

## UVOD U PROGRAMSKO INŽENJERSTVO

**Softver** – uključuje sam program, ali i svu dokumentaciju i potrebne konfiguracijske datoteke.

Razlikujemo dvije vrste softvera: **generički** (igrice, office) i **kustomiziran** (custom-made radi se točno prema narudžbi). Razlika između njih se sve više smanjuje - napravi se generički softver koji se prilagodi potrebama naručitelja.

**Programsko inženjerstvo** je inženjerska disciplina koja se bavi sa svim aspektima razvoja softvera (od specifikacija pa sve do održavanja). Programski inženjeri bi trebali koristiti sistematičan i organiziran pristup – najčešće najučinkovitiji način za proizvodnju visoko kvalitetnog softvera.

**Razlika između programskog inženjerstva i računarstva (kao znanosti)** – Računarstvo se bavi teorijama i metodama, dok se programsko inženjerstvo bavi praktičnim problemima koji se javljaju pri izradi softvera.

**Razlika između programskog inženjerstva i sistemskog inženjerstva** – Sistemsko inženjerstvo se bavi razvojem i uspostavljanjem računalnih sustava. Sistemsko inženjerstvo ima više grana, a jedna od njih je programsko inženjerstvo.

**Softver proces** je niz aktivnosti koje se provode da bi se proizveo softver. Koriste se četiri osnovne aktivnosti:

- Specifikacija – klijent i inženjer definiraju mogućnosti i ograničenja softvera.
- Razvoj – softver se dizajnira i programira
- Validacija – provjerava se da li softver ispunjava zahtjeve klijenta
- Evolucija – softver se prilagođava novim zahtjevima klijenta i tržišta

**Model softver procesa** – pojednostavljeni prikaz procesa koji predstavlja jedan od pogleda na sam proces. Može uključivati aktivnost, uloge zaposlenika. Neki od modela su:

- „**Workflow**“ model – predstavlja niz aktivnosti procesa uključujući ulaz, izlaz i međuovisnosti.
- „**Dataflow**“ model – predstavlja proces kao set aktivnosti koje mijenjaju podatke (kako se neki ulazni podatak koristi za računanje izlaza).
- „**Role/action**“ model – predstavlja uloge zaposlenika i aktivnosti za koje su odgovorni.

Često se koriste paradigme:

- „**Waterfall**“ – aktivnosti se podijele u različite faze. Kada se faza jedanput završi više se ne vraća na nju. Najviše novca na integraciju i testiranje.
- **Iterativni razvoj** – Razvije se prototip koji se pokazuje klijentu. Program se dalje iterativno razvija uz stalno konzultiranje s klijentom. Istodobno se rade i specifikacija, programiranje, dizajniranje. Najviše novca na iterativni razvoj.
- **Razvoj temeljen na komponentama** – softver se izrađuje od već gotovih komponenti. Najviše novca na integraciju i testiranje.

Na evoluciju softver se troši i do četiri puta više sredstava nego na sam razvoj.

**Metode programskog inženjerstva** – je strukturiran pristup razvoju softvera čiji je cilj proizvodnja kvalitetnog i ekonomičnog softvera.

**CASE** (Computer- Aided Software Engineering) – pokriva široku paletu različitih programa koji se koriste kao pomoć pri različitim aktivnostima (modeliranju, debugiranju..).

**Atributi dobrog softvera:**

- **Održivost (Maintainability)** – softver mora biti napisan tako da može lako evolvirati
- **Pouzdanost (Dependability)** – uključuje sigurnost. Pouzdan softver ne bi smio uzrokovati fizičku ili ekonomsku štetu u slučaju greške.
- **Učinkovitost (Efficiency)**
- **Lagan za korištenje (Acceptability, Usability)**

Izazovi programskog inženjerstva:

- Heterogenost – koriste se različiti tipovi mreža, različiti operacijski sustavi, programi napisani u različitim programskim jezicima...
- Isporuka softvera – mora biti što brža, učinkovitija
- Povjerenje – da korisnici vjeruju sustavu

## **SOCIJALNO – TEHNIČKI SUSTAVI (*Socio-Technical Systems*)**

Socijalno –tehnički sustavi su sustavi koji uključuju softver, hardver i ljude. **Sustav** je svrsishodni skup međusobno povezanih komponenti koje rade zajedno radi ostvarivanja cilja.

Softver sustavi se dijele na dvije kategorije:

- **Tehnički sustavi temeljeni na računalima (*Technical computer based systems*)** su sustavi koji uključuju softver i hardver, ali ne i procese i procedure (sustav ne zna za što se koristi). To su npr. televizija, mobiteli, većina programa (word...)
- **Socijalno tehnički sustavi** – bitno da sustav uključuje znanje o tome kako se koristi. Takvi sustavi imaju točno definirane procese, uključuju operatere kao bitne „dijelove“ sustava, ponašaju se prema određenim pravilima.

### **Važne karakteristike socijalno-tehničkih sustava**

- ***Emergent properties*** –svojstva sustava kao cjeline koja ovise i o komponentama sustava, kao i o njihovom međuodnosu. Mogu se procjenjivati tek kada se sustav u potpunosti sastavi.
- **Nedeterministički** – za neki ulaz ne daju uvijek isti izlaz (ponašanje sustava ovisi o operateru).
- **Složeni odnosi s organizacijskim ciljevima** - razina do koje sustav podržava ciljeve organizacije ne ovisi samo o sustavu, već i o stabilnosti tih ciljeva, kako članovi organizacije interpretiraju te ciljeve...

Karakteristika svih sustava je da su svojstva i ponašanje komponenti isprepleteni. Sustavi su obično hijerarhijski organizirani i često sadrže druge sustave (podsustave). Podsustavi su nezavisni i mogu samostalno funkcionirati.

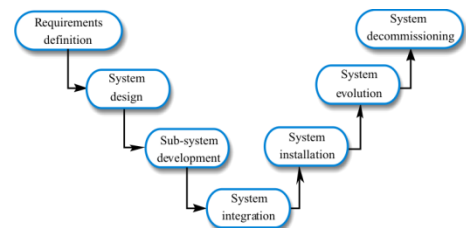
## Emergent system properties

- Funkcionalna svojstva – javljaju se kada svi dijelovi sustava rade zajedno da bi ostvarili neki cilj.
- Nefunkcionalna svojstva – odnose se na ponašanje sustava prilikom rada. To su npr. performanse, sigurnost, pouzdanost i često su jako bitna jer ukoliko nisu zadovoljena sustav može biti neprikladan za korištenje.

**Pouzdanost** je svojstvo koje se mora uvijek promatrati na razini sustava, a ne na razini pojedinih komponenti. Komponente su često međuovisne, pa se greške jedne komponente često propagiraju na cijeli sustav. Tri stvari utječu na pouzdanost sustava: pouzdanost hardvera, softvera i operatora.

Neki *emergent properties*: pouzdanost, sigurnost, jednostavnost održavanja, lakoća korištenja...

**Sistemska inženjerstvo** uključuje aktivnosti kao što su specifikacija zahtjeva, dizajna, implementacija, validacija, instalacija i održavanje socijalno tehničkih sustava. Sistem inženjeri se brinu o softveru, hardveru, ali i o interakciji korisnika i sustava.



*proces sistemskog inženjeringa*

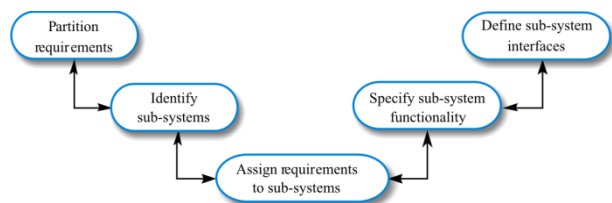
**Definicija sistemskih zahtjeva** uključuje specifikaciju funkcionalnosti sustava, esencijalna i poželjna svojstva. Nužna je suradnja s klijentima. Razlikujemo tri tipa zahtjeva:

- Apstraktni funkcionalni zahtjevi – osnovne funkcije koje sustav treba pružiti se definiraju na apstraktnoj razini
- Svojstva sustava – uključuju *emergent properties* kao što su dostupnost, sigurnost, performanse.
- Ponašanja/svojstva koja sustav ne smije imati.

Važno je da se odredi set zahtjeva koje sustav mora ispuniti.

**Dizajn sustava** – kako će sustav sastavljen od određenih komponenti pružiti traženu funkcionalnost. Aktivnosti dizajna sustava su:

- Podjela zahtjeva – zahtjevi se analiziraju i dijele u povezane grupe
- Identifikacija pod sustava – identificiraju se pod sustavi najčešće se grupe zahtjeva povežu s određenim podsustavima
- Dodjeljivanje zahtjeva podsustavima
- Određivanje funkcionalnosti podsustava pri čemu se određuje i međuodnos između pojedinih podsustava
- Definiranje potrebnih sučelja podsustava. Jednom kada se usuglasi oko tih sučelja razvoj podsustava se može odvijati paralelno.



## Modeliranje sustava

Sustav se može modelirati kao niz komponenti pri čemu se ističu odnosi između tih komponenti. Arhitektura sustava se može prikazati kao blok dijagram koji prikazuje glavne podsustave i veze među njima. Podsustavi se prikazuju kao kvadrati, dok se veze prikazuju kao strelice. Sustav se prikazuje kao niz međuovisnih podsustava. Svaki podsustav se prikazuje na sličan način sve dok se sustav ne rastavi na funkcionalne komponente.

## Razvoj podsustava

Prilikom razvoja podsustav često se počinje potpuno novi proces programskog inženjerstva. Ponekad se podsustavi razvijaju iz početka, dok se neke komponente kupuju. Podsustavi se često razvijaju paralelno i ponekad se javljaju problemi koji se protežu na više podsustava, tada se moraju raditi modifikacije specifikacije, dizajna...

## Integracija sustava

Tijekom integracije sustava nezavisno razvijeni podsustavi se spajaju u sustav. Mogu se spojiti sve podsustavi odjednom ili se sustavi integriraju jedan po jedan (zbog toga što je mala vjerojatnost da će svi podsustavi biti gotovi u isto vrijeme, smanjiva se vrijeme pronalaženja pogreški). Nakon integracije svih komponenti obavlja se iscrpno testiranje.

## Evolucija sustava

Sustav se često mijenja da bi se otklonile pogreške ili da se zadovolje novi zahtjevi. Može doći i do mijenjanja strukture organizacije, hardvera. Evolucija sustava je skupa zbog toga što:

- Predložene izmjene moraju biti pomno analizirane s poslovne i tehničke strane
- Podsustavi gotovo nikad nisu potpuno nezavisni pa promjene jednog podsustava često utječu na druge podsustave.
- Često razlozi zašto su neke stvari tako napravljene na početku nisu dokumentirani
- S starošću struktura sustava se iskvari što povećava cijenu promjena

Sustavi koji su evolvirali često ovise o prastarom hardveru i softveru. Ako takvi sustavi imaju ključnu ulogu u organizaciji nazivaju se „*legacy systems*“.

## Povlačenje sustava iz rada

Za hardver sustava to znači rastavljanje i eventualnu reciklažu, dok kod softver to može značiti pretvaranje korisnih podataka u format korišten na nekom drugom sustavu.

## Organizacije, ljudi i kompjuterski sustavi

Socijalno-tehnički sustavi su sustavi čiji je cilj ispunjenje nekog organizacijskog cilja. Ugrađeni su u samu organizaciju i moraju se ponašati prema pravilima te organizacije. Ljudski i organizacijski faktori su:

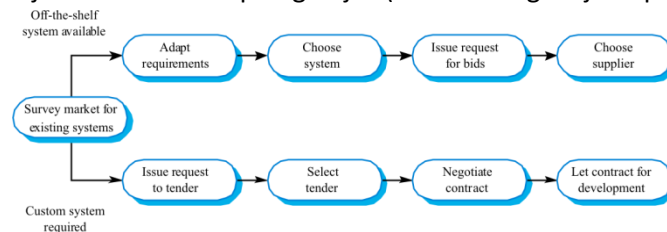
- Promjena procesa – da li sustav uvodi promjene u radni proces, ako su promjene velike možda će trebati dodatno obrazovanje radnika, otpuštanje radnika. To sve može stvoriti otpor prema sustavu.
- Promjena posla – da li sustav previše mijenja način na koji ljudi rade
- Promjena organizacije – Mijenja li sustav strukturu organizacije

Svi organizacijski faktori bi trebali biti uključeni u specifikaciju sustava.

## Organizacijski procesi

Razvoj nije jedini dio sistem inženjeringa. Tu spadaju još i nabava i sam rad sustava. Nabava je obično uključena kod organizacije koja će kupiti i koristiti sustav. Tada organizacija mora odrediti najbolji način za nabavu sustava. Veliki sustavi se često sastoje od generičkih i posebno razvijenih komponenti. Uvijek se radi detaljna analiza tržišta, te se odlučuje da li je ekonomičnije ići u naručivanje već gotovog softvera (nakon toga sustav se prilagođava i odabire se dobavljač) ili će se softver razvijati iz početka.

Operacijski procesi trebaju biti fleksibilni i prilagodljivi (često se događaju nepredvidljive situacije).



## “Legacy” sustavi

*Legacy* sustavi su socijalno tehnički sustavi razvijeni prije nekog vremena (10, 20 god) koji često koriste zastarjele tehnologije. Ti sustavi se ne sastoje samo od hardvera i softvera već i od zastarjelih procesa i procedura. Promjene jednog dijela tog sustava često uzrokuju kaskadne promjene kroz cijeli sustav. Takvi sustavi su često kritični za rad organizacije i održavaju se jer ih je prerizično zamijeniti.

Komponente: hardver, pomoćni softver, aplikacijski softver, podaci, poslovni procesi, poslovna pravila.

## KRITIČNI SUSTAVI

Kritični sustavi su sustavi kod čije greške dolazi do znatnih ekonomskih gubitaka, fizičke štete ili gubitka života. Razlikujemo tri glavna tipa takvih sustava:

- Sigurnosno kritični sustavi su sustavi čija greška može izazvati ozljedu, gubitak života ili onečišćenje okoliša
- Ciljno kritični („*Mission*“) sustavi čija greška može rezultirati neuspjehom nekog cilja
- Poslovno kritični sustavi su sustavi čija je greška jako skupa (banke)

Najvažnije „*emergent*“ svojstvo kritičnih sustava je „*dependability*“ (uključuje sigurnost, pouzdanost, dostupnost). To je zbog toga što korisnici često odbijaju koristiti sustave koji su nepouzđani, nesigurni; cijena greške sustava može biti ogromna; nepouzđani sustavi mogu uzrokovati gubitak informacija. Zbog toga se prilikom razvoja moraju koristiti iskušane i pouzdane metode, a ne nove metode koje se možda čine bolje ali čije se slabosti ne poznaju. Cijena verifikacije i validacije je često više od 50% ukupne cijene. Većina kritičnih sustava su socijalno tehnički sustavi kod kojih ljudi nadziru rad kompjuterskih sustava (jer ljudi mogu reagirati na nepredvidive situacije).

Greške kod kritičnih sustava se mogu dogoditi zbog greške hardvera, softvera ili zbog operatera. Moraju se uzeti u obzir slabosti svih dijelova kao i slabosti sustava kao cjeline.

**“Dependability”** se sastoji od:

- **Dostupnost** – da sustav ispuní zahtjev kad se to od njega zatraži
- **Pouzdanost** – da ispuní zahtjev kako je specificirano; da informacije imaju dovoljno detalja, da se informacija isporuči kada je potrebna
- **Sigurnost** – da sustav funkcionira bez katastrofalnih grešaka, da je zaštićen od upada. Uključuje i osiguranje da se podaci ne oštete.
- Jednostavnost popravka, održavanja, **tolerancija na greške**

To svojstvo pokazuje koliko korisnici vjeruju sustavu. Dizajneri moraju često raditi kompromise između performansi i „dependability“-a.

## Dostupnost i pouzdanost

Pouzdanost sustava je vjerojatnost da će sustav pružiti usluge onako kako su specificirane (problem jer korisnik može očekivati ponašanje koje nije specificirano). Dostupnost sustava je vjerojatnost da će sustav pružiti uslugu kada je korisnik zatraži. Problem s ovim definicijama je što ne uzimaju u obzir težinu greške ili posljedice nedostupnosti. Npr. kod telefonskog sustava važnija je dostupnost nego pouzdanost. Mogu se još definirati i:

- Pouzdanost – vjerojatnost da će sustav raditi bez greške u nekom okolišu, tijekom nekog vremena.
- Dostupnost – vjerojatnost da će sustav u nekom trenutku raditi i moći pružiti traženu uslugu.

Uz pouzdanost se vežu termini: pad sustava (*system failure*), greška (*fault* i *error*). Za povećanje pouzdanosti sustava koriste se pristupi:

- Izbjegavanje grešaka – koriste se razvojne tehnike koje smanjuju vjerojatnost pojave grešaka
- Detekcija i popravljavanje pogreški – koristi se validacija i verifikacija da se poveća vjerojatnost detekcije pogreški tako da se mogu ispraviti prije nego što sustav krene u uporabu.
- Tolerancija na pogreške – tehnike koje osiguravaju da pogreške ne dovode do većih grešaka u radu ili rušenja sustava.

Korisnici socijalno tehnoloških sustava se često prilagode sustavu na način da pronađu način kojim izbjegavaju greške.

## Safety

Sustavi kod kojih je sigurnost kritična u niti kojem slučaju ne smiju ozlijediti ljude ili oštetiti imovinu. Ugrađuje se dodatna kontrola koja može biti hardverska ili softverska. Ovakav tip softvera se dijeli na dvije kategorije:

- Primarni sigurnosno kritični sustavi – softver koji ugrađen kao kontrolor sustava. Nefunkcioniranje takvog softvera može dovesti do kvara hardvera.
- Sekundarno sigurnosno kritični sustavi (*safety critical systems*) – softver koji indirektno može izazvati povredu ili oštećenje.

Ako je softver pouzdan ne mora značiti i da je siguran. Zbog toga što npr. specifikacija može biti nedorečena i ne opisuje kako će sustav reagirati u kritičnim situacijama; greške u hardveru mogu uzrokovati da se sustav ponaša na nepredvidiv način; operatori mogu unijeti unose koji nisu netočni sami za sebe, ali u određenim situacijama mogu dovesti do greške u sustavu.

Opasnosti se mogu izbjegavati, detektirati i otkloniti ili ukoliko se već dogode nastoji se da šteta bude što manja.

## Security

Sigurnost je svojstvo sustava koje odražava sposobnost sustava da se zaštiti od vanjskih napada. Jako je važna u vojnim, bankarskim sustavima. Napad može uzrokovati probleme kao što su: uskraćivanje usluge, promjene podataka, otkrivanje povjerljivih informacija. Sigurnost sustava se može osigurati:

- Izbjegavanjem slabosti – sustav se npr. ne priključuje na Internet
- Otkrivanje i neutraliziranje napada – npr. korištenjem antivirusnih sustava...

## SOFTVERSKI PROCESI

Softver proces je niz aktivnosti koje omogućavaju razvoj softvera. Te aktivnosti mogu uključivati razvoj novog softvera, ali i modifikaciju već postojećeg. Aktivnosti zajedničke za sve softverske procese su:

- Specifikacija – definira se funkcionalnost i ograničenje softvera
- Dizajn i implementacija – „proizvodnja“ softvera
- Validacija – provjerava se da li softver radi ono što klijent želi
- Evolucija – softver se mijenja zbog toga što se mijenjaju potrebe klijenta

## Modeli softverskih procesa

Softver proces model je apstraktna reprezentacija softver procesa i predstavlja proces iz neke perspektive. Najčešći modeli su:

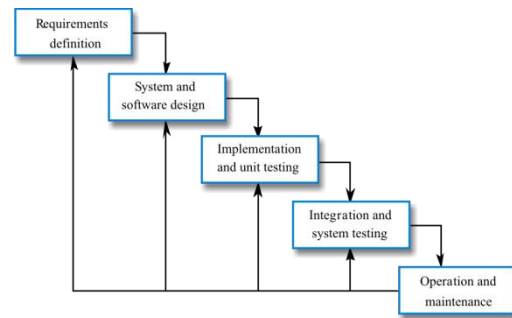
- **„Waterfall“, model vodopada** – aktivnosti se podijele u različite faze. Kada se faza jedanput završi više se ne vraća na nju. Najviše novca na integraciju i testiranje.
- **Iterativni razvoj** – Razvije se prototip koji se pokazuje klijentu. Program se dalje iterativno razvija uz stalno konzultiranje s klijentom. Istodobno se rade i specifikacija, programiranje, dizajniranje. Najviše novca na iterativni razvoj.
- **Razvoj temeljen na komponentama** – softver se izrađuje od već gotovih komponenti. Najviše novca na integraciju i testiranje.

Ovi modeli se međusobno ne isključuju i često se koriste istodobno, naročito kod razvoja velikih sustava. Postoje različite varijante ovih generičkih procesa od kojih je najvažniji „formalni razvoj sustava“ kod kojeg se kreira matematički model sustava kojeg se zatim, koristeći matematičke transformacije, pretvara u izvršni kod. Najpoznatiji takav proces je „CleanRoom“ (IBM).

## Model vodopada

Sastoji se od slijedećih aktivnosti:

- Definicija i analiza zahtjeva – u suradnji s korisnicima detaljno se određuju funkcionalnosti i ograničenja softvera (specifikacija)
- Dizajn softver i sustava – obavlja se podjela zahtjeva, odredi se općenita arhitektura sustava
- Implementacija i testiranje – komponente sustava se povezuju i testiraju kao cjelina. Nakon testiranja obavlja se isporuka klijentu
- Rad i održavanje – obično je ovo najduži dio procesa. Sustav se instalira kod klijenta. Ispravljaju se greške koje nisu otkrivene u fazi testiranja. Evolucija softvera...



Rezultat svake faze su odobreni dokumenti. Slijedeća faza ne bi smjela početi sve dok se prethodna ne završi. U stvarnosti faze se isprepliću, pa se ovaj proces odvija u više iteracija. Nikad se ne radi prevelik broj iteracija (skupe su) već se u jednom trenutku odluči da se ide dalje, a neriješeni problemi se ignoriraju ili zaobilaze „trikovima“. Svi problemi će se poslije morati ispraviti u fazi evolucije.

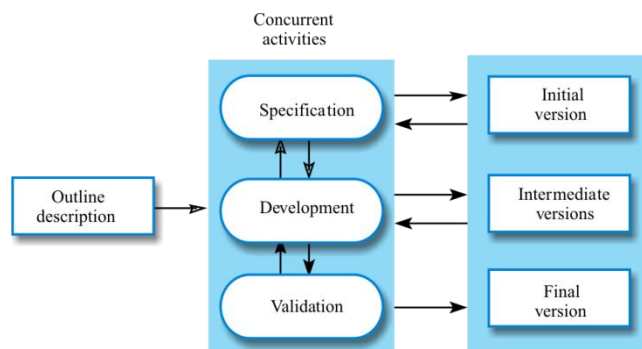
Prednosti su što se dokumentacija izrađuje u svakoj fazi i da se podudara s drugim procesnim modelima. Glavni problem je nefleksibilna podjela procesa u različite faze. Teško je ukomponirati nove korisnikove zahtjeve. Ovaj model bi se samo trebao koristiti kada su zahtjevi dobro specificirani i kada postoji mala vjerojatnost da će se mijenjati tijekom razvoja sustava.

## Evolucijski (iterativni) razvoj

Kod ovog modela razvije se prototip koji se pokaže korisniku i taj prototip se kroz više iteracija (sve u suradnji s korisnikom) pretvara u gotov proizvod. Specifikacija, razvoj i iteracija se isprepliću.

Razlikujemo dva tipa:

- „*Exploratory development*“ – cilj je raditi s klijentom (pri čemu se detaljno pregledavaju zahtjevi). Razvoj počinje s dijelovima sustava koji su razumljivi. Sustav se razvija dodavanjem novih mogućnosti koje predlaže klijent.
- „*Throwaway prototyping*“ – cilj iterativnog razvoja je razumijevanje korisnikovih potreba. Izrađuje se prototip pomoću kojeg se eksperimentira sa zahtjevima koji nisu dovoljno razumljivi.





Ovakav model se koristi kod razvoja malih i srednjih sustava i često je učinkovitiji od modela vodopada (što se tiče realiziranja korisnikovih želja). Prednost je i to što se specifikacija razvija inkrementalno. Nedostatci su:

- Ako se sustav brzo razvija neekonomično je proizvoditi dokumentaciju koja odražava stanje svake verzije
- Sustav je često loše strukturiran – stalna promjena iskvari strukturu samog sustava

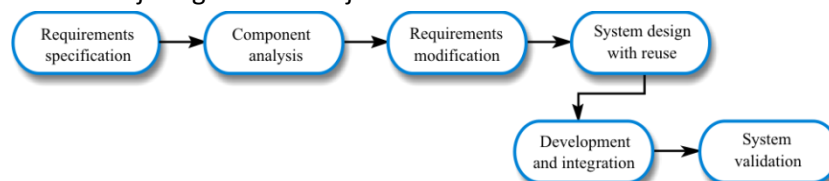
Za veće sustave preporuča se koristiti kombinacija ovog modela i modela vodopada. Npr. dijelovi zahtjeva koji su dobro razumljivi (specificirani) se razvijaju koristeći model vodopada, dok se dijelovi kao npr. sučelje razvijaju koristeći npr. „*Exploratory development*“.

## Razvoj temeljen na komponentama

Često se neke komponente jednostavnim modifikacijama mogu koristiti u više projekata. To je naročito značajno kod brzog evolucijskog razvoja. Kod ovog modela razlikujemo slijedeće faze:

- Specifikacija zahtjeva
- Analiza komponenti – traže se komponente pomoću kojih se mogu implementirati zahtjevi
- Modifikacija zahtjeva – zahtjevi se mijenjaju u skladu s pronađenim komponentama. Ukoliko se zahtjevi ne mogu mijenjati tada se traže druge komponente (ili se modificiraju).
- Dizajn sustava korištenjem komponenti – dizajnira se framework sustava (ili se koristi već gotov) pri čemu se u obzir uzimaju odabrane komponente. Razvija se nove komponente ukoliko nema već gotovih.
- Razvoj i integracija

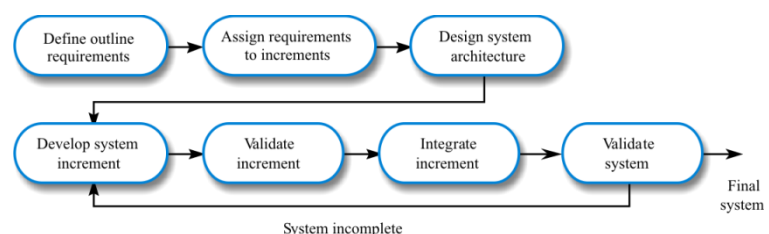
Korištenjem već gotovih komponenti smanjuju se cijena, rizici i vrijeme razvoja. Mana je što je mijenjanje originalnih zahtjeva gotovo neizbježno.



## Iteracija procesa

Softver se mora mijenjati u skladu s mijenjanjem potreba klijenata. Razlikujemo dva procesa iteracije:

**Inkrementalna isporuka** – specifikacija, dizajn i implementacija se razbijaju u niz inkremenata koji se zatim razvijaju. Specifikacija se razvija paralelno sa softverom. To je često u konfliktu s mnogim organizacijama gdje je puna specifikacija dio ugovora. Korisnici okvirno definiraju zahtjeve i njihovu važnost. Zatim se odredi broj inkremenata s time da svaki inkrement ispuni neki od zahtjeva (najvažniji zahtjevi se ispunjavaju najprije). Prije razvoja svakog inkrementa u detalje se razrade njegovi zahtjevi (ukoliko za vrijeme razvoja stignu izmjene za taj inkrement one se ne prihvaćaju). Nakon razvoja inkrement se integrira u sustav. Zatim se obavlja isporuka i klijent testira softver. Ovakav način ima brojne prednosti: klijent ne treba čekati do kraja razvoja da dobije neku funkcionalnost softvera, klijent pomoću inkrementa treba steći dojam o sustavu i na taj način prilagoditi zahtjeve, malen rizik neuspjeha projekta, budući da se najprije ispune najvažniji zahtjevi,



oni se najviše i testiraju. Postoje i problemi kod ovakvog razvoja: inkrementi moraju biti relativno maleni (do 20000 linija koda) pa ponekad zna biti teško odrediti koji zahtjevi idu u koji inkrement; teško je odrediti koje osnovne funkcionalnosti su potrebne za sve inkremente (specifikacija nije do kraja razrađena).

**Spiralni razvoj** – Razvoj sustava ide spiralno prema vani od prototipa pa sve do gotovog sustava. Svaka petlja spirale predstavlja jednu fazu procesa (prva petlja izvedivost, druga definicija zahtjeva...) i podijeljena je u četiri sektora:

- Postavljanje ciljeva – definiraju se ciljevi, ograničenja, rizici za ovu fazu projekta
- Procjena rizika i načini njihovog smanjivanja – za svaki rizik izvodi se detaljna analiza, poduzimaju se određeni koraci koji smanjuju rizik.
- Razvoj i validacija – u skladu s rizicima se izabire razvojni model (ako je problem sučelje – evolucijski, ako je problem sigurnost onda razvoj na temelju formalnih transformacija)
- Planiranje – projekt se pregledava i donose se odluke da li se ide na slijedeću petlju spirale

Glavna razlika spiralnog modela i ostalih modela je u velikom posvećivanju pažnje rizicima.

## Procesne aktivnosti

Četiri osnovne procesne aktivnosti: specifikacija, razvoj, validacija i evolucija. Te aktivnosti su poredane ovisno o tome koji se procesni model koristi.

### Specifikacija softvera

Specifikacija sadrži zahtjeve pomoću kojih se definira točno što sve korisniku treba, koja su ograničenja sustava... Ovo je kritičan dio jer bilo koje greške u ovom dijelu se propagiraju na cijeli sustav. Zahtjevi se predstavljaju u dvije razine: krajnji korisnici dobivaju manje detaljne zahtjeve, dok programeri moraju dobiti potpuno specifičan zahtjev. Sam proces specifikacije se dijeli na četiri dijela:

- Studija izvedivosti – procjenjuje se da li se korisnikove potrebe mogu zadovoljiti koristeći dostupne tehnologije, hoće li traženi sustav biti ekonomičan i može li se razviti koristeći dostupan budžet. Na osnovu ove studije se odlučuje da li će se nastaviti s daljnjom analizom.
- Analiza i opisivanje zahtjeva – određuju se zahtjevi sustava kroz proučavanje već postojećih sustava, razgovora s klijentima... U ovu fazu može biti uključen i razvoj prototipova koji će pomoći korisniku da razumije sustav.
- Specifikacija zahtjeva – na osnovu skupljenih informacije piše se lista zahtjeva. Razlikujemo dva tipa zahtjeva: korisnikove zahtjeve – apstraktne informacije o zahtjevima, sistemski zahtjevi – detaljniji opis funkcionalnosti koje trebaju biti omogućene.
- Validacija zahtjeva – provjerava se da li su zahtjevi konzistentni, proturječni, realni, potpuni...

### Dizajn i implementacija

Ovo je faza u kojoj se na osnovu specifikacije izrađuje sam sustav. Uvijek uključuje programiranje i dizajn, a kod evolucijskog modela i dorađivanje specifikacije. Softver dizajn je opis strukture, podataka, sučelja između različitih komponenti, algoritama... Dizajn se najčešće piše u nizu iteracija. Sam proces dizajna može uključivati i razvoj nekoliko modela sustava na različitim nivoima apstrakcije.

Procesne aktivnosti dizajna su često isprepletene, a razlikujemo slijedeće procese:

- Dizajn arhitekture – Određuju se podsustavi i njihov odnos
- Apstraktna specifikacija – za svaki podsustav radi se apstraktna specifikacija usluga i određuju se njegova ograničenja
- Dizajn sučelja – za svaki podsustav se određuje sučelje s drugim podsustavima.
- Dizajn komponenti – određuju se komponente, njihove uloge i sučelja
- Dizajn struktura podataka – određuju se strukture podataka koje će se koristiti
- Dizajn algoritama

U praksi se često neke aktivnosti odgađaju. Ukoliko se koriste agilne metode rezultat dizajna neće biti dokument, već kod (dokument se piše jedino kod dizajna arhitekture). Ako se koriste strukturirane metode tada se kao rezultat dizajna dobivaju grafički modeli iz kojih se generira kod.

Nakon razvoja programi se testiraju. Testiranje je proces otkrivanja pogreški, dok se proces ispravljanja pogreški u programu naziva *debugiranje*.

### Validacija

Validacija je provjeravanje da li sustav radi prema specifikaciji. Razlikujemo više faza:

- Testiranje komponenti – provjerava se da li pojedine komponente ispravno rade. Svaka komponenta se testira nezavisno.
- Testiranje sustava – Komponente se integriraju u sustav. Ovaj dio je namijenjen otkrivanju greški koje nastaju uslijed nepredvidivog međudjelovanja komponenti, potvrđivanju funkcionalnih i nefunkcionalnih zahtjeva, testiranju *emergent* svojstava...
- Testiranje prihvatljivosti – ovo je posljednja faza testiranja prije nego što se sustav prihvati za stvarni rad.

Ako se koristi inkrementalni razvoj svaki inkrement bi se trebao testirati kako se razvija. Kod ekstremnog programiranja testovi se razrađuju zajedno sa zahtjevima, prije samog razvoja. Testiranje prihvatljivosti se često naziva alfa testiranje koje traje sve dok se klijent ne složi da je sustav prihvatljiva implementacija zahtjeva. Kod generičkog softvera postoji i faza beta testiranja kod kojeg se sustav isporuči većem broju potencijalnih klijenata koji zatim prijavljuju greške razvojnom timu.

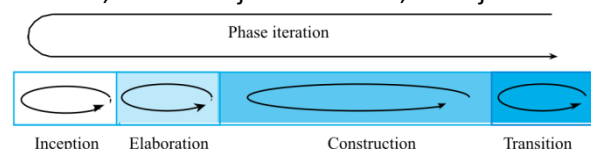
### Rational unified process (RUP)

RUP se razvio iz rada s UML-om i opisuje se iz 3 perspektive:

- Dinamička perspektiva koja pokazuje faze modela kroz vrijeme
- Statička perspektiva koja pokazuje procesne aktivnosti
- Praktična perspektiva koja predlaže dobre načine rada

RUP je model u fazama koji razlikuje četiri faze u softver procesu. Te faze su više povezane s poslovnim sustavom. To su:

- Počinjanje – radi se poslovna analiza sustava. Identificiraju se svi vanjski entiteti (ljudi, sustavi) koji će komunicirati sa sustavom. Radi se procjena koliki će utjecaj sustav imati na posao.
- Objašnjavanje cilj je: razviti razumijevanje problema, određivanje arhitekture, razvoj projektnog plana i identifikacija rizika.
- Konstrukcija – ova faza se bavi dizajnom, programiranjem i testiranjem. Dijelovi sustava se razvijaju paralelno.
- Tranzicija – stavljanje sustava u stvarni rad.



Statička perspektiva RUP-a se fokusira na aktivnosti koje se događaju tijekom razvoja (nazivaju se *workflows*). Prednost korištenja statičkih i dinamičkih pogleda je u tome što faze razvojnog procesa nisu povezane sa specifičnim radnim tokovima. Zbog toga, u teoriji svi radni tokovi mogu biti aktivni u svim fazama projekta, ali u praksi se na početku projekta više vremena ulaže u poslovno modeliranje, a u kasnijim fazama na testiranje.

RUP preporučeni načini rada:

- Iterativan razvoj softvera → razvoj inkremenata se temelji na potrebama klijenta. Najvažniji prioriteti se moraju ispuniti što prije.
- Upravljanje zahtjevima → točno navođenje korisnikovih zahtjeva. Sve izmjene se detaljno dokumentiraju. Analizira se utjecaj izmjena na sustav prije njihovog prihvaćanja.
- Koristi se arhitektura temeljena na komponentama
- Vizualno modeliranje softvera
- Verificiranje kvalitete softvera
- Kontrola promjena softvera

RUP nije praktičan za sve razvojne procese, ali predstavlja novu generaciju generičkih procesa. Najvažnije novosti su razdvajanje faza i radnih tokova, te prihvaćanje da je stavljanje sustava u stvarni rad sastavni dio razvoja. Faze su dinamične i imaju ciljeve. Radni tokovi su statični i to su tehničke aktivnosti koje nisu povezane s pojedinom fazom, ali se mogu koristiti tijekom razvoja da bi se ostvarili ciljevi svake faze.

## CASE (Case –Aided Software Engineering)

CASE je naziv za softver koji se koristi kao pomoć (automatiziraju određene procesne aktivnosti) kod softverskih procesa (specifikacija, dizajn, razvoj...). U tu skupinu spadaju dizajn editori, kompajleri...

Aktivnosti koje mogu biti automatizirane koristeći CASE alate su:

- Razvoj grafičkih modela sustava kao dio specifikacije ili softver dizajna
- Razumijevanje dizajna koristeći podatkovni rječnik koji sadrži informacije o entitetima i odnosima
- Generiranje korisničkog sučelja iz opisa kojeg korisnik kreira interaktivno
- Debugiranje
- Automatsko prevođenje programa napisanih u starijim jezicima u neke novije jezike

CASE tehnologije danas postoje za gotovo sve rutinske aktivnosti koje se javljaju u procesu razvoja softvera što je dovelo do značajnih poboljšanja u kvaliteti i produktivnosti (do 40%).

Glavno ograničenje CASE-a je u tome što je programsko inženjerstvo kreativan proces, a umjetna inteligencija još uvijek nije na tom nivou. Programsko inženjerstvo je timska aktivnost gdje inženjeri dosta vremena provode komunicirajući s drugim članovima tima.

### **Klasifikacija CASE alata**

Klasifikacija CASE alata pomaže razumjeti tipovi CASE alata i njihovu ulogu u različitim aktivnostima. Razlikujemo tri perspektive:

- Funkcionalna perspektiva (za planiranje, editiranje, upravljanje promjenama, prototipove, obradu jezika, testiranje, analizu, dokumentaciju)
- Procesna perspektiva – prema procesnoj aktivnosti koju podržavaju
- Integracijska perspektiva – dijele se prema tome kako su organizirani u integrirane jedinice koje pružaju podršku za jednu ili više procesnih aktivnosti

Postoji i podjela na:

- Alate – daju podršku za individualne procesne zadatke
- Radni stolovi (workbenches) – daju podršku za procesne faze ili aktivnosti (specifikacija, dizajn...). Sastoje se od niza alata
- Okoliš – podržavaju značajan dio softver procesa. Sastoje se od nekoliko „radnih stolova“

## **UPRAVLJANJE PROJEKTOM**

Dobro upravljanje projektom je bitno jer smanjuje vjerojatnost kašnjenja, prekoračenja dostupnih sredstava, nezadovoljavanja zahtjeva... Upravitelj projekta je zadužen za planiranje razvoja, poštivanje rokova, zahtjeva. Upravljanje softverskim projektima je drugačije od upravljanja drugim inženjerskim projektima jer je softver apstraktan (napredak se ne može vidjeti, već se pouzda u dokumentaciju), ne postoje standardni procesi jer je programsko inženjerstvo relativno mlada disciplina, veliki programi se često razlikuju od prethodnih, tehnologija se brzo mijenja pa znanje zastarijeva...

### **Aktivnosti vezane za upravljanje projektom**

Ne postoji standardan set aktivnosti koje vrijede u svim organizacijama ili na svim projektima, ali u velikoj većini slučajeva postoje slijedeće aktivnosti: pisanje prijedloga (prikazuju se ciljevi projekta i načini njihovog izvršavanja, odredi okvirna cijena), planiranje i izrada rasporeda (identificiraju se aktivnosti, kontrolne točke, razrada plana kojeg bi se razvojni tim trebao pridržavati), predviđanje cijene projekta (procjenjuju se resursi potrebni za završetak projekta), nadgledanje napretka projekta i usporedba stvarnog i predviđenog napretka i troškova, izabiranje osoblja, pisanje izvještaja i prezentiranje.

### **Planiranje projekta**

Učinkovito planiranje projekta ovisi o temeljitom planiranju napretka projekta. Upravitelji moraju predviđati moguće probleme i predvidjeti način njihovog rješavanja. Na početku se radi projektni plan kojeg bi se trebalo pridržavati (mijenja se ukoliko se pojave bolje informacije). Prilikom izrade plana upravitelj procjenjuje ograničenja (datum isporuke, dostupna sredstva...) i na osnovu tih procjena izrađuje projektne parametre (struktura projekta, veličina...). Zatim se definiraju kontrolne točke i nakon toga se izrađuje raspored aktivnosti (često se uzima pesimističan pristup). Nakon nekog vremena radi se pregled napretka i odstupanje od plana (plan se često mijenja jer je teško sve predvidjeti).

## Projektni plan

Plan sadrži informacije o dostupnim resursima, projektne aktivnosti, raspored rada... Svaki plan bi trebao imati slijedeće dijelove: Uvod (ukratko se objašnjavaju ciljevi projekta i određuju ograničenja - budžet, vrijeme...), organizacija (način na koji je razvojni tim organiziran, koje su njihove uloge...), analiza rizika (koji su rizici, kako će utjecati na projekt, kolika je vjerojatnost njihovog pojavljivanja), hardverski i softverski zahtjevi, podjela projekta na aktivnosti (određivanje kontrolnih točaka), raspored (ovisnosti između aktivnosti, koliko je vremena potrebno za određenu aktivnost), određivanje mehanizama za nadzor i obavješćivanje. Projektni plan se često pregledava i revidira za vrijeme trajanja projekta (naročito raspored – često se mijenja).

## Kontrolne točke i isporuke

Kod planiranja projekta utvrđuje se niz kontrolnih točki (prepoznatljiv završetak neke aktivnosti) nakon kojih se daje izvještaj koji se zatim predstavlja upravi. Isporuke klijentu (*deliverable*) se najčešće rade nakon završetka neke velike projektne faze (npr. specifikacije ili dizajna). Da bi se utvrdile kontrolne točke projekt se treba razbiti na osnovne aktivnosti. Npr. kod izrade zahtjeva to mogu biti studija izvedivosti, analiza zahtjeva, razvoj prototipa, studija dizajna, specifikacija zahtjeva.

## Izrada projektnog rasporeda

Upravitelj mora predvidjeti vrijeme i resurse potrebne za završetak pojedinih aktivnosti. Često se neke aktivnosti mogu paralelno izvoditi, no tada je nužno dobro koordiniranje i organizacija posla. Jako je bitno spriječiti kašnjenje projekta jer neka kritična aktivnost nije završena. Prilikom izrade rasporeda uvijek se uzima pesimistična procjena (naročito ako se takav tip sustava po prvi put radi).

**Grafovi i mreže aktivnosti** se koriste kao grafički prikazi koji ilustriraju projektni raspored. Grafovi sa stupcima (*bar chart*) prikazuju tko je odgovoran za pojedinu aktivnost i kada će ta aktivnost početi i završiti. Mreže aktivnosti pokazuju međuovisnosti pojedinih aktivnosti, pa se na temelju njih može odrediti koje se aktivnosti mogu paralelno, a koje se moraju sekvencijalno izvršavati. Aktivnosti se prikazuju kao kvadrati, a kontrolne točke kao kvadrati s oblim kutovima. Minimalno vrijeme završetka projekta se može procijeniti uzimajući u obzir najduže putove u grafu. Kritičan put je niz ovisnih aktivnosti koje određuju vrijeme potrebno da se projekt završi (ako jedna od tih aktivnosti zapne cijeli projekt zapinje).

## Upravljanje rizicima

Upravljanje rizicima uključuje predviđanje rizika koji bi mogli utjecati na raspored projekta ili na kvalitetu softvera i ta predviđanja se dokumentiraju u projektnom planu zajedno s analizom posljedica ukoliko se rizik dogodi. Razlikujemo tri kategorije rizika: projektni rizici (utječu na raspored ili resurse npr. gubitak iskusnog člana tima), rizici vezani za proizvod (utječu na kvalitetu ili performanse softvera, npr. kupljena komponenta se ne ponaša kako treba), poslovni rizici (rizici koji utječu na organizaciju koja razvija softver).

Upravljanje rizicima se dijeli na više faza:

- Identifikacija rizika- postoji više tipova rizika. Mogu biti vezani za tehnologiju, ljude, organizaciju, alate, zahtjeve, procjene.
- Analiza rizika – vjerojatnost pojavljivanja (u postotcima) i posljedice rizika (katastrofalno, ozbiljno, podnošljivo) se analiziraju. Pri tome se uglavnom oslanja na iskustvo.
- Planiranje – što će se poduzeti ako se rizik ipak pojavi. Razlikujemo tri strategije a to su:
  - izbjegavanje (smanjuje se vjerojatnost pojavljivanja rizika)
  - minimizacija (smanjuje se posljedica rizika, npr. kod rizika vezano za ljude se napravi češće preklapanje posla tako da više ljudi zna o stvarima koje se rade)
  - pripremanje za najgore i pravljenje strategija za nošenje s rizikom
- Nadzor rizika – redovito se procjenjuju identificirani rizici i odlučuje se da li se promijenila njegova vjerojatnost ili utjecaj na sustav.

## ZAHTJEVI

Zahtjevi sustava su opis usluga/funkcionalnosti koje sustav pruža kao i njegova operacijska ograničenja. Zahtjevi odražavaju potrebe klijenta. Sam proces pronalaženja, analiziranja, dokumentiranja i provjeravanja funkcionalnosti i ograničenja se naziva inženjering zahtjeva (*requirements engineering*).

Razlikujemo dvije vrste zahtjeva:

- **Korisnički zahtjevi** – izjave napisane normalnim, govornim jezikom koje opisuju koje usluge sustav nudi i ograničenja pod kojima mora funkcionirati.
- **Sistemske zahtjevi** – detaljno određuju funkcionalnosti, usluge i ograničenja sustava. Dokument u kojem su opisani ti zahtjevi mora biti precizan i mora točno određivati što će biti implementirano.

## Funkcionalni i nefunkcionalni zahtjevi

**Funkcionalni zahtjevi** su tvrdnje o uslugama koje sustav treba omogućiti, kako treba reagirati na određene ulaze i kako se treba ponašati u određenim situacijama. U nekim slučajevima se određuje i što sustav ne smije raditi. Ovisi o tipu sustava, korisnicima sustava, pristupu organizacije. Moraju biti napisani jasno, potpuno (definiraju se sve usluge koje trebaju korisniku) i konzistentno.

**Nefunkcionalni zahtjevi** – ograničenja na funkcionalnosti sustava (uključuju vremenska ograničenja, ograničenja na razvojni proces). Često se odnose na sustav kao cjelinu, na *emergent properties* kao što su pouzdanost, sigurnost, performanse; mogu definirati ograničenja na sustav kao npr. sposobnosti I/O uređaja i često su važnije od pojedinih funkcionalnih zahtjeva. Neki nefunkcionalni zahtjevi ograničavaju i proces koji se koristi kod razvoja (specificiraju se standardi kvalitete). Tipovi nefunkcionalnih zahtjeva su:

- Zahtjevi proizvoda – određuju kako će se proizvod ponašati (koliko je sustav brz, koliko je memorije potrebno, pouzdanost...)
- Organizacijski zahtjevi – izvode se iz načina rada klijentove i programerove organizacije. Npr. koji će se procesni standardi koristiti, koji programski jezik, metoda dizajna...
- Vanjski zahtjevi – svi zahtjevi koji proizlaze iz faktora van sustava. Npr. kako sustav komunicira s drugim sustavima, da sustav funkcionira u skladu sa zakonom, etički zahtjevi...

Čest problem je da su nefunkcionalni zahtjevi teški za verificiranje. Pa se, kad god je to moguće, nastoje kvantitativno izraziti.



**Zahtjevi domene** (*Domain requirements*) – zahtjevi koji proizlaze iz domene iz koje sustav dolazi mogu biti funkcionalni i nefunkcionalni. To su specijalizirani zahtjevi koji izlaze iz poslovnog okruženja u kojem sustav funkcionira, koristi se stručna terminologija koju programski inženjeri možda ne razumiju.

## Korisnički zahtjevi

Korisnički zahtjevi bi trebali opisivati funkcionalne i nefunkcionalne zahtjeve na način da su lako razumljivi korisnicima sustava bez potrebe za tehničkim znanjem. Trebaju samo specificirati vanjsko ponašanje sustava. Koriste se jednostavni izrazi, tablice, dijagrami. Postoje problemi jer takvo izražavanje može dovesti do nedovoljne jasnoće, miješanja zahtjeva (funkcionalni, nefunkcionalni zahtjevi, ciljevi sustava možda nisu jasno podijeljeni). Preporučljivo je da je korištenje standardnih formata.

## Zahtjevi sustava

Zahtjevi sustava su proširena verzija korisničkih zahtjeva i služe kao polazna točka za dizajniranje sustava. Objašnjavaju kako će sustav ispuniti korisničke zahtjeve. Idealno zahtjevi sustava bi trebali opisivati vanjsko ponašanje i ograničenja sustava i ne bi se trebali baviti dizajnom ili implementacijom sustava. Ali u praksi je jako teško potpuno isključiti sve informacije o dizajnu: često se mora dizajnirati prototip arhitekture koji služi kao pomoć kod strukturiranja zahtjeva; sustav mora komunicirati s drugim sustavima što nameće neka ograničenja kod dizajna. Kod opisa se koristi normalni jezik što ponekad može unijeti nejednoznačnosti, pa se često koriste grafički modeli, matematičke specifikacije, strukturirani normalni jezik... Ukoliko se za opisivanje koristi strukturirani normalni jezik tada se svi zahtjevi pišu na standardni način – zadržava se razumljivost normalnog jezika ali se i osigurava konzistentnost. Ako se koristi standardna forma za određivanje funkcionalnih zahtjeva tada slijedeće informacije moraju biti uključene: opis funkcije/entiteta koji se opisuje, opis ulaza i odakle dolaze, opis izlaza i gdje se proslijeđuju, indikacija drugih entiteta koji se koriste, opis akcija koje se poduzimaju; ako se koristi funkcionalni opis tada se naznači koje svojstvo je istinito prije nego što se funkcija pozove i koje svojstvo je istinito nakon što se pozove; nuspojave (ako postoje).

## Specifikacija sučelja

Gotovo svi sustavi moraju komunicirati s postojećim sustavima. Te specifikacije se moraju definirati što ranije. Razlikujemo tri tipa sučelja koja se moraju definirati:

- Proceduralna sučelja – postojeći programi nude usluge kojima se može pristupiti koristeći procedure sučelja (API)
- Strukture podataka koje se prenose iz jednog sustava u drugi. Najbolje je koristiti grafičke modele
- Reprezentacije podataka

## Dokument sa specifikacijom zahtjevima (*Software requirements document*)

SRS (specifikacija) je službeni dokument koji opisuje što će sve razvojni tim implementirati i trebao bi uključivati i korisnikove zahtjeve u detaljnu specifikaciju zahtjeva sustava (ponekad se mogu i spojiti). Specifikacija može biti pisana za velik broj tipova korisnika što znači da mora postojati kompromis između objašnjavanja zahtjeva korisnicima, preciznog definiranja zahtjeva.

Ako se koristi evolucijski pristup mnogi dijelovi će se izbaciti. Fokus će biti na definiranju korisničkih zahtjeva (na visokom nivou) i nefunkcionalnih zahtjeva. Tada programeri i dizajneri moraju sami odlučivati kako zadovoljiti korisnikove potrebe. Ako je softver dio većeg inženjerskog sustava tada se



zahtjevi moraju detaljno objasniti. Agilne metode tvrde da se zahtjevi mijenjaju tako brzo da je dokument zastario čim je napisan. Ekstremne metode predlažu da se zahtjevi inkrementalno skupljaju i zapisuju na kartice.

Tipičan izgled dokumenta:

- Uvod – svrha, doseg projekta, rječnik,
- Generalni opis – perspektiva, funkcije, karakteristike korisnika, ograničenja, pretpostavke i ovisnosti
- Specifični zahtjevi – pokrivaju funkcionalna i nefunkcionalna svojstva i zahtjeve sučelja
- Dodaci
- Indeksi

## PROCESI IZRADE ZAHTJEVA (*Requirements Engineering Processes*)

Cilj je stvaranje i održavanje dokumenta sa zahtjevima. Ovaj proces se sastoji od četiri pod procesa: studije izvedivosti, otkrivanje i analiza zahtjeva, pisanje specifikacije i validacija.

### Studija izvedivosti

Svi novi sustavi bi trebali početi sa studijom izvedivosti, a ona se temelji na poslovnim zahtjevima, okvirnom opisu sustava i njen rezultat je izvještaj koji preporuča da li se treba nastaviti s razvojem. Studija izvedivosti govori **da li sustav pridonosi ciljevima organizacije**, može li se sustav implementirati koristeći trenutnu konfiguraciju, dana sredstva i vremenska ograničenja. Ako se nastavlja s pisanjem studije izvedivosti daljnje aktivnosti su: procjena i sakupljanje informacija te pisanje izvještaja. U izvještaju se mogu predložiti i promjene budžeta, rasporeda...

### Otkrivanje i analiza zahtjeva (*Requirements elicitation and analysis*)

U ovoj aktivnosti razgovara se klijentima da se dobiju informacije o okruženju aplikacije, koje usluge sustav treba omogućiti, kakve performanse sustav treba imati, hardverska ograničenja... Sakupljanje zahtjeva je teško jer: klijenti često ne znaju što žele, imaju nerealne zahtjeve, izražavaju zahtjeve stručnim frazama, različiti tipovi klijenata mogu imati različite (čak i kontradiktorne) zahtjeve.

Procesne aktivnosti su: otkrivanje zahtjeva, klasifikacija i organizacija zahtjeva,

Zahtjevi se prvo otkrivaju, zatim se klasificiraju i određuju prioriteta. Često se javljaju konflikti u zahtjevima koji se pregovaranjem moraju riješiti. Na kraju se izrađuje dokumentacija. Taj postupak se često ponavlja više puta.

Otkrivanje zahtjeva je proces sakupljanja informacija (od klijenata, sličnih sustava, dokumentacije, poslovnog okruženja) iz kojih se poslije izvode korisnički i sistemski zahtjevi. Svaki izvor informacija predstavlja jednu perspektivu na sustav. Te perspektive (*viewpoint*) se mogu klasificirati i razlikujemo tri osnovna tipa:

- direktni (perspektiva osoba/sustava koji direktno komuniciraju sa sustavom) koji pružaju znanje o svojstvima i sučeljima sustava;
- indirektni (perspektiva osoba/sustava koji ne koriste sustav) koji pružaju informacije o organizacijskim zahtjevima i ograničenjima;
- *domain* (okolina utječe na zahtjeve).

Postoje i specifičnije perspektive: pružatelji ili primatelji usluga sustava; sustavi koji se direktno povezuju sa specficiranim sustavom; standardi koji se primjenjuju na sustav; nefunkcionalni zahtjevi, perspektiva osoblja koja razvija (možda imaju iskustva u radu sa sličnim sustavima), upravlja i održava sustav (najvjerojatnije će oni dati zahtjeve koji će olakšati održavanje)... Za bilo koji veći sustav postoji velik broj perspektiva i često ih je korisno hijerarhijski organizirati.

### Razgovori

Razlikujemo dva tipa razgovora: razgovori s predefiniranim pitanjima i slobodniji razgovori; u praksi je to često kombinacija tih dvaju. Razgovori su dobri jer na taj način razvojni tim dobiva šire razumijevanje o radu klijenata, njihovom odnosu prema sustavu i problemima s trenutnim načinom rada. Postoji problemi jer klijenti često koriste žargon karakterističan za posao koji obavljaju, često imaju zahtjeve koje im je teško dobro objasniti ili neke koje smatraju trivijalnim pa ih niti ne navode, ponekad ne žele otkrivati organizacijske probleme koje bi mogli utjecati na zahtjeve...

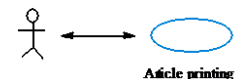
### Scenariji

Scenariji su opisi neke interakcije klijenta sa sustavom i bitan su dio agilnih metoda. Svaki scenarij počinje s okvirnim opisom interakcije koji se s vremenom dopunjuje. Scenarij može uključivati opise: što sustav i klijent očekuju kada scenarij počne, normalnog toka događaja, svega što može poći po „zlu“ i kako se to ispravlja, stanja sustava kada se scenarij završi. Obično se napiše tekst kojem se dodaju dijagrami.

### Primjeri korištenja (*use case*)

Primjer korištenja je tehnika bazirana na scenarijima koja je sastavni je dio UML notacije. Za opis se koriste stilizirani likovi – klijent i imenovane elipse kao klase interakcije. Primjer korištenja identificira interakciju sa sustavom i mogu se dokumentirati s tekstom ili mogu biti povezani s UML modelima. Skup primjera korištenja predstavlja sve moguće interakcije u zahtjevima.

Primjeri korištenja i scenariji su učinkoviti načini opisivanja zahtjeva s perspektive stvarnog korisnika sustava (*interactor viewpoint*).



## Etnografija

Etnografija je promatračka tehnika koja se može koristiti za bolje razumijevanje društvenih i organizacijskih potreba. Promatra se radni okoliš sustava i bilježe se informacije o stvarnim zadacima koje klijenti obavljaju. Klijentima je teško precizno artikulirati detalje njihovog posla, a socijalni i organizacijski faktori koji utječu na posao su često očiti samo nepristranom promatraču. Etnografija je naročito učinkovita u otkrivanju sljedećih tipova zahtjeva:

- Zahtjeva koji se izvode iz načina na koji ljudi zapravo rade, a ne iz načina na koje bi trebali raditi
- Zahtjeva koji se izvode iz suradnje i svijesti o aktivnostima drugih ljudi

Ovaj način je izrazito pogodan za otkrivanje zahtjeva krajnjih korisnika, ali nije prikladan za otkrivanje organizacijskih ili *domain* zahtjeva.

## Validacija zahtjeva

Validacija zahtjeva se bavi s time da dokaže da zahtjevi stvarno definiraju sustav koji klijent želi i jako je bitna jer greške koje se ne otkriju u ovom dijelu mogu biti jako skupe kada se otkriju za vrijeme dizajna ili razvoja. Rade se sljedeće provjere:

- Provjera validacije – da li navedeni zahtjev stvarno zadovoljava korisnikove potrebe. Napisani zahtjevi su najčešće kompromisi između zahtjeva većeg broja korisnika.
- Provjera konzistentnosti – provjerava se da li su zahtjevi kontradiktorni
- Provjera potpunosti – dokument mora sadržavati sve zahtjeve koji definiraju svu funkcionalnost i sva ograničenja.
- Provjerava se da li su zahtjevi realni u okvirima postojećih tehnologija, budžeta...
- Provjerava se da li se zahtjevi mogu potvrditi

Postoji više tehnika koje se mogu koristiti: pregledavanje zahtjeva – cijeli tim sistematski analizira zahtjeve; izrada prototipa; generiranje testnih slučajeva- zahtjevi se moraju moći testirati.

## Pregledavanje zahtjeva

Pregledavanje zahtjeva je proces koji uključuje i članove razvojnog tima i klijente u kojem oni zajedno pregledavaju zahtjeve. Provjerava se da li su zahtjevi konzistentni, potpuni, da li ih je moguće potvrditi, da li su razumljivi korisnicima, da li je podrijetlo zahtjeva moguće pronaći (*traceability*), da li je zahtjev prilagodljiv (ako se promjeni hoće li to uzrokovati promjene na cijelom sustavu).

## Upravljanje zahtjevima

Zahtjevi se često mijenjaju (kako korisnik više shvaća problem, nakon instalacije, promjenom organizacije...). Veliki sustavi imaju veliku korisničku bazu čiji korisnici često imaju proturječne zahtjeve, pa je krajnji rezultat često kompromis; ljudi koji plaćaju sustav i korisnici su rijetko iste osobe (pa se nakon instaliranja javljaju nove potrebe); poslovni i tehnički okoliš sustava se mijenja nakon instalacije i te promjene se moraju reflektirati na sustavu.

Upravljanje zahtjevima je proces razumijevanja i kontroliranja promjena zahtjeva sustava. Preporuča se praćenje nezavisnih zahtjeva i povezivanje ovisnih zahtjeva tako da se lakše može odrediti utjecaj promjene na sustav.

### Trajni i nestalni zahtjevi (*enduring and volatile*)

Trajni zahtjevi su relativno stabilni zahtjevi koji se izvode iz središnje aktivnosti sustava i koje se direktno odnose na poslovno okruženje sustava. Nestalni zahtjevi su zahtjevi koji će se najvjerojatnije mijenjati tijekom razvoja ili nakon što se sustav stavi u stvarni rad.

### Planiranje upravljanja zahtjevima

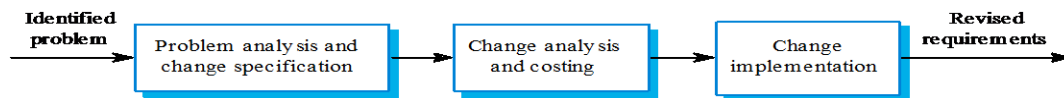
Tijekom faze upravljanja zahtjevima mora se odlučiti o:

- Identifikaciji zahtjeva – svaki zahtjev se mora jedinstveno identificirati
- Procesu izmjena zahtjeva – set aktivnosti koje procjenjuju utjecaj i cijenu promjena
- Načinu praćenja zahtjeva – definiraju odnose između zahtjeva, zahtjeva i dizajna sustava.  
Razlikujemo tri tipa praćenja:
  - praćenje izvora (zahtjev se poveže s klijentom koji ga je predložio),
  - praćenje povezanih zahtjeva,
  - praćenje dizajna (zahtjevi se povezuju s modulima koji ih implementiraju). Često se za to koriste matrice.
- Podršci za CASE alate. Postoje alati za spremanje zahtjeva, upravljanje izmjenama i praćenje.

### Upravljanje promjenama zahtjeva

Prednosti korištenja formalnih procesa za upravljanje promjenama zahtjeva su: prema svim promjenama se ponaša isto, promjene na dokumentu se rade na kontroliran način. Razlikujemo tri faze:

- analiza problema i promjena specifikacije – identificira se problem sa zahtjevima i radi se provjera ispravnosti te promjene;
- analiza promjene i cijene – promatra se koliko promjena utječe na sustav i koliko se mijenja cijena;
- implementacija promjene – modificira se dokument sa zahtjevima i prema potrebi dizajn i implementacija.



## SISTEMSKI MODELI

Sistemske modele se koriste kod pisanja specifikacije i to su grafički modeli koji opisuju poslovne procese, problem koji je potrebno riješiti, i sustav koji se razvija. Modeli se razvijaju s različitim perspektivama:

- Iz vani (okoline sustava) – modelira se kontekst ili okoliš sustava
- Ponašanja - modelira se ponašanje sustava
- Strukturalno – modelira se arhitektura sustava ili struktura podataka

Ti modeli su apstrakcija sustava. Razlikujemo tipove modela:

- Model toka podataka – kako se podaci obrađuju u različitim fazama sustava
- Kompozicijski model – kako se entiteti sustava sastoje od drugih entiteta
- Arhitekturni modeli – pokazuju podsustave od kojih se sastoji sustav
- Klasifikacijski modeli – koriste se dijagrami nasljeđivanja koji pokazuju kako entiteti imaju zajedničke karakteristike
- Model akcije-reakcije ili model prijelaza stanja – kako sustav reagira na podražaje

### Kontekstni modeli

Kod kontekstnih modela se određuju granice sustava – što leži van sustava kojeg rješavamo. Definira se kontekst u kojem se sustav nalazi i ovisnosti koje taj sustav ima o okolišu.

Arhitekturni modeli opisuju okoliš sustava, ali ne prikazuju odnose između drugih sustava i sustava koji se specifikira. Zbog toga se ti modeli često nadopunjuju s npr. procesnim modelima koji pokazuju aktivnosti koje sustav podržava.

### Modeli koji opisuju ponašanje sustava (*Behavioral models*)

Ovakvi modeli se koriste za opisivanje općeg ponašanja sustava. Razlikujemo dva tipa:

- modeli toka podataka (*data flow models*) (koji prikazuju kako sustav obrađuje podatke) – pogodni za modeliranje poslovnih sustava koje primarno pokreću podatci,
- modeli stanja sustava (kako sustav reagira na događaje) (*state machine model*) – pogodni za prikazivanje sustava za rad u realnom vremenu koje većinom pokreću događaji.

### **Modeli toka podatak** (*data flow models*)

Ovi modeli prikazuju kako sustav obrađuje podatke i prikazuju se pomoću: zaobljenih pravokutnika (obrada), pravokutnika (pohrana podataka), strelice (prijenos podataka između funkcija). Jednostavni su za shvaćanje i pokazuju funkcionalnu perspektivu gdje svaka transformacija prikazuje funkciju procesa.

### **Modeli stanja sustava** (*state machine models*)

Pokazuju stanja sustava i događaje koji uzrokuju prijelaz iz stanja u stanje. Često se koristi za modeliranje sustava za rad u realnom vremenu. Zaobljeni pravokutnici prikazuju stanja u kojima se navodi kratak opis što sustav radi u tom stanju. Imenovane strelice prikazuju događaj koji uzrokuje prijelaz sustava iz jednog stanja u drugo. Problem s ovim tipom modela je što broj stanja jako brzo počne rasti, ali tada se koriste „super stanja“ koja sadrže više stanja.

### **Podatkovni modeli** (*data models*)

Semantički podatkovni modeli definiraju logički oblik podatka koje obrađuje sustav. Najpoznatija tehnika je ERA (*Entity-Relation-Attribute*) koja prikazuje podatkovne entitete, njima pridružene atribute i odnose između tih entiteta. Nužno je uz model držati i detaljan opis atributa i entiteta npr. koristeći rječnik podataka (*data dictionary*) –lista imena kojima se pridružuje opis.

### **Objektni modeli** (*objektni modeli*)

Objektni modeli koji se razvijaju za vrijeme analize zahtjeva se mogu koristiti za predstavljanje i sistemskih podataka i njihove obrade, klasificiranje entiteta. Praktični su ako se koristi objektno orijentirano programiranje jer olakšavaju prijelaz s izrade zahtjeva na programiranje, ali problem je što nisu intuitivno razumljivi krajnjim korisnicima softvera. Entitet iz „stvarnog svijeta“ se modelira koristeći klase. Kod UML dijagrama ime modela se piše na vrhu, u sredinu idu atributi, a na dno metode vezane za objekt.

### **Modeli nasljeđivanja** (*Inheritance models*)

Taksonomija – klasifikacijska shema koja pokazuje kako je klasa povezana s drugim klasama preko zajedničkih atributa i metoda. Za prikaz taksonomije se koristi dijagram kod kojeg su klase organizirane prema hijerarhiji nasljeđivanja s najopćenitijim klasama na vrhu. Dizajniranje hijerarhije klasa nije jednostavno jer je potrebno detaljno poznavanje problema.

### **Object aggregation**

Objekti se često sastoje od drugih objekata. Kod UML dijagrama to se prikazuje pomoću romba na početku veze.

### **Modeliranje ponašanja objekta**

Za modeliranje ponašanja objekata nužne su metode tog objekta. Često se koriste UML dijagrami aktivnosti koji prikazuju jedan primjer korištenja (*use-case*). Objekti i korisnici su poredani na vrhu dijagrama. Imenovane strelice predstavljaju operacije. Slijed akcija ide od vrha prema dnu.

## Strukturirane metode

Strukturirana metoda je sistematičan način izrade modela sustava i pružaju okruženje za detaljno modeliranje sustava kao dijela opisivanja i analize zahtjeva. Većina takvih modela ima preferirani skup modela sustava i obično definiraju proces koji se može koristiti da bi se došlo do tih modela. Osiguravaju korištenje standardnih oznaka i izradu standardiziranih dokumenata. Neke slabosti ovih metoda su: ne pružaju učinkovitu podršku za razumijevanje i modeliranje nefunkcionalnih zahtjeva, ne koriste neke smjernice koje mogu pomoći korisniku da izabere metodu za određeni problem, proizvode previše dokumentacije, dokumentacija je detaljna pa je često teško razumljiva.

U praksi se ne ograničava samo na jednu metodu i često se koriste CASE alati npr.: editori dijagrama (kreiranje objektnih modela, podatkovnih modela... alati svjesni nacrtanih entiteta), alati za provjeru dizajna (pregledavaju dizajn i dojavljuju greške), podatkovni rječnici, alati za definiranje formi, generatori koda...