UNIVERZITET METROPOLITAN BEOGRAD

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Razvoj profesionalnog softvera

Lekcija 01

PRIRUČNIK ZA STUDENTE

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Lekcija 01

RAZVOJ PROFESIONALNOG SOFTVERA

- Razvoj profesionalnog softvera
- Poglavlje 1: Šta je softver?
- Poglavlje 2: Softversko inženjerstvo
- Poglavlje 3: Raznovrsnost softverskog inženjerstva
- Poglavlje 4: Softversko inženjerstvo primenom Interneta
- Poglavlje 5: Informacija: Metodologija realizacije nastave
- Poglavlje 6: Vežba - Studija slučaja: Digitalno učenje
- Poglavlje 7: Vežba - Zadaci za individualno rešavanje
- Zaključak

UVOD

Cilj predmeta i sadržaj prve lekcije

Predmet SE101 Razvoj softvera i inženjera softvera je nov predmet specijalno projektovan za brucoše programa OAS Softverskog inženejerstva.
Cilj predmeta je da

- na jednostavan način objasni studentima šta je softver i softversko inženjerstvo;
- upozna studente sa osnovnim metodima i procesima razvoja softvera;
- upozna studente sa programom i planom nastave na studijskom programu osnovnih akademskih studija (OAS) Softversko inženjerstvo.

Zašto je uveden ovaj predmet i zašto se predaje već u prvom semestru? Videćete, u toku studija u prve dve godine dominiraju predmeti programiranja, a u trećoj i četvrtoj godini se izučavaju predmeti koji su ključni za formiranje diplomiranog inženjera softvera. Suočeni smo sa pojavom da studenti, kada savladaju osnovne veštine programiranja, da u jednom delu, počinju da se zapošljavaju , pa i da zbog toga i napuštaju studije, jer im se nude primamljiva mesta programera, ili junior inženjera softvera. Oni dobijaju pogrešnu ocenu da da sa znanjem programiranja, dovoljno znaju o softverskom inženjerstvu i da mogu da odmah unovče to svoje znanje u IT industriji. Neki i rade i nastavljaju studije (preko Interneta), a neki prekidaju studije. Ovim predmetom želimo da im pomognemo da shvate da prave veliku grešku u razvoju svoje profesionalne karijere, jer to što su naučili programiranje, to nije softversko inženjerstvo. Ako neko nauči da piše, ne znači da može da piše romane. Ako neko zna programiranje, ne znači da može da razvija sofver. Razvoj profesionalnog softvera nije isto što i program napisan u nekom programskom jeziku. Za razvoj profesionalnog softvera je potrebno mnogo više znanja nego samo znaje programiranja.
Nadamo se da ćemo uspeti da vam ove poruke prenesemo kako bi vas uputili na pravi način razvoja vaše profesionalne karijere, karijere diplomiranog inženjera softvera. Na predmetu nećemo napisati nijednu programsku instrukciju u nekom od programskih jezika. Predmet je koncipiran da za vas bude lak za praćenje, pa i za polaganje, naravno, jer više vremena ćemo posvetiti našim diskusijama, nego izlaganju celokupnog gradiva svake lekcije, jer očekujemo da proučavate ceo nastavni materijal. Lak predmet ne mora da ynači i beynačajni predmet. Ovaj predmet, ako ga dobro razumete, može biti i jedan od najvažniji predmeta vaših studija, jer treba da vam ukaže na način kako ćete postatii dobar i uspešan diplomirani inženjer softvera. Nadamo se da će to, ako već nije, postati i vaš cilj studiranja. U ovoj lekciji, koja je uvodna za predmet, pokušačemo da objasnimo šta je softver, koja su svojstva kvalitetng softvera, šta je softversko inženjerstvo, šta znači razvoj profesionalnog softvera i kako će se nastava na ovom predmetu odvijati.

Poglavlje 1

Šta je softver?

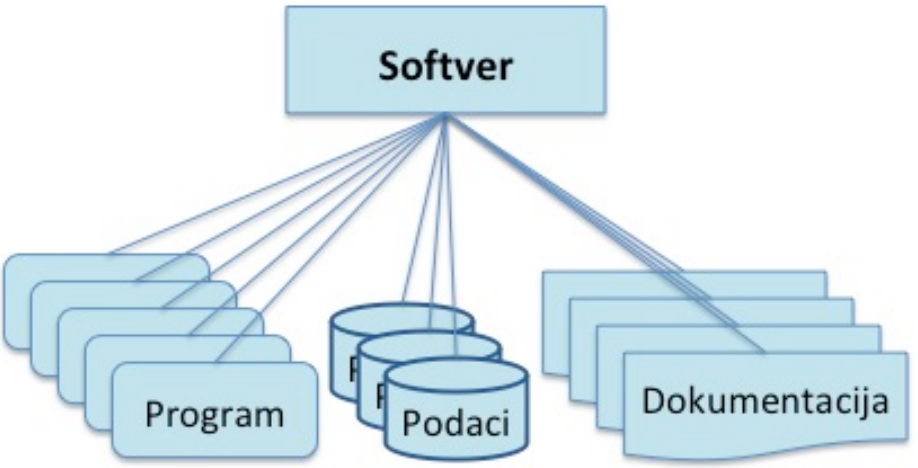
SOFTVER KAO SKUP PROGRAMA, PODATAKA O DOKUMENTACIJE

Program je jednostavan niz operacija koje omogućavaju automatsko izvršavanje digitalnih zapisa na računarskom uređaju, a softver je skup programa, podataka i dokumentacije

Softver je *skup programa/podataka koji su napravljeni da upute računar za njegov rad*. Softver je digitalni deo koji radi na hardveru, kojim upravlja. Može se reči i da je softver je skup uputstava namenjenih da ih mašina tumači za obavljanje različitih zadataka na računarskom uređaju Softver daje smisao radu računara. Tačnije, nijedan zadatak nije moguć na računaru bez softvera. Možemo reći da je hardver telo, a softver mozak ili duša
U čemu je onda razlika između softvera i programa napisan u nekom od programskih jezika, ili kako to najčešće zvome – programski kod? Program je jednostavan niz operacija koje omogućavaju automatsko izvršavanje digitalnih zapisa na računarskom uređaju, a softver je skup računarskih programa (slika 1). Pored računarski programa, softver čini i dokumantacija koja treba da olakša njegovo korišćnje, ali i dalji razvoj softvera. U tom smislu postoji:

1. korisnička dokumentacija, koju koriste korisnici softvera, i
2. razvojna dokumentacija, koju koriste inženjeri softvera koji razvoju i održavaju softver.

Razvojna dokumentacija objašnjava kako se koristi sistem, daje s einformacija o konfiguraciji softvera, tj. o njegovoj strukturi, koju čine različiti softverski elementi, kao što su podsistem, softverske komponente i softverske jedicice.
Stepen korišćenja dokumentacije o softveru može da se bitno da razlikuje u zavisnosti od primenjene metodologije njegovog razvoja, o čemu ćemo kasnije govoriti.



Slika 1.1 Softver kao skup programa, podataka i dokumentacije [autor]

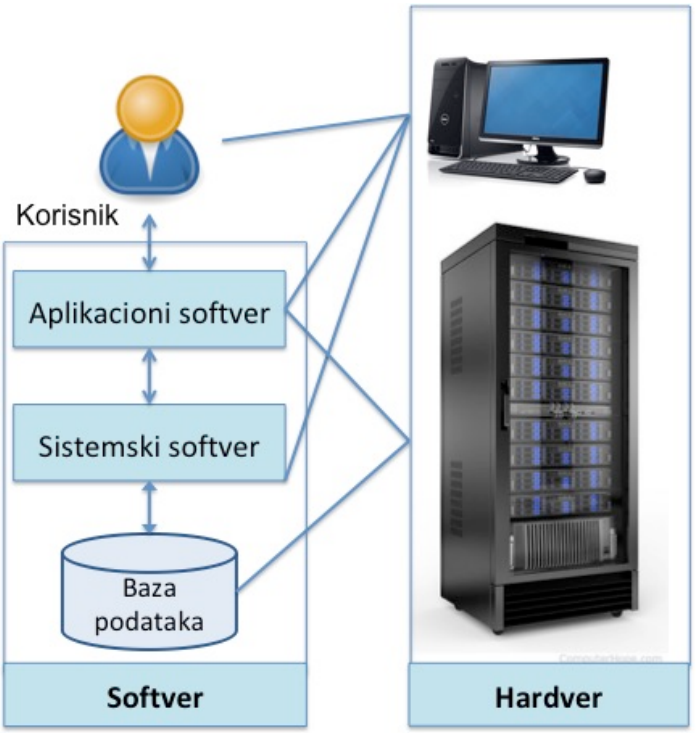
SOFTVERSKI SISTEM

Softverski sistem čini skup povezanih softvera, grupisanih u aplikacioni i sistemski softver.

Softver može da sadrži ili da bude povezan sa drugim softverima, tj. mogu biti složeni te oni postaju softverski sistemi. Softverski sistem čini skup softvera koji su povezani hijerahijskim vezama (jedan sadrži drugi) i funkcionalno grupisani i po toj osnovi povezani. Na slici 2 je prikazana struktura jednog softverskog sistema koji se instališe u okvru hardvera, koji ima klijentski i serverski deo. Softverski sistem čini:

1. **Aplikacioni softver** koga čine programi koje opslužuju korisnika, obično se naziva aplikacijama. Aplikacije mogu da rade sa zvukom, brojevima, tekstovima, kao i kombinacijom svih ovih elemenata. Primeri uključuju igre, sisteme baza podataka, obrazovni ili kancelarijski softver npr. Microsoft Word, Excel, Power Point, Vetransfer za prenos datoteka, itd.)
2. **Sistemski softver** je osnova za aplikativni softver (platforma za drugi softver) računarski programski jezik se obično koristi za pisanje sistemskog softvera. Primeri - uključuju servere, uslužne programe itd. **Operativni sistem** je sistemski softver koji omogućava izvšavanje na računaru, programskih instrukcija koje su napisane u programima koji čine aplikacioni softver. Funkcionalnost aplikacionog softvera zavisi od operativnog sistema. Softver ima mnogo funkcija kao što su GUI, procesi, ulazni/ izlazni podaci itd. Sistemski softver može koristiti i različite podatke neophodne za specifikaciju i podešvanje softverskog sistema sistem, kao i da upravlja sistemima baza podataka i datoteka sa podacima koje koriste aplikacije.

U okviru sistemskog softvera se nalaze i tzv. kompajleri, tj. prevodioci programa napisanog u izvornom programskom jeziku (npr u Python-u, Javi, CC++ ili u C#) u tzv. mašinski jezik koji <razumeju> komponenata hardverskog sistema.



Slika 1.2 Osnovna struktura jednog softverskog sistema [autor]

SOFTVERSKI PROIZVOD

Generički proizvod - razvijen za tržište, i softver razvijen po narudžbini kupca.

Softverski proizvod je apstraktian i nedodirljiv. Zbog svog nefizičkog karaktera,oni nisu ograničeni zakonima fizike, tako da mogu brzo da postanu i vrlo složeni, teško razumljivi i skupi za održavanje, tj. za menjanje. Zato, umesto o softveru, govorimo više više o softverskom sistemu, koji čini složeni softverrski proizvod koji najčešće ima više podsistema, modula i komponenata.

Postoje dve vrste softverskih proizvoda:

- **Generički proizvodi:** To su softverski proizvodi koji su razvijeni radi prodaje na tržištu. Mogu ih koristiti različiti korisnici koji nisu poznati u vreme razvoja. Organizacija koja razvija softver definiše programsku specifikaciju. Primeri: obrađivači tekstova, sistemi za upravljanje bazama podataka, CAD sistemi.
- **Proizvodi razvijeni po narudžbini:** Firma koja razvija softver, to radi po zahtevu kupca, koji definiše specifikaciju za softver. Primeri: kontrolni sistemi elektronskih uređaja, sistemi za upravljanje određenim poslovnim procesima, sistemi za upravljanje saobraćajem.

Kritična razlika između ovih tipova softvera je u tome što u generičkim proizvodima organizacija koja razvija softver kontroliše specifikaciju softvera. To znači da ako naiđu na razvojne probleme, mogu da preispitaju šta treba da se razvije

To znači da ako naiđu na razvojne probleme, mogu da preispitaju šta treba da se razvija. *Za proizvode razvijene po narudžbini, specifikaciju razvija i kontroliše organizacija koja kupuje softver*. Programeri softvera moraju raditi prema toj specifikaciji.
Međutim, razlika između ovih sistemskih tipova proizvoda postaje sve nejasnija. *Sve više i više sistema se sada gradi sa generičkim proizvodom kao osnovom, koji se zatim prilagođava zahtevima kupaca*. Sistemi za planiranje resursa preduzeća (ERP sistemi), kao što su sistemi iz SAP-a i Oracle-a, najbolji su primeri ovog pristupa. Ovde je veliki i složen softverski sistem prilagođen za korisničku kompaniju tako što uključuje informacije o poslovnim pravilima i procesima, potrebnim izveštajima i tako dalje.
I u okviru ove dve dve osnovne vrste softverskih proizvoda, Postoji puno različitih vrsta (tipova) softverskih sistema, počev od vrlo jednostavnih, tzv. ugrađenih sistema (npr. sistem upravljanja pojedinim podsistemima u automobilima) do vrlo kompleksnih globalnih informacionih sistema (npr. sistemi za rezervaciju hotelskog smeštaja, kupovinu avionskih karata i dr.).
Specifičan skup atributa koje možete očekivati od softverskog sistema očigledno zavisi od njegove primene. Na primer, sistem upravljanja avionom mora biti bezbedan, interaktivna igra mora da reaguje brzo na reakciju korisnika, telefonski sistem za komunikaciju mora biti pouzdan, itd. Oni se mogu generalizovati u skup atributa prikazanih tabeli T1, u kojoj su prikazane suštinska svojstva karakteristike profesionalnog softverskog sistema

UZROCI GREŠAKA U SOFTVERU

Greške u softveru su posledica povećanih očekivanja korisnika, s jedne strane, i niskog stepena primene metoda i tehnika softverskog inženjerstva

Još uvek ima mnogo izveštaja o softverskim projektima koji su krenuli naopako i o „greškama softvera“. Softversko inženjersstvo se kritikuje kao neadekvatno za savremeni razvoj softvera. Međutim, mnogi od ovih takozvanih softverskih kvarova su posledica dva faktora:

1. **Povećani zahtevi.** Razvoj softvera sve je više izložen novim zahtevima koji dovode do čestih promena i dopuna koje čine da softver bude sve složeniji. Ti zahtevi traže da se u što kraćem vremenskom roku realizuju. Postojeće metode softverskog inženjerstva ne mogu da se nose sa ovakvim zahtevima, jer ne mogu da ih zadovolje te se moraju razviti nove tehnike softverskog inženjerstva u cilju rešavanja tog problema.
2. **Niska primena metoda softverskog inženjerstva**: Moguć je razvoj računarskih programa i bez primene metoda i tehnika softverskog inženjerstva. Često, organizacije koriste sisteme koji rade nepouzdano, te razvijaju programe isključivo za svoje potrebe. Sistemi su nepouzdani najčešće iz neznanja ili ne koriste metode i tehnike softverskog inženjerstva, već njihovi programeri počnu izradu programa bez odgovarajućih pripremnih aktivnosti, a i bez kasnijih odgovarajućih testiranja. Kao rezultat, često dobijaju softverske sisteme koji rade nepouzdano, te moraju stalno da nalaze i otklanjaju greške u softveru, a to utiče na povećanje troškova razvoja i održavanja softvera. Međutim, te organizacije mnogo veće troškove imaju u oblasti primene softvera, gde kvarovi u softverskim sistemima dovode do prekida ili ometanja u njihovom svakodnevnom poslovanju (na primer, zaustavljanje

proizvodnje ili montaže proizvoda). Troškovi posledica do kojih dovode zastoji u softverskim sistemima su mnogo veći, nego troškovi otklanjanja tih grešaka u samom softveru.

Očigledno, nedostaje veći stepen obrazovanja iz softverskog inženjerstva i obuke ljudi koji razvijaju softverske sisteme da bi se izbegao problem razvoja skupih (zbog stalnih prepravki i dorada), a nepouzdanih softverskih sistema (jer često ulaze u neočekivane zastoje, zbog grešaka u softveru, tzv. bagove).
Pravilnom upotrebom tehnika softverskog inženjerstva, ovi problemi i greške mogu biti smanjeni ili eliminisani, pa je zato od velikog značaja razumeti šta predstavlja softversko inženjerstvo i na koji način se adekvatno primenjuje

SVOJSTVA KVALITETNOG SOFTVERA

Specifičan skup atributa kvaliteta softvera zavisi od njegove primene.

Kada govorimo o kvalitetu profesionalnog softvera, moramo uzeti u obzir da softver koriste i menjaju ljudi osim njegovih programera. Kvalitet se stoga ne odnosi samo na ono što softver radi. Umesto toga, mora da uključi ponašanje softvera dok se izvršava i strukturu i organizaciju sistemskih programa i povezane dokumentacije. Ovo se ogleda u kvalitetu softvera ili u njegovim nefunkcionalnim atributima. Primeri ovih atributa su vreme odgovora softvera na upit korisnika i razumljivost programskog koda.
Specifičan skup atributa koje možete očekivati od softverskog sistema očigledno zavisi od njegove primene. Na primer, sistem upravljanja avionom mora biti bezbedan, interaktivna igra mora da reaguje brzo na reakciju korisnika, telefonski sistem za komunikaciju mora biti pouzdan, itd. Oni se mogu generalizovati u skup atributa prikazanih tabeli T1 na slici 1, u kojoj su prikazane suštinska svojstva karakteristike profesionalnog softverskog sistema.

Tabela T1: Bitna svojstva kvalitetnog softvera

Svojstva proizvoda	Opis
Prihvatljivost	Softver mora biti prihvatljiv za kategorije korisnika za koje je projektovan. Ovo znači da mora da bude razumljiv, koristan i kompatibilan sa sistemima koje korisnici upotrebljavaju.
Pouzdanost i sigurnost	Pouzdanost softvera obuhvata niz karakteristika, kao što su: pouzdanost, sigurnost i bezbednost. Pouzdanost softvera ne bi trebalo da prouzrokuje nikakvu fizičku ili ekonomsku štetu u slučaju pada sistema. Neovlašćena lica ne bi mogla da pristupe sistemu i da mu naprave štetu.
Efikasnost	Softver ne bi trebalo da beskorisno troši resurse sistema, kao što su memorija i ciklusi procesora. Efikasnost obuhvata: odziv sistema, vreme obrade, upotrebu memorije i dr.
Sposobnost održavanja	Softver treba biti tako napisan da može da se lako menja kako bi se zadovoljili zahtevi korisnika za njihovu promenu. Ovo je kritičan atribut , jer je promena softvera neminovni zahtev u poslovnim okruženjima koja se brzo menjaju.

Slika 1.3 Bitna svojstva kvalitetnog softvera [1, Fig. 1.2]

WHY SOFTWARE ENGINEERING? (VIDEO- 2,42 MIN.)

Ian Sommerville, autor našeg udžbenika - Zašto je potrebno softversko inženjerstvo?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

WHAT IS SOFTWARE? (VIDEO-2,02 MIN.)

Jednostavno objašnjenje šta je softver

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

WHAT IS SOTWARE? (VIDEO-3,01 MIN.)

Prikaz različitih softvera po oblasti primene

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Softversko inženjerstvo

SOFTVERSKI INŽENJERSTVO JE INŽENJERSKA DISCIPLINA

Softversko inženjerstvo je grana inženjerstva koja se bavi razvojem softverskih proizvoda

Softversko inženjerino je inženjerska disciplina koja se bavi svim aspektima proizvodnje softvera od ranih faza specifikacije sistema do održavanja sistema nakon što je ušao u upotrebu. U ovoj definiciji postoje dve ključne fraze:

1. **Inženjerska disciplina:** Inženjeri čine da stvari funkcionišu**.** Oni primenjuju teorije, metode i alate tamo gde su prikladni. Međutim, oni ih koriste selektivno i uvek pokušavaju da pronađu rešenja za probleme čak i kada ne postoje primenljive teorije i metode. Inženjeri takođe shvataju da moraju da rade u okviru organizacionih i finansijskih ograničenja i da moraju tražiti rešenja unutar ovih ograničenja.
2. **Svi aspekti proizvodnje softvera:** Softversko inženjerstvo se ne bavi samo tehničkim procesima razvoja softvera. Takođe uključuje aktivnosti kao što su upravljanje softverskim projektima i razvoj alata, metoda i teorija za podršku razvoju softvera.

Inženjerstvo je postizanje rezultata potrebnog kvaliteta u okviru rasporeda i budžeta. Ovo često uključuje pravljenje kompromisa - inženjeri ne mogu biti perfekcionisti. Ljudi koji pišu programe za sebe, međutim, mogu potrošiti koliko god žele vremena na razvoj programa.
Softversko inženjerstvo je grana inženjerstva koja se bavi razvojem softverskih proizvoda. Deluje u okviru niza principa, najboljih praksi i metoda koje su pažljivo usavršavane tokom godina, menjajući se kako se softver i tehnologija menjaju.Softversko inženjerstvo dovodi do proizvoda koji je pouzdan, efikasan i efikasan u onome što radi.
IEEE definicija: Softversko inženjerstvo je primena sistematskog, disciplinovanog, kvantitativnog pristupa razvoju, radu i održavanju softvera; odnosno primena inženjerstva na softver.
Generalno, *softverski inženjeri usvajaju sistematski i organizovan pristup svom radu*, jer je to često najefikasniji način za proizvodnju visokokvalitetnog softvera. Međutim, inženjerstva se svodi na odabir najprikladnije metode za niz okolnosti, tako da kreativniji, manje formalni pristup razvoju može biti
pravi za neke vrste softvera. Fleksibilniji softverski proces koji prilagođava brze promene posebno je prikladan za razvoj interaktivnih sistema zasnovanih na vebu i mobilnih aplikacija, koje zahtevaju mešavinu softvera i grafičkih veštine dizajna

SOFTVERSKI PROCES

Softverski proces je niz aktivnosti koji dovodi do proizvodnje softverskog proizvoda.

Softversko inženjerstvo je važno iz dva razloga:

1. Sve više se pojedinci i društvo oslanjaju na napredne softverske sisteme. Moramo biti u mogućnosti da ekonomično i brzo proizvodimo pouzdane i pouzdane sisteme.
2. Obično je jeftinije, dugoročno gledano, koristiti metode i tehnike softverskog inženjerstva za profesionalne softverske sisteme, a ne samo pisati programe kao lični projekat programiranja. Neupotrebljavanje metode softverskog inženjerstva dovodi do većih troškova za testiranje, osiguranje kvaliteta i dugoročno održavanje.

Sistematski pristup koji se koristi u softverskom inženjerstvu ponekad se naziva softverskim procesom. Softverski proces je niz aktivnosti koji dovodi do proizvodnje softverskog proizvoda. Četiri osnovne aktivnosti su zajedničke za sve softverske procese.

1. **Specifikacija softvera**, gde kupci i inženjeri definišu softver koji će biti proizveden i ograničenja njegovog rada.
2. **Razvoj softvera**, gde je softver dizajniran i programiran.
3. **Validacija softvera**, gde se softver proverava da bi se uverilo da je ono što klijent zahteva.
4. **Evolucija softvera**, gde se softver modifikuje tako da odražava promenljive zahteve kupaca i tržišta.

Različiti tipovi sistema zahtevaju različite razvojne procese. . Na primer, softver u realnom vremenu u avionu mora biti u potpunosti specificiran pre nego što razvoj počne. U sistemima e-trgovine, specifikacija i program se obično razvijaju zajedno. Shodno tome, ove generičke aktivnosti mogubiti organizovane na različite načine i opisane na različitim nivoima detalja, u zavisnosti od vrste softvera koji se razvija.

Softversko inženjerstvo je povezano i sa računarskom naukom (compiuter science) i sa sistemskim inženjeringom (system engineering).

1. Računarska nauka se bavi teorijama i metodama koje su u osnovi računara i softverskih sistema, dok se softversko inženjerstvo bavi praktičnim problemima proizvodnje softvera. Neka znanja iz računarske nauke su neophodna za softverske inženjere. Međutim, teorija reačunarske nauke je često najprimenljivija na relativno male programe. Elegantne teorije ralunarske nauke retko su relevantne za velike, složene probleme koji zahtevaju softversko rešenje.
2. Sistemsko inženjerstvo se bavi svim aspektima razvoja i evolucije složenih sistema u kojima softver igra glavnu ulogu. Sistemsko inženjerstvo se stoga bavi razvojem hardvera, projektovanjem proces i implementacijom sistema, kao i softverskim inženjerstvom.. Sistemski inženjeri su uključeni u specifikaciju sistema, definisanje njegove ukupne arhitekture, a zatim integraciju različitih delova da bi se kreirao gotov sistem.

ČETIRI POVEZANA PROBLEMA KOJI UTIČU NA TIPOVE SOFTVERA

Faktori koji utiču na tip softvera koji se razvija

Ne postoje univerzalne metode ili tehnike softverskog inženjerstva koje se mogu koristiti. Međutim, postoje četiri povezana problema koja utiču na mnoge različite tipove softvera:

1. **Heterogenost:** Od sistema se sve više zahteva da rade kao distribuirani sistemi kroz mreže koje uključuju različite tipove računara i mobilnih uređaja. Osim što se pokreće na računarima opšte namene, softver će možda morati da se izvršava i na mobilnim telefonima i tabletima. Često morate da integrišete novi softver sa starijim zastarelim sistemima napisanim na različitim programskim jezicima. Ovde je izazov razviti tehnike za pravljenje pouzdanog softvera koja je dovoljno fleksibilna da se nosi sa ovom heterogenošću.
2. **Poslovna i društvena promena:** Preduzeća i društvo se menjaju neverovatno brzo kako se ekonomije u razvoju razvijaju i nove tehnologije postaju dostupne. Oni moraju biti u mogućnosti da promene svoj postojeći softver i da brzo razviju novi softver. Mnoge tradicionalne tehnike softverskog inženjeringa oduzimaju mnogo vremena, a isporuka novih sistema često traje duže nego što je planirano. Oni moraju da se razvijaju tako da se smanji vreme potrebno softveru da isporuči vrednost svojim klijentima.
3. **Bezbednost i poverenje:** Pošto je softver isprepleten sa svim aspektima naših života, od suštinske je važnosti da možemo verovati tom softveru. Ovo posebno važi za udaljene softverske sisteme kojima se pristupa preko veb stranice ili interfejsa veb usluge. Moramo da budemo sigurni da zlonamerni korisnici ne mogu uspešno da napadnu naš softver i da se održava bezbednost informacija.
4. **Veličina softvera:** mora biti razvijena u veoma širokom spektru obima, od veoma malih ugrađenih sistema u prenosive ili nosive uređaje do sistema zasnovanih na oblaku na Internetu koji služe globalnoj zajednici.

Da bismo odgovorili na ove izazove, biće nam potrebni novi alati i tehnike, kao i inovativni načini kombinovanja i korišćenja postojećih metoda softverskog inženjeringa.

SOFTVERSKO INŽENJERSTVO STVARA SOFTVERSKI PROIZVOD

Softversko inženjerstvo je sistematski pristup proizvodnji softvera koji uzima u obzir praktične troškove, raspored i pitanja pouzdanosti, kao i potrebe kupaca i proizvođača softvera.

Programeri dobijaju zadatak da naprave neku vrstu softvera. Tim za razvoj softvera deli projekat na zahteve i korake. Ponekad se ovaj posao prepušta nezavisnim izvođačima i honorarno angažovanim samostalnim programerima, tzv. frilanserima (freelancers). Kada je

to slučaj, alati za softversko inženjerstvo pomažu da se osigura da sav bude obavljen posao u saradnji primenom najbolje prakse.

Kako programeri znaju šta da stave u svoj softver?

- Oni ga dele na **specifične potrebe** nakon obavljanja intervjua, prikupljanja informacija, pregleda postojećeg portfelja aplikacija i razgovora sa IT liderima.
- Zatim će **napraviti mapu puta kako da naprave softver**. Ovo je jedan od najvažnijih delova jer je veliki deo „posla“ završen tokom ove faze – što takođe znači da se problemi obično javljaju i ovde.

Prava početna tačka je kada programeri počnu da pišu kod za softver. Ovo je najduži deo procesa u mnogim slučajevima jer programski kod treba da bude u skladu sa postojećim sistemima i jezikom koji se u njima koristi. Nažalost, ovi problemi se često ostaju neopažni na početku i tokom realizacija projekta, već se jave pri njegovom kraju. Tada je neophodno da se vrate unazad da bi dovršili ponovnu obradu. Programski kod treba da se testira onako kako je napisano, a kada je softver završen, treba ga testirati u svim delovima životnog ciklusa. Sa alatima za softversko inženjerstvo, moći ćete kontinuirano da testirate i nadgledate proces razvoja softvera.
Softversko inženjerstvo je sistematski pristup proizvodnji softvera koji uzima u obzir praktične troškove, raspored i pitanja pouzdanosti, kao i potrebe kupaca i proizvođača softvera. Specifične metode, alati i tehnike koje se koriste zavise od organizacije koja razvija softver, vrste softvera i ljudi uključenih u proces razvoja. Ne postoje univerzalne metode softverskog inženjerstva koje su pogodne za sve sisteme i sve kompanije. Možda je najznačajniji faktor u određivanju koje su metode i tehnike softverskog inženjerstva najvažnije jeste vrsta aplikacije koja se razvija

PROFESIONALNI RAZVOJ SOFTVERA

Pravi rad softverskog inženjerstva počinje pre nego što je proizvod uopšte projektovan, nastavlja se dugo nakon što je softer razvijen i lansiran na tržištu.

Softver koji je razvijen metodima softverskog inženjerstva treba da radi na stvarnim mašinama u stvarnim situacijama. Softversko inženjerstvo počinje kada postoji potražnja za određenim rezultatom ili izlazom za kompaniju, radi primene u nekoj oblasti rada. Zato se često takva vresta softvera i naziva aplikacionim softverom, ili kraće – aplikacija. Generalno, *softverski inženjeri usvajaju sistematski i organizovan pristup svom radu, jer je to često najefikasniji način za proizvodnju visokokvalitetnog softvera.* Međutim, inženjerstvo se svodi na odabir najprikladnije metode za određenu situaciju. Fleksibilniji softverski proces koji prihvata brze promene je posebno prikladan za razvoj interaktivnih sistema zasnovanih na vebu i mobilnih aplikacija, koji zahtevaju mešavinu softvera i veština grafičkog dizajna.

Pravi rad softverskog inženjerstva počinje pre nego što je proizvod uopšte projektovan – a osnove softverskog inženjerinstva nalažu da se nastavi dugo nakon što je „rad“ završen. Sve počinje temeljnim i potpunim razumevanjem šta vaš softver treba da ima – ovo uključuje šta softver treba da radi, sistem u kome treba da radi i svu bezbednost koju podrazumeva. Bezbednost je jedna od osnova softverskog inženjerstva jer je tako neophodna za sve aspekte

razvoja. Bez alata koji bi vam pomogli da bolje razumete kako se gradi vaš kod i gde mogu da padnu bilo kakvi bezbednosni problemi, vaš tim se lako može izgubiti u fazi razvoja. Mi te aspekte softverskog inženjerstva nećemo raditi u okviru ovog premeta, jer se to izučava u posebnom predmetu studijskog programa.

Na osnovu projektnog rešenja dobijenog u okviru procesa softverskog inženjerstva, neophodno je kreiranje instrukcija za računar, tj. programirati ono što računar treba da radi. Veliki deo ovoga će se na nivou kodiranja odvijati od strane profesionalaca koji imaju sveobuhvatnu obuku, od strane programera. Ipak, važno je razumeti da softversko inženjerrstvo nije uvek linearan proces, što znači da zahteva temeljnu proveru nakon što je završen

Zato, zapamtite: Programiranje nije isto što i softversko inženjerstvo, jer je samo jedan njegov deo.

10 QUESTIONS TO INTRODUCE SOFTWARE ENGINEERING (VIDEO-6,41 MIN.)

Ian Sommerville, autor našeg udžbenika

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 3

Raznovrsnost softverskog inženjerstva

RAZNOVRSNOST SOFTVERSKOG INŽENJERSTVA I SOFTVERSKIH PROIZVODA

Vrste aplikacija određuju i metode softverskog inženjerstva koje se koriste.

Postoji mnogo različitih tipova aplikacija, uključujući:

1. **Samostalne aplikacije** Ovo su sistemi aplikacija koji se pokreću na ličnom računaru ili aplikacije koje se pokreću na mobilnom uređaju. Oni uključuju sve neophodne funkcionalnosti i možda neće morati da budu povezani na mrežu. Primeri takvih aplikacija su kancelarijske aplikacije na računaru, CAD programi, softver za obradu fotografija, aplikacije za putovanja, aplikacije za produktivnost i tako dalje.
2. **Interaktivne aplikacije** zasnovane na transakcijama Ovo su aplikacije koje se izvršavaju na udaljenom računaru i kojima korisnici pristupaju sa sopstvenih računara, telefona ili tableta. Očigledno, ovo uključuje veb aplikacije kao što su aplikacije za e-trgovinu gde komunicirate sa udaljenim sistemom da biste kupili robu i usluge.
3. **Ugrađeni kontrolni sistemi** Ovo su softverski kontrolni sistemi koji kontrolišu i upravljaju hardverskim uređajima. Brojčano, verovatno postoji više ugrađenih sistema nego bilo koji drugi tip sistema. Primeri ugrađenih sistema uključuju softver u mobilnom (mobilnom) telefonu, softver koji kontroliše kočenje protiv blokiranja u automobil i softver u mikrotalasnoj pećnici za kontrolu procesa kuvanja.
4. **Sistemi za grupnu obradu** Ovo su poslovni sistemi koji su dizajnirani za obradu podataka u velikim serijama. Oni obrađuju veliki broj pojedinačnih ulaza da bi stvorili odgovarajuće izlaze. Primeri paketnih sistema su periodični sistemi naplate, kao što su telefonski sistemi naplate i sistemi isplate plata
5. **Sistemi za zabavu** Ovo su sistemi za ličnu upotrebu koji su namenjeni zabavi korisnika. Većina ovih sistema su igre ove ili one vrste, koje mogu da rade na hardveru konzole posebne namene. Kvalitet ponuđene interakcije korisnika je najvažnija karakteristika sistema za zabavu.
6. **Sistemi za modeliranje i simulaciju** Ovo su sistemi koje su razvili naučnici i inženjeri da bi modelirali fizičke procese ili situacije, koji uključuju mnoge odvojene objekte u interakciji. Oni su često računarski intenzivni i zahtevaju paralelne sisteme visokih performansi za izvršenje.
7. **Sistemi za prikupljanje i analizu podataka.** Sistemi za prikupljanje podataka su sistemi koji prikupljaju podatke iz svog okruženja i šalju te podatke drugim

sistemima na obradu. Softver će možda morati da stupi u interakciju sa senzorima i često je instaliran u neprijateljskom okruženju kao što je unutar motora ili na udaljenoj lokaciji. Analiza „velikih podataka“ može uključivati sisteme zasnovane na oblaku koji sprovode statističku analizu i traže veze u prikupljenim podacima

8. **Sistemi sistema** To su sistemi koji se koriste u preduzećima i drugim velikim organizacijama, a koji se sastoje od niza drugih softverskih sistema. Neki od njih mogu biti generički softverski proizvodi, kao što je ERP sistem. Drugi sistemi u sklopu mogu biti posebno napisani za to okruženje

SOFTVERSKO INŽENJERSTVOAI VRSTE APLIKACIJE

Najznačajniji faktor u određivanju koje su metode i tehnike softverskog inženjerstva najvažnije jeste vrsta aplikacije koja se razvija.

Softversko inženjerstvo je sistematski pristup proizvodnji softvera koji uzima u obzir praktične troškove, raspored i pitanja pouzdanosti, kao i potrebe kupaca i proizvođača softvera. Specifične metode, alati i tehnike koje se koriste zavise od organizacije koja razvija softver, vrste softvera i ljudi uključenih u proces razvoja. Ne postoje univerzalne metode softverskog inženjerstva koje su pogodne za sve sisteme i sve kompanije. Umesto toga, raznovrstano skup metoda i alata softverskog inženjeringa evoluirao je u poslednjih 50 godina. . Možda je **najznačajniji faktor u određivanju koje su metode i tehnike softverskog inženjerstva najvažnije jeste vrsta aplikacije koja se razvija** Granice između ovih tipova sistema su zamagljene. Ako razvijate igru za telefon, morate da uzmete u obzir ista ograničenja (snaga, interakcija između hardvera) kao i programeri softvera telefona. Sistemi za grupnu obradu se često koriste u kombinaciji sa sistemima transakcija zasnovanim na vebu. Na primer, u kompaniji, zahtevi za putne troškove mogu se podneti putem veb aplikacije, ali se obrađuju u paketnoj aplikaciji za mesečno plaćanje.
Svaki tip sistema zahteva specijalizovane tehnike softverskog inženjerства jer softver ima različite karakteristike. Na primer, ugrađeni kontrolni sistem u automobilu je kritičan za bezbednost i ugrađuje se u ROM (memoriju koja služi samo samo za čitanje, tj. ya preuzimanje podataka) kada se instalira u vozilo. Zbog toga je veoma skupo ra menjati. Takvom sistemu je potrebna opsežna provera i validacija kako bi se minimizirale šanse da ćete morati da povučete automobile nakon prodaje da biste rešili probleme sa softverom. Interakcija korisnika je minimalna (ili možda nikakva), tako da ovde nema potrebe da se koristi razvojni proces koji se oslanja na prototip korisničkog interfejsa.

Za interaktivni sistem ili aplikaciju zasnovan na vebu, iterativni razvoj i isporuka su najbolji pristup, pri čemu se sistem sastoji od komponenti koje se mogu ponovo koristiti. Međutim, takav pristup može biti nepraktičan za sistem sistema, gde se detaljne specifikacije interakcija sistema moraju unapred specificirati kako bi svaki sistem mogao da se razvije zasebno.

OSNOVE SOFTVERSKOG INŽENJERSTVA

Postoje osnove softverskog inženjeringa koje se primenjuju na sve tipove softverskih sistema

Postoje osnove softverskog inženjerstva koje se primenjuju na sve tipove softverskih sistema

1. Trebalo bi da se razvijaju **korišćenjem upravljanog i razumljivog razvojnog procesa**. Organizacija koja razvija softver treba da planira proces razvoja i da ima jasne ideje šta će biti proizvedeno i kada će biti završeno. Naravno, konkretan proces koji treba da koristite zavisi od vrste softvera koji razvijate.
2. **Pouzdanost i performanse** su važne za sve tipove sistema. Softver treba da se ponaša kako se očekuje, bez grešaka i treba da bude dostupan za upotrebu kada je to potrebno. Trebalo bi da bude bezbedan u svom radu i, koliko je to moguće, treba da bude zaštićen od spoljašnjih napada. Sistem treba da radi efikasno i ne treba da troši resurse.
3. **Razumevanje i upravljanje softverskom specifikacijom i zahtevima** (šta softver treba da radi) su važni. Morate znati šta različiti kupci i korisnici sistema očekuju od njega, i morate upravljati njihovim očekivanjima kako bi koristan sistem mogao biti isporučen u okviru budžeta i planiranog rasporeda.
4. Trebalo bi da **efikasno koristite postojeće resurse**. To znači da, gde je to prikladno, treba da ponovo koristite softver koji je već razvijen, a ne da pišete novi softver.

Ovi fundamentalni pojmovi procesa, pouzdanosti, zahteva, upravljanja i ponovne upotrebe su važne teme ove knjige. Različite metode ih odražavaju na različite načine, ali one su u osnovi svakog profesionalnog razvoja softvera. *Ove osnove su nezavisne od programskog jezika koji se koristi za razvoj softvera.*

ZNAČAJ SOFTVERSKOG INŽENJERSTVA

Danas, bez softvera i softverskog inženjera, život bi stao. Zanimanje inženjera softvera ima svetlu budućnost.

Softversko inženjerstvo je od suštinskog značaja za funkcionisanje državnih institucija, društva i nacionalnih i međunarodnih preduzeća i institucija. Ne možemo voditi savremeni svet bez softvera Danas, većina električnih proizvoda uključuje računar i upravljački softver. Industrijska proizvodnja i distribucija je potpuno kompjuterizovana, kao i finansijski sistem. Zabava, uključujući muzičku industriju, kompjuterske igrice i film i televiziju, zahteva mnogo softvera. Više od 75% svetske populacije ima mobilni telefon koji kontroliše softve..
Softverski sistemi su apstraktni i nematerijalni. Oni nisu ograničeni svojstvima materijala, niti su vođeni fizičkim zakonima ili proizvodnim procesima. To pojednostavljuje softversko inženjerstvo, jer ne postoje prirodna ograničenja za potencijal softvera. Međutim, zbog nedostatka fizičkih ograničenja, **softverski sistemi mogu brzo postati izuzetno složeni, teško razumljivi i skupi za promenu**.
Postoji mnogo različitih tipova softverskog sistema, od jednostavnih ugrađenih sistema do složenih informacionih sistema širom sveta. Ne postoje univerzalne oznake, metode ili tehnike za softversko inženjerstvo jer različite vrste softvera zahtevaju različite pristupe. Razvoj organizacionog informacionog sistema je potpuno drugačiji od razvoja kontrolera za naučni instrument. Nijedan od ovih sistema nema mnogo zajedničkog sa grafički intenzivnom računarskom igrom. Svim ovim aplikacijama je potrebno softversoi inženjerstvo; ne trebaju svima iste metode i tehnike softverskog inženjerstva.

Softverski inženjeri, tj. inženjeri softvera, što bi trebalo da bude ispravniji naziv zvanja na srpskom jeziku, s pravom mogu biti ponosni na svoja dostignuća. Naravno, i dalje imamo problema sa razvojem složenog softvera, ali bez softverskog inženjerstva ne bismo istraživali prostor i ne bismo imali internet ili moderne telekomunikacije. Svi oblici putovanja bili bi opasniji i skuplji. Izazovi za čovečanstvo u 21. veku su klimatske promene, manje prirodnih resursa, promenljiva demografija i rastuća svetska populacija. Oslonićemo se na softversko inženjerstva da bismo razvili sisteme koji su nam potrebni da se nosimo sa ovim problemima.

U stvarnosti, firme koje razvijaju softver stalno su izložene pritiscima da što pre razviju novu verziju softvera, uz često i menjanjanje zahteva korisnika. S druge strane, softversko inženjerstvo je mlada inženjerska disciplina, i još je u razvoju. Zato, vrlo je važno da je studenti razumeju i primenjuju u zavisnosti od tipa softvera koji rzyvijaju kada se zapsle, težeći uvek da u što većoj meri primenjuju adekvatne metode softverskog inženjerstva, kako bi ostvarili kvalutetan softver sa atributima koji se očekuju od kvalitetnog softverskog proiyvoda.

Poglavlje 4

Softversko inženjerstvo primenom Interneta

EFEKAT PRIMENE VEB-ORIJENTISANOG SOFTVERSKOG INŽENJERSTVA

Koristici komuniciraju sa softverom preko veb pretraživača, a dele različite servisne usluge najčešće posredstvom računarskog oblaka (deljenje aplikacija, podataka i dr.)

Razvoj Interneta i World Wide Web-a, tj. Interneta, imao je dubok uticaj na sve naše živote. U početku je veb bio prvenstveno univerzalno dostupno skladište informacija i imao je mali uticaj na softverske sisteme. Ovi sistemi su radili na lokalnim računarima i bili su dostupni samo iz organizacije. Oko 2000. godine veb je počeo da se razvija, a pretraživačima je dodavano sve više funkcionalnosti. To je značilo da se mogu razviti sistemi zasnovani na vebu gde bi se, umesto korisničkog interfejsa posebne namene, ovim sistemima moglo pristupiti preko veb pretraživača. Ovo je dovelo do razvoja širokog spektra novih sistemskih proizvoda koji su pružali inovativne usluge, kojima se pristupalo preko veba. Oni se često finansiraju oglasima koji se prikazuju na ekranu korisnika i ne uključuju direktno plaćanje korisnika.

Pored ovih sistemskih proizvoda, *razvoj veb pretraživača koji su mogli da pokreću male programe i da obavljaju neku lokalnu obradu doveo je do evolucije poslovnog i organizacionog softvera.* Umesto pisanja softvera i njegovog postavljanja na računare korisnika, softver je postavljen na veb server. Ovo je znatno pojeftinilo promenu i nadogradnju softvera, jer nije bilo potrebe da se softver instalira na svaki računar. Takođe je smanjio troškove, jer je razvoj korisničkog interfejsa posebno skup. Gde god je to bilo moguće, preduzeća su prešla na interakciju zasnovanu na vebu sa softverskim sistemima kompanije.

Pojam softvera kao usluge predložen je početkom 21. veka. Ovo je sada postao standardni pristup isporuci sistemskih proizvoda zasnovanih na vebu kao što su Google Apps, Microsoft Office 365 i Adobe Creative Suite. Sve više softvera radi na udaljenim „oblacima“ umesto na lokalnim serverima i pristupa mu se preko Interneta. Računarski oblak *(*cloud*) čini ogroman broj povezanih računarskih sistema koji dele mnogi korisnici*. Korisnici ne kupuju softver već plaćaju prema tome koliko se softver koristi ili im se daje besplatan pristup u zamenu za gledanje reklama koje se prikazuju na njihovom ekranu. Ako koristite usluge kao što su veb-pošta, skladište ili video, koristite sistem zasnovan na oblaku.

UTICAJ VEBA NA ORGANIZACIJU SOFTVERA I NA SOFTVERSKO INŽENJERSTVO

Sada je softver veoma distribuiran, ponekad širom sveta. Poslovne aplikacije se ne programiraju od nule, već uključuju opsežnu ponovnu upotrebu komponenti i programa.

Pojava veba dovela je do dramatične promene u načinu organizovanja poslovnog softvera. Pre veba, poslovne aplikacije su uglavnom bile monolitne, pojedinačni programi koji su se izvodili na pojedinačnim računarima ili računarskim klasterima. Komunikacija je bila lokalna, unutar organizacije. Sada je softver veoma distribuiran, ponekad širom sveta. Poslovne aplikacije se ne programiraju od nule, već uključuju opsežnu ponovnu upotrebu komponenti i programa.

Ova promena u organizaciji softvera imala je veliki uticaj na softversko inženjerstvo za sisteme zasnovane na vebu. Na primer:

1. **Ponovna upotreba softvera** postala je dominantan pristup za izgradnju sistema zasnovanih na vebu. Kada pravite ove sisteme, razmišljate o tome kako možete da ih sastavite od već postojećih softverskih komponenti i sistema, često povezanih zajedno u ekom radnom okviru (framework).
2. Sada je opšte prihvaćeno da je nepraktično unapred specificirati sve zahteve za takve sisteme. ***Sistemi zasnovani na vebu se uvek razvijaju i isporučuju postepeno***.
3. Softver se može implementirati korišćenjem **servisno-orijentisanog softverskog inženjerstva**, gde su **softverske komponente samostalni veb servisi**.
4. Pojavile su se **tehnologije razvoja interfejsa** kao što su AJAKS (Holdener 2008) i HTML5 koje podržavaju kreiranje funkcionalno bogatih interfejsa unutar veb pretraživača.

Osnovne ideje softverskog inženjerrstva primenjuju se na softveru zasnovan na vebu, kao i na druge tipove softvera. Sistemi zasnovani na vebu postaju sve veći i veći, tako da su tehnike softverskog inženjerstva koje se bave velikim softverskim sistemima i složenošću relevantne za ove sisteme.

Poglavlje 5

Informacija: Metodologija realizacije nastave

PRINCIPI PRIMENE METODLOGIJE ODRŽAVANJA NASTAVE

"*Principi: aktivno učenje, problemski-orijentisano učenje i tačno-na-vreme*"

Kako je SE101 potpuno novi predmet, i verovatno jedini koji se bavi uvodom u softversko inženjerstvo već u prvom semestru, i pre nego što studenti nauče programiranje, to je veliki izazov kako oraganizovati izvođenje nastave koja bi bila intersantna za studente, u kojoj bi učestvovali sa puno pažnje i intersovanje i u koji bi učetsvovali. Kada studenti na bilo koji način učestvuju u nastavi, oni postaju aktivni učesnici nastave i time nesvesno, ostavruju poznati **princip "aktivnog učenja"** , ili "aktivne nastave" . Naš cilj je upravo to, da držimo aktivnu nastavu, u kojoj je student aktivan učesnik u nastavi.

Drugi princip koji bi želeli da primenimo u nastavi je **problemski-orijentisano učenje**. To je slučaj kada nastavnik i student zajedno traže rešenje zadatog problema, i student istražuje način i metode rešavanja datog problema. S obzirom na uvodni karakter ovog predmeta, nije jednostavno ostvariti o primenu ovog principa, jer studentima nedostaje neophodno znanja za rešavanje određenih zadatka razvoja softvera (npr., ne znaju programiranje). Međutim, pokušaćemo da pronađemo adekvatne probleme i teme za domaće zadatke i diskusije, a u okviru kojih ćemo tražiti najbolja rešenja.
Treći princi koj ćemo ostavriti je **princip, "tačno na vreme"** (just-in-time), koji je poznat u proizvodnim sistemima. To znači, uči odmah , kada je planirano programom ili kada ti je neko ynanje potrebno za rešavanje nekog problema i ne ostavljaj ništa za kasnije. To znači da *sve što ti je zadato uradi u zahtevanom roku*. To znači umesto tzv. «kampanjskog učenja» pred ispit, učite svake nedelje, po principu "nedalja-za-nedeljom" i u novu nedelju ulazite za završenim planiranim zadacima, u okviru vaših predispitnim obaveza. Bolje ćete pratiti sledeće predavanje, ako ste dobro razumeli prethodno. Ovaj princip zahteva planiranje vašeg vremen, tj. upravljanje vašeg vremena. Morate da napravite dnevne i nedeljne planova učenja. Zakon zahteva od studenata da posvete 40 sati učenja i studiranja nedeljno. Ako oko 20 sati provedu na nastavi, to znači da imaju za kućno učenje oko 20 sati nedeljno. Ako imaju po 5 predmeta u semestru, onda im za samostalno učenje na predmetu i za rešavanje njegovih predispitnih obaveza ostaje oko 4 sata u proseku nedeljno po predmetu (ako predmet ima 6 ESPB, a više ili manje zavisno od broja ESPB pedmeta). Samodisciplina je preduslov za uspešno upravljanje svojim vremenom, jer smo često u izazovima raznih vrsta, pritiskom da odstupimo od dnevnog plana učenja. Volja za samokontrolom mora biti jača od izazova i zadovoljstva koje ono donosi. Očekivan cilj upravljanja sobom, i uspeh u studiranju,

treba da imate stalno pred sobom, kada ste u dilemi da li odstupiti od dnevnog plana učenja. Oni koji to uspevaju, imaće uspešnu karijeru, bilo šta i gde da rade.

OČEKIVANE KORISTI OD PRIMENJENIH PEDAGOŠKIH METODA

Preduslov svih ovih rezultata je da student ima jaku motivaciju da uči i studira.

Ako uspemo da primenimo ove principe u izvršenju nastave, postignućemo sledeće:

- **Veća efektnost nastave**, tj. student ne samo da bolje da razume i pamti ono što je trebalo da nauči, već je počeo da "inženjetski razmišlja" tj. da analizira problem, kreativno razmišlja o mogućim rešenjima, diskutuje o preblemu sa kolegama, i traži jedno ili više mogučih rešenja, koje onda analizira i bira najbolje.
- **Veća efikasnost,** tj. student brže savladava potrebno gradivo, njegov saznajni proces je kraći, na vreme završava svoje predispitne obaveze, i spreman je za ispit već u prvom roku.
- **Postiže visok prosek ocena,** što može kasnije da mu donese niz prednosti, bilo pri nastavku studiranja, bilo pri zapsolenju.

Preduslov svih ovih rezultata je da student ima jaku motivaciju da uči. Ako voli to što studira, onda verovatno, to ne bi trebalo da bude problem. Ako ne voli to što studira, a želi da to studira, onda mora da zavoli to što studira. Da bi to ostvario, mora da pokuša da razume problematiku i da testira svoju sposobnost razumevanja programa i nastavnog gradiva koje predmet ima.

Da bi se ostvarili navedeni ciljevi i očekivanja, nastava na predmetu SE101 mora i da se izvede i na najsavremeniji pedagoški način:

- više diskusija i debata na predavanjima i vežbama
- aktivno učestvovanje studenata u nastavnom procesu,
- učenje rešavanjem tipičnih problema i izazova,
- na vreme i u roku završavati predviđene predispitne obaveze ,
- uključenje i gostujućih predavača iz IT industrije i dr.

PRIPREMA ZA PREDAVANJA

U periodu od dva dana pre predavanja, student treba da prouči nastavni materijal

Ovo zahteva da student dolazi pripremljen na predavanja i vežbe, da redovno i u roku radi svoje predispitne obaveze, a da one budu usklađene sa raspoloživom vremenu koje student može da posveti svom radu kod kuže.. Preporučujemo studentima da naprave detaljan dnevni i nedeljni plan realizacije svoje 40-to satne radne nedelje, tako da predvide sledeće vremenske okvire svoh rada:

- **Aktivna nastava** (predavanje i vežbe) – 20 sati nedeljno za sve predmete u semestru, tj. 4 sata za predmet SE201.
- **Kućni rad** – 4 sata nedeljn na ovom predmetuo, i to: 40 minuta za rešavanje domaćeg zadatka (10 nedelja) i 3 sata i 20 minuta za proučavanje nastavnog materijala lekcije (pre i posle predavanja, priprema za testiranje, analiza primera i rešavanje domaćeg zadataka).

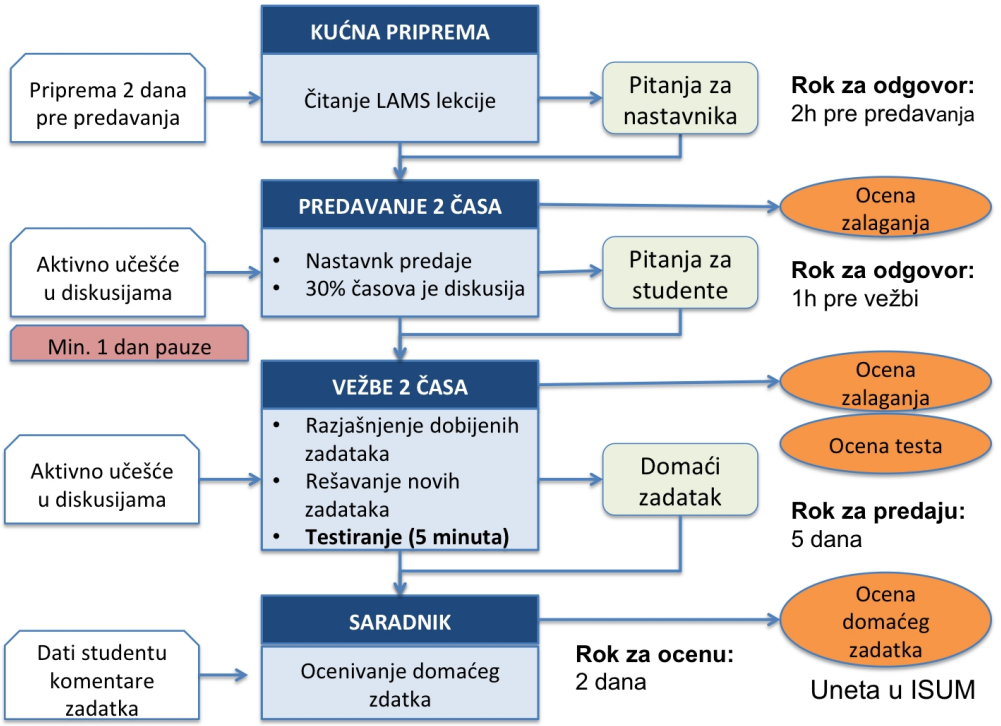
Vrlo je važno da student dođe pripremljen na predavanja, kako bi nastavnik mogao da izlaže samo najbitnije delove natavne materije na času, i time dobio više vremena za diskusije i debatu sa studentima o najinteresantnijim ili nejsnim pitanjima vezana za izloženu materiju.
U periodu od **dva dana pre predavanja**, student treba da prouči nastavni materijal, tj. predavanje kako bi bio spreman za diskusije na časovima predavanja. Ako mu nešto nije jasno, treba da pošalje mejlm pitanje nastavniku, koji će mu odgovoriti tokom diskusije na času predavanja. Da bi se proverilo da li je student proučio nastavni materijal lekcije, na početku predavanja treba da odgovorii na nekoliko pitanja u okviru kontrolnog testa.
U okviru svake LAMS lekcije student bi trebalo da uradi dva testa za samoproveru naučenog. Oni se ne ocenjuju poenima, jer služe da student utvrdi da li je savladao dobro ono što je slušao na predavanjima ili pročitao u nastavnom materijalu. Interni test samoprovere student može više puta da ponavlja, posle dodatnog proučavanja nastavnog materijala. Test pitanja se automatski i slučajno generišu iz baze mogućih pitanja (ima ih oko 2-3 puta više nego broj pitanja koja se postavljaju), kako bi se sprečilo uzajamno prepisivanje test rešenja sa drugim studentima. U Interesu studenta je da dobro proveri svoje znanje, jer je bitno da on nauči ono što treba, a ne da samo položi ispit. Pri radu internih testova samoprovere, nabolje je da **koristi mapu uma**, koja se daje ispred svakog testa, kao i na kraju svake lekcije. Mapa uma (brain map)omogućava brži, direktni pristup određenom delu nastavnih materijala, jer LAMS lekcija normalno zahteva redno pregledavanje svih sekcija lekcije, bez preskakanja. Zato, mapa uma omogućava studentu, kada dođe do testa u lekciji, da , ako mu je potrebno, potraži odgovor na pitanje za koje ne zna odgovor i onda da ponovi test, i da ako dobije isto pitanje, bude uspešniji . U praksi, mnogi studenti preskaću ove testove samoprovere, što je pogrešno, jer im oniipomažu da bolje shvate ono što je predavano, i da douče ono što testovi pokazuju da nije dobro student naučio ili razumeo.

NEDELJNI PLAN NASTAVE

Ovaj model nastave može da da rezultat samo ako je svaki student voljan da ga primenjuje.

Na početku semestra, student može preko LAMS-a da se upozna sa Planom i programom predmeta u kome se detaljno opisuju njegove obaveze, način kako se ocenjuju predispitne obaveze na predmetu. Takođe, u Planu nastave, može da vidi nastavne teme za svaku lekciju, očekivane ishode svake lekcije, navođrenje referenci za svaku temu sa predavanja i opisa šta se radi na vežbama.
Ovaj model nastave može da da rezultat samo ako je svaki student voljan da ga primenjuje. Važno je da na vreme radi predispitne obaveze (testove i domaće zadatke). Student treba domaći zadatak da uradi u roku od 7 dana od dana kada ga dobije, jer se time izbegava «kampanjsko učenje». Zato su predviđeno i umanjenje poena za domaće zadatke za koje se rezultati predaju kasnije, posle datog roka. Vrlo je važan kućni rad

studenata i njegova priprema za predavavanja i vežbe.
Na slici 1 dat je okvir za nedeljni raspored rada i učenja studenta na predmetu SE101



Slika 5.1 Nedeljni plan nastave na predmetu SE101 [autor]

OCENJIVANJE ZALAGANA STUDENTA U NASTAVI

Najvažnije je da student dobro planira svoje vreme i da se pridržava postavljenog plana učenja

Kod ocenjivanja zalaganja u nastavi, uzima se u obzir:

- da li student dolazi prirpemljen na predavanja, što se ogleda u pitanjima koje šalje nastaviku, odn. saradniku prečasova predavanja, odn. vežbanja, ili po diskusijama u kojima učestvuje na časovima predavanja i vežbi
- da li u predviđenim rokovima predaje domaće zadatke i radi testove
- da li radi testove samotestiranja (LAMS sistem daje takve informacije), da li čita nastavne materijale i dr.

Najvažnije je da student dobro planira svoje vreme i da se pridržava postavljenog plana učenja. Na američkim univerzitetma je fokus na radu kod kuće, jer na mnogim predmetima i nemaju vežbe. Očekuje se da studenti dolaze pripremljeni na predavanja, jer im je dato šta treba da pročitaju pre dolaska na predavaja. U slučaju SE101, to je naredna lekcija koju student treba da prouči najkasnije u periodu od dva dana pre predavanja (očekuje se da mu je za to potrebno oko 2 sata nedeljno). Naravno, očekuje se za proučavanje nastavnih materijala pre i posle predavanja i vežbi (zbog domaćeg zadatka), te u zbiru, student treba da planira najmanje 4 sata kućnog rada na predmetu nedeljno. Na taj način može biti spreman za polaganje ispita u prvom (januarsko-februarskom ispitnom roku).

Poglavlje 6

Vežba - Studija slučaja: Digitalno učenje

STUDIJA SLUČAJA: ILEARNING DIGITALNI SISTEM ZA UČENJE

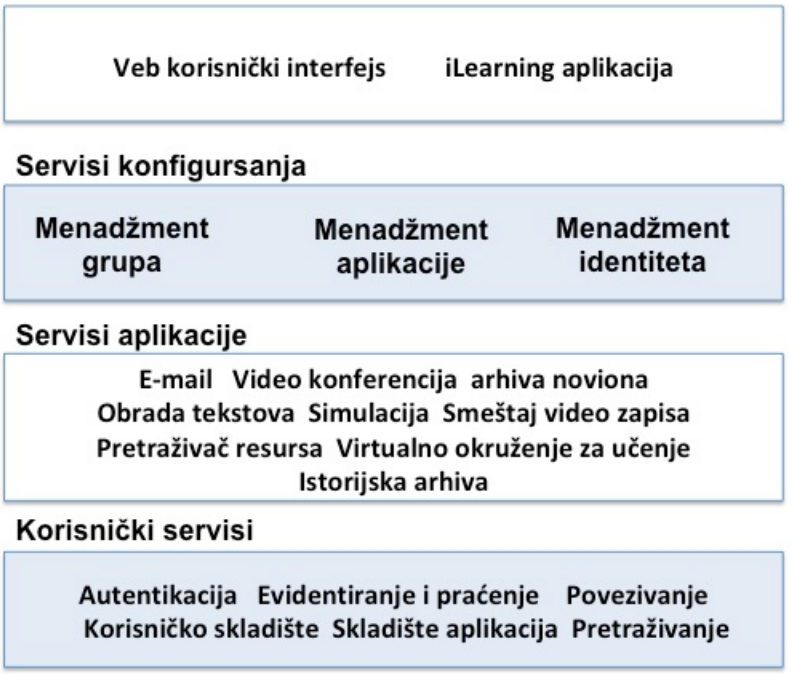
Sve komponente okruženja su servisi kojima se može pristupiti sa bilo kog mesta na Internetu.

Mnogi nastavnici tvrde da korišćenje interaktivnih softverskih sistema za podršku obrazovanju može dovesti do poboljšane motivacije učenika i dubljeg nivoa znanja i razumevanja kod učenika. Međutim, ne postoji opšta saglasnost o „najboljoj“ strategiji za učenje podržano računarom, a nastavnici u praksi koriste niz različitih interaktivnih alata zasnovanih na vebu za podršku učenju. Alati koji se koriste zavise od uzrasta učenika, njihovog kulturnog porekla, njihovog iskustva sa računarima, raspoložive opreme i preferencija uključenih nastavnika.

Digitalno okruženje za učenje je okvir u koji se može ugraditi skup alata opšte namene i specijalno dizajniranih alata za učenje, plus skup aplikacija koje su prilagođene potrebama učenika koji koriste sistem. Okvir pruža opšte usluge kao što su servis autentifikacije, sinhroni i asinhroni komunikacioni servisi i usluga skladištenja

Alati uključeni u svaku verziju okruženja biraju nastavnici i učenici kako bi odgovarali njihovim specifičnim potrebama. To mogu biti opšte aplikacije kao što su tabele, aplikacije za upravljanje učenjem kao što je virtuelno okruženje za učenje (VLE) za upravljanje podnošenjem i ocenjivanjem domaćih zadataka, igre i simulacije. Oni takođe mogu uključivati određeni sadržaj, kao što je sadržaj o američkom građanskom ratu i aplikacije za pregled i komentarisanje tog sadržaja.

Slika 1 je arhitektonski model visokog nivoa apstrakcije digitalnog okruženja za učenje (iLearn) koji je projektovan za upotrebu u školama za učenike od 3 do 18 godina. Usvojen pristup je da je ovo distribuirani sistem u kome su sve komponente okruženja servisi kojima se može pristupiti sa bilo kog mesta na Internetu. Ne postoji uslov da se svi alati za učenje sakupe na jednom mestu.



Slika 6.1 Arhitektura iLearning digitalnog okruženja za e-učenje [1, Fig.1.8]

USLUGE (SERVISI) KOJE NUDI ILEARNING DIGITALNO OKRUŽENJE

iLearning nudi integrisane i nezavisne usluge korisnicima

Sistem je servisno orijentisan sistem sa svim komponentama sistema koji se smatraju zamenljivim servisom. Postoje tri vrste usluga u sistemu:

1. **Uslužni servisi** koji pružaju osnovnu funkcionalnost nezavisnu od aplikacije i koje mogu koristiti druge usluge u sistemu. Komunalne usluge se obično razvijaju ili prilagođavaju posebno za ovaj sistem.
2. **Aplikacione usluge** koje pružaju specifične aplikacije kao što su e-pošta, konferencije, deljenje fotografija, itd., i pristup određenim obrazovnim sadržajima kao što su naučni filmovi ili istorijski izvori. Aplikacione usluge su eksterne usluge koje se ili posebno kupuju za sistem ili su dostupne besplatno preko Interneta.
3. **Usluge konfiguracije** koje se koriste za prilagođavanje okruženja sa određenim skupom aplikacionih usluga i za definisanje načina na koji se usluge dele između učenika, nastavnika i njihovih roditelja.

Okruženje je projektovano tako da se usluge mogu zameniti kako nove usluge postanu dostupne i da obezbedi različite verzije sistema koje su prilagođene uzrastu korisnika. To znači da sistem mora da podržava dva nivoa usluge integracija:

1. **Integrisane usluge** su usluge koje nude API (interfejs za programiranje aplikacije) i kojima drugi servisi mogu pristupiti preko tog API-ja. Direktna komunikacija usluga-usluga je stoga moguća. Usluga autentifikacije je primer integrisane usluge. Umesto da koriste sopstvene mehanizme za autentifikaciju, drugi servisi mogu da pozovu

uslugu autentifikacije da bi potvrdili autentičnost korisnika. Ako su korisnici već autentifikovani, onda servis za autentifikaciju može proslediti informacije o autentifikaciji direktno drugoj usluzi, preko API-ja, bez potrebe da se korisnici ponovo autentifikuju.

2. **Nezavisne usluge** su usluge kojima se jednostavno pristupa preko interfejsa pretraživača i koje rade nezavisno od drugih usluga. Informacije se mogu deliti sa drugim uslugama samo putem eksplicitnih radnji korisnika kao što su kopiranje i lepljenje; može biti potrebna ponovna autentifikacija za svaku nezavisnu uslugu.

Ako nezavisna usluga postane široko korišćena, razvojni tim može da integriše tu uslugu tako da ona postane integrisana i podržana usluga.

Poglavlje 7

Vežba - Zadaci za individualno rešavanje

ZADACI ZA VEŽBANJE

Zadaci za časove vežbi i za samostalni rad kod kuće.

1. Objasnite zašto profesionalni softver koji se razvija za kupca nisu samo programi koji su razvijeni i isporučeni.
2. Koja je najvažnija razlika između razvoja generičkog softverskog proizvoda i razvoja softvera po meri? Šta bi to moglo da znači u praksi za korisnike generičkih softverskih proizvoda?
3. Ukratko prodiskutujte zašto je obično jeftinije dugoročno koristiti metode i tehnike softverskog inženjeringa za softverske sisteme..
4. Na osnovu sopstvenog znanja o nekim tipovima aplikacija, objasnite, uz primere, zašto različite vrste aplikacija zahtevaju specijalizovane tehnike softverskog inženjerstva da podrže njihov dizajn i razvoj.
5. Objasnite zašto su osnovni principi softverskog inženjeringa procesa, pouzdanosti, upravljanja zahtevima i ponovne upotrebe relevantni za sve tipove softverskog sistema.
6. Objasnite kako elektronska povezanost između različitih razvojnih timova može podržati aktivnosti softverskog inženjerstva.
7. Nesertifikovanim pojedincima je i dalje dozvoljeno da se bave softverskim inženjeringom. Razgovarajte o nekim od mogućih nedostataka ovoga.
8. O „revoluciji dronova“ se trenutno raspravlja i raspravlja širom sveta. Dronovi su bespilotne leteće mašine koje su izgrađene i opremljene raznim vrstama softverskih sistema koji im omogućavaju da vide, čuju i deluju. Razgovarajte o nekim društvenim izazovima izgradnje takvih sistema.

Napomena: Očekuje se od svakog studenta, *da posle časova predavanja, prouči nastavni materijal sa predavanja, i da pokuša da odgovori na deo postavljenih zadataka i* da mejlom svoje odgovore pošalje saradniku koji drži časove vežbanja, ali najkasnije jedan sat pre održavanja vežbanja. Može poslati u svoja pitanja za koja bi želeo da se na vežbi razjasne. Saradnik vodi evidenciju studenata koji su poslali odgovore na postaljena pitanja ili zadatke. Na početku časa vežbe, navodi evidentirane probleme pojedinih studenata u rešavanu zadataka koje imaju pojedini studenti, i poziva studente koji su rešili te zadatke da objasne svoja rešenja ostalim studentima, a ako je potrebno, i saradnik daje svoja objašnjenja. Saradnik vodi evidenciju anagžovanja studenata na rešavanju yadataka pre časova vežbi, studenata joj izlažu svoja rešenja na časovima vežbanja kao i onih koji diskutuju. Na kraju semestra, na bazi ove evidencije, saradnik određuje za svakog studenta broj poena na zalaganje u nastavi, tj. na vežbama.

DOMAĆI ZADATAK BR. 1

Treba poslati saradniku vaš odgovor na postavljenu temu zadatjka, najkasnije 7 dana po održanoj vežbi, kako ne bi imali umanjenje broja poena za 50%.

Domaći zadatak ima za cilj da utvrdi da li je student, posle časova predavanja i vežbanja, sposoban da samostalno reši zadatak koji je sličan po karakeru i obimu rada zadacima koji su rađeni na vežbama i koji su diskutovani i razjašnjeni na vežbama. Obim zadatka može da zahteva najviše 30 do 40 minuta rada studenta. Svaki student mora da ima poseban zadatak, a na temu koja se ovde definiše. Studenti koji pošalju identična ili vrlo bliska rešenja, neće im ta rešenjabiti prihvaćena, tj. dobiće 0 poena. Da bi to izvbegao, student treba da pri formulaciji svog zadatka na zadatu temu dormuliše svoj zadatak za koji postoji mala verovatnoća da će drugi student definisaniti isti ili vrlo sličan zadatak, na primer, korišćenjem primera koji je njima poznat i koji je verovazno unikatan. Studenti mogu o svojim zadacima da razmenjuju informacije ili preko foruma predmeta ili preko neke od socijalnih mreža. Međutim, ne smeju kopirati rešenja sa Interneta i socijalnih mreža, već moraju da pripreme svoj sopstveni odgovr na zadatak koji su postavili.

Rok za predaju zadataka je 7 dana nakon izdavanja, tj. od dana održavanja vežbi određene lekcije, a posle tog roka umanjuje ostvaren broj poena za 50%. Krajnji rok za predaju domaćeg zadatka i za studente tradicionalne nastave i za studente onlajn nastave je 10 (deset) dana pre ispitnog roka u kome student polaže ispit.

Tekst domaćeg zadatka br. 1:

Sobzirom na sadržaj prve lekcije, dajte esejski osvrt na dole postavljenu temu, i za nju vezana pitanja na koja bi mogli da date osvrt i svoje mešljenje. Naravno, možete i proširiti broj relevantnih pitanja na koje bi želeli da izrazite svoje poglede i stavove.

Često studenti misle da kada nauče programiranje da su spremni da se zaposle na poslovima razvoja softvera, u početku, kao programeri. Dobar deo njih i prekida dalje studiranje. Dajte u svom odgovoru osvrt na ovu i na sledeća pitanja: Koja je razlika između softverskog inženjerstva i programiranja, kao i između posla programera i inženjera softvera? Koje su početne zarade *u Srbiji programera, a koja diplomiranih inženjera softvera? Da li bi ste se vi opredelili da se zaposlite kao programer i prekinete studije, ili prvo da diplomirate i da se onda zaposlite kao inženjer softvera? Zašto? Da li imate plan vaše karijere? Kakvu karijeru* želelite da *razvijete u oblasti softverskog inženjerstva? Zašto ste upisali Softversko inženjerstvo?*

Poglavlje 8

Zaključak

ZAKLJUČAK

Ključne poruke lekcije

1. Softversko inženjerstvo je inženjerska disciplina koja se bavi svim aspektima proizvodnje softvera.
2. Softver nije samo program ili skup programa, već uključuje svu elektronsku dokumentaciju koja je potrebna korisnicima sistema, osoblju za osiguranje kvaliteta i programerima. Osnovni atributi softverskog proizvoda su: mogućnost održavanja, pouzdanost i sigurnost, efikasnost i prihvatljivost.
3. Softverski proces uključuje sve aktivnosti uključene u razvoj softvera. Aktivnosti na visokom nivou apstrakcije softverskog procesa su: specifikacija zahteva i projektnog rešenja, impementacija (programiranje), validacija softverskog proizvoda i evolucija softverskog sistema tokom njegovog životnog veka.
4. Postoji mnogo različitih tipova sistema i svaki zahteva odgovarajuće alate i tehnike softverskog inženjerstva za njihov razvoj. Nekoliko, ako ih ima, specifičnih tehnika projektovanja i implementacije je primenljivoi na sve vrste sistema..
5. Osnovne ideje softverskog inženjerstva su primenljive na sve tipove softverskog sistema. Ove osnove uključuju: a) upravljanje softverskim procesom, b) pouzdanost i sigurnost softvera, c) inženjerstvo zahteva i d) ponovna upotreba softvera, odn. softverskih komponenti.

LITERATURA

1. Obavezna literatura:

1. Nastavni materijal za e-učenje na predmetu SE101 Razvoj softvera i inženjera softvera, Univeziitet Metropolitan, školska 2022/23. godina
2. Ian Sommervile, Software Engineering - Chapter 1.1 Professional software development, Tenth Edition, Pearson Education Inc., 2016.

2. Dopunska litertura:

1. King Graham, Wang Yingxu, Software Engineering Processes - Principles and Applications,CRC Press (2000)
2. Ian Sommerville, Engineering Software Products - An Introduction to Modern Software Engineering, Global Edition, Pearson (2021)

3. Veb lokacije :

1. **https://www.guru99.com/what-is-software-engineering.html**
2. **https://www.guru99.com/difference-system-software-application-software.html#2**
3. **https://www.softwareengineerinsider.com/articles/what-is-software-engineering.html#.WD37TbIrLIU**