

Upotreba umjetne inteligencije u osiguravanju kvalitete u kontekstu softverskog inženjerstva

Galeković, Katarina

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:211:979946>

Rights / Prava: [Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0](#)

Download date / Datum preuzimanja: **2025-01-30**



Repository / Repozitorij:

[Faculty of Organization and Informatics - Digital Repository](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Katarina Galeković

**UPOTREBA UMJETNE INTELIGENCIJE U
OSIGURAVANJU KVALITETE U
KONTEKSTU SOFTVERSKOG
INŽENJERSTVA**

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ź D I N

Katarina Galeković

JMBAG: 0016136171

Studij: Organizacija poslovnih sustava

**UPOTREBA UMJETNE INTELIGENCIJE U OSIGURAVANJU
KVALITETE U KONTEKSTU SOFTVERSKOG INŽENJERSTVA**

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Valentina Kirinić

Varaždin, kolovoz 2024.

ZAHVALA

Na samom početku, želim izraziti zahvalnost mentorici na ukazanoj prilici za izradu ovog diplomskog rada, kao i za svu pomoć i podršku koju mi je pružila tijekom pisanja rada. Put do diplomskog rada bio je izazovan, osjećaji pomiješani, ali uvijek na tom putu bila je moja obitelj. Posebno hvala mami i ujaku koji su vjerovali u mene čak i kada sam ja sumnjala u sebe.

Hvala i Tebi, dragi čitatelju, koji čitaš ovo jer si odvojio vrijeme za čitanje ovog rada. Vrijeme je danas dragocjeno, pa neću duljiti, uživaj u čitanju!

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U teorijskom dijelu rada opisuje se osiguravanje kvalitetom u kontekstu softverskog inženjerstva te relevantne norme. Nadalje opisuje se umjetna inteligencija, njezin razvoj, trenutni trendovi i perspektive.

U nastavku se detaljizira povezanost umjetne inteligencije i osiguranja kvalitete u softverskom inženjerstvu i daju primjeri kako umjetna inteligencija može unaprijediti različite aspekte kvalitete.

U praktičnom dijelu rada prikazan je postupak i rezultati analize načina i učinkovitosti primjene umjetne inteligencije na konkretnom slučaju (softverskom projektu).

Ključne riječi: umjetna inteligencija, softversko inženjerstvo, osiguravanje kvalitete, norme, analiza učinkovitosti

Sadržaj

1. Uvod	1
2. Umjetna inteligencija.....	2
2.1. Povijest i razvoj.....	2
2.2. O umjetnoj inteligenciji	5
2.3. Primjena umjetne inteligencije.....	10
2.3.1. ChatGPT	11
3. Programsko inženjerstvo	16
3.1. Osnovni i ključni aspekti programskog inženjerstva	16
3.2. Primjena umjetne inteligencije u softverskom inženjerstvu	17
4. Uspostava i održavanje kvalitete.....	24
4.1. Osiguravanje kvalitete.....	24
4.2. ISO norme	26
4.2.1. ISO/IEC 27001	28
4.2.1.1. Primjena umjetne inteligencija za usklađivanje s normom ISO/IEC 27001.	29
4.2.2. ISO/IEC 25010.....	31

4.2.3. ISO/IEC/IEE 29119	38
4.3. Umjetna inteligencija u osiguravanju kvalitete u softverskom inženjerstvu	39
5. Istraživanje i primjena umjetne inteligencije u procesu razvoja programskog proizvoda...43	
5.1. Metode i tehnike rada.....	43
5.2. Kvalitativna analiza koda prema ISO/IEC 25010	63
5.2.1. Slučaj upotrebe 1: Analiza i unapređenje kvalitete programskog koda koristeći umjetnu inteligenciju prema ISO/IEC 25010.....	63
5.2.2. Slučaj upotrebe 2: Vrednovanje programskog koda i jedinice testa pomoću umjetne inteligencije prema normi ISO/IEC 25010.....	73
5.2.3. Slučaj upotrebe 3: Primjena metode Analitičkog hijerarhijskog procesa za vrednovanje programskog koda u skladu s normom ISO/IEC 25010	81
5.3. Upotreba umjetne inteligencije u analizi sigurnosnih mjera u skladu s normom ISO/IEC 27001	90
5.3.1. Slučaj upotrebe 4: Primjena umjetne inteligencije za provjeru usklađenosti programskog koda s normom ISO/IEC 27001 za funkcionalnost arhiviranja kartica.....	90
5.4. Upotreba umjetne inteligencije u analizi testiranja programskog koda i jedinice testa u skladu s normom ISO/IEC/IEEE 29119	98
5.4.1. Slučaj upotrebe 5: Upotreba umjetne inteligencije za kreiranje jedinica testova u skladu s normom ISO/IEC/IEEE 29119.....	98

5.4.2. Slučaj upotrebe 6: Analiza pokrivenosti jedinice testa JT1 za programski kod PK1 pomoću umjetne inteligencije.....	103
5.4.3. Slučaj upotrebe 7: Provjera usklađenosti jedinice testa s normom ISO/IEC/IEE 29119 pomoću umjetne inteligencije	109
5.4.4. Slučaj upotrebe 8: Usporedba testiranja funkcionalnosti arhiviranja dekova između testova razvijenih od strane programera i umjetne inteligencije	114
6. Zaključak	118
Popis literature	120
Popis slika	124
Popis tablica	127
Prilozi	128

1. Uvod

Za početak, vratimo se u doba prije COVID-19 pandemije. Iako se čini da je prošlo mnogo vremena, zapravo prošlo je tek nešto manje od pet godina. Doba prije pandemije izgledalo je sasvim drugačije. Mnogi trgovci, veliki proizvodni lanci i obrtnici nisu imali online platforme za prodaju svojih proizvoda i usluga. Tijekom COVID-19 život je „stao“ i mnoge poslodavce prisilio na otvaranje online prodajnih kanala. Neočekivani razvoj doveo je do značajnog skoka u razvoju programskog inženjerstva, koje je postalo neizostavna komponenta poslovanja kakvog danas poznajemo.

Na kraju dana, zadovoljstvo korisnika prilikom korištenja programskog proizvoda postalo je ključno za uspjeh organizacija. Tehnološka industrija raste iz dana u dan, a kako bi se održala kvaliteta softverskog proizvoda na visokom nivou potrebno je neprestano tražiti inovativne pristupe. Na globalnom tržištu, gdje je konkurencija velika, umjetna inteligencija igra ključnu ulogu u pomaganju poduzećima da ostanu konkurentna.

U ovom diplomskom radu prikazana je analiza primjene umjetne inteligencije u kontekstu osiguravanja kvalitete unutar softverskog inženjerstva. Kroz pregled slučaja upotrebe, programerima se pruža uvid u to koliko daleko umjetna inteligencija može ići u unapređenju kvalitete softvera i olakšavanju procesa razvoja.

2. Umjetna inteligencija

Razvoj tehnologije i računalnih sustava dovelo je do stvaranja umjetne inteligencije (eng. *Artificial Intelligence* - AI) koja je zainteresirala i „zaludila“ ljude širom svijeta. Danas ju smatramo jednim od najvažnijih područja istraživanja u znanosti. Psiholozi ljudsku inteligenciju karakteriziraju kao kombinaciju mnogih sposobnosti, dok se istraživanje umjetne inteligencije usredotočuje uglavnom na pojedine komponente inteligencije kao što su; učenje, rasuđivanje, rješavanje problema, percepcija i korištenje jezika [2].

Ovo poglavlje pruža sveobuhvatan pregled umjetne inteligencije; od njezinih začetaka do današnjeg stanja u kojem dominira model ChatGPT. Kronološkim redoslijedom, od najšireg pojma umjetne inteligencije pa do specifičnih primjena, poput ChatGPT-a, istražuju se ključni pojmovi i tehnologije. Na putu do modela ChatGPT-a važno je razumjeti širi koncept umjetne inteligencije, uključujući pojmove kao što su slaba umjetna inteligencija, strojno učenje, duboko učenje, obrada prirodnog jezika, veliki jezični model i generativna umjetna inteligencija.

2.1. Povijest i razvoj

Razumijevanje povijesnih trendova i razvoja umjetne inteligencije (skraćeno UI) ključno je za predviđanje budućih promjena u tehnologiji i njihovih utjecaja na prakse i norme u softverskom inženjerstvu. Ova analiza oslanja se na promatranje broja publikacija koje uključuju ključne riječi vezane uz UI, pri čemu se bilježi godišnja stopa rasta kako bi se identificirali značajni trendovi i događaji u povijesti istraživanja i razvoja UI [1].

Ideja o strojnom učenju datira još iz antičke Grčke. Važni događaji i prekretnice u evoluciji umjetne inteligencije prikazani su kronološki po godinama:

- 1945. godine Alan Turing je predvidio da će računala jednog dana vrlo dobro igrati šah.
- 1947. godine Turing je održao predavanje u Londonu o računalnoj inteligenciji gdje je rekao “Ono što želimo je stroj koji može učiti iz iskustva”.
- 1948. godine Turing predstavlja središnje koncepte umjetne inteligencije u izvješću *Intelligentni strojevi*. Rad nije nikada objavio, a mnoge njegove ideje kasnije su drugi pretvorili u stvarnost.
- 1950. godine Alan Turing objavljuje *Computing Machinery and Intelligence* i uvodi praktični test računalne inteligencije koji je sada poznat jednostavno kao Turingov test.

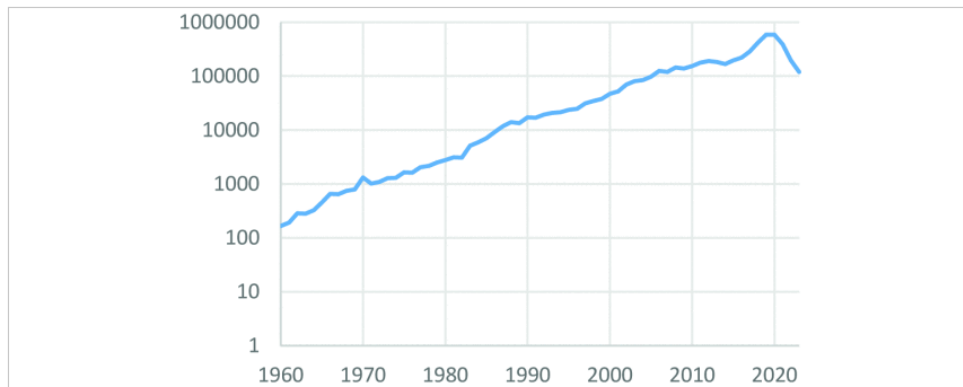
Turingov test je test gdje ljudski ispitivač pokušava razlikovati računalni i ljudski tekstualni odgovor.

- 1951. godine Christopher Strachey je napisao prvi uspješan program umjetne inteligencije.
- 1952. godine program za igru dame izvodio se na računalu Ferranti Marka. Program je mogao igrati cijelu partiju dame razumnom brzinom.
- 1952. godine Arthur Samuel je napisao prvi UI program za igru Dama u Sjedinjenim Američkim Državama.
- 1955. godine Samuel u svoj program za dame uključuje mehanizme za učenje na pamet i generalizaciju.
- 1956. godine John McCarthy skovao je izraz umjetna inteligencija na Dartmouth Collegeu.
- 1956. godine Allen Newell, Shaw, JC. i Herbert Simon stvorili su prvi softverski program za umjetnu inteligenciju pod nazivom *Logic Theorist*.
- 1957. godine Newell, Simon i Shaw pokrenuli su prvu verziju *General Problem Solver* (ili skraćeno GPS).
- 1962. godine Samuelov program za dame pobijedio je bivšeg prvaka Conncticuta.
- 1966. godine objavljena su dva najpoznatija rana UI programa, Eliza (koju je napisao Weizenbaum, J.) i Parry (koju je napisao psihijatar Colby, K.). Eliza je simulirala ljudskog terapeuta, a Parry je simulirao čovjeka koji doživljava paranoju. Oni su se oslanjali na jednostavne programske rečenice.
- 1967. godine Frank Rosenblatt gradi Mark 1 Perceptron, prvo računalo temeljeno na neuronskoj mreži koja je učila metodom pokušaja i pogrešaka.
- 1967. godine Shunchi Amari dizajnirao je duboku neuronsku mrežu s pet slojeva. Amari se smatra izumiteljem duboke neuronske mreže.
- 1968. godine Minsky, M. i Papert, S. objavljuju knjigu Perceptroni, koja postaje prekretnica u radu o neuronskim mrežama.
- 1970-ih godina umjetna inteligencija bila je zanemarena što je poznato kao prva UI zima, a tek 1980-ih pristup ponovo postaje istaknut.
- 1980-e neuronske mreže koriste algoritam povratnog širenja za samoosposobljavanje.
- 1991. godine Hugh Loebner osniva natjecanje Loebnerova nagrada u kojem nudi sto tisuća dolara prvom računalu koje prođe Turingov test.
- 1995. godine Russell, S. i Norving, P. objavljuju *Artificial Intelligence: A modern Approach* koji postaje jedan od vodećih udžbenika u proučavanju umjetne inteligencije.

- 1997. godine IBM-ov Deep Blue šahovsko računalo pobjeđuje svjetskog prvaka Kasparova, G. u šahu gdje je s pomoću dvjesto pedeset i šest paralelnih procesora ispitivao dvjesto milijuna mogućih poteza u sekundi i gledao unaprijed za čak četrnaest poteza igre.
- 2004. godine John McCarthy piše članak "Što je umjetna inteligencija".
- 2011. godine IBM-ov Deep Blue pobjeđuje prvake na Jeopardyju (na američkom televizijskom kvizu).
- 2015. godine Baiduovo superračunalo Minwa koristi konvolucijsku neuronsku mrežu za identifikaciju slika.
- 2016. godine DeepMindovo AlphaGo program pobjeđuje svjetskog prvaka Sodola, L. Go igrača. Pobjeda je još značajnija jer je samo nakon četiri poteza program bio u mogućnosti predvidjeti preko 14,5 bilijuna poteza. Google kasnije kupuje DeppMand za procijenjenih 400 milijuna USD.
- 2022. godine Pojava ChatGPTa ponovno pokreće razgovor o Turingovom testu. Kada se pojavio model ChatGPT neki su stručnjaci vjerovali da je prošao Turingov test.
- 2023. godine porast velikih jezičnih modela (LLM) kao što je ChatGPT, stvara ogromne promjene u izvedbi UI-e. Novim generativnim praksama umjetne inteligencije, modeli dubokog učenja mogu se unaprijed uvježbati na ogromnim količinama neobrađenih, neoznačenih podataka [2].

Poznavanje povijesnih trendova i razvojne dinamike ključno je za razumijevanje kako će umjetna inteligencija nastaviti transformirati softversko inženjerstvo i druge industrije. Nakon više od sedam desetljeća od početnih Turingovih predviđanja, umjetna inteligencija premašila je inicijalne ideje koja se razvila u tehnologiju te sada oblikuje svaki aspekt modernog života [2].

Iz podataka prikupljenih preko Google Scholar, na slici 1., vidljivo je kako je broj akademskih publikacija koje uključuju ključnu riječ "umjetna inteligencija" porastao sa sto šezdeset i šest u 1960. godini na sto dvadeset tisuća u 2023. godini. Ova golema pozitivna razlika prikazuje kako je interes za umjetnom inteligencijom eksponencijalno rastao, odražavajući širenje tehnologije i njezin sve veći utjecaj na različite industrije, uključujući softversko inženjerstvo [1].



Slika 1. Godišnja vremenska serija za broj akademskih publikacija dostupnih u indeksu Google Scholar (izvor: preuzeto s „Landmark events, major trends, and the future of AI research: history, promise, hype“, IEEE Xplore)

Umjetna inteligencija prošla je kroz mnoge faze visokih očekivanja, ali se svi stručnjaci slažu da stavljanje u upotrebu ChatGPT-a predstavlja značajnu prekretnicu. Danas, umjetna inteligencije ne samo da može učiti, već može i sintetizirati i druge vrste podataka uključujući slike, video zapise, programske kodove, pa čak i molekularne strukture [3].

2.2. O umjetnoj inteligenciji

Umjetna inteligencija se definira kao tehnologija koja računalima i strojevima omogućuje simulaciju ljudske inteligencije i sposobnost rješavanja problema. U kombinaciji s drugim tehnologijama, UI može obavljati zadatke koji bi inače zahtijevali intervenciju ljudske inteligencije [3].

Umjetna inteligencija je grana računalnih znanosti koja obuhvaća i često se povezuje sa strojnim učenjem i dubokim učenjem. Spomenute discipline usmjerene su ka razvoju algoritama umjetne inteligencije koji su inspirirani načinom na koji ljudski mozak donosi odluke. Ovakva vrsta algoritma može „učiti“ iz dostupnih podataka i s vremenom postajati sve točnija u klasifikacijama i predviđanjima. Kada je riječ o učenju umjetne inteligencije primjenjuje se jedan iz niza različitih oblika učenja. Kao najjednostavnija metoda izdvaja se metoda pokušaja i pogreške, gdje program isprobava različite opcije dok ne pronađe rješenje. Primjer takvog učenja je jednostavni program za rješavanje šahovskih problema mat-u-jedan gdje bi on isprobavao nasumične poteze sve dok ne bi pronašao mat. Program tada pohranjuje rješenje s položajem te sljedeći put kada se računalo nađe na istom položaju, pozove pohranjeno rješenje [1] [2].

Kako je srž ovog rada usmjerena na uporabu umjetne inteligencije, važno je u potpunosti temeljito razumjeti sve mogućnosti i primjene koje umjetna inteligencija pruža. Umjetna inteligencija, ovisno o perspektivi, se može klasificirati na različite načine. Kategorije umjetne inteligencije razgraničavaju se prema sposobnostima i načinu funkcioniranja, pri čemu se često koristi podjela na slabu umjetnu inteligenciju, snažnu umjetnu inteligenciju i super inteligenciju [3].

Slaba umjetna inteligencija (eng. *Narrow AI* - ANI) poznata još kao uska umjetna inteligencija obučena je i usmjerena za obavljanje specifičnih zadataka. Danas je upravo slaba umjetna inteligencija pokretač većine UI sustava. S druge strane, snažna umjetna inteligencija (eng. *Strong AI*) uključuje umjetnu opću inteligenciju (eng. *Artificial General Intelligence* - AGI) i umjetnu superinteligenciju (eng. *Artificial Superintelligence* - ASI). Umjetna opća inteligencija zamišljena je kao oblik umjetne inteligencije gdje bi stroj imao inteligenciju jednaku ljudskoj na razini samosvjesnosti kako bi imao sposobnost rješavanja problema, učenja i planiranja budućnosti. Umjetna super inteligencija nadmašila bi inteligenciju i sposobnosti ljudskog mozga. Iako snažna umjetna inteligencija još uvijek nema praktične primjene, mnogi istraživači aktivno rade na njenom razvoju [3].

U skladu s time, ciljevi umjetne inteligencije, posebno u kontekstu AGI-a jest proizvesti stroj čija se ukupna intelektualna sposobnost ne može razlikovati od sposobnosti ljudskog bića. Iako je danas veliki napredak u modelima velikog broja jezika, i dalje nije poznato može li AGI proizaći iz još moćnijih modela ili se radi o potpuno krivom pristupu. Veliki broj istraživača smatra kako ova vrsta umjetne inteligencije nije vrijedna traganja za daljnji razvoj. Unatoč skepticima, značajni uspjeh je postigla primijenjena umjetna inteligencija, poznata još kao napredne obrade informacija. Primijenjena umjetna inteligencija usmjerena je na stvaranje komercijalno održivih pametnih sustava. Primjer ove vrste sustava su stručni medicinski dijagnostički sustav i sustav za trgovanje dionicama. Kognitivna simulacija, još jedan važan alat u području UI-a koji koristi računala za testiranje teorija kako funkcionira ljudski um te teži prepoznavanju lica ljudi ili prisjećanje sjećanja [2].

Nakon razmatranja različitih kategorija umjetne inteligencije, važno je razumjeti i tehnološke pristupe koji omogućuju realizaciju slabe umjetne inteligencije. Budući da jaka umjetna inteligencija trenutno nema praktičnu implementaciju, rad je usmjeren ka slaboj umjetnoj inteligenciji koja razlikuje dvije ključne poddiscipline: strojno učenje i duboko učenje, pri čemu je duboko učenje poddisciplina strojnog učenja. Algoritmi strojnog i dubokog učenja

koriste neuronske mreže za “učenje” iz ogromnih količina podataka. Strojno i duboko učenje razlikuju se po vrstama neuronskih mreža koje koriste i količini ljudske intervencije [3].

Za potpuno razumijevanje strojnog i dubokog učenja, neophodno je razumjeti koncept neuronskih mreža. Neuronske mreže, su programske strukture modelirane po uzoru na procese donošenja odluka u ljudskom mozgu. Godine 1954. Belmont Farley i Wesley Clark uspjeli su pokrenuti prvu jednostavnu neuronsku mrežu. Neuronske mreže sastoje se od slojeva međusobno povezanih čvorova koji izdvajaju značajke iz podataka i daju predviđanja o tome što podaci predstavljaju [3] [4].

Klasični algoritmi strojnog učenja koriste neuronske mreže koje se sastoje od ulaznog sloja, jednog ili dva skrivena sloja i izlaznog sloja. Ljudski stručnjaci trebaju strukturirati podatke kako bi algoritam mogao izdvojiti bitne značajke iz podataka. Nasuprot tome, izumom tehnike pohlepnog slojevitog prije treninga (eng. *greedy layer-wise pretraining*) 2006. godine utvrđeno je kako je lakše trenirati svaki sloj neuronske mreže pojedinačno nego trenirati cijelu mrežu od ulaza do izlaza, kao što je slučaj kod strojnog učenja. Unapređenje u obuci neuronskih mreža je dovelo do razvoja algoritma dubokog učenja u kojem neuronske mreže imaju ulazni sloj, tri ili više skrivenih slojeva i izlazni sloj. Višestruki skriveni slojevi omogućuju učenje bez nadzora, automatizirajući izvlačenje informacija iz velikih količina neoznačenih i nestrukturiranih podataka. Ova vrsta algoritma ne zahtijeva ljudsku intervenciju što omogućuje dubokom učenju primjenu strojnog učenja u velikom opsegu [2] [3].

Danas se komercijalna primjena neuronskih mreža proširila izvan početnih područja kao što su prepoznavanje rukopisa, otkrivanje bombi i sl.. Proširila se na širok spektar industrija koje uključuju autonomna vozila, analizu podataka, generativnu umjetnost, razgovor s UI sustavima. Na primjer, Google koristi strojno učenje s pomoću kojeg se model trenira na velikom skupu podataka na kojem prepoznaje obrasce koji su povezani s neželjenom poštom. Na temelju tih obrazaca model automatski može prepoznati i filtrirati neželjenu poštu iz pristigle pošte korisnika Gmaila. Banke i kartične tvrtke koriste se povijesnim podacima za obuku modela za otkrivanje lažnih transakcija, a farmaceutska tvrtka Pfizer prilikom razvoja lijeka za COVID-19, Paxlovid, koristila je tehniku brzog pretraživanja među milijunima mogućih spojeva. Među postignućima dubokog učenja je napredak u klasifikaciji slika pri čemu se upotrebljavaju specijalizirane neuronske mreže nazvane konvolucijske neuronske mreže (eng. *Convolutional Neural Networks* - CNN). CNN modeli specijalizirani su za obradu vizualnih podataka. Ove mreže CNN treniraju se na prepoznavanju značajki koje se nalaze u slikama različitih tipova

objekata, kao što su oblici, boje i rubovi. Konkretno, CNN može uzeti ulaznu sliku i usporediti je sa značajkama naučenim iz prethodnih slika gdje prepoznaje objekte unutar njih ili izdvajajući važne informacije iz vizualnih podataka. Konvolucijska neuronska mreža koristi se u područjima prepoznavanja lica, rukopisa, autonomne vožnje i drugih aplikacija koje zahtijevaju analizu slika. Napredak mreže dokazuje činjenica da je jedna mreža, PReLU je klasificirala slike bolje nego što je to napravio čovjek [2] [4].

Jedno od najvažnijih područja primjene dubokog učenja je obrada prirodnog jezika. Obrada prirodnog jezika (eng. *Natural Language Processing* - NLP) je grana umjetne inteligencije koja se fokusira na omogućavanje računalima da razumiju i obrađuju ljudski jezik. Ona analizira načine na koji računala mogu obraditi i raščlaniti jezik nešto sličnije onom kako to rade ljudi. NLP modeli koriste računalnu lingvistiku, statistiku, strojno učenje i model dubokog učenja [2] [5].

Prvi NLP modeli ručno su kodirani i temeljeni na pravilima, zanemarujući iznimke u jeziku. Statistički NLP koristio je vjerojatnost za dodjeljivanje značenja različitim dijelovima teksta. Danas se NLP sustavi koriste modelima dubokog učenja i tehnikama koje im pomažu da „uče“ dok obrađuju informacije [2].

Primjer su jezični modeli (eng. *Language Models*) koji koriste umjetnu inteligenciju i statistiku za predviđanja konačnog oblika rečenice na temelju postojećih dijelova. Tako se razlikuje veliki jezični model (eng. *Large Language Model* - LLM) gdje se riječ „veliki“ odnosi na parametre, varijable i težine koje koriste modeli kako bi utjecali na ishod predviđanja. Skupovi podataka za obuku LLM-a mogu sadržavati od sto deset milijuna do tristo četrdeset milijardi parametara. Za obuku LLM-a koristi se ogromna količina podataka koja može biti veličine od više petabajta i sadržavati trilijune tokena, osnovnih jedinica teksta ili koda, duge nekoliko znakova, koje model obrađuje [2].

Nadalje, u lipnju 2022. godine je OpenAI objavio jedan od najpopularnijih jezičnih modela koji koriste umjetnu inteligenciju, GPT-3 (eng. *Generative Pre-trained Transformer 3*). GPT3 mogao je rješavati matematičke probleme na razini srednje škole i stvarati računalne programe. GPT3 bio je temelj softvera ChatGPT, kojeg je objavio OpenAI u studenom 2022. godine. GPT-3 jezični model, koji koristi ChatGPT, izračunava da je vjerojatnost da će se neki nizovi riječi pojaviti prije drugih [10]. Objavom ChatGPT-a pojavila se zabrinutost među akademikima zbog nemogućnosti razlikovanja ljudsko pisanja od teksta koje generira

ChatGPT. Jezični modeli proizvode tekst na temelju vjerojatnosti koja će se riječ pojaviti na temelju prethodne riječi u nizu [6] [7].

Nakon predstavljanja ChatGPT-a, uslijedio je nalet LLM-ova i chatbotova. Tako je Microsoft 2023. godine dodao chatbot Copilot u svoj operativni sustav 11, tražilicu Bing i preglednik Edge. Te iste godine Google objavljuje svoj chatbot pod nazivom Bard, koji je kasnije preimenovan u Gemini [2]. Iako efikasan, LLM danas ima sve više problema s halucinacijama. Umjesto da priopći korisniku kada nešto ne zna, model odgovara vjerojatnim, ali netočnim tekstom na temelju korisničkih uputa. Djelomično se problem pripisuje korištenju LLM-ova u ulozi tražilica, a ne njihovoj predviđenoj ulozi koja je generiranje teksta. U borbi protiv halucinacija dizajniraju se upute kojima je cilj izvući optimalne rezultate iz modela. Ovaj pristup poznat je kao brzo inženjerstvo (eng. *Prompt engineering*) [2]. Brzi inženjering je proces dizajniranja ulaznih podataka za modele generativne umjetne inteligencije za pružanje korisnih, točnih i relevantnih odgovora. Koristi se s generativnim UI modelima kao što su ChatGPT i Gemini da bi optimizirali rezultate koje dobivaju, korisnici često moraju koristiti brzi inženjering [8].

Generativna umjetna inteligencija je vrsta modela unutar dubokog učenja koja obrađuje neobrađene podatke kako bi generirala nove podatke, slične uzorke. Generativni UI modeli su modeli strojnog učenja koji mogu generirati tekst, slike i drugi sadržaj temeljem podataka na kojima su obučavani i mogu odgovoriti na upite koji im se daju prirodnim jezikom. Modeli mogu vraćati odgovore na temelju upita korisnika koji zadaje zahtjev, pitanje ili zadatak, čak i jednom riječju [3] [8].

Generativni modeli godinama su učili statističke vjerojatnosti s različitim ishodima poput stvaranja novih tekstova sličnih onima na Wikipediji. Iako su se generativni modeli godinama koristili u statistici za analizu numeričkih podataka, porast dubokog učenja omogućio je proširenje primjene na slike, govor i druge složene vrste podataka. Prvi model dubokog učenja koji se koristio za generiranje realističnih slika i govora bio je Varijacijski Autoencoder (eng. *Variational Autoencoder* - VAE) koji je predstavljen 2013. godine [3].

Iako umjetna inteligencija stvara mogućnosti za rješavanje raznolikih zadataka na različite načine i u brojnim formama, uz sve njene prednosti i dobrobiti ipak postoje i neke opasnosti. Upravo iz razloga što je UI toliko moćan alat, osim pozitivnih strana, valja istaknuti i one negativne koje se javljaju u ovom kontekstu.

Rizici umjetne inteligencije:

- Gubici radnih mjesta

Iako je umjetna inteligencija stvorila nove poslove, ona predstavlja i određene rizike u smislu etičkih i socioekonomskih posljedica gdje zbog sve veće automatizacije odvijanja poslovnih procesa, posebno u industrijama poput marketinga i zdravstvene njege, mnogi radnici strahuju da bi mogli izgubiti posao [2].

- Duboka lažna krivotvorina (engl. *Deepfake*)

Deepfake stavlja u prvi plan pitanje javne politike i etičkih normi što je vidljivo na primjeru iz siječnja 2024. godine kada je pjevačica Taylor Swift bila meta lažnog seksualnog eksplicitnog sadržaja koji je kružio internetom. Deepfake je posebno opasan kad mediji ne provjere izvore materijala koje koriste u izvještavanju pa se dogodi da deepfake materijali prezentiraju kao stvarnu sliku nekog događaja koji se nikad nije dogodio. Primjeri sveprisutnih deepfake sadržaja uključuju sliku bivšeg američkog predsjednika Donalda Trampa u tučnjavi s policajcima, sliku pape Franje u puffer jakni i video direktora Facebooka Marka Zuckerberga koji drži govor o moći svoje tvrtke [2].

- Privatnost

Umjetna inteligencija prikuplja i obrađuje veliku količinu podataka što nosi rizik da tim podacima pristupi 'treća strana' odnosno, neovlaštena osoba ili organizacija. Iako umjetna inteligencija napreduje, posebice u području generativne umjetne inteligencije, postoje mogućnosti manipulacije slikama i stvaranja lažnih profila [2].

2.3. Primjena umjetne inteligencije

Digitalni asistenti, globalni pozicijski sustavi (eng. *Global Positioning System* - GPS), autonomna vozila i generativni UI alati, kao što je ChatGPT, samo su neki od primjera UI u našem svakodnevnom okruženju [3].

U nastavku, izdvojene su primjene umjetne inteligencije iz različitih grana industrije i poslovanja. Primjerice, poznata banka JPMorgan Chase isprva je branila svojim zaposlenicima

korištenje ChatGPT-a, ali se ipak odlučila na veliki korak te im ga je dala na korištenje. Do promjene mišljenja je došlo zbog stava da bi ljudi trebali koristiti generativnu umjetnu inteligenciju kao pomoćnika koji odrađuje svakodnevne rutinske zadatke koje nitko ne želi raditi. Tako se zaposlenici mogu mnogo više posvetiti poslu veće vrijednosti [9].

Virtualni pomoćnici postaju sve više personalizirani i prilagođavaju se ponašanju pojedinog korisnika učeći iz njihovih interakcija. Primjer virtualnih pomoćnika na tržištu su Amazonovu Alexu, Googleov Assistant i Appleovu Siri [2].

Ostali primjeri strojeva koji koriste NLP su GPS sustavi s glasovnim upravljanjem, chatbotovi za korisničku službu i programi za prevođenje jezika. Osim toga, tvrtke koriste NLP kako bi unaprijedilo razumijevanje i pružanje usluga potrošačima automatskim dovršavanjem upita za pretraživanje i praćenjem društvenih medija. NLP modeli mogu izražavati pristranosti koje su prisutne u podacima na kojima su trenirani. Tako je u prošlosti, 2015. godine Amazonov NLP program je bio korišten za provjeru životopisa prilikom odabira kandidata za posao. Program je iskazao diskriminaciju prema ženama, odabrao ih je znatno manje u odnosu na muškarce. Razlog tome je što žene nisu bile dovoljno zastupljene u izvornom skupu podataka za obuku programa, koji je prikupljen od postojećih zaposlenika [6] [9].

Sada kada su objašnjeni svi pojmovi koji su ključni za razvoj pojma ChatGPT-a, nastavak poglavlja rezervirano je za razumijevanje pojma ChatGPT-a.

2.3.1. ChatGPT

OpenAI je 2022. godine predstavio generativni UI chatbot ChatGPT. ChatGPT temelji se na NLP modelu. Model NLP koristi vjerojatnost za predviđanje riječi ili rečenica koje korisnik očekuje. Zbog načina na koji ChatGPT odgovara, njegovi se odgovori ponekad ne razlikuje od ljudske komunikacije. Primarna svrha ChatGPTa je generiranje ljudskog teksta koji se može koristiti za razne aplikacije kao što su chatbotovi, automatizirano stvaranje sadržaja i prevođenje jezika [7].

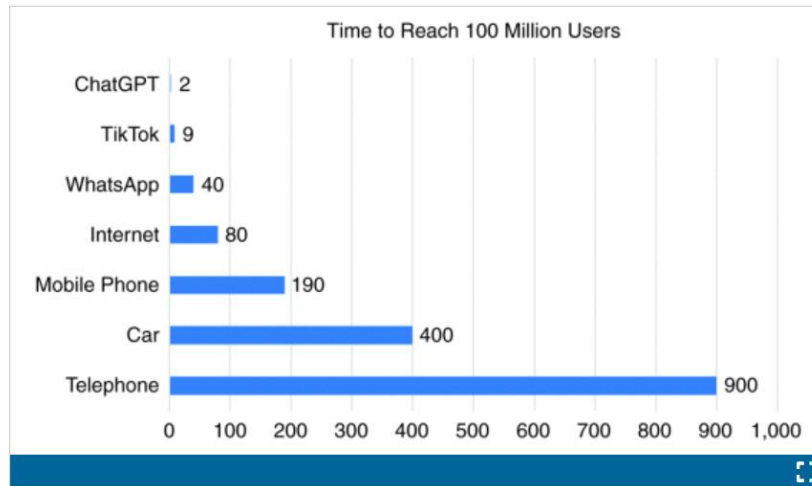
Unatoč tome što je ChatGPT treniran na velikom skupu podataka "cijelog interneta", nema mogućnost pristupa internetu za dohvaćanje najnovijih informacija u stvarnom vremenu te svoje znanje temelji na podacima do određenog datuma. Proces obuke naziva se samonadzirano učenje koje se postiže skrivanjem nekih dijelova poznatog teksta i provjerom kvalitete automatskog dovršavanja. Tako dekođer uči predviđati izlaz koji nedostaje na temelju

kodiranog ulaza. Nakon obuke, model prima upute i upite, a svaki upit će se kodirati kao i prije, a zatim se šalje u dekode. Dekoder radi samo s kodiranim ulazom jer su stvarni izlazi nepoznati [8] [10].

Iako ChatGPT model zapravo ne razumije jezik kao ljudi, već generira odgovore na temelju statističke vjerojatnosti sljedeće riječi ili fraze, on može tečno odgovoriti ljudskim jezikom na pitanja i izjave. Nadalje, ChatGPT generira tekst u različitim stilovima i formatima, kao što su novinski članci, e-pošta i poezija, što ga čini svestranim i korisnim za širok raspon aplikacija. Zanimljiva činjenica je da se više od milijun korisnika prijavilo za besplatni korisnički račun kako bi integrirali ChatGPT u roku od pet dana od njegovog uvođenja [2] [7].

OpenAI koristi korisničke informacije i podatke unesene u ChatGPT za obuku modela, čime potencijalno ili stvarno može narušiti osobnu povjerljivost korisnika ili njihovih poslodavaca. Potencijalno je ugrožena povjerljivost i pacijenata jer liječnici koriste ChatGPT za pisanje medicinskih bilješki koje učitavaju u sustav evidencije pacijenata. ChatGPT ima sposobnost generiranja uvjerljivih, ljudskih informacija i savjeta, ali s time dolazi i rizik da uvjeri korisnika kako je na neki način "osjećajan". OpenAI za razvoj ChatGPT-a uključuje slabo plaćene radnike iz zemalja trećeg svijeta, ali s ograničenim pravima [11].

Objava ChatGPT-a širokoj javnosti 2022. godine započela je neviđenu utrku u razvoju umjetne inteligencije izvanrednom brzinom. U samo dva mjeseca ChatGPT je dosegno sto milijuna korisnika. Za usporedbu, tehnologije poput kotača trebale su tisuće godina da dosegnu široku primjenu. Slika 2. prikazuje evoluciju različitih tehnologija u posljednjih sto godina. Na x-osi su prikazani mjeseci, dok y-os prikazuje različite tehnologije prema vremenu potrebnom da dosegnu sto milijuna korisnika. Nakon ChatGPT-a, najbrže rastuća tehnologija po broju korisnika bio je TikTok. TikTok je dosegno sto milijuna aktivnih korisnika nakon devet mjeseci. WhatsApp je taj broj dosegno nakon četrdeset mjeseci, dok je internetu bilo potrebno šest godina. Šesnaest godina bilo je potrebno da mobilni telefoni dosegnu sto milijuna korisnika, dok je automobilima trebalo nešto više od trideset tri godine. Više od trideset godina bilo je potrebno da milijun ljudi posjeduje automobil, što prikazuje sporiji tempo usvajanja ranih tehnologija u usporedbi s današnjim brzim tehnološkim napretkom [10].



Slika 2. Vremenski prikaz koliko je mjeseci bilo potrebno tehnologijama doći do 100 milijuna korisnika [izvor: preuzeto s „Generative AI for Software Practitioners“, IEE Xplore]

OpenAI je 9. travnja 2024. godine predstavio novu verziju GPT-4 Turbo modela, koji integrira multimodalnu obradu GPT-4 za prepoznavanje sadržaja slika izravno u model, a inicijalno je pokrenut putem API pristupa. Novi model GPT-4 Turbo postao je dostupan korisnicima od 11. travnja 2024. godine. Novi model unapređuje sposobnosti u pisanju, logičkom zaključivanju i kodiranju te koristi izravniji, manje opširan jezik s razgovornijim stilom [36]. Prethodna verzija ponekad je odbijala odgovoriti na upite ili dovršiti projekte kodiranja prema zahtjevu, što je OpenAI unaprijedio u ovoj verziji [12].

8. kolovoza 2024. godine predstavljena je sustavna kartica za ChatGPT-ov novi GPT-4o model koji detaljno opisuje ograničenja modela i postupke testiranja sigurnosti. Dokument otkriva kako je prilikom testiranja napredni glasovni način rada modela u rijetkim slučajevima nenamjerno oponašao glasove korisnika bez dopuštenja što ukazuje na složenost sigurnog projektiranja sustava umjetne inteligencije. Iako je OpenAI implementirao zaštitne mjere kako bi spriječio takve incidente, model može sintetizirati širok raspon zvukova uključujući zvučne efekte, glazbu i glasove. Mjere sigurnosti su vrlo važne, no tehnologija glasovne sinteze brzo napreduje i uskoro bi mogla postati dostupna širem krugu korisnika, čime se otvaraju nova pitanja o sigurnosti i etici [13].

Primjena ChatGPT-a je široka. U nastavku su prikazani primjeri primjene ChatGPT-a:

- Korištenje ChatGPT-a nije zaobišlo ni profesore, koji su otkrili da ChatGPT može položiti napredne ispite na fakultetu, no iznenađenje je da mu se potkradu greške u jednostavnim izračunima na razini matematike šestog razreda [7].
- Brojni nastavnici zabrinuti su da će učenici varati koristeći ChatGPT za pisanje eseja, uslijed čega su neki nastavnici ukinuli eseje kao domaći rad. Iako je široko prihvaćen, ima reputaciju da odgovara na načine koji su pristrani prema očekivanjima korisnika što dovodi do toga da ChatGPT povremeno daje lažne ili pogrešne informacije [14] [7].
- Nizozemski sudac u Nijmegenu priznao je da je koristio ChatGPT za prikupljanje informacija tijekom odlučivanja o slučaju. Upotrijebio je ChatGPT za odgovor o solarnim pločama te dobivene informacije koristio je za generiranje presude što je izazvalo zabrinutost tehničkih i pravnih stručnjaka koji preispituju je li to odgovorno učiniti [15].
- Istraživači s ETH Zurich pokazali su da veliki jezični modeli, poput GPT-4 koji pokreće ChatGPT, mogu identificirati dob, lokaciju, spol i prihod pojedinca s do osamdeset pet posto točnosti jednostavnom analizom njihovih objava na Redditu [11].
- Danas, 2024. godine, iako je umjetna inteligencija dosegla značajan napredak, još uvijek nije napredovala do točke u kojoj bi potpuno autonomni automobili bili dostupni za kupnju i sigurnu integraciju u svakodnevni promet [11].

Ipak, ChatGPT zna dati netočne informacije, kao što su:

- Generirao je lažne citate u znanstvenom radu o stonogama u ime danskog akademika Henrika Enghoffa.
- Izmislio članke i naslove u ime novina The Guardian.
- Dao netočne odgovore na upite o lijekovima.
- Pogrešno dijagnosticirao osam od deset pedijatrijskih studija slučaja.
- S vremenom generira sve manje ispravan računalni kod.
- Prema istraživanju Sveučilišta Purdue, pogrešno je odgovorio na više od polovice pitanja o softverskom inženjerstvu [11].

ChatGPT kritiziran je zbog svoje netočnosti, nepouzdanosti, pristranosti, povjerljivosti te negativnim utjecajima na politiku, zapošljavanje i okoliš.

- Istraživački studiji pokazuju kako je malo informacija dostupno javno o ChatGPT-u i o načinu na koji funkcionira.
- Rizik za radna mjesta u nizu industrija, od prava, marketinga, novinara i softverskih programera.
- Istraživači iz Kalifornije procjenjuju da je potrošnja energije ChatGPT-a za jednostavan razgovor od dvadeset do pedeset pitanja i odgovora ekvivalentna energiji koja je potrebna za zagrijavanje petsto mililitara vode.
- Prema istraživanju zajednice UI Hugging Face procjenjuje se da je treniranje ChatGPT-a generiralo emisije koje premašuju petsto metričkih tona ugljika.
- Generiranje dezinformacija koje zvuče uvjerljivo može naštetiti dobrobiti zajednice [11].

Talijanski regulator privatnosti zabranio je ChatGPT zbog potencijalnog kršenja prava privatnosti Talijana.

Koliko je umjetna inteligencija jaka govori i postavljanje nekoliko zakona koji reguliraju UI-u. Od 2024. godine, Opća uredba Europske unije o zaštiti podataka (eng. *General Data Protection Regulation*, GDPR) i Kalifornijski zakon o privatnosti potrošača (eng. *California Consumer Privacy Act*, CCPA) reguliraju modele umjetne inteligencije, ali samo u mjeri u kojoj koriste osobne podatke. Najopširniji propis je EU-ov umjetne inteligencije „Act“ koji je donesen u ožujku 2024. Prema ovom aktu, zabranjeni su modeli koji vrše socijalno bodovanje ponašanja i karakteristika građana te koji pokušavaju manipulirati ponašanjem korisnika. Modeli umjetne inteligencije koji se bave temama "visokog rizika", kao što su provedba zakona i infrastruktura, moraju biti registrirani u bazi podataka EU-a [2].

Izlazak svake nove verzije umjetne inteligencije praćen je velikim zanimanjem diljem svijeta. Ljudi pomno prate svaku novost, a interes nikada nije bio veći. Svi dijele istu želju, a to je otkriti gdje su granice umjetne inteligencije, ako ih uopće ima. Svaka nova verzija donosi unapređenja kako bi se ispravili nedostaci prethodnih verzija, s ciljem sprječavanja dezinformacija i zloupotrebe informacija.

3. Programsko inženjerstvo

U ovom poglavlju obrađuje se programsko inženjerstvo s fokusom na ključne aspekte, kao što su projektiranje, razvoj, testiranje i održavanje softvera. Razmotreni su temeljni principi i metodologije koje osiguravaju stvaranje visokokvalitetnog i pouzdanog softvera, kao i uloga novih tehnologija, alata i tehnika u kontinuiranom unapređenju procesa razvoja softverskih aplikacija.

3.1. Osnovni i ključni aspekti programskog inženjerstva

Programsko inženjerstvo je proces projektiranja, razvoja, testiranja i održavanja softvera. Za cilj ima stvaranje visokokvalitetnog, pouzdanog softvera koji se može održavati. Uključuje niz tehnika, alata i metodologija, uključujući analizu zahtjeva, dizajn, testiranje i održavanje. Programsko inženjerstvo, područje je koje se brzo razvija, gdje se neprestano razvijaju i novi alati i tehnologije kako bi se unaprijedio proces razvoja softvera. Glavna svrha softverskog inženjeringa je razvoj softverskih aplikacija koje unapređuju kvalitetu, troškovnu efikasnost i brzinu izvršenja. Također, softverski inženjering osigurava da se razvoj softvera odvija dosljedno, točno, unutar zadanog proračuna, vremenskog okvira i u skladu sa zahtjevima [16].

Glavna svojstva softverskog inženjerstva su:

- Učinkovitost (Omogućava učinkovitu procjenu resursa potrebnih za razvoj softverskog proizvoda.)
- Pouzdanost (U isto vrijeme na istom mjestu proizvod će dati isti rezultat.)
- Ponovno korištenje (Ponovno korištenje omogućuje korištenje modula u više aplikacija.)
- Mogućnost održavanja (Svojstvo mogućnost održavanja je sposobnost softvera da se lako modificira, popravi ili unaprijedi s promjenjivim zahtjevima.) [16].

Softversko inženjerstvo postavlja niz ciljeva koji osiguravaju kvalitetu i učinkovitost pri razvoju softverskih aplikacija. Neki od osnovnih, a svakako i vrlo važnih ciljeva su: održavanje, učinkovitost, točnost, testabilnost, pouzdanost, prenosivost, prilagodljivost i ponovna upotrebljivost. Cilj održavanja podrazumijeva da je sve podake potrebno optimizirati i redizajnirati kada bude potreba za time. Cilj učinkovitosti podrazumijeva da je ISP (intellectual service provider – društvo koje pruža intelektualne usluge) odgovorno za korištenje svih

resursa koji su na raspolaganju za izradu softverskog proizvoda, a da bi taj sustav uspješno radio. Cilj točnosti podrazumijeva da softver mora biti točno implementiran i mora svoj posao raditi onako kako je definirano u specifikaciji dizajna softvera (eng. *software design specification* – SDS) [16].

Softverski proizvod pokazuje visoku razinu ponovne upotrebljivosti ako se njegovi moduli mogu lako i brzo adaptirati i koristiti za razvoj novih sustava ili aplikacija. Osnovni zadatak koji bi trebalo riješiti kod ponovne upotrebe softvera jest pitanje ponovne upotrebe modula, jer se pokazalo, i iskustveno i teorijski, da moduli, odnosno zahtjevi koji se postavljaju modulima, jako utječu na ponovnu upotrebu. Ne može se govoriti o ponovnoj upotrebi modula ako se prethodno ne govori o defektima modula. Ove ciljeve, koji su ključni za kvalitetu softvera, postizemo korištenjem posebnog pristupa u projektiranju sustava. Naime, pri projektiranju sustava primjenjujemo modularni pristup. Zahvaljujući modulima i njihovoj interakciji s ostalim modulima, ostvarujemo softver različitih funkcionalnosti [16].

3.2. Primjena umjetne inteligencije u softverskom inženjerstvu

Već prethodno spomenuti alati generativne umjetne inteligencije, kao što su ChatGPT, Copilot i Bard brzo su stekli široku upotrebu te imaju potencijal značajno povećati produktivnosti u softverskom inženjerstvu. Generativna umjetna inteligencija, od svih novijih tehnologija, ima potencijal najviše promijeniti softversku profesiju. Bill Gates to vidi kao najveći korak naprijed od izuma Interneta. Generativna umjetna inteligencija može unaprijediti produktivnost softvera na nekoliko načina:

- Automatizacija ponavljajućih zadataka kao što su testiranje zahtjeva
- Poboljšanje kvalitete softvera generiranjem testnih paketa iz zahtjeva [10].

Generativna umjetna inteligencija stvara rizike jer nije niti deterministička niti objašnjiva što ograničava njezinu upotrebu u profesionalnom softverskom inženjerstvu. Neki od ključnih izazova uključuju:

- prava intelektualnog vlasništva
- kibernetičku sigurnost [10].

Za programere je uobičajeno i normalno oslanjati se na internetske resurse, kao što su Google i StackOverflow kako bi riješili nedoumice, pronašli rješenja za probleme kodiranja, dobili odgovore na tehnička pitanja ili se informirali o različitim drugim izazovima s kojima se svakodnevno susreću. StackOverflow je internetska stranica sa savjetima zajednice koja pruža obrazložena rješenja i vrijedne komentare na pitanja korisnika. Web stranica i tražilice omogućuju pristup informacijama koje su tamo već pohranjene. Generativna umjetna inteligencija odgovara na pitanja koje mu se postavi, kao što joj ime govori, može sintetizirati, odnosno generirati odgovore. Umjesto traženja gotovih odgovora kao što je slučaja s radom klasične tražilice, generativna UI stvara odgovor za korisnika. Odgovori generativne umjetne inteligencije temelje se na golemoj količini podataka na kojima je obučena. Kako bi pružila smislene odgovore, UI prolazi obuku koja se temelji na povratnim informacijama ljudi. Navedena vrsta učenja, gdje treneri postavljaju pitanja i daju povratne informacije o generiranim odgovorima, ima za cilj pružiti što točniji odgovor, štiteći istovremeno od štetnih odgovora. Ovaj način rada dovelo je do brzog inženjeringa [10].

Postoji nekoliko generativnih softverskih platformi koje omogućuju pretvaranje jednostavnih uputa u računalni kod. GitHubov Copilot dostupan je kao proširenje za razvojne alate i uređivače, kao što je Visual Studio Code. Copilot nudi automatsko dovršavanje koda pokrenuto OpenAI Codexom. Najava Copilot X-a i pokrenut ChatGPT-4 predstavljaju značajan napredak. ChatGPT-4 uz unapređeno dovršavanje koda, može pomoći u drugim razvojnim zadacima kao što su:

- razumijevanje koda
- poboljšanje zahtjeva za povlačenjem (engl. *Pull request*)
- pomoć u skriptiranju
- podrška u radu s naredbenim retkom [10].

Slika 3. prikazuje usporedbu četiri alata za automatsko dovršavanje i generiranje koda s pomoću umjetne inteligencije.

Name	URL	Technology	Cost	Use cases
ChatGPT	https://chat.openai.com/	GPT-4	USD\$20 per month for the chat interface; pricing for application programming interface use depends on usage.	Code completion, code generation, code comprehension, reverse engineering, pseudo code reasoning and execution.
CoPilot	https://github.com/features/copilot	OpenAI CodeX, GPT-4	USD\$10 per month/USD\$100 per year for individuals, \$19 per user per month for business plans.	Code completion for CoPilot; CoPilot X uses the more advanced GPT-4 model and can answer questions based on code documentation; aid in pull requests, shell commands, and scripting.
Tabnine	https://www.tabnine.com/	Proprietary ML engine trained with OSS	Basic tier is free, Pro tier starts at USD\$12 month/user; also possible to self-host	Code completion. Runs also on private desktop to protect IPR.
Hugging Face (various different models)	https://huggingface.co/docs/transformers/index	Transformer, details vary depending on the model	Free and open source	Code completion and code generation, depending on the model.

Slika 3. Usporedba alata za automatsko dovršavanje i generiranje koda koristeći UI (izvor: preuzeto s „Generative AI for Software Practitioners“, IEE Xplore.)

Za usporedbu alata korištene su najvažnije karakteristike koje su važne pri odabiru alata: Naziv alata, URL do alata, Tehnologija koja se koristi u alatu, Trošak i Upotreba. Ovi alati nude širok raspon mogućnosti prilagođenih potrebama programera. Primjerice, ChatGPT i Copilot temelje se na naprednim modelima GPT-4, pružajući robusne funkcionalnosti poput generiranja koda, razumijevanja koda i integracije u radne procese poput skriptiranja i zahtjeva za povlačenjem (eng. *pull request*). S druge strane, Tabnine se ističe svojim fokusom na zaštitu intelektualnog vlasništva, omogućujući lokalno hostanje, dok Hugging Face nudi besplatne, modele otvorenog koda (eng. *open-source*) s fleksibilnim opcijama za dovršavanje i generiranje koda. Usporedba pokazuje kako odabirom bilo kojeg od ova četiri alata mogu unaprijediti produktivnost programera, a sami odabir ovisi o specifičnim potrebama korisnika [10].

Generativna umjetna inteligencija ima potencijal unaprijediti produktivnost softvera:

- automatiziranjem zadataka
- povećanjem kreativnosti
- unapređenjem točnosti i učinkovitosti
- pojednostavljenjem razvojnih procesa [10].

Ključna područja primjene koja mogu imati značajan utjecaj na poslovanja uključuje:

- stvaranje medijskih sadržaja poput teksta, slike, zvuka te sadržaja za vijesti i društvene mreže
- unapređenje medijskog sadržaja uključujući poboljšanja, referenciranje i prošireno generiranje sadržaja uz naplatu po korištenju
- razvoj softvera za generiranje koda, generiranje testnih slučajeva iz zahtjeva, ponovna uspostava sljedivosti, objašnjavanje koda, refaktoriranje naslijeđenog koda, održavanje softvera s proširenim smjernicama i unapređenje postojećeg koda [10].

Generativna umjetna inteligencija može unaprijediti produktivnost softvera u različitim aspektima softverskog inženjerstva:

1. Sažimanje dokumentacije, intervjui, zapisnici sa sastanaka:

Generativna umjetna inteligencija oslobađa korisnike za odrađivanje drugih zadataka. Odgovara na upite na način da ih prilagođava ili definira za različite zadatke kao što su automatsko pretraživanje ili unapređenje [10].

2. Unaprijediti kreativnosti:

Generativna umjetna inteligencija pomaže programerima u stvaranju novih ideja i rješenja za razvoj softvera i korisničkog doživljaja. Pomaže programerima u stvaranju dizajna, korisničkog sučelja [10].

3. Rješavanje problema:

Generativna umjetna inteligencija ima sposobnost rješavanja matematičkih i algoritamski problema. Programeri opisuju problem koji treba riješiti, a UI pruža smjernice kako riješiti problem generiranjem koda za zadatak. Ako je zahtjev detaljniji i netrivialan, onda je veća vjerojatnost da kod sadrži greške [10].

4. Učinkovit razvoj:

Generativna umjetna inteligencija može se koristiti za pojednostavljenje procesa razvoja softvera automatiziranjem zadataka, kao što su testiranje, otklanjanje pogrešaka i implementacija [10].

5. Unapređenje kvalitete softvera

Generativna umjetna inteligencija može analizirati velike količine podataka i identificirati obrasce koje su možda programeri propustili. Primjer je odabir odgovarajućih testnih slučajeva što može pomoći programerima da napišu točniji i učinkovitiji kod te brže identificiraju i poprave greške [10].

6. Unapređenje kvalitete podataka

Ključna prednost korištenja generativne umjetne inteligencije za zadatke vezane uz softver je u mogućnosti prilagodbe postojećeg modela na specifičnim podacima. Podaci koji se koriste za treniranje LLM-ova preuzeti su s interneta, a njihovi odgovori temelje se na informacijama koje mogu naučiti iz tih podataka. Svaki model dodatno se trenira kako bi nadogradili svoju prethodnu verziju [10].

7. Postizanje povjerenja

Postizanje povjerenja u moderne UI sustave zahtijeva realistične simulacije i testiranja, jer se spomenuti sustavi mogu lakše prilagoditi te mogu odstupati od početnih specifikacija. Sintetički podaci generirani UI-em olakšavaju razvoj, testiranje i odobrenje autonomnih sustava u ključnim industrijama poput medicine, zrakoplovstva i mobilnosti [10].

Iako generativna umjetna inteligencija može pomoći tvrtkama povećati sposobnosti, postoji nekoliko rizika koje je potrebno uzeti u razmatranje: alati poput ChatGPT-a mogu zamijeniti neovisne medije i širiti dezinformacije koje je teško provjeriti. Iz tog razloga programeri moraju aktivno upravljati navedenim rizicima. Iako su se stvari unaprijedile u novoj verziji ChatGPT 4, korisnici i dalje trebaju provjeriti odgovore. Za razvoj ovog softvera to znači da je još uvijek potreban ljudski nadzor. Nadalje; kada su korisnici počeli objavljivati ispravke grešaka generiranih s pomoću alata poput ChatGPT-a, StackOverflow ih je zabranio. Prepoznavanje lažnog sadržaja postalo je utrka u naoružanju, a postoje učinkoviti alati koji koriste statističke analize za identifikaciju generiranih dokumenata umjetne inteligencije. Stručnjaci trebaju biti svjesni rizika za privatnost i sigurnost pri korištenju alata za analizu koda. Kod otvorenog koda to nije jako važno, ali vlasnički kod može izaći izvan granica privatnih repozitorija. Različiti alati nude različita jamstva, stoga je ključno pažljivo pročitati uvjete korištenja. U današnje vrijeme teško je razlikovati originalne radove od lažnih, poput slika i videa korištenih za dezinformacije. Stručnjaci moraju biti svjesni privatnosti i sigurnosti kod korištenja UI alata za analizu koda.

Kibernetička sigurnost generiranog koda predstavlja rizik jer UI alati mogu nenamjerno umetnuti ranjivosti. Softverski inženjeri moraju definirati granice dozvoljenog ponašanja UI sustava, kako bi spriječili zlouporabu i osigurali sigurnost. Generativna UI brzo mijenja industriju, stoga će programeri trebati nove vještine za rad s automatski generiranim softverom. Iako UI ubrzava razvoj, oprez je potreban u osiguravanju da softver ostane siguran i pouzdan [10].

ChatGPT 4 može generirati kod iz nizova dokumenata i rješavati pitanja kodiranja za softversko inženjerstvo na razini ljudskih performansi ili ih čak nadmašuje. Sposobnosti ChatGPTa 4 su:

- programiranje front-end i komunikacija s LaTeXom
- obrnuti inženjerski kod
- izvršiti Python kod
- izvršiti pseudo kod [10].

OpenAI nudi programski pristup svojim LLM-ovima kako bi ih programeri osim za razgovor, mogli i integrirati u svoje aplikacije. Također, omogućava razvoj dodataka koji povezuju temeljne modele s uslugama treće strane [10].

Model Tabnine ističe se od konkurencije po dovršavanju koda s posebnom pažnjom na pitanja licenciranja i privatnosti. Ovaj model obučen je isključivo za softver otvorenog koda s dopuštenim licencama čime programerima osigurava da se njihov kod ne zadržava nakon što ga podijele s Tabnineom. Uz navedeno, model Tabnine omogućuje preuzimanje i korištenje temeljnih modela lokalno, umjesto da im se pristupa isključivo kao usluzi u oblaku [10].

Ako nekog zanima trenutni poredak velikih jezičnih modela, tu je LMSYS Chatbot arena platforma otvorenog tipa koja uživo prikazuje rang listu velikih jezičnih modela LLM-a. Za potrebe ove liste prikupljeno je preko milijun usporedbi parova koje su napravili korisnici kako se rangirali LLM-ovi [10].

Na slici 4. prikazana je rang lista jezičnih modela s podacima o imenu modela, bodovima u areni, broju glasanja, organizaciji koja je izradila model, licenci i datumu do kojeg je model treniran.

Rank* (UB)	Model	Arena Score	95% CI	Votes	Organization	License	Knowledge Cutoff
1	ChatGPT-4o-Latest-(2024-08-08)	1314	+6/-5	11555	OpenAI	Proprietary	2023/10
2	Gemini-1.5-Pro-Exp-0801	1297	+4/-4	20674	Google	Proprietary	2023/11
3	GPT-4o-2024-05-13	1286	+2/-3	78496	OpenAI	Proprietary	2023/10
4	GPT-4o-mini-2024-07-18	1274	+5/-3	20089	OpenAI	Proprietary	2023/10
4	Claude-3.5-Sonnet	1271	+3/-3	48546	Anthropic	Proprietary	2024/4
4	Gemini-Advanced-App-(2024-05-14)	1266	+4/-3	52249	Google	Proprietary	Online
5	Meta-Llama-3.1-405b-Instinct	1263	+5/-4	19909	Meta	Llama 3.1 Community	2023/12

Slika 4. Lista modela umjetne inteligencije na platformi LMSYS Chatbot arena (izvor: preuzeto s „LMSYS Chatbot Arena Leaderboard“)

Na slici 4. se vidi da se na prvom mjestu, 16. kolovoza 2024. godine kada je stranici pristupano, nalazi ChatGPT-4o koji je treniran do 8. kolovoza 2024. godine. Trenutni pregled podložen je promjenama [17].

Ključne informacije prikazanih na slici 4. su:

- o ukupan broj modela je 128
- o ukupan broj glasova 1 671 145 korisnika
- o zadnje ažuriranje rang liste: 12. kolovoza 2024 [17].

Umjetna inteligencija vrlo brzo postaje važan dio modernog softverskog inženjerstva, s posebnim naglaskom na osiguranje kvalitete (eng. *Quality Assurance* - QA). Tradicionalne metode QA često nisu dovoljne za suočavanje s izazovima koje donosi sve veća složenost softverskih sustava. Prema definiciji iz knjige o softverskom inženjerstvu “Kvaliteta softvera definira se kao sposobnost softverskog proizvoda da zadovolji navedene i implicitne potrebe kada se koristi u određenim uvjetima.” Stoga, UI donosi niz inovativnih rješenja koja mogu unaprijediti proces testiranja, identifikacije grešaka, te općenito povećati pouzdanost i kvalitetu softverskih proizvoda [18].

4. Uspostava i održavanje kvalitete

Ovaj dio rada bavi konceptom osiguranja kvalitete, fokusirajući se na različite norme koje su ključne za implementaciju i održavanje visokih standarda u različitim industrijama. Poseban naglasak stavlja se na međunarodne ISO norme koji predstavljaju okosnicu za uspostavljanje globalno priznatih praksi. Razmatraju se ISO/IEC 27001 - Informacijska tehnologija – Sigurnosne tehnike – Sustavi upravljanja sigurnošću informacija – Zahtjevi (u daljnjem tekstu: ISO/IEC 27001), koji se tiče sigurnosti informacija, ISO/IEC 25010 – Inženjering sustava i softvera – Zahtjevi i evaluacija kvalitete sustava i softvera (SQuaRE) – Modeli kvalitete sustava i softvera (u daljnjem tekst: ISO/IEC 25010), koji ocjenjuje kvalitetu softverskih sistema, i ISO/IEC/IEEE 29119 - Inženjering softvera i sustava – Testiranje softvera (u daljnjem tekst: ISO/IEC/IEEE 29119), koji standardizira procese testiranja softvera. Također, rad istražuje ulogu umjetne inteligencije u osiguravanju kvalitete softverskog inženjerstva, pokazujući kako UI transformira tradicionalne metode testiranja i održavanja, povećavajući efikasnost i preciznost.

4.1. Osiguravanje kvalitete

Prilikom osnivanja poduzeća definiraju ciljeve važne za poduzeće koje planiraju postići. Svako poduzeće u barem jednom cilju naglašava kako mu je najvažnije zadovoljstvo korisnika ili da mu je korisnik na prvom mjestu. Svakom korisniku važno je povjerenje i pouzdanost poduzeća od kojeg kupuje proizvod. Ako korisnik ne stekne povjerenje prema u poduzeće, ono ne može uspjeti u daljnjem poslovanju. Povjerenje je važno steći, jer se ono ne daje bez opravdanja.

Danas se u poslovnom svijetu stalno nešto razvija, što dovodi do čestih promjena očekivanja potrošača na globalnoj razini. Povjerenje potrošača se stječe predanošću, dosljednosti, pouzdanosti i kvalitetnom upravljanja poduzeća. Osiguranje kvalitete obuhvaća svaki element poslovanja organizacije, što pokazuje koliko je osiguravanje kvalitete, jer je ono dio svakog procesa, od proizvodnje do isporuke ili potrošnje dobara i usluga. Tim za upravljanja kvalitetom uključen je u sve faze razvoja: proizvodnju, testiranje, implementaciju i isporuku [19].

Primarni cilj je riješiti greške i smanjiti rizik od nedostataka što je moguće ranije u lancu vrijednosti. Ovo smanjenje broja nedostataka u ranoj fazi smanjuje potrebu za složenim popravcima u završnoj fazi, kada ispravljanje zahtjeva više vremena i povećava financijske troškove. Brže rješavanje grešaka može spriječiti odgodu razvoja i pomicanje rokova za klijente, čime se izbjegava gubitak njihovih povjerenja [19].

Kako bi se osigurala kvaliteta, sustav osiguranja kvalitete provodi tehničke i upravljačke procese za učinkovito praćenje i unapređenje kvalitete proizvoda ili usluge. Ti procesi uključuju testiranja proizvoda, ankete zaposlenika i procjene sigurnosti opreme. Implementacijom svih elemenata osigurava se održavanje i unapređenje standarda kvalitete organizacije [19].

Najbolje prakse za osiguranje kvalitete uključuju snažnu predanost i podršku vodstva, detaljno planiranje kvalitete u dizajnu proizvoda i razvoju procesa, strogu dokumentaciju procesa i kontrolu, validaciju korektivnih radnji te opsežnu suradnju i komunikaciju među timovima [19].

U osiguranju kvalitete analiziraju se potencijalno negativni utjecaji na kvalitetu krajnjeg proizvoda, a zatim se, u najboljem slučaju, eliminiraju. Osiguranje kvalitete primjenjuje prakse i procese za provedbu te strategije. Zajedno, ove koordinirane aktivnosti usmjerene su na vođenje i kontrolu procesa u potrazi za kvalitetom. Sustav osiguranja kvalitete naglašava kredibilitet organizacije, dok istovremeno unapređuje radne procese i učinkovitost. ISO norme igraju ključnu ulogu u osiguravanju kvalitete, osobito kroz normu ISO 9000 Sustavi upravljanja kvalitetom - *Temeljna načela i terminološki rječnik* (eng. *Quality management systems — Fundamentals and vocabulary*) [19].

Organizacije na globalnim tržištima ulažu u osiguranje kvalitete te ostvaruju prednosti nad konkurentima. Ulaganja umanjuju rizike, proaktivno smanjuju šanse za nedostatke, rade prema dosljednom standardu kvalitete, grade povjerenje među potrošačima, jačaju svoju poziciju u industriji i ističu se u ključnim poslovnim pokazateljima [19].

Osiguranje kvalitete softvera (eng. *software quality assurance* - SQA) obuhvaća sustavne postupke koji unapređuju razvojne cikluse i sprječavaju razvoj grešaka, čime se štedi vrijeme i smanjuju troškovi. Iako SQA proces pomaže u smanjenju grešaka, ažuriranja softvera mogu dovesti do novih problema. SQA koristi različite strategije, kao što je modela CMMI (eng. *capability maturity model integration*), koji rangira zrelost organizacije i identificira mogućnosti

za unapređenjem. Različite razvojne metodologije, poput Waterfalla, Agile i Scruma, oslanjaju se na SQA kako bi optimizirale učinkovitost [19].

Timovi za osiguravanje kvalitete obuhvaćaju SQA analitičare i stručnjake za automatizaciju testiranja. Alati, poput Seleniuma, Jenkinsa i Postmana, koriste se za automatizaciju testiranja i postizanje SQA ciljeva [19].

4.2. ISO norme

ISO je Međunarodna organizacija za normatizaciju koja na globalnoj razini okuplja stručnjake kako bi dogovorili o najboljem načinu obavljanja stvari, od izrade proizvoda do upravljanja procesom. Jedna je od najstarijih nevladinih organizacija, od 1946. godine, koja omogućuje trgovinu i suradnju među ljudima i tvrtkama diljem svijeta. Standardi koje objavljuje ISO pomažu učiniti život lakšim, sigurnijim i boljim [20].

Povijest ISO normi

- 1946. godine sastalo se šezdeset i pet delegata iz dvadeset i pet zemalja, u Londonu, kako bi raspravljali o budućnosti međunarodne standardizacije.
- 1947. godine službeno je osnovan ISO sa šezdeset i sedam grupa stručnjaka fokusiranih na određenu temu.
- 1951. godine 1. siječnja objavljen je prvi ISO norma ISO: R 1:1951 Standardna referentna temperatura za industrijska mjerenja duljine.
- 1960-ih godina ISO radi na uključivanju više zemalja u razvoj u svoj rad na međunarodnoj normatizaciji.
- 1971. godine osnovana su prva dva tehnička odbora u području okoliša, fokusirani na kvalitetu zraka vode.
- 1987. godine ISO objavljuje svoj prvu normu upravljanja kvalitetom.
- 1995. godine ISO pokreće web stranicu.
- 1996. godine ISO pokreće svoju normu za sustav upravljanja okolišem, ISO 14001.
- 2000. godine ISO počinje prodavati svoje norme putem interneta.
- 2005. godine pokreće se ISO/IEC 27001 norma za upravljanje sigurnošću informacija.
- 2010. godina pokreće se ISO 26000 norma, prva međunarodna norma koja pruža smjernice za društvenu odgovornost.
- 2015. godina usvajanje najnovijih tehnologija.

- 2017. godine postoji preko dvije tisuće i sto normi.
- 2018. godine pokreće se nova međunarodna norma ISO 25001:2018, Sustav upravljanja zdravljem i sigurnošću na radu, kako bi pomogao organizacijama svih veličina u smanjenju ozljeda i bolesti na radnom mjestu.
- 2019. godine tijekom pandemije COVID-19 objavljena je norma za očuvanje sigurnosti i zdravlja na radnom mjestu [20].

Temelji na kojima počiva Međunarodna organizacija za normatizaciju normi ISO su:

- Inkluzivan (eng. *inclusive*)

Cilj procesa razvoja normi je stvoriti norme koje su relevantne i primjenjive na globalnoj razini, uzimajući u obzir komentare svih dionika.

- Usmjeren na vrijednosti (eng. *value-driven*)

ISO norme služe učiniti život lakšim, sigurnijim i boljim.

- Neovisan (eng. *independent*)

Prodajom normi omogućava se financiranje razvoja i održavanja u neutralnom okruženju.

- Proaktivan pristup (eng. *can-do*)

ISO norme su usmjerene na prepoznavanje najboljih rješenja za globalne izazove kroz proaktivan pristup.

- Globalan (eng. *global*)

Članovi ISO iz svih regija svijeta imenuju stručnjake i glasaju o nacrtima normi [20].

Mnoge tvrtke koriste ISO 9001 kako bi osigurale da je njihov sustav osiguranja kvalitete uspostavljen i učinkovit. Smjernice ISO 9001 uključuju snažnu usredotočenost na kupca, prakse vrhunskog upravljanja i kako ići ukorak sa stalnim unapređenjima. Također sadrži praktične informacije za donošenje odluka temeljenih na riziku [20].

Cilj razvoja normi je osiguravanje globalnog mira, iskorijenjivanje siromaštva te zaštita okoliša. Provedba plana zahtjeva angažman različitih društvenih grupa, što uključuje vladine i

nevladine organizacije, privatne sektore i pojedince. Uspjeh ovisi o suradnjama i inovacijama [20].

ISO je razvio preko 22.000 međunarodnih normi koje služe kao globalno prihvaćene smjernice. Smjernice olakšavaju suradnju i inovaciju te su temelj za ostvarenje navedenih ciljeva uz podržavanje ključnih aspekata od tehničkih do organizacijskih. Svaki od postavljenih ciljeva podržan je odgovarajućim ISO normama. Ciljevi se ostvaruju putem konkretnih tehničkih i organizacijskih rješenja [20].

4.2.1. ISO/IEC 27001

ISO/IEC 27001 je međunarodna norma za sustave upravljanja sigurnošću informacija. Pokrenuta je 2005. godine, a postaje sve popularnija svake godine povećavanjem zabrinutosti i publiciranja o kršenjima kibernetičke sigurnosti. Zadnje ažuriranje norme bilo je u listopadu 2022 godine. Nova norma ISO27001:2022 uključuje manje izmjene sustava upravljanja i usklađivanja s novim kontrolama ISO27002 unutar dodatka [21].

ISO/IEC 27001 pruža sveobuhvatan pristup upravljanju informacijskom sigurnošću, fokusirajući se ne samo na tehničke aspekte, već i na upravljanje ljudskim resursima i procesima [22].

Norma ISO/IEC 27001 se sastoji od dva dijela:

1. Sustav za upravljanje sigurnošću informacija
2. Skup kontrola koje se koriste za smanjenje rizika [21].

ISO/IEC 27001 je važna norma jer organizacijama omogućava uspostavljanje, implementaciju, održavanje i kontinuirano unapređenje vlastitih sustava upravljanja informacijskom sigurnošću (eng. *Information Security Management System* - ISMS). U nastavku, navedeni su razlozi koji ističu njegovu važnost:

1. Certifikacija i povjerenje:

Kada se organizacija odluči za certificiranje prema normi ISO/IEC 27001, time dokazuje svoju posvećenost sigurnosti informacija. Posvećenost sigurnosti informacija rezultira i povećanim povjerenjem klijenata i poslovnih partnera koji cijene trud i rad organizacije [23].

2. **Međunarodno priznanje:**

Norma ISO/IEC 27001, kao međunarodno priznata norma, olakšava poslovanje na međunarodnoj razini. Prednost leži u tome što je prepoznatljiva kao standard za upravljanje sigurnošću informacija [23].

3. **Pravna usklađenost:**

Također, norma podržava i usklađivanje s raznim zakonima i regulativama koje su povezane s informacijskom sigurnošću [23].

4. **Smanjenje troškova:**

Norma ISO/IEC 27001 sprečavanjem sigurnosnih incidenata, doprinosi smanjenju potencijalnih financijskih gubitaka uzrokovanih sigurnosnim propustima [23].

5. **Efikasnost poslovanja:**

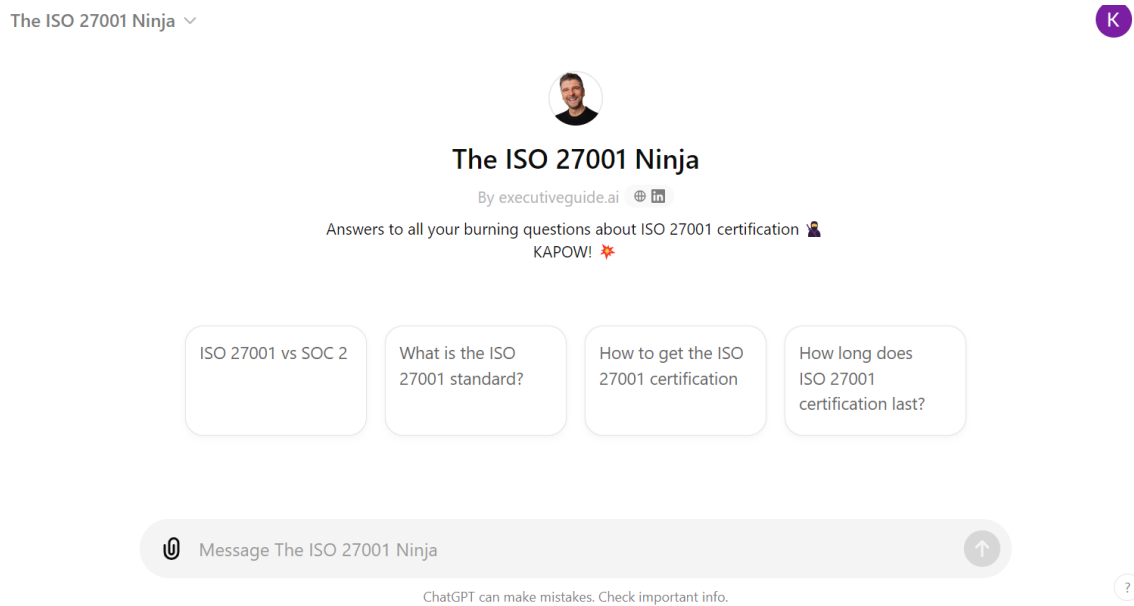
Norma motivira organizacije u definiranju svojih procesa i procedura, čime se povećava kvaliteta rada i smanjuje gubitak vremena [23].

Ova norma pomaže organizacijama da ojačaju svoju zaštitu svojih proizvoda i efikasnije upravljaju informacijama [23].

4.2.1.1. Primjena umjetne inteligencija za usklađivanje s normom ISO/IEC 27001

OpenAI stvorio je UI koji koristi veliki jezični model specifičnih ISO/IEC 27001 resursa na temelju znanja o ISO/IEC 27001, „The ISO 27001 Ninja“. Raspolaže sa znanje do listopada 2023. godine, a trenirao se na velikom broju tekstova i podataka kako bi pomogao ljudima u raznim zadacima. Prva verzija puštena je u javnost u ožujku 2023. godine, a tijekom vremena sposobnost ovog UI-a se unapređuje [23]. Koristi se za automatizaciju zadataka, kao što su procjena rizika, odgovor na incidente i obuku za podizanje svijesti o sigurnosti, koji su potrebni za usklađenost s ISO/IEC 27001. Korištenjem The ISO 27001 Ninje organizacija može brže uočiti prijetnje i pratiti svoje sigurnosne mjere kako bi smanjile rizike. Nadalje, koristi se za razvoj i implementaciju učinkovitijih sigurnosnih kontrola kao i za automatizaciju zadataka odgovora na incidente [23]. Razlika između ChatGPT-a i The ISO 27001 Ninja je u tome što ChatGPT pohranjuje podatke o razgovorima, a The ISO 27001 Ninja ne pohranjuje nikakve razgovore.

Slika 5. prikazuje sučelje web stranica "The ISO 27001 Ninja".



Slika 5. Web sučelje „The ISO 27001 Ninja“ (izvor: vlastita snimka zaslona)

Prednosti upotrebe umjetne inteligencije za normu ISO/IEC 27001 su:

- Brzina i točnost

Automatizacija zadataka organizacijama pomaže da se brže i preciznije usklade sa normom [23].

- Smanjenje troškova

Zamjena ljudskih resursa automatizacijom smanjuje troškove usklađivanja [23].

- Učinkovitost

Poboljšava učinkovitost svojih operacija usklađivanja, oslobađajući resurse za druge zadatke [23].

- Vidljivost

Pružuje bolji uvid u status usklađenosti, omogućavajući identifikaciju i rješavanje nedostataka [23].

S druge strane, izazovi s kojim se organizacija može susresti su:

- Kvaliteta podataka

Kvaliteta podataka na kojima se UI obučava ključna je za njihovu pouzdanost i učinkovitost. Podaci moraju biti točni kako bi umjetna inteligencija mogla učinkovito identificirati nedostatke. Netočnim ili nepotpunim podacima umjetna inteligencija ne može pouzdano identificirati probleme [23].

- Pristranost

Pristranost modela umjetne inteligencije može rezultirati netočnim podacima te je nužna procjena rezultata umjetne inteligencije [23].

- Sigurnost

Umjetna inteligencija zahtijeva temeljitu zaštitu zbog toga što je osjetljiva na sigurnosne rizike [23].

4.2.2. ISO/IEC 25010

ISO/IEC 25010, dio je porodice normi poznata kao SQuaRE (Systems and Software Quality Requirements and Evaluation). Ova serija normi pruža međunarodno priznat okvir za procjenu i specifikaciju kvalitete softverskih sistema i proizvoda [24].

Norma ISO/IEC 25010 nudi detaljne smjernice kako organizacije mogu mjeriti i unaprijediti kvalitetu svojih softverskih rješenja kroz cijeli životni ciklus proizvoda. Glavni cilj ISO/IEC 25010 je omogućiti organizacijama i kupcima softvera efikasno specificiranje i procjenu kvalitete proizvoda. Norma olakšava komunikaciju među dionicima i postavlja jasne kriterije za kvalitete, što pomaže u minimiziranju nesporazuma i povećanju zadovoljstva korisnika [24] [25] [26].

ISO/IEC 25010 klasificira kvalitetu softvera u dvije glavne kategorije:

1. Kvaliteta u upotrebu

Mjeri koliko dobro softver zadovoljava potrebe specifičnih korisnika za ostvarivanje specifičnih ciljeva u specifičnim okruženjima. Uključuje efikasnost, učinkovitost, zadovoljstvo i slobodu od rizika.

2. Kvaliteta proizvoda

Uključuje karakteristike vezane za statička svojstva softvera i dinamička svojstva sustava, poput funkcionalnosti, performansi, kompatibilnosti, upotrebljivosti, pouzdanosti, sigurnosti, održivosti i prenosivosti [25].

Model kvalitete proizvoda sastoji se od devet karakteristika (koje se dalje dijele na potkarakteristike) koje se odnose na svojstva kvalitete proizvoda. Karakteristike i potkarakteristike daju referentni model za kvalitetu proizvoda koju treba specificirati, mjeriti i vrednovati:

1. Funkcionalna prikladnost (eng. *Functional Suitability*)

Ova karakteristika mjeri kako proizvod ili sustav ispunjava izričite i neizričite potrebe korisnika pod specifičnim uvjetima. Sastoji se od tri potkarakteristike:

- Funkcionalna potpunost (eng. *Functional completeness*): mjera u kojoj se provjerava koliko softver zadovoljava sve zahtjeve i očekivanja korisnika koja su definirana tijekom razvoja. Sve funkcije moraju biti ispravno implementirane na temelju prethodno definiranih zahtjeva.
- Funkcionalna ispravnost (eng. *Functional correctness*): karakteristika mjeri koliko proizvod ili sustav pružaju točne rezultate kad se koriste. Provjerava se koliko softver daje ispravne, ali i točne podatke koje moraju biti u skladu s unesenim podacima.
- Funkcionalna prikladnost (eng. *Functional appropriateness*): mjera se odnosi na razinu koliko je softver prilagođen korisnicima. Ova mjera provjerava koliko funkcionalnosti softvera korisnicima pomažu da efikasno obave zadatak uz minimalne poteškoće [24].

2. Učinkovitost izvedbe (eng. *Performance Efficiency*)

Učinkovitost izvedbe ocjenjuje koliko učinkovito proizvod obavlja funkcije unutar zadanog vremenskog okvira i ograničenja resursnih, kao što su CPU, memorije, pohrana i energija, pod specifičnim uvjetima. Ova karakteristika sastoji se od tri potkarakteristike:

- Vremensko ponašanje (eng. *Time behaviour*): vremensko ponašanje je mjera koja se odnosi na to koliko brzo softver može reagirati na korisničke zahtjeve, kao i koliko mu je vremena potrebno da izvrši zadatke. Na konkretnom primjeru, to je vremenski period koji započinje od trenutka kada korisnik pošalje zahtjev do trenutka kada softver odgovori. Cilj je provjeriti koliko brzo softver obrađuje podatke.
- Iskorištenje resursa (eng. *Resource utilization*): ovom mjerom procjenjuje se efikasnost softvera kako bi se provjerilo koliko se koriste dostupni resursi tijekom izvođenja njegovih funkcija. Ti resursi mogu uključivati procesor, memoriju, prostor za pohranu,... Cilj je optimizacija upotrebe tih resursa uz osiguravanje stabilnog i brzog rada softvera bez nepotrebnog dodatnog opterećenja sustava.
- Kapacitet (eng. *Capacity*): mjera u kojoj se procjenjuje koliko proizvod ili sustav može podržati maksimalne zahtjevanje granice parametara kao što su broj korisnika, količine podataka ili broja transakcija koje se mogu obraditi istovremeno. Cilj je osigurati da sustav ostane stabilan u trenutku kad se koristi na vrhuncu svojih mogućnosti [24].

3. Kompatibilnost (eng. *Compatibility*)

Ova karakteristika ocjenjuje sposobnost proizvoda, sustava ili komponente da razmjenjuje informacije s drugim entitetima te obavlja svoje funkcije dok dijeli isto okruženje i resurse. Kompatibilnost sadrži dvije podkarakteristike:

- Koegzistencija (eng. *Co-existence*): mjera u kojoj proizvod može učinkovito funkcionirati u zajedničkom okruženju s drugim proizvodima, bez ometanja njihovih funkcija.
- Interoperabilnost (eng. *Interoperability*): mjera u kojoj se procjenjuje sposobnost sustava ili proizvoda da se poveže s drugim sustavima gdje mogu međusobno razmjenjivati i koristiti informacije, bez dodatnih prilagodba [24].

4. Sposobnost interakcije (eng. *Interaction Capability*)

Ova karakteristika mjeri kako korisnici mogu koristiti proizvod ili sustav za razmjenu informacija i završavanje zadataka preko korisničkog sučelja. Uključuje sljedeće podkarakteristike:

- Prepoznatljivost prikladnosti (eng. *Appropriateness recognizability*): podkarakteristika u kojoj korisnici ocjenjuju da li je neki proizvod prikladan za njihove potrebe ili nije.
- Mogućnost učenja (eng. *Learnability*): podkarakteristika koja procjenjuje koliko jednostavno i brzo korisnici mogu naučiti kako koristiti proizvod.
- Operativnost (eng. *Operability*): mjera u kojoj se procjenjuje koliko su dizajn i funkcionalnosti proizvoda optimizirani za jednostavno rukovanje.
- Zaštita korisnika od pogreške (eng. *User error protection*): podkarakteristika kojom se mjeri koliko proizvod pomaže korisnicima da izbjegnu greške. Kako bi se spriječile greške softver može pužiti upozorenje ili obavijestiti korisnika.
- Angažman korisnika (eng. *User engagement*): mjera pručava koliko korisničko sučelje privlači i motivira korisnike. Estetski ugodan softver zadržat će duže pažnju korisnika što će ih dodatno potaknuti na daljnju upotrebu.
- Inkluzivnost (eng. *Inclusivity*): mjera u kojoj se provjerava koliko različitih korisnika mogu koristiti proizvod. Ova mjera uključuje osobe s invaliditetom te njegovu sposobnost da zadovolji širok raspon korisničkih potreba.
- Pomoć korisnicima (eng. *User assistance*): podkarakteristika procjenjuje mogućnosti proizvoda, može li pomoći korisnicima prilikom ostvarenja ciljeva.
- Samoopisnost (eng. *Self-descriptiveness*): podkarakteristika gdje se mjeri koliko korisnik može sam intuitivno razumijeti koristiti softver bez potrebe za detaljnim uputama [24].

5. Pouzdanost (eng. *Reliability*)

Ova karakteristika ocjenjuje kako sustav, proizvod ili komponenta izvršava funkcije pod određenim uvjetima i tijekom određenog vremena. Uključuje sljedeće podkarakteristike:

- Besprijekornost (eng. *Faultlessness*): podkarakteristika mjeri koliko dobro funkcije rade bez grešaka za vrijeme normalne uporabe. Odnosno, mjeri se koliko sustav ispravno obavlja svoje zadatke bez provođenja neočekivanih ponašanja.

- Dostupnost (eng. *Availability*): mjera u kojoj je sustav ili proizvod operativan i spreman za upotrebu kad je potreban. Visoka dostupnost ključna je za kritične sustavu gdje je stalni pristup neophodan,
- Tolerancija na greške (eng. *Fault tolerance*): podkarakteristika se odnosi na sposobnost u kojoj proizvod nastavlja funkcionirati unatoč prisutnosti grešaka ili problema.
- Oporavivost (eng. *Recoverability*): ova podkarakteristika mjeri sposobnosti sustava da obnovi podatke i vrati se u željeno stanje nakon prekida ili kvara. Oporavivost uključuje procese sigurnosnog kopiranja, povratak podataka. Također, uključuje i brzinu kojom se sustav može vratiti u normalno stanje nakon što se dogodi problem [24].

6. Sigurnost (eng. *Security*)

Ova karakteristika mjeri kako se proizvod ili sustav štiti od zlonamjernih napada i osigurava sigurnost podataka u skladu s razinama ovlaštenja korisnika. Uključuje sljedeće podkarakteristike:

- Povjerljivost (eng. *Confidentiality*): podkarakteristika povjerljivost osigurava da pristup podacima i informacijama imaju samo ovlašteni korisnici. Sprečava se pristup ili mijenjanje osjetljivih podataka neovlaštenim osoba.
- Integritet (eng. *Integrity*): mjera u kojoj su podaci i stanje sustava zaštićeni od neautoriziranih izmjena ili brisanja. Sustav mora osigurati da svi podaci ostanu točni. Mijenjati ih mogu samo ovlašteni korisnici.
- Neporicanje (eng. *Non-repudiation*): sposobnost dokazivanja da su određeni događaji ili akcije zaista izvršeni, sprečavajući njihovo kasnije osporavanje.
- Odgovornost (eng. *Accountability*): mjera u kojoj se aktivnosti mogu nedvosmisleno povezati s pojedincima ili entitetima. Unutar sustavu prati se odgovornost korisnika za sve radnje, a posebno za sigurnosne incidente ili nepravilnosti.
- Autentičnost (eng. *Authenticity*): mjera u kojoj identitet osobe ili sustava može biti potvrđen kao legitiman.
- Otpornost (eng. *Resistance*): Sposobnost sustava da nastavi s radom čak i kada je izložen zlonamjernim napadima. Kako bi se sustav suprotstavio zlonamjernim

napadima, sustav treba imati zaštitne mjere koje će omogućiti sustavu rad bez značajnih prekida ili gubitaka podataka [24].

7. Mogućnost održavanja (eng. *Maintainability*):

Ova karakteristika ocjenjuje koliko efikasno i učinkovito proizvod ili sustav može biti modificiran radi unapređenja, ispravka ili prilagodbe promjenama u okruženju i zahtjevima. Ova karakteristika sadrži sljedeće podkarakteristike (5):

- Modularnost (eng. *Modularity*): mjera u kojoj je sustav podjeljen na zasebne komponente što osigurava da promjene u jednoj komponenti minimalno utječu na ostale.
- Ponovno korištenje (eng. *Reusability*): sposobnost korištenja dijelova proizvoda u različitim sustavima ili prilikom izrade novih proizvoda.
- Mogućnost analize (eng. *Analysability*): efikasnost s kojom se može procijeniti utjecaj promjena na proizvod, dijagnosticirati nedostaci ili identificirati dijelovi za popravak.
- Modifikabilnost (eng. *Modifiability*): mjera u kojoj se proizvod može mijenjati bez unošenja novih grešaka ili smanjenja kvalitete.
- Mogućnost testiranja (eng. *Testability*): efikasnost i učinkovitost uspostavljanja testnih kriterija i provođenja testova za provjeru ispunjava li sustav te kriterije [24].

8. Fleksibilnost (eng. *Flexibility*)

Ova karakteristika ocjenjuje sposobnost proizvoda da se prilagodi promjenama u zahtjevima, kontekstu upotrebe ili okruženju. Uključuje sljedeće podkarakteristike:

- Prilagodljivost (eng. *Adaptability*): mjera u kojoj proizvod može biti prilagođen ili prenesen na različite hardverske, softverske ili korisničke okoline.
- Skalabilnost (eng. *Scalability*): sposobnost proizvoda da se prilagodi rastućim ili smanjujućim opterećenjima i varijabilnostima kapaciteta.
- Instalabilnost (eng. *Installability*): efikasnost i učinkovitost s kojom se proizvod može instalirati ili deinstalirati u specifičnom okruženju.

- Zamjenjivost (eng. *Replaceability*): mjera u kojoj proizvod može zamijeniti drugi softverski proizvod za istu namjenu u istom okruženju [24].

9. Sigurnost (eng. *Safety*)

Ova karakteristika mjeri sposobnost proizvoda da pod specifičnim uvjetima izbjegava situacije koje bi mogle ugroziti ljudski život, zdravlje, imovinu ili okoliš. Sastoji se od sljedećih podkarakteristika:

- Operativno ograničenje (eng. *Operational constraint*): mjera u kojoj proizvod ili sustav može ograničiti svoje funkcije unutar sigurnosnih granica kada se suoči s potencijalnom opasnošću [24].
- Identifikacija rizika (eng. *Risk identification*): sposobnost proizvoda da prepozna situacije ili operacije koje bi mogle predstavljati ozbiljan rizik.
- Sigurnost pri kvaru (eng. *Fail safe*): mjera u kojoj se proizvod može automatski prebaciti u siguran režim rada ili se vratiti u sigurno stanje u slučaju kvara.
- Upozorenje o opasnosti (eng. *Hazard warning*): sposobnost proizvoda ili sustava da izdaje upozorenja o potencijalnim rizicima, omogućujući pravovremenu reakciju za održavanje sigurnosti.
- Sigurna integracija (eng. *Safe integration*): podkarakteristika koja se odnosi na sposobnost sustavu ili proizvoda da zadrži sigurnost tijekom i nakon integracije s drugim komponentama. Cilj je osigurati da se komponente međusobno spajaju ili surađuju, a da ne dođe do smanjenja sigurnosnih standarda ili povećanja rizika [24].

Norma ISO/IEC 25010 se odnosi na ICT proizvod koji je dio informacijskog sustava. Komponente ICT proizvoda uključuju podsustave, softver, firmware, hardver, podatke, komunikacijsku infrastrukturu i druge elemente koji su dio ICT proizvoda. Ovaj model se može koristiti za specifikaciju zahtjeva i procjenu kvalitete ciljnih proizvoda tijekom njihovog životnog ciklusa od strane nekoliko dionika, uključujući programere, naručitelje, osoblje za osiguranje i kontrolu kvalitete [26].

Model kvalitete ISO/IEC 25010 može unaprijediti mnoge faze životnog ciklusa proizvoda, uključujući otkrivanje i definiranje zahtjeva, potvrđivanje njihove sveobuhvatnosti, postavljanje ciljeva za dizajn i testiranje te osiguravanje kvalitete. Ovaj model također pomaže u

postavljanju kriterija prihvatljivosti i mjera za evaluaciju karakteristika kvalitete, sve u cilju unapređenja kvalitete proizvoda [26].

4.2.3. ISO/IEC/IEEE 29119

ISO/IEC/IEEE 29119 je međunarodna norma za testiranje softvera koji pokriva različite aspekte kako bi se osigurala kvaliteta i učinkovitost procesa testiranja softvera u organizacijama.

ISO/IEC/IEEE 29119 pruža skup normi za testiranje softvera koji je međunarodno prihvaćen. Pomaže organizacijama u strukturiranju njihovih praksi testiranja softvera na način koji se usklađuje s globalnim najboljim praksama [27].

Ključne komponente norme ISO/IEC/IEEE 29119 su:

1. Koncepti i definicije (ISO/IEC/IEEE 29119-1)
 - Ovaj dio norme postavlja temeljnu terminologiju i definicije koje se koriste kroz cijelu normu. Poznavanjem terminologije uspostavlja se zajednički jezik za testere softvera što olakšava komunikaciju unutar organizacije [27].
2. Proces testiranja (ISO/IEC/IEEE 29119-2)
 - Proces testiranja opisuje generički model procesa za testiranje koji se može prilagoditi i primijeniti na različite vrste testiranja softvera unutar bilo koje organizacije. Detaljno opisuje faze uključene u proces testiranja, uključujući planiranje, dizajn, izvođenje i zatvaranje testnih ciklusa. Osigurava da svaki aspekt testiranja, od početnog plana do konačnog izvješća, bude pokriven [27].
3. Dokumentacija testiranja (ISO/IEC/IEEE 29119-3)
 - Dokumentacija testiranja određuje predloške i smjernice za izradu dokumentacije koja je potrebna tijekom procesa testiranja. Uključuje planove testiranja, testne primjere i izvješća o testiranju, osiguravajući konzistentnost i potpunost dokumentacije. Cilj je osigurati da svi aspekti testiranja budu pravilno zabilježeni [27].
4. Tehnike testiranja (ISO/IEC/IEEE 29119-4)

- Ovaj dio norme pokriva tehničke aspekte testiranja softvera. Detaljno opisuje različite tehnike testiranja koje se mogu koristiti za identifikaciju nedostataka ili grešaka u softveru. Uključuje metode poput testiranja s crnom kutijom, testiranja s bijelom kutijom i iskustvenog testiranja [27].

5. Testiranje vođeno ključnim riječima (ISO/IEC/IEEE 29119-5)

- Fokusira se na metodu gdje su testovi opisani ključnim riječima, što pojednostavljuje proces testiranja i čini ga dostupnijim ne-tehničkim dionicima [27].

Primjena i koristi ISO/IEC/IECC 29119:

- Norma pruža čvrst okvir koji organizacije mogu usvojiti kako bi unaprijedila učinkovitost svojih procesa testiranja softvera.
- Implementacijom norme ISO/IEC/IEE 29119, organizacije mogu osigurati da su njihovi procesi testiranja temeljiti, ponovljivi i pouzdani što vodi do softvera bolje kvalitete.
- Standard također olakšava bolju komunikaciju i suradnju unutar timova i između timova standardiziranjem termina i procesa korištenih u testiranju [27].

4.3. Umjetna inteligencija u osiguravanju kvalitete u softverskom inženjerstvu

U nastavku su prikazane ključne prednosti i izazovi kod primjene umjetne inteligencije u unapređenju kvalitete softvera koristeći: automatizaciju testiranja, predviđanja grešaka, detekciju anomalija i sigurnosnih propusta.

Automatizacija testiranja

Automatizacija testiranja je jedan od najznačajnijih doprinosa UI u osiguravanju kvalitete (skraćeno QA). Tradicionalne metode zahtijevaju ručno pisanje i izvršavanje testnih slučajeva, što je često dugotrajno i sklono ljudskim pogreškama. UI omogućuje automatizaciju ovih procesa koristeći alate poput Seleniuma i Appiuma, koji se temelje na UI tehnikama za generiranje i izvršavanje testnih slučajeva. Korištenjem strojnog učenja, ovi alati mogu identificirati najkritičnije dijelove koda i automatski ih testirati [18].

Predviđanje grešaka

Prediktivna analiza postaje ključna komponenta u održavanju softverskih sustava. Algoritmi strojnog učenja, obučeni na povijesnim podacima o kvarovima, mogu predvidjeti buduće probleme s visokim stupnjem točnosti. Ovo omogućava proaktivno održavanje i smanjuje vrijeme zastoja, što je posebno važno u sustavima gdje su visoka dostupnost i pouzdanost ključni, poput financijskih aplikacija ili sustava za kontrolu zračnog prometa [18].

Detekcija anomalija i sigurnosnih propusta

Detekcija anomalija: Jedna od ključnih primjena UI u QA je detekcija anomalija. Složenost modernih softverskih sustava i količina podataka koje obrađuju čine ručnu identifikaciju anomalija neodrživom. UI sustavi, poput onih koji koriste duboko učenje, mogu analizirati velike količine podataka u stvarnom vremenu i prepoznati neuobičajene uzorke koji mogu ukazivati na greške ili sigurnosne prijetnje. Primjerice, sustavi za nadzor mrežne sigurnosti koriste UI za otkrivanje neuobičajenih uzoraka mrežnog prometa koji bi mogli ukazivati na napad [18].

Sigurnosni propusti: Sigurnost je ključno pitanje u osiguravanju kvalitete softvera. Tradicionalne metode provjere sigurnosti često se oslanjaju na statičke analize koda i ručne revizije, što može biti nedovoljno za složene sustave. UI može automatski analizirati kod i prepoznati ranjivosti koje bi mogle biti iskorištene za napade. Korištenjem modela koji se temelje na velikim bazama podataka o poznatim napadima, UI sustavi mogu pružiti brze i precizne procjene sigurnosnih propusta [18].

Primjena UI u razvoju i održavanju softvera

Razvoj kvalitetnijeg koda: UI može pomoći inženjerima u pisanju kvalitetnijeg koda od samog početka. Alati koji koriste prirodne jezike za analizu koda mogu automatski otkriti nedosljednosti, složenost i druge potencijalne probleme u kodu prije nego što postanu ozbiljni. Na primjer, GitHub Copilot koristi UI za predlaganje dijelova koda tijekom pisanja, čime pomaže programerima da izbjegnu uobičajene greške i pišu efikasniji kod [18].

Održavanje softverskih sustava: Održavanje softvera često predstavlja velik izazov, osobito u velikim sustavima s dugim životnim ciklusima. Korištenjem UI, proces održavanja može biti značajno unapređen. Na primjer, algoritmi strojnog učenja mogu analizirati povijesne

promjene u kodu i identificirati obrasce koji su doveli do problema u prošlosti. Time pomažu u donošenju odluka o potrebnim promjenama i unapređenjima [18].

Izazovi i etička pitanja

Izazovi u implementaciji UI: Unatoč brojnim prednostima, implementacija UI u osiguravanju kvalitete nije bez izazova. Razvoj pouzdanih UI modela zahtijeva velike količine kvalitetnih podataka, što često predstavlja problem. Nadalje, UI sustavi mogu biti skloni pristranostima koje proizlaze iz podataka na kojima su obučeni, što može dovesti do nepoželjnih rezultata. Potrebno je razviti metode za otkrivanje i ispravljanje tih pristranosti kako bi se osigurala pravednost i točnost UI sustava [18].

Etička pitanja: Primjena UI u softverskom inženjerstvu također otvara niz etičkih pitanja. Jedno od ključnih pitanja je transparentnost UI sustava. Budući da mnogi UI modeli, poput dubokih neuronskih mreža, djeluju kao "crne kutije" (engl. *Black Box*), teško je razumjeti kako donose odluke. Ovo može biti problematično u QA, gdje je važno razumjeti zašto je sustav identificirao određenu komponentu softvera kao problematičnu [18].

Sigurnost i privatnost: Sigurnost i privatnost podataka također su ključni etički izazovi. Kako UI sustavi sve više analiziraju osjetljive podatke, postavlja se pitanje kako osigurati da ti podaci budu adekvatno zaštićeni. Osiguravanje privatnosti i zaštite podataka korisnika mora biti prioritet prilikom implementacije UI u QA procese [18].

Budućnost UI u osiguravanju kvalitete

Integracija s DevOps-om: Budućnost QA u softverskom inženjerstvu usko je povezana s integracijom UI s DevOps praksama. DevOps, koji kombinira razvoj i operacije, zahtijeva brzu i kontinuiranu isporuku softvera. UI može igrati ključnu ulogu u automatizaciji i optimizaciji procesa testiranja unutar DevOps-a, omogućujući brže izdavanje kvalitetnog softvera.

Razvoj naprednih alata: Očekuje se da će razvoj naprednih UI alata za QA nastaviti rasti. To uključuje razvoj alata koji koriste napredne tehnike strojnog učenja za analizu i optimizaciju cijelog procesa razvoja softvera, od dizajna do implementacije i održavanja.

Suradnja čovjeka i stroja: U budućnosti, naglasak će biti na suradnji između ljudi i UI sustava. Dok će UI preuzimati repetitivne i složene zadatke, ljudski inženjeri će se moći

usredotočiti na kreativne i strateške aspekte razvoja softvera. Ova sinergija može dovesti do značajnog unapređenja kvalitete softvera [18].

Korištenje umjetne inteligencije u osiguravanju kvalitete softvera predstavlja transformacijski korak u softverskom inženjerstvu. Unatoč izazovima, prednosti koje UI donosi – od automatizacije testiranja do predviđanja grešaka i detekcije anomalija – omogućuju učinkovitiji, brži i pouzdaniji proces razvoja softvera. Kako tehnologija napreduje, očekuje se da će UI igrati sve značajniju ulogu u QA, čineći softverske sustave sigurnijima, pouzdanijima i prilagodljivijima budućim izazovima [16].

5. Istraživanje i primjena umjetne inteligencije u procesu razvoja programskog proizvoda

U ovom dijelu diplomskog rada prikazano je istraživanje načina i učinkovitosti primjene umjetne inteligencije (UI) u procesu razvoja programskog proizvoda s posebnim naglaskom na standardizirane prakse za osiguravanje kvalitete i sigurnosti koda prema normama. Glavni fokus je ispitivanje mogućnosti alata umjetne inteligencije u vidu usklađivanja razvoja programskog proizvoda s odabranim normama. Cilj je uz pomoć umjetne inteligencije razviti programski kod i popratnu dokumentaciju koji zadovoljavaju visoke standarde normi i osiguravaju funkcionalnost koda. Naglasak je na fazama provjere i validacije koda uz primjenu ISO/IEC normi; usporedbi učinkovitosti i sveobuhvatnosti testiranja UI-a u odnosu na tradicionalnu metodu ručnog pisanja testova.

5.1. Metode i tehnike rada

Za cilj rad ima istražiti mogućnosti umjetne inteligencije u unapređenju kvalitete softverskog koda primjenom tri odabrane ISO norme. Za provođenje analize koristi se kombinacija metoda, kvantitativnih i kvalitativnih metoda primjene umjetne inteligencije u osiguravanju kvalitete. Za praktičnu primjenu provedbe rada, korišten je stvarni primjer, programski kod tvrtke 3MI lab nad kojim će se provesti analiza umjetne inteligencije. Za provedbu slučajeva upotrebe korišten je ChatGPT-4, koji je generirao kod, analizirao kvalitetu te davao smjernice i preporuke za unapređenjem u skladu s ISO normama. Rad se usredotočuje na usklađivanje procesa razvoja programskog koda s relevantnim međunarodnim normama. Kako bi se prikazao način na koji umjetna inteligencija može pomoći u postizanju usklađenosti s normama korišteni su slučajevi upotrebe.

Slika 6. prikazuje proces rada u četiri koraka koji su provedeni za analizu i primjenu umjetne inteligencije, ChatGPT-a u kontekstu osiguravanja kvalitete programskog koda prema ISO normama. Prvi korak je odabir relevantnih normi koje će se koristiti u analizu kvalitete programskog koda i/ili jedinica testa. U drugom koraku, ChatGPT-u postavljaju se pitanja naglašavajući da su odgovori potrebni za diplomski rad. Pitanja se odnose na kvalitetu programskog koda, jedinice testa i norme, s ciljem dobivanja korisnih informacija. Sljedeći korak uključuje prikupljanje odgovora koje generira ChatGPT kako bi se utvrdila njegova primjenjivost u kontekstu normi i kvalitete koda. U posljednjem koraku fokus je na analizi i

primjeni dobivenih odgovora na stvarnom projektu tvrtke 3MI lab. Ovom analizom provjerava se može li i u kojoj mjeri ChatGPT zadovoljiti potrebne norme u praksi.



Slika 6. Proces primjene ChatGPT-a za analizu kvalitete programskog koda prema normama (izvor: vlastita izrada u alatu Canva)

Kako bi se osigurala usklađenost s međunarodnim normativnim dokumentima, analizirani procesi, inputi i outputi su usklađeni sa sljedećim relevantnim normama koji se mogu primijeniti:

- ISO/IEC 27001 – Informacijska tehnologija – Sigurnosne tehnike – Sustavi upravljanja sigurnošću informacija – Zahtjevi (eng. *Information technology – Security techniques – Information security management systems – Requirements*)
- ISO/IEC 25010 - Inženjering sustava i softvera – Zahtjevi i evaluacija kvalitete sustava i softvera (SQuaRE) – Modeli kvalitete sustava i softvera (eng. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*)

- ISO/IEC 29119 - Inženjering softvera i sustava – Testiranje softvera (eng. *Software and systems engineering — Software testing*)

Praktični dio rada je opisan u 3 poglavlja. Svako poglavlje se fokusira na specifičnu ISO normu i uključuje nekoliko ključnih slučajeva upotrebe (engl. *Use Cases*). Svaki slučaj upotrebe tretira se kao zaseban zadatak. Na taj način osigurana je preciznost u analizi i razmatranju specifičnih zadataka. S pomoću slučajeva upotrebe predstavljeni su pojedinačni scenariji u kojima se ispituje na koji način UI može „pomoći“ u kontekstu usklađivanja programskog koda s odabranom normom.

Danas alati umjetne inteligencije postaju standardna praksa među mnogim programerima kako bi se povećala produktivnost i pronašla najbolja rješenja za razvijanje optimalnog programskog koda. Jedan od najpopularnijih alata je OpenAI Codex koji prevodi prirodni jezik u kod kojeg pokreće GitHub Copilot. Iako se OpenAI Codex smatra najboljim generatorom AI kodova u 2024 godine, u diplomskom radu korišten je ChatGPT-4 koji je odabran zbog svoje sveobuhvatnosti. Iako se ne smatra kao najbolji u pisanju programskog koda, ovaj alat ima široku primjenu koja uključuje razumijevanje i primjenu normi te vrednovanje kvalitete koda. Nadalje, alat pruža analize, savjete i smjernice koje su ključne za osiguravanje visoke razine kvalitete i pouzdanosti programskog proizvoda što ga čini idealnim izborom [28].

Za potrebe diplomskog rada koristi se postojeća dokumentacija i dio programskog koda tvrtke 3MI lab d.o.o. projekta XY. U projektu XY administratori sustava generiraju kartice koje su grupirane u dekove (engl. *deck-ove*) s pitanjima za policajce. Policijski službenici ulaze u informacijski sustav gdje odgovaraju na pitanja, a s protekom vremena odgovori i pitanja postavljena svakom pojedinom policijskom službeniku se brišu (nema trajnog zapisa o individualnim ispitima). Zbog promjene zakona u Sjedinjenim Američkim Državama, prema kojem se ispiti moraju bilježiti (arhivirati) u sustavu, stvara se zadatak za programera da omogući takvu funkcionalnost sustava.

- **Softversko rješenje** se odnosi na razvoj i testiranje funkcionalnosti arhiviranja dekova unutar aplikacije za upravljanje karticama.
- **Poslovna potreba** je omogućiti administratoru jednostavnu identifikaciju i odabir dekova za koje postoji potreba za arhiviranjem.

- **Poslovni zahtjevi** su sljedeći:

1. Kreirati novu funkciju arhiviranje deka (engl. *Archive Deck*)

Novu funkcionalnost Arhiviranje deka potrebno je implementirati u postojećem izborniku. Odabirom funkcije prikazuje se skočni prozor za potvrdu arhiviranja kako bi administrator bio svjestan rezultata akcije arhiviranja. Nakon potvrde administratora, dek se treba premjestiti u arhivirano stanje.

2. Arhiviranjem deka treba se arhivirati sve kartice unutar deka

U trenutku kada se dek premjesti u arhivirano stanje arhiviraju se sve kartice deka. Arhivirani dek dobiva status Arhiviran. Status deka prikazuje se korisniku na ekranu pored imena deka. Sustav vraća povratnu informaciju da je dek uspješno arhiviran.

3. Arhiviranje treba biti zabilježeno u sustavu

Pri uspješnom arhiviranju deka sustav bilježi informacije o izvršenoj radnji. Sustav zapisuje ime administratora koji je arhivirao dek, datum i vrijeme arhiviranja i novi status deka.

4. Prilikom kreiranja novog deka ne prikazuje se sadržaj arhiviranog deka

Sustav automatski kreira nove kartice za policijske službenike. Dek s novim karticama ne prikazuje sadržaj iz arhiviranog deka. Arhivirani dek više ne smije biti dostupan prilikom kreiranja novog deka.

5. Kreirat novu funkciju dearhiviranje pojedinačnih kartica iz arhiviranog deka

Nakon kreiranja funkcije arhiviranje deka, potrebno je korisniku omogućiti i akciju dearhiviranje pojedinačnih kartica iz deka. Prilikom dearhiviranja kartica sustav prikazuje skočni prozor s popisom arhiviranih kartica kako bi korisnik odabrao karticu koju želi dearhivirati. Korisnik zatim može odabrati u koji aktivan dek želi premjestiti odabranu karticu.

U nastavku slijedi pregled programskog kod koji čini temeljnu strukturu za provedbu ovog istraživanja. Kod je napisan u programskom jeziku C# te su korištene različite tehnologije i alati, uključujući:

- .NET Framework / .NET Core - osnovni razvojni okvir aplikacije
- Entity Framework Core - ORM (*Object-Relational Mapping*) alat za rad s bazom podataka
- NUnit - framework za pisanje i izvođenje jediničnih testova
- Moq - knjižnica za mockiranje objekata u testovima
- TechTalk.SpecFlow - alat za behavior-driven development (BDD) i pisanje testova u prirodnom jeziku

Za primjer praktične primjene teorijskih koncepata korišten je kod koji implementira rješenja na zadane poslovne zahtjeve, ilustrirajući način na koji se te ideje provode u praksi. U nastavku slijedi pregled programskog koda.

Programski kod za arhiviranje dekova (u nastavku rada: PK1)

```
using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Recall.Shared.MsSql.Context;

using Recall.DeckManagement.Database.Entities;

using Recall.DeckManagement.Repositories.EntityMappers;

using
Recall.DeckManagement.Services.ApplicationServices.Dependencies.Repositories;

using Recall.DeckManagement.Services.Domain.Model.Entities;

using Recall.DeckManagement.Core.Enums;

using Recall.Shared.Framework.Security.Identity;

using Recall.Shared.Framework.Services.ValueObjects;

using System.Linq;

namespace Recall.DeckManagement.Repositories.Implementations
{
public class DeckArchiverRepository : IDeckArchiverRepository
{
private readonly RecallDbContext dbContext;
```



```

private readonly IUserContext userContext;

private readonly CardLifeCycleMapper cardLifeCycleMapper;

private DbSet<DeckEntity> DeckEntityDbSet => dbContext.Set<DeckEntity>();

private DbSet<CardEntity> CardEntityDbSet => dbContext.Set<CardEntity>();

public DeckArchiverRepository(
    RecallDbContext dbContext,
    IUserContext userContext,
    CardLifeCycleMapper cardLifeCycleMapper)
{
    this.dbContext = dbContext;
    this.userContext = userContext;
    this.cardLifeCycleMapper = cardLifeCycleMapper;
}

public async Task<DeckArchiver?> GetById(IntegerId id)
{
    var unarchivedDeckCards = await CardEntityDbSet
        .Include(x => x.CardResources)
        .Where(x => x.DeckId == id.Value && x.CardStatusType !=
            CardStatusType.Archived).ToListAsync();

    var deck = await DeckEntityDbSet.DefaultIfEmpty().SingleOrDefaultAsync(x =>
        x.Id == id);

    var deckWtihUnarchivedCards = new DeckArchiver(deck.Id,
        deck.DeckStatusType, deck.ArchiveOnDate, unarchivedDeckCards.Select(x =>
            cardLifeCycleMapper.ConvertToCardLifeCycle(x)).ToList());

    return deckWtihUnarchivedCards;
}

public async Task<DeckArchiver> UpdateDeckArvhiver(DeckArchiver deck)
{
    var deckEntity = await
        DeckEntityDbSet.FindAsync((EncryptedInt)deck.EntityId,
            (EncryptedInt)userContext.TenantId);
}

```

```

deckEntity.DeckStatusType = deck.DeckStatusType;

deckEntity.ArchiveOnDate = deck.ArchiveOnDate;

// archived on date

foreach(var cardLifeCycle in deck.CardLifeCycles)

{

var cardEntity = await
CardEntityDbSet.FindAsync((EncryptedInt)cardLifeCycle.EntityId,
(EncryptedInt)userContext.TenantId);

cardLifeCycleMapper.ConvertToCardEntity(cardLifeCycle, cardEntity);

}

dbContext.SaveChanges();

return deck;

}

}

}

using System;

using System.Threading.Tasks;

using Recall.DeckManagement.Services.ApplicationServices.Commands.Deck;

using
Recall.DeckManagement.Services.ApplicationServices.Dependencies.Repositories;

using Recall.DeckManagement.Services.Domain.Model.Validators;

using Recall.Shared.Framework.Utills.Time;

namespace
Recall.DeckManagement.Services.ApplicationServices.Implementation;

public class DeckArchiverService : IDeckArchiverService

{

private readonly IDeckArchiverRepository deckArchiverRepository;

private readonly IValidatorFactory validatorFactory;

```

```

private readonly ISystemClock systemClock;

public DeckArchiverService(
    IDeckArchiverRepository deckArchiverRepository,
    IValidatorFactory validatorFactory,
    ISystemClock systemClock)
{
    this.deckArchiverRepository = deckArchiverRepository;
    this.validatorFactory = validatorFactory;
    this.systemClock = systemClock;
}

public async Task<ArchiveDeckCommandResult>
ArchiveDeck(ArchiveDeckCommand command)
{
    await
validatorFactory.Validate(command).Using<CanArchiveDeckCommandValidator>();

    var deckArchiver = await
deckArchiverRepository.GetById(command.DeckId);

    deckArchiver.Archive(systemClock.UtcNow);

    await deckArchiverRepository.UpdateDeckArvhiver(deckArchiver);

    return new ArchiveDeckCommandResult(deckArchiver.EntityId,
deckArchiver.CardLifeCycles.Count, deckArchiver.DeckStatusType);
}
}

using Recall.Shared.Framework.Services.ValueObjects;
using Recall.DeckManagement.Core.Enums;
using System.Collections.Generic;
using System;

```

```

namespace Recall.DeckManagement.Services.Domain.Model.Entities
{
    [AggregateRoot]
    public class DeckArchiver : BaseDomainModel<IntegerId>
    {
        public DeckArchiver(IntegerId entityId) : base(entityId)
        {
        }

        public DeckArchiver(IntegerId entityId, DeckStatusType
deckStatusType, DateTimeOffset? archiveOnDate, List<CardLifeCycle>
cardLifeCycles) : base(entityId)
        {
            DeckStatusType = deckStatusType;
            CardLifeCycles = cardLifeCycles;
            ArchiveOnDate = archiveOnDate;
        }

        public DeckStatusType DeckStatusType { get; set; }
        public DateTimeOffset? ArchiveOnDate { get; set; }
        public List<CardLifeCycle> CardLifeCycles { get; set; }
        public void Archive(DateTimeOffset utcNow)
        {
            DeckStatusType = DeckStatusType.Archived;
            ArchiveOnDate = utcNow;
            foreach (var cardLifeCycle in CardLifeCycles)
            {
                cardLifeCycle.Archive(utcNow);
            }
        }
    }
}

```

```

    }
}

using Recall.DeckManagement.Core.Enums;
using Recall.DeckManagement.Services.ApplicationServices;
using Recall.DeckManagement.Services.ApplicationServices.Commands.Deck;
using
Recall.DeckManagement.Services.ApplicationServices.Dependencies.Repositories;
using Recall.Shared.Framework.Exceptions;
using System.Threading.Tasks;

namespace Recall.DeckManagement.Services.Domain.Model.Validators
{
    public class CanArchiveDeckCommandValidator :
    IObjectValidator<ArchiveDeckCommand>
    {
        private readonly IPermissionValidator permissionValidator;
        private readonly IDeckArchiverRepository deckArchiverRepository;

        public CanArchiveDeckCommandValidator(IPermissionValidator
permissionValidator, IDeckArchiverRepository deckArchiverRepository)
        {
            this.permissionValidator = permissionValidator;
            this.deckArchiverRepository = deckArchiverRepository;
        }
        public async Task Validate(ArchiveDeckCommand command)
        {
            var deck = await
deckArchiverRepository.GetById(command.DeckId);
            if (deck == null)

```

```

        {
            throw CommonException.NotFound();
        }

        if (deck.DeckStatusType != DeckStatusType.Active)
        {
            throw CommonException.DeckNotActive();
        }

        if (!await permissionValidator.CanUpdateDeck(deck.EntityId))
        {
            throw CommonException.NotAuthorized();
        }
    }
}
}

```

Kako bi programer bio siguran da je kreirao ispravan programski kod, programer osim koda kojim je kreirao programsko rješenje piše i jedinične testove (eng. *unit tests*). Ovi testovi su pisani sa svrhom da svaka funkcionalnost radi kako je predviđeno. Spomenute funkcionalnosti uključuju preuzimanje podataka o deku, validaciju podataka, ažuriranje statusa deka te promjenu stanja kartica na arhivirano. Nakon prikupljanja poslovnog zahtjeva i programskog koda, u nastavku, prikazani jedinični testovi koje je programer napisao.

Testovi jedinica za provjeru ispravnosti programskog koda (u nastavku rada: TJ1) – jedinični testovi:

@JiraId:PRC-599

@unit

Scenario: User has no update rights

Given I have a Deck to archive

| DeckId |

| 1 |

And I have an Archive Deck command

```
| DeckId |  
| 1 |
```

And the permission to archive a deck is false

And the deck is active

When I try to archive a deck

Then I should get an exception: 'Not authorized to perform this action.'

@JiraId:PRC-599

@unit

Scenario: Deck does not exist

Given I have an Archive Deck command

```
| DeckId |  
| 1 |
```

When I try to archive a deck

Then I should get an exception: 'Resource not found.'

@JiraId:PRC-599

@unit

Scenario: Deck is already archived

Given I have a Deck to archive

```
| DeckId |  
| 1 |
```

And I have an Archive Deck command

```
| DeckId |  
| 1 |
```

And the permission to archive a deck is true

And the deck is archived

When I try to archive a deck

Then I should get an exception: 'Deck is not active.'

@JiraId:PRC-599

@unit

Scenario: I archive a deck

Given I have a Deck to archive

DeckId
1

And The deck has unarchived cards

Status	CardId	
	1	Draft
	2	Published
	3	PendingReview

And I have an Archive Deck command

DeckId
1

And the permission to archive a deck is true

And the deck is active

When I try to archive a deck

Then I should get ArchiveDeckCommandResult with correct DeckId and status

And all unarchived cards should be archived

```
using Autofac.Extras.Moq;
```

```
using Recall.DeckManagement.Services.Domain.Model.Entities;
```

```
using System;
```

```
using TechTalk.SpecFlow;
```



```

using TechTalk.SpecFlow.Infrastructure;

using Recall.DeckManagement.Tests.Unit.MockExtensions;

using Recall.DeckManagement.Services.ApplicationServices.Implementation;

using System.Threading.Tasks;

using FluentAssertions;

using Recall.DeckManagement.Services.ApplicationServices;

using Recall.DeckManagement.Services.ApplicationServices.Commands.Card;

using
Recall.DeckManagement.Services.ApplicationServices.Dependencies.Repositories;

using TechTalk.SpecFlow.Assist;

using Recall.Shared.Framework.Services.ValueObjects;

using Recall.DeckManagement.Core.Enums;

using Recall.DeckManagement.Services.ApplicationServices.Commands.Deck;

using System.Collections.Generic;

using System.Linq;

using Recall.Shared.Framework.Utils.Time;

namespace Recall.DeckManagement.Tests.Unit.DeckArchiverServiceTests
{
    [Binding]
    [Scope(Feature = "ArchiveDeck")]
    public class ArchiveDeckStepDefinitions
    {
        private Exception? actualException;

        private DeckArchiver? deck;

        private bool canUpdateDeck;

        private int numOfUnarchivedCards;

        private ArchiveDeckCommand command = null!;
    }
}

```

```

private ArchiveDeckCommandResult? commandResult;

private readonly AutoMock mock;

private readonly ISpecFlowOutputHelper outputHelper;

    public ArchiveDeckStepDefinitions(AutoMock mock,
ISpecFlowOutputHelper outputHelper)
    {
        this.mock = mock;

        this.outputHelper = outputHelper;
    }

private void MockDependencies()
{
    mock.Mock<IDeckArchiverRepository>()
        .MockGetById(deck)
        .MockUpdateDeckArchiver(deck);

    mock.Mock<IPermissionValidator>()
        .MockCanUpdateDeck(canUpdateDeck);

    mock.Mock<ISystemClock>()
        .MockUtcNow(DateTimeOffset.UtcNow);
}

[Given(@"I have a Deck to archive")]
public void GivenIHaveADeckToArchive(Table table)
{
    var deckId = int.Parse(table.Rows[0]["DeckId"]);
    deck = new DeckArchiver(IntegerId.FromInt(deckId));
}

```

```

}

[Given(@"I have an Archive Deck command")]
public void GivenIHaveAnArchiveDeckCommand(Table table)
{
    var deckId = int.Parse(table.Rows[0]["DeckId"]);
    command = new ArchiveDeckCommand(IntegerId.FromInt(deckId));
}

[Given(@"The deck has unarchived cards")]
public void GivenTheDeckHasUnarchivedCards(Table table)
{
    var cards = new List<CardLifeCycle>();
    foreach(var row in table.Rows)
    {
        var cardId = int.Parse(row["CardId"]);
        var status =
(CardStatusType)Enum.Parse(typeof(CardStatusType), row["Status"]);
        var card = new CardLifeCycle(IntegerId.FromInt(cardId));
        card.CardStatusType = status;
        cards.Add(card);
    }
    deck.CardLifeCycles = cards;
    numOfUnarchivedCards = cards.Count();
}

[Given(@"the permission to archive a deck is (.*)")]
public void GivenThePermissionToArchiveADeckIs(bool permission)
{
    canUpdateDeck = permission;
}

```

```

[Given(@"the deck is active")]
public void GivenTheDeckIsActive()
{
    deck.DeckStatusType = DeckStatusType.Active;
}

[Given(@"the deck is archived")]
public void GivenTheDeckIsArchived()
{
    deck.DeckStatusType = DeckStatusType.Archived;
}

[When(@"I try to archive a deck")]
public async Task WhenITryToArchiveADeck()
{
    MockDependencies();

    var deckArchiverService = mock.Create<DeckArchiverService>();

    try
    {
        commandResult = await
deckArchiverService.ArchiveDeck(command);
    }

    catch (Exception ex)
    {
        actualException = ex;
    }
}

[Then(@"I should get an exception: '(.*)'")]
public void ThenIShouldGetAnException(string message)

```

```

        {
            actualException.Should().NotNull("Exception should be
generated.");

            if (actualException != null)
            {
                actualException.Message.Should().Be(message, $"Exception
should be: {message}");
            }
        }

        [Then(@"I should get ArchiveDeckCommandResult with correct DeckId
and status")]

        public void
ThenIShouldGetArchiveDeckCommandResultWithCorrectDeckIdAndStatus()

        {

            actualException.Should().BeNull("Exception should not be
generated.");

            if (actualException != null)
            {

                outputHelper.WriteLine("Exception: {0}",
actualException.ToString());
            }

            commandResult!.DeckId.Should().Be(deck.EntityId);

            commandResult!.Status.Should().Be(DeckStatusType.Archived);
        }

        [Then(@"all unarchived cards should be archived")]

        public void ThenAllUnarchivedCardsShouldBeArchived()

        {

            commandResult!.NumberOfArchivedCards.Should().Be(numOfUnarchive
dCards);

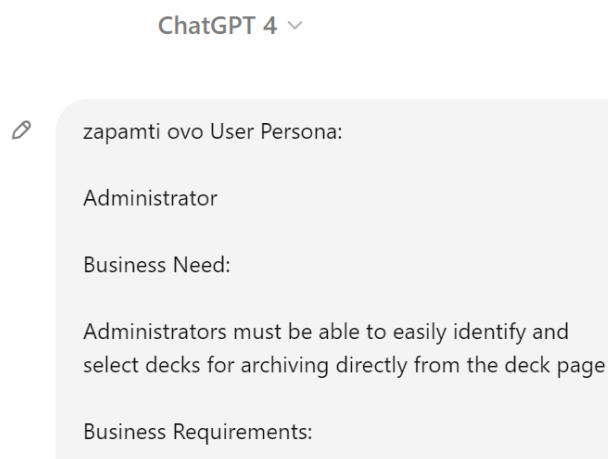
```

```
}  
}  
}
```

Za svaki zadatak koristi se novi razgovor kako bi se osiguralo da ChatGPT ne zadržava informacije iz prethodnog razgovora što znači da informacije iz jednog razgovora zadatka nisu pohranjene niti dostupne u drugom. Ovaj način rada omogućuje da se svaki zadatak obrađuje neovisno, bez prethodnih podataka koji bi mogli utjecati na objektivnost odgovora budući da se svaki zadatak tretira kao zaseban entitet.

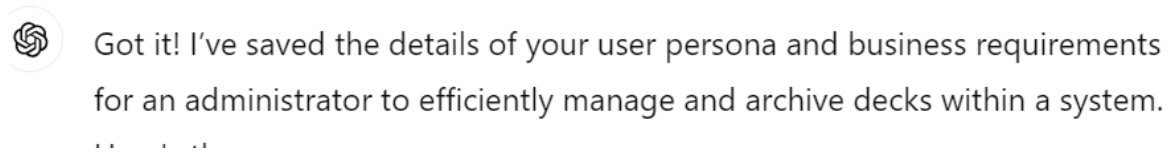
U procesu prikupljanja podataka za diplomski rad, u svakom zadatku su ChatGPT-u jasno istaknute tražene informacije. Precizno specificiran pristup omogućio je dobivanje odgovora koji su prilagođeni zahtjevima i ciljevima akademskog istraživanja. Da bi se izbjeglo preopterećivanje ChatGPT-a s informacijama, programski zadatak, programski kod i jedinice testa dostavljeni su odvojeno. Tako na primjer, nakon što je ChatGPT potvrdio primanje programskog zadatka, zatim i programskog koda, dostavljena je jedinica testa. Postavljena pitanja uključivala su napomenu da su informacije potrebe za diplomski rad. Tako metodološki pristup osigurao je strukturiranu i efikasnu komunikaciju koja je prilagođena potrebama istraživačkog procesa.

U nastavku slijedi prikaz komunikacije s ChatGPT-om koji je primijenjen kroz sve slučajeve upotrebe. Slika 7. prikazuje trenutak kada se poslovni zahtjev predaje ChatGPT-u s ciljem zapamćivanja poslovnog zahtjeva.



Slika 7. Ulazni podatak: poslovni zahtjev (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Na slici 8. vidi se odgovor ChatGPT-a koji potvrđuje da je zapamtiio dostavljene podatke.



Slika 8. Odgovor ChatGPT-a da je zapamtiio programsku dokumentaciju (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Slike 7. i 9. vizualno predstavljaju ključne korake u metodološkom procesu koji su bitni za istraživanje. ChatGPT-u pružaju specifične informacije kako bi se osiguralo da točno odgovori na postavljen upit (slika 10).

```
sad zapamti sljedeci programski kod napisan na temelju gornjeg zahtjeva using System.Threading.Tasks;
```

Slika 9. Ulazni podatak za ChatGPT - programski kod (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Na slici 10. prikazan je primjer zadatka koji je postavljen ChatGPT-u za daljnju obradu.

ChatGPT 4 ▾

```
Za potrebe diplomskog rada na temu: Upotreba umjetne inteligencije u osiguravanju kvalitete u kontekstu softverskog inženjerstva, na temelju dostavljenog programskog koda i dokumentacije za Kontrolu pristupa iz ISO/IEC 27001 provjeri je li kod u skladu s kontrolom pristupa, napisi preporuku poboljšanja i izdvoji linije iz mog dostavljenog progamrskog koda kojima potrepljujes zadovoljava li kod cilj kontrole pristupa.
```

Slika 10. Primjer zadatka postavljenog ChatGPT-u (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

5.2. Kvalitativna analiza koda prema ISO/IEC 25010

Budući da standard ISO/IEC 25010 definira modele kvalitete softvera i sustav odnosno modele kvalitete softverskog proizvoda, cilj je kroz slučaj upotrebe na konkretnom softverskom projektu primijeniti ISO/IEC 25010 normu. Na taj način bi se procijenila i unaprijedila kvaliteta koda. Fokus je na analizi osam karakteristika kvalitete softverskog proizvoda koje su definirane normom.

5.2.1. Slučaj upotrebe 1: Analiza i unapređenje kvalitete programskog koda koristeći umjetnu inteligenciju prema ISO/IEC 25010

Svrha ovog slučaja upotrebe je s pomoću umjetne inteligencije (ChatGPT-a) identificirati područja karakteristika kvalitete programskog proizvoda koja zahtijevaju unapređenja na prikazanom programskom kodu (PK1). Uz navedeno, umjetna inteligencija predlaže konkretne preporuke za unapređenje identificiranih nedostataka u programskom kodu. Korišten je ChatGPT 4 kojem se za ulazne podatke dao poslovni zahtjev, programski kod (PK1) i kreirani testovi jedinica (TJ1) – jedinični testovi. ChatGPT kao napredni sustav poznaje normu ISO/IEC 25010 te ima mogućnost evaluacije kroz karakteristike koje su njen dio. Kao izlazne informacije od njega očekuje se detaljna analiza koda prema karakteristikama kvalitete.

U nastavku slijedi prikaz odgovora ChatGPT-a koji je strukturiran prema osam karakteristika i potkarakteristikama norme ISO/IEC 25010 za koje je napravio analizu koda (u nastavku: Analiza) i preporuku unapređenja programskog koda (u nastavku: Preporuke). Karakteristike prema ChatGPT-u su: „

1. Funkcionalna prikladnost

- **Funkcionalna cjelovitost:**

Analiza: Kod sadrži sve potrebne značajke za arhiviranje deka, uključujući provjere prava pristupa, rukovanje raznim stanjima deka (aktivno, arhivirano) i postupak arhiviranja kartica unutar deka. [22]

Preporuka: Osigurati da svi potencijalni scenariji (npr. dekovi s različitim statusima kartica) budu pokriveni kroz dodatne testove ili validacije kako bi se izbjegle situacije gdje se funkcionalnost može ponašati neočekivano.

- **Funkcionalna točnost:**

Analiza: Implementirani su odgovarajući uvjeti i provjere kako bi se osiguralo da se dekovi ispravno arhiviraju ili da se generiraju odgovarajuće iznimke kada uvjeti nisu zadovoljeni.

Preporuka: Implementirati dodatne provjere ili logove kako bi se osigurala točnost svih operacija, posebno pri radu s kompleksnim objektima poput dekova i kartica.

- **Funkcionalna prikladnost:**

Analiza: Funkcionalnosti su adekvatno raspoređene, omogućujući administratorima da izvrše zadatak arhiviranja deka na siguran i kontroliran način.

Preporuka: Pružiti korisnicima više informacija ili povratnih informacija tijekom procesa arhiviranja, kako bi imali jasniji pregled što točno arhiviranje podrazumijeva.

2. Pouzdanost

- **Besprijekornost:**

Analiza: Kod pokazuje visoku razinu zrelosti kroz korištenje provjerenih obrazaca dizajna kao što su repozitoriji i servisi.

Preporuka: Redovito pregledavati i refaktorirati kod kako bi se osigurala njegova zrelost, osobito kroz uvođenje novih obrazaca dizajna ili tehnologija kada je to prikladno.

- **Dostupnost:**

Analiza: Nije eksplicitno navedeno u danom kodu, ali uz pravilnu implementaciju, sustav bi trebao osigurati dostupnost potrebnih funkcionalnosti bez neočekivanih prekida.

Preporuka: Implementirati sustave za praćenje dostupnosti funkcionalnosti, uključujući logiranje neuspješnih pokušaja arhiviranja ili drugih ključnih radnji.

- **Tolerancija na greške:**

Analiza: Kod provjerava i pravilno rukuje potencijalnim problemima, kao što su nepostojeći dekovi ili nedovoljna prava pristupa, te generira odgovarajuće iznimke.

Preporuka: Unaprijediti rukovanje iznimkama dodavanjem specifičnih iznimki za različite vrste grešaka, te implementirati mehanizme za automatsko obnavljanje ili obavještanje korisnika u slučaju greške.

- **Oporavak:**

Analiza: Proces arhiviranja je siguran; međutim, nije eksplicitno navedeno kako bi sustav postupao u slučaju kvara tijekom arhiviranja.

Preporuka: Implementirati opcije za vraćanje promjena (rollback) ili obnavljanje podataka u slučaju neuspjelog arhiviranja, čime bi se povećala sigurnost podataka.

3. Upotrebljivost

- **Prepoznatljivost prikladnosti:**

Analiza: Administratorsko sučelje i funkcionalnosti trebaju biti jasno označeni, što se može dodatno unaprijediti dokumentacijom i uputama za korištenje.

Preporuka: Unaprijediti korisničko sučelje s intuitivnim ikonama i jasnim opisima za svaku funkcionalnost, te osigurati konzistentnost u terminologiji.

- **Mogućnost učenja:**

Analiza: Korištenje funkcionalnosti arhiviranja čini se jednostavnim i intuitivnim, pod pretpostavkom da korisnici već poznaju osnovne koncepte rada sa špilovima.

Preporuka: Pružiti detaljne upute i dokumentaciju za korištenje funkcionalnosti, posebno za nove ili složene značajke kao što je arhiviranje špila.

- **Operabilnost:**

Analiza: Kod omogućuje administratorima jednostavnu interakciju s funkcionalnostima arhiviranja, s jasnim povratnim informacijama tijekom procesa.

Preporuka: Dodati vizualne ili zvučne povratne informacije kako bi korisnici bili sigurni da je radnja uspješno izvršena (npr. potvrda uspješnog arhiviranja).

4. Učinkovitost

- **Vremensko ponašanje:**

Analiza: Performanse koda nisu detaljno analizirane, ali korištenje asinkronih metoda (kao što je `async/await`) sugerira optimizaciju za bolje performanse.

Preporuka: Optimizirati upite prema bazi podataka kako bi se smanjilo vrijeme potrebno za preuzimanje i obradu podataka, te osigurati da se asinkrone operacije pravilno koriste za minimiziranje čekanja korisnika.

- **Iskorištenost resursa:**

Analiza: Kod ne sadrži vidljive resursno intenzivne operacije koje bi mogle negativno utjecati na učinkovitost sustava.

Preporuka: Redovito profilirati performanse koda i osigurati da nema nepotrebnih upotreba resursa (npr. učitavanje nepotrebnih podataka iz baze).

5. Kompatibilnost

- **Koegzistencija:**

Analiza: Kod koristi zajedničke biblioteke i standarde, što omogućava lakšu integraciju s drugim sustavima unutar veće aplikacije.

Preporuka: Osigurati da kod ostaje kompatibilan s drugim modulima i sustavima kroz temeljito testiranje integracije i korištenje standardiziranih API-ova.

- **Interoperabilnost:**

Analiza: Postoji mogućnost integracije s drugim modulima i komponentama, s obzirom na jasno definirana sučelja i metode.

Preporuka: Dokumentirati sve javne API-e i metode kako bi se olakšala integracija s vanjskim sustavima i drugim dijelovima aplikacije.

6. Sigurnost

- **Povjerljivost:**

Analiza: Kod koristi provjere prava pristupa kako bi osigurao da samo ovlašteni korisnici mogu arhivirati dekovu.

Preporuka: Implementirati dodatne sigurnosne slojeve poput enkripcije podataka i autentifikacije kako bi se osigurala povjerljivost prilikom arhiviranja i rukovanja osjetljivim podacima.

- **Integritet:**

Analiza: Arhiviranje je osigurano protiv neovlaštenih promjena kroz validacije i provjere unutar servisa.

Preporuka: Dodati mehanizme za detekciju i prevenciju neovlaštenih promjena u bazi podataka, te redovito provjeravati integritet podataka.

- **Odgovornost:**

Analiza: Logiranje akcija arhiviranja može biti uključeno za praćenje i auditiranje, što pomaže u održavanju sigurnosti sustava.

Preporuka: Povećati razinu logiranja akcija kako bi administratori imali detaljan uvid u sve promjene unutar sustava, što bi moglo uključivati identifikaciju korisnika koji su izvršili određene radnje.

7. Održavanje

- **Modularnost:**

Analiza: Kod je podijeljen u različite klase i metode, omogućujući lakoću održavanja i proširivanja funkcionalnosti.

Preporuka: Razdijeliti složene funkcionalnosti na manje, specifične module koji se mogu samostalno razvijati i održavati.

- **Ponovno korištenje:**

Analiza: Korištenje sučelja i modula omogućava ponovno korištenje komponenata u drugim dijelovima sustava.

Preporuka: Identificirati dijelove koda koji se često ponavljaju i pretvoriti ih u zajedničke biblioteke ili funkcije koje se mogu ponovno koristiti u različitim dijelovima aplikacije.

- **Mogućnost analize:**

Analiza: Kod je jasan i strukturiran, što olakšava analizu i razumijevanje funkcionalnosti.

Preporuka: Poboľšati komentiranje koda i dokumentaciju kako bi se olakšala analiza i razumijevanje različitih dijelova sustava, posebno za nove programere.

- **Modifikabilnost:**

Analiza: Funkcionalnost se može lako modificirati zahvaljujući dobro organiziranoj strukturi koda.

Preporuka: Osigurati da su sve funkcionalnosti pravilno testirane i dokumentirane prije izmjena, te implementirati sustav verzioniranja za praćenje promjena.

- **Testabilnost:**

Analiza: Postoje implementirani jedinični unit testovi koji osiguravaju da promjene u kodu ne uzrokuju regresije.

Preporuka: Povećati pokrivenost testovima, posebno jedinicama testova i testovima integracije, kako bi se osiguralo da sve promjene u kodu ne uzrokuju neočekivane probleme.“ [29]

Tako, na primjer, slika 11. prikazuje zaključak ChatGPT-a koji se odnosi na održivost koda. U ovom zaključku, ChatGPT navodi da je potrebno izmijeniti 50 linija koda kako bi se unaprijedilo održivost koda, uz procjenu da je kod relativno održiv, ali je potrebno pojednostaviti nekoliko funkcija.

- Održljivost
 - Broj linija koda za promjenu: 50
 - Procjena: Kod je relativno održiv, ali potrebno je pojednostaviti nekoliko funkcija.

Slika 11. Zaključak ChatGPT-a koji se odnosi na održivost koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

“8. Prenosivost

○ Prilagodljivost:

Analiza: Kod bi se mogao relativno lako prilagoditi za druge sustave, s obzirom na to da se koriste standardne tehnologije i pristupi.

Preporuka: Koristiti standardizirane tehnologije i metode kako bi se omogućila lakša prilagodba koda za različite platforme ili okruženja.

○ Instalabilnost:

Analiza: Kod nema izravne informacije o instalaciji, ali modularni pristup sugerira lakšu implementaciju u različitim okruženjima.

Preporuka: Razviti automatizirane skripte za postavljanje i instalaciju aplikacije kako bi se olakšalo uvođenje sustava u nova okruženja.

- **Zamjenjivost:**

Analiza: Komponente su neovisne, što omogućava zamjenu dijelova sustava bez značajnih problema.

Preporuka: Osigurati da sve komponente imaju jasno sučelje i minimalne ovisnosti kako bi se omogućila njihova zamjena bez velikih utjecaja na ostatak sustava.“ [29]

Nakon provedene detaljne analize ChatGPT-a u kontekstu norme ISO/IEC 25010 kreira zaključak s prikazanim konkretnim učinkom:

- **Zaključci o kvaliteti koda:** „Kod je općenito kvalitetan, ali postoji nekoliko područja koja treba unaprijediti.“ [29]
- **Konkretni učinak koji ChatGPT navodi kroz izraz (slika 11):** “Broj linija koda za promjene 50”. [29] Što znači da postoji 50 linija koda koje bi trebalo modificirati kako bi se kod učinio jednostavnijim za održavanje. Slika 12. prikazuje konkretne smjernice za unapređenje održivosti koda:“
 1. Smanjenje složenosti funkcija:
 - Podijelite složene funkcije na manje, samostalne funkcije koje obavljaju jednu zadataku.
 - Ova praksa olakšava testiranje, razumijevanje i ponovno korištenje koda.
 2. Unapređenje modularnosti:
 - Učinite vaš kod modularnijim tako da grupirate povezane funkcionalnosti zajedno.
 - Koristite klase i module za organizaciju koda.
 3. Dodavanje komentara i dokumentacije:
 - Dodajte komentare kako biste objasnili svrhu složenijih dijelova koda.
 - Koristite XML dokumentaciju za opisivanje metoda i njihovih parametara.
 4. Refaktoriranje ponavljajućeg koda:
 - Identificirajte i refaktorirajte ponavljajući kod u zajedničke funkcije ili metode.
 - Ovo smanjuje dupliciranje koda i olakšava održavanje.“ [29]

Održljivost se odnosi na lakoću kojom se kod može održavati, što uključuje ispravljanje grešaka, dodavanje novih funkcionalnosti, poboljšanje performansi itd.

Što to znači u vašem kontekstu?

U kontekstu vašeg koda, ovo znači da postoji 50 linija koda koje bi trebalo modificirati kako bi se kod učinio jednostavnijim za održavanje. Ovdje su konkretne smjernice za poboljšanje održivosti vašeg koda:

1. **Smanjenje složenosti funkcija:**
 - Podijelite složene funkcije na manje, samostalne funkcije koje obavljaju jednu zadataku.
 - Ova praksa olakšava testiranje, razumijevanje i ponovno korištenje koda.
2. **Poboljšanje modularnosti:**
 - Učinite vaš kod modularnijim tako da grupirate povezane funkcionalnosti zajedno.
 - Koristite klase i module za organizaciju koda.
3. **Dodavanje komentara i dokumentacije:**
 - Dodajte komentare kako biste objasnili svrhu složenijih dijelova koda.
 - Koristite XML dokumentaciju za opisivanje metoda i njihovih parametara.
4. **Refaktorisanje ponavljajućeg koda:**
 - Identificirajte i refaktorisirajte ponavljajući kod u zajedničke funkcije ili metode.
 - Ovo smanjuje dupliranje koda i olakšava održavanje.

Slika 12. Prijedlozi za unapređenje 50 linija programskog koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Linije programskog koda koje je ChatGPT modificirao:

```
public async Task<DeckArchiver?> GetById(IntegerId id)
{
    var unarchivedDeckCards = await CardEntityDbSet
        .Include(x => x.CardResources)
        .Where(x => x.DeckId == id.Value && x.CardStatusType !=
CardStatusType.Archived).ToListAsync();

    var deck = await
DeckEntityDbSet.DefaultIfEmpty().SingleOrDefaultAsync(x => x.Id == id);

    var deckWithUnarchivedCards = new DeckArchiver(deck.Id,
deck.DeckStatusType, deck.ArchiveOnDate, unarchivedDeckCards.Select(x =>
cardLifeCycleMapper.ConvertToCardLifeCycle(x)).ToList());

    return deckWithUnarchivedCards;
}
```

Modificirane linije koda koje je ChatGPT kreirao:

```
public async Task<DeckArchiver?> GetById(IntegerId id)
{
```



```

        var unarchivedDeckCards = await GetUnarchivedDeckCards(id);

        var deck = await GetDeckById(id);

        return CreateDeckArchiver(deck, unarchivedDeckCards);
    }

private async Task<List<CardEntity>> GetUnarchivedDeckCards(IntegerId id)
{
    return await CardEntityDbSet
        .Include(x => x.CardResources)
        .Where(x => x.DeckId == id.Value && x.CardStatusType !=
CardStatusType.Archived).ToListAsync();
}

private async Task<DeckEntity> GetDeckById(IntegerId id)
{
    return await DeckEntityDbSet.DefaultIfEmpty().SingleOrDefaultAsync(x =>
x.Id == id);
}

private DeckArchiver CreateDeckArchiver(DeckEntity deck, List<CardEntity>
unarchivedDeckCards)
{
    var deckWithUnarchivedCards = new DeckArchiver(deck.Id,
deck.DeckStatusType, deck.ArchiveOnDate, unarchivedDeckCards.Select(x =>
cardLifeCycleMapper.ConvertToCardLifeCycle(x)).ToList());

    return deckWithUnarchivedCards;
}

```

ChatGPT uz modificirani kod pruža i narativno objašnjenje unapređenja kako bi se programer lakše navodio: „

1. Smanjenje složenosti funkcija:

- Originalna metoda `GetById` je podijeljena na tri manje metode: `GetUnarchivedDeckCards`, `GetDeckById` i `CreateDeckArchiver`.
- Ovo smanjuje složenost pojedinačnih metoda i čini kod lakšim za razumijevanje i testiranje.

2. Unapređenje modularnosti:

- Razdvajanje funkcionalnosti u zasebne metode unapređuje modularnost koda.
- Sada je lakše ponovno koristiti ove metode u drugim dijelovima koda, ako je potrebno.

3. Dodavanje komentara:

- Možete dodati XML komentare za svaku metodu kako biste dodatno unaprijedili čitljivost i dokumentaciju koda.“ [29]

Pregledom detaljne analize i konkretnog primjera pruženog od strane umjetne inteligencije, utvrđeno je da poznaje normu, da može kritički analizirati programski kod i pružiti konkretna unapređenja. Generalno gledano, umjetna inteligencija ima svoju svrhu u unapređenju programskog koda u skladu s normom ISO/IEC 25010. Njegova učinkovitost je vidljiva i prikazana. Optimiziranim programskim kodom olakšava se održavanje i proširivost programskog koda u budućnosti. Ovim promjenama s jedne strane smanjuje se rizik stvaranja grešaka, a s druge strane se povećava mogućnost prilagodljivosti programskog koda različitim potrebama i okruženjima. Provedenom analizom ChatGPT je pokrio većinu važnih karakteristika i potkarakteristika koje su u skladu s normom ISO/IEC 25010. Međutim, postoji nekoliko iznimaka, kao što je kategorija fleksibilnosti koja nije detaljno razrađena te kategorija **sigurnosti** (eng. *safety*), koju ChatGPT nije uopće spomenuo prilikom provedbe analize. Glavni zaključak je da je ChatGPT značajno olakšao izradu ove analize, ali još nije dovoljno samostalan da bi se njegovu zaključci uzeli u obzir bez temeljite provjere od strane stručne osobe, u ovom slučaju programera.

5.2.2. Slučaj upotrebe 2: Vrednovanje programskog koda i jedinice testa pomoću umjetne inteligencije prema normi ISO/IEC 25010

U ovom slučaju upotrebe zadatak umjetne inteligencije, ChatGPT-a je vrednovanje i ocjenjivanje programskog koda prema karakteristikama kvalitete softvera prema normi ISO/IEC 25010. Cilj je potvrditi u kojoj mjeri su programski kod i jedinice testa usklađeni s normom ISO/IEC 25010. Usklađenost s ovom normom doprinosi izradi visokokvalitetnog programskog proizvoda koji je siguran, pouzdan i lagan za održavanje što rezultira većim zadovoljstvom i uspjehom u korištenju programskog proizvoda. Za ostvarenje vrednovanja koristi se tablica ocjenjivanja za svaku karakteristiku i potkarakteristiku kvalitete softvera. Za zadatak potrebno je umjetnoj inteligenciji omogućiti podatke za rad. U skladu s time, umjetnoj inteligenciji se za rad u ovom slučaju upotrebe dostavljeni su programski kod i jedinice testa.

Vrednovanje programskog koda provodi se odabirom ocjena na skali od 1 do 5. U ovom slučaju, ocjena 1 označava najmanje zadovoljavanje kriterija što ukazuje na značajne nedostatke koji zahtijevaju dodatnu pažnju programera. Suprotno tome, ocjena 5 predstavlja maksimalno zadovoljavanje kriterija koji ukazuje da programski kod i jedinice testa zadovoljavaju sve postavljene zahtjeve kvalitete i pokazuju vrhunski izvedbu. Tablica se sastoji od četiri stupaca: Potkarakteristika, Opisa za zadanu potkarakteristiku o programskom kodu, Ocjene i komentara za ocjene.

Odgovor ChatGPT-a ocijenjen je i kvalificiran u tablicama prema definiranim kriterijima i potkriterijima. Svaka tablica sadrži ocjenu za pojedini kriterij, zajedno s komentarima koji objašnjavaju odabranu ocjenu.

U tablici 1. ChatGPT je ocijenio karakteristiku Funkcionalna prikladnost:

Tablica 1. Ocjena ChatGPTa za karakteristiku Funkcionalna prikladnost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Funkcionalna potpunost (eng. <i>Functional completeness</i>)	Implementirane sve potrebne funkcionalnosti	5	Sve potrebne funkcionalnosti su implementirane.
Funkcionalna ispravnost (eng. <i>Functional correctness</i>)	Funkcije daju točne rezultate	4	Funkcije su dobro implementirane, ali bi bilo korisno dodati više testiranja.
Funkcionalna prikladnost (eng. <i>Functional appropriateness</i>)	Funkcije su prikladne za zadatke	5	Funkcije su prikladne za arhiviranje deckova.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Tablica 1. prikazuje ocjene kvalitete programskog koda koje je ChatGPT ocijenio na temelju tri karakteristike unutar karakteristike Funkcionalna prikladnost. Karakteristika Funkcionalna prikladnost se odnosi na to koliko dobro programski kod ispunjava funkcionalne zahtjeve. Sveukupno, programski kod se ocjenjuje kao vrlo dobar, gdje ima prostora za dodatno unapređenje, a posebno u pogledu na testiranje.

U tablici 2. ChatGPT je ocijenio karakteristiku Pouzdanost:

Tablica 2. Ocjena ChatGPT-a za karakteristiku Pouzdanost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Besprijekornost (eng. <i>Faultlessness</i>)	Softver ima malo grešaka	4	Kod je robustan, ali nema mnogo provjerenih testova.
Dostupnost (eng. <i>Availability</i>)	Softver je dostupan kad je potreban	5	Kod koristi asinkrone operacije za osiguranje dostupnosti.
Otpornost (eng. <i>Recoverability</i>)	Softver se oporavlja od grešaka	3	Nema mnogo error handling mehanizama.
Tolerancija na greške (eng. <i>Fault tolerance</i>)	Softver se nosi s greškama	3	Tolerancija na greške bi se mogla unaprijediti.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

ChatGPT u tablici 2. ocjenjuje ključne aspekte pouzdanosti programskog koda. Programski kod ocijenjen je kao prilično zreo i dostupan, ali kao i za prethodnu karakteristiku, postoji prostor za unapređenje. Prostor za unapređenje vidljiv je posebno u pogledu otpornosti

i sposobnosti za rukovanje pogreškama. Preporuke za unapređenje usmjerene su ka dodatnom testiranju i unapređenju mehanizama za upravljanje greškama. Unapređenjem bi se povećala ukupna pouzdanost programskog koda.

U tablici 3. ChatGPT je ocijenio karakteristiku Upotrebljivost:

Tablica 3. Ocjena ChatGPT-a za karakteristiku Upotrebljivost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Prepoznatljivost (engl. Appropriateness Recognizability)	Korisnici prepoznaju potrebne funkcije	5	Funkcije su intuitivne i jasno imenovane.
Naučivost (engl. Learnability)	Softver se lako uči	4	Kod je dobro organiziran, ali bi dokumentacija mogla biti bolja.
Operabilnost (engl. Operability)	Softver se lako koristi	4	Funkcije su jednostavne za korištenje.
Zaštita od pogrešaka (engl. user error protection)	Smanjuje rizik korisničkih pogrešaka	3	Nema dovoljno provjere ulaza.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Ocjene koje je dodijelio ChatGPT u tablici 3. prikazuje da je programski kod vrlo intuitivan i jednostavan za korištenje, iako postoji prostor za unapređenjem. Ta unapređenja su najpotrebnija u područjima dokumentacije i zaštite od pogrešaka. Unapređenjem ovih

aspekata, dodatno bi se unaprijedilo korisničko iskustvo i smanjio rizik od grešaka tijekom korištenja programskog proizvoda.

U tablici 4. ChatGPT je ocijenio karakteristiku Učinkovitost:

Tablica 4. Ocjena ChatGPT-a za karakteristiku Učinkovitost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Vrijeme izvršavanja (eng. <i>Time behaviour</i>)	Prihvatljivo vrijeme odziva	4	Asinkrone operacije poboljšavaju vrijeme odziva.
Iskorištenost resursa (eng. <i>Resource utilization</i>)	Učinkovita upotreba resursa	4	Kod efikasno koristi resurse.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Tablica 4. prikazuje dva ključna aspekta ove karakteristike, Vrijeme izvršavanja i iskorištenost resursa. Oba aspekta ChatGPT je ocijenio ocjenom vrlo dobar. Naglasak je na pozitivnim aspektima, uz prepoznavanje prostora za dodatna unapređenja.

U tablici 5. ChatGPT je ocijenio karakteristiku Kompatibilnost:

Tablica 5. Ocjena ChatGPT-a za karakteristiku Kompatibilnost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Koegzistencija (eng. <i>Co-existence</i>)	Softver koegzistira s drugim proizvodima	5	Nema konflikata s drugim modulima.

Interoperabilnost (eng. <i>Interoperability</i>)	Softver komunicira s drugim proizvodima	5	Dobar dizajn API-ja.
--	--	---	----------------------

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Tablica 5. prikazuje dvije potkarakteristike prema kojima se ocjenjuje programski kod unutar karakteristike kompatibilnost. Ocjene 5 sugeriraju visoku razinu kompatibilnosti programskog koda. Navedeni aspekti osiguravaju da programski kod može učinkovito raditi u različitim sustavima i okruženjima, a ocjena 5 samo potvrđuje da je programski kod spreman za suradnje i integraciju u složena IT okruženja.

U tablici 6. ChatGPT je ocijenio karakteristiku Sigurnost:

Tablica 6. Ocjena ChatGPT-a za karakteristiku Sigurnost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Povjerljivost (eng. <i>Confidentiality</i>)	Zaštita podataka od neovlaštenog pristupa	4	Dobro upravljanje pristupom, ali bi moglo biti dodatne provjere.
Integritet (eng. <i>Integrity</i>)	Zaštita podataka od neovlaštenih promjena	4	Implementirani su osnovni mehanizmi zaštite.
Neporecivost (eng. <i>Non-repudiation</i>)	Mogućnost provjere izvršenja radnje	3	Logiranje i praćenje aktivnosti bi moglo biti unapređeno.

Odgovornost (eng. <i>Accountability</i>)	Praćenje aktivnosti korisnika	3	Nedovoljno praćenje aktivnosti.
Autentičnost (eng. <i>Authenticity</i>)	Provjera identiteta korisnika	4	Dobra implementacija provjere identiteta.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

U tablici 6. ocjenjuje se programski kod prema nekoliko potkarakteristika (5) unutar karakteristike sigurnost. Karakteristika sigurnost odnosi se na zaštitu podataka i sigurnost sustava. Ocjene i popratni komentari ukazuju da postoji prostor za unapređenjem s obzirom na to da je prosječna ocjena 3,6. Unapređenja su potrebna u području neporecivosti i odgovornost gdje bi praćenje i logiranje aktivnosti korisnika unaprijedilo ukupnu sigurnost programskog koda, a samim time i cijelog programskog proizvoda.

U tablici 7. ChatGPT je ocijenio karakteristiku Održavanje:

Tablica 7. Ocjena ChatGPT-a za karakteristiku Održavanje

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Modularnost (eng. <i>Modularity</i>)	Softver je podijeljen na module	5	Kod je dobro modulariziran.
Ponovna upotrebljivost (eng. <i>Reusability</i>)	Dijelovi softvera su ponovno upotrebljivi	4	Mnogo koda se može ponovno koristiti.
Analizabilnost (eng. <i>Analysability</i>)	Softver je lako analizirati	4	Kod je relativno lako analizirati.

Izmjenjivost (eng. <i>Modifiability</i>)	Softver je lako modificirati	4	Kod se lako može modificirati.
Testabilnost (eng. <i>Testability</i>)	Promjene u softveru su provjerljive	3	Nema dovoljno testova za provjeru promjena.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Tablica 7. ukazuje da je ChatGPT ocijenio da je softver generalno dobro održavan, s jakim aspektima poput modularnosti i ponovne upotrebljivosti, dok postoji prostor za unapređenjem u testabilnosti koja se pokazala kao najslabija točka.

U tablici 8. ChatGPT je ocijenio karakteristiku Prenosivost:

Tablica 8. Ocjena ChatGPT-a za karakteristiku Prenosivost

Potkarakteristika	Opis	Ocjena (1-5)	Komentari
Prilagodljivost (eng. <i>Adaptability</i>)	Softver se prilagođava različitim okruženjima	4	Softver se lako prilagođava različitim okruženjima.
Instalabilnost (eng. <i>Installability</i>)	Softver se lako instalira	4	Instalacija je jednostavna.
Zamjenjivost (eng. <i>Replaceability</i>)	Softver se može zamijeniti drugim softverom	4	Softver se može lako zamijeniti drugim rješenjem.

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

ChatGPT na kraju iznos plan unapređenja za programski kod: „

1. Povećati pokrivenost testovima: Dodati više jedinica testova i integracijskih testova kako bi se unaprijedila ispravnost, pouzdanost i provjerljivost softvera.
2. Unaprijediti upravljanje greškama (eng. *error handling*): Implementirati robusnije mehanizme za rukovanje greškama.
3. Unaprijediti dokumentaciju: Povećati kvalitetu dokumentacije kako bi softver bio lakši za učenje i održavanje.
4. Provjera ulaza: Implementirati provjere ulaza kako bi se spriječile korisničke pogreške i povećala sigurnost.
5. Logiranje i praćenje: Dodati više logiranja i mehanizama za praćenje aktivnosti kako bi se unaprijedila sigurnost i odgovornost.“ [29]

Analiza programskog koda koju je proveo ChatGPT uvelike se podudara s karakteristikama kvalitete prema ISO 25010 standardu. Prema ISO/IEC 25010 ChatGPT je vrednovanjem programskog koda pokazao koliko programski kod zadovoljava ključne karakteristike kvalitete, ali i otkrio je područja za unapređenje. Uz navedeno, predložene su promjene usmjerene na povećanje pokrivenosti koda što može biti dodatna kontrola za programere što je ključno zbog složenosti projekta gdje su propusti mogući. Primjenom ovih preporuka podiže se razina kvalitete i usklađenosti s međunarodnom normom. ChatGPT učinkovito pruža detaljne preporuke koje pomažu programeru u detekciji nedostataka i njihovoj implementaciji. Nakon izvršenih izmjene, programer može ponovo unijeti kod u ChatGPT kako bi provjerio je li programski kod unaprijeđen. Međutim, važno je istaknuti kako ChatGPT nije uzeo u razmatranje sve potkarakteristike, kao što su kapacitet, otpornost i skalabilnost. Unapređenjem ovih aspekata, analiza bi bila još više usklađena s ISO/IEC 25010 normom te na taj način pružila sveobuhvatniju procjenu kvalitete programskog koda.

5.2.3. Slučaj upotrebe 3: Primjena metode Analitičkog hijerarhijskog procesa za vrednovanje programskog koda u skladu s normom ISO/IEC 25010

Cilj ovog slučaja upotrebe je osigurati objektivnost u analizi programskog koda primjenom metode Analitičkog hijerarhijskog procesa (AHP). AHP je jedna od najprikladnijih metoda za donošenje odluka u složenim situacijama. U okviru analize programskog koda u skladu s normom ISO/IEC 25010 koristi se AHP metoda za preciznije ocjenjivanje usklađenosti

programskog koda s definiranom kvalitetom softvera. S pomoću AHP metode želi se smanjiti subjektivnost ChatGPT-a prilikom ocjenjivanja.

U nastavku slučaja upotrebe prikazuje se provedeno vrednovanje ChatGPT-a.

Kao prvi korak u procesu vrednovanja potrebno je prikupiti podatke. Ulazni podaci kojima ChatGPT raspolaže uključuje programski kod.

1. Prikupljanje podataka

Prilikom prikupljanja podataka ChatGPT ističe da koristi subjektivnu procjenu na temelju ručnog pregleda koda koja je uključila stručno vrednovanje programskog koda prema definiranim kriterijima iz norme IOS/IEC 25010. Iako se radi o subjektivnim ocjenama, AHP metoda omogućuje strukturirano i nepristrano ponderiranje ocjena što na kraju donosi objektivnosti ukupnog vrednovanja.

Drugi korak koji ChatGPT provodi zove se usporedba u parovima glavnih karakteristika kvalitete softvera prema ISO/IEC 25010.

Kriteriji koje ChatGPT navodi kao ključnim aspektima kvalitete softverskog koda prema kojima će uspoređivati su: „

- **Funkcionalnost:** Koliko dobro kod ispunjava poslovne zahtjeve.
- **Čitljivost:** Koliko je kod jasan i lako razumljiv za druge developere.
- **Učinkovitost:** Koliko je kod optimiziran u pogledu performansi.
- **Sigurnost:** Koliko je kod siguran od potencijalnih ranjivosti.
- **Održavanje:** Koliko je kod modularan i jednostavan za održavanje i proširenje.” [29]

Odabrani su kriteriji prema ključnim aspektima kvalitete softverskog koda.

2. Matrica usporedbe u parovima

Kako bi se odredila relativna važnost unutar tablice 9. parna usporedba uspoređuju se karakteristike međusobno jedna s drugom. Tablicu parne usporedbe s pet glavnih karakteristika kreirao je ChatGPT i prikazana je u nastavku:

Tablica 9. Tablica matrice parnih usporedbi

	Funkcionalnost	Čitljivost	Učinkovitost	Sigurnost	Održavanje
Funkcionalnost	1	3	3	5	5
Čitljivost	1/3	1	1	3	3
Učinkovitost	1/3	1	1	3	3
Sigurnost	1/5	1/3	1/3	1	1
Održavanje	1/5	1/3	1/3	1	1

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Objašnjenje koje ChatGPT kreira uz kreiranu matricu je sljedeće: „

- **Funkcionalnost** je dvostruko važnija od **Čitljivosti** (vrijednost 3).
- **Funkcionalnost** je tri puta važnija od **Sigurnosti** i **Održavanja** (vrijednost 5).
- **Čitljivost** i **Učinkovitost** su jednako važni (vrijednost 1).
- **Čitljivost** je tri puta važnija od **Sigurnosti** i **Održavanja** (vrijednost 3).“ [29]

3. Normalizacija matrice

Završetkom izračunatih odnosa postavljenih kriterija u matrici parnih usporedbi, pristupa se normalizaciji matrice. Normalizacijom matrice cilj je dobiti normalizirane vrijednosti koje se mogu uspoređivati. U tablici 10. se izračunava suma svake kolone i podijele svakog elementa u koloni s tom sumom.

ChatGPT za svaku kolonu izračunava sumu. Suma kolone Funkcionalnost iznosi 2.07 ($1 + 0.333 + 0.333 + 0.2 + 0.2 = 2.066$). Za kolonu čitljivost suma iznosi 5.67 ($3 + 1 + 1 + 0.333 + 0.333 = 5.666$). Suma za kolonu učinkovitost iznosi 5.67 ($3 + 1 + 1 + 0.333 + 0.333 = 5.66$). Nadalje, suma za kolonu održavanje je 13.00 ($5 + 3 + 3 + 1 + 1 = 13$).

Za izračun normalizacije matrice, svaki element u matrici dijeli se s odgovarajućom sumom kolone kako bi se dobila normalizirana matrica.

„Rezultati sume svake kolone su:

- **Funkcionalnost:** 2.066
- **Čitljivost:** 5.666
- **Učinkovitost:** 5.666
- **Sigurnost:** 13
- **Održavanje:** 13” [29]

Sljedeći korak za provedbu normalizacije koji ChatGPT navodi je da se svaki element matrice podijeli s odgovarajućom sumom kolone. U tablici 10 prikazane su normalizirane vrijednosti.

Tako na primjer za kolonu Funkcionalnost ChatGPT izračunava vrijednosti: “

- **Funkcionalnost vs Funkcionalnost:** $\frac{1}{2.066} = 0.484$
- **Čitljivost vs Funkcionalnost:** $\frac{1/3}{2.066} = 0.161$
- **Učinkovitost vs Funkcionalnost:** $\frac{1}{2.066} = 0.161$
- **Sigurnost vs Funkcionalnost:** $\frac{1}{2.066} = 0.097$
- **Održavanje vs Funkcionalnost:** $\frac{1}{2.066} = 0.097$ “ [29]

Tablica 10. Tablica normalizacije matrice

	Funkcionalnost	Čitljivost	Učinkovitost	Sigurnost	Održavanje
Funkcionalnost	0.484	0.529	0.529	0.385	0.385
Čitljivost	0.161	0.176	0.176	0.231	0.231
Učinkovitost	0.161	0.176	0.176	0.231	0.231

Sigurnost	0.097	0.059	0.059	0.077	0.077
Održavanje	0.097	0.059	0.059	0.077	0.077

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

Kao i za prethodnu tablicu, ChatGPT i za normaliziranu matricu parnih usporedbi objašnjava svaku vrijednost u matrici predstavljanjem normalizirane relativne važnosti kriterija u odnosu na druge kriterije. Tako na primjer, funkcionalnost ima vrijednost 0.484 u svojoj koloni te to predstavlja njenu relativnu važnost.

4. Izračunavanje težinskih faktora

Težinski faktor za svaku karakteristiku izračunava se kao prosjek normaliziranih vrijednosti za tu karakteristiku. Težinski faktor ima veliku važnost kod svake karakteristike u ukupnoj ocjeni programskog koda.

ChatGPT je kreirao težinski faktor za svaku karakteristiku prikazanu u tablici 11.

Tablica 11. Tablica težinskih faktora

	Težinski faktor
Funkcionalnosti	0.462
Čitljivost	0.195
Učinkovitost	0.195
Sigurnost	0.074

Održavanje	0.074
	1

(Izvor: Izrada autorice prema danim odgovorima ChatGPT-a)

ChatGPT kreira i postupak izračuna svakog težinskog faktora: „

- Funkcionalnost: prosjek redaka $\frac{0.484+0.529+0.529+0.385+0.385}{5} = 0.462$
- Čitljivost: prosjek redaka $\frac{0.161+0.176+0.176+0.231+0.231}{5} = 0.195$
- Učinkovitost: prosjek redaka $\frac{0.161+0.176+0.176+0.231+0.231}{5} = 0.195$
- Sigurnost: prosjek redaka $\frac{0.097+0.059+0.059+0.077+0.077}{5} = 0.074$
- Održavanje: prosjek redaka $\frac{0.097+0.059+0.059+0.077+0.077}{5} = 0.074$

Provjera zbroja težinskih faktora je: $0.462+0.195+0.195+0.074+0.074=1$ [29]

Sljedeći korak u AHP metodi je primjena težinskih faktora na ocjene svakog kriterija kako bi se izračunala ukupna ocjena za opcije koje se procjenjuju.

4. Ocjenjivanje softvera

ChatGPT navodi kako je na temelju prethodnih koraka programski kod prema svakom kriteriju ocijenio koristeći skalu od 1 do 5, koju je sam odabrao. 1 označava najmanje zadovoljavanje kriterija, a 5 predstavlja potpuno zadovoljavanje kriterija: „

- Funkcionalna prikladnost: 4/5
- Čitljivost: 3/5
- Učinkovitost: 4/5
- Sigurnost: 3/5
- Održavanje: 4/5 “ [29]

5. Izračunavanje ponderirane ocjene

Ponderirana ocjena dobiva se množenjem ocjene za svaki kriterij s odgovarajućim težinskim faktorom. ChatGPT izračunava ponderirane ocjene: „

- **Funkcionalnost:** $4 * 0.462 = 1.848$
- **Čitljivost:** $3 * 0.195 = 0.585$
- **Učinkovitost:** $4 * 0.195 = 0.78$
- **Sigurnost:** $3 * 0.074 = 0.222$
- **Održavanje:** $4 * 0.074 = 0.296$ “ [29]

6. Izračunavanje ukupne ocjene

Ukupna ocjena koju je ChatGPT izračunao je: $1.848 + 0.585 + 0.78 + 0.222 + 0.296 = 3.731$. Ocjena je izračunata zbrojem svih ponderiranih ocjena.

Uzevši u obzir sve karakteristike kvalitete ChatGPT ocjenjuje kvalitetu programskog koda prema normi ISO/IEC 25010 s ocjenom 3.731 od mogućih 5. Ova ocjena pokazuje da programski kod zadovoljava solidnu kvalitetu koda, ali se preporučuje kontinuirano praćenje i unapređenje, posebno u područjima s nižim ocjenama kako bi se osigurala potpuna usklađenost s normom, posebno u kriterijima Čitljivost i Sigurnost.

Nakon provedene analize AHP metodom i dobivene ocjene 3.731, pred ChatGPT se stavlja novi izazov, a to je unaprijediti programskog kod kako bi i ocjena bila viša, što će rezultirati dodatnim povećanjem ukupne kvalitete i usklađenosti softvera s normom ISO/IEC 25010. S obzirom na to da je kvaliteta programskog koda ima ključnu ulogu u pouzdanosti, sigurnosti, ali i u upotrebljivosti ova analiza vrednovanja metodom AHP omogućuje dubinsku analizu trenutnog stanja programskog koda.

U odgovoru ChatGPT navodi sljedeće: „

Unaprijediti čitljivost:

- Dodaj komentare: u slučaju kad kod sadrži složenu logiku, dodavanje komentara koji objašnjavaju svrhu svakog bloka značajno unapređuje čitljivost.
- Refaktoriraj imena varijabli i funkcija: provjeri jesu li imena varijabli i funkcija deskriptivna i intuitivna. Zamijeni nejasna imena s onima koja bolje opisuju njihovu svrhu.
- Konzistentan stil kodiranja: usvoji normatizirani stil kodiranja (npr. korištenje konzistentnih uvlača, razmaka i formata), što može olakšati čitanje i održavanje koda.

2. Povećanje sigurnosti:

- Provjere unosa: uvjeri se da su svi korisnički unosi pravilno provjereni i očišćeni kako bi se spriječili potencijalni napadi (npr. SQL injekcije ili XSS napadi).
- Dodaj enkripciju: u slučaju ako kod obrađuje osjetljive podatke, implementiraj enkripciju za pohranu ili prijenos tih podataka.
- Unaprijedi provjere pristupa: osiguraj da korisnici imaju samo one privilegije koje su im potrebne, te da su sve provjere pristupa pravilno implementirane.

3. Unapređenje održavanja:

- Modularnost koda: razdvoji velike funkcije na manje, specijalizirane funkcije koje obavljaju samo jedan zadatak. Ovo će olakšati održavanje i ponovnu upotrebu koda.
- Korištenje standardnih biblioteka: ako koristiš vlastite implementacije za određene funkcije, razmotri korištenje provjerenih standardnih biblioteka koje mogu smanjiti kompleksnost koda i unaprijediti njegovu pouzdanost.“ [29]

Također, uz navedeno ChatGPT daje i konkretan prijedlog na programskom kodu za provjere unosa i sigurnosne mjere:

```
public void SaveUserInput(string userInput)
{
    // Provjera da li unos sadrži nepoželjne znakove
    if (!IsValidInput(userInput))
```

```

    {
        throw new Exception("Invalid input detected");
    }

    // Spremanje unosa u bazu podataka
    var sanitizedInput = SanitizeInput(userInput);
    dbContext.SaveInput(sanitizedInput);
}

private bool IsValidInput(string input)
{
    // Implementiraj logiku za validaciju unosa
    return !input.Contains("<script>") && !input.Contains("DROP TABLE");
}

private string SanitizeInput(string input)
{
    // Implementiraj logiku za čišćenje unosa
    return input.Replace("<", "&lt;").Replace(">", "&gt;");
}

```

Primjena AHP metode u vrednovanju programskog koda u skladu s normom ISO/IEC 25010 pokazala se kao učinkovit pristup za smanjenje subjektivnosti prilikom procjene kvalitete softvera. Još jednom očituje se sposobnost ChatGPT-a da pruži uravnotežene procjene, što uključuje i dozu samokritičnosti, jer nije sve ocjene dodijelio s najvišom ocjenom 5. To pokazuje da ChatGPT može objektivno sagledati prednosti i nedostatke programskom koda, čine omogućuje precizniju analizu. Vidljivo je kako ovaj alat može imati značajnu ulogu u primjeni softverskog inženjerstva, pomažući programerima ne samo u identificiranju područja za unapređenje, već i u postizanju veće kvalitete softverskih rješenja. Iako je AHP metoda učinkovita u smanjenju subjektivnosti i omogućava strukturirano vrednovanje, ona također ima svoja ograničenja. Odluke o težinskim faktorima i parnim usporedbama u velikoj mjeri ovise o procjenama koje su subjektivne i ovise o tome tko provodi analizu.

5.3. Upotreba umjetne inteligencije u analizi sigurnosnih mjera u skladu s normom ISO/IEC 27001

Glavni izazov koji se rješava u ovom poglavlju je osiguranje kvalitete koda i sigurnosnih mjera na programskom kodu uz pomoć umjetne inteligencije. To se postiže implementacijom sigurnosnih mjera u skladu s normom ISO/IEC 27001.

5.3.1.Slučaj upotrebe 4: Primjena umjetne inteligencije za provjeru usklađenosti programskog koda s normom ISO/IEC 27001 za funkcionalnost arhiviranja kartica

Ovaj slučaj upotrebe prikazuje primjenu umjetne inteligencije za provjeru usklađenosti programskog koda s normom ISO/IEC 27001. ChatGPT detaljno analizira kod prema kontrolama i ciljevima norme. Za ulazne podatke ovog slučaja upotrebe ChatGPT-u zadani su poslovni zahtjev, programski kod i testne jedinice.

U nastavku su pobrojani izdvojene kontrole (6) iz norme ISO/IEC 27001. Za svaku kontrolu ChatGPT opisuje što ona predstavlja, daje preporuku kako je provesti, konkretno na programskom kodu izdvaja relevantne programske linije te predlaže unapređenje. „

1. Kontrola pristupa (eng. *Access Control* - A.9.)

- **Ciljevi kontrole A.9 koje je ChatGPT izdvojio:**
 1. Provjera prava za arhiviranje - Kod uključuje provjeru dozvola za arhiviranje dekov s pomoću `IPermissionValidator` servisa. Ovo je ključni dio za kontrolu pristupa koji osigurava da samo ovlašteni korisnici mogu izvršiti akciju arhiviranja.
 2. Pristup podacima - Pristup dekovima i karticama reguliran je preko `IDeckArchiverRepository`, što osigurava da se operacije nad bazom podataka obavljaju preko definiranih metoda koje implementiraju sigurnosne provjere.
 3. Autentifikacija i sigurnost podataka - Korištenje `EncryptedInt` za identifikaciju korisnika i tenant ID-a pruža dodatni sloj sigurnosti podataka u

transakcijama. Ovo pomaže u zaštiti identiteta i sprječavanju neovlaštenog pristupa.“ [29]

- **Izdvojene linije programskog koda kojima ChatGPT potkrepljuje zadovoljavanje cilja norme ISO/IEC 27001 prikazane su na slikama 13., 14. i 15.:**

ChatGPT 4 ▾

Relevantni odlomci iz koda:

- ``mock.Mock<IPermissionValidator>().MockCanUpdateDeck(canUpdateDeck);`` (Mock setup)
- ``if (!await permissionValidator.CanUpdateDeck(deck.EntityId)) { throw
CommonException.NotAuthorized(); }`` (Provjera prava pristupa)

Slika 13. Relevantne linije programskog koda prvog cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

ChatGPT 4 ▾

Relevantni odlomci iz koda:

- ``var deck = await deckArchiverRepository.GetById(command.DeckId);`` (Dohvat spila)

Slika 14. Relevantne linije programskog koda drugog cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

ChatGPT 4 ▾

Relevantni odlomci iz koda:

- ``var deckEntity = await DeckEntityDbSet.FindAsync((EncryptedInt)deck.EntityId,
(EncryptedInt)userContext.TenantId);``

Slika 15. Relevantne linije programskog koda trećeg cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

- **Preporuka ChatGPT-a (slika 16.) za unapređenje sigurnosti kontrole programskog koda:**

ChatGPT 4 ▾

```
// U servisu za arhiviranje
public async Task<ArchiveDeckCommandResult> ArchiveDeck(ArchiveDeckCommand command)
{
    // Prvo provjeravamo dozvolu za arhiviranje
    if (!await permissionValidator.CanUpdateDeck(command.DeckId))
    {
        logger.LogWarning($"Unauthorized attempt to archive deck by user {userContext.UserName}");
        throw new Exception("Not authorized to perform this action");
    }

    var deckArchiver = await deckArchiverRepository.GetById(command.DeckId);
    deckArchiver.Archive(systemClock.UtcNow);
    await deckArchiverRepository.UpdateDeckArvhiver(deckArchiver);

    return new ArchiveDeckCommandResult(deckArchiver.EntityId, deckArchiver.CardLifeCycles);
}
```

Slika 16. Kreirani kod od strane UI za A.9 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Preporuka je kreirana kako bi programski kod detaljnije provjerio prava pristupa da samo autorizirani korisnici mogu arhivirati dekove.

2. Revizija zapisa (eng. *Logging and Monitoring* - A.12.)

- **Cilj kontrole A.12 koju je ChatGPT izdvojio:** „Kod mora evidentirati akciju arhiviranja kartice, uključujući podatke kao što su datum, vrijeme i identitet administratora.“ [29]
- **Izdvojene linije programskog koda kojima ChatGPT potkrepljuje zadovoljavanje cilja norme ISO/IEC 27001:**
 - ChatGPT nije pronašao linije koda koje zadovoljavaju zadani cilj, već je kreirao linije programskog koda koje je potrebno implementirati u programski kod.
- **Preporuka ChatGPT-a za zadovoljavanje cilja:**
 - ChatGPT je uočio kako fali važna funkcija u programskom kodu, logiranje aktivnosti. „Detaljno logiranje omogućuje praćenje aktivnosti unutar sustava bilježenjem uspješnih i neuspješnih pokušaja arhiviranja. Implementiranje funkcionalnosti olakšat će se identifikacija ako dođe do sigurnosnih incidenata (slika 17.).“ [29]

- Implementacija:

```

csharp Copy code

var logEntry = new LogEntry
{
    Action = "Archive",
    Entity = "Deck",
    EntityId = deck.Id,
    UserId = userContext.UserId,
    Timestamp = DateTimeOffset.UtcNow,
    Details = $"Deck {deck.Id} archived by {userContext.UserName}"
};
dbContext.LogEntries.Add(logEntry);
await dbContext.SaveChangesAsync();

```

Slika 17. Kreiran kod od strane UI za A.12 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

3. Integritet podataka (eng. *Data Integrity* - A.12.)

- Cilj kontrole integriteta podataka koji je ChatGPT izdvojio: „Kod treba osigurati da se sve povezane kartice pouzdano arhiviraju kada se arhivira kartica.“ [29]
- Izdvojene linije programskog koda kojima ChatGPT potvrđuje ispunjavanje zahtjeva norme ISO/IEC 27001 prikazane su na slici 18.:

Kod:

```

csharp Copy code

public void Archive(DateTimeOffset utcNow)
{
    DeckStatusType = DeckStatusType.Archived;
    ArchiveOnDate = utcNow;

    foreach (var cardLifeCycle in CardLifeCycles)
    {
        cardLifeCycle.Archive(utcNow);
    }
}

```

Slika 18. Relevantne linije koda za A.12 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Nakon izdvojene linije koda ChatGPT zaključuje kako se u kodu koristi „*foreach*“ petlja za arhiviranje svih povezanih kartica unutar deka. Ovo osigurava da su sve povezane kartice arhivirane zajedno s dekom što nam dokazuje kako programski kod zadovoljava postavljeni cilj.

- **Preporuke ChatGPT-a za unapređenje programskog koda:**

“**Dodavanje evidentiranja promjena:** Osigurajte da se sve promjene, uključujući arhiviranje kartica, pravilno evidentiraju za potrebe revizije.” [29] Slika 19. prikazuje preporuku unapređenja programskog koda.

ChatGPT 4 ▾

```
deckEntity.Id.EncryptedInt = deck.Id.EncryptedInt;

foreach(var cardLifeCycle in deck.CardLifeCycles)
{
    var cardEntity = await CardEntityDbSet.FindAsync((EncryptedInt)cardLifeCycle.E
    cardLifeCycleMapper.ConvertToCardEntity(cardLifeCycle, cardEntity);
}

dbContext.SaveChanges();

// Dodavanje revizijskog zapisa
var logEntry = new LogEntry
{
    Action = "Archive",
    Entity = "Deck",
    EntityId = deck.EntityId,
    UserId = userContext.UserId,
    Timestamp = DateTimeOffset.UtcNow
};
dbContext.LogEntries.Add(logEntry);
await dbContext.SaveChangesAsync();
```

Slika 19. Preporuka unapređenja koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Ovaj prijedlog unapređenja ChatGPT je već predložio kod prethodnog cilja, Revizija zapisa gdje je kreirao programske linije samo za unapređenje. U ovom cilju ChatGPT je išao korak dalje i konkretno predložio u zadanom programskom kodu gdje unapređenje treba smjestiti.

- **“Validacija Ulaznih Podataka:** Provjerite da su svi podaci koji se arhiviraju točni i valjani prije nego što se promjena izvrši.” [29]

Nakon kreiranja prijedloga unapređenja od strane umjetne inteligencije, ChatGPT nije dao način na koji bi se zadano unapređenje moglo implementirati. Ovaj prijedlog unapređenja ne daje konkretno rješenje programeru, stoga mu ostavlja veći dio posla kod usklađivanja programskog koda s ciljem norme.

4. Svijest korisnika (eng. *User Awareness* - A.7)

- **Cilj kontrole u svijesti korisnika koji je ChatGPT izdvojio:** „Osigurati da korisnici dobiju jasne i informativne obavijesti o akcijama koje poduzimaju.“[29]

- **Izdvojene linije programskog koda kojima ChatGPT potkrepljuje zadovoljavanje cilja ISO/IEC 27001:** „Prikazivanje dijaloga za potvrdu s obavijesti o broju kartica koje će biti arhivirane nije eksplicitno navedeno u kodu. Implementacija za obavješavanje korisnika nije jasno prikazana. „*NotifyUser*“ je naveden kao potencijalna implementacija, ali nije detaljno opisana.“ [29]
- **Preporuke ChatGPT-a za unapređenje programskog koda:**
 1. **Implementacija Dijaloga za Potvrdu:** Slika 20. prikazuje dijalog za potvrdu s jasnim obavijestima o broju kartica koje će biti arhivirane.

ChatGPT 4 ▾

```

csharp
// Prikazivanje dijaloga za potvrdu
var confirmationMessage = $"Archiving this deck will archive all {deckArchiver.CardLif
bool userConfirmed = ShowConfirmationDialog(confirmationMessage);
if (!userConfirmed)
{
    throw new OperationCanceledException("User canceled the archiving operation.");
}

```

Slika 20. Implementacija dijaloga za potvrdu (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

2. **Implementacija Obavijesti Korisniku:** Slika 21. prikazuje prijedlog ChatGPT-a za dodavanje funkcionalnosti za obavješavanje korisnika nakon uspješnog arhiviranja kartica.

ChatGPT 4 ▾

2. Obavijest Korisniku:

- Obavješavanje korisnika nakon uspješnog arhiviranja:

```

csharp
// Obavijest korisniku
NotifyUser($"Deck has been successfully archived. Total {deckArchiver.CardLifeCycles.Cou

```

Slika 21. Obavijest korisnik nakon uspješnog arhiviranja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

5. Nabava, razvoj i održavanje sustava (eng. *Acquisition, Development, and Maintenance of Information Systems - A.14*)

- Cilj kontrole koju je ChatGPT izdvojio: „Sprječiti SQL injekcije i druge napade validacijom unosa.“ [29]
- Na slici 22. prikazane su izdvojene linije programskog koda kojima ChatGPT potkrepljuje zadovoljavanje cilja norme ISO/IEC 27001:

ChatGPT 4 ▾

```
public async Task<DeckArchiver?> GetById(IntegerId id)
{
    var unarchivedDeckCards = await CardEntityDbSet
        .Include(x => x.CardResources)
        .Where(x => x.DeckId == id.Value && x.CardStatusType != CardStatusType.Archived)
        .ToListAsync();

    var deck = await DeckEntityDbSet
        .DefaultIfEmpty()
        .SingleOrDefaultAsync(x => x.Id == id);

    var deckWithUnarchivedCards = new DeckArchiver(
        deck.Id,
        deck.DeckStatusType,
        deck.ArchiveOnDate,
        unarchivedDeckCards.Select(x => cardLifeCycleMapper.ConvertToCardLifeCycle(x)).ToL
    );

    return deckWithUnarchivedCards;
}
```

Slika 22. Dio programskog koda PK1 koji je potkrijepljen zadovoljavanje norme ISO/IEC 27001

- **Slika 23.** prikazuje preporuku ChatGPT-a za unapređenje dodavanjem validacije unosa u programski kod:

```

public async Task<DeckArchiver?> GetById(IntegerId id)
{
    ValidateInput(id);
    var unarchivedDeckCards = await CardEntityDbSet
        .Include(x => x.CardResources)
        .Where(x => x.DeckId == id.Value && x.CardStatusType != CardStatusType.Archived)
        .ToListAsync();

    var deck = await DeckEntityDbSet
        .DefaultIfEmpty()
        .SingleOrDefaultAsync(x => x.Id == id);

    var deckWithUnarchivedCards = new DeckArchiver(
        deck.Id,
        deck.DeckStatusType,
        deck.ArchiveOnDate,
        unarchivedDeckCards.Select(x => cardLifeCycleMapper.ConvertToCardLifeCycle(x)).ToL
    );

    return deckWithUnarchivedCards;
}

private void ValidateInput(IntegerId id)
{
    if (id == null || id.Value <= 0)
    {
        throw new ArgumentException("Invalid ID value.");
    }
}

```

Slika 23. Preporuka ChatGPT-a za unapređenje dodavanjem validacije u programski kod PK1 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

„Validacija unosa je ključna za sprječavanje SQL injekcija i drugih napada koji se temelje na manipulaciji ulaznim podacima. „ [29]

Na temelju pregleda programskog koda i analize pomoću ChatGPT alata dolazi se do sljedećeg zaključka: „

- **Kontrola pristupa:** Implementirana je provjera prava pristupa, što osigurava da samo ovlašteni korisnici mogu mijenjati kartice.
- **Revizijski zapisnik:** Dodavanjem zapisa aktivnosti u kod, osigurano je evidentiranje svih važnih događaja za potrebe revizije.
- **Integritet podataka:** Osigurano je pouzdano arhiviranje deкова zajedno s poveznim karticama.
- **Svijest korisnika:** Implementacija dijaloga za potvrdu i obavijesti korisnicima unapređuje svijest korisnika o akcijama koje poduzimaju.

- **Nabava, razvoj i održavanje sustava:** Dodana je funkcija ValidateInput za sprječavanje SQL injekcija.“ [29]

Dodatno, analizom unutar slučaja upotrebe zaključeno je da programski kod barem djelomično osigurava integritet podataka. Nije moguće utvrditi da programski kod u potpunosti osigurava sve ciljeve unutar norme ISO/IEC 27001 zato što ChatGPT svojom analizom izdvaja i analizira samo neke manje ciljeve norme. ChatGPT ne može u potpunosti zamijeniti ljudsko znanje, iskustvo i stručnost; osobito kada se radi o složenijim sigurnosnim provjerama. Iako nije proveo u potpunosti analizu svih ciljeva, na temelju preporuka ChatGPT-a došlo do bitnih unapređenja u programskom kodu.

5.4. Upotreba umjetne inteligencije u analizi testiranja programskog koda i jedinice testa u skladu s normom ISO/IEC/IEEE 29119

Kako bi osigurali da programski kod bude u skladu s normom ISO/IEC/IEEE 29119 za testiranje softvera, cilj je proći kroz nekoliko scenarija slučaja upotrebe koji će pokriti različite aspekte osiguravanja kvalitete s pomoću umjetne inteligencije, ChatGPT-a.

5.4.1. Slučaj upotrebe 5: Upotreba umjetne inteligencije za kreiranje jedinica testova u skladu s normom ISO/IEC/IEEE 29119

Za ovaj slučaj upotrebe korištena je umjetna inteligencija kojoj je zadatak kreirati jedinice testova uz poštivanje smjernica iz norme ISO/IEC/IEEE 29119. Umjetnoj inteligenciji, ChatGPT-u je dan kao ulazni podatak poslovni zadatak i programski kod. Od ChatGPT-a se očekuje da kreira jedinice testova koje će pokriti cijeli programski kod i testirati njegove funkcionalnosti.

Odgovor je ChatGPT-a kreirao u obliku specifikacije testnog slučaja:

„Naziv testnog slučaja: Testni slučaj za funkcionalnost

DeckArchiverRepository.GetById(IntegerId id)

Prva jedinica test: Testiranje dohvata deka s karticama koje nisu arhivirane.

- **Opis jedinice testa:** Provjerite da li su vraćene kartice odgovarajućeg statusa i da li pripadaju zadanom deku.
- **Ulazni podaci:** Valjani IntegerId za postojeći dek.
- **Očekivani rezultat:** Metoda vraća instancu DeckArchiver koja sadrži ispravan popis kartica koje nisu arhivirane.

Druga jedinica testa: Testiranje dohvata deka kada nema niti jedne kartice koja nije arhivirana.

- **Opis jedinice testa:** Provjerite je li lista kartica prazna.
- **Ulazni podaci:** IntegerId za dek gdje su sve kartice arhivirane.
- **Očekivani rezultat:** Metoda vraća instancu DeckArchiver bez kartica.

Treći jedinica testa: Testiranje dohvata deka s nevaljanom ili nepostojećom ID vrijednošću.

- **Opis jedinice testa:** Provjerite je li izuzetak ispravno bačen.
- **Ulazni podaci:** Nevaljani IntegerId.
- **Očekivani rezultat:** Metoda baca izuzetak koji ukazuje na nepostojeći dek.“ [29]

ChatGPT implementira jedinice testa:

```
using NUnit.Framework;
using System.Threading.Tasks;
using Moq;
using Microsoft.EntityFrameworkCore;
using Recall.DeckManagement.Repositories.Implementations;
using Recall.Shared.MsSql.Context;
using Recall.Shared.Framework.Security.Identity;
using System.Linq;
using Recall.DeckManagement.Database.Entities;
using Recall.DeckManagement.Repositories.EntityMappers;
using System;
```

```

namespace Recall.Tests
{
    [TestFixture]
    public class DeckArchiverRepositoryTests
    {
        private Mock<RecallDbContext> _dbContextMock;
        private Mock<IUserContext> _userContextMock;
        private CardLifeCycleMapper _cardLifeCycleMapper;
        private DeckArchiverRepository _repository;

        [SetUp]
        public void Setup()
        {
            _dbContextMock = new Mock<RecallDbContext>(new
DbContextOptions<RecallDbContext>());
            _userContextMock = new Mock<IUserContext>();
            _cardLifeCycleMapper = new CardLifeCycleMapper();
            _repository = new DeckArchiverRepository(_dbContextMock.Object,
_userContextMock.Object, _cardLifeCycleMapper);

            // In-memory data setup
            var decks = new List<DeckEntity>
            {
                new DeckEntity { Id = 1, DeckStatusType =
DeckStatusType.Active },
                new DeckEntity { Id = 2, DeckStatusType =
DeckStatusType.Archived }
            };
        }
    }
}

```

```

var cards = new List<CardEntity>
{
    new CardEntity { DeckId = 1, CardStatusType =
CardStatusType.Active, CardResources = new List<CardResource>() },
    new CardEntity { DeckId = 2, CardStatusType =
CardStatusType.Archived, CardResources = new List<CardResource>() }
};

_dbContextMock.Setup(x =>
x.Set<DeckEntity>()).Returns(DbContextMock.GetQueryableMockDbSet (decks));
_dbContextMock.Setup(x =>
x.Set<CardEntity>()).Returns (DbContextMock.GetQueryableMockDbSet (cards));
}

[Test]

public async Task
GetById_WithNonArchivedCards_ReturnsDeckWithCards()
{
    // Act

    var result = await _repository.GetById(new IntegerId(1));

    // Assert

    Assert.IsNotNull(result);

    Assert.AreEqual(1, result.CardLifeCycles.Count);

    Assert.AreEqual(DeckStatusType.Active, result.DeckStatusType);
}

[Test]

public async Task GetById_WithAllCardsArchived_ReturnsEmptyDeck()
{
    // Act

```

```

        var result = await _repository.GetById(new IntegerId(2));

        // Assert

        Assert.IsNotNull(result);

        Assert.IsEmpty(result.CardLifeCycles);
    }

    [Test]
    public void GetById_WithInvalidId_ThrowsNotFoundException()
    {
        // Arrange

        var invalidId = new IntegerId(99);

        // Act & Assert

        var ex = Assert.ThrowsAsync<NotFoundException>(() =>
            _repository.GetById(invalidId));

        Assert.That(ex.Message, Is.EqualTo("Deck not found"));
    }
}

```

Implementacijom jediničnih testova u skladu s normom ISO/IEC/IEEE 29119 ChatGPT pokazao je svoju sposobnost kreiranja jediničnih testova u vrlo zavidnom roku. Ovaj pristup osigurava razvojnom timu smanjenje vremena koje bi inače bilo utrošeno za kreiranje jediničnih testova, dok se istodobno osigurava usklađivanje s relevantnim normama. Ovo je velik korak za implementaciju programskih proizvoda jer ušteda vremena kod kreiranja jediničnih testova omogućava programeru više vremena za razvoj ključnih dijelova programskog proizvoda. Važno je naglasiti da i dalje postoji potreba za ljudskom intervencijom tj. provjerom i prilagodbom dobivenih rezultata jer umjetna inteligencija trenutno je ograničena dubinom razumijevanja poslovnih zahtjeva i složenih logičkih interakcija unutar programskog

koda. Ipak, činjenica da je umjetna inteligencija sposobna implementirati jedinice testova predstavlja značajan napredak u svijetu razvoja programskih proizvoda.

5.4.2.Slučaj upotrebe 6: Analiza pokrivenosti jedinice testa JT1 za programski kod PK1 pomoću umjetne inteligencije

U ovom kontekstu slučaja upotrebe svrha umjetne inteligencije je ispitati pokrivaju li jedinice testa JT1 sve dijelove programskog kod PK1. Kako bi ispitali navedenu tezu kao ulazni parametri za umjetnu inteligenciju su poslovni zahtjev, programski kod i jedinice testa kreirani od strane programera. Rezultat ChatGPT-a pomaže u otkrivanju je li programer pokrio sve testove za testiranje programskog kod ili je potrebno još koju jedinicu testa provesti.

Uvodom rečenicom u odgovoru ChatGPT-a vidljivo je kako je obuhvaćeno nekoliko scenarija funkcionalnost. ChatGPT detaljnije razmatra dostavljene jedinice testa s programskim kodom.

Povratna informacija ChatGPT-a na istraživanje je sljedeće:

“Provedene jedinice testa pokrivaju nekoliko ključnih scenarija za funkcionalnost *"ArchiveDeck"*, ali da bismo utvrdili pružaju li potpunu pokrivenost, potrebno je provjeriti jesu li obuhvaćeni svi mogući putovi i uvjeti u kodu koji si mi poslao. Razmotrimo to detaljnije:

1. Korisnička dopuštenja:

- **Test 1:** Provjerava ispravno generiranje iznimke ako korisnik nema dopuštenje za arhiviranje deka. Ovo pokriva provjeru unutar `CanArchiveDeckCommandValidator` za `CanUpdateDeck`.

2. Nepostojeći dek:

- **Test 2:** Osigurava da se generira iznimka kada se pokuša arhivirati dek koji ne postoji. Ovo pokriva scenarij `CommonException.NotFound()`.

3. Dek je već arhiviran:

- **Test 3:** Provjerava da se generira iznimka kada se pokuša arhivirati dek koji je već arhiviran. Ovo pokriva provjeru statusa deka unutar metode `CanArchiveDeckCommandValidator` gdje se provjerava je li dek aktivan.

4. Arhiviranje deka:

- **Test 4:** Provjerava ispravno arhiviranje deka kada su svi uvjeti zadovoljeni (dek je aktivan, korisnik ima dopuštenje, postoje kartice koje treba arhivirati). Također, provjerava da se ispravno ažurira status deka i kartica.

Pokrivenost:

- **Korisnička dopuštenja** su adekvatno pokrivena.
- **Provjere postojanja deka** su pokrivena.
- **Status deka (aktivan/arhiviran)** je pokriven.
- **Ažuriranje statusa deka i kartica** tijekom arhiviranja je pokriveno.

Mogući nedostaci:

- **Validacija za prazne ili nevažeće ulazne podatke (npr. neispravan ID deka u naredbi)** nije eksplicitno testirana.
- **Provjere prilikom ažuriranja u bazi podataka:** Nije testirano kako se sustav ponaša ako dođe do pogreške prilikom spremanja promjena u bazu podataka (SaveChanges).” [29]

ChatGPT zaključuje kako testovi pokrivaju ključne funkcionalnosti arhiviranja deka, ali bi se dodatnim testovima mogla osigurati potpuna pokrivenost svih mogućih uvjeta i scenarija u kodu.

Budući da je umjetna inteligencija prepoznala nedostatke, ChatGPT donosi i prijedlog unapređenja jedinice testa. U nastavku, prikazani su prijedlozi ChatGPT-a: “

1. Pokriti više rubnih slučajeva:

- **Neispravan ID deka:** Testiraj što se događa kada se proslijedi neispravan ID deka (npr. negativan broj ili nula).
- **Prazan popis kartica:** Provjeri kako se sustav ponaša ako dek nema nijednu karticu.

2. Provjera iznimki:

- Dodaj test koji provjerava što se događa ako dođe do iznimke prilikom ažuriranja u bazi podataka (npr. greška prilikom poziva SaveChanges). Ovo može pomoći u identificiranju potencijalnih problema s trajnošću podataka.

3. Validacija inputa:

- Testiraj scenarije s nevažećim ulaznim podacima, poput praznog ili null ID-a deka u ArchiveDeckCommand.

4. Mocking složenih objekata:

- Razmisli o korištenju mocking alata kako bi se provjerilo kako sustav reagira na različite situacije, kao što su povrat vrijednosti null iz repozitorija ili neuspješne validacije dozvola.

5. Performance test:

- Ako je moguće, dodaj test koji provjerava performanse funkcije kada dek ima veliki broj kartica. Ovo može biti korisno za identifikaciju potencijalnih problema s performansama u stvarnim uvjetima.

6. Testovi za provjeru sigurnosti:

- Dodaj testove koji osiguravaju da se ne može arhivirati dek bez odgovarajućih sigurnosnih provjera, čak i ako se pokušaju zaobići standardne metode (npr. direktnim pozivom metode repozitorija).

7. Testovi za više korisnika:

- Dodaj testove koji simuliraju scenarije s više korisnika kako bi se osiguralo da različiti korisnici ne mogu utjecati na dekove za koje nemaju prava pristupa.

8. Detaljnija provjera povratnih vrijednosti:

- Trenutno se provjerava samo ID deka i status, ali moglo bi biti korisno provjeriti i druge povratne vrijednosti ili stanja sustava nakon arhiviranja, kao što su promjene u dnevnicima aktivnosti ili u drugim dijelovima sustava.

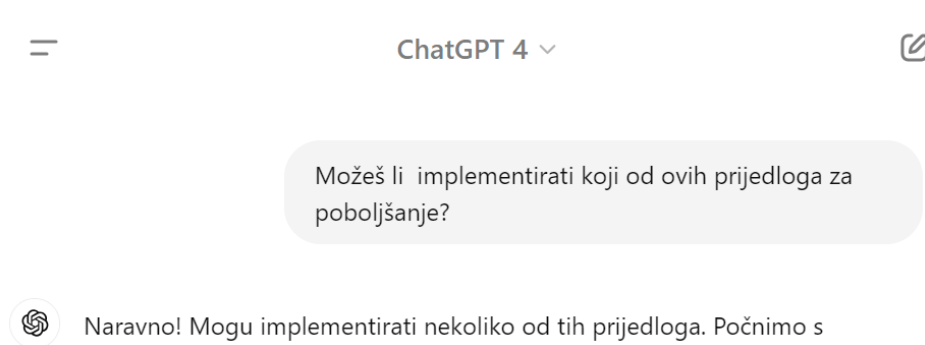
9. Koristiti fluent API za bolju čitljivost:

- Već koristiš FluentAssertions, što je odlično. Razmisli o daljnjem korištenju fluent API-ja za dodatne provjere koje bi se unaprijedila čitljivost testova.

10. Automatizirano pokretanje testova:

- Osiguraj da se svi testovi automatski pokreću kao dio CI/CD procesa kako bi se odmah uočile bilo kakve regresije.” [29]

Iako unapređenja koje ChatGPT navodi ne obuhvaćaju konkretnu implementaciju, isti može kreirati kod koji je moguće implementirati nakon dodatnog upita: “Možeš li implementirati koji od ovih prijedloga za poboljšanje?” Slika 24. prikazuje kako izgleda kreirani upit za ChatGPT:



Slika 24. Upit ChatGPT-a može li implementirati unapređenje (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

U nastavku nalazi se implementacija testova ChatGPT-a:

1. Test za neispravan ID deka

Test za neispravan id deka, prikazan na slici 25., provjerava što se događa kada se proslijedi neispravan ID deka (npr. negativan broj).

```

[Scenario("Invalid Deck ID")]
public void ArchiveDeck_InvalidDeckId_ShouldThrowException()
{
    var invalidDeckId = -1;
    command = new ArchiveDeckCommand(IntegerId.FromInt(invalidDeckId));

    Func<Task> act = async () => await deckArchiverService.ArchiveDeck(command);

    act.Should().Throw<ArgumentException>().WithMessage("Invalid Deck ID");
}

```

Slika 25. Provjera testa (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

2. Test za provjeru iznimki prilikom SaveChanges

Test na slici 26. simulira situaciju gdje se dogodi greška prilikom spremanja promjena u bazu podataka.

```

[Scenario("Database Save Error")]
public async Task ArchiveDeck_SaveChangesFails_ShouldThrowException()
{
    // Arrange
    GivenIHaveADeckToArchive(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenIHaveAnArchiveDeckCommand(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenTheDeckIsActive();
    GivenThePermissionToArchiveADeckIs(true);

    // Mock the SaveChanges method to throw an exception
    mock.Mock<RecallDbContext>()
        .Setup(x => x.SaveChanges())
        .Throws(new Exception("Database error"));

    // Act
    Func<Task> act = async () => await WhenITryToArchiveADeck();

    // Assert
    await act.Should().ThrowAsync<Exception>().WithMessage("Database error");
}

```

Slika 26. Simulacija situacije gdje se dogodi greška prilikom spremanja promjena u bazu podataka (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

3. Test za prazni popis kartica

Test za prazni popis kartica, na slici 27., provjerava kako se sustav ponaša ako dek nema nijednu karticu.

```
[Scenario("Empty Deck")]
public async Task ArchiveDeck_EmptyDeck_ShouldArchiveSuccessfully()
{
    // Arrange
    GivenIHaveADeckToArchive(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenIHaveAnArchiveDeckCommand(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenTheDeckIsActive();
    GivenThePermissionToArchiveADeckIs(true);

    // Ensure the deck has no cards
    deck.CardLifeCycles = new List<CardLifeCycle>();

    // Act
    await WhenITryToArchiveADeck();

    // Assert
    commandResult!.NumberOfArchivedCards.Should().Be(0);
    commandResult!.Status.Should().Be(DeckStatusType.Archived);
}
```

Slika 27. Provjera ponašanja koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Analiza umjetne inteligencije, ChatGPT-a pokazala je kako umjetna inteligencija može igrati ključnu ulogu u osiguravanju potpunog pokrivanja programskog koda jedinicama testiranja. ChatGPT demonstrirao je sposobnost umjetne inteligencije, a to je detaljno analizirati postojeće programske testove JT-1 i identificirati potencijalne nedostatke u kodu u skladu s definiranim parametrima testa. Preporuke koje je ChatGPT iznio značajno doprinose kvaliteti i pouzdanosti programskog koda; te olakšava proces testiranja omogućujući brže otkrivanje nedostataka. Također, brže otkrivanje nedostataka i pružanje razvojnom timu smjernice za unapređenje doprinosi sigurnijem i stabilnije softverskom rješenju. Ovim slučajem upotrebe jasno je pokazano koliko je ChatGPT moćan alat i koliko može doprinijeti u radu programera.

5.4.3.Slučaj upotrebe 7: Provjera usklađenosti jedinice testa s normom ISO/IEC/IEE 29119 pomoću umjetne inteligencije

Kao i za prethodni slučaj upotrebe, za ulazne podatke ovog slučaja upotrebe korišteni su poslovni zahtjev, programski kod i jedinice testa. Od umjetne inteligencije, ChatGPT-a, očekuje se detaljna analiza jesu li jedinice testa napisane u skladu s normom ISO/IEC/IEE 29119. Osim provjere u kojoj su mjeri jedinice testa napisane u skladu s normom ISO 29119, cilj je od umjetne inteligencije dobiti odgovor na koji način se jedinice testa mogu unaprijediti kako bi se dodatno osigurala potpuna usklađenost s normom ISO/IEC/IEEE 29119. ChatGPT prilikom provjere uzima nekoliko ključnih aspekata norme ISO/IEC/IEE 29119, a to su:

- Test Planiranje i Dokumentacija
- Test Dizajn
- Test Izvršenje
- Testni Rezultati i Izvještavanje
- Upravljanje Rizicima i Praćenje Grešaka

Već na samom početku, vidljivo je kako ChatGPT dobro rukuje s ključnim elementima norme ISO 29119.

U ostatku odgovora, ChatGPT izlaže sljedeća zapažanja o jedinici testa:

„Test Planiranje i Dokumentacija: Testovi su dobro opisani, ali bi bilo korisno uključiti dodatne informacije, poput jasnih očekivanih rezultata u dokumentaciji. Također, svaka testna metoda bi trebala imati jasnu dokumentaciju koja navodi što test provjerava i koji su kriteriji za prolaz.

Test Dizajn: Testovi su dizajnirani s jasnim ciljem, ali bi mogli biti razrađeni dodatnim testnim slučajevima koji pokrivaju rubne slučajeve (što smo djelomično već učinili u prethodnim prijedlozima). Svaki test provjerava jednu funkcionalnost, što je u skladu s najboljim praksama.

Test Izvršenje: Testovi se izvršavaju u izolaciji koristeći mocking, što je u skladu s ISO standardom.

Testni Rezultati i Izvještavanje: Rezultati testova se bilježe, ali bi moglo biti korisno generirati detaljnije izvještaje koji uključuju informacije o svim izvršenim testovima, pogotovo u kontekstu sustavnog testiranja na višim razinama (npr. integracijsko testiranje).

Upravljanje Rizicima i Praćenje Grešaka: Ova komponenta nije direktno pokrivena u jediničnim testovima, ali je važna na višoj razini testiranja i upravljanja testnim procesima.“ [29]

Za zaključak postavljene hipoteze ChatGPT iznosi činjenicu da jedinice testova prate dobar dio preporuka i najboljih praksi koje su u skladu s ISO/IEC/IEE 29119, pogotovo u kontekstu test dizajna i izvršenja. Međutim, kako bi se u potpunosti slijedio standard, bilo bi korisno uvesti dodatnu dokumentaciju, planiranje, te sistematičnije izvještavanje i praćenje rizika. U suštini, testovi su na dobrom putu, ali bi dodatno formaliziranje procesa prema standardu osiguralo potpunu usklađenost.

Uz detaljnu analizu umjetne inteligencije postavlja se i dodatno pitanje mogu li se unaprijediti jedinice testa kako bi bile u potpunosti u skladu s ISO 29119?

ChatGPT iznosi prosudbu u kojoj navodi da će unapređenje uključivati: dodavanje novih jedinice testova za rubne slučajeve, unapređenje dokumentacije te dodavanje automatiziranog izvještavanja i upravljanja rizicima.

U nastavku, prikazani su načini kako umjetna inteligencija, ChatGPT smatra da se mogu provesti unapređenja: „

1. Unapređenje dokumentacije testova

Počnimo s dodavanjem detaljne dokumentacije svakom testnom slučaju. Slika 28. prikazuje primjer unapređenja dokumentacije testova.

Originalni test: Invalid Deck ID

```
csharp Copy code

/// <summary>
/// Test provjerava ponašanje sustava kada se pokuša arhivirati dek s neispravnim ID-om.
/// Očekivani rezultat: Generira se iznimka ArgumentException s porukom "Invalid Deck ID".
/// Test ID: UT-001
/// </summary>
[Scenario("Invalid Deck ID")]
public void ArchiveDeck_InvalidDeckId_ShouldThrowException()
{
    var invalidDeckId = -1;
    command = new ArchiveDeckCommand(IntegerId.FromInt(invalidDeckId));

    Func<Task> act = async () => await deckArchiverService.ArchiveDeck(command);

    act.Should().Throw<ArgumentException>().WithMessage("Invalid Deck ID");
}
}
```

Slika 28. Unapređenje dokumentacije testova (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

2. Formaliziranje testnih scenarija i planiranja

Dodao bi formalizaciju i kreirao testni plan s jasnim identifikatorima testnih slučajeva. To uključuje sljedeće korake prikazane na slici 29.:

Dodaj unikatne identifikatore za testove:

```
csharp Copy code

/// <summary>
/// Test ID: UT-001
/// Test provjerava ponašanje sustava kada se pokuša arhivirati dek s neispravnim ID-om.
/// Očekivani rezultat: Generira se iznimka ArgumentException s porukom "Invalid Deck ID".
/// </summary>
[Scenario("Invalid Deck ID")]
public void ArchiveDeck_InvalidDeckId_ShouldThrowException()
{
    var invalidDeckId = -1;
    command = new ArchiveDeckCommand(IntegerId.FromInt(invalidDeckId));

    Func<Task> act = async () => await deckArchiverService.ArchiveDeck(command);

    act.Should().Throw<ArgumentException>().WithMessage("Invalid Deck ID");
}
}
```

Slika 29. Dodavanje identifikatora za testove (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

3. Razrađivanje pokrivenosti testova

Dodao bi testove koji pokrivaju rubne slučajeve, kao što su greške prilikom spremanja promjena u bazu podataka i prazni popis kartica. Slika 30. prikazuje dodavanje testova koji pokrivaju rubne slučajeve.

```
Test: Database Save Error

csharp Copy code

/// <summary>
/// Test ID: UT-002
/// Test provjerava ponašanje sustava kada dođe do greške prilikom spremanja promjena u ba
/// Očekivani rezultat: Generira se iznimka Exception s porukom "Database error".
/// </summary>
[Scenario("Database Save Error")]
public async Task ArchiveDeck_SaveChangesFails_ShouldThrowException()
{
    // Arrange
    GivenIHaveADeckToArchive(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenIHaveAnArchiveDeckCommand(new Table("DeckId") { new TableRow { ["DeckId"] = "1" }
    GivenTheDeckIsActive();
    GivenThePermissionToArchiveADeckIs(true);

    // Mock the SaveChanges method to throw an exception
    mock.Mock<RecallDbContext>()
        .Setup(x => x.SaveChanges())
        .Throws(new Exception("Database error"));

    // Act
    Func<Task> act = async () => await WhenITryToArchiveADeck();

    // Assert
    await act.Should().ThrowAsync<Except ( ).WithMessage("Database error");
}
```

Slika 30. Dodavanje testova koji pokrivaju rubne slučajeve (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

4. Automatizacija testiranja i izvještavanja

Za automatizirano praćenje rezultata i generiranje izvještaja, mogli bi se uvesti dodatni alati i postupci, no to je često povezano s infrastrukturom CI/CD (Continuous Integration/Continuous Delivery). Na razini koda, može se osigurati da testovi bilježe sve potrebne podatke. Slika 31. prikazuje primjer automatiziranog bilježenja rezultata:

Evo primjer automatiziranog bilježenja rezultata:

```
csharp Copy code

/// <summary>
/// Test ID: UT-004
/// Test provjerava arhiviranje deka s karticama.
/// Očekivani rezultat: Svi kartice u deku trebaju biti arhivirane.
/// </summary>
[Scenario("Archive Deck with Cards")]
public async Task ArchiveDeck_WithCards_ShouldArchiveSuccessfully()
{
    // Arrange
    GivenIHaveADeckToArchive(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenIHaveAnArchiveDeckCommand(new Table("DeckId") { new TableRow { ["DeckId"] = "1" } });
    GivenTheDeckIsActive();
    GivenThePermissionToArchiveADeckIs(true);
    GivenTheDeckHasUnarchivedCards(new Table("CardId", "Status")
    {
        new TableRow { ["CardId"] = "1", ["Status"] = "Draft" },
        new TableRow { ["CardId"] = "2", ["Status"] = "Published" }
    });

    // Act
    await WhenITryToArchiveADeck();

    // Assert
    ThenIShouldGetArchiveDeckCommandResultWithCorrectDeckIdAndStatus();
    ThenAllUnarchivedCardsShouldBeArchived();

    // Log results (for illustrative purposes, replace with your logging mechanism)
    Console.WriteLine($"Test ID: UT-004 | Deck ID: {commandResult!.DeckId} | Status: {comm
}
}
```

Slika 31. Dodavanje automatizacije (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

5. Upravljanje rizicima

Upravljanje rizicima može uključivati identificiranje i praćenje svih potencijalnih rizika povezanih s arhiviranjem deka, kao što su gubitak podataka, neispravno arhiviranje, itd. Za to je potrebna dublja integracija s alatima za praćenje grešaka i izvještavanje unutar softverskog projekta.

Na primjer, u CI/CD sustavu moglo bi se postaviti praćenje rizika za svaki test koji ne uspije.“ [29] Slika 32. prikazuje postavljanje za praćenje rizika.

```
if (actualException != null)
{
    // Log risk in the test report
    RiskLogger.Log($"Rizik: {actualException.Message} | Test ID: UT-004");
}
```

Slika 32. Postavljanje praćenja rizika (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)

Umjetna inteligencija kroz unapređenje jedinica testa pružila je bolju dokumentaciju, proširila je pokrivenost rubnih slučajeva te ponudila primjer kako bi izgledalo automatizirano izvještavanje. Također je pružila smjernice za bolje upravljanje rizicima u procesu testiranja. Sve ovo zajedno doprinosi usklađenosti s ISO/IEC/IEE 29119 normom.

Ovim slučajem upotrebe prikazan je način na koji umjetna inteligencija može rješavati izazove u softverskom inženjerstvu. Primjer nam pruža uvid u to kako umjetna inteligencija, ChatGPT može analizirati jedinice testa, pomoći, ali i asistirati u unapređenju procesa pisanja, testiranja i dokumentiranja testova kako bi jedinice testa bile u skladu sa normom ISO/IEC/IEE 29119. Iz detaljnih odgovora umjetne inteligencije kroz razgovor, uočljivo je kako umjetna inteligencija ima veliki potencijal prilikom rješavanje ovakvih kompleksih zadataka.

5.4.4. Slučaj upotrebe 8: Usporedba testiranja funkcionalnosti arhiviranja dekova između testova razvijenih od strane programera i umjetne inteligencije

U ovom slučaju upotrebe 8 fokus je na analiziranju usporedbe testova generiranih s pomoću ChatGPT i testova razvijenih tradicionalnim pristupom od strane programera. U slučaju upotrebe 5, ChatGPT je za programski kod kreirao jedinice testa. Implementirane jedinice testa iz slučaja upotrebe 5 i jedinice testa - JT1 koje je kreirao programer korišteni su za ulazne podatke ChatGPT-a kako bi ih usporedio i dao zaključak o kvaliteti istih. Cilj je identificirati tri ključne stavke:

1. ključne razlike i prednosti
2. potencijalne nedostatke oba pristupa
3. pružiti preporuke za integraciju i unapređenje procesa testiranja.

Prednosti jedinice testova implementiranih ChatGPT-em za automatizirano testiranje koje ChatGPT navodi su: „

- **Pokrivenost:** UI sustav generira veći broj testova, uključujući dodatne scenarije za granični slučaj (eng. *edge caseove*), čime osigurava sveobuhvatnu pokrivenost.
- **Kvaliteta poruka o greškama:** UI sustav generira detaljnije poruke o greškama, što olakšava dijagnostiku i ispravljanje problema.
- **Vrijeme razvoja:** UI sustav značajno smanjuje vrijeme potrebno za razvoj testova.
- **Otpornost:** UI sustav pokazuje veću otpornost na greške i bolju podršku za simultane pristupe.“ [29]

S druge strane, kao ključne prednosti testova razvijenih od strane developera ChatGPT ističe:“

- **Kontrola i prilagodba:** Developeri imaju potpunu kontrolu nad testovima i mogu ih prilagoditi specifičnim zahtjevima projekta.
- **Specifično znanje o domeni:** Developeri imaju detaljno znanje o aplikaciji i mogu dizajnirati testove koji ciljaju na specifične funkcionalnosti i potencijalne probleme.“ [29]

Isto tako, ChatGPT navodi i sličnosti provedenih jedinica testova: „

- Oba seta testova pokrivaju iste scenarije: korisnik bez prava na ažuriranje, deka koji ne postoji, deka koji je već arhiviran, i uspješno arhiviranje deka.
- Oba seta testova provjeravaju ispravnost izuzetaka i rezultata funkcionalnosti arhiviranja deka.
- Testovi uključuju provjere prava korisnika i stanja deka.“ [29]

Glavne tri razlike koje ChatGPT navodi između jedinica testova kreiranih od strane programera i umjetne inteligencije, ChatGPT-a: „

- **Alati:** Prvi set testova koristi NUnit i Moq za mockanje i testiranje, dok drugi set koristi SpecFlow za opisivanje scenarija i FluentAssertions za provjeru rezultata.
- **Struktura:** Drugi set testova koristi Given-When-Then strukturu iz SpecFlow-a, što omogućava lakše razumijevanje testnih scenarija, posebno za ne-tehničke osobe.
- **Mockanje:** U drugom setu se koristi AutoMock za automatsko generiranje mockova, dok se u prvom setu ručno postavljaju mockovi.“ [29]

Preporuke koje ChatGPT ističe: „

1. **Integracija UI i ručno razvijenih testova:** Kombinirati prednosti oba pristupa kako bi se osigurala sveobuhvatna pokrivenost i otpornost testova. UI sustav može generirati osnovne i granične slučajeve, dok developeri mogu prilagoditi testove specifičnim zahtjevima projekta.
2. **Kontinuirano unapređenje:** Prikupljati povratne informacije od korisnika i kontinuirano ažurirati testne slučajeve kako bi se osigurala relevantnost i prilagodljivost promjenama u zahtjevima.“ [29]

Budući da su jedinični testovi kreirani različitim pristupima, ChatGPT napravio je procjenu između ta dva korištena pristupa (NUnit i SpecFlow): „

Pristup s NUnit-om:

1. **Prednosti:**
 - Jednostavnost: Jasno je kako su testovi strukturirani i lako se prate.
 - Automatizacija: NUnit se lako integrira s CI/CD alatima za automatizaciju testiranja.
 - Dobra podrška za mockanje: Moq omogućava jednostavno mockanje ovisnosti.
2. **Nedostaci:**
 - Manje čitljiv za ne-tehničke osobe.
 - Potrebno je ručno postavljanje i upravljanje mockovima.

Pristup sa SpecFlow-om:

1. **Prednosti:**
 - Čitljivost: Given-When-Then struktura je intuitivna i laka za razumijevanje, čak i za ne-tehničke osobe.
 - BDD pristup: SpecFlow olakšava suradnju između tehničkih i ne-tehničkih timova.
 - Automatizacija: SpecFlow se također lako integrira s CI/CD alatima.
2. **Nedostaci:**
 - Potrebno je više postavljanja i održavanja alata.
 - Može biti složeniji za početnike.

Razlozi za odabir SpecFlow:

1. **Strukturirani scenariji:** Given-When-Then scenariji su strukturirani i lako razumljivi, što je u skladu s ISO smjernicama za jasnoću i održivost.
2. **Bolja komunikacija:** BDD pristup omogućava bolju komunikaciju između različitih timova, što je važno za osiguravanje da su svi zahtjevi pravilno pokriveni.
3. **Dokumentacija:** Scenariji napisani u SpecFlow-u služe kao živa dokumentacija koja je lako dostupna i razumljiva svim sudionicima.“ [29]

Nakon provedene detaljne analize usporedbe pristupa, ChatGPT zaključuje da izrada jedinica pomoću SpecFlow-a može biti bolja opcija za poštivanje norme ISO/IEC/IEEE 29119 zbog svoje čitljivosti, strukturiranog pristupa i mogućnosti bolje komunikacije među timovima. Međutim, oba pristupa (NUnit i SpecFlow) mogu biti pravilno implementirana u skladu sa normom ako se poštuju osnovni principi testiranja softvera.

Ovaj slučaj upotrebe pruža sveobuhvatan i detaljan prikaz kako se umjetna inteligencija može koristiti za unapređenje procesa testiranja. Također, usporedba s tradicionalnim pristupom testiranja koji provode programeri nudi vrijedne uvide i konkretne preporuke za praksu, a može poslužiti kao temelj daljnja istraživanja i unapređenje testnih procesa.

Rezultati ovog istraživanja imaju potencijalnu primjenu u različitim razvojnim kontekstima te mogu poslužiti kao smjernice za uvođenje alata umjetne inteligencije u normatizaciju procesa testiranja softvera. Slučaj upotrebe dokazao je da ChatGPT ima sposobnost detaljno usporediti jedinice testa, pružajući pritom precizna obrazloženja i analize. Ovdje se naglašava važnost daljnjeg razvoja alata umjetne inteligencije u različitim fazama razvoja softvera, čine se otvara prostor unapređenja kvalitete i efikasnosti u programskom inženjerstvu.

6. Zaključak

Ovaj rad pokazuje koji su benefiti i potencijalni rizici korištenja umjetne inteligencije pri razvoju programskog koda.

Uzimajući u obzir zaključke i teze ovog rada programerima se otvara nova perspektiva; jedan potpuno drukčiji pogled na način razvijanja softvera kao i velika pomoć za bolje korištenje raznih alata prilikom implementacije programskog koda.

Umjetna inteligencija može biti značajan asistent kojeg se treba koristiti u situacijama kada programiranje treba promatrati u širem kontekstu. To je vidljivo iz primjera normi kojima sam se koristila u ovom radu. One su izrazito bitan dio osiguravanja kvalitete, a programeri često nisu detaljno upoznati s njima niti su osviješteni o njihovoj važnosti.

Na temelju provedenih slučajeva upotrebe, zaključuje se da umjetna inteligencija, posebno ChatGPT, ima značajan potencijal u unapređenju procesa testiranja softvera. Uvođenje alata umjetne inteligencije u standardizirane procese testiranja, poput onih definiranih normama ISO/IEC/IEE 29119 i ISO/IEC 25010, može značajno poboljšati kvalitetu, efikasnost i pokrivenost testova. ChatGPT je pokazao sposobnost da detaljno analizira, uspoređuje i predlaže poboljšanja jedinica testa, što ukazuje na njegovu korisnost kao dodatnog alata u razvoju softvera.

Umjetna inteligencija koja detaljno poznaje pojmove, kvalitete i načine implementacije normi iz ovog rada pokazala je svoju sposobnost i puni potencijal. Pristupom koji je korišten kod ocjenjivanja koda umjetna inteligencija je također pokazala svoju objektivnost i profesionalnost.

Ono što se pokazalo u ovom radu je da integracija umjetne inteligencije s tradicionalnim metodama razvoja dovodi do lakšeg i bržeg razvoja koda; te smanjenja vremena potrebnog za razvoj testova; te dovodi do poboljšanja u dijagnosticiranju i ispravljanju problema.

ChatGPT pokazao se kao jako koristan pomoćni alat zbog njegove sposobnosti: kritičke analize koda, lociranja područja za unapređenje programskog koda kao i pružanjem konkretnih rješenja u kontekstu složenih projekata gdje su pogreške svakodnevna događanja.

Umjetna inteligencija može značajno unaprijediti kvalitetu softverskih rješenja, ali njezina primjena zahtijeva pažljivo planiranje i integraciju s postojećim praksama softverskog inženjerstva. Važno je napomenuti da ljudska stručnost i dalje igra ključnu ulogu. Daljnji razvoj i integracija alata umjetne inteligencije u fazama razvoja softvera otvara nove mogućnosti za povećanje kvalitete i učinkovitosti u programskom inženjerstvu, čineći ove alate neprocjenjivima za budućnost industrije.

Zaključno, u ovom radu je dokazano da umjetna inteligencija ima ogromne mogućnosti i prednosti, ali samo dok se koristi kao asistent u razvoju, a ne kao posve autonoman sustav koji nitko ne kontrolira i oblikuje. Moje viđenje budućnosti umjetne inteligencije je da će se nastaviti unapređivati i da će s vremenom biti sve više i više raširenije njegovo korištenje, ali cijelo vrijeme trebamo imati na umu njegova ograničenja, a to su da za razliku od ljudi ona nije kreativna i samo služi za olakšavanje i poboljšavanje ljudskih koncepata.

Popis literature

- [1] Ritesh Singh, Swati Singh, Amulya Singh, „Landmark events, major trends, and the future of AI research: history, promise, hype“, svibanj 2024., *IEE Explorer*. [Na internetu] Dostupno: <https://ieeexplore.ieee.org/document/10531477> (pristupljeno 15. kolovoz 2024.).
- [2] B.J. Copeland, „Artificial intelligence“, 16. kolovoz 2024., Britannica. Dostupno: <https://www.britannica.com/technology/artificial-intelligence> (pristupljeno 15. kolovoza 2024.).
- [3] „What is artificial intelligence (AI)?“, (bez dat.) IBM. [Na internetu]. Dostupno: <https://www.ibm.com/topics/artificial-intelligence> (pristupljeno 15. kolovoz 2024.).
- [4] Vladimir Zwass, „Neural network“, 25. lipanj 2024, Britannica. [Na internetu] Dostupno: <https://www.britannica.com/technology/neural-network> (pristupljeno 15. kolovoz 2024.).
- [5] Ming Zhou, Nan Duan, Shujie Liu, [Heung-Yeung Shum](#), „Progress in Neural NLP: Modeling, Learning, and Reasoning“, ScienceDirect. [Na internetu]. Dostupno: <https://www.sciencedirect.com/science/article/pii/S2095809919304928> (pristupljeno 15. kolovoz 2024.).
- [6] Tara Ramanathan, „Natural language processing“, Britannica. [Na internetu] Dostupno: <https://www.britannica.com/technology/natural-language-processing-computer-science> (pristupljeno 15. kolovoz 2024.).
- [7] Erik Gregersen, „ChatGPT“, 16. kolovoz 2024., Britannica. [Na internetu]. Dostupno: <https://www.britannica.com/technology/ChatGPT> (pristupljeno 15. kolovoza 2024.).
- [8] Frannie Comstock, „Prompt engineering“, 31. lipanj 2024., Britannica. Dostupno: <https://www.britannica.com/technology/prompt-engineering> (pristupljeno 15. kolovoza 2024.).
- [9] Hugh Son, „JPMorgan Chase is giving its employees an AI assistant powered by ChatGPT maker OpenAI“, 9 kolovoz 2024., CNBC. Dostupno: <https://www.cnbc.com/2024/08/09/jpmorgan-chase-ai-artificial-intelligence-assistant-chatgpt-openai.html> (pristupljeno 15. kolovoza 2024.).

[10] Christof Ebert, Panos Louridas, „Generative AI for Software Practitioners“, 7. srpanj 2023, IEE Explorer. Dostupno: <https://ieeexplore.ieee.org/abstract/document/10176168> (pristupljeno 15. kolovoz 2024.).

[11] „ChatGPT chatbot“, svibanj 2024, AIAAIC. [Na internetu]. Dostupno: <https://www.aiaaic.org/aiaaic-repository/ai-algorithmic-and-automation-incidents/chatgpt-chatbot> (pristupljeno 15. kolovoza 2024.).

[12] „Words are flowing out like endless rain: Recapping a busy week of LLM news“, 12. travanj 2024., Ars technica. [Na internetu]. Dostupno: https://arstechnica.com/information-technology/2024/04/words-are-flowing-out-like-endless-rain-recapping-a-busy-week-of-llm-news/?itm_source=parsely-api (pristupljeno 15. kolovoz 2024.).

[13] Benj edwards, „Words are flowing out like endless rain: Recapping a busy week of LLM news“, 12. travanj 2024., Ars technical. [Na internetu]. Dostupno: <https://arstechnica.com/information-technology/2024/08/chatgpt-unexpectedly-began-speaking-in-a-users-cloned-voice-during-testing/> (pristupljeno 15. kolovoza 2024.).

[14] Nicholas Gissona, „Chatbot“, 1. kolovoz 2024., Britannica, Dostupno: <https://www.britannica.com/topic/chatbot> (pristupljeno 15. kolovoz 2024.).

[15] Dutch judge uses ChatGPT to help reach a verdict, 5. kolovoz 2024, DutchNews. [Na internetu]. Dostupno: <https://www.dutchnews.nl/2024/08/dutch-judge-uses-chatgpt-to-help-reach-a-verdict/> (pristupljeno 15. kolovoz 2024.).

[16] „Introduction to Software Engineering“, 27. lipnja 2024., *Geek for geeks*. [Na internetu] Dostupno: <https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/> [pristupljeno 16. kolovoz 2024.].

[17] LMSYS Chatbot Arena, 12. kolovoz 2024. [Na internetu] Dostupno: <https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard> (pristupljeno 16. kolovoz 2024.).

[18] Software Quality - Future Perspectives on Software Engineering Quality 2021.pdf

[19] Alexander S. Gillis, „What is quality assurance (QA?)“, (bez dat.) Techarget. [Na internetu]

Dostupno: <https://www.techtarget.com/searchsoftwarequality/definition/quality-assurance>

(pristupljeno 16. kolovoz 2024.)

[20] „About ISO“, (bez dat.) ISO [Na internetu]. Dostupno: <https://www.iso.org/about>

(pristupljeno 15. kolovoz 2024.).

[21] „What is ISO27001? A short guide“, (bez dat.) Certikit [Na internetu]. Dostupno:

<https://certikit.com/what-is->

[iso27001/?gad_source=1&gclid=Cj0KCQjwllG2BhC4ARIsADBgpVRSimxtESb5QZt5yQkOyh](https://certikit.com/what-is-iso27001/?gad_source=1&gclid=Cj0KCQjwllG2BhC4ARIsADBgpVRSimxtESb5QZt5yQkOyh)

[cSVK3eskij7wT3FipTuhV8bbPyQpXa7jkaAkYOEALw_wcB](https://certikit.com/what-is-iso27001/?gad_source=1&gclid=Cj0KCQjwllG2BhC4ARIsADBgpVRSimxtESb5QZt5yQkOyh) (pristupljeno 15. kolovoz 2024.).

[22] „ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security management systems — Requirements“, 2022, ISO [Na internetu].

Dostupno: <https://www.iso.org/standard/27001> (pristupljeno 15. kolovoz 2024.).

[23] „What is ISO 27001? A quick and easy explanation “, (bez dat.) ISO 27001 [Na

internetu]. Dostupno: <https://advisera.com/27001academy/what-is-iso-27001/> (pristupljeno

15. kolovoz 2024.).

[24] „ISO/IEC 25010“, (bez dat.) ISO 2500 [Na internetu]. Dostupno:

<https://iso25000.com/index.php/en/iso-25000-standards/iso-25010> (pristupljeno 15. kolovoz

2024.).

[25] John Estdale, App Stores & ISO/IEC 25000: Product Certification at Last, 2016, [Na

internetu]. Dostupno: [\(PDF\) App Stores & ISO/IEC 25000: Product Certification at Last?](#)

[\(researchgate.net\)](#) (pristupljeno 15. kolovoz 2024.).

[26] „ISO/IEC 25010:2023 Systems and software engineering — Systems and software

Quality Requirements and Evaluation (SQuaRE) — Product quality model“, (bez dat.), ISO

[Na internetu]. Dostupno: <https://www.iso.org/standard/78176.html> (pristupljeno 15. kolovoz 2024.).

[27] Chadatarn Raksawat and Pattama Charoenporn, „Software Testing System, 2021., *JAIT Development Based on ISO 29119*“ [Na internetu]. Dostupno: <https://www.jait.us/index.php?m=content&c=index&a=show&catid=206&id=1149> (pristupljeno 14. kolovoz 2024.).

[28] Tracy Phillips, „AI Code Tools: The Ultimate Guide in 2024“, 9. kolovoz 2024., *Code submit* [Na internetu]. Dostupno: <https://codesubmit.io/blog/ai-code-tools/> (pristupljeno 10. kolovoz 2024.).

[29] ChatGPT. (GPT-4), (bez dat.), *OpenAI*. [Na internetu]. Dostupno: <https://chatgpt.com/> (pristupljeno 10. kolovoz 2024.).

Popis slika

Slika 1. Godišnja vremenska serija za broj akademskih publikacija dostupnih u indeksu Google Scholar (izvor: preuzeto s „Landmark events, major trends, and the future of AI research: history, promise, hype“, IEEE Xplore)	5
Slika 2. Vremenski prikaz koliko je mjeseci bilo potrebno tehnologijama doći do 100 milijuna korisnika [izvor: preuzeto s „Generative AI for Software Practitioners“, IEE Xplore]	13
Slika 3. Usporedba alata za automatsko dovršavanje i generiranje koda koristeći UI (izvor: preuzeto s „Generative AI for Software Practitioners“, IEE Xplore.)	19
Slika 4. Lista modela umjetne inteligencije na platformi LMSYS Chatbot arena (izvor: preuzeto s „LMSYS Chatbot Arena Leaderboard“)	23
Slika 5. Web sučelje „The ISO 27001 Ninja“ (izvor: vlastita snimka zaslona)	30
Slika 6. Proces primjene ChatGPT-a za analizu kvalitete programskog koda prema normama (izvor: vlastita izrada u alatu Canva)	44
Slika 7. Ulazni podatak: poslovni zahtjev (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	61
Slika 8. Odgovor ChatGPT-a da je zapamtio programsku dokumentaciju (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	62
Slika 9. Ulazni podatak za ChatGPT - programski kod (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	62
Slika 10. Primjer zadatka postavljenog ChatGPT-u (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	62

Slika 11. Zaključak ChatGPT-a koji se odnosi na održivost koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	69
Slika 12. Prijedlozi za unapređenje 50 linija programskog koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	71
Slika 13. Relevantne linije programskog koda prvog cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	91
Slika 14. Relevantne linije programskog koda drugog cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	91
Slika 15. Relevantne linije programskog koda trećeg cilja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	91
Slika 16. Kreirani kod od strane UI za A.9 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	92
Slika 17. Kreiran kod od strane UI za A.12 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	93
Slika 18. Relevantne linije koda za A.12 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	93
Slika 19. Preporuka unapređenja koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana) .	94
Slika 20. Implementacija dijaloga za potvrdu (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	95
Slika 21. Obavijest korisnik nakon uspješnog arhiviranja (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	95

Slika 22. Dio programskog koda PK1 koji je potkrijepljen zadovoljavanje norme ISO/IEC 27001	96
Slika 23. Preporuka ChatGPT-a za unapređenje dodavanjem validacije u programski kod PK1 (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	97
Slika 24. Upit ChatGPT-a može li implementirati unapređenje (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	106
Slika 25. Provjera testa (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	107
Slika 26. Simulacija situacije gdje se dogodi greška prilikom spremanja promjena u bazu podataka (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	107
Slika 27. Provjera ponašanja koda (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	108
Slika 28. Unapređenje dokumentacije testova (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	111
Slika 29. Dodavanje identifikatora za testove (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	111
Slika 30. Dodavanje testova koji pokrivaju rubne slučajeve (izvor: ChatGPT (OpenAI), vlastita snimka ekrana).....	112
Slika 31. Dodavanje automatizacije (izvor: ChatGPT (OpenAI), vlastita snimka ekrana)	113
Slika 32. Postavljanje praćenja rizika (izvor: ChatGPT (OpenAI), vlastita snimka ekrana) ..	114

Popis tablica

Tablica 1. Ocjena ChatGPTa za karakteristiku Funkcionalna prikladnost	74
Tablica 2. Ocjena ChatGPT-a za karakteristiku Pouzdanost.....	75
Tablica 3. Ocjena ChatGPT-a za karakteristiku Upotrebljivost.....	76
Tablica 4. Ocjena ChatGPT-a za karakteristiku Učinkovitost	77
Tablica 5. Ocjena ChatGPT-a za karakteristiku Kompatibilnost	77
Tablica 6. Ocjena ChatGPT-a za karakteristiku Sigurnost	78
Tablica 7. Ocjena ChatGPT-a za karakteristiku Održavanje	79
Tablica 8. Ocjena ChatGPT-a za karakteristiku Prenosivost.....	80
Tablica 9. Tablica matrice parnih usporedbi	83
Tablica 10. Tablica normalizacije matrice	84
Tablica 11. Tablica težinskih faktora.....	85

Prilozi

Prilog 1. Izjava poduzeća 3MI lab za korištenje imena i prikupljene poslovne dokumentacije

3MI lab d.o.o. za informatičke usluge
Ulica majstora Radonje 10, Zagreb
tel: 099 2108417
OIB: 49145205842
Zagreb, 15. srpnja 2024.godine

KATARINA GALEKOVIĆ
OIB: 46304210939

IZJAVA

Potpisom ove izjave ja Marin Visković, OIB 66163092192, kao direktor trgovačkog društva 3MI lab d.o.o. i osoba ovlaštena za zastupanje istog, dajem suglasnost KATARINI GALEKOVIĆ da za potrebe izrade Diplomskog rada koristi ime trgovačkog društva 3MI lab d.o.o. kao i prikupljenu poslovnu dokumentaciju.

**MARIN
VISOVIĆ** Digitalno potpisao:
MARIN VISOVIC
Datum: 2024.08.16
12:42:53 +02'00'