P7 Zasnova arhitekture

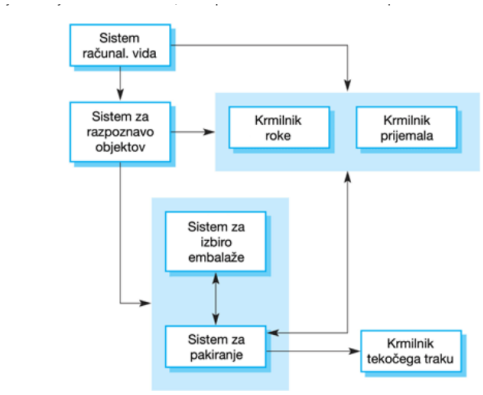
1 Uvod

Pri **zasnovi arhitekture** se ukvarjamo z razumevanjem, kako bi bilo treba organizirati programsko opremo in oblikovati celotno strukturo tega sistema.

Rezultat zasnove arhitekture je **načrt arhitekture**, ki opisuje, kako je sistem organiziran kot sklop sodelujočih komponent.

1.1 Agilnost in arhitektura

Pri agilnih pristopih se v zgodnji fazi ukvarjamo prav z načrtovanjem celovite arhitekture sistemov.



Slika 13.1: **Abstraktni model arhitekture** za sistem robotskega pakiranja izdelkov

Figure 1: Abstraktni model arhitekture za sistem robotskega pakiranja izdelkov

1.2 Posploševanje arhitekture

- **arhitektura v ožjem kontekstu** se ukvarja z arhitekturo posameznih programov
- **arhitektura v širšem kontekstu** zadeva arhitekturo kompleksnih poslovnih sistemov

1.3 Prednosti nedvoumne arhitekture

- pri **komuniciranju z deležniki** lahko arhitekturo uporabimo kot osrednjo temo razprave
- možno je izvesti **sistemsko analizo**, kjer preverimo, ali lahko zadostimo nefukncionalnim zahtevam
- arhitektura se lahko **ponovno uporabi** v različnih sistemih

1.4 Predstavitev arhitekture

Za dokumentiranje arhitekture programske opreme se najbolj pogosto uporabljajo enostavni, neformalni **blokovni diagrami**, ki prikazujejo entitete in razmerja.

1.5 Uporaba načrta arhitekture

- **način za lažjo razpravo o načrtu sistema** - *deležniki razumejo abstrakten pogled na sistem. O sistemu se je mogoče pogovarjati kot o celoti, ne da nas zmedejo podrobnosti*

- **način dokumentiranja načrtovane arhitekture** - *cilj je izdelati popoln model sistema, ki prikazuje različne komponente sistema, njihove vmesnike in njihove medsebojne povezave*

2 Odločitve pri zasnovi arhitekture

Načrtovanje arhitekture je **ustvarjalni proces**, zato se postopek razlikuje glede na vrsto sistema, ki se razvija.

Slika 13.2: **Odločitve** pri zasnovi arhitekture

Figure 2: Odločitve pri zasnovi arhitekture

2.1 Ponovna uporaba arhitekture

- Sistemi v isti problemski domeni imajo pogosto **podobno arhitekturo**, ki doraža domenske koncepte.
- Produktna linija aplikacij je zgrajena okrog jedrne arhitekture z različicami, ki zadovoljujejo določene zahteve strank.
- Arhitektura sistema je lahko zasnovana okoli enega od več arhitekturnih vzorcev ali stilov. Le-ti zajemajo bistvo arhitekture in jih lahko uporabimo na različne namene.

2.2 Arhitektura in značilnosti sistema

- **zmogljivost** - *lokalizacija kritičnih operacij, zmanjševanje medsebjne komunikacije, uporaba splošnih in ne podrobnih komponent*
- **varnost** - *uporaba večplastne arhitekture, lokaliziranje varnostno kritičnih funkcij v majhnem številu podsistemov*
- **razpoložljivost** - *vključevanje redundantnih komponent in mehanizmov za toleranco pri okvarah*
- **vzdržljivost** - *uporaba zamenljivih komponent na nizki ravni podrobnosti*

3 Pogledi na zasnovo arhitekture

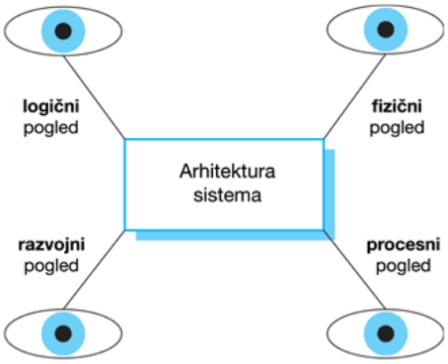
Sprašujemo se:

- kateri **pogledi** ali **perspektive** so **uporabne** pri oblikovanju in dokumentiranju sistema?
- katere **zapise** oz. **notacije** je treba uporabiti za opis arhitekturnih modelov?

Vsak akhitekturni vzorec **prikazuje samo en pogled** ali perspektivno sistema, in sicer:

- kako je sistem razčlenjen na module
- kako procesi med izvajanjem medsebojno delujejo ali
- različne načine, na katere so sistemske komponente razdeljene po omrežju

Pri načrtovanju in dokumentiranju je treba po navadi **predstaviti več pogledov na arhitekturo programske opreme**.



Slika 13.3: **Pogledi** na arhitekturo

Figure 3: Pogledi na arhitekturo

Logični pogled prikazuje *ključne abstrakcije* v sistemu v obliki *objektov oz. razredov - predstavljene entitete logičnega pogleda mora biti mogoče povezati s sistemskimi zahtevami.*

Procesni pogled prikazuje, kako je sistem sestavljen iz povezanih procesov v času izvajanja - *koristen za presojo ustreznosti sistema z vidika nefukncionalnih zahtev (zmogljivost, dosegljivost, . . .).*

Razvojni pogled prikazuje, *kako je programska oprema sestavljena pri razvoju.* Prikazuje razdelitev programske opreme *v komponente*, ki jih kasneje razvija posamezen razvijalec oz. razvojna skupna - *koristen za vodje razvoja in programerje*

Fizični pogled prikazuje *sistemsko strojno opremo* in *kako bo programska oprema porazdeljena po procesorjih v sistemu - koristen za sistemske inženirje, ki načrtujejo namestitev sistema*

3.1 Predstavitev pogledov na arhitekturo

Ker UML ne podpira abstrakcij, primernih za opis sistema na visoki ravni, so nastali opisni jeziki arhitekture (**ADL**), ki pa se ne uporabljajo pogosto.

4 Arhitekturni vzorci

Vzorec je sredstvo za predstavitev, izmenjavo in ponovno uporabo znanja.

Arhitekturni vzorec je stiliziran opis dobre načrtovalske prakse, ki je preizkušena v različnih okoljih.

4.1 Vozrec model-pogled-krmilnik (MVC)

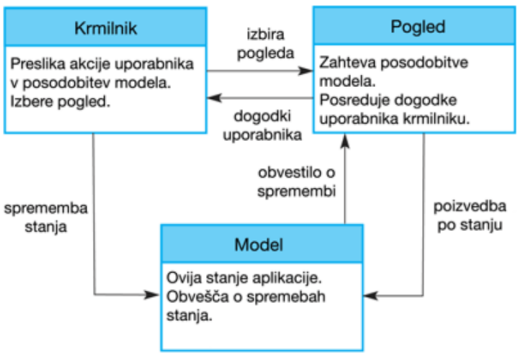
4.2 Večplastna arhitektura

Večplastna arhitektura se uporablja za načrtovanje povezovanja podsistemov.

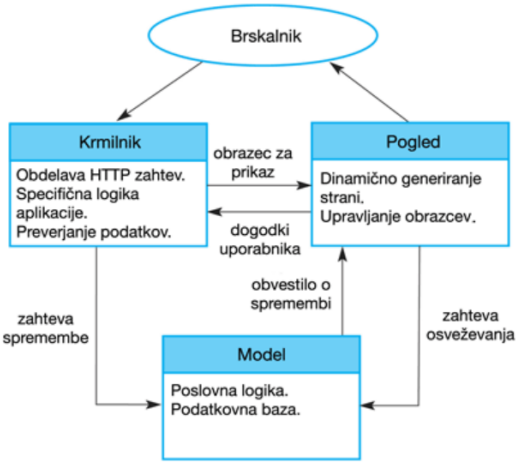
Tabela 13.1: Vzorec **model-pogled-krmilnik** (**MVC**)

Naziv	Model-pogled-krmilnik (MVC)
Opis	Ločuje predstavitev in interakcijo od sistemskih podatkov. Sistem je strukturiran v tri logične komponente, ki medsebojno sodelujejo. Komponenta **model** (**M**) upravlja sistemske podatke in povezane operacije na teh podatkih. Komponenta **pogled** (**V**) opredeljuje in upravlja z načinom predstavitve podatkov uporabniku. Komponenta **krmilnik** (**C**) upravlja uporabniško interakcijo (npr. pritiski tipk, kliki miške itd.) in te interakcije prenese v pogled in model, kot je prikazano na sliki 13.4.
Primer	Slika 13.5 prikazuje arhitekturo spletnega informacijskega sistema, ki je organiziran z uporabo vzorca MVC.
Kdaj se uporablja	Uporablja se takrat, ko obstaja več načinov prikaza in interakcije s podatki ali so prihodnje zahteve interakcij in predstavitev podatkov neznane.
Prednosti	Omogoča spreminjanje podatkov neodvisno od njihove predstavitve in obratno. Podpira predstavitev istih podatkov na različne načine s spremembami v eni predstavitvi, ki so potem prikazane v vseh.
Slabosti	Lahko vključuje dodatno kodo in poveča kompleksnost kode, medtem ko sta podatkovni model in interakcija preprosta.

Figure 4: Organizacija vzorca MVC



Slika 13.4: **Organizacija** vzorca MVC

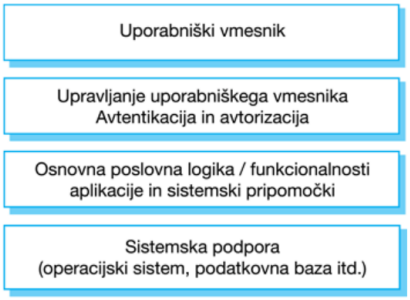


Slika 13.5: **Arhitektura spletne aplikacije** z uporabo vzorca MVC

Figure 5: Arhitektura spletne aplikacije z uporabo vzorca MVC

Tabela 13.2: Vzorec **večplastne arhitekture**

Naziv	Večplastna arhitektura
Opis	Sistem organizira v plasti s sorodno funkcionalnostjo. Sloj zagotavlja storitve sloju nad njim, tako da sloji najnižje ravni predstavljajo osnovne storitve, ki se bodo verjetno uporabljale v celotnem sistemu, kot je prikazano na sliki 13.6.
Primer	Večplastni model sistema za izmenjavo dokumentov o avtorskih pravicah v različnih knjižnicah, kot je prikazano na sliki 13.7.
Kdaj se uporablja	Uporablja se pri dopolnitvi obstoječih sistemov; pri razvoju več ekip, kjer je vsaka ekipa odgovorna za raven funkcionalnosti; če obstaja zahteva po večstopenjski varnosti.
Prednosti	Če ohranimo vmesnik, omogoča zamenjavo celotnih slojev. Za povečanje zagotovljivosti sistema se lahko v vsakem sloju omogoči redundantne zmogljivosti (npr. avtentikacija).
Slabosti	V praksi je zagotavljanje čiste ločnice med plastmi pogosto težavno in plasti na visoki ravni lahko neposredno vplivajo na plasti nižje ravni in ne zgolj na plast, ki je neposredno pod njo. Zaradi procesiranja storitve na več ravneh so lahko prisotne težave zaradi zmanjšane zmogljivosti.



Slika 13.6: **Splošna** večplastna arhitektura

Slika 13.7: **Arhitektura** iLearn digitalnega učnega okolja

Figure 6: Večplastna arhitektura

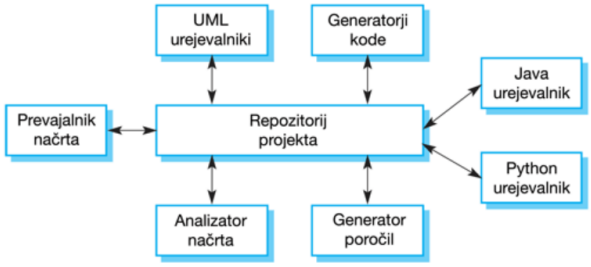
4.3 Arhitektura repozitorija

Podsistemi morajo med seboj izmenjavati podatke, kar lahko dosežemo na dva načina:

- skupni podatki se hranijo v **osrednji shrambi podatkov** ali **repozitoriju** in do njih lahko dostopajo vsi podsistemi
- vsak podsistem vzdržuje svojo **lastno bazo podatkov** in podatke izrecno posreduje drugim podsistemom

Tabela 13.3: Vzorec **repozitorija**

Naziv	Vzorec repozitorija
Opis	Vsi podatki v sistemu se upravljajo v centralnem repozitoriju, ki je dostopen vsem komponentam sistema. Komponente medsebojno neposredno ne vplivajo, ampak samo preko repozitorija.
Primer	Na sliki 13.8 je primer integriranega razvojnega okolja, kjer komponente uporabljajo repozitorij z informacijami o načrtu sistema. Vsako programsko orodje generira informacije, ki so nato na voljo za uporabo z drugimi orodji.
Kdaj se uporablja	Uporablja se pri sistemu, kjer nastajajo velike količine informacij, ki jih je treba shraniti za dlje časa. Uporabi se ga lahko tudi pri podatkovno usmerjenih sistemih, kjer vključitev podatkov v repozitorij sproži dejanje.
Prednosti	Komponente so lahko neodvisne - ni se jim potrebno zavedati obstoja drugih komponent. Sprememba, ki jo naredi ena komponenta, se lahko širi na vse komponente. Vse podatke je mogoče upravljati dosledno (npr. hkrati opravljene varnostne kopije), saj je vse na enem mestu.
Slabosti	Repozitorij predstavlja kritično točko odpovedi[62], tako da težave v repozitoriju vplivajo na celoten sistem. Po repozitoriju lahko obstaja neučinkovita organizacija komunikacije. Težavna je lahko tudi porazdelitev skladišča na več računalnikov.



Slika 13.8: Arhitektura repozitorija za **integrirano razvojno okolje**

Figure 7: Arhitektura repozitorija za integrirano razvojno okolje

4.4 Arhitektura odjemalec/strežnik

Arhitektura odjemalec/strežnik je model porazdeljenega sistema, ki prikazuje, kako so podatki in obdelava podatkov porazdeljeni po različnih komponentah - *se lahko izvaja na enem računalniku.*

4.5 Arhitektura cev in filter

Pri **arhitekturi cev in filter** funckionalne transformacije obdelujejo svoje vhode, da vrnejo rezultat na izhodu. Kadar imamo opravka z zaporednimi transformacijami, gre za **serijski sekvenčni model**, ki se pogosto uporablja v sistemih za obdelavo podatkov.

5 Arhitekture aplikacij

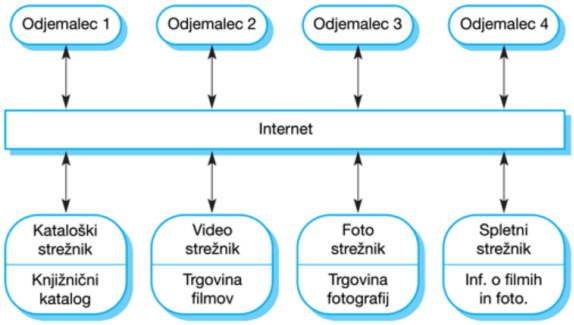
Generična arhitektura aplikacij je arhitektura za tip programske opreme, ki se lahko konfigurira tako, da ustvarimo sistem, ki ustreza posebnim zahtevam.

5.1 Uporaba arhitektur aplikacij

- kot izhodišče za zasnovo arhitekture

Tabela 13.4: Vzorec **odjemalec/strežnik**

Naziv	Vzorec odjemalec/strežnik
Opis	V arhitekturi odjemalec/strežnik je funkcionalnost sistema organizirana v storitve, kjer se vsaka storitev dostavi iz ločenega strežnika. Odjemalci so uporabniki teh storitev in dostopajo do strežnikov, ki jih uporabljajo.
Primer	Slika 13.9 je primer knjižnice filmov in video posnetkov, ki je organizirana kot sistem odjemalec/strežnik.
Kdaj se uporablja	Uporablja se, ko je treba dostopati do podatkov v skupni bazi podatkov iz različnih lokacij. Ker je mogoče strežnike podvojiti, se lahko uporabi tudi ob spremenljivi obremenitvi sistema.
Prednosti	Glavna prednost vzorca je sposobnost porazdelitve strežnikov po omrežju. Splošna funkcionalnost (npr. storitev tiskanja) je lahko dostopna vsem odjemalcem in je ni treba izvajati v vseh storitvah.
Slabosti	Vsaka storitev predstavlja kritično točko odpovedi, ki je dovzetna za napade na zavrnitev storitve ali napako strežnika. Zmogljivost je lahko nepredvidljiva, ker je odvisna od omrežja in sistema. Pojavijo se lahko težave z upravljanjem, če so strežniki v lasti različnih organizacij.

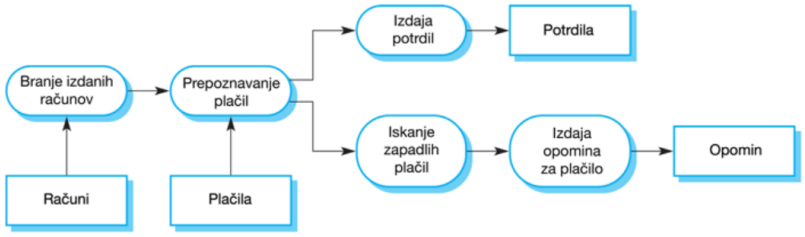


Slika 13.9: Arhitektura odjemalec/strežnik za **knjižnico filmov**

Figure 8: Arhitektura odjemalec/strežnik za knjižnico filmov

Tabela 13.5: Vzorec **cev in filter**

Naziv	Vzorec cev in filter
Opis	Obdelava podatkov v sistemu je organizirana tako, da je vsaka komponenta za obdelavo (**filter**) diskretna in izvaja eno vrsto pretvorbe podatkov. Podatki pri obdelavi tečejo (kot v **cev**i) iz ene komponente v drugo.
Primer	Slika 13.10 predstavlja primer sistema cevi in filtrov, ki se uporablja za obdelavo računov.
Kdaj se uporablja	Običajno se uporablja v aplikacijah za obdelavo podatkov (tako na osnovi paketne obdelave kot tudi transakcij), kjer se vhodi obdelujejo v ločenih stopnjah, da se ustvarijo povezani rezultati.
Prednosti	Enostaven za razumevanje in podpira ponovno uporabo transformacij. Slog dela se ujema s strukturo mnogih poslovnih procesov. Evolucija z dodajanjem transformacij je enostavna. Lahko se izvaja kot zaporedni ali sočasni sistem.
Slabosti	Oblika prenosa podatkov med posameznimi transformacijami mora biti dogovorjena. Vsaka transformacija mora razčleniti njen vnos in razčleniti njegov izhod na dogovorjeno obliko. To poveča obremenitev sistema in lahko pomeni, da je nemogoče ponovno uporabiti funkcionalne transformacije, ki uporabljajo nezdružljive podatkovne strukture.



Slika 13.10: Primer arhitekture cev in filter

Figure 9: Arhitektura cev in filter

- kot kontrolni sistem pri načrtovanju
- kot način organiziranja dela razvojne skupine
- kot sredstvo za ocenjevanje komponent za ponovno uporabo
- kot slovar pri pogovoru o vrstah aplikacij

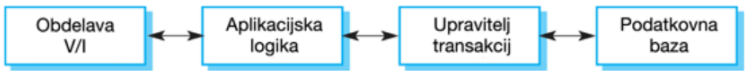
5.2 Vrste aplikacij

- **Sistem za obdelavo podatkov** je podatkovno usmerjen in obdeluje podatke v paketih brez izrecnega posredovanja uporabnika med obdelavo.
- **Sistem za obdelavo transakcij** je osredotočen na podatke, ki obdelujejo zahteve uporabnikov in posodabljajo informacije v podatkovni bazi sistema.
- **Sistem za obdelavo dogodkov** predstavlja aplikacijo, kjer so sistemska dejanja odvisna od interpretacije dogodkov iz okolja sistema.
- **Sistem za obdelavo jezika** je aplikacija, kjer so uporabniške zahteve v formalnem jeziku, ki ga sistem obdeluje in interpretira.

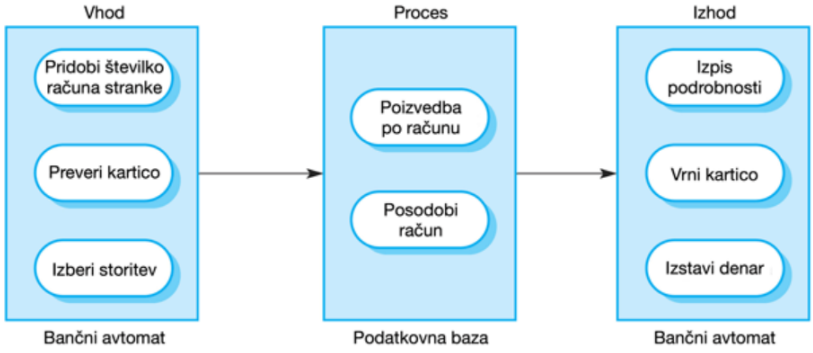
5.3 Sistem za obdelavo transakcij

Sistem za obdelavo transakcij procesira zahteve uporabnikov po informacijah iz podatkovne baze ali zahteve za posodobitev podatkovne baze.

Z vidika **uporabnika** velja:
"***Transakcija*** *je vsako skladno zaporedje operacij, ki izpolnjuje cilj (npr. poišči čase odhodov letal iz Ljubljane do Madrida)*



Slika 13.11: Struktura sistema za obdelavo transakcij



Slika 13.12: Arhitektura aplikacije za sistem bankomata

Figure 10: Arhitektura aplikacije za sistem bankomata

5.4 Informacijski sistem

Informacijski sistem omogoča nadzorovan dostop do velike količine informacij in skoraj vedno gre za spletne sisteme, kjer je uporabniški vmesnik implementiran v spletnem brskalniku.

5.5 Spletni informacijski sistemi

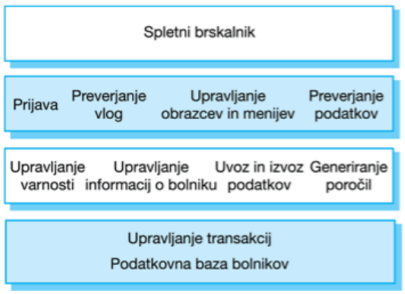
Sistem za upravljanje informacij in virov so zdaj ponavadi spletni sistemi, kjer so uporabiški vmesniki implementirani v spletnem brskalniku.

Slika 13.13: Večplastna arhitektura informacijskega sistema

Figure 11: Večplastna arhitektura informacijskega sistema

Spletni sistemi so pogosto implementirani v obliki večplastne odjemalec/strežnik arhitekture:

- **spletni strežnik** je odgovoren za vso komunikacijo z uporabnikom in je implementiran v spletnem brskalniku
- **aplikacijski strežnik** je odgovoren za izvajanje logike aplikacije, kot tudi za shranjevanje informacij in izpolnjevanje zahtev po pridobivanju informacij
- **podatkovni strežnik** skrbi za prenos informacije v podatkovno bazo in iz nje, obravnava tudi upravljanje transakcij



Slika 13.14: Arhitektura sistema Mentcare

Figure 12: Arhitektura sistema Mentcare

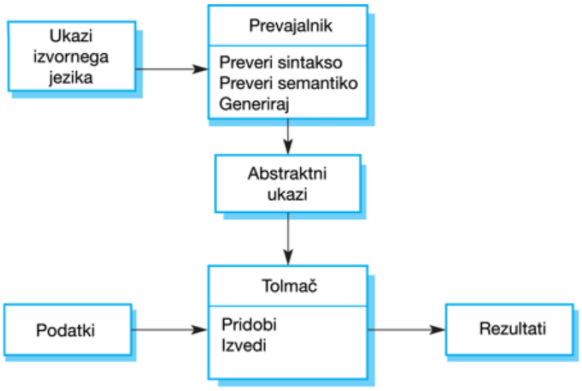
5.6 Sistem za obdelavo jezika

Sistem za obdelavo jezika na vhodu sprejme naravni ali umetni jezik in to pretvori v drugo predstavitev tega jezika.

5.6.1 Komponente prevajalnika

Komponente prevajalnika, ki jih uporabimo v obliki arhitekture repozitorija ali arhitekture cev in filter, so:

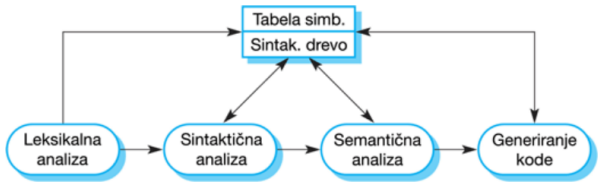
- **leksikalni analizator**
- **tabela simbolov**
- **sintaktični analizator**
- **razčlenitveno sintaktično drevo**
- **semantični analizator**
- **generator kode**



Slika 13.15: Arhitektura sistema za obdelavo jezika

Figure 13: Arhitektura sistema za obdelavo jezika

Slika 13.16: **Arhitektura repozitorija** sistema za obdelavo jezika



Slika 13.17: **Arhitektura cev in filter** sistema za obdelavo jezika

6. Zaključne ugotovitve

- **Arhitektura programske opreme** opisuje organizacijo sistema programske opreme.
- **Odločitve pri zasnovi arhitekture** so odločitve o vrsti aplikacije, porazdelitvi sistema, uporabljenih arhitekturnih slogih itd.

- Arhitektura se lahko **dokumentira iz več različnih pogledov**, kot so logični ali fizični pogled in pogled razvoja ali procesa.

- S pomočjo **arhitekturnih vzorcev** ponovno uporabimo znanje o generičnih sistemskih arhitekturah. Arhitekturo opisujejo in pojasnjujejo, kdaj jo lahko uporabimo, ter izpostavijo njene prednosti in slabosti.

- **Model arhitekture aplikacijskih sistemov** nam pomaga razumeti in primerjati aplikacije, preveriti skladnost aplikacijskega sistema in ovrednotiti obsežne komponente z vidika ponovne uporabe.

- **Sistem za obdelavo transakcij** je interaktivni sistem, ki omogoča dostop do podatkov v podatkovni bazi in njihovo spreminjanje s strani številnih uporabnikov.

- **Sistem za obdelavo jezika** se uporablja za prevajanje besedila iz enega jezika v drugega in za izvajanje navodil, podanih v jeziku vnosa. Vključuje prevajalnik in abstraktni stroj, ki izvrši ustvarjeni jezik.