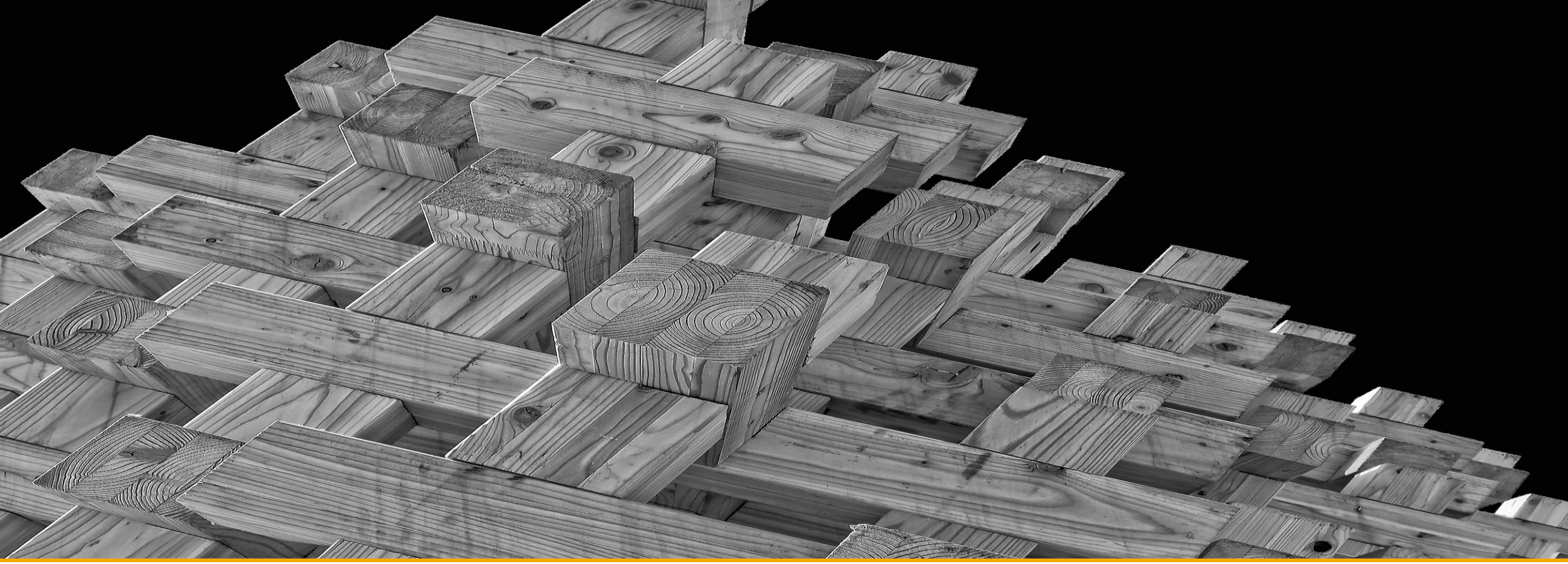


Design patterns - MVC in PHP web applications

Visoka škola elektrotehnike i računarstva strukovnih studija

Predmet: Programiranje veb aplikacija, Gostujuće predavanje

24. Decembar 2019, Beograd, Srbija

Miša Angeleski, M.S. CS, Sr. Software Engineer, Lead Software Engineer

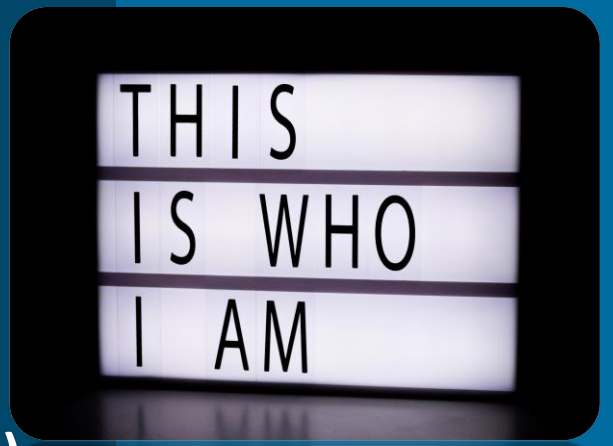
Model

View

Controller

Nešto o meni

- Sr. Software Engineer/Architect, Lead Engineer at SAP West Balkans
- 20 godina iskustva
- Preko 10 većih aplikativnih sistema u produkciji (30-40k korisnika)
- Iskustvo na sistemu koji je bio u produkciji od 1999 do 2018 (Projektovanje, implementacija i održavanje)
- Ko-autor interaktivnog sistema koji se koristi na predmetu Programiranje 1 i 2 za učenje programskog jezika u ovoj školi (S. Đenić)
- Karijera: Software developer (Contractor), Head of IT, Sr. Software Engineer/Architect, Lead Software Engineer



Agenda

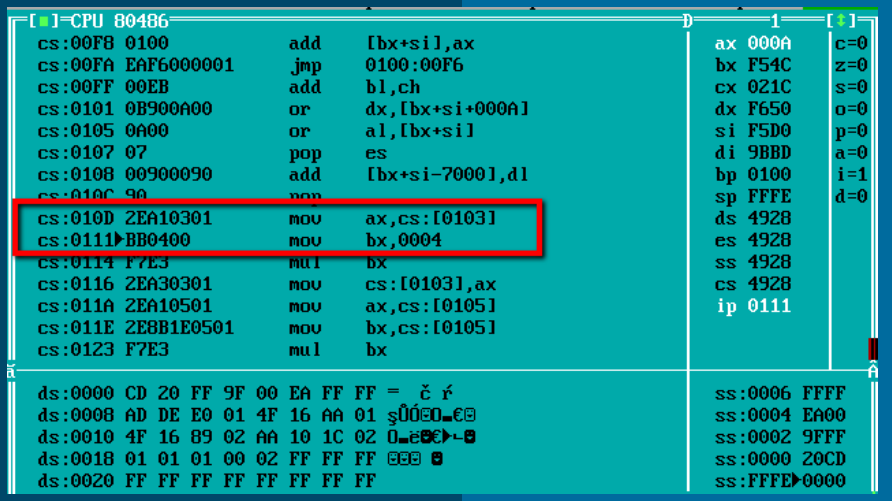
- Osnovni ciljevi kvalitetno strukturiranog koda
- Šta su softverski projektni uzori
- Uticaj korišćenja softverskih uzora na softverski proizvod
- Klasifikacija softverskih uzora
- Model View Controller
- Primena MVC - koje probleme rešava (Demo primer u PHP-u)
- Q&A

„Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.“

Martin Fowler

Zašto ne programiramo (često/uopšte) u asembleru?

- Loša **čitljivost** koda
- Velika **kompleksnost**
- **Zavisnost** od procesorske arhitekture



Koriste se samo u embedded sistemima i vremenski kritičnim sistemima (drajveri)

Da li kod viših programskih jezika imamo slične stvari?

- Kod takođe može biti teško razumljiv
- Loše struktuiran
- Težak za održavanje

Osnovni ciljevi

- Veća razumljivost koda
- Lakše održavanje (nema špageti koda)
- Lakše testiranje (Unit, Integracioni testovi, UI)
- Brzo dodavanje novih funkcionalnosti
- Veće performanse

Rešenje

Dobra stuktura koda

je

Dobra arhitektura

a to su

Softverski uzori

Softverski projektni uzori (Design patterns)

- Opšte, ponovo upotrebljivo rešenje (recept) za česte probleme koji se sreću prilikom projektovanja
- Nisu završena rešenja, već startna pozicija za rešenje, nisu algoritmi niti specifične implementacije ili klase
- Opisuje određena znanja prikupljena iskustvom u domenu - nastali su iz inženjerske prakse

Softverski projektni uzori (Design patterns)

- Imaju svoja imena kako bi u profesionalnom smislu mogli da pričamo o njima
- Dobro struktuiran sistem je pun softverskih uzora
- Dobili na popularnost kada je objavljena knjiga Design Patterns: Elements of Reusable Object-Oriented Software (GoF) 1994. Te godine održana prva konferencija o projektnim uzorima

Uticaj korišćenja softverskih uzora na softverski proizvod

Tehnički faktori:

- **Uglavnom unapređenje performansi i svih ostalih nefunkcionalnih zahteva: brzina odziva, skalabilnost, sigurnost, pouzdanost…**

„Many architectural decisions are about performance.“

Uticaj korišćenja softverskih uzora na softverski proizvod

Ekonomski i organizacioni:

- Dobra struktura omogućava paralelizam u radu tj. veću brzinu izrade (npr. dizajner/programer kao u slučaju MVC/MVP/MVVM)
- Bolje testiran kod znači manje bagova i veći kvalitet
- Razumljivost koda - da se može nastaviti razvoj a da se ne kreće iz početka

  (U protivnom postoji strah „Ako nešto promenim ko zna gde će da pukne i šta će da se desi. Ne diraj ništa, ovo sve radi" . Kolege neće da preuzmu dalju odgovornost i refaktorišu kod. Neki čak napuštaju projekat iz ovih razloga, razvoj proizvoda uglavnom stagnira)
- Bolja softverska struktura nekad znači jeftiniju cenu infrastrukture na kojoj se softver izvršava

Nije samo kod kojeg ljudi razumeju razlog korišćenja softverskih uzora već bolji sveukupni

Cost-Benefit

Klasifikacija

- Uzori makroarhitekture (Architectural Patterns)
  - Generalne strukture: layers, pipes, filters…
  - Interaktivni sistemi: MVC, MVP, MVVM…
  - Distribuirani sistemi: clientserver, microservices, n-tiers, broker…
  - Ostalo: batch, interpreters, rule-based…

Klasifikacija

- Uzori mikroarhitekture (Design Patterns)
  - Uzori kreiranja: builder, factory, prototype, singleton…
  - Uzori strukture: adapter, bridge, composite, decorator, façade, flyweight, proxy…
  - Uzori ponašanja: chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor…

Kritike na račun korišćenja softverskih uzora

- Nepotrebna dupliranja
- Softverski uzori su zaobilazna rešenja za nedostatne programskog jezika

"Even when you choose a pattern, you'll have to modify it to meet your demands. You can't build enterprise software without thinking, and all any book can do is give you more information to base your decisions on.„

Martin Fowler

Razumevanje softverskih uzora je važno
za karijeru softverskog inženjera.

Razumevanje softverskih uzora je razumevanje
principa projektovanja softvera

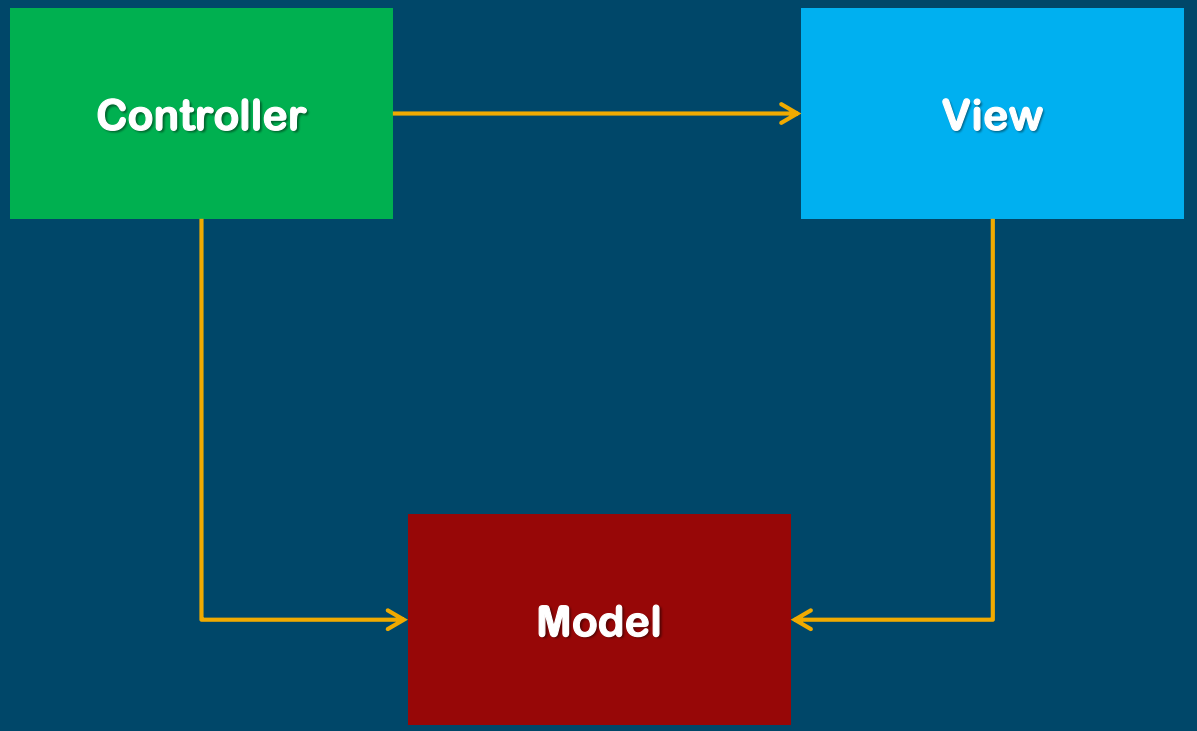

M
V
C

Model View Controller

- MVC uzor razvijen je u XEROX PARC 1979. godine od strane Norveškog informatičara i profesora Trygve Reenskaug
- Implementiran u Smalltalk-80 programskom sistemu
- Formulisan kao GUI softverski uzor za interaktivne aplikacije
- Do sada evoluirao u nekoliko varijanti: HMVC, MVA, MVP, MVVM
- Jedan od najviše korišćenih softverskih uzora
- Standardno podržan u većini najpopularnijih frejmvorka za razvoj web aplikacija

Model View Controller

Kontroler – prima inpute sa tastature, miša, ekrana i sl. od strane korisnika, manipuliše modelom i uzrokuje promenu Pogleda

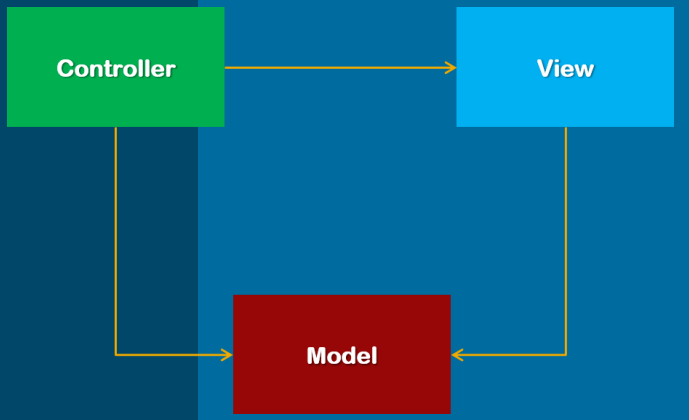


Pogled (View) upravlja prikazom informacija

Model upravlja poslovnom logikom i podacima aplikativnog domena, odgovara na zahteve za informacijama od strane korisničkog interfejsa i na instrukcije za promenom od strane Kontrolera

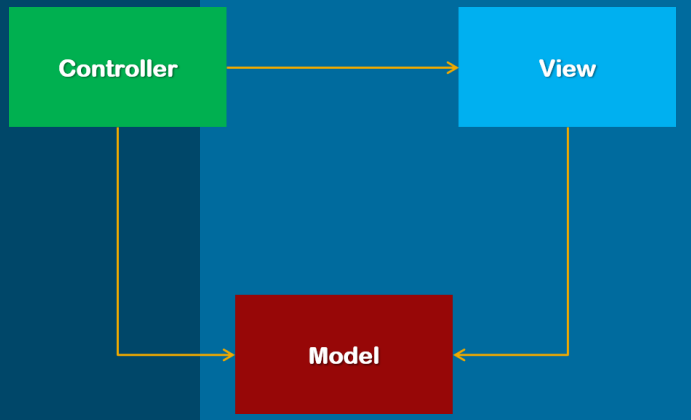
Model View Controller

- MVC je fundamentalni softverski uzor koji razdvaja prezentaciju od poslovne logike
- Razdvajanje je važno iz više razloga:
  - Promena prikaza za isti model (tabelarni, grafički, …) na različitim pa i na istim web stranama
  - Mogućnost korišćenja istog modela za različite klijente (rich client, Web browser, remote API, CLI…)
  - Korisnički interfejs teži da bude više zavistan od uređaja na kojem se prikazuje nego poslovna logika (web, mobilna aplikacija). Jasna separacija ubrzava migraciju i minimizuje greške na strani poslovne logike



Model View Controller

- Dizajniranje HTML strana (View) obično zahteva druge veštine za razliku od razvoja kompleksne poslovne logike pa razvoj možemo podeliti na dva segmenta. Ljudi koji razvijaju poslovnu logiku su potpuno nezavisni od razvoja prezentacionog sloja.
- Bolje testiranje softvera tako što svaki segment možemo zasebno testirati



Demo primer u PHP-u
Primena MVC - koje probleme rešava

Primeri:

1. Bez separacije poslovne logike i prezentacije – bez MVC (Zašto nije dobro?)
2. Nakon primene MVC (Šta smo dobili?)
3. Uvođenje Bootstrap-a na web stranu bez modifikacije poslovne logike i kontrolera (Šta je olakšano i zašto je to dobro?)
4. Uvođenje Ajax pristupa (Korak ka uvođenju REST API-ja i prelasku na Single Page Application. Zašto?)

Zaključak

- Posle 40 godina i dalje aktuelan, mnogo varijacija
- Mnogo kurseva, predavanja i knjiga napisano na tu temu a sa druge strane mnogi i dalje ne razumeju suštinu principa. Uglavnom zbog abstrakcije koju unosi frejmvork.
- Osnovni principi projektovanja softvera se ne menjaju tako često

Na pravom ste mestu i u pravo vreme!

Radite ono što volite, a volite ono od čega možete dobro da živite ☺

Hvala na pažnji

Pitanja?

Kontakt:

Miša Angeleski
misa.angeleski@gmail.com
https://www.linkedin.com/in/misa-angeleski/