OSNOVE ARHITEKTURE SOFTVERA

Osnove arhitekture softvera – savremen pristup

Uvod

Iako se u globalnim istraživanjima o platama zanimanje **softverskog arhitekte** redovno svrstava među **najpoželjnije i najplaćenije profesije**, dugo nije postojao sveobuhvatan vodič koji bi programerima pomogao da naprave sledeći korak u karijeri i postanu arhitekte. Ova knjiga popunjava upravo tu prazninu.

„Osnove arhitekture softvera: moderan pristup“ predstavlja detaljan pregled svih ključnih aspekata savremene softverske arhitekture, obogaćen najnovijim saznanjima i praksama iz industrije. Knjiga obuhvata teme kao što su **arhitektonske karakteristike, obrasci, određivanje komponenti, dijagrami, upravljanje arhitekturom, rad sa podacima, primena generativne veštačke inteligencije, struktura timova** i mnoge druge.

Autori Mark Ričards i Nil Ford, iskusni praktičari i instruktori, fokusiraju se na **principijelni pristup arhitekturi** koji je primenljiv na sve tehnološke stekove i okruženja. Pristup u knjizi je savremen, praktičan i baziran na desetogodišnjem razvoju discipline, pružajući čitaocu uvid u ono što danas znači biti arhitekta u softverskoj industriji.

Ovaj vodič namenjen je kako **ambicioznim programerima** koji žele da razviju arhitektonsko razmišljanje, tako i **iskusnim arhitektama** koji žele da unaprede svoju praksu i ostanu u toku sa evolucijom profesije.

NAPOMENA: *Ovaj prikaz je moje razmišljanje na osnovu sadržaja knjige i smernice za dublje promišljanje teme.*

I. TEMELJI ARHITEKTURE

Uvod: Temelji arhitekture

Razumevanje osnova softverske arhitekture započinje razlikovanjem strateških i taktičkih odluka – upravo na toj liniji se deli arhitektura od dizajna. Dok se arhitektura bavi dugoročnim odlukama koje oblikuju čitavu strukturu i ponašanje sistema, dizajn se fokusira na implementacione detalje unutar tih okvira. Ključni koncepti kao što su kompromisi, modularnost, metrički pokazatelji (kohezija, spregnutost, connascence) i komponentno razmišljanje pomažu arhitekti da donosi održive odluke. Istovremeno, praksa pokazuje da uspešan arhitekta ne može biti izdvojen iz procesa razvoja – on mora biti istovremeno strateški mislioc i praktični vodič tima.

Pored strukturalnih aspekata, savremena arhitektura zahteva i jasno definisanje i merenje arhitektonskih karakteristika – performansi, skalabilnosti, sigurnosti i drugih kritičnih osobina sistema. Ove karakteristike se ne samo identifikuju kroz poslovne zahteve, već se i meri njihova realizacija kroz konkretne metrike i fitness funkcije. Uloga arhitekte postaje sve kompleksnija: on ne samo da projektuje sistem, već upravlja granicama između komponenti, predviđa posledice svake odluke i povezuje tehnički svet sa poslovnim realnostima. Temelji arhitekture, obrađeni u ovom delu knjige, pružaju čitaocu alat za donošenje odluka koje su dugoročno održive, merljive i prilagođene stvarnim potrebama.

1. Arhitektonsko razmišljanje

Razlika između arhitekture i dizajna:

- Arhitektura se bavi **strateškim odlukama** – onima koje imaju dugoročan uticaj.
- Dizajn je **taktički** – rešenja na nivou implementacije.

Primer:

- Arhitektonska odluka: izbor između mikroservisa i monolita.
- Dizajnerska odluka: izbor između klase i funkcije u okviru servisa.

Ostali koncepti:

- *Trade-offs* (kompromisi) su osnova svakog arhitektonskog izbora.
- *20-minutno pravilo*: Ako tehnologija ne može da se razume za 20 minuta, nije zrela za primenu.

Najbolje prakse i saveti:

- Ne može se sve unapred planirati – arhitekta mora da **prati promene i brzo reaguje**.
- Balans između **kodiranja i arhitekture** povećava autoritet i razumevanje problema.

2. Modularnost i metrički pokazatelji

Modularnost vs granularnost:

- Modularnost se odnosi na **organizaciju funkcionalnosti**, dok je granularnost **nivo detalja**.

Ključne metrike:

- **Kohezija**: što je viša, to je modul "kompletniji".
- **Spregnutost (Coupling)**: što je niža, to su moduli nezavisniji.
- **Connascence**: mera zavisnosti i promena između komponenti.
- **Distance from the main sequence**: upoređivanje fleksibilnosti i robusnosti komponenti.

Najbolje prakse i saveti:

- Moduli ne treba da budu ni preveliki ni premali.
- Svaka komponenta mora imati jasno definisanu **odgovornost i granicu**.

3. Arhitektonske karakteristike

Vrste karakteristika:

1. **Operativne**: performanse, skalabilnost, dostupnost
2. **Strukturne**: fleksibilnost, portabilnost, proširivost
3. **Unakrsne (cross-cutting)**: sigurnost, auditabilnost

Pojam "least-worst architecture" – umesto idealnog rešenja, traži se **najmanje loša opcija** u datim ograničenjima.

Najbolje prakse i saveti:

- Arhitektonske karakteristike uvek treba povezati sa **poslovnim potrebama**.
- Poželjno je rangirati karakteristike i izabrati one **najkritičnije** za uspeh sistema.

4. Identifikacija karakteristika

Kako izvući karakteristike iz zahteva:

- Poslovni zahtevi → domen → karakteristike
- Korišćenje *kata* (arhitektonskih vežbi) pomaže u učenju identifikacije

Implicitne vs eksplicitne karakteristike:

- Implicitne nisu jasno izrečene, ali se podrazumevaju (npr. bezbednost).

- Eksplicitne su jasno izražene (npr. sistem mora biti dostupan 99,9%).

Najbolje prakse i saveti:

- Koristiti pitanja: "Zašto je ovo važno korisniku?" da bi se izvukle karakteristike.
- Kombinovati karakteristike u **kompozitne** kada se pojavljuju zajedno.

5. Merenje i upravljanje karakteristikama

Vrste merenja:

- **Operativne** (npr. latency, throughput)
- **Strukturne** (npr. broj nezavisnih modula)
- **Procesne** (npr. vreme potrebnog razvoja)

Upravljanje karakteristikama kroz:

- **Fitness funkcije**: automatski testovi koji proveravaju arhitektonske ciljeve
- **Granice** i **pravila razvoja**

Najbolje prakse i saveti:

- Fitness funkcije postaju **arhitektonski testovi** – deo CI/CD-a.
- Svaka karakteristika mora imati merni **indikator uspeha**.

6. Komponentno razmišljanje

Logički vs fizički dizajn:

- Logičke komponente: poslovne funkcionalnosti
- Fizičke komponente: deployment jedinice

Pojmovi:

- **Statička spregnutost**: direktne zavisnosti
- **Vremenska spregnutost**: potreba da se izvrše u određenom redosledu

Zakon Demetera: "Ne pričaj sa tuđim prijateljima" – izbegavanje posrednih zavisnosti.

Najbolje prakse i saveti:

- Svaka komponenta mora imati **jasno definisanu odgovornost**.
- Koristiti user storije za **mapiranje komponenti**.

II. STILOVI ARHITEKTURE

Izbor arhitektonskog stila predstavlja jednu od najvažnijih i najdelikatnijih odluka u razvoju softverskog sistema. Ne postoji univerzalno najbolje rešenje – pravi izbor zavisi od niza faktora kao što su složenost poslovnog domena, veličina i organizacija tima, zahtevi za performansama i skalabilnošću, brzina isporuke kao i dostupni resursi za infrastrukturnu podršku.

Upravo zbog tih brojnih varijabli, savremeni pristup podrazumeva biranje **„najmanje loše opcije“**, odnosno stila koji se u konkretnom kontekstu pokazuje kao najpraktičnije rešenje, čak i ako nije savršeno. Arhitekta mora da donosi odluke vođene realnim ograničenjima, a ne idealima – jer uspešna arhitektura nije ona koja je teoretski besprekorna, već ona koja funkcioniše u datom okruženju.

1. Layered Architecture Style *(slojna arhitektura)*

Osnovna ideja: Svaka komponenta ima svoju odgovornost i komunicira samo sa slojem ispod i iznad.

Tipični slojevi:

- Prezentacioni
- Poslovna logika
- Pristup podacima
- Baza podataka

Prednosti:

- Jednostavna za razumevanje i razvoj
- Dobar separation of concerns (razdvajanje odgovornosti)
- Lako testiranje slojeva

Rizici:

- Teško skaliranje
- Pojava “slojne infekcije” – logika curi iz jednog sloja u drugi

Kada koristiti:

- Male i srednje aplikacije
- Timske organizacije sa jasnim ulogama

Kada ne koristiti:

- Sistemi sa visokim performansnim zahtevima
- Distribuirani sistemi

Najbolje prakse i saveti:

- **Izolovati slojeve pravilno** (npr. interfejsima)
- **Izbegavati skakanje slojeva** ("leaking abstractions")

2. Modular Monolith Style *(modularni monolit)*

Osnovna ideja: Sve je u jednoj deploy jedinici, ali **logički razdvojeno u module**.

Prednosti:

- Lakša integracija nego kod mikroservisa
- Jednostavno testiranje i deployment
- Dobra baza za kasniji prelazak u mikroservise

Rizici:

- Zavisnosti među modulima ako nisu jasno definisane
- Može prerasti u "big ball of mud"

Kada koristiti:

- Složeni domeni sa potrebom za modularizacijom
- Ekipe koje tek počinju sa decompozicijom sistema

Najbolje prakse i saveti:

- **Jasne granice modula**
- **Interna API komunikacija** čak i kada je sve u istoj aplikaciji

3. Pipeline Style *(protok podataka kroz faze)*

Struktura: Podaci se obrađuju kroz niz **filtera**, koji su povezani **cevima** (*pipes*).

Primer: Kompajler (tokenizacija → parsiranje → optimizacija → generisanje koda)

Prednosti:

- Visoka modularnost
- Lako paralelizovati procese

Rizici:

- Pogodna samo za određeni tip obrada (stream-like)
- Može biti teško pratiti stanje kroz faze

Najbolje prakse i saveti:

- Svaka faza mora imati **jasan ulaz/izlaz**
- Idealno za **ETL sisteme**, obradu slika, obradu zvuka, itd.

4. Microkernel Style *(jezgro i dodaci)*

Struktura: Postoji **core sistem** sa minimalnom funkcionalnošću i **plugin dodaci** za proširenja.

Primer: Eclipse IDE + dodaci

Prednosti:

- Visoka fleksibilnost
- Dobar za alate sa raznim varijacijama

Rizici:

- Zavisnosti među pluginovima mogu postati komplikovane
- Teško definisati stabilan kontrakt između jezgra i dodatka

Najbolje prakse i saveti:

- Koristiti **registracione mehanizme**
- Pisati **jasne kontrakte za komunikaciju**

5. Service-Based Architecture *(servisno orijentisana)*

Osnovna ideja: Sistemi se grade od **manjih servisa**, često sličnih mikroservisima, ali **manje granularnih**.

Prednosti:

- Delimična nezavisnost servisa
- Bolja organizacija timova

Rizici:

- Teška koordinacija servisa
- Zavisi od kvalitetne infrastrukture (gateway, logging, auth)

Najbolje prakse i saveti:

- Pravilna definicija **granica servisa**
- Upravljanje komunikacijom (sinkroni vs asinhroni modeli)

6. Event-Driven Architecture *(EDA - događaji kao pokretači)*

Karakteristike:

- Komponente komuniciraju preko događaja
- Visok stepen asinhronosti

Prednosti:

- Odlična skalabilnost
- Slaba povezanost komponenti

Rizici:

- Teškoća u otkrivanju toka događaja
- Problem sa redosledom i greškama u obradi

Najbolje prakse i saveti:

- Koristiti **event logove** za audite i dijagnostiku
- Jasna razlika između **eventa** i **komande**

7. Space-Based Architecture *(prostor kao jedinica raspodele)*

Struktura:

- *Processing Units*, *Data Grid*, *Messaging Grid*
- Idealna za sisteme sa **visokom propusnošću i promenljivim opterećenjem**

Primeri:

- Sistem za prodaju karata
- Online aukcije

Prednosti:

- Otpornost na nagli rast korisnika
- Eliminacija uskog grla baze

Rizici:

- Složena implementacija
- Teško praćenje stanja

Najbolje prakse i saveti:

- Koristiti **virtualizaciju middleware sloja**
- Implementirati **mehanizme za sinhronizaciju podataka**

8. Orchestration-Driven SOA

Razlika u odnosu na EDA:

- *SOA* koristi **centralizovanu orkestraciju**, dok je EDA zasnovana na **distribuisanoj koordinaciji**

Prednosti:

- Pregledna kontrola toka procesa
- Jednostavnije upravljanje zavisnostima

Rizici:

- Centralna tačka otkaza
- Kompleksne orkestracije

9. Microservices Architecture

Granularniji od SOA, svaki servis:

- Ima svoj deployment
- Ima svoj tim
- Ima svoju bazu

Karakteristike:

- Granularnost i izolacija
- Komunikacija: REST, gRPC, događaji
- Potreba za orkestracijom i sagama (za transakcije)

Prednosti:

- Potpuna nezavisnost
- Skalabilnost na nivou servisa

Rizici:

- Operativna kompleksnost
- Distributed data management

Najbolje prakse i saveti:

- Koristiti *Bounded Context* kao bazu za deljenje servisa
- Ugraditi observabilnost od starta (tracing, logging, metrics)

10. Kako odabrati odgovarajući stil?

Kriterijumi:

- Složenost domena
- Veličina tima
- Performanse i skalabilnost
- Brzina isporuke
- Budžet za DevOps

Pristup "najmanjeg zla":

- Nema idealnog – biraj stil koji **najmanje šteti** u datom kontekstu

III. TEHNIKE I SOFT VEŠTINE

1. Donošenje arhitektonskih odluka

Pojam arhitektonske odluke (ADR):

- Svaka značajna odluka u sistemu mora se **zabeležiti, objasniti i kontekstualizovati**.
- Ključne komponente:
  - **Problem** koji se rešava
  - **Kontekst i ograničenja**

- **Opcije razmatrane**
- **Odabrano rešenje i razlog**

Antipaterni:

- *Email-Driven Architecture*: odluke se donose preko mejlova bez tragova
- *Groundhog Day*: ista odluka se ponavlja jer prethodna nije dokumentovana

Primer zapisa ADR:

```
#1: Odluka o asinhronoj komunikaciji
Datum: 2025-04-12
Problem: Sporost procesa potvrde porudžbine u realnom
vremenu
Rešenje: Uvođenje događajnog sistema preko Kafka
redova
Razlog: Povećanje responzivnosti i skalabilnosti
sistema
```

Najbolje prakse i saveti:

- Koristiti **jedinstven format** za ADR zapise
- Redovno **revidirati i deliti** dokumente sa timom
- Uključiti **AI alate** za predloge na osnovu sličnih obrazaca

2. Analiza arhitektonskog rizika

Faze Risk Storming-a:

1. **Identifikacija** – svi članovi tima iznose rizike
2. **Konsenzus** – diskusija i rangiranje rizika
3. **Strategija ublažavanja** – plan za najkritičnije rizike

Kategorije rizika:

- Dostupnost

- Elastičnost
- Bezbednost
- Performanse
- Održavanje

Koristan alat: **Risk Matrix** (opasnost × verovatnoća)

Najbolje prakse i saveti:

- Analizu rizika povezati sa **user stories**
- Ponoviti vežbu tokom trajanja projekta – rizici se menjaju

3. Dijagrami i alati

Vrste dijagrama:

- **UML** (Unified Modeling Language) – klasični standard
- **C4 model** – od konteksta do koda (4 nivoa: kontekst, kontejner, komponenta, kod)
- **ArchiMate** – za enterprise arhitekture

Vodič za dobar dijagram:

- Svaki dijagram mora **odgovarati na konkretno pitanje**
- Ne pretrpavati detaljima
- Poštovati doslednu legendu i konvencije

Najbolje prakse i saveti:

- Uvesti standardizovane šablone u timu
- Prikazati **veze i smer komunikacije** među komponentama

4. Efikasni timovi

Uloge i upozorenja:

- *Kontrolor*: ne dozvoljava timsku autonomiju

- *Teoretičar*: teorija bez praktičnog rada
- *Efektivan arhitekta*: mentor, komunikator, lider

Koncept "process loss":

- Gubitak efikasnosti zbog loše koordinacije

Checkliste za kvalitet:

- Validacija koda
- Testiranje (unit, integracija, funkcionalni testovi)
- Provera spremnosti za release

Najbolje prakse i saveti:

- Arhitekta ne komanduje – **usmerava timove**
- Checkliste su **alat za izgradnju poverenja**

5. Veštine pregovaranja i liderstva

Pregovaranje sa stakeholderima:

- Fokus na **zašto**, ne na **kako**
- Prilagoditi komunikaciju nivou sagovornika

Pregovaranje unutar tima:

- Arhitekta treba da **posreduje među developerima** i drugim arhitektama

Model "4C" arhitekte:

1. **Communication** – jasno izražavanje
2. **Collaboration** – rad u timu
3. **Curiosity** – stalno učenje
4. **Confidence** – poverenje u svoje odluke

Najbolje prakse i saveti:

- **Liderstvo primerom** – kodiraj, dokumentuj, saslušaj
- Veštine arhitekte su često više **društvene nego tehničke**

IV. ARHITEKTONSKE INTERSEKCIJE

Savremeni softverski sistemi ne postoje u vakuumu — arhitektonske odluke duboko su isprepletane sa implementacijom, infrastrukturom, timovima, podacima, pa čak i sa poslovnim i tehnološkim okruženjem u kom se razvijaju. U ovom poglavlju fokus je na tim **presudnim dodirnim tačkama**, koje odlučuju da li će arhitektura ostati samo lepo zamišljen model ili postati delotvorna stvarnost. Od toga da li se arhitektonska pravila zaista primenjuju u kodu, preko usklađenosti sa infrastrukturom i DevOps alatima, pa sve do načina na koji su timovi organizovani — sve ovo određuje uspeh sistema na duže staze.

Takođe, uloga arhitekte sve više uključuje i razumevanje šireg konteksta: od vlasništva nad podacima, preko poštovanja zakonskih regulativa, do primene generativne veštačke inteligencije kao podrške u svakodnevnom radu. Intersekcije arhitekture sa ovim domenima zahtevaju ne samo tehničko znanje, već i komunikacione veštine, sposobnost adaptacije i strateško razmišljanje. Pravi arhitekta ne gradi sistem samo iznutra – on **povezuje svetove**, pretvarajući kompleksne zahteve u rešenja koja su i tehnički održiva i poslovno relevantna.

1. Arhitektura i implementacija

Ključna pitanja:

- Kako odluke o arhitekturi utiču na konkretan kod?
- Da li se arhitektonska pravila sprovode u praksi?

Primeri:

- Arhitekta definiše CQRS pattern, ali tim koristi tradicionalni pristup
- Odluka o asinhronom modelu, ali se koristi sinhroni REST

Najbolje prakse i saveti:

- Arhitekta mora **biti prisutan u kod bazi**
- **Code reviews** su mesto gde arhitektura "oživi" ili "umre"

2. Arhitektura i infrastruktura

Zavisnosti:

- Deployment model (cloud vs on-prem)
- CI/CD praksa i alatke
- Mrežna sigurnost i dostupnost servisa

Primer:

- Event-Driven sistem zahteva Kafka, ali infrastruktura ne podržava to

Najbolje prakse i saveti:

- Arhitektonske odluke **moraju biti izvodljive** na postojećoj infrastrukturi
- Arhitekta treba da sarađuje sa DevOps timovima

---

3. Arhitektura i topologija podataka

Vrste topologije:

- *Monolithic Database* – svi servisi koriste jednu bazu

- *Domain Database* – po domenima, ali deljena
- *Dedicated Database* – svaki servis ima svoju bazu

Kritične odluke:

- Kako se sinhronizuju podaci?
- Gde se nalazi izvor istine?
- Da li sistemi pišu i čitaju iz iste baze?

Najbolje prakse i saveti:

- Čitanje i pisanje razdvojiti kad god je moguće
- Uvek dokumentovati **data ownership**

4. Arhitektura i inženjerske prakse

Veze sa praksama kao što su:

- CI/CD (Continuous Integration / Delivery)
- Infrastructure as Code
- Automated Testing
- Monitoring i observability

Primer:

- Arhitektura podržava skalabilnost, ali **nema automatskog testiranja** pod opterećenjem

Najbolje prakse i saveti:

- Arhitekta mora da razume **alatke koje koristi tim**
- Arhitektura mora biti **testabilna i merljiva**

5. Arhitektura i timske strukture

Team Topologies koncept:

- Organizacija tima mora odgovarati arhitekturi
- Conway’s Law: struktura sistema odražava strukturu organizacije

Vrste timova:

- Stream-aligned teams
- Enabling teams
- Complicated subsystem teams
- Platform teams

Najbolje prakse i saveti:

- Pre nego što se odredi stil, treba znati **ko gradi sistem**
- Dobar arhitekta **crta strukturu prema ljudima, ne obrnuto**

6. Arhitektura i poslovno okruženje

Šira slika:

- Tržišna dinamika
- Budžet
- Regulative (npr. GDPR, HIPAA)

Primeri:

- Sigurnosna arhitektura mora zadovoljiti zakon
- Performanse isporuke moraju biti konkurentski faktor

Najbolje prakse i saveti:

- Arhitektura je **poslovni alat**, a ne akademski eksperiment

- Arhitekta mora znati **koja je vrednost sistema za krajnjeg korisnika**

7. Arhitektura i generativna veštačka inteligencija

Uloga GenAI u arhitekturi:

- Pomoć u generisanju ADR zapisa
- Kreiranje primera dijagrama iz teksta
- Analiza rizika na osnovu korisničkih priča
- Simulacija stilova pre implementacije

Primeri primene:

- AI kao **asistent arhitekte**, ne zamena
- Kreiranje nacrta mikroservisa iz korisničkog opisa

Najbolje prakse i saveti:

- AI treba koristiti za **ubrzanje procesa**, ne za donošenje odluka
- Arhitekta mora **validirati rezultate** koje AI predloži

V. ZAKONI ARHITEKTURE I ZAVRŠNE PORUKE

U osnovi svake dobre arhitekture ne stoje samo tehnički alati i obrasci, već i **zakoni razmišljanja** koji oblikuju način donošenja odluka. Prvi zakon softverske arhitekture – da je *sve kompromis* – podseća nas da nijedna odluka ne dolazi bez posledica. Svaki izbor nosi sa sobom dobitke i gubitke: brzina razvoja može ugroziti održivost, fleksibilnost može zahtevati kompleksnije upravljanje, a jednostavnost može doći na uštrb sigurnosti. Važno je ne samo da kompromis bude prepoznat, već i da bude jasno komuniciran i stalno preispitivan tokom životnog ciklusa sistema.

Drugi zakon – *zašto je važnije od kako* – naglašava potrebu da se svaka arhitektonska odluka zasniva na razumevanju konteksta. Odluke kopirane

iz drugih sistema bez dubljeg sagledavanja okruženja često vode do problema. Arhitekta koji zna *zašto* može lako menjati *kako*, dok onaj koji zna samo *kako*, rizikuje da ostane bez smernica pri promeni zahteva. U završnim porukama autori nas podsećaju da je arhitektura duboko **ljudska aktivnost**, usmerena na komunikaciju, saradnju i realna rešenja – vođena vizijom, ali ukorenjena u praksi.

1. Prvi zakon softverske arhitekture

"Sve u softverskoj arhitekturi je kompromis."

Bez obzira na stil, tehnologiju ili alat, uvek postoji balansiranje između:

- Performansi i skalabilnosti
- Sigurnosti i jednostavnosti
- Fleksibilnosti i robustnosti
- Brzine razvoja i održivosti

Primer kompromisa:

- **Deljena biblioteka**: jednostavna integracija, ali zateže održavanje
- **Deljeni servis**: bolja izolacija, ali kompleksnija komunikacija

Prva posledica (korolar):

„Ako ne znaš koji je kompromis u tvojoj odluci, verovatno si napravio lošu odluku.“

Druga posledica:

„Komunikacija kompromisa mora biti trajna – ne postoji ‘jednom odlučeno, zauvek rešeno’.“

Najbolje prakse i saveti:

- Svaku odluku sagledati kroz **šta dobijamo** i **šta gubimo**
- Kompromis mora biti **svesno prihvaćen, ne slučajan rezultat**

2. Drugi zakon softverske arhitekture

"Zašto je važnije od kako."

U knjizi se snažno insistira na **objašnjavanju razloga iza svake arhitektonske odluke**. Ako se zna „zašto“, lako se kasnije može promeniti „kako“.

Antipattern: Out of Context Decision

Odluka prekopirana iz drugog projekta bez razumevanja konteksta

Spektar između ekstrema:

- Nije sve 100% monolit ili 100% mikroservis
- Arhitektura se često nalazi u **sivoj zoni između**

Najbolje prakse i saveti:

- Arhitekta mora **dokumentovati razlog**, a ne samo rešenje
- Tim mora **razumeti kontekst** u kom odluka funkcioniše

3. Završne poruke autora

Autori zaključuju knjigu sa **toplim, praktičnim i iskrenim savetima**, među kojima se izdvajaju sledeće poruke:

- **Arhitektura je ljudska aktivnost**: važni su odnosi, komunikacija i poverenje
- **Nema savršenog rešenja**: težimo optimalnom u konkretnom kontekstu

- **Budi pragmatičan, ali ne zaboravi viziju**: vodi tim ka cilju, ali ne po svaku cenu
- **Uči iz istorije sistema**: sve se ponavlja – greške se dešavaju u ciklusima
- **Ne brkaj alat sa rešenjem**: tehnologija je sredstvo, a ne cilj

Diskusiona pitanja (predložena u knjizi):

- Koje arhitektonske odluke su vam se pokazale kao loše? Zašto?
- Kako merite uspeh arhitekture?
- Kada se poslednji put vaša arhitektonska odluka menjala zbog poslovne promene?

📘 Zaključak celokupnog vodiča

Knjiga *Osnove arhitekture softvera* nije samo referenca za stilove i šablone – to je **kompas za profesionalni razvoj softverskog arhitekte**. Ona nudi kombinaciju:

- Teorije i prakse
- Tehnike i veština
- Strategije i psihologije tima