# UVOD U PROGRAMSKO INŽENJERSTVO

## PROGRAMSKO INŽENJERSTVO

Ekonomija **SVIH** razvijenih zemalja ovisi o softveru.

* Sve više i više sustava je upravljano softverom
* Izdaci na softver predstavljaju značajan dio BND razvijenih zemalja

Osnovna definicija:

* *Programsko inženjerstvo je inženjerska disciplina koja se bavi svim aspektima razvoja softvera, tj. bavi se teorijom, metodama i alatima za profesionalan razvoj softvera.*

Pojam PI se po prvi put pojavljuje 1968. na konferenciji koja je održana zbog tzv. "softverske krize".

## ŠTO JE SOFTVER?

Računalni program **i pripadajuća dokumentacija** (zahtjevi, modeli dizajna , korisnička dokumentacija)

Pri tome se razlikuju dva osnovna tipa softverskih produkata:

* ***generički*** – organizacija koja razvija program kontrolira i njegovu specifikaciju, prodaje se velikom broju različitih korisnika – ( eng. Comercial Of The Shelf - COTS)
* ***po narudžbi*** (eng. custom) – kupac koji naručuje program definira njegovu specifikaciju.

Novi softver se može napraviti korištenjem novih programa, konfiguriranjem generičkih produkata ili ponovnim korištenjem postojećeg softvera.

## NAJČEŠĆA PITANJA O PI-U

* Što je softver?
* Što je programsko inženjerstvo?
* Koja je razlika između PI-a i inženjerstva sustava?
* Što je softverski proces?
* Što je model softverskih procesa?
* Kako su raspoređeni troškovi kod korištenje PI-a?
* Što je CASE (Computer Aided Software Engineering)?
* Koje su karakteristike dobrog softvera?
* Koji su osnovni izazovi s kojima se susreće PI?

## KOJA JE RAZLIKA IZMEĐU PI-A I INŽ. SUSTAVA?

* Inženjerstvo sustava je starija disciplina od PI i bavi se svim aspektima razvoja i održavanja kompleksnih sustava (npr. zrakoplovi).
* S vremenom se udio softver u takvim sustavima značajno povećao.

Sistem inženjeri su uključeni u specifikaciju sustava, definiranje njegove arhitekture, te integraciju različitih dijelova s ciljem stvaranja gotovog sustava.

## ŠTO JE SOFTVERSKI PROCES?

Softverski proces je niz aktivnosti čiji je cilj stvoriti softver.

Postoje četiri osnovne aktivnosti koje su zajedničke za sve softverske projekte:

* ***Specifikacija*** – što sustav treba raditi i o kojim ograničenjima treba voditi računa pri razvoju.
* ***Razvoj***(eng. development) – produkcija specificiranog softvera.
* ***Validacija*** – provjera da softver radi ono što je kupac naručio.
* ***Održavanje***(eng. evolution) – promjena softvera u skladu s promjenom zahtjeva tržišta.

## ŠTO JE MODEL SOFTVERSKIH PROCESA?

Pojednostavljena prezentacija softverskog procesa, prikazana iz određene perspektive.

Primjeri procesnih perspektiva su:

* radni slijed (eng. workflow) – slijed aktivnosti,
* tok podataka / model aktivnosti – tok informacija i njihovu transformaciju u sustavu,
* Uloga/akcija perspektiva – perspektiva iz koje se vidi tko radi što u sustavu.

Općeniti procesni modeli:

* vodopadni
* iterativni razvoj
* PI bazirano na komponentama (eng. component based) .

## KAKO SU RASPOREĐENI TROŠKOVI KOD KORIŠTENJE PI-A?

Približno 60% troška otpada na specifikaciju i razvoj aplikacije, 40 % su troškovi testiranja.

* Za softver po narudžbi troškovi održavanja često premašuju troškove razvoja.

Troškovi razvoja softvera ovise o vrsti sustava koji se razvija, te o karakteristikama koje bi taj sustav trebao zadovoljiti (performanse, pouzdanost, …).

Raspodjela troškova po fazama razvoja ovisi o razvojnom modelu koji se koristi.



*Raspodjela Troškova Po Aktivnostima*

## ŠTO JE CASE (COMPUTER AIDED SOFTWARE ENGINEERING)?

Softverski sustavi koji pružaju automatiziranu podršku u aktivnostima vezanim za softverske procese

* MS Visio, Oracle Designer, Rational Rose, Rational XDE

***Viši CASE alati***(eng. Upper-CASE)

* Alati koji podržavaju rane procesne aktivnosti poput zahtjeva i dizajna.

***Niži CASE alati***(eng. Lower-CASE)

* Alati koji podržavaju kasnije procesne aktivnosti poput programiranja, pronalaženja grešaka i testiranja.

**Integrirani CASE** (ICASE)

* podržava cijeli životni ciklus.

## KOJE SU KARAKTERISTIKE DOBROG SOFTVERA?

Softver bi trebao omogućiti korisniku traženu funkcionalnost i performanse, a pri tome bi ga trebalo biti lako održavati, biti pouzdan i prihvatljiv.

* **ODRŽAVANJE** - Program bi trebao biti pisan tako da ga se lako može mijenjati u skladu s korisničkim zahtjevima.
* **POUZDANOST** - Ne smije uzrokovati fizičke i/ili ekonomsku štetu u slučaju pada sustava.
* **EFIKASNOST** - Ne smije prekomjerno koristiti resurse (procesor, memorija) i odziv mora biti što brži
* **PRIHVATLJIVOST I ISKORISTIVOST** (ENG. ACCEPTABILITY & USABILITY) - Korisnik mora biti prihvaćen od strane korisnika .

## KOJI SU OSNOVNI IZAZOVI S KOJIMA SE SUSREĆE PI?

* + **Heterogenost -** Ovo je izazov kako razviti tehniku za izradu softvera koji je dovoljno fleksibilan da se može nositi s različitim platformama i izvršnim okolinama-
  + **Isporuka -** Ovo je izazov čiji je cilj što više skratiti vrijeme razvoja softvera bez da se kompromitira njegova kvaliteta.
  + **Povjerenje -** Razvoj tehnika koje demonstriraju da korisnici mogu vjerovati softveru.

## PROFESIONALNA I ETIČKA ODGOVORNOST

PI uključuje mnogo širi spektar odgovornosti nego jednostavno primjenu tehničkih znanja.

Etično ponašanje je mnogo više od jednostavnog podržavanja zakona a između ostalog se odnosi na:

* Povjerenje
* Kompetencija
* Intelektualno vlasništvo
* Zlouporaba računala

Organizacije poput ACM-a, IEEE i “British Computer Society” su objavili kod profesionalnog ponašanja tzv. “kod etike“ koji definira osnove etičnog ponašanja

## PRIMJERI ETIČKIH DILEMA

Ne slaganje s odlukama višeg menadžmenta

Zaposlenik radi na neetičan način:

* bez završenog testiranja otpušta sustav s kritičnom sigurnošću

Sudjelovanje u razvoju vojnog naoružanja ili nuklearnih sustava.

# SOCIJALNO – TEHNIČKI SUSTAVI

## ŠTO JE SUSTAV?

* Svrhovita kolekcija međusobnih povezanih komponenti koje rade zajedno kako bi postigli neki zajednički cilj.
* Sustav može uključiti softver, mehaničke i elektroničke komponente te biti upravljan ljudima.
* Svojstva i ponašanje sustava su jako zamršeno povezani.

## SUSTAVI KOJI UKLJUČUJU SOFTVER

Tehnički sustavi bazirani na računalima:

* Sustavi koji uključuju i hardver i softver, ali operatori i operacijski procesi ne smatraju se dijelom sustava.
* Osobe i organizacije koriste takav sustav za neku namjenu, ali sam sustav nije svjestan svrhe za koju se koristi (npr. TV, mobilni telefoni, softver).

Socijalno-tehnički sustavi:

* Sustavi koji uključuju tehničke sustave ali i operacijske procese i ljude koji su u interakciji s tehničkim sustavom.
* Ovakvi sustavi se razvijaju prema organizacijskoj politici i pravilima.

## OSNOVNE KARAKTERISTIKE S.-T. SUSTAVA

***Izranjajuća svojstva (eng. emergent properties) -***

* Svojstva sustava kao cjeline koja ovise o komponentama sustava i njihovim međuvezama.

***Ne-deterministička svojstva***

* Sustavi ne generiraju uvijek iste izlazne podatke za neki ulazni set parametara jer ponašanje sustava ovisi o čovjeku (operatoru).

***Kompleksne veze s organizacijskim ciljevima***

* U kolikoj mjeri sustav podržava organizacijske ciljeve ne ovisi samo o samom sustavu.

## IZRANJAJUĆA SVOJSTVA SUSTAVA

* Svojstva sustava kao cjeline prije nego svojstva koja se mogu izdvojiti iz svojstava njegovih komponenti.
* Posljedica su veze između komponenti sustava
* Može ih se stoga jedino promatrati i mjeriti jednom kada se komponente integriraju u sustav.

## PRIMJERI IZRANJAJUĆIH SVOJSTAVA

|  |  |
| --- | --- |
| Svojstvo | Opis |
| **Volumen** | Ukupni prostor koji zauzima sustav ovisi o tome kako je organizirano spajanje i povezivanje komponenti. |
| **Pouzdanost**  **(eng. reliability)** | Pouzdanost sustava ovisi o pouzdanosti njegovih komponenti, ali neočekivane interakcije mogu uzrokovati nove vrste grešaka i na taj način ugroziti pouzdanost cijelog sustava. |
| **Sigurnost**  **(eng. security)** | Sigurnost sustava (njegova otpornost na napade) je složena karakteristika i ne može biti jednostavno izmjerena. Dizajneri možda nisu predvidjeli gdje se sve može dogoditi napad. |
| **Popravljivost**  **(eng. repairability)** | Prikazuje koliko je jednostavno popraviti grešku u sustavu kada se otkrije. Ovisi o tome je li sustav u stanju prepoznati grešku, pristupiti tim komponentama i ispraviti ih ili izmijeniti |
| **Jednostavnost korištenja (eng. usability)** | Prikazuje koliko je jednostavno koristit sustav. Ovisi o tehničkim komponentama sustava, operatorima i radnoj okolini. |

## VRSTE IZRANJAJUĆIH SVOJSTA

***Funkcionalna***

* Javljaju se kada svi dijelovi sustava rade zajedno kako bi se postigao neki cilj.
* Npr. bicikl ima funkcionalno svojstvo da je prijevozno sredstvo kada se sastavi od pojedinačnih komponenti.

***Ne-funkcionalna***

* Povezani su s ponašanjem sustava u radnoj okolini.
* Npr. pouzdanost, performanse, osiguranje (eng. safety), sigurnost.
* Najčešće su kritični jer ukoliko se ne ostvari neki minimalni definirani nivo ovih svojstava sustav može biti nestabilan.

## POUZDANOST SUSTAVA

* Zbog međuovisnosti komponenti greške se mogu propagirati kroz sustav.
* Greške se vrlo često događaju zbog nepredviđenih međuovisnosti komponenti.
* Nemoguće je predvidjeti sve međuovisnosti.
* Mjera pouzdanosti softvera može dati krivu sliku o pouzdanosti sustava.

## UTJECAJI NA POUZDANOST SUSTAVA

* **Hardverska pouzdanost**
* Koja je vjerojatnost da će doći do kvara na nekoj od hardverskih komponenti i koliko treba da se to popravi?
* **Softverska pouzdanost**
* Kolika je vjerojatnost da će neka od softverskih komponenti generirati krivi rezultat operacije.
* Greške su najčešće kratkotrajne i sustav nastavlja s radom.
* **Pouzdanost operatera**
* Kolika je vjerojatnost da će operator sustava napraviti neku grešku?

## NEŽELJENA SVOJSTVA

Sustav ne smije imati neke karakteristike:

* osiguranje – sustav se ne smije ponašati na nesiguran način.
* zaštita – sustav ne smije dozvoliti pristup neautoriziranim osobama.

Mjerenje ovakvih svojstava je jako složeno.

## INŽENJERSTVO SUSTAVA

Niz aktivnosti koje obuhvaćaju:

* specifikacija,
* dizajn,
* implementacija,
* validacija,
* puštanje u rad,
* održavanje socijalno-tehničkih sustava.

Brine se o:

* softveru, hardveru te interakciji s korisnicima i okolinom,
* uslugama koje pruža sustav, ograničenjima u izradi i radu te načinima na koji će se koristiti.

## KARAKTERISTIKE PROCESA U INŽENJERSTVU SUSTAVA

Najčešće prate vodopadni model zbog potrebe za *paralelnim razvojem različitih dijelova sustava*

* Jako je malo prostora za iteracije između pojedinih dijelova jer su promjene u hardveru najčešće vrlo skupe, pa često softver treba kompenzirati hardverske nedostatke.

Uključuje inženjere iz različitih disciplina koji moraju raditi zajedno. Tu se javlja mnogo potencijalnih nesporazuma jer različita područja koriste različiti rječnik.



## DEFINIRANJE ZAHTJEVA SUSTAVA

Specificira što bi sustav trebao raditi te definira njegova

osnovna svojstva.

Ova faza rezultira s tri tipa zahtjeva:

* ***Apstraktni funkcionalni zahtjevi***– osnovne funkcije

sustava se definiraju na apstraktan način.

* ***Svojstva sustava***– definiraju se svi ne-funkcionalni zahtjevi .
* ***Neželjene karakteristike***– specificira se što se smatra neprihvatljivim ponašanjem sustava.

Potrebno je još i definirati cjelokupni organizacijski ciljevi za sustav – tj. zašto se sustav nabavlja.

Npr. Poslovna zgrada nabavlja alarmni sustav za detekciju požara i provale:

* Funkcionalni ciljevi - Osigurati internu i eksternu dojavu u oba slučaja.
* Organizacijski ciljevi - Osigurati normalno funkcioniranje …

## PROBLEMI KOD DEFINIRANJA ZAHTJEVA

Kompleksni sustavi se često razvijaju kako bi riješili "čudni" problemi:

* Problemi koji nisu u potpunosti jasni
* Složeni problemi kod kojih ima jako puno povezanih entiteta da je nemoguće dati jasnu specifikaciju
* Problemi koji se mijenjaju za vrijeme izrade specifikacije

Potrebno je predvidjeti hardverski/komunikacijski razvoj za vrijeme života sustava

Teško je definirati ne-funkcionalne zahtjeve bez poznavanja strukture komponenti sustava.

## PROCES DIZAJNA SUSTAVA



Brine se o tome koje će komponente sustava

osigurati koje funkcionalnosti.

Sastoji se od nekoliko faza:

* Podjela zahtjeva – zahtjevi se organiziraju u grupe.
* Identifikacija podsustava – identificira se niz podsustava

koji zajedno zadovoljavaju zahtjeve.

* Dodjela zahtjeva podsustavima – uzrokuju određene probleme kada se integriraju komercijalni produkti.
* Specifikacija funkcionalnosti podsustava



* Definiranje sučelja podsustava- vrlo važna

aktivnost za paralelan razvoj podsustava.

## PROBLEMI KOD DIZAJNA SUSTAVA

* Podjela zahtjeva na hardverske, softverske i ljudske komponente često zahtjeva jako puno pregovora.
* Probleme koje je teško dizajnirati se vrlo često smatra da se trebaju riješiti na jednostavan način korištenjem softvera.
* Hardverske platforme nogu biti neprikladne za softverske zahtjeve, što se treba kompenzirati softverom.

## VEZA IZMEĐU ZAHTJEVA I DIZAJNA

Jako zamršena i nije jednostavna, npr.:

* Ograničenja koja nameće okolina sustava i drugi sustavi ograničavaju dizajn, čak se može dogoditi da određeni dizajn bude jedan od zahtjeva.

Korištenje prototipa može biti neophodno za strukturiranje zahtjeva.

Dizajniranjem sustava se više uči o samim zahtjevima.



## SPIRALNI MODEL ZAHTJEVA/DIZAJNA

## MODELIRANJE SUSTAVA

U fazi zahtjeva/dizajna korisno je prikazati sustav preko modela – prikazuje sustav kao niz komponenti i veza među tim komponentama.

Osnovne karakteristike:

* Grafički prikaz - predstavlja se blok dijagramom
* Uključuje glavne tokove informacija između podsustava
* Identificira različite tipove ili funkcionalne komponente modela

## PROTUPROVALNI SUSTAV ARHITEKTURA SUSTAVA ZA KONTROLU ZRAČNOG PROMETA

## OPIS PODSUSTAVA PROTUPROVALNOG SUSTAVA



|  |  |
| --- | --- |
| **PODSUSTAV** | **OPIS** |
| **Senzor pokreta** | Detektira pokrete us sobi koju nadzire sustav |
| **Senzor vrata** | Detektira otvaranje vanjskih vrata zgrade |
| **Kontroler alarma** | Kontrolira rad sustava |
| **Sirena** | Emitira zvučno upozorenje kod neovlaštenog ulaza |
| **Generator glasa** | Generira glasovnu poruku koja upućuje na lokaciju uljeza |
| **Telefonski poziv** | Naziva vanjski kontrolni centar (zaštitarsku agenciju, policiju, …) |



## RAZVOJ PODSUSTAVA

* U ovoj fazi svi podsustavi identificirani

u prethodnim fazama se razvijau.

* Tipični paralelni projekti razvoja hardvera,

softvera i komunikacije među njima.

* Može uključivati COTS (eng. Comercial Off The Shelf) sustave.
* Nedostatak komunikacije među implementacijskim timovima
* Usporen mehanizam za predlaganje promjena na sustavu može značiti da planirano vrijeme razvoja može biti produženo zbog potrebe za ispravljanjem dijelova.

## INTEGRACIJA SUSTAVA

* Proces sastavljanja hardvera, softvera i ljudi kako bi se stvorio konačan sustav.
* Trebao bi se raditi u inkrementima tako da se podsustavi dodaju jedan po jedan.
* Javljaju se problemi među sučeljima podsustava.
* Može biti problema s nekoordiniranim isporukama komponenti sustava.

## INSTALACIJA SUSTAVA

Nakon što završi izrada sustava sustav treba instalirati u radnu okolinu naručitelja:

* Pretpostavke o radnoj okolini mogu biti pogrešne
* Ljudi se mogu odupirati uvođenju novog sustava
* Sustav treba koegzistirati s alternativnim sustavima neko vrijeme
* Može biti problema s fizičkom instalacijom

## ODRŽAVANJE SUSTAVA

Veliki sustavi imaju dug životni vijek zbog čega moraju evoluirati kako bi bili u skladu s promjenom zahtjeva.

Održavanje je jako skupo:

* Svaka promjena se mora sagledati s tehničke i poslovne perspektive i treba doprinijeti poslovnim ciljevima.
* Podsustavi su u međudjelovanju pa se mogu pojaviti nepredviđeni problemi.
* Najčešće nisu dokumentirani razlozi za određeni dizajn, pa oni koji rade izmjene to trebaju dokučiti.
* Promjene narušavaju strukturu osnovnog sustava i što se radi više promjena u sustavu on postaje nestabilniji.

## NASLIJEĐENI SUSTAVI (ENG. LEGACY).

Postojeći socijalno-tehnički sustavi koji su u upotrebi jako dugo vremena, koriste zastarjelu tehnologiju (hardver i softver) a od ključne su važnosti za poslovanje.

Često su važni za poslovanje i pre riskantno ili nemoguće ih je u potpunosti odbaciti.

* Sustav računa korisnika banke;
* Sustav održavanja zrakoplova.

Naslijeđeni sustavi često ograničavaju nove poslovne procese i odnose velik dio budžeta kompanije.

## LOGIČKI DIJELOVI I MEĐUDJELOVANJA NASLIJEĐENIH SUSTAVA



Hardver – zastarjela "mainframe" računala koja više nisu dostupna

* **Softverska podrška** – sustav se oslanja podršku (npr. operacijski sustav, prevodioce, ) koju više ne nude originalni dobavljači.
* **Aplikacijski softver** – može biti pisan u nekom od zastarjelih programskih jezika.
* **Aplikacijski podaci** – velika količina podataka koji mogu biti nekonzistentni, duplicirani, …
* **Poslovni procesi** (procesi koji se koriste kako bi se ostvario cilj poslovanja) – dizajnirani su oko naslijeđenih sustava i ograničeni funkcionalnostima koje nude.
* **Poslovna politika i pravila** – Definiraju na koji način se poslovanje treba obavljati te njegova ograničenja – mogu biti implicitni i ugrađeni u softver sustava.

## RASHOD SUSTAVA

* Uklanjanje sustava iz rada po završetku životnog ciklusa.
* Može zahtijevati uklanjanje materijala (npr. opasnih kemikalija) koji mogu zagaditi okoliš.
* Može zahtijevati restrukturiranje i konverziju podataka kako bi se mogli koristiti u nekom drugom sustavu.

## ORGANIZACIJA, LJUDI I RAČUNALNI SUSTAVI

* Socijalno-tehnički sustavi su organizacijski sustavi koji se koriste s namjerom da isporuče neke organizacijske ili poslovne ciljeve.
* Ukoliko onaj koji radi sustav ne razumije organizacijsku okolinu u kojoj se sustav koristi, vjerojatnije je da sustav neće odgovarati stvarnim potrebama poslovanja i njegovih korisnika.
* Promjene poslovnih procesa
* Je li sustav zahtjeva promjene poslovnih procesa u radnoj okolini?
  + Promjene poslova
* Je li sustav oduzima posao radnicima ili ih prisiljava da promjene način rada?
  + Organizacijske promjene
* Je li sustav mijenja strukturu političke moći u organizaciji?

## NABAVA SUSTAVA

Nabava sustava za neku organizaciju s ciljem dostizanja jednog ili više poslovnih ciljeva

Dio specifikacije sustava kao i dizajna arhitekture je obično neophodan prije nabave:

* Specifikacija je potrebna za definiranje ugovora za razvoj sustava
* Specifikacija može dopustiti nabavku komercijalnog (COTS) sustava, što je najčešće jeftinije nego njegov razvoj od početka.

Veliki kompleksni sustavi se obično sastoje od mješavine komercijalnih i komponenti po narudžbi.

* Procesi nabave se razlikuju za različite komponente.

## PROCES NABAVE SUSTAVA



## SOCIJALNO-TEHNIČKI SUSTAVI

|  |
| --- |
| Poslovni procesi |
| Aplikacijski softver |
| Softver za podršku |
| Hardver |

# SOFTVERSKI PROCESI

## SOFTVERSKI PROCESI

Niz aktivnosti koji vodi do razvoja softvera.

Iako postoji mnogo softverskih procesa slijedeće aktivnosti su im svima zajedničke:

* **Specifikacija** – funkcionalnosti softvera i njegova ograničenja
* **Dizajn i implementacija** – produkcija softvera koji odgovara zahtjevima
* **Validacija i verifikacija** – provjera radi li softver ispravno i ono što je kupac tražio
* **Održavanje (evolucija)** – izmjene softvera kako bi pratio promjene zahtjeva.

Model softverskog procesaje apstraktni prikaz procesa, koji ga predstavlja iz određene perspektive – daje parcijalni prikaz procesa.

## SPECIFIKACIJA SOFTVERA

* Naziva se još na engleskom “**requirements engineering**”.
* Proces u kojem se utvrđuje koje usluge treba pružiti sustav i koja su njegova ograničenja (npr. broj korisnika koji mogu istovremeno pristupiti nekom podatku).
* Greška koja nastane u ovoj fazi jako je skupa pogotovo ako se pronađe tek kod verifikacije.
* Procesi specifikacije softvera:
* Studija izvedivosti (eng. feasibility study) – ići u projekt ili ne?
* Opisivanje i analiza zahtjeva
* Specifikacija zahtjeva



* Validacija zahtjeva

**PROCES SPECIFIKACIJE**

**ZAHTJEVA**

## DIZAJN I IMPLEMENTACIJA SOFTVERA

Proces konverzije specifikacije u sustav koji obavlja svoju funkciju.

Dizajn softvera - struktura programa koji se realizira, sučelja među komponenta, algoritama koji se koriste, organizacije podataka, …

Implementacija - Pretvorba ove strukture u izvršni program

Aktivnosti dizajna i implementacije su usko povezane i mogu se isprepletati ovisno koji se procesni model koristi.

## PROCES DIZAJNA SOFTVERA



## AKTIVNOSTI PROCESA DIZAJNA

* **Dizajn arhitekture** – identifikacija pod sustava
* **Apstraktna specifikacija** – usluga i ograničenja svakog podsustava
* **Dizajn sučelja** –prema drugim pod sustavima
* **Dizajn komponenti** – usluga dodijeljenih komponentama
* **Dizajn strukture podataka** – detaljan dizajn organizacije podataka u sustavu
* **Dizajn algoritama** – koji osiguravaju neke od usluga

## STRUKTURIRANE METODE

Sistematski pristup razvoju dizajna softvera koji se dokumentira kao niz grafičkih modela.

Mogu podržavati sve ili samo neke od slijedećih modela:

* ***Objektni model***– prikazuje klase objekata u sustavu i njihove međuovisnosti
* ***Slijedni model***– prikazuje akcije objekata u sustavu koji obavlja svoje funkcionalnosti
* ***Model prijelaza stanja***– prikazuje stanja sustava te akcije koje uzrokuju prijelaz iz jednog stanja u drugo
* ***Strukturni model***– prikazuje komponente sustava i načina na koji su povezane
* ***Model toka podataka***– prikazuje transformacija podataka koja se događa za vrijeme rada sustava.

## PROGRAMIRANJE I PRONALAŽENJE GREŠAKA

Programiranje je slijedeći korak u fazi razvoja softvera s koji se često kreće već pri kraju faze dizajna (osim kod kritičnih sustava).

Programiranje je osobna aktivnost – ne postoji generički programerski proces.

Programeri provode programsko testiranje kako bi se otkrile i uklonile greške u programu.

Proces uklanjanja pogreški:



## VALIDACIJA SOFTVERA

* Verifikacija i validacija (V&V) se provodi s ciljem da se pokaže kako odgovara specifikaciji i zahtjevima korisnika.
* Uključuje provjeru procesa i testiranje sustava.
* Testiranje sustava uključuje pokretanje sustava po testnim scenarijima koji se pišu iz specifikacije zahtjeva.



***PROCES TESTIRANJA***

## FAZE TESTIRANJA

**Testiranje komponenti ili jedinično testiranje**

* Komponente se pojedinačno testiraju
* Komponente mogu biti funkcije ili objekti ili smislena grupa jednog od njih.

**Testiranje sustava**

* Testiranje sustava kao cjeline odgovara li funkcionalnim i nefunkcionalnim zahtjevima. Testiranje izranjajućaih svojstava je posebno važno.

**Test prihvaćanja**

* Testiranje s korisničkim podacima kako bi se provjerilo je li on odgovara zahtjevima korisnika.



## ODRŽAVANJE (EVOLUCIJA) SOFTVERA



Softver je sam po sebi fleksibilan i može

se lako mijenjati. Kako se zahtjevi

mijenjaju kroz promjene poslovnih

okolnosti, softver koji podržava poslovanje

također treba evoluirati i mijenjati se.

## MODELI SOFTVERSKIH PROCESA

**Generički modeli**

* Vodopadni model
* Evolucijski razvoj
* Razvoj baziran na komponentama

**Iteracije procesa**

* Inkrementalne isporuke
* Spiralni razvoj

**Hibridni (RUP)**

## \*GENERIČKI MODELI SOFTVERSKIH PROCESA

**Vodopadni model -** Strogo su odvojene faze specifikacije i razvoja .

**Evolucijski razvoj -** Specifikacija, razvoj i validacija se preklapaju.

**Programsko inženjerstvo bazirano na komponentama -** Sustav se sklapa od postojećih komponenti.

Postoje mnoge varijante ovih modela:

Formalni razvoj gdje se koriste vodopadni procesi, ali formalna specifikacija se stvara kroz nekoliko faza dok se ne postigne zadovoljavajuća verzija.



### VODOPADNI MODEL

##### FAZE VODOPADNOG MODELA

* Analiza i definiranje zahtjeva
* Dizajn sustava i softvera
* Implementacija i jedinično testiranje
* Integracija i testiranje sustava
* Rad i održavanje

##### PROBLEMI VODOPADNOG MODELA

* Nefleksibilna podjela projekta u zasebne faze otežava reakciju na promjene korisničkih zahtjeva.
* Jedna faza treba biti u potpunosti gotova prije nego se krene na iduću.
* Zbog toga je prikladan samo kada su zahtjevi jasni a promjene ograničene tijekom dizajna.
* Jako malo poslovnih procesa ima tako stabilne zahtjeve.
* Vodopadni model se koristi kod razvoja:
* velikih sustava gdje jako puno ljudi radi na projektu i oni su fizički dislocirani;
* medicinske opreme, …

### EVOLUCIJSKI RAZVOJ

Cilj je što bolje razumjeti zahtjeve korisnika čemu se pristupa na načina:

* ***Istraživački razvoj***(eng. exploratory development) – kreće se s zahtjevima koji su jasni i njih se implementira i tek se onda dodaju novi elementi koje predlaže korisnik. Cilj je u radu s korisnikom jednostavnu početnu specifikacijurazvijati dok se ne realizira konačan sustav.



* ***Odbacivanje prototpa***(eng. throw-away

prototiping) – Kreće se od zahtjeva koji se

najlošije razumiju kako bi se pojasnilo što

je stvarno potrebno. Cilj je što bolje razumjeti

zahtjeve.

Nedostatci:

* Loša vidljivost procesa unutar projekta, pa je nemoguće procijeniti koliko je posla napravljeno i kada će sustav biti gotov.
* Loša struktura konačnog sustava, pa je problematičan za održavanje.
* Zahtjeva posebna znanja i vještine (npr. jezici za brzu izradu prototipova).

Koristi se kod izrade softverskih sustava za:

* male ili srednje velike interaktivne sustave
* dijelove velikih sustava (npr. korisnička sučelja)
* sustave s kratkim životnim vijekom.

## SOFTVERSKO INŽ. BAZIRANO NA KOMPONENTAMA

Bazira se na sistematskom ponovnom korištenju komponenti u slučajevima gdje su sustavi integrirani od postojećih komponenti ili COTS sustava.

Kod razvoja ovakvih sustava postoje četiri osnovne faze:

* Analiza komponenti (potraga za komponentama)
* Modifikacija zahtjeva (kako bi se uskladili s komponentama)
* Dizajn sustava uz korištenje komponenti
* Razvoj i integracija

Ovaj pristup se počeo sve više koristiti s pojavom standarda za izradu komponenti.

###### RAZVOJ BAZIRAN NA PONOVNOJ UPOTREBI



**Prednost** – reducira se vrijeme potrebno za izradu softvera jer se ne troši toliko vremena na razvoj.

**Nedostatci**:

* Kompromisi na zahtjevima mogu uzrokovati da funkcionalnosti sustava ne odgovaraju kupcu.
* Nove verzije komponenti ne ovise o onome ko ih koristi.

## \*ITERACIJE PROCESA

Zahtjevi sustava UVIJEK evoluiraju tijekom projekta, pa iteracije u kojima se ponovo prolazi kroz ranije faze procesa je dio izrade velikog sustava.

Iteracije se mogu primijeniti na bilo koji od generičkih procesnih modela

Postoje dva pristupa:

* Inkrementalne isporuke
* Spiralni razvoj

### INKREMENTALNE ISPORUKE

* Razvoj i isporuka sustava su podijeljeni u niz inkrementa od kojih svaki isporučuje neku od zadanih funkcionalnosti.
* Svi zahtjevi dobivaju prioritet i oni s najvećim se uključuju u početne inkremente.
* Jednom kada se krene u razvoj inkrementa ne smiju s mijenjati njegovi zahtjevi, ali se to može raditi za slijedeće inkremente.



##### INKREMENTALNI

##### RAZVOJ

##### PREDNOSTI INKREMENTALNOG RAZVOJA

* Isporuka pojedinačnih funkcionalnosti kroz inkremente je od velike poslovne važnosti klijentu, jer one funkcionalnosti koje su bitne dobiva ranije.
* Početni inkrementi služe kao prototip koji iznosi na vidjelo zahtjeve za kasnije inkremente.
* Umanjuje se rizik od potpunog promašaja projekta.
* Zahtjevi s najvišim prioritetom se najviše testiraju.
* Nedostatak - Inkrementi bi trebali biti mali (<20 000 linija koda), a nekada je nemoguće mapirati zahtjeve u tako male inkremente.

##### EKSTREMNO PROGRAMIRANJE

Pristup razvoja softvera koji se bazira na razvoju i isporuci vrlo malih inkrementa funkcionalnosti.

Bazira se na konstantnim poboljšanjima koda, uključuje korisnika u razvojni tim i programiranje u paru.

### SPIRALNI RAZVOJ

Softverski procesi su prikazani spiralom, a ne kao niz aktivnosti s povratnim vezama.

Svaka petlja u spirali predstavlja jednu fazu procesa

Ne postoje fiksne faze kao što su specifikacija ili dizajn – petlje se odabiru ovisno što je potrebno.

* Rizici su utvrđeni eksplicitno i riješeni u tijeku procesa.

##### SPIRALNI MODEL SOFTVERSKOG PROCESA



###### SEKTORI SPIRALNOG RAZVOJA

* **Postavljanje ciljeva -** Identificiraju se ciljevi za pojedinačne faze
* **Utvrđivanje i umanjivanje rizika -** Utvrđuju se rizici i pokreću aktivnosti koje umanjuju ključne rizike.
* **Razvoj i validacija -** Odabire se jedan od razvojnih model za sustav (može biti jedan od generičkih)
* **Planiranje -** Projekt se revidira i planira se slijedeća faza spirale.

## \*RATIONAL UNIFIED PROCES (RUP)-HIBRIDNI

Moderni procesni model koji je nastao kao rezultat rada na UML-u i sličnim projektima.

Razvio ga je IBM / Rational

* Booch, Rumbaugh, Jaccobson

Prikazuje se najčešće iz tri perspektive:

* Dinamička perspektiva – vremenski raspored faza
* Statička perspektiva – prikazuje procesne aktivnosti
* Perspektiva iz prakse – sugerira što je dobro raditi.

##### FAZE RUP-A

* **Početak (eng. inception) -** Uspostavlja poslovni okvir za sustav
* **Razrada (eng. elaboration) -** Povećava se razumijevanje problema i sistemske arhitekture (UML use cases)
* **Izrada (eng. construction) -** Dizajn sustava, programiranje i testiranje
* **Tranzicija -** Primjena sustava u radnoj okolini

****

##### MODEL RUP FAZA

###### itermodelGRAF ITERATIVNOG MODELA

* Horizontalna os predstavlja vrijeme, prikazuje dinamičke aspekte procesa od kada su određeni, u obliku ciklusa, faza iteracija i rokova.
* Vertikalna os predstavlja statičke aspekte procesa: kako je opisan u obliku aktivnosti, rukotvorina (eng. artifacts), radnika i radnog slijeda.

###### STATIČKI RADNI SLIJED

|  |  |
| --- | --- |
| **Radni slijed** | **Opis** |
| **Modeliranje poslovanja** | Poslovni procesi se modeliraju korištenjem poslovnih use case-ova |
| **Zahtjevi** | Identificiraju se sudionici (eng. actor) koji su u interakciji sa sustavom i razvijaju se use case-ovi predstavljaju zahtjeve. |
| **Analiza i dizajn** | Stvara se i dokumentira model dizajna korištenjem modela arhitekture, modela komponenti, objektnog modela i slijednih modela. |
| **Implementacija** | Komponente sustava strukturiraju u implementacijske podsustave i implementiraju se. Korištenje automatskog generiranja koda može ubrzati ovaj proces. |
| **Testiranje** | Iterativan proces koji se izvodi zajedno s implementacijom, točnije s krajem implementacije. |

|  |  |
| --- | --- |
| **Radni slijed** | Opis |
| **Primjena** | (eng. deploymen) Stvara se verzija za puštane u promet, distribuira korisnicima i instalira na radnim mjestima. |
| **Upravljanje konfiguracijom i promjenama** | Upravlja naknadnim promjenama u sustavu i detaljnije je objašnjena u poglavlju 29. |
| **Upravljanje projektom** | Ovaj radni slijed podržava upravljane projektom što je detaljnije objašnjeno u poglavlju 5. |
| **Okolina** | Brine se o izradi prikladnih softverskih alata koji bi olakšali posao razvojnom timu. |

##### PREDNOSTI RUP-A

* Iterativan razvoj softvera
* Upravljanje zahtjevima (korištenje scenarija i use case-ova)
* Koristi se arhitektura bazirana na komponentama
* Softver za vizualno modeliranje (UML)
* Provjera kvalitete softvera
* Kontrola promjena u softveru
* opisuje se kako kontrolirati, pratiti i nadgledati promjene kako bi se omogućio uspješan iterativan razvoj.

## RAČUNALIMA POTPOMOGNUTO PROGRAMSKO INŽENJERSTVO

CASE (Computer Aided Softvare Engineering) je softver koji podržava procese razvoja i evolucije softvera.

* MS Visio, Oracle Designer, Rational Rose , ….

Automatizacija aktivnosti

* grafički editori za razvoj modela sustava
* katalozi podataka za upravljanje razvojnim entitetima
* grafički alati za izradu korisničkih sučelja
* programi koji olakšavaju pronalaženje pogrešaka (eng. debugger)
* automatski prevodioci koji programe iz starijih verzija programskog jezika prebacuju u nove .

## CASE TEHNOLOGIJA

Korištenje CASE tehnologije je uvelike doprinijelo poboljšanjima softverskih procesa, ali ne u onoj mjeri kako se predviđalo:

* Programsko inženjerstvo zahtjeva kreativnost – nije jednostavno za automatizaciju.
* Programsko inženjerstvo je timska aktivnost i kod velikih projekta veliki dio vremena se troši u interakcijama timova, što CASE ne podržava.

# SPECIFIKACIJA SOFTVERSKIH ZAHTJEVA

## SPECIFIKACIJA ZAHTJEVA

* Proces utvrđivanja (pronalaženja, analize, dokumentiranja, provjere) usluga koje korisnik zahtjeva od sustava te ograničenja pod kojima taj sustav mora raditi se naziva **specifikacija zahtjeva**.

## ŠTO JE ZAHTJEV?

Zahtjevi predstavljaju opis usluga koje pruža sustav i ograničenja u radu sustava.

Problem je što se pojam zahtjev u softverskoj industriji ne koristi na konzistentan način, pa može biti:

* vrlo apstraktna definicija osnovnih funkcionalnosti
* detaljna matematička specifikacija ili detaljan opis svih funkcionalnosti.

## VRSTE ZAHTJEVA

Moguće je zahtjeve podijeliti u dvije razine apstrakcije:

* ***Korisnički zahtjevi*** – zahtjevi opisani na način da ih razumije krajini korisnik (korištenjem opisa i dijagrama usluga koje pruža taj sustav) – pišu se za korisnika.
* **Zahtjevi sustava** – strukturirani dokument s detaljnim opisima funkcija, usluga i ograničenja sustava. Definira što će se implementirati



pa može se koristiti za potpisivanje

ugovora – namijenjen je razvojnom timu.

TKO ČITA ZAHTJEVE? 🡪

## LIBSYS SUSTAV

* Bibliotečni sustav koji pruža jedinstveno sučelje prema brojnim bazama časopisa u različitim bibliotekama.
* Korisnici mogu pretraživati, preuzimati i ispisivati te članke za osobnu upotrebu.

## LIBSYS SUSTAV - PRIMJER ZAHTJEVA

Korisnički zahtjevi:

1. LIBSYS bi trebao čuvati informacije potrebne agencijama za zaštitu autorskih prava.

Zahtjevi sustava:

* 1. Kad se pošalje zahtjev za dokument iz LIBSYS-a, potrebno je zapamtiti podatke o korisniku koji je poslao zahtjev i detalje samog zahtjeva.
  2. Sve to je potrebno čuvati 5 godina od datuma kada je zahtjev poslan.
  3. Svi LIBSYS zahtjevi moraju biti indeksirani po korisniku, naslovu dokumenta te vlasniku dokumenta.
  4. LIBSYS bi trebao čuvati zapise o svim zahtjevima koji su ikad poslani.

## FUNKCIONALNI I NE-FUNKCIONALNI ZAHTJEVI

Tri osnovne grupe zahtjeva za softverske sustave:

* ***Funkcionalni zahtjevi*** - Popis usluga koje pruža sustav, kako sustav treba reagirati na određene ulaze, te kako se sustav ponaša u određenoj situaciji, u nekim slučajevima u funkcionalnim zahtjevima može pisati i što sustav ne bi trebao raditi.
* ***Ne-funkcionalni zahtjevi*** - Ograničenja na usluge ili funkcionalnosti sustava (vremenska ograničenja, ograničenja razvojnih procesa, standarda, performanse, …)
* ***Zahtjevi domene*** - Zahtjevi koje nameče aplikacijska domena sustava, a mogu biti funkcionalni i ne-funkcionalni.

## FUNKCIONALNI ZAHTJEVI

* Opisuju funkcionalnosti ili usluge sustava.
* Mogu biti vrlo apstraktne izjave koje opisuju što sustav treba raditi ili u detalje opisivati usluge koje će pružiti taj sustav ovisno o tome kome su namijenjeni.
* Ovise o vrsti softvera, korisnicima i tipu sustava na kojem se softver koristi.

## LIBSYS SUSTAV – FUNKCIONALNI ZAHTJEVI

1. Korisnik bi trebao biti u mogućnosti pretraživati unaprijed zadan niz baza podataka ili odabrati podniz od interesa.
2. Sustav treba osigurati odgovarajuće preglednike za čitanje svih dostupnih dokumenata.
3. Svakoj narudžbi će biti dodijeljen jedinstveni broj (id\_narudžbe) kojeg će korisnik moči kopirati u prostor za trajnu pohranu na svom računu.

## NEPRECIZNOST ZAHTJEVA

Neprecizno utvrđeni zahtjevi uzrokuju probleme - korisnici i razvojni tim mogu na različite načine interpretirati nejasne zahtjeve.

U LIBSYS sustavu na koji način se može tumačiti zahtjev za "odgovarajućim preglednikom“:

* Namjere korisnika – poseban preglednik za sve vrste dostupnih dokumenata.
* Interpretacija razvojnog tima – Ponudit preglednik teksta koji prikazuje sadržaj različitih vrsta dokumenta.

## KONZISTENTNOST I POTPUNOST ZAHTJEVA

U principu zahtjevi bi trebali biti:

* **Potpuni** – Trebaju uključiti opis svih zahtijeva koje su naveli korisnici.
* **Konzistentni** – Ne smije biti konflikta ili kontradikcija kod opisa stava.

U praksi je nemoguće stvoriti potpun i konzistentan dokument zahtjeva zbog:

* grešaka i propusta koji se događaju kod pisanja dokumenta,
* različiti ljudi unutar firme daju kontradiktorne informacije.

## NE-FUNKCIONALNI ZAHTJEVI

To nisu zahtjevi koji su direktno povezani s funkcionalnošću sustava, oni se odnose na:

* svojstva i ograničenja sustava npr. pouzdanost, brzina izvršavanja, zahtjevi za pohranom, …
* ograničenja U/I uređaja ili prezentacije podataka na sučeljima u sustavu, …
* korištenje određenog CASE sustava, programskog jezika ili razvojne metode.

Ne-funkcionalni zahtjevi mogu nekada biti važniji za prihvaćanje sustava od funkcionalnih. Ukoliko se ne ostvare sustav je beskoristan.

## KLASIFIKACIJA NE-FUNKCIONALNIH ZAHTJEVA

* **Zahtjevi aplikacije** - da se aplikacija ponaša na određeni način (npr. brzina izvođenja, pouzdanost, …)
* **Organizacijski zahtjevi** - posljedica su organizacijske politike i procedura (npr. procesi koji se standardno koriste, implementacijski zahtjevi, …)
* **Vanjski zahtjevi** – javljaju se kao posljedica faktora van samog sustava (npr. zakonski okviri, …)

VRSTE NE-FUNKCIONALNIH



ZAHTJEVA

## LIBSYS SUSTAV - NE-FUNKCIONALNI ZAHTJEVI

* ***Zahtjevi na aplikaciju -*** Korisničko sučelje LIBSYS-a treba biti implementirano kao jednostavni html bez korištenja okvira ili java apleta.
* ***Organizacijski zahtjevi -*** Proces razvoja sustava i dokumenti koji će se isporučiti treba prilagoditi procesima i isporukama definiranim u XYZCo-SP-STAN-95 standardu.
* ***Vanjski zahtjevi -*** Sustav operatoru smije prikazati jedino ime i broj reference klijenta, dok su sve ostale osobne informacije skrivene.

## CILJEVI I ZAHTJEVI

Ne-funkcionalni zahtjevi mogu biti jako komplicirani za jasan opis, a neprecizni zahtjevi su teški za provjeru. Zbog čega se često izražavaju kroz ciljeve sustava

* Cilj opisuje generalnu namjeru korisnika (npr. jednostavnost korištenja).

Dokazivanje ne-funkcionalnih zahtjeva

* Izjava koja koristi neku mjeru koja se može koristiti za objektivnu provjeru cilja (npr. prosječni korisnik nakon jednog sata treninga ne smije proizvoditi više od 5 grešaka dnevno).

Ciljevi su korisni razvojnom timu jer oni opisuju namjere korisnika.

## MJERENJE ZAHTJEVA

|  |  |
| --- | --- |
| **Svojstvo** | **Mjera** |
| BRZINA | obavljene transakcije/s, korisnik/vrijeme odziva, vrijeme osvježavanja ekrana |
| VELIČINA | MBajt-ima, broj ROM čipova |
| JEDNOSTAVNOST KORIŠTENJA | Vrijeme potrebno za trening, broj stranica za pomoć |
| POUZDANOST | Prosječno vrijeme rada bez greške, vjerojatnost da su usluge nedostupne, brzina događanja grešaka, dostupnost |
| ROBUSTNOST | Vrijeme potrebno da se sustav ponovno podigne nakon pada  Postotak grešaka koje uzrokuju pad sustava  Vjerojatnost grešaka na podacima u slučaju pada sustava |
| PORTABILNOST | Broj ciljanih sustava |

## INTERAKCIJA MEĐU ZAHTJEVIMA

Konflikti među različitim ne-funkcionalnim zahtjevima.

Kod zrakoplova:

* Potrebno je što više smanjiti težinu – broj odvojeni čipova u sustavu treba minimizirati
* Umanjiti potrošnju energije – koristiti čipove sa slabijom potrošnjom
* Korištenje čipova s manjom potrošnjom može značiti veći broj čipova – koji je zahtjev važniji?

## ZAHTJEVI DOMENE

* Izvode se iz domene aplikacije opisuje karakteristike sustava i osobine koje opisuju domenu.
* Zahtjevi domene mogu biti novi funkcionalni zahtjevi, ograničenja na postojeće zahtjeve ili definirati određene proračune.
* Ukoliko nisu zadovoljeni zahtjevi domene moguće je da se sustav neće moči koristiti.

**LIBSYS sustav - primjer zahtjeva domene**

* Prema svim bazama podataka se treba koristiti standardno sučelje koje se bazira na Z39.50 standardu.
* Zbor autorskih prava dokumente je potrebno izbrisati čim stignu. Ovisno o tome što korisnik želi ovi dokumenti se ili ispisuju lokalno na serveru sustava i šalju poštom naručitelju ili preusmjeravaju na mrežni printer.

## PROBLEMI KOD ZAHTJEVA DOMENE

* ***Razumijevanje -*** Zahtjevi su opisani jezikom domene, pa ga razvojni tim ga ne mora razumjeti.
* ***Podrazumijevanje -*** Korisnik za dosta stvari smatra da se podrazumijevaju, pa nema potrebu to posebno navesti ili objasniti.

## KORISNIČKI ZAHTJEVI

* Trebali ni objasniti funkcionalne i ne-funkcionalne zahtjeve na takav način da ih razumiju oni korisnici sustava koji nemaju detaljna tehnička znanja.
* Korisničke zahtjeve treba opisati korištenjem prirodnog jezika, tablica, dijagrama a trebaju biti takvi da ih razumiju svi korisnici.

## ZAHTJEVI SUSTAVA

* Detaljnija specifikacija funkcija i ograničenja sustava od korisničkih zahtjeva.
* Služi kao osnova za dizajn sustava.
* Može biti uključena u ugovor.
* Zahtjevi mogu biti ilustrirani ili definirani korištenjem grafičkih modela objašnjenih u poglavlju 8.

## ZAHTJEVI I DIZAJN

U teoriji:

* **Zahtjevi** – opisuju što bi sustav trebao raditi;
* **Dizajn** – opisuje na koji način će se sustav ostvariti zahtjeve.

U praksi – zahtjevi i dizajn se ne mogu jednostavno odvojiti:

* Arhitektura sustava može biti potrebna za strukturiranje zahtjeva;
* Sustav može surađivati s drugim sustavima koji utječu na njegov dizajn;
* Korištenje određenog dizajna može biti zahtjev domene.

## PROBLEMI S PRIRODNIM JEZIKOM

* ***Nedovoljno jasno -*** Teško je postići potrebnu dozu preciznosti bez da dokument postane težak za čitanje.
* ***Miješanje zahtjeva -*** Nemoguće je odvojiti funkcionalne i ne-funkcionalne zahtjeve.
* ***Integracija zahtjeva -*** Nekoliko različitih zahtjeva su opisani zajedno.

## PROBLEMI OPISA PRIRODNIM JEZIKOM

* ***Dvosmislenost*** - Onaj koji piše i onaj koji čita zahtjeve mora interpretirati iste riječi na isti način. Kako je prirodni jezik sam po sebi dvosmislen to je jako teško postići.
* ***Previše fleksibilan*** - Istu stvar se može reći na brojne načine u specifikaciji.
* ***Nedostatak modularnosti*** - Struktura prirodnog jezika nije prikladna za strukturiranje zahtjeva sustava.

## PRIMJER PRIKAZA MREŽE U EDITORU CASE ALATA

2.6. **Mreža –** Kako bi se pojednostavnilo pozicioniranje entiteta na dijagramu korisnik može uključiti ili isključiti prikaz mreže preko opcije na upravljačkoj traci. Početno je mreža isključena. Mreža se može uključiti i isključiti u bilo kojem trenutku, te se može mijenjati između inča i centimetra.

Opisani zahtjevi miješaju tri različita tipa zahtjeva:

* Konceptualni funkcionalni zahtjevi – potreba za mrežom
* Ne-funkcionalni zahtjevi – kako se uključuje i isključuje mreža
* Ne-funkcionalni U/I zahtjevi – prebacivanje inča i centimetra.

## UPUTE ZA PISANJE ZAHTJEVA

-Postaviti stand. format i koristiti ga za pisanje svih zahtjeva (excel ili word template, use case-ovi).

-Koristiti jezik na konzistentan način. Koristiti:

* **“mora”** za glavne zahtjeve,
* **“trebalo bi / bilo bi dobro”** za zahtjeve koje nisu osnovne funkcionalnosti.

-Korištenje podebljanja za isticanje glavnih dijelova zahtjeva.

-Izbjegavati korištenje računalnog žargona, jer korisnici to ne razumiju.

## ALTERNATIVE SPECIFIKACIJI PRIRODNIM JEZIKOM

|  |  |
| --- | --- |
| **Notacija** | **Opis** |
| Strukturirani prirodni jezik | Pristup ovisi o definiranoj standardnoj formi ili predlošku za prikaz specifikacije zahtjeva. |
| Jezik opisa dizajna | Koristi poseban jezik, sličan programskom jeziku, ali s više apstraktnih osobina kako bi se zahtjevi specificirali korištenjem operacijskog modela sustava. |
| Grafička notacija | Grafički jezik s dodatkom tekstualnih opisa (use case, slijedni dijagrami) |
| Matematička specifikacija | Notacije bazirane na matematičkim osnovama poput nizova ili modela konačnih stanja. Jednoznačna specifikacija koja raspravu svodi na najmanju mjeru. Većina korisnika ne razumije ovakav model i ne žele ga prihvatiti kao osnovu za ugovor. |

## SPECIFIKACIJA STRUKTURIRANIM JEZIKOM

* Sloboda onoga koji piše zahtjeve je limitirana predefiniranim predloškom.
* Svi zahtjevi se pišu na standardan način.
* Može biti ograničena terminologija koja se koristi za opis.
* Prednost je što se zadržava izražajnost prirodnog jezika, ali je nametnut stupanj uniformnosti.

### SPECIFIKACIJA ČVORA BAZIRANA NA FORMI GRAFIČKA SPECIFIKACIJA





### C:\Users\Vana\Desktop\spec.JPGTABLIČNA SPECIFIKACIJA

## SPECIFIKACIJA SUČELJA

Većina sustava u radu treba surađivati s nekim drugim sustavima zbog čega je neophodno u sklopu specifikacije zahtjeva definirati i njihova sučelja.

Može biti potrebno definirati tri tipa sučelja:

* ***Sučelja procedura***– postojeći programi ili podsustavi nude niz usluga kojima se pristupa pozivom procedura sučelja (ponekad se ta sučelja nazivaju eng. Application Programing Interfaces - APIs).
* ***Struktura podataka koji se prosljeđuju između podsustava***– najbolja je grafička notacija.
* ***Prezentacija podataka*** *–* npr. raspored grupa bitova i njihova uloga u određenom podsustavu.

Formalne metode su efektivna tehnika za specifikaciju sučelja, ali se rijetko koriste.

## DOKUMENT SPECIFIKACIJE ZAHTJEVA

Na eng. Softwer Requirements Specification – SRS.

Dokument specifikacije zahtjeva je službena izjava što se traži od razvojnog tima, a treba uključiti:

* korisniče zahtjeve,
* zahtjeve sustava.

**NIJE** dokument dizajna.

Koliko je god moguće fokus treba postaviti na ono ***ŠTO***sustav radi a ne ***KAKO***.

## KORISNICI SPECIFIKACIJE ZAHTJEVA

|  |  |
| --- | --- |
| **KORISNICI** | **OPIS** |
| KRAJNJI KORISNICI SUSTAVA | Navode zahtjeve i čitaju specifikaciju kako bi provjerili odgovara li njihovim zahtjevima. Predlažu promjene zahtjeva. |
| MENADŽERI | Koriste specifikaciju zahtjeva za planiranje cijene sustava i planiranje razvojnog procesa. |
| INŽENJERI SUSTAVA | Koriste specifikaciju zahtjeva kako bi shvatili kakav sustav treba razviti. |
| TESTERI | Koriste specifikaciju zahtjeva kao osnovu testova validacije. |
| ODRŽAVANJE | Koriste specifikaciju zahtjeva kako bi što lakše razumjeli sustav i veze među njegovim dijelovima. |

## IEEE STANDARD ZA SPECIFIKACIJU ZAHTJEVA

Definira generičku strukturu dokumenta zahtjeva koja se treba koristiti za svaki sustav:

* Uvod
* Osnovni opis
* Specifični zahtjevi
* Dodaci
* Indeks

## SOMMERVILLE-OV PRIJEDLOG NADOPUNE IEEE-A

* Predgovor
* Uvod
* Rječnik pojmova
* Definicija korisničkih zahtjeva
* Arhitektura sustava
* Specifikacija zahtjeva sustava
* Modeli sustava
* Planovi za budući razvoj
* Dodaci
* Indeks

# PROCES PRIKUPLJANJA ZAHTJEVA

## PROCES PRIKUPLJANJA ZAHTJEVA

Procesi koji se koriste kod prikupljanja zahtjeva mogu biti vrlo različiti ovisno o domeni aplikacije, ljudima koji su uključeni u proces te organizaciji koja prikuplja zahtjeve.

Ipak postoji niz osnovnih aktivnosti koje su zajedničke svim procesima:

* opisivanje zahtjeva
* analiza zahtjeva
* validacija zahtjeva
* upravljanje zahtjevima



## STUDIJA ISPLATIVOSTI

Vrlo kratka studija koja provjerava:

* Doprinosi li sustav organizacijskim ciljevima,
* Može li se realizirati korištenjem postojeće tehnologije i okviru proračuna,
* Može li se integrirati s postojećim sustavima?

Na osnovu studije isplativosti se donosi odluka ima li smisla krenuti u realizaciju projekta.

Neka od pitanja na koje je potrebno odgovoriti kod izrade studije isplativosti:

* Što ako se sustav ne implementira?
* S koji se procesnim problemima trenutno susrećemo u poslovanju?
* Na koji način će predloženi sustav pomoći?
* Koji će biti problemi u integraciji novog i postojećih sustava?
* Jesu li potrebne nove tehnologije? Nove vještine?
* …

## OPIS I ANALIZA (OTKRIVANJE) ZAHTJEVA

* Uključuje rad tehničkog osoblja s korisnicima kako bi prikupili što više znanja o domeni aplikacije, uslugama koje bi trebao pružiti sustav i ograničenjima u radu sustava.
* Može uključivati: krajnje korisnike, menadžere, inženjere uključene u održavanje, eksperte iz te domene, sindikate, …
* Svi oni se nazivaju **zainteresirane strane (eng. stakeholders)**

## PROBLEMI KOD ANALIZE ZAHTJEVA



* Zainteresirane strane ne znaju što stvarno žele!!!!!
* Zahtjeve opisuju korištenjem vlastite terminologije.
* Različiti sudionici imaju različite/konfliktne zahtjeve.
* Organizacijski i politički faktori mogu utjecati na

zahtjeve.

* Zahtjevi se mijenjaju tijekom procesa analize.

Mogu se pojaviti nove zainteresirane strane

i promijeniti poslovna okolina.

SPIRALA ZAHTJEVA

## PROCESNE AKTIVNOSTI

* ***Otkrivanje zahtjeva*** - U suradnji s zainteresiranim stranama (otkriva i zahtjeve domene).
* ***Klasifikacija i organizacija zahtjeva*** - Grupiraju se povezani zahtjevi i organiziraju u koherentne skupine.
* ***Postavljanje prioriteta i pregovaranje*** - Postavljanje prioriteta (**mora imati, trebao bi imat, bilo bi dobro**), te rješavanje konfliktnih zahtjeva
* ***Dokumentiranje zahtjeva*** - Svi zahtjevi se dokumentiraju i služe kao ulazni podaci za slijedeću fazu spirale.

## OTKRIVANJE ZAHTJEVA

Proces prikupljanja informacija o sustavu koji će se razvijati kao i o postojećim sustavima, te na osnovu tih informacija se izvode korisnički zahtjevi i zahtjevi sustava.

Izvori informacija mogu biti:

* Dokumentacija, zainteresirane strane, specifikacije sličnih sustava.

Načini otkrivanja zahtjeva:

* Pogledi na sustav (eng. viewpoints)
* Razgovori (intervjui)
* Scenariji
* Use-case dijagrami
* Etnografija

## POGLEDI NA SUSTAV

Koriste se za klasifikaciju zainteresiranih strana i ostalih izvora koji utječu na zahtjeve.

Postoje tri osnovna tipa pogleda na sustav:

* Pogled osobe ili drugog sustava koji su u direktnoj interakciji sa sustavom (npr. bankomat – korisnik i baza računa ) – **pogled interakcije**.
* Pogled onih zainteresiranih strana koji sami po sebi ne koriste sustav, ali utječu na njegove zahtjeve (npr. bankomat – menadžment i inženjeri odgovorni za zaštitu podataka) – **indirektni pogled**.
* Karakteristike i ograničenja domene koja utječu na zahtjeve (npr. bankomat – standard za međubankovnu komunikaciju) – **pogled domene**.

Svaki od pogleda daje različitu vrstu zahtjeva.



PRIMJER POGLEDA

NA SUSTAV - LIBSYS

## RAZGOVORI

Bilo da se radi o formalnom ili neformalnom razgovoru tim za prikupljanje zahtjeva postavlja pitanja svim zainteresiranim stranama o sustavu koji trenutno koriste i sustavu koji treba razviti.

Postoje dva osnovna tipa razgovora:

* **Zatvoreni razgovori** – odgovara se na predefinirani niz pitanja (koristi se u slučaju kad se već zna o čemu se radi jer je ranije rađena slična aplikacija)
* **Otvoreni razgovori** – ne postoje pre-definirana pitanja, već se u razgovoru s zainteresiranim stranama istražuje problem.

## RAZGOVORI U PRAKSI

Kombinacija otvorenog i zatvorenog razgovora.

Dobri u za razumijevanje toga što rade zainteresirane strane i kakva bi bila njihova interakcija sa sustavom.

Loši su za razumijevanje zahtjeva domene:

* Tim za prikupljanje zahtjeva ne može razumjeti terminologiju specifičnu za tu domenu.
* Neke stvari su ljudima iz te domene normalne i ne smatraju da to treba napomenuti.

## SCENARIJI

Scenariji su primjeri iz stvarnog korištenja sustava koji opisuju kako se sustav može koristiti.

Trebali bi uključiti:

* Opis početne situacije
* Opis normalnog tijeka događaja
* Opis što može krenuti loše
* Informacije o drugim istovremenim aktivnostima
* Opis stanja sustava kad scenarij završi

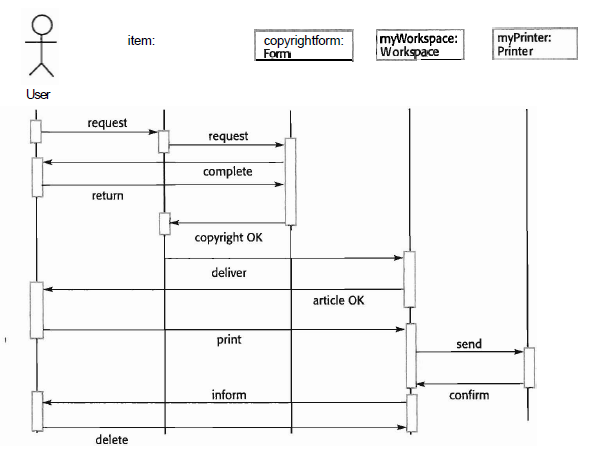
## LIBSYS SCENARIJ



## USE-CASE-OVI

* UML tehnika bazirana na scenarijima koja identificira aktore/sudionike koji su u interakciji sa sustavom.
* Niz use-case-ova bi trebao opisati sve moguće interakcije sa sustavom.
* Uz use-case može se koristiti i slijedni dijagram kako bi se opisao vremenski slijed obrade događaja u sustavu.

*Primjer Use-Case-A – Libsys*  *Libsys Slijedni Dijagram Za Ispis Članka*



## ETNOGRAFIJA

Etnografija je promatračka tehnika koja se koristi za bolje razumijevanje socijalnih i organizacijskih zahtjeva.

* Analitičar postaje dio radne okoline u kojoj će se koristiti sustav – promatra ljude na njihovom radnom mjestu.

Prednosti:

* Pomaže analitičaru otkriti implicitne zahtjeve sustava koji predstavljaju stvarne a ne formalne procese.
* Ljudi ne moraju objasniti što rade.
* Promatraju se važni socijalni i organizacijski faktori.

Nedostatak: daje prikaz radnih procesa u trenutku promatranja.

## VALIDACIJA ZAHTJEVA

Provjerava da zahtjevi definiraju sustav koji korisnik zbilja želi.

Greška u specifikaciji je jako skupa, pa je validacija jako važna.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cijena otkrivanja pogreške** | | **Vrijeme otkrivanja pogreške** | | | | |
| **Specifikacija**  **zahtjeva** | **Specifikacija dizajna** | **Implementacija** | **Testiranje** | **Puštanje u rad** |
| **Vrijeme uvođenja greške** | **Specifikacija zahtjeva** | 1x | 3x | 5-10x | 10x | **10-100x** |
| **Specifikacija dizajna** | - | 1x | 10x | 15x | 25-100x |
| **Implementacija** | - | - | 1x | 10x | 10-25x |

Za vrijeme procesa validacije zahtjeva provode se različite provjere:

* **Valjanost** – pruža li sustav funkcionalnosti koje na najbolji način podržavaju korisničke potrebe
* **Konzistentnost** – Jesu li neki zahtjevi konfliktni?
* **Potpunost** – Jesu li sve funkcije koje zahtjeva korisnik uključene? Jesu li zahtjevi potpuni?
* **Realnost** – Jesu li zahtjevi realni - mogu se ostvariti zadanom tehnologijom i s dostupnim budžetom?
* **Provjerljivost** – Mogu li se zahtjevi provjeriti/testirati?

## TEHNIKE VALIDACIJE

Postoje brojne tehnike validacije koje se mogu koristiti pojedinačno ili u kombinaciji:

* **Kontrola zahtjeva** – Kontrolni tim sistematično pregledava zahtjeve.
* **Prototipiranje** – Korištenje prototipova sustava kako bi se provjerili zahtjevi.
* **Stvaranje test case-ova** – Prave se testovi zahtjeva da se vidi je li zadani ulaz daje očekivani izlaz.

## UPRAVLJANJE ZAHTJEVIMA

Proces upravljanje promjenama zahtjeva za vrijeme procesa prikupljanja zahtjeva, te za vrijeme razvoja sustava.

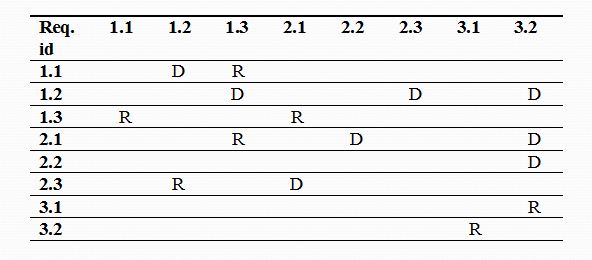
Zahtjevi su uvijek nepotpuni i nekonzistentni:

* Novi zahtjevi se javljaju kao rezultat poslovnih promjena ili kao posljedica boljeg razumijevanja sustava koji se razvija.
* Promjene prioriteta zahtjeva za vrijeme razvojnih procesa.
* Naručitelji softvera mogu specificirati zahtjev iz poslovne perspektive koji je u konfliktu s zahtjevima krajnjih korisnika.
* Promjene u poslovnoj i tehničkoj okolini za vrijeme razvoja (nova verzija softvera, novi zakoni, uprava, ..).

## PLAN UPRAVLJANJA ZAHTJEVIMA

Za vrijeme trajanja procesa prikupljanja zahtjeva potrebno je planirati:

* Identifikaciju zahtjeva – npr. identifikacijski broj zahtjeva.
* Proces upravljanja promjenom zahtjeva – na koji način će se pratiti proces promjene zahtjeva.
* Politika praćenja veza među zahtjevima – na koji način čuvati informacije o vezama između pojedinih zahtjeva.
* CASE alat koji će se koristiti za upravljanje promjenama zahtjeva



*Matrica za praćenje veza među zaht.*

* D – jaka veza
* R – slaba veza

## UPRAVLJANJE PROMJENAMA ZAHTJEVA

Isti predložak se treba primijeniti na sve promjene zahtjeva.

Osnovne faze su:

* **Analiza problema** – raspraviti u čemu je problem zahtjeva i predložiti izmjenu;
* **Analiza promjene i njen trošak** – procijeniti efekt promjene na druge zahtjeve
* **Promjena implementacije** – promijeniti specifikaciju zahtjeva i druge dokumente u skladu s promjenom.



# MODELI SUSTAVA

## MODELIRANJE SUSTAVA

Modeliranje sustava pomaže analitičarima da bolje shvate funkcionalnosti sustava i modeli se koriste za komunikaciju s korisnicima.

Različiti modeli predstavljaju sustav iz različitih perspektiva:

* Vanjska perspektiva prikazuje kontekst sustava (njegovu okolinu) – **kontekstni model**.
* Ponašanje sustava prikazuje reakcije sustava na vanjske utjecaje –**modeli ponašanja**
* Strukturna perspektiva – prikazuje arhitekturu podataka i sustava – **semantički modeli sustava.**

## KONTEKSTNI MODEL SUSTAVA

Koristi se za prikaz radne okoline sustava.

Model arhitekture koji prikazuje sustav i njegove veze prema drugim sustavima .

Radna okolina bankomata: 🡪

## PROCESNI MODELI

Korištenjem jednostavnih modela arhitekture opisuje procese koje podržava sustav.

Mogu se koristiti i modeli toka podataka kako bi se prokazali procesi zajedno s tokom podataka od jednog procesa prema drugom

PROCES NARUDŽBE OPREME

* **ovali** – procesi
* **pravokutnici** – dokumentacija ili baze podataka
* iscrtani dio prikazuje čime se bavi aplikacija

## MODELI PONAŠANJA

Koriste se za opis ponašanja cijelog sustava.

Postoje dva osnovna tipa ovih modela:

* **Model obrade podataka** – prikazuju što se događa s podacima u sustavu, tj. promjene koje se događaju na podacima kako se kreću kroz sustav.
* **Model promjene stanja** (eng. state machine ili state transition) – prikazuje reakcije sustava na događaje.

S obzirom da ovi sustavi prikazuju različite perspektive najčešće je potrebno koristiti oba za opis ponašanja sustava.

## MODELI OBRADE PODATAKA

Spadaju u grupu dijagrama toka koji se koriste za prikaz obrade podataka u sustavu.

Prikazuju korake od početka do kraja i to korak po korak što se događa s podacima dok se kreću kroz sustav.

Jednostavan i intuitivan prikaz kojeg korisnici lako razumiju.

Važan dio mnogih metoda analize.

*Primjer Modela Obrade Podataka – Proces Narudžbe*

## MODELI PROMJENE STANJA

Ovi modeli prikazuju reakcije sustava na vanjske ili unutarnje podražaje (događaji, akcije).

Kako prikazuju reakciju koriste se za prikaz sustava koji se izvode u realnom vremenu.

Karakteristike modela:

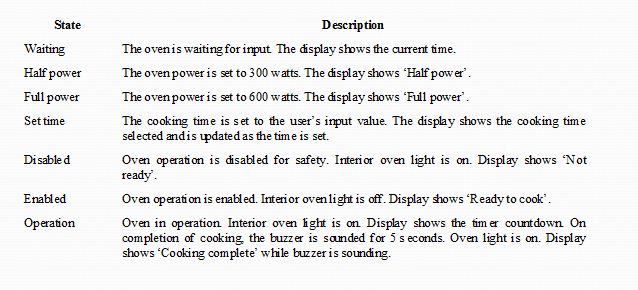
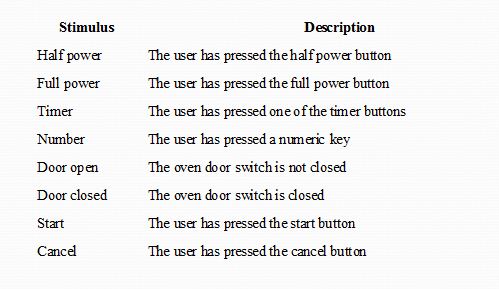
* Stanja sustava su prikazana kao čvorovi.
* Događaji su veze među čvorovima.
* Ključna riječ "DO" daje kratki opis akcija u svakom stanju, "ENTRY" ulazne akcije, "EXIT" izlazne akcije.

Dozvoljavaju dekompoziciju modela u pod-sustave.

Često uz ove dijagrame idu prateće tablice koje opisuju stanja i podražaje.

PRIMJER MODELA PROMJENE STANJA – MODEL MIKROVALNE PEĆNICE



Opis stanja Opis podražaja

PRIMJER MODELA PROMJENE STANJA – MODEL MIKROVALNE PEĆNICE

## SEMANTIČKI MODEL PODATAKA

Opisuje logičku strukturu podataka koji se koriste u sustavu.

Jedan od najkorištenijih modela je dijagram entiteta –veza (eng. entity-relationship).

* Entiteti je prikaz nekog elementa sustava, a sadrži atribute.
* Veze prikazuju na koji način su entiteti međusobno vezani.
* Koristi se za opis baza podataka i jednostavno se implementiraju kod relacijskih baza podataka.

*Primjer Modela Podataka – Libsys Biblioteka 🡪*

## OBJEKTNI MODELI

Opisuju sustav u obliku klasa i veza među klasama.

Klasa je apstraktni prikaz niza objekata koji imaju zajedničke atribute i pružaju iste usluge (operacije).

Prikaz klase u UML-u je postao standard za objektno orijentirano modeliranje:

* Objekti klasa su pravokutnici s imenom na vrhu, atributima u srednjem dijelu i operacijama u donjem dijelu.
* Veze između klasa se prikazuju linijama koje ih povezuju.
* Nasljeđivanje prikazuje generalizaciju i pokazuje prema super-klasama.

## NASLJEĐIVANJE

* Omogućava organizaciju klasa u hijerarhijski model.
* Klase na vrhu prikazuju zajednička svojstva svih nižih klasa.
* Objekti klase nasljeđuju atribute i usluge od jedne ili više super-klasa.
* Višestruko nasljeđivanje – neka klasa može imati više

"roditeljskih" klasa

PRIMJER OBJEKTNIH MODELA – LIBSYS



## ZDRUŽIVANJE OBJEKATA

Model združivanja objekata prikazuje da određeni objekti mogu sadržavati više grupa drugih objekata.



## MODELI PONAŠANJA OBJEKATA

Modeli ponašanja prikazuje niz interakcija među objektima koje su potrebne kako bi se prikazalo određeno ponašanje sustava koje je prikazano kroz use-case.

U UML se za to koriste slijedni dijagrami.

## STRUKTURIRANE METODE

* Strukturirane metode podrazumijevaju korištenje modela sustava.
* Definiraju niz modela, procesa za izvođenje metoda, pravila i uputa koje bi se trebale primijeniti kod izrade modela.
* CASE alati podržavaju modeliranje sustava kao dio strukturirane metode.

## NEDOSTATCI STRUKTURIRANIH METODA

* Ne modeliraju ne-funkcionalne zahtjeve.
* Ne uključuju informacije o tome je li neka metoda prikladna za neki problem.
* Mogu generirati previše dokumentacije.
* Modeli sustava su nekad presloženi i komplicirani da bi ih korisnik mogao shvatiti.

## CASE ALATI



Obuhvaća čitav niz alata dizajniranih za podršku

aktivnostima softverskog procesa kao što su

analiza i dizajn (u fazi specifikacije zahtjeva i

dizana arhitekture) ili testiranje.

Primjer CASE alata: 🡪

## PRIMJER CASE ALATA

|  |  |
| --- | --- |
| **MODUL** | **OPIS** |
| EDITORI DIJAGRAMA | Koriste se za stvaranje raznih modela a nisu samo alati za crtanje već informacije o entitetima koji se koriste na dijagramima pohranjuju u središnji repozitorij. |
| ALATI ZA DIZAJN, ANALIZU I PROVJERU | Obrađuju dizajn te dojavljuju sve pronađene greške i anomalije. |
| JEZIK ZA POSTAVLJANJE UPITA NA REPOZITORIJU | Omogućava dizajnerima da pronađu sve potrebne informacije o dizajnu u repozitoriju podataka. |
| RJEČNICI PODATAKA | Sadrži sve informacije o entitetima korištenim u dizajnu sustava. |
| IZRADA I GENERIRANJE IZVJEŠTAJA | Alat koju uzima informacije iz centralnog repozitorija i generira potrebnu dokumentaciju. |
| ALAT ZA IZRADU FORMI | Olakšava definiranje specifikacije ulaznih formi i dokumenata . |
| MOGUĆNOSTI UVOZA /IZVOZA | Omogućavaju izmjenu informacija između centralnog repozitorija i drugih razvojnih alata. |
| GENERATORI KODA | Automatski generiraju nekakvu okosnicu koda iz izrađenog dizajna. |