Korišćenje veštačke inteligencije u OO projektovanju

Principi softverskog inženjerstva, *Elektrotehnički fakultet Univerziteta u Beogradu*

Nova uloga projektovanja u AI eri

- Više ne projektujemo samo da bi programer razumeo sistem, već da bismo precizno usmerili AI asistente (LLM).
- Ključna promena: Fokus se pomera sa pisanja koda na **oblikovanje upita** (engl prompt engineering).

Inženjering upita (prompt engineering)

- Proces pažljivog oblikovanja instrukcija (promptova) kako bi AI model (poput ChatGPT-a ili Claude-a) generisao što precizniji, efikasniji i bezbedniji programski kôd.
- Umesto samo jednog pitanja, to podrazumeva definisanje konteksta, programskog jezika, željenih biblioteka i specifičnih ograničenja.
- Iterativan proces, dorada koda kroz dodatna pitanja radi ispravljanja bagova ili optimizacije.

CRC kartice i inženjering upita

- CRC kartica je savršen šablon za prompt
- CRC kartica je zapravo "ugovor o ponašanju”, arhitektonski nacrt za AI agenta:
  1. **Preciznost**: CRC definiše granice (šta klasa NE sme da radi).
  2. **Kontekst**: LLM dobija tačan spisak "saradnika", što ga sprečava da izmišlja nepostojeće funkcije.
  3. **Modularnost**: AI je primoran da piše kod koji se uklapa u širu sliku.
- Ako znate da napišete dobru CRC karticu, AI će vam izgenerisati 90% tačnog koda iz prvog pokušaja.

Primer: Bankomat

1. Korak: Klasična CRC Kartica

- **Klasa:** BankMachine
- **Odgovornosti:**
  Autentifikacija korisnika, Prikaz opcija, Isplata gotovine.
- **Saradnici:**
  BankCustomer, BankServer.

2. Korak: Inženjerski Prompt

"Ti si softverski inženjer koji implementira klasu Bankomat prema CRC specifikaciji.

Odgovornosti:

1. Autentifikacija klijenta pozivanjem BankServer API-ja.
2. Prikaz dostupnih opcija (isplata, stanje).
3. Upravljanje fizičkom isplatom (simulirano).

Saradnici: Koristi isključivo interfejse za BankCustomer i BankServer.

Zadatak: Generiši Python kod (Django) koji striktno prati ove odgovornosti. Nemoj dodavati logiku za obradu baze podataka direktno ovde, jer to nije tvoja odgovornost."

Solid i AI – odbrambeno projektovanje

Primer: AI generiše "Svemoćnu" klasu (Kršenje SRP i DIP)

Recimo da programer zatraži:

- "Napiši mi Django pogled (view) koji prima tekst posta koji je napisao korisnik, poziva AI za sažetak i čuva to u bazu."

Primer: AI generiše "Svemoćnu" klasu (Kršenje SRP i DIP)

- AI će često generisati nešto ovako (Loša praksa):

```
# LOŠ KOD: AI "ugurao" sve u jednu funkciju/klasu
class CreatePostView(APIView):
    def post(self, request):
        text = request.POST.get('text')
        # 1. DIREKTNA ZAVISNOST (Krši DIP) - Zaključani smo na OpenAI
        import openai
        client = openai.OpenAI(api_key="sk-...")
        # 2. LOGIKA OBRADE UNUTAR VIEW-A (Krši SRP)
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "Ti si asistent koji piše kratke i jasne sažetke teksta na srpskom."},
                {"role": "user", "content": f"Sažmi sledeći tekst u jednu rečenicu: {text}"}
            ], max_tokens=100
        )
        summary = response.choices[0].message.content
        # 3. RAD SA BAZOM
        post = Post.objects.create(content=text, ai_summary=summary)
        return render(request, "success.html")
```

Primer: AI generiše "Svemoćnu" klasu (Kršenje SRP i DIP)

Zašto je ovo loše:

1. Testiranje je nemoguće: Ne možete testirati snimanje posta bez da stvarno pozovete OpenAI i potrošite novac (jer je zavisnost hardkodovana).
2. Vendor Lock-in: Ako sutra želite da pređete na besplatni HuggingFace model, morate da menjate kod u srcu vašeg views.py.
3. Krhkost: Ako OpenAI API padne, ceo vaš pogled puca.

Rešenje: SOLID (Odbrambeno projektovanje)

- **Pravilan Prompt**: "Generiši mi klasu AISummarizer koja implementira interfejs ITextProcessor. Koristi OpenAI, ali neka zavisnost bude injektovana (DI)."
- **Rezultat** (Dobar kod):

```
# Interfejs (Apstrakcija) -- DIP
class ITextProcessor(ABC):
    @abstractmethod
    def process(self, text: str) -> str:
        pass

# Konkretna implementacija koja poštuje SRP
class OpenAISummarizer(ITextProcessor):
    def process(self, text: str) -> str:
            # Poziv APIja...
        return response.choices[0].message.content.strip()
```

Rešenje: SOLID (Odbrambeno projektovanje)

- **Rezultat** (nastavak):

```
# 3. Čist View - Ne zna ništa o OpenAI-u (Decoupling)
class CreatePostView(View):
    processor: ITextProcessor = None
    def post(self, request):
        text = request.POST.get('text')
        # Poziv AI servisu preko injektovanog procesora
        summary = self.processor.process(text)
        # Čuvanje u bazu
        Post.objects.create(content=text, summary=summary)
        # Vraćanje HTML stranice sa podatkom o sažetku
        return render(request, "success.html")
```

- U ruteru (urls.py) ubrizgavamo konkretnu AI klasu:

```
urlpatterns = [
    path('post/create/', CreatePostView.as_view(processor=OpenAISummarizer())),
]
```

Primer 2: AI I kršenje LSP

- Zamislite da se od AI-a traži da projektuje sistem za slanje obaveštenja (Notifications). AI će, vođen intuicijom, često predložiti nešto što na prvi pogled deluje logično, ali ruši arhitekturu.

Primer 2: AI I kršenje LSP

- Šta AI generiše (Loša praksa):

```
# Bazna klasa
class Notification:
    def send(self, message, user_email):
        print(f"Sending '{message}' to {user_email}")
# AI logično zaključuje: SMS je vrsta notifikacije
class SMSNotification(Notification):
    def send(self, message, phone_number):
        # PROBLEM: SMS ne koristi email!
        # send sada mora da "glumi" da je phone_number zapravo email
        # ili da ignoriše parametre, što ruši kod koji očekuje baznu klasu.
        print(f"Sending SMS '{message}' to {phone_number}")
# KOD KOJI PUCA:
def notify_all_users(notification_service, users):
    for user in users:
        # Ovaj kod očekuje email, ali SMSNotification traži telefon!
        notification_service.send("Hello!", user.email)
```

Primer 2: AI I kršenje LSP

- **AI je "copy-paste" arhitekta**: On vidi da Notification ima metodu send i samo je preklopi (override), ali ne razume da je potpis metode (parametri) deo ugovora.
- **Problem zamenljivosti**: Ako SMSNotification ne može da zameni Notification bez da polomimo klijentski kod (funkciju notify_all_users), prekršili smo LSP.
- **Posledica**: dobija se kod koji radi za Email, ali baca TypeError čim pokuša da ubaci SMS.

Primer 2 – pravilno rešenje

- Umesto nasleđivanja konkretne klase, koristimo interfejse (apstrakcije) koji su dovoljno opšti.
- Pravilan Prompt:

  Dizajniraj sistem obaveštenja koristeći SOLID. Koristi apstraktnu baznu klasu koja ne zavisi od kanala komunikacije (email/telefon).

Primer 2 – pravilno rešenje

- Rezultat (Dobar kod):

```
class Notification(ABC):
    @abstractmethod
    def send(self, message): # Parametri primaoca su unutar objekta, ne u metodi!
        pass

class EmailNotification(Notification):
    def __init__(self, email):
        self.email = email
    def send(self, message):
        print(f"Email sent to {self.email}")

class SMSNotification(Notification):
    def __init__(self, phone):
        self.phone = phone
    def send(self, message):
        print(f"SMS sent to {self.phone}")

# SADA RADI: Svaka klasa je zamenljiva jer sve imaju isti send(message)
```

AI prompting preporuke za softverske inženjere

- Veštačka inteligencija je tvoj **Junior programer**. Ti si **Arhitekta**. Da bi tvoj kod bio održiv, koristi sledeće principe prilikom pisanja promptova.

AI prompting preporuke za softverske inženjere

1. CRC Struktura (osmisli pre nego što tražiš kod)

- Ne piši: *"Napravi mi sistem za plaćanje."*
  Piši po CRC modelu:

- **Klasa:** PaymentProcessor
- **Odgovornosti:** Validacija kartice, komunikacija sa bankom, čuvanje statusa transakcije.

- **Saradnici:** BankAPI, DatabaseLogger.

AI prompting preporuke za softverske inženjere

2. SRP: Drži prompt "malim"

- Ako tvoj prompt sadrži reč **"i"** (npr. *"Napravi funkciju koja obrađuje sliku **i** šalje mejl **i** čuva u bazu"*), kršiš Single Responsibility Principle.

- **Prompting pravilo:** Traži od AI-a jednu po jednu klasu/funkciju. Tako ćeš dobiti kod sa manje bagova i lakšim Unit testovima.

AI prompting preporuke za softverske inženjere

3. DIP: "Ne spominji imena"

- Izbegavaj da AI-u direktno tražiš konkretne biblioteke unutar biznis logike.

- **Loše:** *"Napiši kod koji koristi OpenAI API za analizu teksta."*
- **Dobro:** *"Napiši klasu koja implementira interfejs* `ITextAnalyzer`*. Koristi Dependency Injection tako da kasnije mogu da zamenim konkretnu implementaciju."*
- **Cilj:** Tvoj kod mora biti nezavisan od konkretnog AI modela ili baze podataka.

AI prompting preporuke za softverske inženjere

4. LSP: Proveri "ugovor"

- Kada ti AI predloži nasleđivanje (`class B(A)`), uvek postavi sebi pitanje:
- **Pitanje:** *"Da li mogu da zamenim svaki objekat klase A objektom klase B, a da se program ne slomi?"*
- **Ako ne možeš:** Reci AI-u: *"Nemoj koristiti nasleđivanje ovde, koristi kompoziciju ili zajednički interfejs."*

AI prompting preporuke za softverske inženjere

5. AI Code Review (Tvoj glavni zadatak)

- Nakon što AI izgeneriše kod, proveri ga pomoću ove tri kontrolne tačke:

1. **Hardkodovanje:** Da li je AI hardkodovao API ključeve ili putanje?
2. **Skalabilnost:** Da li je kod zatvoren za izmene, a otvoren za proširenja (OCP)?
3. **Testabilnost:** Da li mogu da napišem Unit test za ovo bez interneta i baze podataka?