**Strukturiranje sustava:** Bavi se dekompozicijom sustava u komunicirajuće podsustave. Dizajn arhitekture se često prikazuje kao blok dijagram koji pokazuje pregled strukture sustava. Također se mogu razviti posebni modeli koji pokazuju kako podsustavi dijele podatke, kako su distribuirani i sučelje među njima.

**Dijagrami „boxa“ i linija:** Veoma su apstraktni – ne pokazuju prirodu veza komponenti, niti vidljiva svojstva podsustava. Korisni su u komunikaciji sa korisnikom i za planiranje projekta.

**Odluke arhitekturalnog dizajna:** Kako je dizajn arhitekture kreativan proces, razlikuje se ovisno o tipu sustava koji se razvija. Ipak, određeni broj odluka je zajednički za sve dizajn procese.

**Ponovno korištenje arhitekture:** Sustavi u istoj domeni često imaju slične arhitekture koje pokazuju koncepte domene. Aplikacije se razvijaju oko arhitekturalne jezgre sa varijacijama koje zadovoljavaju određene zahtjeve korisnika.

**Stilovi arhitekture:** Model arhitekture nekog sustava može odgovarati generičkom arhitekturalnom modelu ili stilu. To može pojednostavniti problem definiranja arhitekture sustava. Ipak, većina velikih sustava su heterogeni i ne prate jedan stil arhitekture.

**Prednosti i nedostatci REPOSITORIJ MODELA**: **Prednosti**: učinkovit način izmjene velike količine podataka; podsustavi se moraju brinuti kako se proizvode podaci; imaju centralizirani menadžment (backup, security); model dijeljenja se objavljuju kao shema repozitorija. **Nedostatci**: podsustavi se moraju dogovoriti oko repozitorij modela podataka – kompromis; evolucija podataka je teška i skupa; nema područja za određene načine menadžmenta; teško je efikasno distribuirati.

**Klijent-server model:** **Prednosti:** Distribucija podataka je neposredna; efektivno koristi mrežne sustave; može zahtjevati jeftiniji hardver; lako je dodati nove servere ili unaprijediti postojeće servere. **Nedostatci:** Nema modela dijeljenja podataka, pa podsustavi koriste različitu organizaciju podataka; izmjena podataka može biti neefikasno; redundantni menadžment u svakome serveru; nema središnjeg registra imena i servisa pa je teško saznati koji serveri i servisi su dostupni.

**Modularna dekompozicija:** Druga strukturalna razina gdje su podsustavi podijeljeni u module. Postoje dva modela:Objektni model (sustav se rastavlja u komunicirajuće objekte),Cjevovod ili model toka podataka (sustav se rastavlja u funkcionalne module koji transformiraju ulaze u izlaze).Istovremene odluke se uobičajeno odgađaju dok se moduli ne implementiraju.

**Objektni modeli:** Strukturira se sustav kao skup slabo vezanih objekata sa dobro definiranim sučeljima. Objektno orijentirana dekompozicija se bavi identificiranjem klasa objekata, njihovih atributa i operacija. Kada se implementiraju, objekti se stvaraju iz ovih klasa i nekoh moela kontrole korištenog za koordiniranje operacija objekata.

**Prednosti objektnog modela:** objekti su slabo vezani pa se njihova implementacija može mijenjati bez utjecanja ostalih objekata; objekti mogu prikazivati stvarne entitete; objektno orijentirani jezivi su u širokoj upotrebi. Ipak, promjene sučelja objekata mogu uzrokovati probleme i nekad je teško kompleksne entitete prikazat kao objekte.

**Funkcijski orijentirani cjevovod:** Funkcionalne transformacije obrađuju ulaze da bi proizveli izlaze. Varijacije na ovaj pristup je čest. Kada su trasnformacije sekvencijalne, ovo je sekvencijalni model koji se koristi u sustavima procesiranja podataka. Ovakav cjevovod nije prigodan za interaktivne sustave.Prednosti: Podržava ponovno korištenje transformacija; intuitivna organizacija za komunikaciju sa korisnikom; lako je dodati nove transformacije; relativno je lako implementirati kao istovremen ili sekvencijalan sustav. Ipak, zahtjeva zajednički format za prijenos podataka kroz cjevovod i teško podržava interakciju baziranu na *eventovima*.

**Case reference model (model referentnih slučajeva):** Servisi repozitorija podataka, Servisi integracije podataka,Servisi upravljanja zadacima**,** Servisi poruka**,**Servisi korisničkih sučelja.

**Karakteristike distribuiranih sustava:**Dijeljenje resursa**,** Otvorenost, Konkurentnost Skalabilnost,Tolerancija na grešku.

**Nedostatci distribuiranih sustava:** Kompleksnost, *Security***,** Upravljivost**,** Nepredvidljivost.

**Korištenje klijent-server arhitekture:** **Mršavog klijenta**: (*Legacy* sustavi gdje je nepraktično odvajanje procesiranja aplikacija i upravljanja podataka, Računski neosjetljive aplikacije kao kompajleri sa malo ili nimalo upravljanja podacima, Aplikacije neosjetljive na podatke sa malo ili nimalo procesiranja aplikacija).**Debelog klijenta**: (Aplikacije gdje je aplikacijsko procesiranje omogućeno pomoću *off the shelf* proizvoda, Aplikacije gdje je nužno procesiranje podataka neosjetljivo na računanje, Aplikacije sa relativno stabilnom funkcionalnošću za krajnjeg korisnika, koja se koristi sa dobro utemeljenim upravljanjem sustava).**Three-tier**: (Velike aplikacije sa stotinama i tisućama klijenata, Aplikacije gdje su i podaci i aplikacija promjenjivi, Aplikacije gdje su podaci iz više izvora integrirani).

**Prednosti i gdje se koristi distribuirano objektna arhitektura**: **Prednosti**: dopušta dizajneru sustava odgađanje odluka o tome gdje i kako bi usluge trebale biti pružene; veoma je otvorena arhitektura koja dopušta dodavanje novih resursa po potrebi; sustav je fleksibilan i skalabilan; moguće je rekonfigurirati sustav dinamično uz objekte koji prolaze mrežom kako se zahtjeva.**Korištenje**: kao logički model dopušte strukturiranje i organiziranje sustava, kao fleksibilan pristup implementaciji klijent-server sustava (i klijent i server su realizirani kao distribuirani objekti koji komuniciraju kroz zajednički komunikacijski framework).

**CORBA standardi:** objektni model za aplikacijske objekte,*object request broker* koji upravlja zahtjevima i uslugama objekta**,**skup općenitih objektnih usluga koje koriste mnogim distribuiranim aplikacijama**,** skup zajedničkih komponenti koje se izrađuju povrh tih usluga

**Object Request Broker (ORB):**ORB se baci komunikacijom objekata. Upoznat je sa svim objektima u sustavu i njihovim sučeljima. Koristeći ORB, pozivajući objekt vezuje IDL dio koji definira sučelje pozivajućeg objekta. Pozivanje ovoga dijela rezultira pozivima ORB-u koji onda poziva potrebni objekt preko objavljenog IDL kostura koji povezuje sučelje implementaciji usluge. **Komunikacija međe ORB-ovima**: ORB-ovi nisu uobičejeno odvojeni programi, ali su skup objekata u nekoj bibiloteci (*library*) koji su povezani sa aplikacijama kada se razviju. ORB-ovi rukuju komunikacijom izmđu objekata koji se izvršavaju na istom uređaju. Više ORB-ova može biti dostupno i svako računalo u distribuiranom sustavu će imati vlastiti ORB.

**Servisno orijentirane arhitekture:**

Bazira se na ideju eksterno pruženih usluga– web usluga.Web usluga je standardan pristup omogužavanja pristupa i dostupnosti ponovnom korištenju određene komponente. Generička usluga je čin ili izvedba koju jedna strana nudi drugoj. Iako procesi mogu biti vezabu uz fizički produkt, izvedba je u osnovi nedodirljiva i ne rezultira vlasništvom nekog od faktora proizvodnje. Zato je pružanje usluge neovisno o palikaciji koja koristi određenu uslugu.

**LIBSYS arhitektura:** Bibliotekarski sustav LIBSYS je primjer informacijskog sustava.Sloj korisničke komunikacije(LIBSYS login komponenta, Upravitelj formi i upita, Upravitelj ispisa). Sloj povrata informacija(Distribuirano pretraživanje, Povrat dokumenta, Upravitelj prava, Računovodstvo).

**Implementacija slojevitih sustava:** Svaki sloj se može implementirati kao komponenta velikih razmjera koja se izvodi na odvojenom serveru. Ovo je najčešće korištena arhitektura za web bazirane sustave. Na jednom uređaju, srednji slojevi su implementirani kao odvojeni program koji komunicira sa bazom podataka preko svoj API-a. *Fine grain* komponente unutar slojeva mogu biti implementirani kao web usluge.

**Konkurentni objekti:** Priroda objekta kao samosadržanog entiteta ih čini prikladnim za konkurentnu implementaciju. Model prosljeđivanja poruka se mže implementirati izravno ako se objekti izvršavaju na odvojenim procesorima u distribuiranom sustavu.

**Serveri i aktivni objekti: Server** su objekti implementirani kao paralelni proces sa ulaznim točkama koje odgovaraju objektnim operacijama. Ako nema poziva prema serveru, objekt se sam suspendira i čeka na iduće zahtjeve za uslugom. **Aktivni objekti** su objekti implementirani kao paralelni procesi i objekt može sam mijenjati svoje unutrašnje stanje, a ne vanjskim pozivom. Aktivnim objektima se određenim operacijama može mijenjati atribute, ali se mogu također sami ažurirati unutrašnjim operacijama.

**Java niti (threadovi):** Niti u Javi su jednostavne konstrukcije za implementiranje konkurentnih objekata. Moraju uključivati metodu *run()* koju pokreće Javin *run-time* sustav. Aktivni objekti obično sadrže beskonačnu petlji, tako da se stalno izvode izračuni.

**Proces objektno orijentiranog dizajna:** Strukturirani dizajn procesi uključuju razvoj određenog broja različitih modela sustava. Oni zahtjevaju mnogo truda prilikom razvoja i održavanja ovih modela, i za male sustave nisu isplativi. Ipak, za velike sustave koji su razvijeni velikom grupom raličitih dizajn modela su osnovni mehanizam komunikacije.

**Faze procesa** (ključne aktivnosti): Definirati kontekst i modove koji su korisni za sustav, Dizajnirati arhitekturu sustava, Prepoznati osnovne objekte sustava, Razviti dizajn modele, Specificirati sučelja objekta.

**Kontekst sustava i modeli korištenja:** Potrebno je razviti razumijevanje veza između softvera koji se dizajnira i njegove vanjske okoline. **Kontekst sustava** je statički model koji opisuje ostale sustave u okolini. Koristi model posustava za prikaz ostalih sustava. **Model korištenja** je dinamički model koji opisuje kako sustav interaktira sa svojom okolinom, i koristi *use-caseove* za opisivanje interakcija. *Use case* modeli pokazuju svojstva sustava kao elipse i entitete koji komuniciraju kao čovječuljke.

**Arhitekturalni dizajn:** Jednom kad su veze između sustava i njegove okoline shvaćene, te informacije se koriste za dizajniranje arhitekture sustava. U arhitekturalnom modelu ne bi trebalo biti više od sedam entiteta.

**Identifikacija objekata (ili objektnih klasa):** Ovo je najteži dio objektno orijentirano dizajna. Ne postoji siguran način za identifikaciju, nego se oslanja na sposobnosti, iskustvo i poznavanje domene dizajnera sustava. Objektna identifikacija je iterativan proces, i rijetko se uspije iz prvog pokušaja. **Pristupi identifikaciji**:Gramatički pristup baziran na prirodnom opisu jezika sustava**,** Pristup baziran na krajnjim stvarima aplikacijske domene**,** *Behavioral* pristup i identificiranje objekata baziranih na onome što sudjeluje u ponašanju**,** Analiza bazirana na scenariju: objekti, atributi i metode u svakom scenariju se identificiraju.

**Daljni objekti i usavršavanje objekta:** Koristi se znanje o domeni za identifikaciju više objekata i operacija. Objekte dijelimo na aktivne i pasivne, ovisnoda li sudjeluju u akcijama ili samo čekaju (pasivno) na neki zahtjev.

**Dizajn modeli:** prikazuju objekte i objektne klase i veze među ovim entitetima. Statički model opisuje statičku strukturu sustava u smislu objektnih klasa i veza. Dinamički model opisuje dinamičke interakcije među objektima.Primjeri:Modeli podsustava**,** Slijedni modeli**,** *State machine* modeli, Ostali modeli uključuju Use-case modele, agregacijske modele, generalizacijske modele..

**Modeli podsustava** pokazuju kako je dizajn organiziran u logički povezane grupe objekata. U UML-u, ovo se pokazuje korištenjem paketa - enkapsuliranih konstrukcija. Ovo je logički model. Stvarna organizacija objekata u sustavu može biti različita.

**Slijedni modeli**  pokazuju slijed interakcija objekata. Objekti su organizirani horizontalno preko vrha. Vrijeme se prikazuje okomito pa se mofeli iščitavaju od vrha prema dnu. Interakcije su prikazane kao označene strelice, različiti stilovi strelica predstavljaju različite tipove interakcija. Tanki pravokutnik predstavlja vrijeme kada je objekt kontrolirajući objekt sustava.

Grafikon stanja pokazuje kako objekt odgovara na različite zahtjeve za uslugom i kako se mijenjaju stanja ovisno o tim zahtjevima.

**Specifikacija sučelja objekta:** Sučelja objekta se moraju specificirati tako da se objekti i ostale komponente mogu paralelno ravijati. Dizajneri bi trebali izbjegavati razvoj prikaza sučelja, i trebali bi to skriti u sami objekt. Objekti mogu imati više sučelja koja su gledišta na dane metode. UML koristi dijagrame klasa za specifikaciju sučelja, ali se također može koristiti Java.

**Evolucija dizajna:** skrivanje informacija u objektiva znači da promjene načinjene objektu ne utječu na druge objekte na nepredvidljiv način.