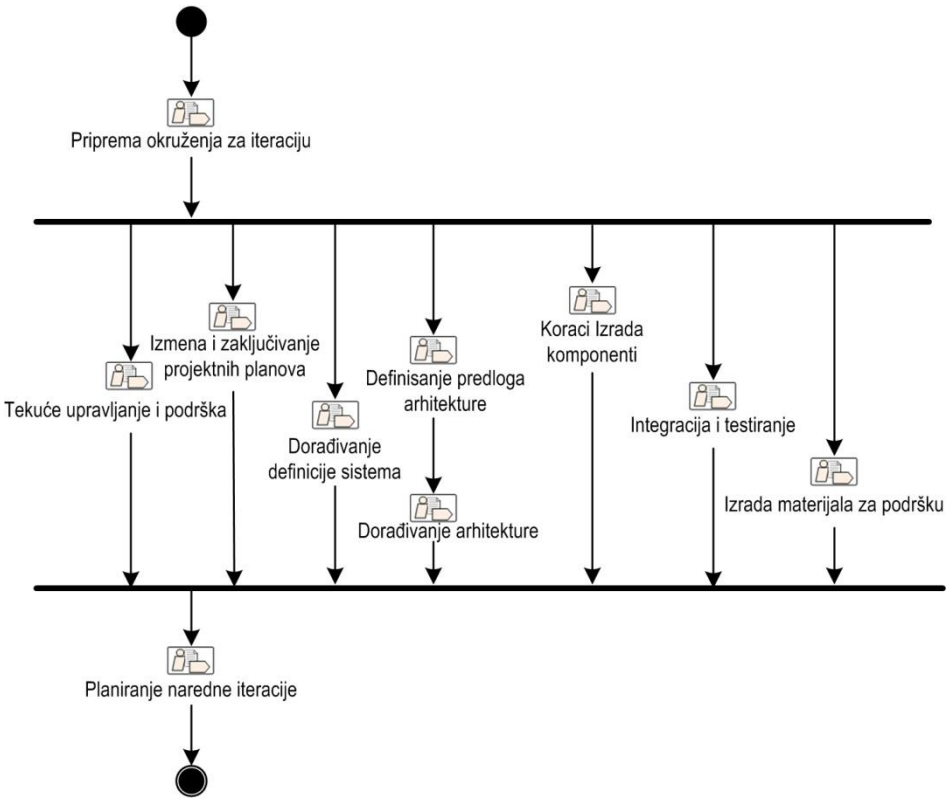
Tabela 3.2. Stanje najznačajnijih artifakta u ključnoj tački faze E-laboracije

Ključni artifakti poredani po važnosti	**Stanje artifakta u ključnoj tački**
Razvojni prototip	Kreiran razvojni prototip (moguće i više razvojnih prototipa) u cilju utvrđivanja načina funkcionisanja aplikativnih scenarija.
Lista rizika	Dopunjena i odobrena. Dodati su rizici vezani za arhitekturu sistema, primarno vezani za nefunkcionalne zahteve.
Razvojni proces	Izmenjen na osnovu iskustva u ranijim fazama projekta i pripremljen za naredne faze.
Razvojna infrastruktura	Alati i podrška za proces u fazi konstrukcije dopunjena u dokumentu Razvojne infrastrukture.
Dokument softverske arhitekture	Kreiran i postavljen kao ključni arhitekturalni dokument. Dat detaljan opis arhitekturalno značajnih slučajeva upotrebe, identifikovani ključni mehanizmi i elementi dizajna, opisan procesni pogled i pogled rasporeda.
Dizajn model	Dizajn arhitekturalno značajnih slučajeva upotrebe je definisan i utvrđeno je ponašanje pojedinačnih elemenata dizajna. Komponente informacionog sistema su identifikovane i donešena je odluka o njihovoj izgradnji, kupovini ili ponovnoj upotrebi, te je na osnovu toga utvrđen raspored poslova u fazi konstrukcije.
Model podataka	Ključni elementi modela podataka (značajni entiteti, njihovi atributi i relacije među njima) su definisani i verifikovani.
Implementacioni model	Inicijalna struktura je kreirana i izrađen je prototip ključnih komponenti.
Dokument vizije	Prepravljen na osnovu informacija dobijenih u prve dve faze, pre svega u pogledu arhitekturalnih odluka i odluka vezanih za buduće planove.
Plan razvoja softvera	Dopunjen i proširen radi pokrivanja narednih faza.
Plan iteracija	Plan iteracija u ovoj tački sadrži u potpunosti razrađenu i odobrenu prvu iteraciju za fazu Konstrukcije.
Use Case model	Svi slučajevi upotrebe su identifikovani. Oko 80% slučajeva upotrebe je u potpunosti razrađeno, a svi akteri su identifikovani.
Dodatne specifikacije	Dodatne specifikacije su obuhvaćene, dokumentovane i verifikovane.
Test paket	Testovi su izgrađeni i sprovedeni za sve izvršne elemente koji su izgrađeni u fazi elaboracije (elementi koji su služili za uspostavljanje arhitekture).
Arhitektura automatizivanih testova	Identifikovani ključni elementi koji će biti predmet testiranja kroz automatizovani sistem testiranja.

Izvor: (Rational Team, 2005)

3.3.1. Koraci u fazi elaboracije

Slika 3.5. Rational Unified Process – Workflow faze E-laboracije



Izvor: (Rational Team, 2005)

Koraci koji se sprovode u fazi uvođenja su:

1) Priprema okruženja za iteraciju;
2) Izmena i zaključivanje projektnih planova;
3) Tekuće upravljanje i podrška;
4) Dorađivanje definicije sistema;
5) Definisanje predloga arhitekture;
6) Dorađivanje arhitekture;
7) Izrada komponenti;
8) Izrada materijala za podršku;
9) Integracija i testiranje i
10) Planiranje naredne iteracije

3.4. Faza Konstrukcije

Krajnja svrha faze Konstrukcije jeste dostizanje potpune razumljivosti zahteva i kompletiranje razvoja sistema, poštujući unapred definisanu arhitekturu sistema. Faza Konstrukcije se može posmatrati kao proizvodna faza u kojoj se na osnovu dobro razrađenog modela (dobijenog kroz prethodne faze) izrađuje u potpunosti upotrebljiv proizvod, koji će biti završen kroz faze Konstrukcije i Tranzicije.

Ciljevi koje je potrebno ispuniti tokom ove faze su sledeći:

- Minimiziranje razvojnih troškova kroz optimizaciju resursa;
- Dostizanje paralelizma u radu razvojnih timova;
- Dostizanje odgovarajućeg kvaliteta proizvoda;
- Dostizanje upotrebljive verzije proizvoda, kroz izbacivanje i testiranje verzija;
- Iterativno i inkrementalno razvijanje proizvoda sve do dostizanja njegove spremnosti za transfer u ruke korisnika;
- Prepoznavanje trenutka u kojem su i proizvod i korisnici spremni za isporuku.

Ključne tačke koje je potrebno dostići tokom ove faze su prikazane u tabeli 3.3.

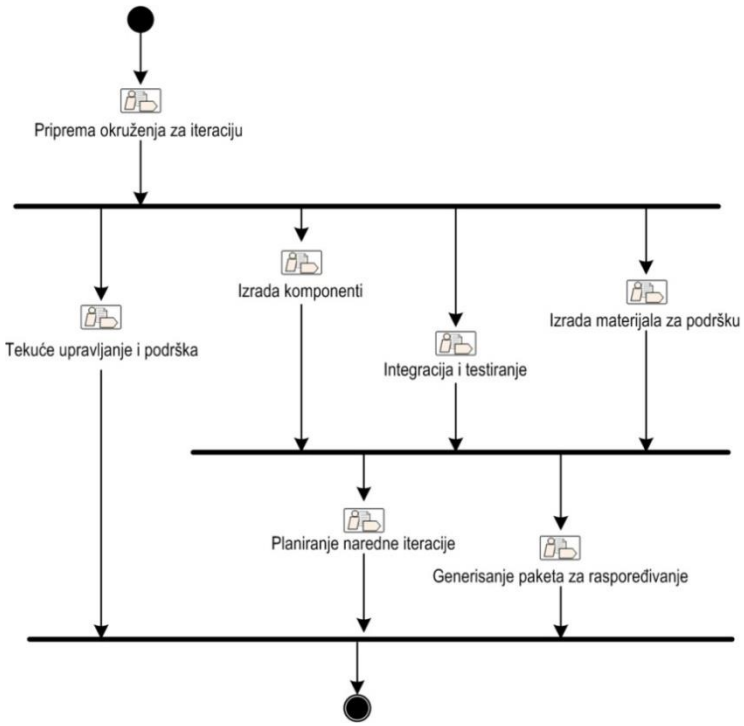
Tabela 3.3. Stanje najznačajnijih artifakta u ključnoj tački faze Konstrukcije

Ključni artifakti poredani po važnosti	Stanje artifakta u ključnoj tački
Sistem	Izgrađen do te faze da je izvršiv i da je spreman za „beta“ testiranje.
Plan raspoređivanja	Izgrađen, pregledan i spreman za sprovođenje.
Implementacioni model	Implementacioni elementi su kreirani i spremni za raspoređivanje.
Test paket	Izgrađen i sproveden, a njegovo sprovođenje potvrdilo stabilnost proizvoda za svaku isporučenu verziju.
Materijali za podršku	Izrađeni, posebno u slučaju kada korišćenje sistema jako zavisi od upotrebe korisničkog interfejsa.
Plan iteracija	Plan iteracija za fazu tranzicije je izgrađen i odobren.
Dizajn model	Dopunjen sa novim elementima nastalim usled novonastalih zahteva.
Razvojni proces	Prerađen na osnovu dosadašnjeg projektnog iskustva i prilagođen narednoj fazi.
Razvojna infrastruktura	Pripremljena za narednu fazu. Svi alati i automatizovani procesi prilagođeni i spremni za narednu fazu projekta.
Model podataka	Dopunjen sa svim elementima neophodnim za podršku implementaciji.

Izvor: (Rational Team, 2005)

3.4.1. Koraci u fazi konstrukcije

Slika 3.6.
Workflow faze Konstrukcije



Izvor: (Rational Team, 2005)

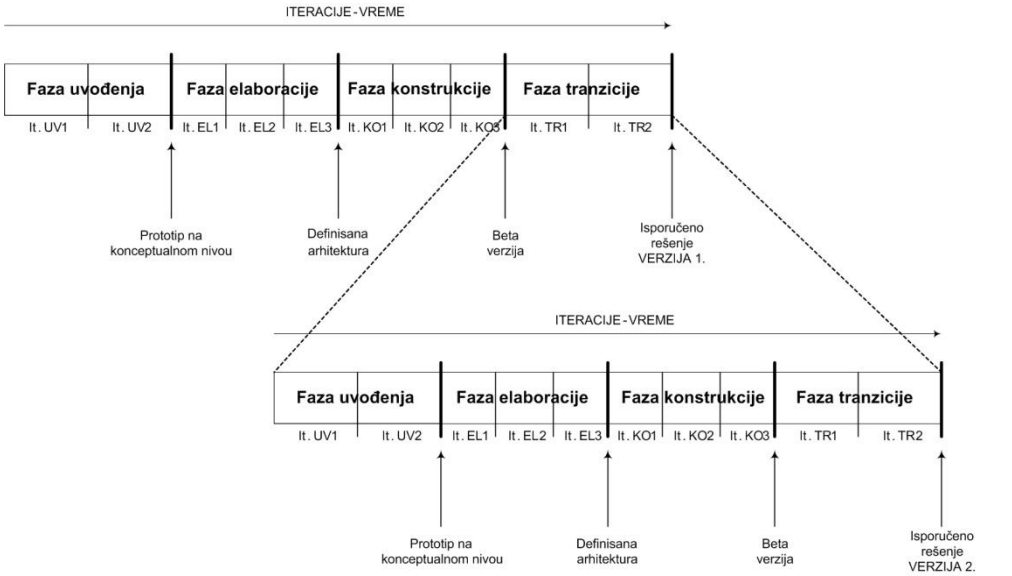
Na slici 3.6. je prikazan workflow faze Konstrukcije. Koraci koje je potrebno realizovati u ovoj fazi su sledeći:

1) Priprema okruženja za iteraciju;
2) Izrada komponenti;
3) Izrada materijala za podršku;
4) Tekuće održavanje i podrška;
5) Integracija i testiranje;
6) Planiranje naredne iteracije i
7) Generisanje paketa za raspoređivanje.

3.5. Faza Tranzicije

Glavni zadatak faze Tranzicije jeste da se softverski proizvod koji se razvija učini dostupnim za korišćenje krajnjim korisnicima. Kroz fazu Tranzicije potrebno je sprovesti i testiranje proizvoda kao obaveznu aktivnost u pripremi za isporuku. Povratne informacije koje se dobiju kroz takav vid testiranja mogu poslužiti za manje korekcije proizvoda. Na kraju ove faze proizvod treba da je isporučen krajnjem korisniku, a projekat treba da je u potpunosti spreman za zatvaranje. U nekim slučajevima kada proizvod nije razvijen u skladu sa zahtevima korisnika, a kada korekcije potrebne da se dovede u željeno stanje nisu male, neophodno je pokretanje novog životnog ciklusa za razvoj proizvoda (slika 3.7.).

Slika 3.7. Novi razvojni ciklus započet u fazi Tranzicije



Izvor: (Rational Team, 2005)

Faza Tranzicije može da bude vrlo jednostavna, najčešće kod projekata koji su se bavili razvojem nove verzije postojećeg proizvoda, ili vrlo kompleksna, kod velikih i sigurnosno zahtevnih sistema. Međutim, postoje operativni ciljevi koji se moraju zadovoljiti u svakom od navedenih slučajeva:

- Beta testiranje treba da potvrdi da je novi proizvod u skladu sa očekivanjima korisnika,
- Beta testiranje se treba sprovesti paralelno sa funkcionisanjem postojećeg proizvoda koji će biti zamenjen,
- Potrebno je izvršiti konverziju operativne baze podataka,
- Potrebno je sprovesti trening korisnika i osoba koje će vršiti održavanje proizvoda,
- Potrebno je pripremiti marketinške, distributivne i prodajne aktivnosti,

- Potrebno je sprovesti aktivnosti za fino podešavanje proizvoda, poput popravke grešaka (bug fixing) i unapređenja proizvoda u smislu podizanja performantnosti i upotrebljivosti,
- Neophodno je sprovesti procenu uočenih karakteristika proizvoda pri raspoređivanju u odnosu na definisanu viziju proizvoda i kriterijume prihvatljivosti proizvoda,
- Korisnike je potrebno obučiti do te mere da mogu određene probleme samostalno rešavati,
- Od korisnika je potrebno dobiti saglasnost da je proizvod raspoređen i spreman za upotrebu,
- Takođe, od korisnika je potrebno dobiti saglasnost da raspoređeni proizvod odgovara kriterijumima u čijem postavljanju su oni učestvovali i koji su definisani kroz dokument vizije.

Faza Tranzicije se, kao što je prikazano na slici 3.8., sprovodi u jednoj ili više iteracija. Kada se sprovedu sve planirane iteracije stanje artifakta bi trebalo biti u skladu sa planiranim. Stanje u ključnoj tački u ovoj fazi je najprepoznatljivije, jer svi artifakti treba da su kompletirani i proizvod treba da je isporučen.

Slika 3.8. Ključna tačka faze Tranzicije

Izvor: (Rational Team, 2005)

Stanje artifakta u ključnoj tački faze Tranzicije je prikazano u tabeli 3.4.

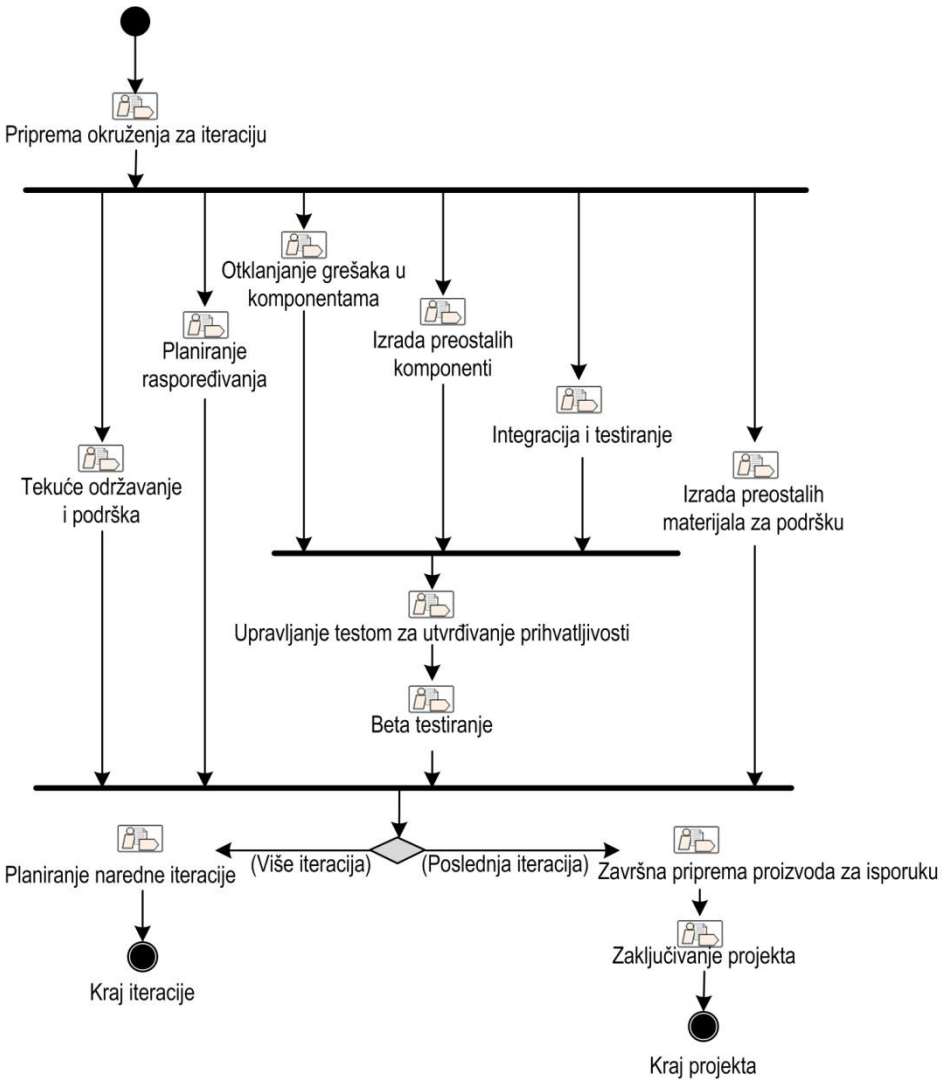
Tabela 3.4. Stanje artifakta u ključnim tačkama faze Tranzicije

Ključni artifakti (poređani po značaju)	**Stanje u ključnoj tački**
Verzija proizvoda	U potpunosti završena u skladu sa zahtevima korisnika. Finalni proizvod bi trebao biti u takvom stanju da je u potpunosti upotrebljiv od strane korisnika.
Materijali za podršku	Materijali koji treba da pomažu krajnjim korisnicima u korišćenju proizvoda treba da su u ovoj tački u potpunosti završeni.
Elementi implementacije	Elementi koji su se dorađivali u ovoj fazi bi trebali biti u potpunosti završeni i inkorporirani u finalni proizvod.
Opcionalni artifakti	**Stanje u ključnoj tački**
Test paket („smoke test“)	Test paket koji je razvijen da utvrdi primenjivost treba da je u potpunosti razvijen i sproveden nad finalnim proizvodom.
Upakovan proizvod za dalju prodaju	U slučaju kada se radi o proizvodu namenjenom nepoznatom kupcu, tj. široj prodaji, potrebno je proizvod upakovati i pripremiti ga za isporuku. U ovoj tački proizvod mora da je u potpunosti spreman za konačnu isporuku.

Izvor: (Rational Team, 2005)

3.5.1. Koraci u fazi tranzicije

Slika 3.9.
Rational Unified Process – Workflow Tranzicije faze



Izvor: (Rational Team, 2005)

U fazi tranzicije potrebno je realizovati sledeće korake:

1) Priprema okruženja za iteraciju;
2) Otklanjanje grešaka u komponentama ;
3) Planiranje raspoređivanja;
4) Izrada preostalih komponenti
5) Integracija i testiranje;
6) Tekuće održavanje i podrška;
7) Izrada preostalih materijala za podršku;
8) Upravljanje testom za utvrđivanje prihvatljivosti;
9) Beta testiranje;
10) Završna priprema proizvoda za isporuku;
11) Planiranje naredne iteracije i
12) Zaključivanje projekta.

4. Jedinstveni jezik za modelovanje UML (Unified Modeling Language)

4.1. Šta je UML?

UML je jedna o najpoznatijih skraćenica u informatičkom svetu. Skraćenica potiče od englskog termina Unified Modeling Language, što u prevodu znači jedinstveni jezik za modelovanje. Najuopštenija definicija UML-a bi mogla biti sledeća. UML predstavlja jedinstveni jezik za vizuelizaciju, specifikaciju, konstrukciju i dokumentaciju softverskih sistema.

Jezik za modelovanje može biti bilo koji opis koji pomaže izgradnji sistema. Taj opis, govoreći o softverskim proizvodima, može biti predstavljen pseudo kodom, dijagramima, slikama, tekstualnim opisima, itd. Elementi kojima se vrši modelovanje čine notaciju za modelovanje. Ukoliko se kao elementi koriste grafički simboli jezik za modelovanje je grafičkog tipa, tj. grafički jezik za modelovanje. UML poseduje grafičku notaciju, te se može kategorizovati kao grafički jezik za modelovanje.

Grafički jezici poput UML-a koriste se već dugo u softverskoj industriji. Međutim ono što je specifično za sve te grafičke jezike, preteče UML-a, jeste njihova neusaglašenost. Upravo ta neusaglašenost je bila inicijalna kapsula koja je uticala na nastanak UML-a i koja pomaže da se razumeju kvaliteti koje je UML doneo svojim nastankom softverskoj industriji.

UML je nastao kao posledica saradnje Grady Booch-a, Jamesa Rumbaugh-a i Ivar Jacobson-a. Svako od njih se bavio razvojem sopstvenih notacija, a njihovim ujedinjenjem, pod okriljem kompanije „Rational Software“, nastao je UML. Popularan naziv za trojicu idejnih tvoraca UML je „tri amigosa“. Danas kompanija „Rational Software“ posluje kao deo IBM-a.

UML danas predstavlja relativno otvoren standard, kontrolisan od strane Object Management Group (u nastavku OMG), nezavisnog konzorcijuma kompanija koji upravlja standardima u objektno orijentisanom razvoju. Najnovije specifikacije UML je moguće pronaći na sajtu OMG-a, www.omg.org.

Istorijski posmatrano UML je nastao ujedinjenjem sledećih pristupa:

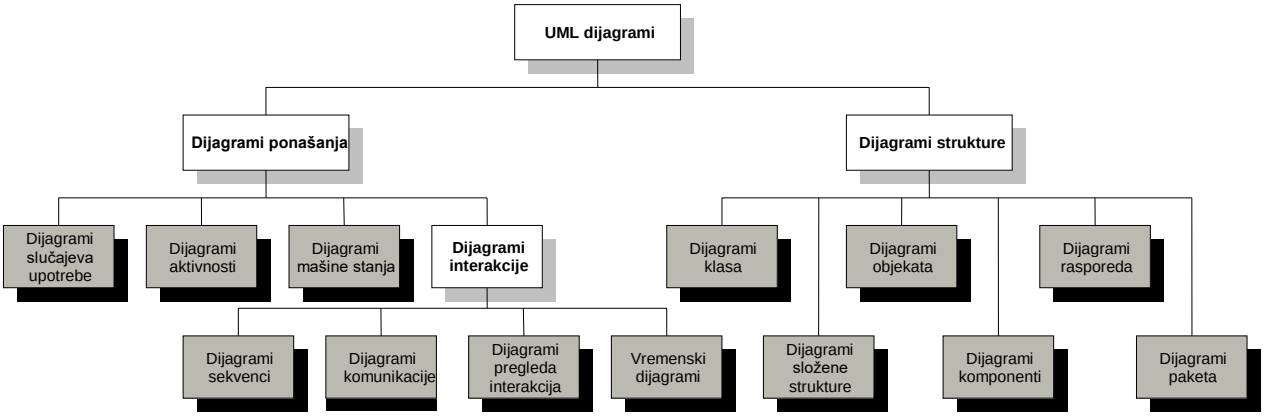
- *Object-Oriented Analysis and Design (OOAD)*, čiji je tvorac Grady Booch,
- *Object-Oriented Software Engineering (OOSE)*, čiji je tvorac Ivar Jacobson,
- *Object Modeling Technique (OMT)*, čiji je tvorac James Rumbaugh,
- *Probrane objektno orijentisane tehnike*, dodatne tehnike koje su se inicijalno našle u UML-u.

Inicijativa za stvaranje UML-a je krenula još 1996. godine, kao pokušaj da se prevaziđe rat objektnih notacija, te da se jedna notacija nametne kao standard. Kao kompanija za realizaciju ovoga projekata nametnula se „Rational Software“ korporacija. To nije bilo

jednostavno, te je za ovakav izbor bila potrebna saglasnost velikih softverskih giganata poput IBM-a ili Microsoft-a. Zajednički interes je doneo prevagu, te „Rational Software“ korporacija ujedinjuje Boocha-a, Jacobson-a i Rumbaugh-a, te izbacuje UML kao jedinstveni jezik za modelovanje. U saradnji sa OMG-om UML se 1997. godine nameće kao standard, a njegov munjevit razvoj dokazuje opravdanost političkog delovanja na razvoju UML-a. Trenutno aktuelna verzija UML-a je 2.0.

4.2. Kratak pregled UML dijagrama

Trenutno aktivna verzija UML-a jeste 2.0. U ovoj verziji postoji 13 vrsta dijagrama (predstavljeni tamnim pravougaonicima na slici br. 4.1.). U procesu razvoja razne uloge koriste različite tehnike dijagramiranja, za rešavanje različitih problema. To u stvari znači da različiti učesnici u procesu razvoja „razgovaraju“ istim jezikom. Da ne bi došlo do zabune potrebno je napomenuti da se proces razvoja ne može obaviti samo sa UML-om. Međutim, standardizacijom i opšteprihvaćenošću UML-a stvoreni su preduslovi da sve aktivnosti koje se ne realizuju UML-om budu usaglašene sa aktivnostima koje se realizuju pomoću njega. U nastavku je dat kratak opis 13 vrsta UML dijagrama.



Slika 4.1. Vrste UML dijagrama u verziji 2.0

Dijagrami slučajeva upotrebe

Dijagrami slučajeva upotrebe služe da daju grub opis funkcionalnosti posmatranog sistema ili posmatranog dela organizacije. U principu se može konstatovati da postoje dve vrste ovih dijagrama: dijagrami slučajeva upotrebe (Use Case Diagrams) i dijagrami slučajeva upotrebe poslovnog procesa (Business Use Case Diagrams). Dijagrami slučajeva upotrebe treba da daju odgovor na pitanje „Šta sistem radi?“, dok dijagrami slučajeva upotrebe poslovnog procesa treba da daju odgovor na pitanje „Šta organizacija radi?“. Pomoću dijagrama slučajeva upotrebe predstavljaju se funkcionalnosti koje će biti automatizovane, a pomoću dijagrama slučajeva upotrebe poslovnog procesa i automatizovane i manuelne.

Gradivni elementi ovih dijagrama su akteri, slučajevi upotrebe i relacije. Akter predstavlja nekoga ili nešto što se nalazi izvan sistema ili organizacije (u zavisnosti od vrste

dijagrama), a u interakciji je sa njom. Slučajevi upotrebe i slučajevi upotrebe poslovnog procesa služe da bi se pomoću njih prikazale konkretne funkcionalnosti sistema, odnosno organizacije.
Ovi dijagrami predstavljaju vodilju za kompletan proces razvoja, pa se često za razvoj zasnovan na UML-u kaže da je usmeravan slučajevima upotrebe.

Dijagrami aktivnosti

Dijagrami aktivnosti služe za opisivanje logike procedura, poslovnih postupaka i toka posla. Tok funkcionalnosti predstavljenih dijagramima slučajeva upotrebe se opisuje dijagramima aktivnosti, prikazujući sve aktivnosti koje se odvijaju u okviru posmatrane funkcionalnosti. Pomoću jednog dijagrama aktivnosti moguće je prikazati više potencijalnih scenarija koji se mogu desiti pri izvršavanju neke funkcionalnosti. Ukoliko se na dijagramima slučajeva upotrebe slučaj upotrebe posmatra kao „crna kutija“, na dijagramima aktivnosti se prikazuje redosled izvršavanja aktivnosti u okviru te „crne kutije“.

Dijagrami stanja mašine

Ovi dijagrami služe za prikazivanje ponašanja dela sistema, odnosno ponašanje objekta kao instance posmatrane klase. Na njima se predstavljaju stanja posmatranog objekta, tranzicije između stanja i događaji koji uzrokuju tranzicije objekta iz jednog u drugo stanje. Crtanje dijagrama stanja mašine se ne preporučuje za sve klase sistema. Najčešće se crtaju za najznačajnije klase sistema ili se uopšte ne crtaju ukoliko razvojni tim informacije koje se dobijaju ovim dijagramima dobije na drugi način.

Dijagrami sekvenci i dijagrami komunikacije

Ove dve vrste dijagrama prikazuju iste informacije iz različitih perspektiva. Pomoću njih se predstavljaju objekti koji se pojavljuju u okviru slučaja upotrebe i poruke koje oni razmenjuju. Razlika između ovih dijagrama je u tome što dijagrami sekvenci u prvi plan stavljaju vremensku dimenziju, tj. razmenu poruka sa aspekta vremena, dok dijagrami komunikacije zanemaruju vremensku dimenziju i prikazuju saradnju objekata kroz razmenu poruka. CASE alati pomoću kojih se crtaju ovi dijagrami omogućavaju automatsko generisanje jedne vrste dijagrama, na osnovu nacrtanog dijagrama druge vrste.

Dijagrami pregleda interakcija

Ovi dijagrami su jedna od novina verzije 2.0. Predstavljaju kombinaciju dijagrama aktivnosti i dijagrama sekvenci. Mogu se posmatrati kao dijagrami aktivnosti u kojima su aktivnosti zamenjene sa dijagramima sekvenci.

Vremenski dijagrami

Vremenski dijagrami su takođe novina u verziji 2.0. Iako se odavno koriste pri rešavanju elektro-tehničkih problema, tek su u verziji 2.0 uvršteni u UML. Ova vrsta dijagrama je slična dijagramima stanja mašine, sa tom razlikom što se vreme pojavljuje kao inicijator promene stanja objekta. Pored mogućnosti praćenja promena stanja jednog objekata moguće je pratiti i porediti promenu stanja više objekata. Dakle, ovi dijagrami prikazuju stanja u koja objekti dolaze nakon unapred predefinisanog vremenskog intervala.

Dijagrami klasa

Dijagrami klasa predstavljaju ključne dijagrame za opisivanje strukture sistema. Klasa predstavlja osnovni pojam u objektnom razvoju, te kao takva predstavlja i osnovnu komponentu objektno razvijanog sistema. Ovi dijagrami opisuju klase sistema i to kroz opis njima pripadajućih atributa, operacija i relacija između klasa.

Atributi predstavljaju obeležja koja opisuju svojstva klase. Navode se u okviru klase, a svaki atribut može da poseduje sledeća svojstva: vidljivost, ime, tip, multiplicitet, podrazumevanu vrednost, kao i opis nekih dodatnih svojstava atributa (npr. čitljivost).

Operacije opisuju poslove koje će objekat, kao konkretna pojava klase, znati da obavi. Kada se od objekta zahteva da obavi neki posao, to se može zahtevati samo preko operacije koju on poseduje. I operacije, kao i atributi poseduju odgovarajuća svojstva, i to: vidljivost, ime, listu parametara, tip rezultata operacije, itd.

Relacije služe da bi se prikazao međusobni odnos klasa. Između klasa mogu da postoje sledeće vrste relacija: asocijacija, agregacija, zavisnost i generalizacija.

Dijagrami objekata

Dijagrami objekata su postojali i u ranijim verzijama UML-a, ali kao neformalni dijagrami. Od verzije 2.0 postaju i zvanični UML dijagrami. Služe da prikažu objekte posmatranog dela sistema u posmatranom trenutku. Najsličniji su dijagramima saradnje, ali bez tretiranja poruka koje objekti razmenjuju. Koriste se da bi se dodatno opisala struktura sistema, u situacijama kada dijagrami klasa ne daju dovoljno kvalitetan opis iste.

Dijagrami komponenti

Dijagrami komponenti služe da bi se prikazale komponente sistema. Pod komponentama se podrazumevaju takvi delovi sistema koji se mogu samostalno isporučivati krajnjim korisnicima. Naravno da sve ove komponente moraju biti tako međusobno usaglašene da dodavanje nove komponente u sistem ne izazove poremećaje kompletnog sistema. Uobičajeno svaka komponenta se sastoji od jedne ili više klasa i predstavlja nezavisnu celinu, koja je povezana sa ostatkom sistema pomoću interfejsa. Na ovakav

način se postiže da promene u jednoj komponenti ne deluju destruktivno na ostatak sistema, jer interfejs i dalje čuva integritet komponente i ostatka sistema.

Dijagrami paketa

Paketi predstavlju mehanizme za grupisanje UML elemenata. Iako se najčešće koriste za grupisanje klasa, mogu se koristiti i za grupisanje drugih elemenata, npr. za grupisanje slučajeva upotrebe, za grupisanje entiteta ili za grupisanje komponenti sistema. Predstavljaju se pomoću pravougaonika sa jezičkom u levom gornjem uglu, na kojem je ispisan naziv paketa. Između paketa se povlače relacije zavisnosti, koje govore da se neki od elemenata smeštenih u pakete između kojih postoji zavisnost nalaze u međusobnom odnosu.

Ukoliko govorimo o paketima klasa, tada bi do njihovog kreiranja moglo doći, npr. radi grupisanja hijerarhijske strukture klasa ili grupisanja klasa čije su instance u međusobnoj zavisnosti predstavljenoj na dijagramima sekvenci ili dijagramima saradnje. Moguće je takođe praviti pakete na osnovu stereotipova klasa. Dakle, ukoliko se klase nalaze u logičkoj ili fizičkoj povezanosti moguće ih je smestiti u paket klasa i predstaviti više takvih paketa na dijagramu paketa. Ovi su dijagrami, iako ranije nezvanično korišćeni, novina u UML verziji 2.0.

Dijagrami složene strukture

Dijagrami složene strukture prikazuju saradnju klasa, interfejsa ili komponenti u cilju opisa strukture zadužene za izvršavanje posmatrane funkcionalnosti. Ovi dijagrami su slični dijagramima klasa. Razlika je u tome što dijagrami klasa prikazuju statičku strukturu sistema, kroz prikaz klasa sa njihovim atributima i operacijama, a dijagrami složene strukture prikazuju izvršnu arhitekturu, relacije između njenih gradivnih elemenata i odnos posmatrane arhitekture sa okruženjem, u cilju prikazivanja informacija koji se ne mogu prikazati pomoću statičkih dijagrama. Dijagrami složene strukture su takođe novina u verziji UML-a 2.0.

Dijagrami raspoređivanja

Dijagrami raspoređivanja služe za predstavljaje hardverske arhitekture sistema. Oni prikazuju delove sistema raspoređene po fizičkim lokacijama. Ovi dijagrami se mogu posmatrati i kao prikaz arhitekturalnog rešenja celokupnog sistema.