**Arhitekture raspodijeljenih sustava**

Oblikovanje arhitekture programske potpore

◼ Donošenje ispravnih odluka u oblikovanju arhitekture

◼ Stilovi arhitekture programske potpore

◼ Modeli i pogledi na arhitekturu

◼ Procjena arhitekture (metoda kompromisa te analiza troškova i dobiti)

◼ Vrednovanje i sukladnost arhitekture s principima oblikovanja (12 + 5)

◼ Dokumentiranje arhitekture

**Arhitekturni stil** - familija sustava definirana sličnim oblicima strukturne organizacije i opisana jasno definiranim rječnikom komponenti i konektora te pripadajućim topološkim ograničenjima

**Raspodijeljeni sustav -** sustav sastoji se od skupa nezavisnih računala, povezanih mrežom i posredničkim programima, koja omogućuje koordinaciju rada i dijeljenje resursa, tako da korisnici doživljavaju sustav kao jedinstveni računalni sustav

Izazovi

◼ Složenost

◼ Sigurnost

◼ Upravljanje (engl.

Manageability)

◼ Nepredvidljivost (engl.

Unpredictability)

Svojstva

◼ Dijeljenje resursa

◼ Otvorenost (heterogenost)

◼ Konkurentnost (engl.

Concurrency)

◼ Skalabilnost

◼ Neosjetljivost na greške

◼ Transparentnost

Primjeri >>> Klijent – poslužitelj - najčešća

◼ Sustavi ravnopravnih sudionika (engl. Peer-to-Peer) P2P

◼ Srodne socijalne mreže (engl. affinity communities)

◼ Kolaborativno izračunavanje (engl. collaborative computing)

◼ Slanje poruka u stvarnom vremenu (engl. Instant Messaging)

◼ Upravljanje složenim sustavima (automobil, tramvaj, …)

Problemi oblikovanja raspodijeljenih sustava

◼Kako se alociraju i pokreću funkcije poslužitelja?

◼ Kako se definiraju i šalju parametri između klijenta i poslužitelja?

◼ Kako se rukuje neuspjesima (pogreškama) u komunikaciji?

◼ Kako se postavlja i rukuje sa sigurnošću?

◼ Kako klijent pronalazi poslužitelja?

◼ Koje strukture podataka koristiti i kako rukovati s njima?

◼ Koja su ograničenja u istovremenom radu dijelova raspodijeljenog

sustava?

◼ Kako se uopće skupina komponenata usuglašava oko zajedničkih

pitanja?

**ARHITEKTURA KLIJENT - POSLUŽITELJ**Poslužitelj (*engl. server*)

◼ Program koji dostavlja uslugu drugim programima koji su spojeni na njega preko komunikacijskog kanala.

Klijent (*engl. client*):

◼ Program koji pristupa poslužitelju (ili više njih) tražeći uslugu

◼ Poslužitelju mogu pristupiti mnogi klijenti istovremeno

Primjeri : The World Wide Web, E-mail, Network File System -NFS, Transaction Processing System, Remote Display System, Communication System, Database System

Sekvenca aktivnosti:

1. Poslužitelj započinje s radom. Spajanje još nije dozvoljeno.

2. Poslužitelj dozvoljava spajanje („sluša”) i čeka na dolazak klijentskog zahtjeva.

3. Klijenti započinju s radom i obavljaju razne operacije

4. Kada klijent pokuša spajanje na poslužitelja, poslužitelj mu to omogući (ako želi).

5. Poslužitelj čeka na poruke koje dolaze od spojenih klijenata.

6. Kada pristigne poruka nekog klijenta poslužitelj poduzima akcije kao odziv na tu poruku.

7. Klijenti i poslužitelj nastavljaju s navedenim aktivnostima sve do odspajanja ili prestanka rada.

**Funkcionalnost poslužitelja**

1. Inicijalizacija poslužitelja
2. Započinje slušati klijentska spajanja
3. Rukuje sljedećim tipovim događaja koje potiču klijenti:
   1. Prihvaća spajanje
   2. Odgovara na poruke
   3. Rukuje odspajanjem klijenta
4. Može prestati slušati i mora čisto završiti rad

**Funkcionalnost klijenta**

1. Inicijalizira klijenta
2. Inicijalizira spajanje na poslužitelja
3. Šalje poruke
4. Rukuje sljedećim tipovima događaja koje potiče poslužitelj:
   1. odgovara na poruke
   2. rukuje odspajanjem od poslužitelja
5. Mora čisto završiti rad

**Komunikacijski protokoli**Klijent i poslužitelj u komunikaciji razmjenjuju poruke uporabom dva jezika: jezik klijenta i jezik poslužitelja >>> ta dva jezika i pravila konverzacije čine zajedničkim imenom **protokol**

**Primjeri** : Internet Protocol (IP), Transmission Control Protocol (TCP)

**Oblikovanje arhitekture klijent-poslužitelj**

* Oblikuj temeljne poslove poslužitelja i klijenta.
* Odredi kako će se posao raspodijeliti.
  + tanki (manje posla na toj strani) nasuprot debelog klijenta.
* Oblikuj detalje skupa poruka koje se razmjenjuju --- komunikacijski protokol.
* Oblikuj mehanizme:

Inicijalizacije || Rukovanja spajanjima || Slanja i primanja poruka || Završetka rada

* U oblikovanju koristi princip “Povećaj uporabu postojećih rješenja i investicija”

**Alternative arhitekturi klijent-poslužitelj**

+ Implementacija jednog programa na jednom računalu koji obavlja sve poslove

+ Računala nisu spojena u mrežu, već svako računalo obavlja svoj posao odvojeno

+ Ostvariti neki drugi mehanizam (osim klijent-poslužitelj) kako bi računala u mreži razmjenjivala informacije

**Prednosti arhitekture klijent-poslužitelj**

1. Posao se može raspodijeliti na više računala (strojeva)
2. Klijenti udaljeno pristupaju funkcionalnostima poslužitelja
3. Klijent i poslužitelj mogu se oblikovati odvojeno
4. Oba entiteta mogu biti jednostavnija
5. Svi podaci mogu se držati na jednom mjestu (na poslužitelju)
6. Obrnuto, podaci se mogu distribuirati na više udaljenih klijenata i poslužitelja
7. Poslužitelju može istodobno pristupiti više klijenata
8. Klijenti mogu ući u natjecanje za uslugu poslužitelja (a i obrnuto)

**Rizici arhitekture klijent-poslužitelj**

**1. Sigurnost**

◼ Prijetnje **klijentu**: virusi, trojanci i drugi maliciozni programi

◼ Prijetnje **poslužitelju**: DoS (engl. Denial of Service, service overloading, message overloading), prisluškivanje komunikacije

**2. Potreba za adaptivnim održavanjem**

>>> potrebno je osigurati da sva programska potpora bude:

• kompatibilna prema unatrag (engl. backwards) i

• prema unaprijed (engl. forwards) – vidi princip “Planiraj zastaru”,

• te kompatibilna s drugim verzijama klijenata i poslužitelja

**OBJEKTNI RADNI OKVIR KLIJENT-POSLUŽITELJ(OCSF)**

Metoda oblikovanja arhitekture klijent-poslužitelj temeljena na ponovnoj i višestrukoj uporabi

Pravila uporabe:

1. ne mijenjati apstraktne razrede u OCSF
2. kreirati podrazrede
3. konkretizirati metode u podrazredima
4. ponovo definirati (engl. override) neke metode u podrazredima
5. napisati kod koji kreira instance i inicira akcije

**Vrednovanje arhitekture klijent-poslužitelj kroz principe oblikovanja**

1. Podijeli pa vladaj: Podjelom sustava na klijenta i poslužitelja je uspješan način optimalne podjele
2. Povećaj koheziju: Poslužitelj osigurava kohezijski spojenu uslugu
3. Smanji međuovisnost: Uobičajeno je da postoji samo jedan komunikacijski kanal preko kojega se prenose jednostavne poruke
4. Povećaj apstrakciju: Odvojene raspodijeljene komponente su dobar način povećanja apstrakcije.
5. Povećaj ponovnu uporabivost: Skup operacija <<hook>> omogućuje dodavanje opcijskih funkcionalnosti
6. Povećaj uporabu postojećeg: često je moguće pronaći odgovarajući radni okvir (engl. framework) temeljem kojega se oblikuje raspodijeljeni sustav. Međutim, klijent-poslužitelj arhitektura je često specifična s obzirom na primjenu.
7. Oblikuj za fleksibilnost: Raspodijeljeni sustavi se često vrlo lako mogu rekonfigurirati dodavanjem novih poslužitelja ili klijenata.
8. Planiraj zastaru: Paziti na tehnologije.
9. Oblikuj za prenosivost: Klijenti se mogu oblikovati za nove platforme bez promjene poslužiteljske strane.
10. Oblikuj za ispitivanje: Klijenti i poslužitelji mogu se ispitivati neovisno
11. Oblikuj konzervativno: U kod koji rukuje porukama mogu se ugraditi stroge provjere (npr. rukovanje iznimkama).
12. Oblikuj po ugovoru: po potrebi može se ugraditi u komunikaciju

**VIŠERAZINSKA ARHITEKTURA**

Višerazinska organizacija

1. Organizira program u razinama
2. Svaki sloj je grupa modula koja nudi kohezivni skup usluga
3. Upotreba mora biti jednosmjerna
4. Klijenti i poslužitelji se organiziraju u više razina
5. Svaka razina pruža uslugu razini iznad
6. Svaka razina oslanja se na razinu ispod

◼ Razina zatvara (skriva, enkapsulira) skup usluga i

implementacijske detalje niže razine o kojoj ovisi

**Trorazinska klijent-poslužitelj arhitektura**

+ Znatno bolje promovira skalabilnost i mogućnost jednostavnije modifikacije

+ Nastoji otkloniti neke nedostatke klasične klijent-poslužitelj arhitekture, a posebice nastoji povećati performanse, raspoloživost i sigurnost

Općenito **trorazinska arhitektura sadrži**

◼ korisničku razinu ◼ logičku ili poslovnu razinu ◼ podatkovnu razinu

npr. Frontend Backend Baza podataka

**Tanki i debeli klijent**

**Sustav tankog klijenta**

1. klijent je oblikovan da bude što je moguće manji i jednostavniji
2. većina posla obavlja se na poslužiteljskoj strani
3. oblikovnu strukturu klijenta i izvršni kod jednostavno se preuzima preko računalne mreže

**Sustav debelog klijenta**

1. što je moguće više posla delegira se klijentima
2. poslužitelj na taj način može rukovati s više klijenata

**Prednosti i nedostaci arhitekture s više razina**

**Prednosti:**

1. oblikovanje temeljem više razine apstrakcije
2. podupire povećanje i poboljšanje sustava
3. promjena na jednoj razini utječe samo još na razinu ispod i iznad
4. podupire ponovnu uporabu, prenosivost i sl.

**Nedostaci:**

1. teško je odrediti optimalno preslikavanje odgovornosti na razine
2. ponekad se izračunavanje i funkcionalnosti sustava ne mogu razbiti na razine
3. ako se želi poboljšati performanse, mora se preskakati ili „tunelirati” kroz razinu

**POSREDNIČKA I ZASTUPNIČKA ARHITEKTURA**

**Posrednička arhitektura**

Logička razina u trorazinskoj arhitekturi može se prošireno promatrati kao posrednička razina definirana međuslojem (engl. middleware)

**Posrednička razina** (engl. middleware) je sveobuhvatna programska podrška koja omogućava uzajamno djelovanje aplikacija bez potrebe za poznavanjem i kodiranjem svih operacija nužnih za implementaciju usluge

**Posrednička razina** ima definiran skup rutina/sučelja (API)

Postoje tri vrste ovih arhitektura:

1. Transakcijski usmjerena (komunikacija s bazama podataka)
2. Zasnovana na porukama (pouzdana, asinkrona komunikacija)
3. Objektno usmjerena (komunikacija između raspodijeljenih/udaljenih objekata)

Primjeri: DotNET, EJB, CORBA, MPI

Interakcija komponenata u ovoj raspodijeljenoj arhitekturi temelji se na nekom

mehanizmu udaljenog pozivu procedura – RPC

**Prednosti i nedostaci posredničke i zastupničke arhitekture**

Nedostaci

* smanjena efikasnost
* veća osjetljivost na pogrešk

Prednosti

* transparentnost lokacija;
* izmjenjivost i proširivost komponenti
* prenosivost
* interoperabilnost različitih sustava
* ponovna uporaba

Principi oblikovanja i posredničko/zastupnička arhitektura

1. Podijeli pa vladaj:

udaljeni objekti mogu biti neovisno oblikovani

1. Povećaj ponovnu uporabu:

često je moguće oblikovati udaljene objekte tako da ih i drugi sustavi mogu koristiti

1. Povećaj uporabu postojećeg:

mogu se koristiti objekti koje su drugi oblikovali

1. Oblikuj za fleksibilnost:

posrednik (zastupnik) se može ažurirati

objekti se mogu zamijeniti

1. Oblikuj za prenosivost:

može se oblikovati klijente za novu platformu i još uvijek pristupati postojećim zastupnicima i udaljenim klijentima na drugim platformama

11. Oblikuj konzervativno: može se rigorozno provjeriti nedvosmisleno ponašanje i obilježja udaljenih objekata

**USLUŽNO USMJERENA ARHITEKTURA**

Uslužno usmjerena arhitektura organizira primjenski program (cjelovitu aplikaciju) kao kolekciju usluga koje međusobno komuniciraju uporabom dobro definiranih sučelja i protokola

**Model web uslužne arhitekture**

1. Oglašavanje, otkrivanje selekcija i uporaba web usluge

+ tehnologije zasnovane na XML-u

1. Simple Object Access Protocol (SOAP)

+ razmjena poruka Web usluga

1. Web Service Description Language (WSDL)

+ definira web uslugu

1. Universal Description, Discovery, and Integration (UDDI)

+ API za registar usluga

**SOAP (*Simple Object Access Protocol*) komunikacijski protokol**

• neovisan o platformi

• zasnovan na XML-u

• namijenjen komunikaciji aplikacija

• zamjenjuje ranije razvijene protokole u Internet okruženju

***Hypertext Transfer Protocol* (HTTP**) je temeljni protokol u raspodijeljenoj Web mreži namijenjen razmjeni hipertekstualnih dokumenata koji sadrže hiperlinkove na druge izvore

HTTP se temelji na zahtjevu-odzivu u klijent-poslužitelj modelu

**WSDL** (Web Service Description Language) je XML baziran jezik za strojno čitljiv opis funkcionalnosti Web usluge (WSDL datoteka), klijentski program čita **WSDL datoteku** kako bi odredio koje operacije ta usluga nudi

**UDDI - *Universal Description, Discovery and Integration*** je specifikacija registra

Web usluga i drugih elektroničkih i neelektroničkih usluga

Ponuditelji Web usluga koriste **UDDI** za oglašavanje svojih usluga a korisnici

usluga za njihovo otkrivanje.

>>> Ponuditelj opisuje uslugu s **WSDL**, a korisnik se spaja sa **SOAP**

**REST arhitekturni stil**

REST je arhitekturni stil za specifikaciju ograničenja u komunikaciji komponenata koje čine web-usluge

1. Arhitektura je klijent – poslužitelj, s jasno definiranim sučeljem
2. Klijent šalje identifikator resursa URI i operaciju nad resursom
3. Zahtjev klijenta mora sadržavati potpunu informaciju za razumijevanje
4. zahtjeva (ne pamti se kontekst na poslužitelju između dva zahtjeva)
5. Informacija mora biti samodostatna
6. Poslužitelj vraća reprezentaciju stanja resursa koji se traži npr. u formatu HTML, XML, JSON
7. Ta je reprezentacija koncepcijski potpuno odvojena od interne reprezentacije resursa (npr. resurs iz baze podataka)

**Mikroservisi**

**Temeljni problem:**

evolucija i proširenje sustava (skaliranje) s promjenom organizacije ili (dijela) poslovnog modela

**Tradicijsko rješenje:**

Trorazinska arhitektura programske potpore s monolitnom podatkovnom razinom

**Problemi rasta:**

Složena komunikacija, vrlo veliki broj linija koda, povećana međuovisnost (narušen princip oblikovanja #3), nema modularnosti.

Arhitektura zasnovana na **mikroservisima** je uslužno usmjerena arhitektura s ciljem rješavanja problema rasta monolitne strukture

**Mikroservisi** su mali servisi jednostavne funkcionalnosti koji rade u odijeljenim procesnim okruženjima i s raspodijeljenim bazama podataka ali zajedno oko jedne određene poslovne domene

**Svaki mikroservis**:

* Izvodi svoje vlastite procese i ne zavisi o posebnoj platformi.
* Može se pisati u bilo kojem programskom jeziku.
* Jednostavno se klonira ili uklanja ovisno o potrebi.
* Može koristiti različite tehnologije repozitorija podataka.
* Zahtjeva minimalno središnje upravljanje.

**Principi oblikovanja i uslužna arhitektura**

* **Podijeli pa vladaj:** Cijeli primjenski program sastoji se iz neovisno oblikovanih komponenata - usluga.
* **Povećaj koheziju:** Web usluge su strukturirane kao slojevi i imaju dobru funkcionalnu koheziju.
* **Smanji međuovisnost:** Web zasnovani primjenski programi su slabo vezani i oblikovani su spajanjem raspodijeljenih komponenata.
* **Povećaj ponovnu uporabu:** Web usluga je posebice značajno ponovno uporabiva komponenta.
* **Povećaj uporabivost postojećeg**: Web zasnovane usluge su oblikovane ponovnom uporabom postojećih web usluga.
* **Planiraj zastaru:** Zastarjele usluge mogu se zamijeniti novim implementacijama bez utjecaja na primjenske programe koje ih koriste.

**Radni okviri za web razvoj**

**Oblikovni obrasci**

Predstavlja općenito rješenje problema koje se može ponovno koristiti u često pojavljujućim problemima oblikovanja programske potpore

Cilj: ponovna uporaba rješenja oblikovanja

◼ Značajno unapređuje objektno usmjereno oblikovanje.

**Klasifikacija oblikovnih obrazaca**

**Stvaralački (engl. creational)**

* usredotočeni na načine stvaranja razreda i objekata prema zadanim kriterijima i na upravljiv način
* Abstract factory, Builder, Factory method, Prototype, SingletonStructural

**Strukturni (engl. structural)**

* usredotočeni na organizaciju razreda i objekata i omogućavanje nove funkcionalnosti
* Adaptor, Bridge, Composite, Decorator, Façade, Flyweight, Proxy

**Ponašajni (engl. behavioral)**

* usredotočeni na identifikaciju i ostvarenje komunikacije razreda i objekata
* Chain of responsibility, Command, Interpreter Iterator, Mediator, Mementor, Observer, State, Strategy, Template Method, Visitor

**Specifični**

◼ J2EE Patterns

◼ CORBA

**Elementi oblikovnog obrasca**

* **Naziv** - Izražajno kratko ime
* **Struktura** - Skup dijagrama razreda i objekata koji opisuju obrazac
* **Uporaba** - Opis namjene obrasca
* **Motivacija** - Primjer problema i način rješavanja uporabom obrasca
* **Posljedice** - Consequences Opis uvjeta obrasca, prednosti i ustupaka
* **Implementacija** - Opis implementacije
* **Sinonim** - Lista sinonima za isti obrazac
* **Primjenjivost -** Popis slučajeva pogodnih za uporabu
* **Elementi** - Opis razreda i objekata te njihovih odgovornosti
* **Primjer** koda - Primjer uporabe u konkretnom programskom jeziku

**Player-Role pattern**

Omogućava modeliranje promjena uloga objekta i uvođenje više uloga

**Singleton pattern**

osigurava samo jednu instancu razreda

**Observer pattern**

pojednostavljuje međusobne veze razreda iz različitih modula

**Delegation pattern**

omogućuje uporabu implementirane metode iz druge hijerarhije

**Adapter pattern**

mogućuje indirektnu ugradnju razreda iz neke druge hijerarhije

**Façade pattern**

pojednostavljuje pogled na razrede u složenom podsustavu (paketi)

Immutable patternstvaranje razreda čiji objekti nakon konstrukcije ne mijenjaju svoje stanje

**Read-Only Interface pattern**

razred implementira javno sučelje koje sadrži operaciju samo tipa čitanja (engl. „read-only”) pa neke metode ne mogu mijenjati atribute

**Factory pattern**

uvođenje u radni okvir dodatnih posebnih (ne izvornih) razreda s mogućnošću njihovog instanciranja tim okvirom

**Proxy pattern**

omogućava interakciju s objektom kada ga klijent ne može izravno referencirati (npr. velik objekt koji je potreban sporadično, a troši puno resursa)

**Radni okviri**

Radni okvir je skup integriranih komponenti koji omogućava ponovnu uporabu arhitekture za učestalo korišten dio programske potpore

**Ciljevi:**

1. Efikasna uporaba principa ponovne uporabe
2. Aplikacije/primjenski programi namijenjeni obavljanju različitih poslova slične
3. namjene slično su oblikovani
4. Poticaj za pisanje ponovo uporabljivog koda
5. Poboljšanje kvalitete programa i produktivnosti programera
6. Omogućavanje neiskusnim programerima pisanje dobrog koda
7. Smanjenje količine ispitivanja

**Svojstva radnih okvira**

**1. Predefinirano ponašanje**

* implementirana osnovna funkcionalnost
* Radni okvir implementira korisne funkcionalnosti potrebne za rješavanje jednog skupa problema

**2. Proširivost**

* Radni okvir omogućava prilagodbu pojedinih funkcionalnosti mehanizmima
* nadjačavanja i specijalizacije te zahtijeva implementaciju specifičnih funkcionalnosti
* Omogućava primjenu radnog okvira za rješavanje više sličnih problema

**3. Inverzija upravljanja (engl. inversion of control)**

* Za razliku od biblioteka ili korisničkih programa tijekom programa upravlja radni okvir
* Radni okvir ima ulogu upravljanja programom

**4. Nepromjenjivost dijela koda (engl. non-modifiable framework code)**

* Izvorni kod radnog okvira nije predviđen za izravne promjene

**Problemi radnih okvira**

1. Složeniji radni okviri mogu imati zahtjevnu krivulju učenja
2. Cijena radnog okvira
3. Zrelost radnog okvira
4. Sigurnost radnog okvira
5. Razvoj radnog okvira zahtjeva eksperte iz domene primjene
6. Osiguranje generičke ponovne uporabe je zahtjevno
7. Složeno ispitivanje radnog okvira
8. Složeno održavanje radnog okvira

**Uporaba radnih okvira**

Program se iz radnih okvira razvija:

|| nasljeđivanjem i specijalizacijom komponenti

|| instanciranjem parametriziranih komponenti

|| razvojem funkcija koje nisu implementirane

Uspješno primjenjiv za ciljanu domenu primjene

Radni okvir implementira osnovnu logiku aktiviranja komponenti

U **objektno usmjerenoj paradigmi** radni okvir se sastoji iz **knjižnica razreda**

**Arhitekture radnih okvira**

**Horizontalni radni okvir** osigurava širi spektar općih usluga koja mogu koristiti više različitih primjenskih programa

**Vertikalni radni okvir** ima užu domenu primjene i njegova uporaba zahtjeva manji broj prilagodbi

**Linija proizvoda**

*Linija proizvoda* je skup svih produkata koji dijele funkcionalnosti koje su prilagođene ciljanim korisnicima ili tržištu, a izrađene su na zajedničkoj osnovnoj tehnologiji i dijele resurse

Različiti **produkti u liniji proizvoda** imaju različite **značajke** kako bi zadovoljili različite segmente tržišta. || npr. “demo”, “pro”, “lite”, “enterprise” i sl. verzije ++ lokalizirane verzije su također linije proizvoda.

**Prednosti:**

Jednostavnija izrada prilagođenih rješenja (engl. Tailor-made)Niža cijena

Poboljšana kvaliteta

Brže vrijeme izlaska na tržište

Niža cijena

**Primjeri radnih okvira**

1. Tehnološki radni okviri
2. Aplikacijski radni okviri
3. Radni okviri poslovnih rješenja
4. Web radni okviri

**WEB RADNI OKVIRI**

**Izazovi razvoja web aplikacije**

1. Korisničko sučelje i iskustvo
2. Skalabilnost
3. Performanse
4. poznavanje radnih okvira i platformi
5. Sigurnosne prijetnje

**Model-View-Controller MVC**

**Model** – model podataka, izvršavanje upita nad bazom podataka

**Pogled** – sučelje vidljivo korisniku, najčešće HTML + potporne tehnologije klijentske strane (CSS, JavaScript) + potporne tehnologije poslužiteljske strane (JSP, ASP.NET, PHP), može imati direktni uvid u model ili uvid ide preko nadglednika

**Nadglednik** – upravljanje poslovnom logikom, komunikacija s modelom,obrada HTTP zahtjeva i slanje odgovora pogledu

Vrste MVC modela

Zasnovani na guranju (npr. Spring, Django, Ruby on Rails, Symfony)

Zasnovani na povlačenju (npr. Struts2, JavaServer Faces, Apache Tapestry Lift,

Angular2, React )

**Model-pogled-pogled na model, MVVM**

Sloj „pogled na model“ (engl. viewmodel) djeluje kao poveznica između „modela” i „pogleda”

Pretvara podatkovne objekte „modela” (one označene kao Observable) u tokove podataka prema „pogledu”, reagira na promjene u modelu

**model-view-presenter, MVP**

predstavljač dobavlja podatke iz modela i formatira ih za pogled, nema potrebe za posebnim nadglednikom

**model-view-adapter, MVA**

kod ove varijante, adapter ili posrednik odvaja model od pogleda tako da jedan nema znanja o drugome

svu komunikaciju, uključujući preinake i transformacije podataka obavlja adapter, time omogućujući da se model i pogled razvijaju neovisno jedno o drugome

linearna veza pogled -> adapter -> model i obrnuto, nema izravne povezanosti pogleda i modela

**Inverzija upravljanja**

Kako bi bilo jasno koju implementaciju i s kojim parametrima se zaista poziva u nekom trenutku (npr. prilikom događaja instanciranja nekog razreda), radni okvir preuzima od korisničke aplikacije ulogu organizatora povezivanja između slojeva, čime ostvaruje inverziju upravljanja

Konačni cilj inverzije upravljanja je smanjenje međuovisnosti

**Ubacivanje ovisnosti**

- određeni korisnički kod (klijentski ili poslužiteljski) obavezno prima kao argumente metoda određene objekte, koji se zovu usluge, a koje specificira radni okvir kako bi sustav uspješno radio

- radni okvir, koji se naziva ubacivač ili injektor (engl. injector) ubacuje ovisnost o svojem određenom ugrađenom objektu u korisnički kod i definira sučelje koje se koristi za pristup toj usluzi

**Temeljne funkcionalnosti radnih okvira**

1. Posrednička arhitektura (engl. middleware)
2. Usmjeravanje (engl. routing)
3. Upravljanje perzistencijom podataka (engl. data persistency management)
4. Sigurnost (engl. security)

**Preslikavanje URL-a** (engl. URL mapping) ili **usmjeravanje** (engl. routing) je mehanizam radnog okvira koji se koristi za uspješno tumačenje klijentskih zahtjeva za određenim web resursom

**Sigurnost**

Temeljni zadatak radnog okvira za web po pitanju sigurnosti >>> ponuditi implementiranu funkcionalnost za rješavanje osnovnih sigurnosnih pitanja web aplikacije

Radni okviri najčešće olakšavaju sljedeća osnovna sigurnosna pitanja:

1. Sigurno upravljanje registracijom, prijavom i odjavom korisnika
2. Kontrolu pristupa URL-ovima

* samo ovlašteni korisnici smiju pristupiti određenim resursima web aplikacije
* pristupne uloge (engl. roles) i detaljnije privilegije (engl. privilege) korisnika definiraju se najčešće u bazi podataka

1. Upravljanje HTTP sjednicom

Pročitati si o Reactu, Angularu i Springu

**Ispitivanje programske potpore**

◼ Ispitivanje je aktivnost s ciljem otkrivanja informacija o ispravnosti i kvaliteti, te poboljšanja ispitivane programske podrške pronalaženjem kvarova.

**Ispitni slučaj** (engl. test case) je *uređeni par* (ulaz, izlaz), gdje je ulaz ulazni podatak, a izlaz očekivani izlazni podatak iz programa, zabilježen prije provođenja ispitivanja

Usporedba stvarnih rezultata s postavljenim standardima - **bez specifikacije nema ispitivanja!**

**Validacija**

1. zadovoljava li sustav specifikaciju
2. Izgrađujemo li dobar sustav?
3. provodi se aktivnostima ispitivanja programske potpore: ispitivanje komponenti, sustava, ispitivanje prihvatljivosti

**Verifikacija**

1. zadovoljava li sustav zahtjeve na ispravan način → odsustvo kvarova
2. Izgrađujemo li sustav na ispravan način? “Are we building the system right?”
3. Obuhvaća statičku provjeru dokumenata, oblikovanja, programskog koda

Ispitivanje programske podrške zasniva se na dinamičkoj verifikaciji ponašanja programa u izvođenju na konačnom broju ispitnih slučajeva, pogodno odabranih iz uobičajeno beskonačne domene izvođenja, obzirom na očekivano ponašanje

**Svojstva ispitljivih programa**

Ispitljivost je mjera mogućnosti jednostavnog ispitivanja programa

**Osmotrivost (engl. observability)**

* jednostavna identifikacija rezultata, različiti izlazi za različite ulaze,
* jednostavno uočavanje neispravnih rezultata

**Upravljivost (engl. controllability)**

* jednostavnost upravljanja tijekom provođenja ispitivanja
* mogućnost pogodne specifikacije, automatizacije i ponovne uporabe ispitivanja

**Dekompozicija (engl. decomposability)**

* neovisno ispitivanje modula

**Jednostavnost (engl. simplicity)**

* odnosi se na složenost arhitekture, logike programa i kodiranja

**Stabilnost (engl. stability)**

* promjene programa tijekom ispitivanja utječu na rezultate provedenih ispitivanja

**Razumljivost (engl. understandability)**

* dobre informacije o strukturama, međuovisnostima i organizacija tehničke dokumentacije

**Ciljevi ispitivanja**

1. Osigurati pouzdanost, ispravnost, otkrivanje pogrešaka
2. Minimizirati rizike
3. Definirati način sigurne uporabe
4. Procjena kvalitete

**Kvar (engl. fault)**

* proglašeni uzročnik pogreške
* unosi se pri oblikovanju ili programiranju
* može biti prikriven neko vrijeme

**Pogreška (engl. error)**

* manifestacija kvara
* uzrokuje pogrešku obrade/izvođenja programa i potencijalno dovodi do zatajenja
* dio stanja sustava odgovorno za stvaranje zastoja

**Zatajenje (engl. failure)**

* sustav ne zadovoljava specifikacije

**Razlozi kvarova**

* Ljudski faktor
* Zatajenje komunikacije
* Neodrživi vremenski okviri
* Neostvarivi/nejasni ciljevi
* Loše kodiranje
* Nedostaci upravljanja projektom

Primjeri:

1. Pogrešno napisano slovo

2. Pogrešna informacija - pomoć

3. Nepotpun ispis poruke

4. Neprovedena transakcija

5. Gubitak transakcije

6. Neispravna transakcija

7. Učestale vrlo ozbiljne pogreške

8. Oštećena baza

9. Pad sustava

10. Pad jednog poslužitelja se širi

Tipovi pogrešaka

1. blage (engl. mild)

2. umjerene (engl. moderate)

3. dosadne (engl. annoying)

4. uznemiravajuće (engl. disturbing)

5. ozbiljne (engl. serious)

6. vrlo ozbiljne (engl. very serious)

7. ekstremne (engl. extreme)

8. nepodnošljive (engl. intolerable)

9. katastrofalne (engl. catastrophic)

10. zarazne (engl. infectious)

**Zatajenje programske potpore**

Za manifestaciju zatajenja potrebno je ispuniti uvjete:

**Doseg** (engl. reachability) - mjesto kvara u programu mora biti dohvaćeno

**Infekcija** (engl. infection) - stanje programa mora biti neispravno

**Propagacija** (engl. propagation) - inficirano stanje mora uzrokovati promjenu nekog izlaza programa

Razlozi zatajenja:

* zahtjevi su nepotpuni, nekonzistentni, nemogući za implementaciju
* pogrešna interpretacija zahtjeva
* kvar u oblikovanju arhitekture
* kvar u oblikovanju programa
* upotrijebljen neodgovarajući algoritam
* kvar u programskom kodu
* problem implementacije
* dokumentacija nekorektno opisuje ponašanje sustava

Incident, nepredviđen događaj ->> Simptomi povezani sa zatajenjem koji upozoravaju korisnika na postojanje zatajenja

**Klasifikacija programskih defekata**

**Heisenbug**

* Defekt iznenada mijenja karakteristiku ili nestaje - princip neizvjesnosti (engl. uncertainty principle)
* Pri traženju i ispravljanju defekta utječe se na program

**Bohrbug**

* Programeri više vole ovu vrstu za razliku od Heisenbuga – “klasični bug”
* Kvar se lako izolira i dosljedno se ponavlja pod dobro definiranim skupom uvjeta, aktiviranje i širenje pogreške nije složeno

**Mandelbug**

* Kvar čija aktivacija i/ili širenje je složeno
* teško ga je izolirati i sustavno ponoviti, npr. Pogreška uslijed raspoređivanja

**Schroedinbug**

* Kvar koji se očituje u ispravnom radu programa nakon što primijete da taj kod uopće nije trebao raditi
* Schrödinger's Cat Paradox

**Aging Bug**

* Kvar koji dovodi do akumulacije pogrešaka tijekom rada i uzrokuje zatajenje Ili gubitak performansi

**Obrada pogrešaka**

◼ Prevencija (engl. Error prevention)

◼ Detekcija (engl. Error detection)

◼ Oporavak (engl. Error recovery)