> Razlika između samog programiranja i softverskog inženjerstva slična je razlici između betoniranja dvorišta i izgradnje mosta.
>
> Eric Braude

1. Softver i softverski inženjering

Proteklo je više od pet decenija razvoja računarskih sistema. Tokom prve tri decenije, primarno je razvijan hardver sa ciljem da se snize troškovi obrade i čuvanja podataka. Osamdesetih godina i dalje se najviše pažnje poklanjalo razvoju hardvera i povećanju njegove brzine i moći. Tek u poslednjoj deceniji razvoja, izazov su postali viši kvalitet i niži troškovi softvera.

Danas softver predstavlja ključ uspeha većine računarskih sistema i ujedno faktor diferencijacije organizacija koje ga poseduju. Softver je postao bitna komponenta u poslovnom odlučivanju i osnova u naučnim istraživanjima i inženjerskom rešavanju problema. On takođe predstavlja značajnu komponentu u industrijskim, transportnim, medicinskim, telekomunikacionim, vojnim i brojnim drugim vrstama sistema. U savremenom svetu softver je praktično neizbežan i svuda prisutan. Na početku 21. veka izmenjeno je poimanje softvera i isti javnost prihvata kao tehnološku stvarnost u budućem razvoju.

Društvo informatičara Nemačke sprovelo je pre nekoliko godina, među mlađim eksperima u informatici, istraživanje u kojem su oni navodili koje discipline sa studija (koje su imali ili bi trebalo da su imali) smatraju najvažnijim za svoj stručni rad. Dobijeni su sledeći rezultati, dati u obliku rang-liste, pri čemu viši rang označava i veću značajnost:

1. timski rad,
2. upravljanje projektima,
3. softverski inženjering,
4. vođenje tima,
5. baze podataka,
6. retorika,
7. komunikacioni sistemi i mreže,
8. sistem kvaliteta,
9. strukture podataka i algoritmi,

10. operativni sistemi,
11. ergonomija softvera,
12. upravljanje poslovanjem,
13. osnovi algoritama,
14. osnovi matematike i logike,
15. distribuirani sistemi,
16. metode naučnog istraživanja,
17. bezbednost podataka,
18. informacioni sistemi,
19. koncepti programskih jezika i
20. arhitektura računara.

Na ovoj listi se softverski inženjering nalazi veoma visoko, na 3. mestu. Ako pogledamo njegov širi kontekst, možemo primetiti da od prvih 8 disciplina, čak 5 spadaju u discipline softverskog inženjerstva, retorika je disciplina čiji se segmenti takođe vrlo često prezentiraju u okviru seminara iz softverskog inženjerstva, baze podataka se vrlo snažno prožimaju sa njim, dok komunikacioni sistemi i mreže predstavljaju neophodnu sistemsku osnovu za softverske inženjere. Treba takođe napomenuti i da se koncepti programskih jezika nalaze tek na 19. mestu, što nedvosmisleno ukazuje da maštanja početnika u ovom poslu ne nailaze na realnost u kasnijoj praksi.

1.1. Pojmovno određenje softvera

Pre trideset godina, manje od 1% javnosti je bilo u mogućnosti da inteligentno objasni značenje pojma "softver". Međutim, sada sve veći deo te javnosti smatra da i razume šta je to softver i kako se on razvija. Da li je to stvarno tako?

Softverski proizvod (engleski: consumer software package) predstavlja softversku podršku računarskim sistemima, a čine ga skup računarskih programa, datoteke i odgovarajuća dokumentacija, namenjeni realizaciji određenih zadataka (definicija preuzeta iz standarda JUS ISO 9127/94). Za razliku od softverskog proizvoda, softver se sastoji iz skupa računarskih programa i datoteka sa istom namenom koju ima softverski proizvod. Kako su prema usvojenim standardima softver i odgovarajuća prateća dokumentacija nerazdvojivi, upakovani za prodaju kao jedinstvena celina i prodaju se zajedno, u nastavku se ova dva pojma poistovećuju.

Iz izloženog se vidi da softverski proizvod predstavlja jedinstvo računarskih programa, struktura podataka sadržanih u datotekama, i dokumentacije koja opisuje način funkcionisanja i upotrebe programa. Pri tome, posebno se ističe značaj dokumentacije, koju projektanti i programeri često zanemaruju, ali bez koje

se softver ne može smatrati kompletnim. Informacije sadržane u pratećoj dokumentaciji su često jedino sredstvo putem koga proizvođač softvera i/ili distributer mogu komunicirati sa kupcem ili korisnikom. Samo ukoliko dokumentacija sadrži dovoljno informacija, korisniku se omogućuje uspešno korišćenje softvera.

Za pojam softvera vezan je i pojam softverska podrška. Ona predstavlja rad na održavanju softvera i prateće dokumentacije u funkcionalnom stanju. Mogu je pružati: proizvođač (organizacija koja je razvila softver), predstavnik (organizacija koja plasira softver na tržištu), distributer (organizacija koja neposredno prodaje softver korisniku) ili neka druga organizacija.

1.2. Paradigme arhitekture softvera

Primena računara u naučno-istraživačkom radu prouzrokovala je ovladavanje složenim tehnološkim procesima i novim proizvodima, među kojima se nalaze i kvalitetniji računari. Ova povratna sprega dovela je do eksplozije znanja kojom su nastale hardverske arhitekture izuzetno visokog kvaliteta. Razvoj softvera, međutim, nije pratila opisana dinamika, jer se on mogao meriti porastom od svega nekoliko procenata godišnje. Poboljšanje performansi u razvoju i korišćenju informacionih sistema javljalo se skokovito, sa pojavom novih paradigmi arhitekture softvera.

U početnom periodu primene računara, od 1948. do 1965. godine, hardver je bio izuzetno skup i ljudski rad potreban za razvoj softvera je bio zanemarljivo mali u odnosu na hardver, pa je softver bio u potpunosti prilagođen hardveru. Ovaj period nazivamo faza monolitne programske podrške i u njoj su programi proistekli iz opisa postupaka bili čvrsto vezani za ulazno-izlazne uređaje računara i njegov monitorski program. Programi su pisani u mašinskom jeziku ili asembleru i to tako da njihovo vreme rada bude što kraće. Korisnički interfejs je bio vezivan za određeni hardver i sadržao je male mogućnosti, a njegovo korišćenje je bilo krajnje nekomforno i svodilo se na električnu pisaću mašinu u funkciji konzole.

Povratna sprega nastala razvojem softvera uticala je na značajno sniženje cena hardvera, poboljšanje njegovih performansi i pojavu niza gotovih programskih proizvoda koje su, zajedno sa hardverom, nudili njegovi proizvođači. Ovako formirani interfejs poznat je pod nazivom sistemski softver i predstavlja prvi korak u razvoju korisničkih interfejsa. Tako je nastala faza poznata pod nazivom dihotomija aplikacija - sistemski softver i karakteristična je za period od 1965. do 1985. godine. Deo sistemskog softvera bliži hardveru naziva se operativni sistem, a deo bliži aplikacijama - pomoćni programi. U ovoj fazi pojavljuje se multiprogramski rad. Ova promena paradigme podigla je produktivnost programera i smanjila troškove za njihovu obuku, koji u ovoj fazi koriste više programske jezike i biblioteke napisanih delova programa. Najvažnija karakteristika ove faze je mogućnost prenosa napisanih aplikacija sa jednog računara na drugi i to istog ili različitih proizvođača. Korisnici počinju da se dele na profesionalce za

razvoj programa - programere i na krajnje korisnike, koji u ovoj fazi postaju aktivni, ali ne i dominantni. Korišćenje vizuelnog alfanumeričkog komuniciranja pomoću terminala u interaktivnom režimu i razmena podataka pomoću telekomunikacija sa udaljenim računarima postaju standardi obrade podataka u ovoj fazi.

Tabela 1: Pregled paradigmi arhitekture softvera i njihovih glavnih osobina

Naziv paradigme	Glavne osobine
faza monolitne programske podrške	▪ proces obrade vezan za U/I uređaje, ▪ programi u mašinskom jeziku ili asembleru, ▪ interfejs nekomforan i malih mogućnosti
dihotomija aplikacija – sistemski softver	▪ pojava multiprogramskog rada, ▪ mogućnost prenosa programa, ▪ interaktivno komuniciranje i razmena podataka pomoću telekomunikacija
okruženje blisko korisniku	▪ kupovina gotovih rešenja, ▪ pojava grafičkog korisničkog interfejsa, ▪ aplikacije kao konfekcijski proizvodi
računarske mreže	▪ povezivanje računara sa različitim hardverom i operativnim sistemima, ▪ korišćenje Interneta, ▪ globalizacija informacionih sistema
globalni multimedijalni informacioni sistem	▪ upotreba različitih tipova informacija, ▪ interaktivna isporuka informacija i pratećih programa, ▪ jedinstveno praćenje informacija iz svetskog okruženja

Osamdesetih godina, sa pojavom personalnog računara, dolazi do nove paradigme arhitekture programske podrške, okruženja bliskog korisniku. Karakteristika ovog perioda je kupovina gotovih rešenja, što znači da korisnici kupuju mikroračunare sa skupovima gotovih aplikacija koje služe za automatizaciju kojom se podiže produktivnost rada korisnika. Pad cena je drastičan i sada su troškovi eksploatacije personalnog računara daleko ispod cene rada korisnika. U takvoj situaciji aplikacije moraju korisniku pružiti okruženje slično onom na koje je navikao. Veći deo hardvera i softvera preuzimaju ulogu interfejsa između korisnika i aplikacije koji treba da ispunjava njegove želje. Tako nastaje grafički korisnički interfejs zasnovan na korišćenju "ikona" koji nije karakterističan samo za hardver (ekrani u boji visoke

rezolucije i grafičke kartice), već i za softver (Windows). Aplikacije počinju da se prodaju kao konfekcijski proizvodi, čijim se kombinovanjem vrši automatizacija određenog poslovnog okruženja.

Pojavom lokalnih računarskih mreža personalni računar postaje komparativno najatraktivniji za realizaciju informacionih sistema odeljenja ili organizacija manje ili srednje veličine. Razmena podataka sa širim okruženjem vrši se pomoću telekomunikacione računarske mreže u koju se, pomoću transportnih komunikacionih protokola, povezuju računari sa različitim hardverom i operativnim sistemima. Paradigmu računarske mreže karakteriše podela aplikacije na sistemski deo (ljusku) i korisničku, distribuiranu aplikaciju. Između velikog broja modela povezivanja distribuiranih aplikacija u primeni je dominantan TCP/IP, koji koristi najveća svetska mreža Internet. Mreže ovog tipa predstavljaju osnovu za globalne informacione sisteme, kod kojih sve informacije, bez obzira na geografski položaj, postaju pristupačne svim korisnicima u svetu.

Tehnologija skladištenja, povezivanja i organizovanja različitih tipova informacija (tekstovi, numeričke vrednosti, tabele, grafici, slike, zvuk i video zapisi) zasnovana je na World Wide Web (WWW) servisu, TCP/IP protokolu, jeziku za označavanje hipertekstualnih informacija HTML i aplikativnom transportnom protokolu HTTP. Ona omogućava korisnicima da sve informacije, koje se nalaze na računarima raspoređenim po celom svetu i povezanim u Internet, vide kao jedan globalni, distribuirani, multimedijalni informacioni sistem. Kreiranje efikasne infrastrukture za trenutnu isporuku uskladištenih informacija na proizvoljno mesto uključilo je i distribuciju pratećih programa kao specifičnih podataka. Ovu distribuciju programa moguće je realizovati razvojem programa za virtuelni računar i simulatorima ovog virtuelnog računara na različitim platformama (SunSoft Java) ili razvojem različitih verzija istog programa za različite platforme (Microsoft).

Pomaci u aplikativnoj paradigmi prouzrokovani primenom WWW servisa ukazuju da će se u skoroj budućnosti za dominaciju boriti mrežni računar NC (grafički interfejs, procesor, operativna memorija i komunikacioni procesor) i Microsoftov NetPC, koji omogućava naizmenično korišćenje mrežnog i personalnog računara. Prikazane paradigme arhitekture softvera mogu se posmatrati i kroz četiri faze razvoja softvera.

Prvu, najraniju etapu karakteriše razvoj softvera koji su realizovali proizvođači računarske opreme, razvijajući kompletan neophodni softver (operativne sisteme, programske jezike, pomoćne programe i dr.) za svoje proizvode. To je bio softver batch orijentacije, projektovan i razvijan za pojedinačne potrebe naručilaca. Ceo postupak od inicijalizacije, preko projektovanja do implementacije i održavanja softvera je obavljala ista osoba, te projektno-programska dokumentacija nije postojala. U računarskim centrima je bila realizovana koncentracija vrhunskih kadrova značajnih za razvoj softvera.

Međutim, ubrzani razvoj računarske opreme, uslovio je da se menjaju i zahtevi korisnika prema softveru. Viši zahtevi u odnosu na softver isticali su potrebu za višim nivoom znanja onih koji softver razvijaju. Sopstvenim snagama ove organizacije su sve teže mogle da prate takav razvoj. Viši troškovi razvoja softvera uslovljavali su i postepeni rast zahteva za sredstvima odnosno kapitalom potrebnim za razvoj. Samo veoma velike računarske kuće su dozvoljavale sebi taj luksuz i pravo da održavaju računarske centre u celini i da se bave istovremeno razvojem, kako računarske opreme, tako i softvera.

Drugu etapu karakteriše softver razvijen za višekorisnički rad u on-line režimu rada i pojava prvih sistema za upravljanje bazama podataka. Sedamdesetih godina se pojavljuju prve softverske kuće. One su u početku razvijale specifični softver za određenu računarsku opremu, da bi se kasnije usmerile ka razvoju univerzalnijeg softvera upotrebljivog na različitoj opremi. U ovim organizacijama se koncentrisalo kvalitetno osoblje, koje je timskim radom i dobrom koordinacijom poslova dolazilo do najkvalitetnijih rešenja.

Kada bi se želeli ukratko navesti faktori, koji su uslovili potrebu novog načina razvoja softvera, onda bi to bili:

- eksplozija informacija,
- pojava softverskih kuća,
- raznovrsna i brojna primena računara,
- potreba za softverom koji se lako upotrebljava i nije skup,
- brze i dinamične promene u društvu,
- zahtevi za visokim nivoom kvaliteta softvera,
- brza realizacija, odnosno kraće vreme dostupnosti i
- zahtevi za univerzalnom primenom.

Treću etapu razvoja karakteriše softver za podršku distribuiranih računarskih sistema koji su međusobno komunicirali i pojava personalnih računara, koja je još više iskazivala potrebu da se izmeni način razvoja softvera, odnosno potrebu da se brzo i jeftino dolazi do univerzalnih i opšte primenljivih programa. Javlja se veći uticaj i potreba korisnika za jedinstvenim načinom korišćenja softvera i smanjenjem troškova u primeni računara. Takođe, pojavljuju se i snažnija sredstva za razvoj softvera.

Softver je postao opšte prihvaćeni pojam, a potreba za računarskom opremom sve veća. Počela je realizacija industrijske proizvodnje softvera, umesto, do tada aktuelne, pojedinačne zanatlijske proizvodnje. Softver se proizvodi na identičan način kao i svi ostali industrijski proizvodi. Na kraju XX veka, razvoj softvera predstavlja značajnu industrijsku granu.

Tržište je postavljalo takve zahteve i imalo očekivanja prema softveru, kakve nije mogla zadovoljiti pojedinačna zanatska već samo industrijska proizvodnja. To su:

- zadovoljenje raznovrsnih potreba odnosno opšta primenljivost softvera,
- razvoj softvera na kompleksan način uz računarsku podršku i CASE tehnologije,
- uobičajeno primenjen model prototipskog razvoja softvera,
- realizacija jeftinijeg softvera za korisnika obzirom da se ukupni troškovi dele na veliki broj kupaca,
- mnogo kraće vreme realizacije softvera,
- savremeniji i delotvorniji softveri koji se uobičajeno realizuju u softverskim kućama sa kompetentnijim i stručnijim osobljem i
- visok nivo kvaliteta softvera.

Četvrtu etapu razvoja softvera karakterišu dalji razvoj softvera za primenu snažnih personalnih računara, računara povezanih u mreže i paralelnu obradu. Takođe, iz eksperimentalne u široku upotrebu se stavljaju nove vrste informacionih sistema kakvi su ekspertni sistemi, veštačka inteligencija i neuralne mreže, kao i objektno orijentisane tehnologije. Posebno značajno za ovu poslednju etapu u razvoju je rušenje mita po kome samo softverske kuće razvijaju računarske programe. Naime, korisnici primenom tehnika četvrte generacije i sami razvijaju softver i menjaju ustaljenu praksu i način razvoja softvera.

1.3. Osobine softvera

Treba reći da se u praksi vrlo često sreće softver niskog kvaliteta. Tako veliki broj njegovih korisnika, ali i eksperata koji se bave njegovim razvojem, smatra da je softver sklon greškama, neočekivanog je ponašanja, skoro uvek je skuplji od očekivanog, završen je prekasno i čak je i neprimenljiv. U praksi je poznat niz vrlo loših implementacija softvera, a ovde se pominju nekoliko karakterističnih promašaja:

- aerodrom u Denveru nije mogao biti otvoren 1994. godine nakon završetka izgradnje, zbog problema sa softverom za transport prtljaga,
- Siemens je izgubio milijardu maraka jer je razvoj softvera za knjigovodstvo lekova Fonda zdravstvenog osiguranja Nemačke kasnio,
- Avion F18 se za vreme vežbe 1983. sam okrenuo naglavačke nakon prelaska ekvatora zbog greške u programu koja je nastala pri promeni znaka jednog polja,

- prva raketa na Veneri nije pronašla svoj cilj i napravila je štetu od više stotina miliona dolara, jer je u FORTRAN programu korišćena tačka umesto zareza, a to nije eksplitno bilo deklarisano,
- Deutche Telekom pretrpeo je gubitke od više stotina miliona maraka jer softver nije obračunavao prazničnu tarifu za 1.1.96. i
- takozvani Y2K problem 2000. godine nastao je jer je godina bila registrovana u operativnim sistemima i podacima koji su odlagani kao dvocifreni broj, za čiju su ispravku napravljeni troškovi veći od $5.000.000.000.

Kada govorimo o veličini softvera, tada odmah treba definisati o čemu se tu radi. Tako se pod malim softverom smatra onaj koji sadrži do 2.000 linija koda, srednji je od 2.000 do 100.000 linija, veliki je između 100.000 i 1.000.000 linija koda, dok se pod vrlo velikim smatra onaj od preko 1.000.000 kodnih linija. Da ovo nije samo fantazija, jasno govore sledeće činjenice:

- srednja veličina softvera u 100 najvećih kompanija SAD iznosi oko 35.000.000 linija koda,
- programi Ministarstva odbrane SAD sadrže preko 1.400.000.000 linija koda, distribuirani su u 1.700 računskih centara, a operativni troškovi njihovog održavanja iznose oko 9 milijardi dolara godišnje,
- Windows 2000 sadrži preko 60.000.000 linija koda.

Teorijske osnove industrijskog razvoja softverskih proizvoda opisao je sredinom osamdesetih godina Brad Cox. On perspektive razvoja složenih sistema softverskih proizvoda vidi kao proces povezivanja manje složenih gotovih objekata i komponenti. Do ovakvog načina industrijskog razvoja sve se više dolazi evolucijom iz zanatskog razvoja, koji i dalje postoji, ali prestaje biti dominantan način.

Da bi se mogla organizovati industrijska proizvodnja, potrebno je obezbediti sve komponente koje ulaze u konkretan proizvod. Ti delovi moraju biti međusobno usaglašeni, odnosno neophodno je da postoje standardi po kojima se oni razvijaju. Kako je razvoj softvera vrlo dinamičan, donošenje zvaničnih standarda nije sposobno da ga prati, pa se na tržištu prvo pojavljuju standardi koji predstavljaju zajednički dogovor određene grupe proizvođača ili standardi neke velike korporacije (Microsoft) koji su na tržištu opšte prihvaćeni. Takvi standardi se nazivaju industrijski standardi i predstavljaju osnovu za pokretanje procedure i donošenje zvaničnih standarda od strane međunarodnih ili nacionalnih organizacija koje se bave standardizacijom (npr. International Standard Organisation - ISO, American National Standard Institute - ANSI, Savezni zavod za standardizaciju).

Tržište softverskih proizvoda beleži rast koji nadmašuje rast tržišta hardvera i ta se razlika stalno povećava. Vidljivi su osnovni trendovi koji karakterišu tržište softverskih proizvoda. Uočljiv je stalni trend smanjivanja cena. Razvojne i korisničke mogućnosti softverskih proizvoda najprisutnije su na desktop okruženjima. Sve su češće investicije u skladišta podataka koja obezbeđuju važne informacije i u Internet proizvode. Širi se tržište serverskih operativnih sistema i aplikacija, posebno na klijent-server platformi.

Odnos cena i performansi računara smanjen je za proteklih 20 godina milion puta, a prema Murovom zakonu (Moor) predviđa se isti trend i za dva nastupajuća perioda od po 20 godina, nakon čega se očekuje da će doći do zasićenja i ovaj odnos će biti znatno umanjen. Kao posledice Murovog zakona, ustanovljeni su osnovni softverski zakoni, koji opisuju trendove razvoja softvera:

- softver se širi kao gas, sa stopom rasta od 33,9% godišnje,
- softver će se širiti sve dok ne bude ograničen Murovim zakonom,
- rast softvera omogućava održavanje Murovog zakona o rastu hardvera,
- rast softvera je ograničen jedino ljudskim ambicijama i mogućnostima i
- softver je "nepoderiv" proizvod, što omogućava njegovo korišćenje neograničeno dugo (iz čega je proizišao i navedeni problem 2000. godine, jer niko nije očekivao da će se softverska rešenja tako dugo održati u produkciji).

Najznačajnije posledice ovih zakona su da će razvoj softvera predstavljati najvažniju industrijsku granu i u 21. veku i da će se softverska kriza, o kojoj će biti reči u sledećem poglavlju, i dalje nastaviti.

1.4. Pojmovno određenje softverskog inženjeringa

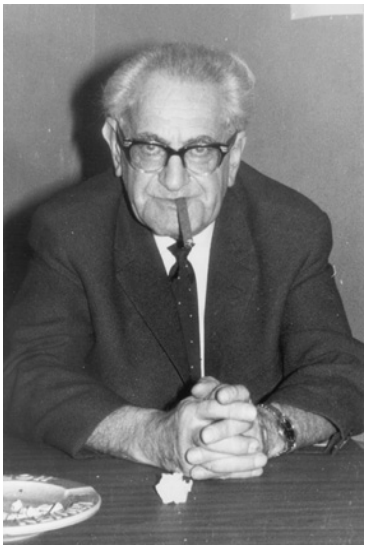
Pojam softverski inženjering se prvi put pominje krajem šestdesetih godina na konferenciji o krizi softvera, prouzrokovanoj trećom generacijom računara (pre svega IBM S/360), koja je svojom snagom inicirala razvoj velikih softverskih sistema. Pošto za ove sisteme nisu postojali odgovarajući teorijski modeli razvoja, realizacija projekata pojedinih informacionih sistema je trajala izuzetno dugo, kasnila po više godina, što je višestruko uvećavalo planirane troškove razvoja, komplikovalo održavanje, a sama realizacija je bila na vrlo niskom nivou. Tako je počeo rad na teorijskim aspektima metodologija razvoja softvera, kasnije na njihovoj primeni u praksi, da bi se danas jasno formirala disciplina koja proučava razvoj informacionih tehnologija, poznata pod nazivom softverski inženjering.

Softverski inženjering nema za cilj samo izradu softverskog proizvoda, nego i razvoj na troškovno delotvoran način, koji podrazumeva da se sa konačno definisanim obimom resursa u predviđenom roku postigne proizvod visokog

kvaliteta. Pri tome, kritični kriterijumi kvaliteta koji određuju dobro razvijen softverski proizvod su:

- održavanje - mogućnost prilagođavanja softvera izmenjenim zahtevima kupca,
- odgovornost - pouzdanost, sigurnost, bez štete uslovljene greškama u sistemu,
- efikasnost - bez nepotrebnog korišćenja resursa sistema i
- upotrebljivost - sa odgovarajućim korisničkim interfejsom i adekvatnom dokumentacijom.

U zavisnosti od karaktera sistema koji bi se razvijao i predmeta razvoja, mogu se razlikovati različite grane u razvoju softvera: softverski inženjering, sistemski inženjering, informacioni inženjering i inženjering znanja.



Fritz Bauer

Softverski inženjering je prvi definisao Fritz Bauer još 1968. godine na konferenciji koju je organizovao Naučni komitet NATO. Upotrebio ga je kao termin koji označava primenu principa inženjeringa u cilju ostvarivanja ekonomičnog softvera, efikasnog i pouzdanog u realnosti na računarskim mašinama. Nakon toga su nastale brojne definicije, ali su sve one posebno naglašavale potrebu primene inženjeringa u razvoju softvera.

Tako je u standardu ISO 9000-3 softverski inženjering formulisan kao definisan proces, korak po korak, koji omogućava specifikaciju, dizajn, implementaciju i testiranje softverskih rešenja grupe zahteva na najefikasniji i najdohodovniji način.

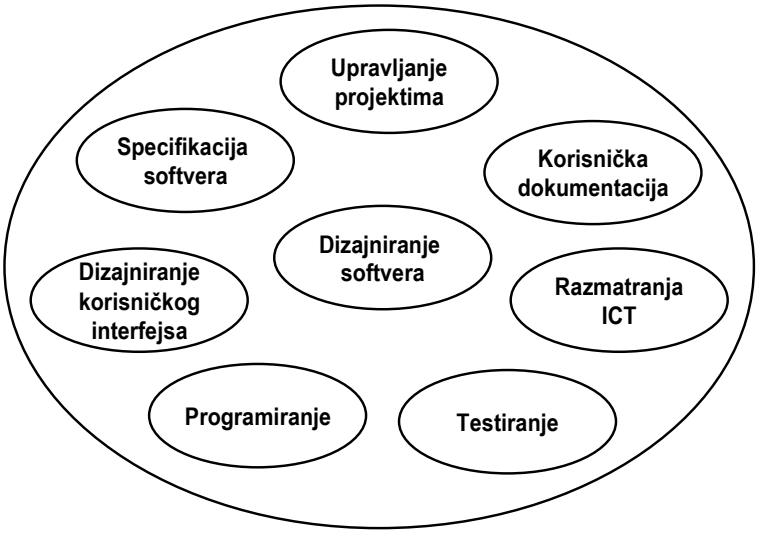
Najkraću i najrazumljiviju, ali i vrlo široku definiciju navode Pagel i Six, prema kojoj softverski inženjering predstavlja usmerenje ka ekonomičnom razvoju softvera visokog kvaliteta.

Sa druge strane, Sommerville pozicionira softverski inženjering znatno snažnije u razvoj poslovnih aplikacija, navodeći da je on inženjerska disciplina koja je usmerena na praktične probleme razvoja velikih softverskih sistema.

Najpreciziju definiciju softverskog inženjeringa nalazimo u standardnom rečniku termina softverskog inženjerstva koji je 1990. godine objavio IEEE (Institute for Electronics, Energetics and Engineering), a prema kojem softverski inženjering podrazumeva primenu sistematičnog, disciplinovanog i merljivog pristupa razvoju, uvođenju i održavanju softvera, tj. primeni inženjeringa na softver.

Autori ove knjige smatraju da bi se na osnovu izloženog, softverski inžinjering mogao definisati kao stroga primena inženjeringa, naučnih i matematičkih principa i metoda u ekonomičnoj proizvodnji kvalitetnog softvera.

Slika 1: Aktivnosti softverskog inženjerstva



Opštiji pojam od pojma softverski inženjering je sistemski inženjering. On obuhvata aktivnosti specificiranja, projektovanja, primene, provere, instaliranja i održavanja sistema kao celine. Sistemski inženjering podrazumeva i niz pratećih komponenti koje nisu strogo informatičke, a koje omogućuju funkcionisanje sistema. Da bi se realizovao proces sistemskog inženjeringa kao niz interdisciplinarnih aktivnosti, potrebno je angažovati niz timova sa različitim znanjima i ulogama, koji će zajednički definisati sistem. Kada je sistem jednom definisan, ne preporučuje se ulazak u redizajn sistema bez preke potrebe, jer su sve kasnije izmene izuzetno komplikovane i skupe.

1.5. Struktura softverskog inženjeringa

Strukturu softverskog inženjeringa sačinjavaju tri ključne komponente: metodi, alati i postupci (procedure). Njihovo jedinstvo opredeljuje kvalitet razvoja, pa je zbog toga značajno da se odaberu one komponente koje se postavljenim zadacima i problemima u razvoju najlakše prilagođavaju. Donošenje prave odluke nije jednostavan zadatak, i to zbog:

- individualnog i kreativnog karaktera postupka razvoja,
- različitosti pojedinih sistema za koji se razvija softver i različitog okruženja sistema,
- nepostojanja recepta za izbor.

Značajno je istaći da se za rešavanje konkretnog zadatka mogu izabrati najpogodnije komponente samo ukoliko projektanti poznaju njihove mogućnosti, poznaju njihove alternative i znaju ih uspešno primeniti.

a) Metodi

Metodi predstavljaju neophodan i uz određene pretpostavke propisani sistematski način na koji se izvršavaju pojedini zadaci softverskog inženjeringa. Metodi pokrivaju široki spektar zadataka među kojima su: planiranje i procenjivanje projekata, analiza sistemskih i softverskih zahteva, projektovanje strukture podataka, definisanje arhitekture programa, kodiranje, testiranje i održavanje. Oni su bazirani na jednom ili više principa realizacije. Proces razvoja softvera je celishodno realizovati primenom određenih metoda. Izabrati odgovarajući metod koji će se primeniti u razvoju nije jednostavno, obzirom da su raspoložive brojne mogućnosti. Metodi se mogu međusobno kombinovati nezavisno od toga koji je metod najpogodniji za rešavanje datog problema. Najznačajnija težnja prilikom pravilnog izbora metoda je minimizacija troškova razvoja i obezbeđenje visoko kvalitetnog proizvoda.

Metodi u softverskom inženjeringu uvek uvode specijalne jezičke ili grafičke notacije i grupe kriterijuma u definisanju kvaliteta softvera. Oni treba da budu nezavisni od područja primene. Drugim rečima, moraju obezbediti rešenje

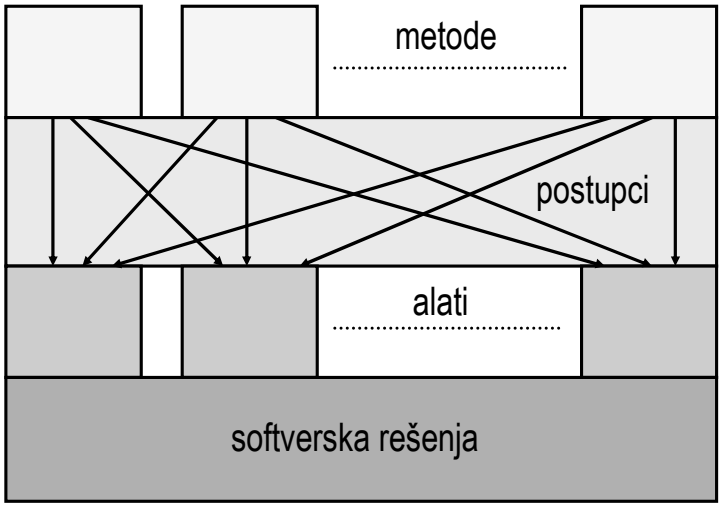
različitim problemima, na različite načine. Metodi razvoja su know-how koji se može uspešno primeniti u razvojnom procesu pri izvršenju pojedinih aktivnosti.

Naravno, ne postoji takav razvoj kojem nedostaju principi i metodi, ali ne postoje ni metodi koji se mogu primeniti u rešavanju svakog problema. U tom smislu, primarni i veoma odgovoran zadatak je izabrati za korisnika, sistem i proces razvoja najpogodniji i najprilagodljiviji metod. Pravilan izbor ne podrazumeva definisanje jednog određenog metoda, već u zavisnosti od kompleksnosti i vrste problema kombinaciju više različitih metoda.

Metodi razvoja softvera se mogu grupisati prema različitim kriterijumima, ali ih je veoma teško jednoznačno razvrstati. Najčešće se ipak grupišu s obzirom na faze razvoja softvera u kojima se primenjuju.

Kao izraz potreba za rešavanjem sve složenijih problema putem računara, razvijene su metodologije. One predstavljaju jedinstveni sistem metoda, čijim povezivanjem je omogućeno projektantima da na sistematizovani način pokriju više ili sve faze razvoja softvera.

Slika 2: Struktura softverskog inženjeringa



b) Alati

Alati obezbeđuju automatizovanu ili poluautomatizovanu podršku u primeni metoda. Oni predstavljaju pomoć neophodnu da bi se automatizovale aktivnosti razvoja softvera kao: upravljanje projektom odnosno planiranje, procenjivanje, terminiranje, raspoređivanje, modeliranje, analiza, projektovanje, kodiranje,

dokumentovanje, testiranje, integracija elemenata sa sistemom, upravljanje konfiguracijom, kontrola kvaliteta softvera, upravljanje podacima i dr.

U stvarnosti, danas svaki metod poseduje određeno pomoćno sredstvo, instrument, alat. Kada su alati na takav način integrisani u jedan sistem da se rezultati kreirani od jednog alata mogu upotrebiti i od drugog alata, tada govorimo o CASE alatima.

CASE alati čine posebnu grupu alata, koji automatizuju metode i procedure razvoja softvera i istovremeno skraćuju vreme, smanjuju troškove izrade i podižu kvalitet razvijenog softverskog proizvoda.

c) Postupci

Postupci predstavljaju "lepak" koji povezuje metode i alate. Postupkom se naziva niz konkretnih koraka koje je potrebno izvršiti prilikom rešavanja datog problema ili grupe problema primenom određenog metoda. Postupak može imati alternative, a međusobnim povezivanjem više postupaka mogu se kreirati i potpuno novi postupci. Postupci definišu:

- redosled izvođenja metodoloških koraka i primene pojedinih metoda,
- koji se rezultati trebaju realizovati u pojedinim metodološkim koracima,
- kakve kontrole treba ugraditi u razvoj softvera u cilju obezbeđenja kvaliteta ili koordinacije izmena i
- putokaze za softver menadžere u ocenjivanju izvršenog razvoja.

Za primenu određenog postupka se takođe ne može dati recept, jer priroda sistema koji se razvija i očekivanja korisnika se međusobno značajno razlikuju.

1.6. Ciljevi i značaj softverskog inženjeringa

Softverski inženjering je u stanju permanentnih, brzih izmena. Krajem 1998. godine registrovano je zanimanje softverski inženjer, i to prvi put u Teksasu. Izvršena istraživanja ukazuju da je oko 75% razvojnih softverskih kuća u SAD na primitivnom nivou. Ako se zna da je razvoj softvera u ovoj zemlji verovatno na najvišem nivou u svetu, ovaj podatak pokazuje kakav razvojni put tek predstoji disciplinama softverskog inženjeringa, dok je primena danas njegovo još bolnije mesto.

Značaj inženjeringa je u sistematskom pristupu razvoju, uvođenju i održavanju softvera kroz faze njegovog životnog ciklusa. Treba posebno naglasiti da ne postoji jedno najbolje rešenje ili pristup u rešavanju problema razvoja softvera. Ipak kombinovanjem velikog broja široko obuhvatnih metoda u svim fazama razvoja softvera, kvalitetnih alata za automatizovanje ovih metoda, snažnih blokova za implementaciju softvera, boljih tehnika za obezbeđenje kvaliteta softvera i što je najvažnije filozofijom koordinacije, kontrole i upravljanja, može se postići

disciplina u razvoju softvera ili softverski inženjering. Primena softverskog inženjeringa uključuje:

- izradu pouzdanog softvera,
- izradu softvera koji radi po originalnim specifikacijama,
- izradu softvera koji je prilagodljiv promenama u svim fazama životnog ciklusa,
- izradu softvera koji se može ponovo koristiti na različitim sistemima,
- razvoj zavisan od raspoloživih resursa,
- razvoj izveden na vreme,
- efektivan razvoj sa aspekta troškova,
- razvoj koji zadovoljava korisničke zahteve i
- razvoj koji obezbeđuje kvalitetan proizvod.

Softverski razvoj kao nauka i kao praktična aktivnost imaju nešto zajedničko – raznovrstnost. U cilju uređenja ove raznovrsnosti, softverski inženjering predstavlja strogo definisan pristup koji čine principi i ciljevi koji se primenjuju u različitim fazama razvoja softvera. Polazno odredište za određivanje ciljeva softverskog inženjeringa može se naći u takozvanom **4P** pravilu, prema kojem je cilj svakog **p**rojekta razvoja softvera da stvori softverski **p**roizvod u kojem **p**rojektanti stvaraju u efektivnom **p**rocesu. Glavni ciljevi softverskog inženjeringa su: ostvarenje pouzdanosti, promenljivosti, jasnosti, prilagodljivosti, ponovljivosti, efikasnosti, prenosivosti i mogućnosti održavanja softvera.

Koncept softverskog inženjeringa u realizaciji ciljeva poseduje veoma značajne prednosti u odnosu na zanatlijski - pojedinačno razvijeni proizvod, a to su sledeće:

- troškovi razvoja softverskih proizvoda su veoma visoki ali je proizvod, obzirom na brojnost primene, jeftin,
- vreme raspoloživosti softverskih proizvoda je kraće, odnosno period vremena od nastanka ideje do njene realizacije je skraćen i
- rezultat razvoja su softverski proizvodi koje karakteriše visok nivo kvaliteta odnosno ovakav način razvoja softverskih proizvoda predstavlja garanciju kvaliteta.

Prema tome, pojedinačno razvijeni proizvod je skup, njegov razvoj traje dugo, kupac/korisnik mora čekati da mu se proizvod razvije i nije siguran da će dobiti proizvod koji i očekuje. Tek na kraju razvojnog perioda će videti da li proizvod ispunjava njegova očekivanja.

1.7. Karakteristike softverskog inženjeringa

Karakteristike softverskog inženjeringa kakve su jedinstvenost i kompleksnost su opšte kategorije koje opredeljuju proces razvoja softverskih proizvoda. Međutim, određene kategorije konkretnih zahteva uključuju i druge karakteristike kao što su: pouzdanost, dopuštenost grešaka (tolerantnost), performantnost i kompatibilnost. Kod postojećih sistema, posebna pažnja se posvećuje i odlučivanju kao odgovarajućem procesu u životnom ciklusu, kako u tehničkom tako i u upravljačkom smislu.

Planiranje razvoja mora uzeti u obzir i razmatranje tehničkih karakteristika koje su potrebne u svakom pojedinom projektu sistema. To su:

- karakteristike sistema (veličina, kompleksnost, savremenost),
- karakteristike zahteva (priroda, stepen definisanosti),
- potrebna obuka,
- troškovi projekta,
- raspoložive svrsishodne metodologije/metodi,
- savremeni CASE proizvodi,
- tehnike efektivnog upravljanja projektom i dr.

Što se tiče modela procesa, oni moraju uključivati i procese razmatranja razloga grešaka, definisanja okvira grešaka i definisanja šema načina samokorekcije. Primenjeni modeli životnog ciklusa zahtevaju i realizaciju osobina prilagodljivosti, fleksibilnosti i tolerantnosti.

Navedene karakteristike određuju: karakter rizika, ograničenja, upotrebu modela procesa, zahtevanu tehnologiju razvoja, primenljivost standarda i organizacionu odgovornost.

Primarna osobina dobrog sistema softverskog inženjeringa je da finalni proizvod postigne ciljeve softverskog inženjeringa koji zadovoljavaju potrebe i zadovoljavaju kupca. Softverski inženjering koji uključuje maksimum mogućih ciljeva i principa softverskog inženjeringa poseduje i sledeće osobine:

- funkcionalnost - korektno i tačno,
- performantnost - vreme pristupa, protok i brzina,
- efektivnost,
- temeljitost u analizi zahteva,
- robustnost - mogućnost ponovne upotrebe proverenih komponenti,
- upotreba odgovarajućih metodologija,

- odgovarajuća dokumentacija za operativni rad i održavanje,
- upotreba CASE alata,
- sposobnost prilagođavanja promenama tokom evolucije sistema,
- dobro definisani korisnički interfejs,
- lakoća upotrebe softvera i dr.

Među navedenim karakteristikama, posebno su značajne one karakteristike softverskog inženjeringa koje ukazuju na mogućnost prototipskog razvoja softvera i na mogućnost ponovnog korišćenja razvijenih delova ili celine softvera.

1.8. Razvoj softvera u praksi

Softverski inženjering predstavlja opšti inženjerski model sistemski organizovanog rešavanja problema u razvoju i održavanju softvera. U praksi softverskog inženjerstva pojavljuje se niz problema sa kojima se inženjeri svakodnevno suočavaju. Među njima su najočigledniji:

- Korisnik se bavi svojim redovnim i svakodnevnim poslom i nikada nema vremena za druge, govori žargonskim jezikom vezanim za svoju struku, ne veruje u promene, a pogotovu one koje će doneti softver koji se razvija, pun je predubeđenja i jedva čeka da završi sa radom na razvoju softvera koji mu je nametnut.
- Pregovori sa kupcem pre početka razvoja obično izgledaju tako da korisnik misli da će dobiti jako malo, da je preskupo platio i da je suviše dug rok do završetka softvera, dok je razvojni tim ubeđen da je ugovoreno suviše posla za malu cenu i da je rok neodrživ, jer se u tako kratkom roku može jako malo uraditi.
- Česte izmene želja korisnika su deo realnosti, jer korisnici evoluiraju u mišljenju u toku razvoja, a sa druge strane zakonski i organizacioni ambijent takođe stalno nameću promene.
- Pogrešna tumačenja su posledica toga da se stručne terminologije korisnika i softverskog inženjera razlikuju i svaki od njih podrazumeva da onaj drugi zna mnogo više o njegovom poslu.
- Kompleksan posao po pravilu znači i da će program biti kompleksan.
- Softver je i nakon testiranja i implementacije prepun grešaka i za njihovo ispravljanje neophodno je izdvojiti dosta vremena i uložiti velike napore.
- Veliki pritisci na održanje rokova su posledica korisnikovih realnih potreba, ali i pogrešnog sagledavanja obima posla sa bilo koje od strana koje učestvuju u razvoju.

- Timski rad je novi ambijent, sa kojim softverski inženjeri nisu dovoljno familijarni.
- Veliki obim dokumentacije prati kompleksnost problema koji se rešavaju. Softverski inženjeri najčešće beže od njenog kreiranja, dok je kasnije njeno održavanje vrlo retko ozbiljno organizovano.
- Programiranje je samo 20% aktivnosti u razvoju softvera, što predstavlja vrlo neprijatno iznenađenje za većinu novih softverskih inženjera, koji svoj posao zamišljaju tako što će im neko pripremiti šta će programirati, a oni će onda sami za računarom napisati taj program. Naravno, realnost je obično sasvim suprotna.
- Dugogodišnje korišćenje softvera (10-25 godina) znači da će se softverski inženjeri suočavati sa sličnim problemima još dugo godina, što znači da se najverovatnije neće menjati ni informacione tehnologije koje su korišćene u razvoju.
- Izmene softvera nakon isporuke su realnost svakog uspešnog softverskog rešenja. Ove izmene će vremenom degradirati mnoge osobine sistema, koji će postajati sve nejasniji za nove izmene.

Ovo su problemi sa kojima se softverski inženjeri svakodnevno suočavaju i treba reći da se većina njih uspešno nosi sa njima. Organizovan pristup u razvoju, prema pravilima softverskog inženjerstva, značajno im pomaže u tome.

> Kada razmišljamo o beskrajnom rastu i propadanju života i civilizacija, ne možemo da se otmemo utisku apsolutne ništavnosti. Pa ipak, nikada nisam izgubio osećaj za nešto što raste i proživljava večni tok.
>
> Karl Gustav Jung

2. Životni ciklus softvera

Razvoj softvera predstavlja ciklus aktivnosti u razvoju, korišćenju i održavanju softvera. Tokom života, softver prolazi kroz više faza razvoja: od začetka, preko inicijalnog razvoja, produktivnog funkcionisanja, održavanja do povlačenja.

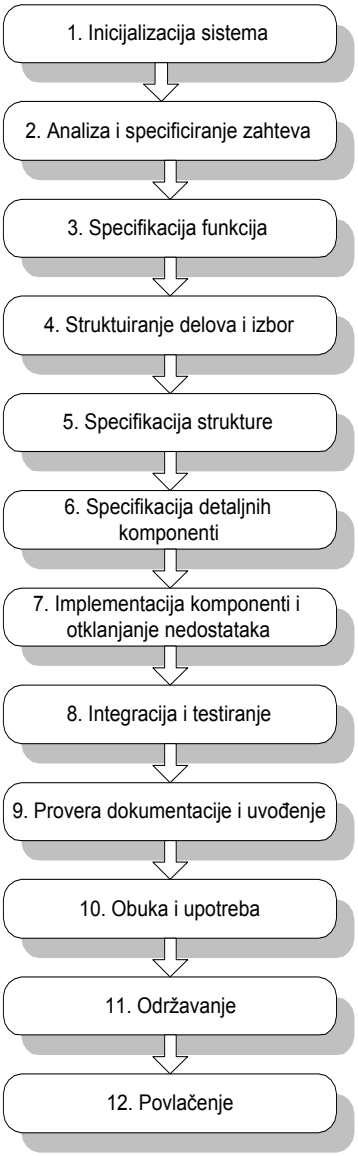
Životni ciklus razvoja softvera mogao bi biti karakterisan sledećim aktivnostima:

- Inicijalizacija sistema je aktivnost u kojoj se navodi poreklo softvera. Primarni cilj inicijalizacije je realizacija novog softvera kojim se zamenjuju ili dopunjuju postojeća softverska rešenja.
- Analiza i specificiranje zahteva je aktivnost u kojoj se identifikuju problemi koje je potrebno rešiti novim softverom. Proces softverskog inženjeringa određuje šta se mora učiniti da bi se problemi rešili. Podaktivnosti ove aktivnosti su: identifikacija zahteva, analiza i predstavljanje zahteva i razvoj kriterijuma i procedura za prihvatanje novog softvera.
- Specifikacija funkcija je aktivnost u kojoj se identifikuju i formalizuju podaktivnosti kao što su: definisanje predmeta obrade, identifikovanje atributa i veza objekata i operacija koje transformišu ove objekte i utvrđivanje ograničenja koja određuju ponašanje softvera.
- Strukturiranje delova i izbor je aktivnost kojom se na osnovu identifikovanih zahteva i specifikacije funkcija strukturira softver na takve delove kojima se može upravljati, a koji predstavljaju logičke celine. Nakon toga se vrši izbor i opredeljenje, da li novi, postojeći ili softverski sistem koji se može ponovo koristiti odgovara takvim celinama.
- Specifikacija strukture je aktivnost u kojoj se definišu međusobne veze između delova strukture i interfejs između modula sistema, na način koristan za njihov detaljni dizajn i upravljanje celokupnom konfiguracijom.
- Specifikacija detaljnih komponenti dizajna je aktivnost u kojoj se definišu procedure putem kojih se izvori podataka svakog pojedinog modula transformišu iz potrebnih ulaza u zahtevane izlaze.

- Implementacija komponenti i otklanjanje nedostataka je aktivnost u kojoj se kodiraju dizajnirane procedure i procesi i pretvaraju u izvorni kod.
- Integracija i testiranje softvera je aktivnost koja afirmiše i održava celokupnu integralnost komponenti softvera putem verifikacije konzistentnosti i kompletnosti uvedenih modula, verifikujući izvorni interfejs, vezu softvera sa specifikacijama i upoređujući performanse sistema i podsistema sa identifikovanim zahtevima.
- Provera dokumentacije i uvođenje softvera je aktivnost koja obuhvata izradu sistemske dokumentacije i uputstava za korisnika. To su dokumenta koja su po formi pogodna za proveru i kao podrška sistema.
- Obuka i upotreba je aktivnost koja obezbeđuje korisnike softvera sa instrukcijama i uputstvom za razumevanje mogućnosti i ograničenja u cilju efektivne upotrebe sistema.
- Održavanje softvera je aktivnost koja podržava operacije sistema u ciljnom okruženju na način da obezbedi potrebna unapređenja, proširenja, popravke, zamene i dr. Održavanje je stalni proces koji se realizuje putem iteracije aktivnosti koje mu prethode.
- Povlačenje softvera je poslednja aktivnost u životnom ciklusu. To nije ni malo jednostavna aktivnost jer se softver razmatran kao beskoristan ipak smatra proizvodom koji bi se mogao u delovima ponovo upotrebiti. One komponente softvera koje su već jednom testirane su ekonomski pogodnije od novih koje će se tek razvijati.

Ovakav konceptualni model na visokom stepenu apstrakcije primenjen u razvoju softvera se naziva životnim ciklusom softvera. On predstavlja opštu metodologiju, kod koje se za realizaciju pojedinih aktivnosti razvoja primenjuju različite metode, uz primenu različitih modela razvoja softvera.

Slika 3: Životni ciklus softvera



> Sistem analitičar, organizator obrade, administrator baze podataka ili programer, koji zanemaruje obrazovanje o novim metodologijama, igra ruski rulet sa svojom poslovnom karijerom.
>
> James Martin

3. Principi i modeli razvoja softvera

Softverski inženjering čine skupovi koraka koji uključuju metode, alate i procedure. Ti koraci su zasnovani na razvojnim principima i upućuju na modele razvoja softvera u softverskom inženjeringu. Model razvoja se odabira u zavisnosti od prirode projekta i aplikacije, tehničke orijentacije ljudi koji će učestvovati u razvoju, metoda i alata koji će se upotrebljavati pri razvoju, načina kontrole i samih proizvoda koji se zahtevaju.

Razvojni principi su takvi opšte važeći osnovni principi koji određuju način izvršenja rada i omogućuju uopštavanje određenih specifičnosti i zakonitosti objektivne stvarnosti. Oni mogu biti osnovni principi načina izvršenja rada i principi razvoja obzirom na metodološke korake i redosled njihovog izvršenja.

Osnovni principi načina izvršenja rada su: princip modelovanja, princip iteracija, princip simulacije, princip dokumentovanja i dr. To su opšte poznati i prihvaćeni principi u različitim područjima, koji se samo i u razvoju softvera primenjuju.

Principi razvoja obzirom na metodološke korake se međusobno razlikuju po tome koliki značaj pridaju pojedinim fazama u razvoju softvera, koliko ih detaljno posmatraju i u kojem redosledu izvršavaju.

Modeli razvoja se pojavljuju od vremena kada su se projektima razvijali veliki softverski sistemi i prikazuju različite poglede na proces razvoja softvera. Osnovni razlog njihove pojave je bila želja da se obezbedi uopštena šema razvoja softvera, koja bi služila kao osnova u planiranju, organizovanju, snabdevanju, koordinaciji, finansiranju i upravljanju aktivnostima razvoja softvera.

Uopšteno, modeli su apstrakcije koje pomažu u procesu razvoja softvera. Model softvera predstavlja komponente razvoja softvera i razvijen je na osnovu ideja konstruktora i njegove predstavke šta je najznačajnije u tom razvoju. Model može predstavljati: model procesa razvoja, model softvera ili model načina upravljanja softverom. Primarni cilj kreiranja modela je da se obezbede softverski proizvodi koji odgovaraju zahtevima korisnika.

U zavisnosti od značaja koji se pojedinim fazama i aktivnostima razvoja softvera pridaje, zatim forme organizacije i upravljanja razvojem, kao i iskustva zaposlenih i prirode proizvoda, razlikuju se:

Preskriptivni modeli razvoja – konvencionalni modeli sa delimično različitim tokom procesa razvoja ali sa istim generičkim aktivnostima;

- Model vodopada,

Inkrementalni modeli razvoja:

- Inkrementalni model,
- RAD model,

Razvojni modeli:

- Model prototipskog razvoja,
- Spiralni model,
- Istovremeni model razvoja (Concurrent Development)

Specijalizovani modeli:

- Model zasnovan na komponentama,
- Model zasnovan na formalnim metodama,

Model unificiranog procesa razvoja (Unified Process)

Modeli agilnog razvoja:

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Feature Driven Development (FDD)
- Agile Modeling (AM)

U nastavku se opisuju neki od odabranih modela razvoja softvera.

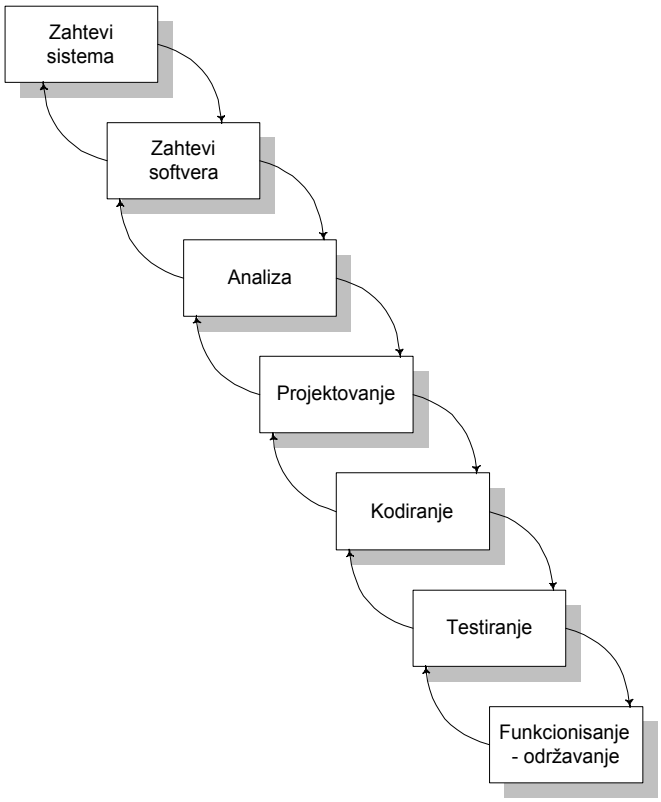
3.1. Model vodopada

Model je uveo W. Royce 1970. godine. Prema njemu razvoj softvera zahteva sistematičan pristup jer se odvija po strogo definisanom sekvencijalnom redosledu koraka postepenim prevođenjem rezultata od prve do poslednje faze razvoja softvera. Razvoj započinje na sistemskom nivou da bi se nastavio preko analize, projektovanja, kodiranja, testiranja i završio održavanjem.

Faze i aktivnosti razvoja prema ovom modelu su sledeće:

- Analiza i definisanje zahteva sistema - Obzirom da softver predstavlja samo deo nekog sistema, rad na razvoju softvera započinje definisanjem zahteva prema svim elementima sistema i alociranjem jednog dela adekvatnih i određenih zahteva prema softveru. Ovaj sistemski pogled je posebno značajan kada se softver povezuje sa ostalim elementima strukture kao što su hardver, menver (organizacija i zaposleni), podaci i dr.
- Analiza i definisanje zahteva softveru – Ovom fazom i njenim aktivnostima se intenzivira prikupljanje specifičnih i posebnih zahteva softveru. Da bi softverski inženjer razumeo prirodu softvera koji treba razviti, on mora razumeti domen koji informacija ima za softver kao i zahtevane funkcije, performanse i međusobne veze. Zahtevi sistema i zahtevi softveru se dokumentuju, a zatim analiziraju i pregledaju sa korisnicima.

Slika 4: Model vodopada



- Projektovanje ili dizajn softvera - Projektovanje softvera je faza razvoja koju čini više aktivnosti, a koje se fokusiraju na nekoliko aspekata razvoja softvera: korisnički interfejs, ulazne ekranske forme, izlaze, bazu podataka, procedure obrade i sistemsku kontrolu. Ova faza prevodi zahteve korisnika u određeni softverski proizvod koji se može oceniti sa aspekta kvaliteta pre nego što započne kodiranje. Kao i zahtevi, rezultati projektovanja se dokumentuju i predstavljaju deo konfiguracije softvera.
- Kodiranje – Ovom fazom se izvršava zadatak prevođenja rezultata projektovanja u mašinski prepoznatljivu formu. Ukoliko je projektovanje urađeno dovoljno detaljno, tada se kodiranje obavlja mehanički.
- Testiranje - Kada se jednom izgeneriše kod programa, tada započinje njegovo testiranje. Testiranje se svodi na unutrašnju logiku softvera, sa ciljem da se svi iskazi provere odnosno da se proveri da li su isti tačni. Takođe, testiranje se svodi i na spoljnu funkciju softvera, da bi se otkrile greške i proverilo da li će definisani ulazi proizvesti rezultate koji se podudaraju sa identifikovanim zahtevima.
- Održavanje - Softver će sigurno pretrpeti određene izmene nakon što se distribuira korisniku. Potrebe za izmenama se javljaju zbog proširenja funkcija ili performansi koje zahteva korisnik, zbog potreba da se softver prilagođava promenama koje uzrokuje promenjeno okruženje ili zbog razvoja tehnologija koje se upotrebljavaju.

Svaka od navedenih faza u modelu poseduje specifičnosti, rezultira izvesnim proizvodom i omogućuje reviziju. Aktivnosti faze se izražavaju neophodnim ulazom, procesom i realizovanim izlazom. Razvoj softvera tako prolazi kroz niz koraka sa iterativnim interakcijama između aktivnosti koje su sukcesivne odnosno povezane kao susedne. Rezultat realizacije određene aktivnosti je izlaz - uglavnom proizvod koji se koristi kao ulaz u sledeću aktivnost.

U stvarnosti, razvoj nikad nije tako eksplicitno određen. Uvek postoji povratna sprega između aktivnosti, ali samo onih koje su susedne i u neposrednoj vezi. Zbog toga razvijeni proizvodi u svakoj fazi zahtevaju proveru i reviziju pre prihvatanja.

Model je posebno efikasan u struktuiranju i upravljanju malim projektima razvoja softvera u organizacijama. To je najstariji model razvoja koji je najviše i najšire primenjivan do danas. Uspešno se kombinuje sa drugim modelima razvoja. Veoma mnogo je kritikovan, ali se ipak zadržao na visokim pozicijama sa stanovišta primene. Primena ovog modela se predlaže u sledećim situacijama:

- prilikom razvoja softvera koji je po osnovnim karakteristikama jedinstven i ima zadatak da zadovolji posebne zahteve korisnika, koji još do tada nisu realizovani u sistemima drugih organizacija i ne mogu se šire upotrebiti,

- kada korisnik jednoznačno može definisati svoje zamisli i potrebe u odnosu na softver, koji na toj bazi izgrađen ne sadrži suvišne komponente i funkcioniše veoma brzo i uspešno,
- postoji dovoljno vremena i strpljenja kod korisnika za dugi period razvoja i
- visoki razvojni troškovi i saobrazno potrebna finansijska sredstva nisu ograničavajući faktor razvoja.

Slabost modela je nedostatak povratne sprege između koraka koji nisu sukcesivni i ne odvijaju se u sekvencijalnom redosledu. Takođe, kao nedostaci se navode sledeće činjenice:

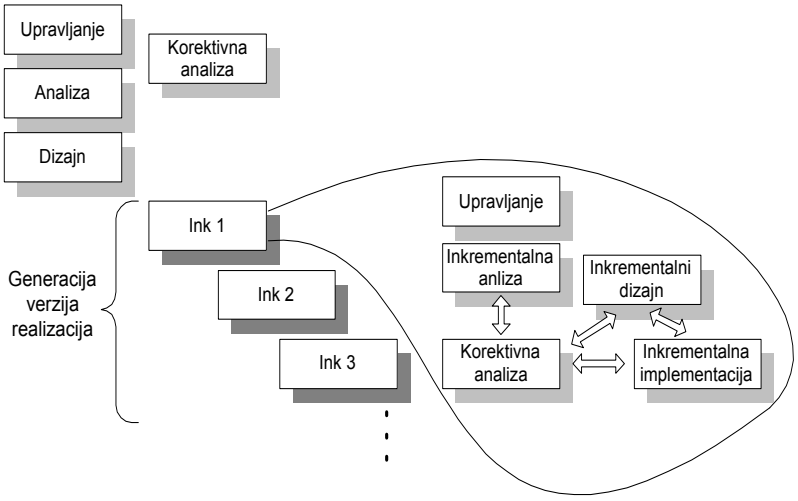
- realni projekti veoma retko prate modelom definisani sekvencijalni tok, a iteracije uvek izazivaju ili se kod njih javljaju problemi u primeni modela,
- uvek je teško za korisnika da u početku rada na razvoju softvera navede eksplicitno sve svoje zahteve (a što model zahteva), jer se teško prilagođava neizvesnosti koja uglavnom egzistira na startu,
- kupac mora biti veoma strpljiv i istrajan jer će mu radne verzije programa biti dostupne tek na kraju aktivnosti razvoja softvera,
- greške koje se ne otklone u fazi testiranja programa, mogu imati stravično distorziono dejstvo na projekat razvoja.

3.2. Inkrementalni model

U ovom modelu razvoja se prvobitno potpuno razvija inicijalni podskup funkcija softvera, a zatim se sukcesivnim koracima razvijaju, kao nadgradnja prethodnog koraka, stalno novije i komplikovanije verzije. Projektovanje softvera se izvodi u prvom koraku, ali se uvođenje softvera odvija sukcesivnom razradom (usavršavanjem inicijalnog podskupa). Softver se razvija malim dodacima (inkrementima) kojima se može jednostavno i lako upravljati. Svaki inkrement dodaje postojećem softveru pojedine nove funkcije, pri čemu se postojeće zadržavaju. Prednost ovakvog razvoja je u tome da se dodaci odnosno nove funkcije lakše razumeju i testiraju. Korišćenje mogućnosti da se stalno dodaju nove funkcionalnosti softverskom proizvodu, daje mogućnost ugradnje bogatog korisničkog iskustva u redefinisani proizvod na manje skup način.

Ovaj model predstavlja kombinaciju klasičnog modela životnog ciklusa softvera sa iterativnim mogućnostima razvoja. On takođe obezbeđuje način da se periodično distribuira ažuriranje i održavanje softvera različitim korisnicima. Posebno je popularan i koriste ga u softverskim kućama.

Slika 5: Inkrementalni model



3.3. Model prototipskog razvoja

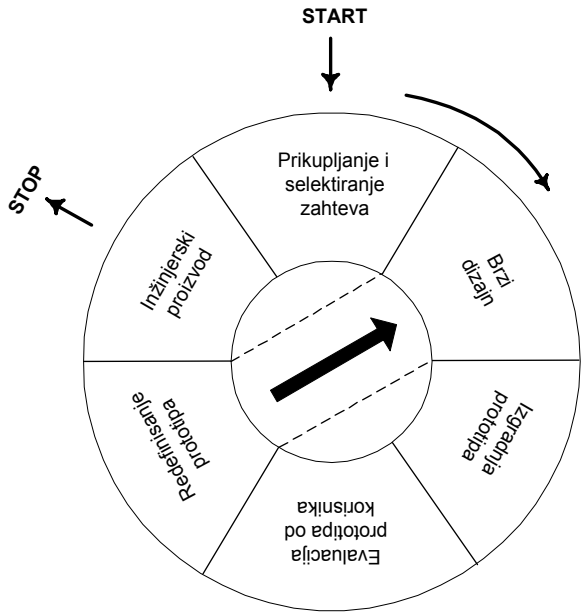
Model prototipskog razvoja se koristi da bi se za potrebe korisnika razvio inicijalni model budućeg softvera koji simulira njegove stvarne funkcije sa ciljem da korisnik da svoje mišljenje i odluči koji i kakvi su njegovi zahtevi. Kod razvoja softvera po ovom modelu korisnik već u najranijem stadijumu može videti na koji će se način zadovoljiti njegovi zahtevi.

Komponente razvijenog softvera često predstavljaju samo korisnički interfejs. Implementacija kostura ovog interfejsa je sa željom da se obezbedi mogućnost za povratnu spregu od korisnika, pre nego da se specificira i projektuje konačna verzija rešenja. Mada je razjašnjenje korisničkog interfejsa jedan od ciljeva, prototip može biti takođe iskorišćen kao koncept unutar konteksta drugog modela. U tom slučaju, drugi model može posmatrati prototip kao jedan elemenat procesa, koji se koristi u razjašnjenju ponašanja sistema u različitim tačkama razvoja softvera.

Model obično prihvata neku vrstu funkcionalne specifikacije softvera kao ulaz, koji se simulira, analizira i izvršava. Ova tehnologija omogućuje da aktivnosti projektovanja softvera budu inicijalno preskočene ili premoštene. Takođe, omogućuje da se brzo izgrade primitivne verzije softvera, koje kasnije korisnik može i sam razvijati. Korisnički razvoj je uključen u povratnu spregu za redefinisanje sistemskih specifikacija i dizajna. U zavisnosti od tehnologije prototipskog razvoja, ceo radni sistem može biti razvijen kontinuiranim procesom

prečišćavanja ulaznih specifikacija. Ključna prednost ovog modela je što stalno obezbeđuje radne verzije sistema koje razvija i što u većoj meri od drugih modela angažuje korisnika na razvoju, te unapređuje kvalitet procesa.

Slika 6: Model prototipskog razvoja



Model prototipskog razvoja se najčešće upotrebljava i daje solidne rezultate u situacijama:

- kada su od strane korisnika samo uopšteno definisani ciljevi razvoja softverskog proizvoda, ali ne i detalji u pogledu ulaza, procedura i izlaza,
- kada je moguće simulirati rad softvera da bi korisnik mogao videti kako će budući softverski proizvod funkcionisati i
- kada same razvojne organizacije žele proveriti efikasnost algoritama ili adaptibilnost sistema.

Prototipski razvoj omogućuje onome koji razvija softver da kreira model softvera. Model uobičajeno može imati tri oblika:

- prototip u obliku papira koji opisuje vezu čoveka i mašine na način da korisniku omogući razumevanje tog odnosa,
- radni prototip koji implementira neke od funkcija postavljenih kao zahtevi softverskom proizvodu ili
- realni program koji izvršava deo ili celinu zahtevanih funkcija.

Model prototipskog razvoja započinje prikupljanjem zahteva. Projektant i korisnik, zajednički definišu opšte ciljeve razvoja softverskog proizvoda, identifikuju sve njima poznate zahteve i određuju područja na kojima su obavezne dalje aktivnosti preciznijeg definisanja. Sledi "brzi" dizajn u kome se fokusira na realizaciju onih aspekata softvera koji će biti vidljivi za korisnika (ulazni i izlazni formati i dr.). Nakon takvog dizajna, razvija se prototip. On služi da bi se prečistili zahtevi prema softveru koji se razvija. Prečišćavanje je iterativno i odvija se dok prototip ne zadovolji zahteve korisnika i istovremeno omogući projektantu potpuno razumevanje potreba koje mora zadovoljiti. Idealno, prototipski razvoj i služi kao mehanizam za identifikovanje zahteva prema softveru. Ukoliko se razvija radni prototip, korisniku se omogućuje da koristi delove softvera ili primenjuje alate koji brzo generišu radne verzije softvera.

Ovaj model ima i svoje nedostatke odnosno primena modela prototipskog razvoja može biti diskutabilna iz dva razloga:

- Korisnik uočava radnu verziju softvera neznajući na koji su način delovi softvera međusobno povezani, neznajući da u brzini realizacije nisu razmatrani aspekti kvaliteta ili održavanja u dužem vremenskom periodu. Kada dođe do informacija da je potrebno izvršiti "remont" ili dalju dogradnju još ne uvedenog softverskog proizvoda, korisnik se oseća prevarenim i insistira da se putem izvesnih intervencija brzo realizuje njemu potreban proizvod. Upravljanje razvojem softvera u ovakvim situacijama postaje nekontrolisano.
- Projektant često čini kompromise u implementaciji sa ciljem da izgrađeni prototip što pre stavi u funkciju. Neadekvatan operativni sistem ili programski jezik se jednostavno upotrebljavaju samo zato što su raspoloživi ili poznati; neefikasan algoritam se primenjuje samo da bi se demonstrirala sposobnost softvera. Nakon izvesnog vremena, zaboravlja se na način odabira i njihove uzroke te ovako manje idealna rešenja ili bolje rečeno manje kvalitetna rešenja ostaju integralni deo konačnog softverskog rešenja.

Zbog toga, značajno je da se projektant i korisnik, u cilju efikasnosti ovog modela, dogovore i definišu 'pravila igre' na početku procesa razvoja softvera. Drugim rečima oni se moraju složiti da se prototip razvija kao mehanizam za definisanje zahteva, a softver se razvija u cilju zadovoljenja kriterijuma kvaliteta i mogućnosti održavanja.

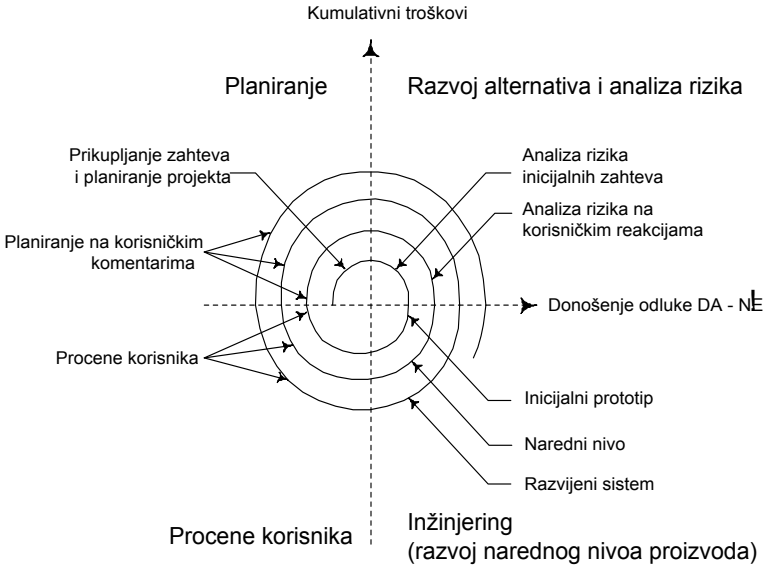
3.4. Spiralni model

Spiralni model je razvijen od. B Boehma 1988. godine, da bi se objedinile dobre osobine modela vodopada i modela prototipskog razvoja uz istovremeno uključivanje aktivnosti analize rizika. Model se predstavlja spiralom na kojoj su definisane četiri faze razvoja:

- planiranje – faza koju čine aktivnosti određivanja ciljeva, alternativa i ograničenja,
- analiza rizika – faza koju čine aktivnosti analize alternativa i identifikovanja rizika,
- inženjering – faza razvoja novih nivoa proizvoda i
- ocenjivanje – faza procene rezultata inženjeringa.

Posmatranjem spirale, svakom iteracijom se progresivno razvijaju kompletnije i potpunije verzije softvera. Tokom prvog ciklusa kretanja spiralom, prikupljaju se zahtevi i planira projekat razvoja, da bi se izvršila analiza rizika inicijalnih zahteva. Ukoliko analiza rizika indicira neizvesnosti u zahtevima, tada se može upotrebiti prototipski razvoj da bi se zahtevi detaljnije spoznali. U iste svrhe, mogu se koristiti simulacija ili druge vrste modela.

Slika 7: Spiralni model



Nakon što se donese odluka o daljem razvoju, obavlja se inžinjering u svakom ciklusu spirale i to odabranim modelom razvoja softvera (model vodopada i-ili model prototipskog razvoja). Broj aktivnosti u inženjeringu raste, ukoliko se ciklusi udaljuju od centra spirale. Istovremeno, aktivnosti su složenije i uvek sa mnogo manje apstrakcije.

Po završetku inžinjerskog posla razvoja, korisnik isti ocenjuje i daje sugestije za modifikaciju softverskog proizvoda. Zasnovana na korisničkom inputu, javlja se sledeća faza planiranja novog ciklusa razvoja i analize rizika. Svaki ciklus razvoja na spirali, zahteva analizu rizika i donošenje odluke "nastaviti" ili "ne nastaviti" sa daljim razvojem. Ukoliko je rizik isuviše velik i ulaganja nesrazmerno visoka u odnosu na efekte koji se očekuju, terminira se dalji rad i zadržava u upotrebi proizvod nastao u prethodnom ciklusu ili prethodnim ciklusima. Svaki novozapočeti ciklus spirale donosi kompletniji proizvod, ali i značajnije i više troškove.

Tipična raspodela vremena, za koja ipak treba reći da variraju od ciklusa do ciklusa, je:

- planiranje i dizajn 20%,
- procena rizika 5%,
- realizacija (sa testiranjem) 40%,
- revizija 15% i
- ocenjivanje 20%.

Rezimirajući izloženo treba podvući da model uključuje multiplikovane iteracije kroz cikluse, koji analiziraju rezultate prethodnih faza određujući procenu rizika za buduće faze. U svakoj od faza, razvijaju se alternative u skladu sa ciljevima i potrebama koje zajedno čine bazu za sledeći ciklus u spirali. Svaki ciklus se završava ocenom. Model podrazumeva da se svaki deo proizvoda ili svaki nivo proizvoda na istovetan način provlači kroz ovu spiralu odnosno ocenu.

Osnovna premisa modela je da se određeni redosled koraka ponavlja u razvoju i održavanju softvera. Koraci se prvo izvršavaju sa visokim stepenom apstrakcije. Svaka petlja spirale predstavlja ponovljene korake na nižem stepenu apstrakcije odnosno sa više detalja.

Prednost modela je u njegovoj fleksibilnosti za upravljanje softverskim inženjeringom. Procena rizika se može izvesti u svakom trenutku i nivou apstrakcije. Model prilagođava svaku kombinaciju različitih pristupa u razvoju softvera.

Ovo je trenutno najrealniji pristup u razvoju softvera za velike sisteme. Model omogućuje brzu reakciju na uočene rizike, a primenom prototipskog razvoja pruža mehanizam za njihovo smanjenje. Tako se primenom ovog modela rizici mogu

smanjiti pre nego što izazovu veće probleme i velike troškove. Model podržava sistematski pristup preuzet iz modela vodopada uz mogućnost izvođenja iteracija.

I pored velikog broja prednosti, model poseduje i nedostatke. Nedostatak ovog modela je odsustvo veze prema postojećim standardima, odnosno ne postojanje standarda za ovaj način razvoja softvera. Takođe, model zahteva više uniformnosti i konzistentnosti u razvoju. Velike probleme stvara situacija kada se na vreme ili uopšte ne otkriju rizici. Konačno, model je relativno nov i nije bio široko primenjivan. Stoga, biće potrebno još dosta vremena da se sa više sigurnosti i verovatnoće priđe njegovoj ozbiljnijoj primeni.

3.5. Model zasnovan na komponentama

Osnovni pristup u ovom modelu je konfigurisati i specijalizirati već postojeće komponente softvera u novi aplikativni sistem. Međutim, osobine komponenti zavise od njihove veličine, kompleksnosti i funkcionalnih mogućnosti. Većina pristupa pokušava da iskoristi slične komponente obzirom na zajedničke strukture podataka sa algoritmima za njihovu manipulaciju. Drugi pristupi pokušavaju da iskoriste komponente funkcionalno sličnih kompletnih sistema ili podsistema kao što su korisnički interfejs. Postoje i brojni načini iskorišćavanja softverskih komponenti za razvoj softverskih sistema. Svi ovi pokušaji i nastojanja zagovaraju inicijalnu upotrebu već urađenih komponenti u specificiranju strukture ili detaljnom dizajnu komponenti radi ubrzavanja postupka implementacije. Ove komponente se mogu upotrebiti i pri prototipskom razvoju softvera ukoliko je raspoloživa takva tehnologija.

Višestruko korišćenje softvera je proces uključivanja u novi proizvod pojedinih komponenti:

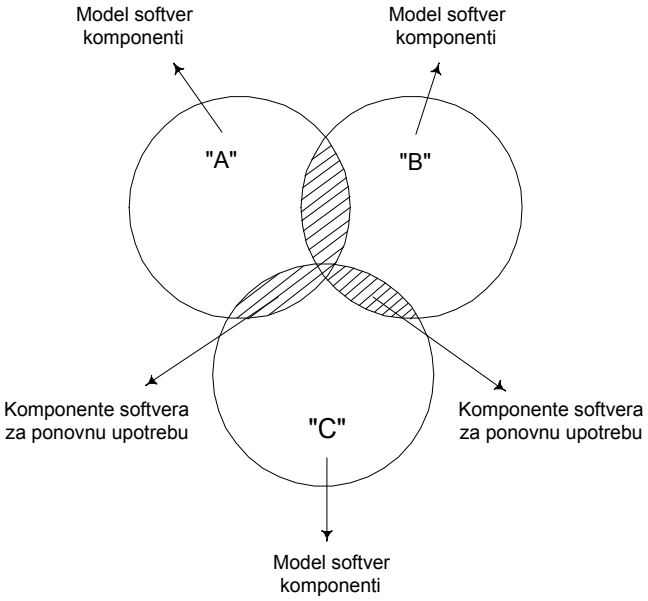
- prethodno testiranog koda,
- prethodno proverenog dizajna,
- pretnodno razvijene i korišćene specifikacije zahteva i
- prethodno korišćenih procedura za testiranje.

Koristi koje sobom donosi ponovno korišćenje komponenti razvijenog softvera su sledeće:

- podiže robustnost softvera,
- povećava produktivnost izrade softvera,
- povećava kvalitet softvera,
- smanjuje troškove razvoja softvera,
- štedi odnosno skraćuje vreme izrade,
- zadovoljava ciljeve softverskog inženjeringa,

- širi korišćenje softvera,
- obezbeđuje adekvatnu dokumentaciju,
- olakšava održavanje softvera,
- modelira sistem za lakše razumevanje i dr.

Slika 8: Model ponovnog korišćenja komponenti softvera



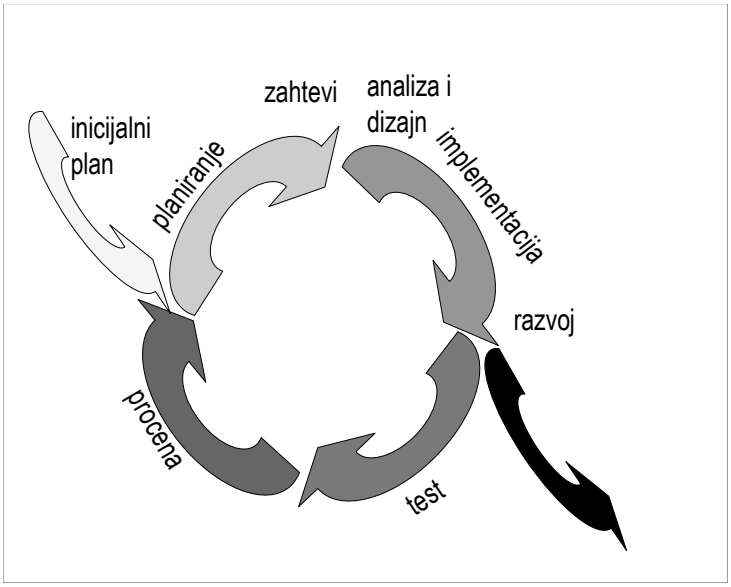
3.6. Model unificiranog procesa razvoja

Za razliku od prethodno opisanih modela razvoja softvera, Ivar Jacobson, Grady Booch i James Rumbaugh su 1999. godine objavili USDP – model unificiranog procesa razvoja softvera. Ovaj model opisuje proces razvoja korišćenjem UML - objedinjenog jezika za modelovanje u objektno-orijentisanom razvoju softvera. Ovaj model se zasniva na 3 osnovna principa - osnovu čine dijagrami slučajeva korišćenja, u centru modela se nalazi njegova arhitektura koja sazreva sa razvojem novih dijagrama korisnika i razvija se kroz iteracije koje donose nove inkremente konačnog proizvoda. Suštinski posmatrano, u svakoj od iteracija odvijaju se analiza, projektovanje, implementacija i testiranje proizvoda.

Prema autorima, model se može opisati kao proces klasificiranja iteracija, koje se mogu podeliti u 4 grupe:

- u prvoj grupi se nalaze početne iteracije interakcija sa stekholderima, tj. značajnim učesnicima u razvoju softvera,
- drugu grupu sačinjavaju razrađene iteracije želja i potreba korisnika,
- iteracije konstruisanja inicijalnih operacionih mogućnosti sačinjavaju treću grupu i
- prelazne iteracije kompletiranja proizvoda su četvrta, konačna iteracija razvoja softvera prema ovom modelu.

Slika 9: Model unificiranog procesa razvoja



3.7. Agilni modeli razvoja

Česta kašnjenja projekata razvoja softvera, probijanje budžeta i postavljenih vremenskih rokova u njihovoj realizaciji, permanentni rast složenosti tehnologije i učestale promene korisničkih zahteva, doveli su krajem dvadesetog veka do pojave novih modela razvoja. Nastali su agilni modeli, po osobinama mnogo gipkiji i prilagodljiviji promenama, koji omogućuju korisnicima aktivno učešće tokom svih faza i aktivnosti razvoja.

Agilni pristup se dakle suočio sa osnovnim problemom savremenog i ujedno brzog razvoja softvera. Dominantna ideja je da timovi mogu biti efikasniji u realizaciji promena ako su u stanju da smanje vreme i troškove razmene informacija između

osoba koje učestvuju u razvoju na način da skrate vremenski period od donošenja odluke do povratne informacije o posledici te odluke.

Polazne pretpostavke agilnih modela su bile da je turbulentnim poslovnim i tehnološkim okruženjima neophodan proces razvoja softvera koji istovremeno kreira promene, ali brzo i odgovara na iste. Istovremeno, proces koji uključuje odgovorne učesnike i njihovu dobru organizaciju. Učesnicima odnosno njihovom talentu, veštinama i sposobnostima, kao i njihovoj međusobnoj komunikaciji se poklanja posebna pažnja. Usmerenost na učesnike je i najznačajnija osobina agilnih modela, prema pojedincima se prilagođava i kompletan proces razvoja.

U agilnim razvojnim timovima, kompetencije pojedinaca predstavljaju kritičan faktor uspešnosti projekta. Prema agilnim modelima, ukoliko su pojedinci na projektu dovoljno kvalitetni, tada oni mogu uz bilo koji proces razvoja realizovati očekivani cilj. U suprotnom, nema procesa razvoja koji može nadomestiti njihovu nekompetentnost. Istovremeno, nedostatak korisničke podrške može lako uništiti projekat razvoja, kao što i neadekvatna podrška može sprečiti završetak projekta.

Agilni procesi ističu jedinstvene sposobnosti pojedinaca i tima. Naime, procesi ne mogu nadvladati nedostatak kompetencija pojedinaca. Timovi su samoorganizovani, sa intenzivnim komunikacijama u okviru i van organizacionih granica. Ovi timovi mogu u svakom trenutku promeniti svoju strukturu kako bi se prilagodili promenama. Agilnost podrazumeva da tim ima zajednički cilj, uzajamno poverenje i poštovanje, zajednički i brz postupak donošenja odluka i sposobnost savladavanja svih dvosmislenosti. Agilan tim koji radi u okviru krute organizacije ima poteškoća, kao što ih ima svaki pojedinac koji radi u krutom timu. U ovim timovima, dominira saradnja svih nivoa upravljanja. Za donošenje odluke nije važno ko donosi odluke, već je važna saradnja i obezbeđenje informacija za donošenje odluka. U projektu razvoja učestvuju osobe različitih veština, talenta i sposobnosti, koje rade u bliskom fizičkom okruženju i poštuju organizacionu kulturu. Osobe, okruženje i kultura su u strogoj međuzavisnosti.

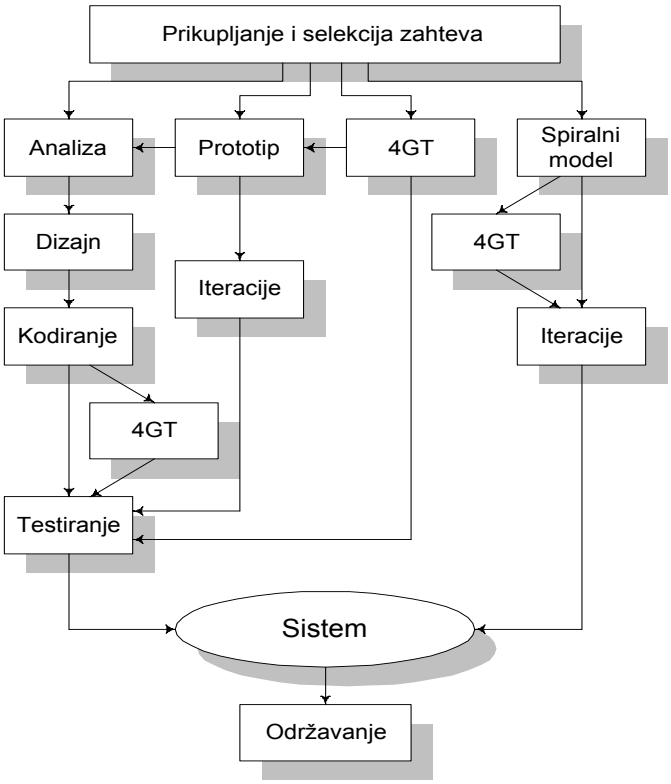
Agilni razvoj nije prikladan za sve situacije. Nametanje agilnih principa procesno usmerenim i nekooperativnim organizacijama ne dovodi do uspeha. Nametanje izuzetno promenljivog procesa mirnim i staloženim timovima, vodi sigurno raspadu tima. Takođe, agilni razvoj se teško izvodi u timovima sa većim brojem članova. Najviše uspeha u agilnom razvoju pokazuju timovi do devet članova. Agilni razvoj se pokazao kao uspešan u ekstremnim, kompleksnim i visoko-promenljivim projektima. Okruženje u kojem ovaj pristup daje najbolje rezultate je organizaciona kultura koja je orijentisana na ljude i saradnju.

3.8. Kombinovani modeli

Napred opisani modeli su uglavnom prikazivani kao alternativni, a manje kao komplementarni modeli softverskog inženjeringa. Međutim, u mnogim situacijama modeli se mogu kombinovati tako da se postignu prednosti od svih na samo

jednom projektu. Spiralni model je i sam primer dobre kombinacije dva modela, ali i drugi modeli mogu poslužiti kao osnova na koju će se integrisati neki modeli.

Slika 10: Kombinovani modeli



Ne treba biti dogmatičan u izboru određenog modela u softverskom inženjeringu. Priroda aplikacije će diktirati model koji bi trebalo primeniti. Kombinovanjem modela, rezultat postignut u celini može biti povoljniji nego što bi to bio prosti zbir rezultata postignutih pojedinim modelima.

> Mi u računarskoj industriji ponekad zaboravljamo da je upotreba naših sopstvenih alata jedan od razloga toliko brzog razvoja ove industrijske grane.
>
> Bill Gates

4. Sredstva u razvoju softvera

4.1 UML

Akronim UML je danas među najpoznatijim u informatičkom svetu i označava termin Unified Modeling Language, što u prevodu znači jedinstveni jezik za modelovanje. UML predstavlja jedinstveni jezik za vizuelizaciju, specifikaciju, konstrukciju i dokumentovanje softverskih rešenja.

Grafički jezici poput UML-a se koriste već duže vremena u razvoju softvera. Međutim, ono što je specifično za sve njegove preteče, jeste njihova neusaglašenost, koja je i bila ključna za nastanak i razvoj UML-a.

UML je nastao objedinjavanjem više objektno orijentisanih grafičkih jezika za modelovanje. Tvorci UML-a su trojica stručnjaka, poznata u informatičkom svetu pod nadimkom „tri amigosa“, Grady Booch, Jim Rumbaugh i Ivar Jacobson. Svaki od njih je razvijao sopstveni objektno orijentisani jezik za modelovanje, da bi se udružili pod okriljem firme Rational (danas u sastavu IBM-a) i miksom pojedinačnih jezika stvorili UML. Prva zajednička verzija UML-a, nastala radom ova tri autora, jeste verzija 1.0 lansirana na tržište 1997. godine, koja je posedovala osobine standarda i bila opšteprihvaćena od svih učesnika u razvoju informacionih sistema. Od tada brigu za UML preuzima grupa OMG, tj. grupa za upravljanje objektima (Object Management Group, OMG).

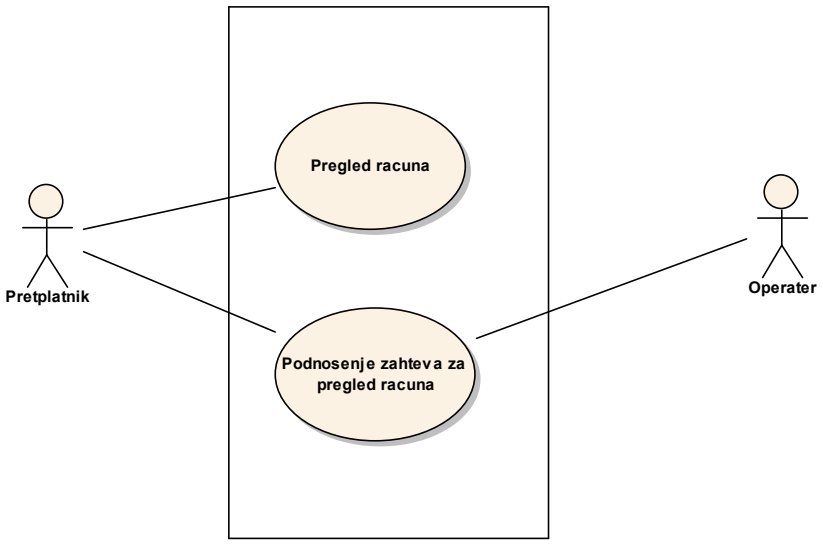
Trenutno aktivna verzija UML-a jeste 2.0. U njoj postoji 13 vrsta dijagrama, strukturiranih u dve osnovne grupe: dijagrami ponašanja i dijagrami strukture. U procesu razvoja softvera pojavljuju se razne uloge koje koriste različite tehnike dijagramiranja, za rešavanje različitih problema. To u stvari znači da brojni 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 opštim prihvatanjem 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 se ukratko opisuju 13 vrsta UML dijagrama.

Dijagrami slučajeva upotrebe (Dijagram korisničkih funkcija)

Dijagrami slučajeva upotrebe služe za 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 funkcije sistema koje će biti automatizovane, a pomoću dijagrama slučajeva upotrebe poslovnog procesa i automatizovane i manuelne funkcionalnosti.

Slika 11: Dijagram slučajeva upotrebe



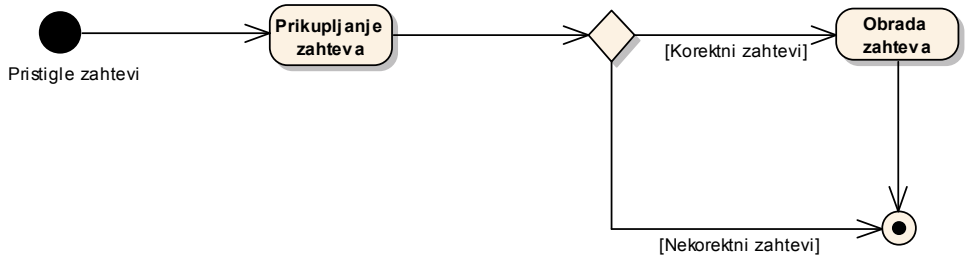
Komponente ovih dijagrama su akteri, uloge, slučajevi upotrebe i relacije. Akteri predstavljaju nekoga ili nešto što se nalazi izvan sistema ili organizacije (u zavisnosti od vrste dijagrama), a u interakciji je sa njim. Uloge se koriste samo prilikom izrade dijagrama slučajeva upotrebe poslovnog procesa i predstavljaju nekoga ili nešto što se nalazi unutar organizacije i u interakciji je sa funkcionalnostima posmatranog dela organizacije. Slučajevi upotrebe i slučajevi upotrebe poslovnog procesa služe da bi se prikazale konkretne funkcionalnosti sistema, odnosno organizacije. Ovi dijagrami predstavljaju vodilju za kompletan proces razvoja softvera, 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. Tokovi funkcionalnosti predstavljeni dijagramima slučajeva upotrebe se opisuju dijagramima aktivnosti, koji prikazuju 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 aktivnosti sadrže: stanja aktivnosti i stanja akcije, tranzicije, objekte, grananja, početnu (označava se crnim krugom) i krajnju tačku (označava se crnim krugom u belom krugu). Između početne tačke i prve akcije navodi se događaj koji je pobuđuje. Tranzicije povezuju početnu tačku, akcije i krajnju tačku i predstavljaju se usmerenim linijama. Ukoliko se tranzicije granaju, tačka u kojoj se vrši grananje prikazuje se rombom.Stanja aktivnosti i akcija navode se u elipsama, a za objekte važi standardna notacija.

Slika 12: Dijagram aktivnosti



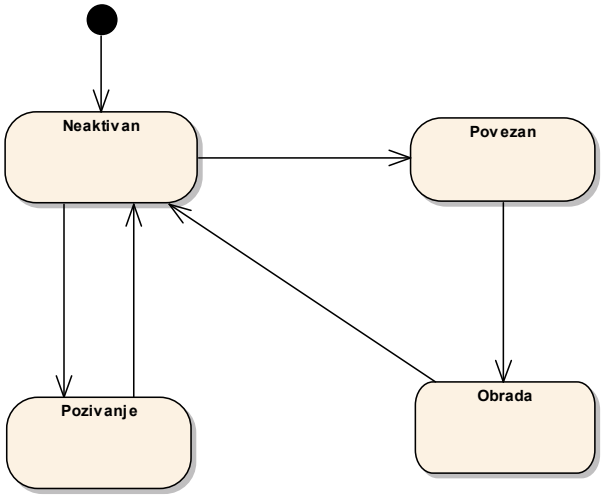
Dijagrami stanja mašine

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.

Predstavljaju uopštenje dijagrama aktivnosti. Pomoću dijagrama stanja se modeluju dinamički aspekti sistema koji se projektuje. Koriste se za modelovanje životnog veka objekta, a najčešće se modeluju stanja objekata koji reaguju.

Na dijagramu se, u obliku konačnog automata, prikazuje odvijanje upravljanja od stanja do stanja. Ovo se prikazuje tako što se stanje prikazuje pravougaonikom sa zaobljenim ivicama, dok je tranzicija relacija izmedju dva stanja i prikazuje se strelicom, a stanje od kojeg počinje kreiranje ovog konačnog automata se prikazuje crnim kružićem.

Slika 13: Dijagram stanja



Dijagrami sekvenci i dijagrami saradnje

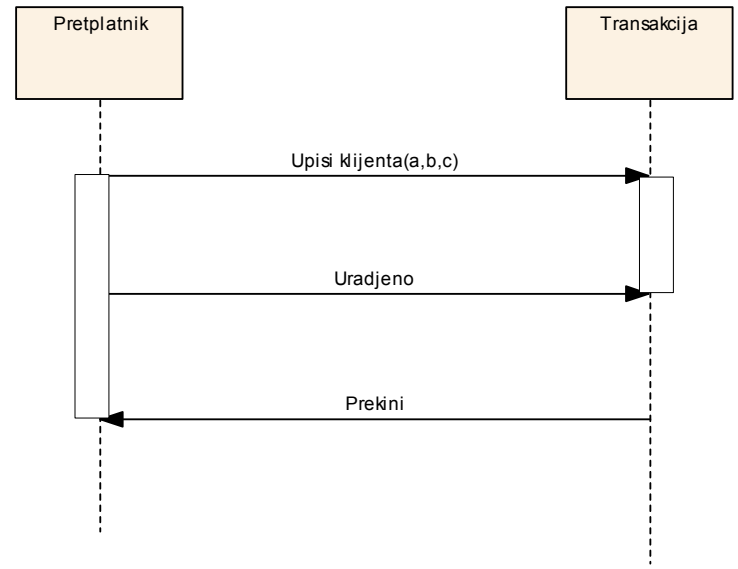
Ove dve vrste dijagrama prikazuju iste informacije iz različitih perspektiva. Pomoću njih se predstavljaju objekti koji se pojavljuju u okviru slučaja upotrebe i poruke koje oni razmenjuju. Razlika između ovih dijagrama je u tome što dijagrami sekvenci u prvi plan stavljaju vremensku dimenziju, tj. razmenu poruka sa aspekta vremena, dok dijagrami saradnje 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.

Dijagram sekvence predstavlja specijalni slučaj dijagrama interakcije, a njime se prikazuje skup poruka razmenjenih između objekata koji sarađuju, da bi se realizovala određena operacija ili dobio neki rezultat. Ovaj dijagram se prikazuje u dve dimenzije: vertikalna dimenzija prikazuje vreme, a horizontalna određene objekte.

Na dijagramu se prikazuju poruke koje se razmenjuju u definisanom vremenskom redosledu između uočenih objekata. Među objektima se uspostavljaju veze, a dijagramom se prikazuje linija života svakog prikazanog objekta. Isto tako ovi

dijagrami sadrže fokus upravljanja kojim se prikazuje vremenski period u kojem objekat vodi neku akciju.

Slika 14: Dijagram sekvenci



Elementi dijagrama sekvenci se predstavljaju na sledeći način:

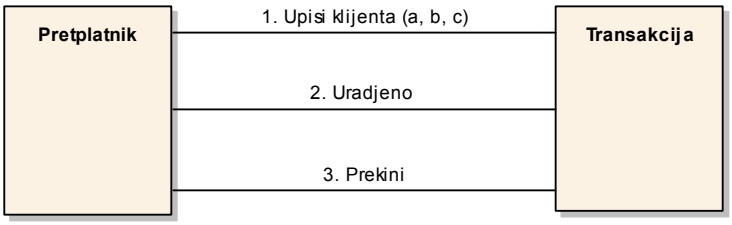
- objekti se prikazuju na ranije opisani način,
- veze između objekata su poruke, koje se prikazuju položenim usmerenim linijama,
- linija života objekta je vertikalna isprekidana linija,
- fokus upravljanja se predstavlja uzanim, vertikalno postavljenim pravougaonikom.

Dijagrami saradnje su semantički ekvivalentni dijagramima sekvence. Pomoću njih se ističe strukturna organizacija objekata koji učestvuju u interakciji. Za razliku od dijagrama sekvence, dijagrami saradnje ne sadrže liniju života objekta niti fokus upravljanja, već sadrže objekte iz interakcije sa vezama koje sadrže poruke između njih.

Objekti se prikazuju kao čvorovi grafa, a poruke između ovih objekata peikazuju se kao grane grafa. Poruke mogu imati svoje redne brojeve (sa prefiksom), a koji

označavaju redosled razmene poruka. Ovaj redosled formira putanju poruka, koja se takođe označava i usmerenim strelicama na granama grafa.

Slika 15: Dijagram saradnje



Dijagrami pregleda interakcija

Dijagrami pregleda interakcija 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 upoređivati 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.

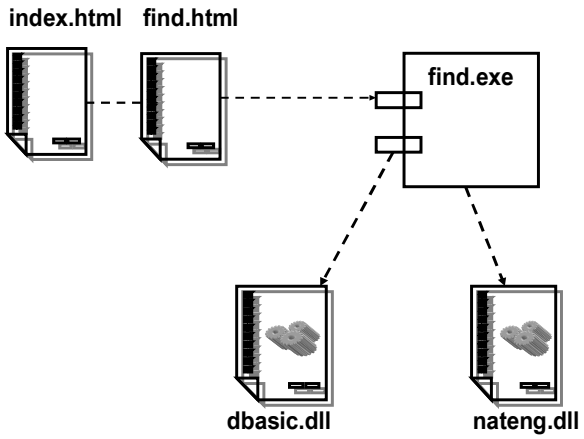
Dijagrami objekata

Dijagrami objekata su postojali i u ranijim verzijama UML-a, ali kao neformalni dijagrami. Od verzije 2.0 zvanično postaju 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. To su, u stvari, dijagrami klasa kod kojih je pažnja usmerena na komponente sistema. Na taj način se prikazuje konstrukcija izvršnih softverskih sistema.

Slika 16: Dijagram komponenti



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

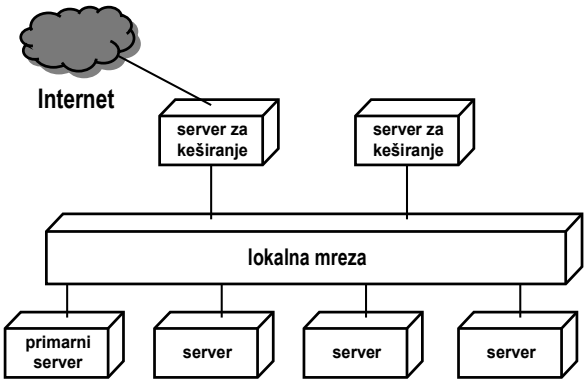
Dijagrami složene strukture prikazuju saradnju klasa, interfejsa ili komponenti u cilju opisa strukture zadužene za izvršavanje posmatrane funkcionalnosti. Ovi dijagrami su slični dijagramima klasa. Razlika je u tome što dijagrami klasa prikazuju statičku strukturu sistema, kroz prikaz klasa sa njihovim 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 predstavljanje hardverske arhitekture sistema. Oni prikazuju delove sistema raspoređene po fizičkim lokacijama. Ovi dijagrami se mogu posmatrati i kao prikaz arhitekturalnog rešenja celokupnog sistema. Pomoću njih se prikazuje konfiguracija procesnih čvorova u toku izvršavanja i komponenti koje se nalaze u njima. Predstavljaju posebnu vrstu dijagrama klasa namenjenih statičkom prikazu raspoređenosti hardvera sistema. Sastoje se iz čvorova, koji se prikazuju u obliku kvadra, kao i relacija između ovih čvorova.

Opisani UML dijagrami mogu se koristiti u različitim fazama razvoja softvera. U fazi analize najčešće se koriste: dijagrami slučajeva upotrebe, dijagrami aktivnosti i dijagrami stanja mašine, dok se u fazi projektovanja uobičajeno koriste: dijagrami klasa, dijagrami sekvenci i saradnje, dijagrami paketa, dijagrami raspoređivanja.

Slika 17: Dijagram raspoređivanja



4.2 CASE tehnologije

Computer Aided Software Engineering (CASE) označava aktivnost razvoja softverskih proizvoda uz pomoć računara. CASE tehnologije su namenjene automatizaciji procesa razvoja softverskih proizvoda. U najopštijem slučaju, pojam CASE se upotrebljava za svaki softverski proizvod namenjen za automatizaciju bilo kojeg zadatka razvoja softvera. Saglasno tome, CASE tehnologije pokrivaju dijapazon od pojedinačnih alata za automatizaciju određenih zadataka, do kompletnih softverskih alata za automatizaciju većine koraka metodologije razvoja softverskog proizvoda. CASE tehnologije ne predstavljaju zamenu za bilo koji metod ili tehniku razvoja, već samo dodatak metodu ili tehnici u generisanju kvalitetnog proizvoda. Njihovo korišćenje je interaktivno, prilagođeno korisniku uz naglasak na upotrebu grafike.

4.2.1. Pojam, osobine i struktura CASE

CASE tehnologije su razvijene kao rezultat nastojanja osoba koje se bave razvojem softvera da unaprede sopstvenu produktivnost. Naime, ironičnom se smatrala situacija da se u oblasti primene informacionih tehnologija teži povećati produktivnost rada drugih, ignorišući pri tome potencijal računara za unapređenje sopstvene produktivnosti. Pored povećanja produktivnosti, osnovni ciljevi primene CASE tehnologije su: skraćenje vremena izrade projekata, povećanje kvaliteta i nivoa performansi softvera putem stroge primene razvojne procedure. Da bi se navedeni ciljevi postigli bila je neophodna disciplinirana primena konzistentne metodologije, čiji koraci bi se realizovali uz primenu računara. Jednom rečju,

rešenje se tražilo u automatizaciji postupaka razvoja softvera primenom CASE tehnologija.

Osnovni ciljevi primene CASE tehnologije su dakle:

- povećanje produktivnosti projektanta u aktivnostima razvoja softvera,
- skraćenje vremena izrade projekta,
- povećanje kvaliteta softvera i
- unapređenje performansi sistema.

Ideja je bila da se ovi ciljevi postignu putem disciplinirane primene konzistentne metodologije, čiji koraci bi se realizovali uz primenu računara. Jednom rečju, rešenje se tražilo u automatizaciji postupaka projektovanja softvera putem CASE tehnologije.

CASE je akronim koji je prva koristila Nastec Corporation 1982. godine. Njihov proizvod, DesigneAid, je alat kojim se logički i semantički procenjuju softver i sistemski dizajn dijagrama i gradi se rečnik podataka, a podržava strukturnu analizu i dizajn, kao i dijagrame Warnier-Orr.

Ovaj akronim poseduje dva značenja. CASE se koristi za označavanje pojma computer-aided software engineering, ali takođe i za označavanje pojma computer-aided systems engineering. U ovom udžbeniku, CASE je akronim koji označava pojam Computer - Aided Software Engineering i može se prevesti kao inžinjersko projektovanje softvera (i informacionih sistema) uz pomoć računara. U najopštijem slučaju, pojam CASE se odnosi na svaki softverski proizvod namenjen za automatizaciju bilo kojeg zadatka izrade softvera odnosno on podrazumeva korišćenje informacionih sistema u razvoju informacionih sistema.

Saglasno tome, CASE tehnologije pokrivaju dijapazon od pojedinačnih alata za automatizaciju određenih zadataka projektovanja softvera do kompletnih rešenja za automatizaciju izrade većine koraka metodologije "životnog ciklusa" za projektovanje softvera kao celine. Tako je započeo razvoj CASE tehnologije, koju danas bilo u eksperimentalnoj fazi ili u samoj praksi primenjuju sve značajnije organizacije koje se bave razvojem softvera.

CASE tehnologije se usled toliko brzog razvoja i trenutnog stanja razvijenosti ne mogu više posmatrati kao prost zbir računarskih programa. One predstavljaju mnogo više od toga. Kako tehnologija bez metodologije nije zamisliva, i organizacije koje su usvojile CASE tehnologije bez izmene primenjene metodologije nisu ostvarile značajnije koristi od njihove primene.

Značajno je napomenuti da se danas CASE tehnologije ne mogu shvatiti kao prost zbir alata koji su namenjeni razvoju softvera, već kao sistemi koji integrišu sledeće komponente: (a) hardver, (b) softver, (c) bazu podataka, (d) procedure, (e) kadrove. U CASE terminologiji integralna, celina hardverskih i softverskih komponenti se

naziva CASE alatom. Procedure se nazivaju CASE metodologijom, a baza podataka CASE enciklopedijom. Uspešna primena CASE tehnologije pretpostavlja usvajanje odgovarajuće metodologije razvoja softverskog proizvoda. Ukoliko se ovom zahtevu ne udovolji, izostaće pozitivni efekti primene CASE tehnologije.

Bitne karakteristike automatizacije predstavljaju:

- provera međusobne usaglašenosti rezultata koraka metodologije,
- provera kompletnosti realizovanih aktivnosti na projektu i
- vođenje ažurne dokumentacije.

CASE alati zahtevaju novi i disciplinirani pristup softverskom inženjeringu, odnosno CASE alati moraju biti selektirani i primenjeni u onim aplikacijama za koje su i namenjeni. Osoblje koje radi sa njima mora biti na adekvatan način obučeno. Naime, CASE alati su samo sredstva u rukama softver inženjera ili drugih korisnika, čiji efekat zavisi ne samo od kvaliteta alata već i od sposobnosti i znanja korisnika.

Opštu strukturu CASE tehnologije čine:

- alati za strateško planiranje,
- alati za sistem analizu,
- alati za dizajn baze podataka,
- alati za razvoj sistema,
- alati za izgradnju sistema,
- alati za upravljanje sistemima,
- alati za podršku procesima,
- alati za upravljanje projektima i
- enciklopedija.

CASE enciklopedija predstavlja bazu podataka o svim elementima razvoja informacionog sistema. Ona predstavlja sponu između svih nabrojanih komponenti strukture CASE tehnologije, i ne samo to, već pruža mogućnost da se putem nje pojedine faze razvoja informacionog sistema automatski međusobno nadovezuju i pri tome rezultati iz određene faze istovremeno stavljaju na raspolaganje alatima u narednim fazama procesa razvoja.

Treba naglasiti, da ne poseduju svi CASE tehnologije napred navedene alate u svojoj osnovnoj strukturi, mada dobavljači uglavnom teže da razviju u sopstvenoj režiji ili u saradnji sa ostalim isporučiocima određene CASE tehnologije koji čine zaokružene celine. Takođe, nemaju sve CASE tehnologije ni istu moć. Dok

pojedine poseduju dobar alat za analizu sistema, druge poseduju dobar alat za dizajn baze podataka, treće poseduju dobar alat za strateško planiranje itd. Zbog toga, organizacije u primeni kombinuju alate različitih CASE tehnologija. Upotrebljavaju tako alate iz jedne tehnologije u realizaciji jedne faze razvoja, a druge alate iz druge tehnologije u realizaciji druge faze razvoja.

Ovakva primena CASE tehnologije stvara određene probleme jer ne postoji definisana standardna struktura CASE enciklopedije. Podaci u enciklopediji koje pruža određena tehnologija uglavnom su nekompatibilni sa podacima u enciklopediji potrebnim za druge tehnologije. Različite tehnologije imaju različite standarde, pa se o tome pri izboru mora povesti računa.

CASE tehnologija ne predstavlja automatizovani razvoj sistema već metodologiju i niz alata za razvoj sistema uz visoku produktivnost rada.

4.2.2. Klasifikacija CASE tehnologije

Postoji više kriterijuma za klasifikaciju CASE tehnologije. Tako se one klasifikuju: obzirom na funkcije koje poseduju, obzirom na ulogu koju kao instrumenti u rukama upravljača ili izvršioca aktivnosti imaju, obzirom na mogućnost primene u različitim fazama razvoja softvera, obzirom na hardver i softver koji ih podržava, obrzirom na poreklo i troškove i dr. Jedan od mogućih kriterijuma pri klasifikaciji je i kompletnost CASE tehnologije koja ukazuje na broj zadataka metodologije "životnog ciklusa" čiju automatizaciju CASE tehnologija podržava. Prema ovoj klasifikaciji diferenciraju se:

- Upper CASE - CASE tehnologije namenjene za automatizaciju faze strategijskog planiranja sistema i faze upravljanja projektima.
- Middle CASE - CASE tehnologije namenjene za automatizaciju faze analize i faze dizajna.
- Lower CASE - CASE tehnologije za automatizaciju faza programiranja, testiranja i uvođenja informacionog sistema.

Prema integralnosti CASE tehnologije se strukturiraju na:

- CASE tool - CASE alati koji automatizuju pojedine aktivnosti u fazama razvoja informacionog sistema. Koriste moćnu grafičku podršku za opis i dokumentovanje sistema kao i za dizajn korisničkog interfejsa.
- CASE toolkit - CASE paket alata ili komplet predstavlja paket koji se koristi za automatizaciju razvoja jedne faze ili određene funkcije kroz više faza razvoja informacionog sistema (npr. projektovanje baze podataka).
- CASE workbench - CASE tehnologije koje služe za automatizaciju svih zadataka kroz faze razvoja informacionog sistema i predstavljaju integrisanu kolekciju CASE paketa. Kombinacijom kolekcije CASE paketa sa odgovarajućom hardverskom jedinicom dobija se radna stanica za razvoj

softvera - CASE workstation. Prilikom izbora kolekcije CASE alata treba imati na umu, sa jedne strane, njihove karakteristike i vrste sistema kojima su namenjene, a takođe i zahteve korisnika i karakteristike sistema za koji se želi primeniti CASE kolekcija.

U zavisnosti koje faze životnog ciklusa sistema pokriva CASE tehnologije se dele na Avison, D.E., Fitzgerald, G.(1995):

- Projektanski CASE - automatizuju prve tri faze životnog ciklusa: strategijsko planiranje, analizu i dizajn;
- Programerski CASE - automatizuju naredne tri faze životnog ciklusa: programiranje, implementaciju i eksploataciju i održavanje;
- Integrisani CASE - podržava sve faze životnog ciklusa razvoja sistema.

Ukoliko se kao primarni kriterijum u klasifikaciji CASE tehnologije koriste funkcije koje isti poseduju onda se mogu razlikovati sledeće grupe proizvoda:

- Alati za planiranje poslovnih sistema - CASE tehnologije ove klase podržavaju aktivnosti praćenja tokova informacija između organizacionih jedinica.
- Alati za upravljanje projektima - CASE tehnologije ove klase podržavaju aktivnosti: procene vrednosti projekata, planiranja projekata, lociranja resursa, prikupljanja informacija i upravljanja podacima, analize rizika, praćenja troškova, obezbeđenja kvaliteta, verifikacije standarda, merenja produktivnosti i dr.
- Alati podrške - CASE tehnologije ove klase podržavaju aktivnosti koje se upražnjavaju kroz ceo proces softverskog inženjeringa. To su alati za podršku dokumentovanja, alati za podršku sistemskog softvera, alati za podršku obezbeđenja kvaliteta, alati za podršku upravljanja bazama podataka i dr.
- Alati za analizu i dizajn - CASE tehnologije ove klase omogućuju softver inžinjeru da kreira model sistema koji će se graditi. Model sadrži prikaz tokova podataka, sadržaj podataka, prikaz procesa, prikaz kontrola i dr. Alati za analizu i dizajn pomažu u kreiranju modela i razvoju njegovog kvaliteta. Obavljanjem provere konzistentnosti i vrednosti modela ovi alati omogućuju da se otklone greške pre nego što se priđe dizajnu ili samoj implementaciji.
- Alati za programiranje - CASE tehnologije ove klase podržavaju aktivnosti kreiranja programskih rešenja. To su konvencionalni alati za podršku programskim jezicima (kompajleri, linkeri, asembleri), editori, simulatori setova instrukcija, generatori koda, generatori modula i dr.

- Alati integracije i testiranja - CASE tehnologije ove klase se koriste pri integraciji proizvedenog softvera sa okolinom u kojoj će isti funkcionisati. To su alati koji omogućuju prikupljanje podataka koji će se koristiti tokom testiranja, alati kojima se analizira izvorni kod bez vršenja testa, alati kojima se analizira izvorni kod tokom obrade, alati koji simuliraju funkcije hardvera i ostalih spoljnih elemenata, alati koji pomažu u planiranju, razvoju i proveri testa i dr.
- Alati prototipskog razvoja - CASE tehnologije ove klase služe u modelu prototipskog razvoja softvera.
- Alati za podršku održavanja - CASE tehnologije ove klase služe kao podrška održavanju koje angažuje otprilike 70% ukupnih napora na razvoju softvera. Podržavaju tri grupe funkcija: inženjering u suprotnom smeru (na osnovu izvornog koda kao ulaza, generisanje analitičkog i razvojnog modela), rekonstrukcija koda i reinženjering.

4.2.3. Efekti primene CASE tehnologije

Pri razvoju CASE tehnologije specijalna pažnja se posvećuje automatizaciji prve tri faze životnog ciklusa. Razlog za to su visine relativnih troškova otklanjanja greške u pojedinim fazama životnog ciklusa. Što se greška kasnije otkrije, to više košta njeno otklanjanje.

Efektivna primena CASE tehnologije značajno doprinosi kvalitetu softverskog inženjeringa. CASE tehnologije su korisne jer izvršavaju svoju funkciju, štede vreme, štede radnu snagu, štede novac ili omogućuju nešto što je bez njih teško ili uopšte ne izvodivo. Produktivnost organizacije se povećava, a greške u razvoju softvera se značajno smanjuju. CASE tehnologije u razvoju softvera doprinose nizu pozitivnih efekata:

- grafička prezentacija modela sistema,
- detekcija grešaka i korekcija nekonzistentnosti,
- interaktivna izrada prototipa sistema,
- identifikacija komponenti sistema koji se mogu ponovo upotrebiti u razvoju,
- efektivno upravljanje razvojem sistema,
- efikasna kontrola utrošenog vremena u razvoju,
- kontrola trošenja sredstava predviđenih za razvoj,
- automatizovano generisanje uvek ažurne dokumentacije i drugi.

Upravo zbog navedenih efekata, CASE tehnologije se preporučuju kao rešenje mnogih problema imanentnih tradicionalnom načinu razvoja sistema, kao što su kašnjenje u izvedbi, netačnost, teško izvođenje izmena i loša dokumentovanost. Za

sada, period primene CASE tehnologija je relativno kratak da bi se mogli tačnije proceniti efekti. Postojeće studije pokazuju da se poboljšanja kreću od 30 do 600%. Bez obzira na neusaglašenost ovih rezultata ohrabruje da većina studija ukazuje na poboljšanja. Povećanje produktivnosti, izgradnja mnogo tačnijih sistema u kraćem vremenskom intervalu, uticali su na one koji se bave razvojem sistema da izmene ili kompletno restruktuiraju svoje metodologije razvoja sistema kako bi imlementirali CASE tehnologije.

4.2.4. Osobine CASE tehnologije

Generalno posmatrano, razvoj CASE tehnologije zavisi od specifičnosti zahteva korisnika, okruženja u kojem se ista želi primeniti i ideja o tome kako bi tehnologija mogla da funkcioniše. Standardi u izgradnji CASE tehnologija ne postoje. CASE tehologije su brojne, što se uočava i kroz napred navedenu grubu i opštu analizu mogućnosti primene određenih tehnologija u pojedinim fazama razvoja softvera. One pokrivaju većinu aktivnosti tih faza.

Obeležja koja treba da poseduju proizvodi CASE tehnologije koji pripadaju klasi kvalitetnih tehnologija su sledeća:

1. Jednostavno i lako korišćenje - Ova osobina predstavlja meru efektivnosti tehnologije u odnosu na korisnika. Nezavisno od funkcionalnosti i potpunosti, CASE tehnologija treba da je tako kreirana da je korisnik može lako, jednostavno i bez puno razmišljanja i neizvesnosti koristiti u rešavanju projektantskih zadataka. Ukoliko to nije slučaj, tada korisnik svoje vreme provodi u razmišljanju kako bi tehnologiju koristio ili kako tehnologija uopšte radi, a to znači da ista svojom ulogom ne pomaže već samo predstavlja smetnju ili čak prepreku u radu.

2. Tehnologija treba da poseduje mogućnost da otkriva greške korisnika, a poželjno je da sama te greške i otklanja. Tehnologija treba da omogući interakciju sa korisnikom putem dijaloga koji je razumljiv i prihvatljiv za korisnika. Ona treba da je fleksibilna i da se može kombinovati sa drugim tehnologijama kako bi se zadovoljili različiti zahtevi korisnika. Posebno je značajno da se minimiziraju ili potpuno isključe nepredviđena reagovanja tehnologije, koja obično rezultiraju nezadovoljstvom korisnika. CASE tehnologija ne bi smela svojim izlazima da iznenađuje, zbunjuje i blokira rad korisnika. Naredbe koje korisnik upotrebljava treba da su jasne, jednostavne i konzistentne.

3. Podobnost - Kvalitetna je ona tehnologija koja poseduje takav nivo podobnosti i performansi da može da podrži rešavanje velikog broja zadataka u razvoju softvera. Podobnost se izražava kroz obim kojim jednostavne naredbe tehnologije mogu uzrokovati njene glavne efekte. Uz ovo obeležje pridružuje se i zahtev da CASE tehnologija treba da bude u mogućnosti da pruža informacije o sopstvenom stanju. Dobra tehnologija može impresionirati i mnogo većom podobnošću nego što je ona ustvari, ukoliko poseduje više znanja o sopstvenom stanju.

4. Velika snaga tj. moć - Snaga tehnologije se ocenjuje na osnovu kombinacije sledećih faktora:

- pouzdanost tehnologije,
- osobine tehnologije koje se ispoljavaju pri oskudnim ili lošim uslovima,
- funkcionisanje,
- težina posledica nedostataka tehnologije,
- konzistentnost aktivnosti u funkcionisanju tehnologije i
- način na koji se tehnologija integriše u okruženje.

5. Pouzdanost je značajan zahtev koji se ispoljava kao sposobnost alata da rastereti korisnika od rizika greške koju sam napravi. CASE tehnologija treba da otkriva odnosno otklanja greške i posledice koje zbog istih nastanu. Tehnologija treba da poseduje sopstveni mehanizam samotestiranja koji obezbeđuje njegovo pravilno funkcionisanje.

6. Konzistentnost aktivnosti tehnologije potvrđuje veličinu tj. obim njene snage. Konzistentnost podrazumeva dobro definisanu sintaksu i semantiku. Istovremeno, tehnologija mora biti tako razvijana da podržava kompatibilnost između pojedinih verzija.

7. Funkcionalnost - Ova osobina nije definisana samo zadatkom zbog kojeg je tehnologija dizajnirana već i metodima koji se upotrebljavaju u izvršenju zadataka. Broj tehnologija koje podržavaju metodologije je veoma velik. Efikasnost koju ispoljava tehnologija u podršci metoda može neposredno doprineti razumevanju i definisanju osobina tehnologije, kao i određenju kvaliteta i korisnosti izlaza koje obezbeđuje.

8. CASE tehnologije integrišu metode i povezuju ih sa metodologijom. Pojedine tehnologije podržavaju jedno, više ili sva područja metodologije i prenose rezultate između faza. Koncepcija i struktura CASE tehnologije je determinirana izabranom metodologijom i njoj pripadajućim metodama i tehnikama. Tehnologija mora da obezbedi konzistentnu primenu metodologije i metoda na kojima se zasniva. Tako ona mora funkcionisati korektno i kontrolisati da li se metodologija sprovodi u potpunosti. Pored toga, tehnologija mora generisati izlaze koji su korektni i striktno definisani metodologijom.

9. Lako povezivanje sa postojećim sistemom - CASE tehnologija se mora lako i nesmetano uvesti u postojeći informacioni sistem. Ona se jednostavno instalira i omogućuje da se postojeće strukture datoteka ili baze podataka koriste na isti način kao i pre njene upotrebe. CASE tehnologija mora omogućiti i prenos podataka odnosno njihovu razmenu između različitih CASE tehnologija koje se već koriste u organizaciji.

10. Kvalitet podrške CASE tehnologije - Prilikom vrednovanja tehnologije sa aspekta kvaliteta podrške, značajno je sagledati i sledeće elemente podrške:

- reputacija dobavljača,
- zrelost tehnologije i njena rasprostranjenost,
- mogućnost smanjenja troškova pri kupovini više kopija,
- mogućnost iznajmljivanja tehnologije,
- mogućnost vraćanja tehnologije uz povrat sredstava,
- mogućnost dobijanja punih prava i pristupa izvornom kodu,
- mogućnost i uslovi održavanja,
- vreme odziva u održavanju,
- pružanje pomoći u obezbeđenju problematičnih odgovora,
- da li korisnik raspolaže pravom na nove verzije tehnologije bez naknade,
- koji je rok garancije,
- koji su rokovi isporuke,
- kakvi su uslovi obuke za korisnike tehnologije,
- da li postoje efikasni programi obuke i
- kakva su stručna i pedagoška svojstva kadrova koji vrše obuku.

CASE tehnologije obezbeđuju integralno razvojno okruženje koje je nezavisno od metodološkog prilaza u razvoju sistema i pruža podršku svim aktivnostima od definisanja, preko razvoja do održavanja softvera. Pri tome, omogućujući optimalno funkcionisanje sistema uz najmanji mogući procenat grešaka.

4.2.5. Načini integracije CASE tehnologija

CASE tehnologije se mogu na različite načine integrisati. Mali je broj alata koji su potpuno nezavisni i tako se koriste. Takvi alati kreiraju samostalno izlaze u vidu dokumenata, programa, podataka i njihova veza prema ostalima u okruženju je papir koji prenosi graditelj softvera. U stvarnosti, CASE tehnologije se povezuju i razlikujemo sledeće načine integracije:

Razmena podataka (Data exchange) - je najčešći slučaj i mogućnost koju poseduje većina alata. To je slučaj kada alat izlaze, koje kreira, prevodi u oblik nestruktuirane datoteke u formatu štampe. To omogućuje da se zaštite podaci alata, eliminiše se potreba ponovnog unošenja elemenata specifikacije dizajna i sprečavaju štamparske greške. Prevodioci se uglavnom razvijaju zajedničkom akcijom isporučilaca alata i mogu se od istih nabaviti. Međutim, prevodioci su razvijeni i od strane konsultanata i korisnika od kojih se takođe mogu kupiti.

Nedostatak ovog načina je što se uglavnom samo deo prevedenih podataka može na ovaj način prihvatiti i koristiti u drugom alatu, sve dok se ne definiše kompatibilnost. Takođe, nedostatak je i što stalni razvoj softvera zahteva da se i nakon manjih izmena datoteke prevode i prenose. Razne verzije mogu lako dovesti do odsustva sinhronizacije.

Ukoliko se koristi veliki broj alata na jednom projektu, neprihvatljivo je masovno transferisanje i prevođenje, posebno što je ono još i jednosmerno. Naime, ne postoje mogućnosti da se vrše izmene obostrano, a što znači da dolazi u pitanje održavanje integralnosti konfiguracije kroz nizove alata koji se upotrebljavaju.

Zajednički pristup alatima (Common Tool Access) - predstavlja sledeći nivo integracije, koji omogućuje korisniku da poziva veći broj različitih alata na isti način, npr. iz pull-down menija. U multitasking okruženju to znači da korisnik može simultano otvarati alate, ručno koordinirati ulaze u njih i upoređivati izlaze dizajna na način kako želi. Npr. korisnik može istovremeno prikazati dijagrame tokova podataka, dijagrame strukture, rečnik podataka i izvorni kod koje održavaju različiti alati. U ovom okruženju, razmena podataka iz alata u alat može biti uprošćena uvođenjem procedure prevođenja običnim izborom menija ili izborom makro funkcije.

Zajedničko upravljanje podacima (Common Data Management) - predstavlja način integracije pri kojoj se podaci iz različitih alata održavaju u jednoj logičkoj bazi podataka, koja može biti fizički decentralizovana ili centralizovana. Integracija na ovaj način uprošćava razmenu informacija i podiže nivo integralnosti podataka, pošto svaki alat ima stalni i trenutni pristup najaktuelnijim podacima odnosno uvek ažurnim podacima. Istovremeno, prava pristupa se mogu kontrolisati, a može se upravljati i verzijama alata. Način kontrole je ručni putem procedura prijavljivanja i odjavljivanja. Tipično, postoji funkcija integrisanja podataka koja omogućuje graditeljima softvera da rade na različitim delovima aplikacija i kombinuju poslove. Ukoliko ovakva grupa alata poseduje karakteristiku provere, tada je moguće utvrđivanje nekonzistentnosti između rezultata pojedinih autora.

Mada se podacima iz različitih alata upravlja zajedno na zajedničkom nivou, alati ne poznaju međusobno interne strukture podataka i semantiku predstavljanja u dizajnu. Potreban je i dalje korak prevođenja koji se manuelno aktivira da bi se jednom alatu omogućilo korišćenje izlaza drugog alata.

Podela podataka (Data Sharing) - je način integracije pri kojem alati poseduju kompatibilne strukture podataka i semantiku i mogu se direktno upotrebljavati bez ikakve transformacije. Svaki alat je tako dizajniran da bi bio kompatibilan sa ostalima u uslovima podele podataka. Zbog toga, podela podataka se javlja uglavnom kod alata istog proizvođača ili kod više proizvođača koji su se strateški povezali upravo da bi proizveli takav proizvod (čak možda i na zahtev jednog korisnika). Za realizaciju ovog načina povezivanja, ključnu ulogu su odigrali standardi koji obezbeđuju dobru osnovu za integraciju CASE alata.

Međusobna operatibilnost (Interoperability) - je najviši nivo integracije pojedinačnih alata i javlja se ukoliko je realizovan zajednički pristup i podela podataka.

Da bi se postigla puna integrisanost CASE okruženja, neophodna su još dva dodatna elementa: upravljanje meta podacima i sredstva kontrole.

Meta podaci su podaci proizvedeni od pojedinih CASE alata, a nose informacije o procesu softverskog inženjeringa. Meta podaci uključuju:

- definicije objekata,
- veze i zavisnosti između objekata proizvoljne detaljnosti,
- pravila dizajna softvera,
- tokove procesa, procedure i događaje.

Sredstva kontrole omogućuju pojedinačnim alatima da obaveste ostatak okruženja (ostale alate, rukovodioca podataka,...) o značajnim događajima i pošalju zahteve za akcijom ostalim alatima i servisima putem trigera. Npr. alat za dizajn bi morao obavestiti alat za upravljanje konfiguracijom da je kreirana nova verzija dokumenta dizajna i da je potrebno da ovaj izvrši proveru konzistentnosti. Sredstva kontrole pomažu u održavanju integriteta okruženja i obezbeđuju sredstva za automatizaciju standardnih procesa i procedura.

4.2.6. Ocena i izbor CASE tehnologije

Proces ocene vrednosti i izbor CASE tehnologije se izvodi od strane organizacije koja ga želi nabaviti. Svaka organizacija ima sopstvene potrebe i zahteve pri nabavci tehnologije. Proces procene čine sledeći metodološki koraci:

- analiza potreba i zahteva korisnika,
- analiza postojećeg okruženja,
- identifikovanje potencijalne liste CASE tehnologija i
- primena kriterijuma za ocenu kvaliteta i izbor tehnologije.

Analiza potreba i zahteva korisnika predstavlja veoma značajan korak u izboru CASE tehnologije. Organizacija treba da opredeli koji model razvoja softvera će primeniti, koji su osnovni tehnički i upravljački zadaci. Tačno se moraju identifikovati oni zadaci koji se žele potpuno ili delimično realizovati uz pomoć automatizovanih alata.

Analiza postojećeg okruženja je u potpunosti povezana sa prethodnim metodološkim korakom. CASE tehnologija mora potpuno da bude prikladna okruženju. Ograničenja su moguća i ona se moraju uzeti u obzir. Tako npr. novac, vreme, iskustvo zaposlenih, dosadašnja praksa, odnosi sa dobavljačima i dr. Ova

ograničenja ne samo da se identifikuju nego se i analiziraju sa aspekta njihove moguće promene ili otklanjanja.

Identifikovanje potencijalne liste CASE tehnologija predstavlja korak kojim se potrebama i zahtevima korisnika pridružuju moguće tehnologije koje bi iste zadovoljili. Trenutno je izbor tehnologija veoma velik, a poseduje tendenciju daljeg brzog razvoja. Prezentacije, reklamni materijali i dr. obezbeđuju inicijalne informacije o postojećim tehnologijama.

Poslednji korak ove metodologije je sigurno najznačajniji. Svi identifikovani kriterijumi se primenjuju na svaku od CASE tehnologija koje su identifikovane u potencijalnoj listi. Kriterijumi se postavljaju za potrebe najobjektivnije selekcije. Troškovi i vreme su pri tome na vrhu liste. Ukoliko postoji mogućnost nije na odmet da se dobavljač poseti i na licu mesta tehnologija upozna i testira.

Beleške i zapažanja sa testa treba da su opredeljujući u određenju koliko tehnologija zadovoljava postavljene kriterijume. Posebna pažnja se pridaje kriterijumima koji su najviše rangirani. Konačna odluka treba da se bazira i na mišljenju zaposlenih u organizaciji koji bi trebali da ostvare najveću korist od primene tehnologije.

4.2.7. Pravci razvoja CASE tehnologije

CASE tehnologije kao elementi softverskog inženjeringa treba da ponude odgovor na njegove trenutne i buduće potrebe. To su: potreba za utvrđivanjem jedinstvenih standarda, primena instrukcija na prirodnom jeziku čoveka, stvaranje opštih šablona i dr. Standardizovane CASE tehnologije moraju obuhvatati brojne formalne metode i standarde i obezbeđivati adekvatan tj. visok nivo dokumentovanosti. Takođe, CASE tehnologije bi morale pomoći u upravljanju projektom razvoja softvera i omogućiti njegovo bolje i lakše održavanje. Pravilna primena CASE tehnologija doprinela bi efikasnosti upravljanja troškovima, skraćenju vremena izrade i izradi kvalitetnih proizvoda.

CASE tehnologije od kojih se očekuje da zadovolje ove zahteve svrstavaju se u sledeće tri grupe:

- inteligentne CASE tehnologije,
- CASE tehnologije za prototipski razvoj softvera i
- CASE tehnologije podrške.

Inteligentne CASE tehnologije su u pravom smislu alati budućnosti. One treba da omoguće razvoj softvera za kompleksne računarske sisteme uz minimalno učešće čoveka. One treba da su efektivne i efikasne sa aspekta troškova. Tehnologije iz ove grupe treba da omoguće da se automatizmom odaberu pogodni metodi za razvoj softvera i kompletiraju zahtevi. Pored toga, treba da vode brigu o svim detaljima i spreče pojavu nekonzistentnosti.

Komponente ovih tehnologija budućnosti su:

- moduli funkcija koji odražavaju zadatke u analizi i dizajnu softverskog sistema,
- simulacija, kojom se proverava ponašanje projektovanog softvera,
- dokumentacija, koja treba da pruži mogućnost inžinjerima da prate razvoj zahteva kroz različite faze razvoja softvera,
- matrice, koje će signalizirati nedostatak inputa ili outputa i pomagati u očuvanju softvera od logičkih grešaka,
- rečnik podataka, kao skladište za module funkcija, dokumentaciju i matrice i
- programski jezik, koji bi bio primeren zahtevima opisa funkcija u formi tvrdnji, uslova i promena.

Neke osnovne karakteristike inteligentnih CASE tehnologija uključivale bi i sledeće elemente:

- pokrivaju razvoj softvera u svim fazama životnog ciklusa softvera,
- uključuju planiranje,
- poseduju alate zasnovane na znanju i bogatu grafičku podršku,
- podržavaju automatizovano kodiranje, testiranje i dokumentovanje,
- podržavaju softverski reinženjering,
- omogućuju ponovno korišćenje softvera,
- obezbeđuju potpunu konzistentnost programa i sistema u celini,
- testiraju projekat i pre kodiranja, a kodirani moduli automatski se integrišu u celinu itd.

CASE tehnologije za prototipski razvoj softvera treba da potpomognu u budućnosti efikasnu i efektivnu komunikaciju između krajnjih korisnika i projektanata. Osnovna ideja je da se upotrebe u izradi eksperimentalnog prototipa pre nego što se korisnik odluči da uđe u velike investicije. Ulazni ekrani i meniji se moraju razvijati dok izlazni ekrani i izveštaji prikazuju šta će korisnik videti. Programi se pišu tek posle odluke da se softver izrađuje.

Simulacija će se upotrebljavati za proveru i definisanje performansi kao što su: vreme pristupa bazi podataka, vreme generisanja odgovora, vreme prolaza i dr. potrebnih u sistemu. Simulacijom će se demonstrirati ulazi i u zavisnosti od njih promenljivi izlazi kakve će ih korisnik dobijati u izvršenju.

CASE tehnologije podrške su koncipirane tako da mogu sa uspehom podržati, nezavisno jedna od druge, aktivnosti upravljanja projektom, vrednovanje

kvalitativnih performansi, generisanje test slučajeva. Ove tehnologije treba da se u daljem periodu razvoja integrišu sa inteligentnim CASE tehnologijama i CASE tehnologijama za prototipski razvoj softvera.

U nastavku su dati nazivi poznatijih CASE tehnologija i njihovih proizvođača:

CASE tehnologija	Metodologije razvoja	Tehnike i razvojni alati	Sistemi upravljanja bazama podataka
IEW	Warnier-Orr		SQL, DB2, IMS-DL/1, Oracle
Promod PLUS	Yourdon, de Marco, Hatley/Pirbhai	Uniface Six, Source Pilot, C, Fortran	Sybase, Oracle, Informix, Ingres, OCM
Oracle CASE	J. Martin, de Marco, Ernst & Young	Uniface Six, CASE Generator SQL, Forms	Oracle, DB2
Synthesis	Yourdon, Coad, Constantine, Ross	C, C++, Fortran, Cobol, Magic	Novell Btrieve, SQL, Sybase, Oracle, Informix
Westmount ISEE	Yourdon, de Marco, Chen	Ingres 4GL, Informix 4GL, Uniface Six	Ingres, Informix, Sybase
Westmount I-CASE	Ward-Mellor, de Marco, Chen, SSADM	Ingres 4GL, Informix 4GL	Ingres, Informix, SQL
PTECH	Martin-Odell, OOAD	sopstveni ugrađeni C++ generator koda	OODBMS
Paradigm Plus	Rumbaugh OMT, Martin-Odell, OOIE, Booch OOAD, Coad, Yourdon, Shalaer-Mellor, AOOD	ProtoScrpt, C, C++, Ada, SmallTalk, PowerBuilder, SQL, JAVA, Corba IDL, Visual Basic	ORACLE 7, dBase, DB2, uniSQL, Access, Centura, SQLB ase, Sybase/SQL, objectStore, gemStone
Rational CASE family	Rumbaugh OMT, Booch OOAD, Jacobson Objectory Use Case, UML	C, C++, Forté, Java, SmallTalk, PowerBuilder, Gupta SQLWindows, VisualBasic	Oracle 7, Sybase, SQLBase, SQLServer, Watkom SQL, Ansi SQL

Ništa nije praktičnije od dobre teorije.

Nils Bor

6. Standardi softverskog inžinjeringa

Da bi se oblast softverskog inžinjeringa pokrila neophodnim standardima, uložen je ogroman trud i rad na međunarodnom, regionalnom i nacionalnom nivou. Proces donošenja i revizije već donetih standarda je stalan i sve obuhvatniji. U svetu softvera tako danas egzistira više od 1000 standarda, a u svetu softverskog inženjerstva ima ih više od 350 aktuelnih. Pod ovim se ne podrazumevaju tzv. de facto standardi, kao što su, na primer, brojna rešenja Microsofta, a kojih je verovatno i više od zvaničnih standarda. Mnoge kompanije se osećaju izgubljeno u ovj pravoj "šumi" standarda.

Ne treba očekivati da su svi međunarodni standardi savršeno sistematizovani, usklađeni po metodologiji i sadržaju, ali su van svake sumnje snažno sredstvo, uz čiju pomoć se može vršiti unapređenje projektantskih metoda i obezbediti kvalitet gotovih softverskih proizvoda.

Međunarodni standardi vezani za informacione tehnologije donose se u združenom tehničkom komitetu (JTC) koji su formirali International Organization for Standardization (ISO) i International Electrotehnical Committee (IEC). Tehnički komitet je podeljen u 4 grupe u kojima rade 19 podkomiteta koji pokrivaju pojedine oblasti informacione tehnologije. Tako je među ISO standardima njih 88 vezano na softver. Softverski inžinjering je pod podkomitetom SC7 koji se angažuje na izradi uputstava za upravljanje i standardizaciju metoda i alata za razvoj, održavanje i ocenu kvaliteta tokom životnog ciklusa softvera. U okviru podkomiteta radi 9 radnih grupa.

Savezni zavod za standardizaciju je prihvatio preporuke međunarodnih i evropskih komiteta za standardizaciju i za jugoslovenske standarde JUS uglavnom preuzima standarde ISO/IEC pri tome dajući im nacionalni predgovor i potrebna objašnjenja. Nažalost, proces standardizacije softvera u Srbiji je u priličnom za svetskom standardizacijom u ovoj oblasti.

6.1. Karakteristike dobrog standarda

U razvoju softvera prema izloženom se može izvesti konstatacija da postoji veliki broj standarda. Pored navedenih organizacija i pojedine razvojne organizacije su uspostavile svoje specifične standarde. Međutim, većina od njih je slabo dokumentovana. Karakteristike dobrog standarda su pored ostalog da:

- poseduje adekvatnu strukturu,

- uspostavlja uniformnost,
- odgovara disciplini softverskog inžinjeringa,
- rigorozan jer precizno prati procese, standarde i procedure,
- kompatibilan i može biti upotrebljen uz druge standarde,
- obezbeđuje kvalitetan proizvod,
- uključuje kvantitativnu povratnu ocenu,
- uključuje razumnu dokumentaciju,
- obezbeđuje kvalitetne aktivnosti i razvoj,
- standardizuje proces razvoja softvera i upravljanje procesima i
- obezbeđuje sredstva za razvoj sistema koja su efikasna i efektivna.

U nastavku se daje pregled standarda koji su predmet rada Komisije za softverski inženjering (KSI 1/07) Saveznog zavoda za standardizaciju, kao i izvestan broj standarda koji se vezuje na ovu oblast te čini celinu. Svi dole navedeni standardi su preuzeti od ISO/IEC, od kojih je veći deo usvojen kao JUS standard. Standardi koji se prikazuju su:

- standard termina,
- standardi dokumentovanja,
- standardi vrednovanja softverskih proizvoda i
- standardi životnog ciklusa razvoja softverskih proizvoda.

6.2. Standard termina

JUS ISO/IEC 2382-20/94: Rečnik deo 20 - Razvoj sistema

Ovaj standard je u prvom redu namenjen jednoobraznom međunarodnom komuniciranju u oblasti informacione tehnologije. Srpski prevod obezbeđuje lakše praćenje standarda softverskog inžinjeringa. Sem toga on predstavlja pogodnu osnovu za formiranje i usvajanje jedinstvene terminologije.

Standard je urađen u vidu rečnika termina, sa uporednim nazivima na engleskom i francuskom jeziku. Date su definicije 60 najčešće korišćenih termina iz oblasti opštih pojmova informacione tehnologije, analize i projektovanja sistema, implementacije sistema, obezbeđenja kvaliteta, dokumentovanja sistema i upravljanja projektima.

6.3. Standardi dokumentovanja

JUS ISO 6592/94: Smernice za dokumentovanje aplikativnih sistema zasnovanih na računarima.

Definisane smernice ovog standarda imaju sledeće ciljeve:

- razmena informacija i obezbeđenje usaglašenosti između učesnika u razvoju aplikativnih sistema,
- izrada dobro planirane i standardizovane dokumentacije aplikativnih sistema,
- izrada dokumentacije uporedo sa razvojem aplikativnih sistema,
- kontrola procesa razvoja aplikativnih sistema.

Ovaj standard pruža elemente i sadržaj dokumentacije svake faze životnog ciklusa razvoja proizvoda, što omogućuje planiranje i kontrolu procesa razvoja aplikacija od faze studije izvodljivosti do faze korišćenja. Standard sadrži:

- princip izrade dokumentacije,
- sadržaj studije izvodljivosti,
- sadržaj idejnog projekta sistema,
- sadržaj rezultata projektovanja i izvođenja sistema (sadržaj programske dokumentacije, način i sadržaj dokumentacije podataka, sadržaj dokumentacije procedura za izradu aplikacija),
- sadržaj dokumentacije potrebne za podršku radu aplikativnih sistema (način podrške, priručnici, priručnici za korisnike, programere, operatere, podatke i dr.),
- sadržaj dokumentacije za uvođenje aplikativnog sistema,
- periodične preglede posle uvođenja,
- način upravljanja dokumentima.

Ovaj standard je do skora bio najdetaljniji standard za izradu dokumentacije razvoja IS ili njegovih delova. Nažalost standard je star (1985) kako koncepcijski tako i terminološki. Stoga je njegova revizija u toku kako bi uobličio ideje o savremenoj organizaciji dokumentacije projekata softverskog inžinjeringa. Tako inovirani sistem sadržavao bi:

- dokumentaciju zahteva sistema (problema, ciljeva, terminologije, zakonskih uslova i dr.)
- dokumentaciju opisa sistema (detaljni zahtevi, hardverske i softverske komponente sistema, metode i alati za testiranje, instaliranje i korišćenje sistema, obuka kadrova i dr.)
- indikatore vrednovanja (troškove, beneficije, dobre strane i slabosti, kriterijume vrednovanja, metode vrednovanja, rezultate vrednovanja) i

- zaključke i preporuke.

JUS ISO/IEC TR 9294/94: Smernice za upravljanje softverskom dokumentacijom

Ovaj standard je namenjen rukovodiocima razvoja informacionih sistema ili njegovih delova u cilju formiranja kvalitetne dokumentacije projekta i softverskog proizvoda. Obrađuje principe, standarde, procedure, resurse i planove u svim fazama životnog ciklusa razvoja softvera bez obzira na veličinu projekta. Smernice opisuju postulate i značaj dokumentacije, a zatim utvrđuju principe izrade dokumentacije u razvoju IS. Standard tretira:

- ulogu rukovodioca,
- funkcije softverske dokumentacije,
- utvrđivanje politike dokumentovanja,
- utvrđivanje standarda i smernica za izradu dokumentacije,
- utvrđivanje procedura izrade i rukovanja dokumentacijom,
- raspored resursa za izradu dokumentacije,
- planiranje izrade dokumentacije.

JUS ISO9127/94: Dokumentacija za korisnika i propratne informacije softverskih paketa masovne upotrebe

Korisnički softverski paketi za masovnu upotrebu su softverski proizvodi za standardne namene. Informacije sadržane u propratnoj dokumentaciji koju korisnik dobija uz softverski paket su često jedino sredstvo preko koga proizvođač ili isporučilac komunicira sa kupcem. Stoga je obezbeđenje adekvatnog obima informacija korisniku za potrebe uspešnog korišćenja softvera od posebnog značaja.

Standard utvrđuje potrebne elemente dokumentacije namenjene korisniku za uspešnu instalaciju i korišćenje softverskog proizvoda i ujedno pomaže potencijalnom kupcu da utvrdi mogućnost primene softverskog proizvoda saobrazno svojim zahtevima. Standard sadrži:

- dokumentaciju za korisnika (elemente, zahteve, obuku i dr.),
- propratne informacije (namena, uputstvo za upotrebu, adrese servisa i dr.),
- standarde karakteristika kvaliteta.

6.4. Standardi vrednovanja softverskih proizvoda

JUS ISO/IEC 9126/94: Vrednovanje softverskih proizvoda i smernice za njihovu upotrebu

Standard definiše šest karakteristika softvera koje opisuju kvalitet softvera. Ispitivanjem i ocenjivanjem karakteristika kvaliteta vrši se vrednovanje kvaliteta softvera. Svaka karakteristika softvera se sastoji od niza podkarakteristika. To su sledeće karakteristike:

- funkcionalnost - skup svojstava kojima se utvrđuje ispravnost rada pojedinih funkcija softverskog proizvoda,
- pouzdanost - skup svojstava softvera da održi nivo performansi pod određenim uslovima za određeni period vremena,
- upotrebljivost - skup svojstava softvera kojima se procenjuje lakoća njegovog korišćenja i rukovanja,
- efikasnost - skup svojstava softvera koja određuju odnose između nivoa performansi softvera i upotrebljenih resursa pod određenim uslovima,
- pogodnost za održavanje - skup svojstava softvera koja predstavljaju napore da se izvrši određena modifikacija sistema i
- prenosivost - skup svojstava softvera koja predstavljaju mogućnost softvera da se prenese iz jednog u drugo okruženje.

JUS ISO/IEC 12119: Zahtevi kvaliteta i ispitivanje kvaliteta

Standard predstavlja proširenje i dalju razradu prethodnog standarda. On postavlja skup zahteva za softverske proizvode, na bazi karakteristika softvera i daje instrukcije saglasno ovim zahtevima. Namenjen je isporučiocima i kupcima softverskih proizvoda, ovlašćenim organizacijama i laboratorijama koje vrše kvalitativna ispitivanja, vrednovanja i atestiranja softverskih proizvoda. Standard sadrži:

- zahteve kvaliteta (opis proizvoda, dokumentaciju korisnika, programi, podaci kvalitetne karakteristike i dr.),
- uputstva za ispitivanje (preduslovi ispitivanja, aktivnosti ispitivanja, predmet ispitivanja, izveštaj o ispitivanju).

ISO/IEC 9126 1-6: Vrednovanje softverskih proizvoda

Veoma važna komponenta razvoja softvera je njegovo vrednovanje. Predmet vrednovanja je softverski proizvod ali se vrednovanje proširuje na metode, alate i tehnike razvoja radi povećanja kvaliteta gotovog proizvoda. Vrednuju se karakteristike kvaliteta softvera. Standardi vrednovanja utvrđuju metode, principe, procedure i stvaraju osnovu za formiranje metodologije vrednovanja i atestiranja softverskih proizvoda. Ovim standardom se definišu glavne karakteristike softverskih proizvoda i njihove osobine, a to su:

- funkcionalnost: stepen u kojem softverski proizvod zadovoljava zadate potrebe, izražene osobinama – pogodnost, tačnost, interoperatibilnost, saglasnost i bezbednost,
- pouzdanost: procenat vremena u kojem je softverski proizvod raspoloživ za upotrebu, a može se odrediti razlažući ga na zrelost, osetljivost na greške i mogućnost oporavka,
- korisnost: stepen u kojem je softverski proizvod jednostavan za upotrebu, a koji se sagledava kroz razumljivost, jednostavnost obučavanja i lakoću rukovanja,
- efikasnost: stepen u kojem softverski proizvod koristi sistemske resurse koji se određuje analizom ponašanja u vremenu i ponašanja prema resursima,
- lakoća održavanja: jednostvanost kojom se mogu napraviti izmene na softverskom proizvodu izražava se pomoću mogućnosti analize, mogućnosti izmene, stabilnosti i mogućnosti testiranja i
- portabilnost: jednostavnost kojom se softverski proizvod može premestiti iz jednog okruženja u drugo označena je sledećim osobinama - adaptibilnost, mogućnost instalacije, podesivost i zamenljivost.

6.5. Standardi procesa životnog ciklusa razvoja softvera

ISO/IEC 12207-1/93: Procesi životnog ciklusa softvera

Koncept životnog ciklusa sadrži glavne procese razvoja, korišćenja i održavanja softvera ali i proces za kontrolu upravljanja razvojem softvera. Namenjen je isporučiocima softverskih proizvoda i usluga (projektantima, operaterima, osoblju u održavanju, osoblju koje obezbeđuje i kontroliše kvalitet) i korisnicima. Standard opisuje strukturu procesa pri čemu ne specificira detalje i ne definiše metode razvoja softvera. Ovo je po svojoj obuhvatnosti, detaljnosti i konzistentnosti najznačajniji do sada izrađen standard. On može da posluži kao dobra osnova za izradu detaljnih standarda i metodologija koje su u nadležnosti projektantskih organizacija. Standard sve aktivnosti grupiše u:

- primarne procese
- procese podrške i
- organizacione procese.

Svaki proces sadrži niz aktivnosti a svaka aktivnost niz zadataka. Tako primarni procesi obuhvataju sledeće aktivnosti:

- nabavka (aktivnosti naručioca na definisanju zahteva, izboru isporučioca, nabavci sistema ili softverskog proizvoda),

- isporuka (aktivnosti isporučioca pri isporuci sistema ili softverskog proizvoda),
- razvoj (aktivnosti projektanta pri razvoju softverskog proizvoda),
- korišćenje (aktivnosti organizacije koja obezbeđuje eksploataciju računarskog sistema, sistemskog i aplikativnog softvera) i
- održavanje (aktivnosti osoblja pri modifikaciji programa i dokumentacije).

Procesi podrške se sastoje iz:

- razvoj dokumentacije,
- upravljanje konfiguracijom (predstavlja primenu administrativnih i tehničkih postupaka radi upravljanja resursima i procedurama razvoja),
- obezbeđenje kvaliteta (aktivnosti kojima se obezbeđuje saglasnost razvoja softvera zahtevima, programu, planu),
- verifikacija (aktivnosti utvrđivanja da li su zahtevi sistemu i softveru potpuni),
- validacija (aktivnosti utvrđivanja da li sistem ili softver ispunjavaju namenu određene upotrebe),
- zajednička recenzija,
- revizija (aktivnosti određivanja saglasnosti sa potrebama, planovima i ugovorom)
- rešavanje problema tokom ciklusa.

Aktivnosti organizacionih procesa su:

- upravljanje projektom tokom životnog ciklusa,
- infrastruktura (obezbeđenje i održavanje potrebne infrastrukture),
- unapređenje (uspostavljanje, procene i unapređenja procesa životnog ciklusa)
- obuka kadrova.

6.6. Drugi domaći standardi

Značajnu grupu domaćih standarda čine standardi kvaliteta. To su:

JUS ISO 9000-3: Standardi za upravljanje kvalitetom i obezbeđenje kvaliteta,

JUS ISO 9000: Sistem kvaliteta. Model obezbeđenja kvaliteta u projektovanju / razvoju, proizvodnji, ugradnji i servisiranju,

ISO 9004-6: Upravljanje kvalitetom i elementi sistema kvaliteta (Uputstvo za obezbeđenje kvaliteta kod upravljanja projektima) i

ISO 9004-7: Upravljanje kvalitetom i elementi sistema kvaliteta (Smernice za upravljanje konfiguracijom).

Standardi ove grupe su opisani u poglavlju kvaliteta softvera.

Pored standarda kvaliteta, značajna je i grupa standarda vezanih za elektronsku trgovinu. Ovde spadaju:

JUS ISO 9735: Pravila sintakse na nivou primene,

JUS ISO 7372: Direktorij elementarnih podataka i

JUS ISO/IEC 14662-1998: Referentni model za otvoreni edi.

6.7. Standardi softverskog inženjerstva

Grupa međunarodnih ISO standarda vezanih za softversko inžererstvo sastoji se iz 29 različitih standarda. Za softversko inženjerstvo osnovni i najstariji standard je:

ISO/IEC 14102 Software Engineering.

Pored ovog prvu značajniju grupu među njima čine standardi kvaliteta softverskih proizvoda:

ISO/IEC 9126 Software engineering - Product quality.

Ova grupa sastoji se iz 4 standarda, a koji su vezani za model kvaliteta, eksterne metrike, interne metrike i metrika kvaliteta u upotrebi. Prvi od ovih standarda definiše 6 karakteristika kvaliteta, drugi opisuje merenje ponašanja softvera, trećim je dat opis merenja njegovih navedenih karakteristika, dok četvrti daje elemente za merenje efekata softvera.

Primena CASE tehnelogija opisana je u standardu:

ISO/IEC TR 14471:1999 Information technology - Software engineering - Guidelines for the adoption of CASE tools.

Iz familije ISO standarda vezanih za evaluaciju standarda dostupno je 4 standarda. Ova familija standarda nosi zajednički naziv:

ISO/IEC 14598 Software engineering - Product evaluation.

Od 6 najavljenih standarda ove grupe, ISO je stavio na tržište delove vezane za planiranje i upravljanje, zatim procese za razvojni tim i one koji ih koriste, kao i dokumentacije modula procene.

Upotreba prototajpinga standardizovana je pomoću:

ISO/IEC TR 14759: 1999 Software engineering - Mock up and prototype - A categorization of software mock up and prototype models and their use.

Proces održavanja softvera, zasnovan na navedenom standardu životnog ciklusa softvera, opisan je u standardu:

ISO/IEC 14764: 2006 Software Engineering - Software Life Cycle Processes -- Maintenance.

Standard koji je takođe zasnovan na standardu životnog ciklusa, a glavna namena mu je da pomogne korisnicima u razumevanju faza životnog ciklusa je:

ISO/IEC 15289: 2006 Systems and software engineering - Content of systems and software life cycle process information products (Documentation).

Standard koji prati razvoj softvera u okruženju Web sajta je:

ISO/IEC 23026: 2006 Software Engineering - Recommended Practice for the Internet - Web Site Engineering, Web Site Management, and Web Site Life Cycle.

Među preostalim standardima softverskog inženjerstva, u značajnije spadaju:

ISO/IEC 24570: 2005 Software engineering - NESMA functional size measurement method version 2.1 -- Definitions and counting guidelines for the application of Function Point Analysis,

ISO/IEC 25000: 2005 Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE,

ISO/IEC 25051: 2006 Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) -- Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing,

ISO/IEC 25062: 2006 Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) -- Common Industry Format (CIF) for usability test reports i

ISO/IEC 90003: 2004 Software engineering - Guidelines for the application of ISO 9001:2000 to computer software.

Pored ISO standarda, značajna je i grupa standarda koju je u ovoj oblasti kreirala profesionalna inženjerska organizacija IEEE. Ovu grupu sačinjavaju:

IEEE 1012-1986 Software Verification & Validiation Plans,

IEEE 828-1990 Softvare Configuration Management Plan,

IEEE 1058.1-1987 Software Project Management Plan,

IEEE 830-1993 Software Requirements Specification,

IEEE 982.2-1988 IEEE Guide for The Use of IEEE Standard Dictionary of Measures to Produce Relible Softvare,

IEEE 730-1989 Software Quality Assurance Plans,

IEEE P1233 Guide to Developing System Requirements Specifications,

IEEE 1016.1-1993 Software Design Document,

IEEE 1044.1 Severity Clasification,

IEEE 1008-1987 Road Map for Unit Testing,

ANSI/IEEE 829-1991 Software Test Documentation i

IEEE 1219-1992 Software Maintanance.

6.8. Međunarodni standardi

Pored navedenih, važniji međunarodni standardi koji kod nas za sada nisu usvojeni, a odnose se na primenu informacionih tehnologija, mogu se podeliti u više grupa. Najstariji proces standardizacije u oblasti informacionih tehnologija vezan je za izradu standarda programskih jezika. Tako danas razlikujemo standarde svih raprostranjenijih programskih jezika. Među njima su:

ISO 1539 Information Technology - Programming Languages - FORTRAN, 1991.

ISO 1989 Information Technology - Programming Languages - COBOL, 1985.

ISO 6160 Information Technology - Programming Languages - PL/I, 1979.

ISO 6373 Information Technology - Programming Languages - Minimal Basic, 1984.

ISO/IEC 6522 Information Technology - Programming Languages - PL/I general Purpose Subset, 1992.

ISO/IEC 7185 Information Technology - Programming Languages - Pascal, 1990.

ISO 7846 Information Technology - Programming Languages - Industrial Real-time Fortran, 1985.

ISO 8485 Information Technology - Programming Languages - APL, 1989.

ISO 8652 Information Technology - Programming Languages - Ada, 1987.

ISO/IEC 9899 Information Technology - Programming Languages - C, 1990.

ISO/IEC 10206 Information Technology - Programming Languages - Extended Pascal, 1991.

ISO/IEC 10279 Information Technology - Programming Languages - Full Basic, 1991.

Razvoj programskih jezika prati sledeća grupa standarda:

ISO 9547 Information Technology - Programming Languages Processors - Test Methods: Guidelines for their Davelopment and Acceptibility

ISO 9945 Information Technology - Portable Operating System Interface (POSIX)

ISO/IEC 10034 Information Technology - Programming Languages - Guidelines for the Preparation of Conformity Clauses in Programming Language Standards

ISO/IEC 10182 Information Technology - Programming Languages - Their Environments and System Software Interfaces Guidelines for Language Bindings

ISO 10967-1 Information Technology - Programming Languages - Language Independent Arithmetic: Integer and Floating Point Arithmetic

Standardi baza podataka najčešće se vezuju za strukturirani upitni jezik SQL i elemente podataka. U ovoj grupi se nalaze:

ISO/IEC 9075 Information Technology - Database Languages - Structured Queery Language (SQL)

ISO/IEC 9579-1 Information Technology - Open systems Interconnection Remote Database Access - Generic Model

ISO/IEC 9579-2 Information Technology - Open systems Interconnection Remote Database Access - SQL Specialisation

ISO/IEC 9579-3 Information Technology - Open systems Interconnection Remote Database Access - SQL specialisation PICS

ISO/IEC 10032 Information Technology - Reference Model of Data Management

ISO/IEC 10728 Information Technology - Information Resource Dictionary System (IRDS) Services Interface

ISO/IEC 11179 Information Technology - Specification and Standardization of Data Elements

ISO/IEC 14957 Information Technology - Notation of Format for Data Element Values

Standarde otvorenih sistema predložio je odbor POSIX (Portable Operating System Interfacebased on UNIX) koji je formirao Institute of Electrical and Electronics Engineers (poznatiji kao IEEE). Ovim standardima definišu se uslovi portabilnog UNIX operativnog sistema za različita računarska okruženja. Grupu standarda otvorenih sistema naknadno usvajaju i ISO i ANSI. Ovu grupu čine sledeći standardi:

IEEE 1003.0 The POSIX Guide Project

IEEE 1003.1 Portable Operating System Interfaces for Computing Environments - System Services Interface

IEEE 1003.2 Portable Operating System Interfaces for Computing Environments - Shell, tools and user utilities

IEEE 1003.3 Portable Operating System Interfaces for Computing Environments - Testing and Verification

IEEE 1003.4 Portable Operating System Interfaces for Computing Environments - Real-time extensions

IEEE 1003.5 Portable Operating System Interfaces for Computing Environments - Ada language binding

IEEE 1003.6 Portable Operating System Interfaces for Computing Environments - Security extensions

IEEE 1003.7 Portable Operating System Interfaces for Computing Environments - System administration

IEEE 1003.8 Portable Operating System Interfaces for Computing Environments - Networking administration

IEEE 1003.9 Portable Operating System Interfaces for Computing Environments - FORTRAN language binding

IEEE 1003.10 Portable Operating System Interfaces for Computing Environments - Supercomputing

IEEE 1003.11 Portable Operating System Interfaces for Computing Environments - Transaction processing

Grupu standarda vezanih za zaštitu čine dva standarda, od kojih je jedan formirala međunarodna International Standard Organisation (ISO), dok je drugi izdat od strane Ministarstva odbrane USA (DoD). To su:

ISO 7498-2 Information Processing Systems - Open Systems Interconnection - Basic Reference Model: Security Architecture

DoD 5200.28 Trusted Computer System Evaluation Criteria (TCSEC)

Pored navedenih, veoma su značajne još dve grupe standarda koje, zbog velikog broja formiranih standarda, neće biti detaljno prikazane. Prva grupa opisuje računarsku grafiku i te standarde su formirali ISO i konzorcijumi proizvođača računara X/Open i Open Software Foundation (OSF). Izuzetno veliku i značajnu drugu grupu čine komunikacioni standardi koje pored ISO, donose i American National Standard Institute (ANSI) i IEEE.