***Arhitektura softvera***

Neke definicije:

* fundamentala organizacija sistema
* pogled na sistem sa visokog nivoa
* detaljno razumevanje sistema koje su razvili eksperti
* bitne stvari-sta god to bilo

Jasno je da je arhitektura nešto što stoji nasuprot impelemntacije kada govorimo o delu pisanja samog koda, nešto što treba da prethodi pisanju koda i da to budu stvari gde ćemo odlučiti kako da organizujemo softver koji razvijamo.

Zašto nam je potrebna organizacija softvera:

|  |  |
| --- | --- |
| ***Arhitektura građevinskih objekata*** | ***Arhitektura softvera*** |
| 1. Pomoćni objekat  * Gde on nije u vezi sa svojom okolinom i neće smetati nekom drugom objektu * **Možemo odmah početi sa samom izgradnjom** * Izgradnja ovakvih objekata se svede na prethodno iskustvo onih koji učestvuju u izgradnji | 1. Jednostavan softver  * Možemo lako napraviti bez prethodne organizacije samog procesa njegovog pisanja |
| 1. Kuće za stanovanje (kompleksniji objekat)  * **Može** se izvesti **bez** arhitekte * Ukoliko angazujemo **arhitektu** prvo da isplanira i isprojektuje imaćemo **veću korist** od te kuce, bolje ćemo moći da iskoristimo prostor, imacemo bolji kvalitet življenja * Glavni posao arhitekte je da što bolje iskoristi resurse i da u skaldu sa njima oganizuje prostor najbolje moguće | 1. Softver relativne složenosti  * Može se napisati i bez organizacije * Ali šanse da će rezultati biti bolji ako razmotrimo i arhitekturu * Obično u toku tih razmatranja otkrivamo i neke nove potencijalne probleme kojih ranije nismo bili svesni |
| 1. Izuzetno kompleksne građevine  * **Nemoguće** napraviti **bez** prethodnog **projektovanja** * Pored toga zahteva znanja i iz mnogih drugih disciplina * Ali arhitekte upravljaju organizacijom i vide celinu sastavljenu od svih delova i aspekata objekta | 1. Izuzetno složen softver  * Nemoguće je tako nešto realizovati bez temeljnog razmatranja arhitekture * Veoma često u tome pomažu ljudi koji se time konkretno bave |

***Arhitektonski stilovi***

Kako možemo da organizujemo arhitekturu visokog nivao, kako možemo da organizujemo neke delove softverskog rešenja tako da oni medjusobno saradjuju:

1. Monolitna aplikacija
   * Aplikacije koje:
     + sadrže sve što je neophodno za rad **na jednom mestu**
     + pokreću se sa jednog računara i služe jednom korisniku
     + imaju sve što im je potrebno za **manipulaciju podacima** (unos,čuvanje,obrada)
     + poseduju **korisnicki interfejs** pomoću kojeg upravljamo tim mehanizmima
   * Mogu ali ne moraju da budu jednostavne, možemo imati i složene koje pružaju mnoštvo funkcionalnositi
   * Primeri: softver za tabelarne proračune, kancelarijske aplikacije, aplikacije za obradu slika, crtanje, obradu videa,...
   * Prednosti :
     + Korisnici mogu samostalno da ih instaliraju i koriste (**jednostavni**)
     + Dovoljan je **jedan računar** da bi obavljali posao što je i ideja personalnih računara

* Mana:
  + Nad tim podacima može da radi **samo jedan čovek**
* Može se relativno ublažiti tako sto držimo te resurse na nekoj deljenoj lokaciji,kada je pristup lakši ali i dalje je u jednom trenutku teško ili nemoguće sinhronizovati da više ljudi radi u isto vreme

1. Klijent-server
   * Dvoslojna organizacija (arhitekrura) aplikacije
   * Jedan server kom u isto vreme može pristupati **više klijenata** koji mogu da budu **na različitim računarima** što je češće (tada klijent i server komuniciraju pomoću računarske mreže), ali ponekada može biti i na jednom računaru
   * Svi klijenti mogu u isto vreme da rade nad istim podacima i vide rezultate do kojih su došli drugi korisnici na istom poslu
   * Prednost: imamo mogućnost da **više ljudi** radi na istom poslu
   * Mana :
     + **neophodan** nam je dodatni računar koji je **server**
     + neophodna **infrastruktura** (mrežu, računari) koja takođe zahteva neko planiranje
     + neko ko **održava** server, mrežu, čitavu infrastrukturu
   * Podvrste ove organizacije:
     + **Klijent** poseduje delo sa korisničkim interfejsom koji prikazuje podatke, služi za unos podataka i koji **obrađuje** podatke, a server služi kao mesto gde se podaci čuvaju tj. gde se sinhronizuju međusobne izmene koje čine klijenti
       - Prednost : kada je veoma velika količina podataka koja se prenosi između klijenta i servera
     + Klijent poseduje samo korisnički interfejs koji prikazuje podatke i pribavlja akcije korisnika, a **server** osim čuvanja, i obrada podataka, prima korisničke akcije i na osnovu njih **obradjuje** podatke
       - Mana : server možemo previše opteretiti
       - Prednost: klijent može biti jednostavan (hardverski i/ili softverski i/ili za upravljanje)
2. Troslojna arhitektura
   * **Dodatna** **podela** uloga
     + Korisnički interfejs - prezentacija i prikupljanje korisničkih akcija
     + Sloj za obradu - gde se implemetnira poslovna logika aplikacije
     + Sloj za čuvanje podataka
   * Ovako možemo bolje da **upravljamo raspodelom** resursa
     + Deo za obradu može biti na zasebnim serverima u odnosu na servere gde čuvamo podatke (komuniciraju putem mreže)
     + Može i neki sloj da bude raspoređen na više računara
       - sloj za obradu ili za čuvanje ako je obrada podataka intenzivnija onda obrađujemo na više servera a čuvamo na jednom
       - ako je količina podataka velika onda i za čuvanje podataka dodajemo još servera i proširujemo resurse koji su nam na raspolaganju
     + Moramo i softverski da implementiramo tako da je moguće da se ovi delovi izvršavaju na više računara koji će komunicirati
3. Višeslojna arhitektura
   * Svaki od slojeva možemo dodatno izdeliti po nameni i svaki od tih delova možemo izvršavati na različitim lokacijama
   * Na svakom delu mogu da rade različiti ljudi koji su specijalizovani za neki vid razvoja softvera (delom za čuvanje oni koji su specijalizovani za bazu podataka, korinsički interfejs oni koji su skloniji dizajenerskoj strani,..)
4. Servisno orjentisane arhitekture (SOA)
   * Servisi su komponente nezavisne od naših aplikacija koje pozivamo da obave neki posao i vrate rezultat
     + Lokalni - da se izvršavaju na istom računaru
     + Udaljeni - da se izvršavaju preko mreže
     + Web servisi - ako je ta mreža web mreža
   * Ideja ovih arhitektura je da sistem organizujemo kao **mnoštvo servera** od kojih **svaki implementira neki deo** **funkcionalnosti**
   * Pretpostavka je da ćemo imati već raspoložive servise koji je neko već impelementirao koji nam mogu obavljati te funkcionalnosti
   * Želimo da pronađemo ta nezavina rešenja koja nam daju neku funkcionalnost a mi moramo da ih **orgaizujemo da međusobno razumeju** rezultate odnosno delove ukupnih rezultata kako bi na kraju dobili ono što je ukupno potrebno (takođe ti servisi se mogu obraćati drugim servisima za nešto što je njima potrebno)
   * Servisi mogu biti na različitim sistemima, mogu biti udaljeni, različitih proizvođača,..
   * Onda nam je glavni posao kako naći servis koji nešto implementira što nama treba i zbog toga se u SOA uvode **brokeri** što su mehanizmi za pronalaženje servisa određene funkcionalnosti

***Načini implementacije servisa***

1. Rest
   * Koristi se za implementaciju **web servisa** jer koristi neke osnovne **koncepte http protokola** i nastao je u vreme kada i http specifikacija
   * Namera je da se na kraju učini moguće da na **automatizovan** ili **poluatutomatizovan** omogućimo da servisi **sarađuju jedni sa drugima**, postoje mehanizmi za to ali postoje i načini da se to radi ručno, pitanje je sta je za koju primenu primerenije
2. Mikro servisi
   * Podvrste SOA
   * Insistira se na tome da ovi servisi budu **sitno granulirani**, da svaki servis obavlja **samo jednu funkcionalnost**, da i ti mikro servisi budu sto je moguće jednostavniji
   * Ideja da budu **jednostavni** ali da svoj posao **obavljaju najbolje moguće**
   * Prednost :
     + **Lakše** ćemo se opredeliti da ih **implementiramo**, bolje ćemo moći da **sagledamo** šta servisi treba da rade i bolje ćemo moći da ih implementiramo
   * Mana:
     + **Teško** je **naterati ljude** koji godinama **rade kompleksnije** stvati da prave nešto jednostavno i da ne lepe neke stvari na mikro servis koje bi trebale da budu zaseban servis
     + Postojanje više komponenti uvek usložnjava **organizaciju**
     + Postavljaju se **veći zahtevi na infrastrukturu** u ovkakvim resenjima jer ima puno komponenti koje treba da komuniciraju i razmenjuju rezultate
3. Softverski agenti
   * U svim dosadašnjim slučajevima mi smo nešto pozivali ili je nešto pozivalo
   * Ideja da to bude neki **softver** koji je **autonoman** na neki način, gde ćemo napraviiti agenta koji ima neki zadatak koji radi za nas nesto, ali ga onda pustimo da radi svoj posao relativno autonomno
   * Agenti:
     + Obično **nemaju** neku **interakciju** sa korisnikom
     + Imaju pristup nekim podacima i mogu da **obradjuju** te podatke
     + Mogu da **reaguju na dogadjaje**
     + Mogu da budu da se **prilagođavaju promenama** u okolini (intaligentni agenti)
     + Mogu da zarad svog cilja **saradjuju** sa drugim **agentima**
     + Mogu biti **mobili** da putuju sa sistema na sistem i tamo prikupe deo podataka koji im je neophodan da bi na kraju dosli do nekih rezultata
4. Računarstvo u oblaku (cloud computing)
   * Nije zaseban arhitektonski stil već kombinacija raznih prethodno spomenutih softverskih arhitektura koje su upakovane tako da možemo deo ili svu infrastruktue da prepusitmo nekom drugom, način na koji deo ili cela infrastruktura ili deo softvera može neko da nam proda ili mi da prodamo
   * Ovim je tehnologija poboljšana jer je više korišćena pa se više problema uočava i rešava
   * Takođe mi prepuštamo infrstrukturu i podatke nekom drugom

***Softverski obrasci***

* Do sada smo govorili o tome kako organizovati arhitekturu u većim celinama, za šta će svaka celina biti zadužena
* Kada to odlučimo i dalje svaki deo za sebe treba implementirati i dizajnirati
* Motivacija za postojanje šablona : možemo razvijati softver da on bude ponovo iskoristiv ali razvijanje na takav način je dodatna oteđavajuća okolnost, teško je i nekada zahteva i iskustvo da bi bili svesni da neki problemi postoje
* Zato korisitmo iskustvo drugih
* Ponovna iskoristivost nam omogućava da ne moramo ispočetka implementirali nešto što je već implementirano , i takođe se možemo osloniti na to da su već korišćena i testirana
* Kako ponovno koristiti te stvari
  + Na nivou koda ( biblioteke, radni okviri,..)
  + Na nivou dizajna softvera (dizajn obrasci)
* Softverski obrazac
  + nešto što bi opisalo problem koji se često pojavljuje u našem okruženju u nekoj oblasti razvoja softvera
  + opisuje suštinu rešenja tog problema
  + na osnovu toga možemo da implementiramo svoje konkretno rešenje za svoj problem
* Možemo nacin rešavanja primeniti milion puta a nikada ne rešimo problem na idnetičan način
* Da bi učinili rad sa softverskim obrascima moguć potreban nam je način da ih struktuiramo, kako da nadjemo obrazac koji nam pomaže u odredjenom trenutku, zbog čega se ustalio format dokumentovanja obrazaca
* Svaki obrazac ima:
  + Naziv - omućava da se lako referenciramo na taj obrazac kada komuiciramo sa nekim, kada pričamo sa nekim kako to naše rešenje izgleda
  + Opis problema koji taj obrazac može da reši i kontekst u kom se taj obrazac može primeniti, kontekst u kom se javalja problem. Često se ovde navodi i lista preduslova, kada je moguće i preporučljivo primeniti obrazac, šta su ograničenja koja pokazuju da neki obrazac nije dobro primeniti u konkretnom slučaju
  + Opis resenja – navode se elementi rešenja sa nazivima, veze tih elemenata, opisuje kako saradjuju i za šta je odgovoran koji element rešenja, obično apstraktno opisano kako bi moglo da se primeni u različitim konkrentim kontekstima često se daje i neki primer izvornog koda na nekom jeziku kako bi se rešenje moglo bolje razumeti jer je i sam kod svojevrsna formalna dokumentacija, kod je izvršiv i može se tumaciti samo na jedan način ako razumemo kod i jezik
  + Opis posledica - šta dobijamo takvim rešenje a šta gubimo – najkorisnijij deo , prenosi tuđa iskustva
* Ideja šablona je da to kako bi rešili problem u početku može izgledati jednostavno ali ponekada postoji potreba za time da se dodatno zakomplikuje smo rešenje iako to na prvi pogled ne deluje potrebno ali to radimo zato što ćemo možda dobiti nečto u budućnosti i možda ćemo sagledati neki problem kog nismo bili svesni do tada i možda će tada neki malo kompleksniji način rešavanja na duže staze biti bolji

***Klasifikacija sablona***

* Kada su ljudi počeli da definisati šablone, počeli su se pojavljivati rečnici šablona, kolekcije,...
* Kalasifikacija omogućavaju da lakše pronađemo šablon, tj rešenje problema
* Podela prema nameni
  + - Kreacioni - načini instaciranja objekata
    - Strukturalni – načini organizacije modela podataka
    - Obrasci ponašanja se bave ponašanjem raznih komponenti koje imamo u rešenju
* Prema sagledavaju
  + na nivou klasa
  + na nivou instanci tj objekata
* Ovaj nacin razmisljanja o dizajnu softvera kasnije od strane mnogih i kritikovan što je posledica nepravilnog korišćenja šablona, neki koriste šablon za sve a to nije njihova prava namena, oni su namenjeni dokumentaciji i opisu problema koji se uvek javljaju u raznim oblastima, pored tih problema nezavisno od razvoja aplikacije, imaćemo uvek i neke probleme koji su specifični za dati cilj na kom trenutno radimo

Primer kreacionog obrasca - opisno

* **Singleton**
* Ako imamo potrebu da postoji **samo jedna instanca** klase u sistemu i da se onda **svi** obraci **obraćaju toj** jednoj **instanci**
* Razlozi
  + sve komponente koriti jedan isti loger za beleženje događaja
  + želimo da imamo samo jednu konekciju na bazu koju koriste sve komponente
* Želimo da ta jedna instnca **bude dostupna svim klijntima**, na **jendnostavan** način
* Sama klasa singleton ima **privatni konstruktor** – to znači da tu klasu ne možemo spolja instancirati na onaj klasičan način new Singleton van klase nemamo pristup konstruktor pa ne možemo tako da instanciramo, tako smo rešili zahtev da ne može da se napravi više instanca klase na sistemu, klasi se pristupa preko javnih metoda koja se zove npr getInstance
* Imamo privatno polje instance i tipa je singleton, proverimo svaki put kada neko kaze get instance da li je već napravljena ako nije napravimo je, i vratimo instancu onom ko poziva, ukoliko je pozovemo a već imamo instancu ova metoda samo vraća tu instancu i tako svi koji koriste singleton dobijaju pristup toj jednoj instanci
* Metoda koja radi neku akciju, singletonOperation
* **Prendost** : laka implementacija i lako se korisiti,lako iz bilo koje druge klase doći do te jedne instance koja nam treba
* **Mana** : oni unose velike muke tokom **testiranja**, ne samog singletona već testiranje svih ostalih komponenti koje zavise od njega gde šta god drugo da testiramo što koristi njegove usluge, moramo obezbediti da on postoji i u testnom okruženju što nije jednostavno
* Takođe i dizajn uvezujemo preko singletona i sve ostale komponente su vezane za njega i bez njega ne možemo pa je razvoj ostalih komponenti, da budu ponovno iskoristive, daleko teži
* Zato se danas singleton smatra **anti patternom**, način rešavanja za koji su uvideli da nisu dobri, koji nose teškoće kojih nismo svesni na početku ali posle se pojave , izuzetno je korisno što i takve stvari imamo dokumentovane pa na osnovu takvog iskustva učimo
* Ljudi su skloni kritikama pa mnogi nešto smatraju anti paternima, ali zato dobro moramo proučiti sve, da li će te mane nama da umanje vrednost rešenja

Primer arhitektonskog obrazac

* Model View Controller
* Način **organizacije** komponenti po **osnovu njihove uloge**
* Koristi se u aplikacijama koje imaju grafičko korisničko okruženjo, koje na vizuelan način predstavalju neke podatke i imaju mogućnost interakcije sa korisnikom
* Elementi:
  + **Model** – domenski spcifična predstava strukture podataka, ono što modeluje podatke iz realnog sistema
  + **Kontroler** – prikuplja akcije korisnika i na osnovu njih ažurira model, tj poziva metode modela koje odrade neku stvar i tako kontroler menja model
  + **View** – prikazuje podatke na neki način i ima neku mogućnost da bude površina za prikupljanje korisničkog unosa koji će biti prosleđen kontroleru na koji će kontroler reagovati, ne menja direktrno podatke
* Kontroler samo na osnovu korisničkih akcija tumači šta su akcije i ažurira model
* Ni model ni kontroler se ne bave vizuelnom predstavom podataka
* Ovakva organizacija nam **razgraničava uloge**, omogućava da **svaka** može biti **implementirana** **na najbolji način**, razdvaja poslovnu logiku od koda koji je zadužen za prikaz podataka
* Omogućava da imamo **više pogleda**, koji prikazuju **isit model** iste podatke na različite načine, i na bilo kom od tih prikaza ukoliko korisnik **nešto izmeni** ili se sa strane nešto izmeni kontroler će da odregajue na to i da ažurira pogled i biće prikazano **na svim pogledima u isto vreme**