Analiza posebnosti zagotavljanja kakovosti pri razvoju decentraliziranih aplikacij v okolju Ethereum

Mitja Gradišnik, Tina Beranič, Muhamed Turkanović
Fakulteta za elektrotehniko, računalništvo in informatiko, Univeza v Mariboru, 2000 Maribor
mitja.gradisnik@um.si, tina.beranic@um.si, muhamed.turkanovic@um.si

Izvleček

Pri razvoju decentraliziranih aplikacij (dApps) se tradicionalni razvojni procesi pogosto izkažejo za nezadostne. Tovrstne rešitve zahtevajo večji poudarek na tehničnih, varnostnih in uporabniških vidikih kakovosti aplikacij, kot smo jih sicer vajeni pri razvoju klasičnih rešitev. Ker je spreminjanje pametnih pogodb po namestitvi v omrežje verig blokov zahtevno oziroma nemogoče, sta temeljito testiranje ter presoja programske kode ključnega pomena za uspešen razvoj tovrstnih rešitev. Optimizacija stroškov goriva, nujna za izvrševanje programov v javnih omrežjih, predstavlja enega ključnih razvojnih izzivov, ki ga je potrebno ustrezno obravnavati. Poleg tega specifično okolje omrežij veriženja blokov zahteva ustrezne ukrepe za obvladovanje tveganj povezanih z ranljivostmi aplikacij in morebitnimi povezanimi finančnimi izgubami. Nespremenljivost, stroški goriva in zagotavljanje varnosti so le nekateri izmed ključnih razvojnih izzivov, ki jih je treba uspešno nasloviti pri izgradnji kakovostnih in stabilnih decentraliziranih aplikacij. Prispevek obravnava izzive, sodobne pristope in strategije razvoja decentraliziranih aplikacij ter podaja priporočila za njihov zanesljivejši in učinkovitejši razvoj, s čimer naslavlja ključne izzive uvajanja tehnologij veriženja blokov v industrijska okolja ter razvoja pametnih pogodb. Poseben poudarek je namenjen pametnim pogodbam, ki temeljijo na omrežju Ethereum.

Ključne besede: verige blokov, proces razvoja dApps, testiranje, vzdrževanje, obvladovanje stroškov transakcij, presoje

Analysis of Quality Assurance Specifics in the Development of Decentralized Applications in the Ethereum Environment

Abstract

In the development of decentralised applications, traditional development processes often prove insufficient. Such solutions require greater emphasis on the technical, security, and user experience aspects of application quality than is common in developing conventional applications. Since modifying smart contracts after deployment on a blockchain network is challenging or even impossible, thorough testing and code review are crucial for successfully developing such solutions. Optimising gas consumption, which is essential for executing programmes in public networks, represents one of the key development challenges that must be adequately addressed. Furthermore, the specific environment of blockchain networks demands appropriate measures to manage risks associated with application vulnerabilities and potential financial losses. Immutability, gas consumption, and security are critical development challenges that must be successfully tackled to build high-quality and stable decentralised applications. This paper addresses the challenges and modern approaches and strategies for dApp development. It offers recommendations for their more reliable and efficient implementation, thereby tackling key issues related to adopting blockchain technologies in industrial settings and developing smart contracts. Emphasis is placed on smart contracts based on the Ethereum network.

Keywords: Blockchain, dApps development process, testing, maintainability, cost management, audits

1 UVOD

Pospešen razvoj tehnologij veriženja blokov v zadnjih letih odpira vrata številnim inovacijam na področjih elektronskega poslovanja in shrambe podatkov ter pomembno prispeva k uvajanju novih poslovnih modelov [1]. Ključni pospeševalec teh inovacij je predvsem decentralizirana narava okolja verig blokov. Pred vpeljavo tehnologij veriženja blokov je naša družba temeljila na vzpostavljanju zaupanja med deležniki, pri čemer so ključno vlogo odigrali zaupanja vredni posredniki [2]. Tipičen primer takega posrednika so banke. Banke, kot zaupanja vreden posrednik, poskrbijo za prenos sredstev med komitenti, pri čemer morajo ti banki zaupati. Komitenti morajo banki zaupati svoja denarna sredstva ter zaupati v pravilnost izvedenih transakcij in zanesljivost njihovih zapisov, da bi omenjeni mehanizem lahko uspešno deloval.

Na tehnologijah veriženja blokov temelječe programske rešitve odpravijo potrebo po zaupanja vrednem posredniku med akterji, ki so vpleteni v elektronsko poslovanje. Zaupanje se prenese na decentralizirane sisteme same, konkretno na uporabo ustreznih kriptografskih metod in na delovanje porazdeljenega in decentraliziranega omrežja [3]. V praksi to pomeni, da tako grajene rešitve omogočajo varne in verodostojne transakcije med akterji, med katerim sicer ni potrebe po vzpostavitvi predhodnega zaupanja, za katerega bi skrbel zaupanja vreden posrednik. Tehnologije veriženja blokov ponujajo možnost izvajanja transakcij med dvema ali več anonimnimi deležniki preko mehanizma vnaprej dogovorjenih in zapisanih procedur, imenovanih pametne pogodbe [4]. Izboljšana varnost, transparentnost in decentralizacija elektronskega poslovanja so tako ključni stebri inovacij, ki jih prinaša vpeljava tehnologij veriženja blokov.

Inovatorji na področju vpeljave tehnologij veriženja blokov v poslovne procese izhajajo iz različnih gospodarskih domen, pri čemer prednjačijo področja decentraliziranih financ, zavarovalništva, dobavnih verig in logistike, zdravstva, upravljanja digitalnih identitet ter javne uprave [5], [6]. Tehnologije veriženja blokov omogočajo razvoj novega tipa programskih rešitev, ki realizirajo njihove ideje. Posledično se v zadnjem času pojavlja čedalje več aplikacij razvitih na temeljih tehnologij verig blokov, ki jih imenujemo decentralizirane aplikacije ali krajše dApps [7].

Jedro tehnologij predstavljajo linearno strukturirane verige blokov, v katere se zapisujejo stanja in transakcije izvršene med deležniki omrežja [8]. Operacije upravljanja s podatki so pri tem enosmerni proces. Omogočeno je zapisovanje oz. dodajanje zapisov, ne pa tudi njihovo spreminjanje ali brisanje. Tako zapisani podatki ostanejo v verigah blokov trajno [9]. Ker imajo podatkovni bloki omejeno kapaciteto, se zapisi razpršeni čez več podatkovnih blokov, ki so urejeni po času ustvarjanja tvorijo verige [8]. Celotna strukturo, na kateri decentralizirane aplikacije temeljijo in vanjo zapisujemo podatke, imenujemo porazdeljena knjiga transakcij (angl. distributed ledger) [10]. Posege v integriteto tako zapisanih podatkov preprečuje vrednost zgoščevalne funkcije vsebine bloka. Ta se zapiše v sledeči blok in tako ustvarja trajne povezave med podatkovnimi bloki. S tem se gradi verižna podatkovna struktura podprta z Merklovimi drevesi [11]. Nezmožnost brisanja zapisanih podatkov daje programskih produktom, ki temeljijo na verigah blokov, eno izmed ključnih lastnosti, in sicer nespremenljivost zapisov. Nespremenljivosti zapisov je ključna lastnost verig blokov, na kateri temelji varnost in zaupanje v ustvarjene zapise. Varnost zapisanih podatkov krepi odpornost na manipulacije in možnost sledenja spremembam [9].

Zaupanje v zapise krepi transparentnost in integriteto zapisanih podatkov, saj so, zaradi distribuiranega okolja omrežja verig blokov, vsi zapisi vidni vsem deležnikom. Za slednje je ključna porazdeljenost zapisov med vozlišči omrežja verig blokov. Protokoli pri tem poskrbijo za replikacijo, tako da ima vsako vozlišče v omrežju popolno kopijo podatkov, kar naredi zapise robustne in odporne na izgubo. Slednje dodatno prispeva k varnosti zapisanih podatkov. Potrjevanje stanja podatkov v porazdeljenem okolju s pomočjo algoritmov porazdeljenega soglasja omogoča decentralizacijo, saj prepreči odvisnost od enega samega strežnika ali ene centralne entitete, ki upravlja zapise [9]. Decentralizacija je še ena izmed ključnih edinstvenih lastnosti tehnologij blokov, ki omogoča dinamično vzpostavitev skupnosti akterjev brez potrebe po vzpostavitvi vrhnje avtoritete, ki to skupnost upravlja, hkrati pa onemogoča preprosto in dinamično spreminjanje kakršnih koli atributov delovanja takšnega omrežja.

Verige blokov bi imele močno omejeno uporabnost za poslovna okolja, če ne bi bile programabilne. Tekom razvoja v zadnjih letih so tehnologije veriženja

blokov prerastle platforme digitalnih valut ter vpeljale decentralizirane in zaupanja vredne programske rešitve [1]. Številne domene s pridom izkoriščajo prednosti decentralizacije, transparentnosti in varnosti zapisov ter možnost zaupanja, ki ga je mogoče vzpostaviti med sicer anonimnimi deležniki. Posledično je čedalje več aplikacij, ki temeljijo na tehnologijah porazdeljenih knjig [7], čemur se morajo ustrezno prilagoditi tudi razvijalci programske opreme. Čeprav je na voljo več platform za razvoj pametnih pogodb, se v nadaljevanju osredotočamo predvsem na Ethereum [12], ki predstavlja eno izmed vodilnih platform za razvoj pametnih pogodb [7]. S platformo imamo tudi največ praktičnih izkušenj. Ob tem je treba poudariti, da se članek osredotoča izključno na aplikacijski nivo verig blokov, medtem ko infrastrukturni nivo, ki v osnovi zagotavlja delovanje tovrstnih omrežij, ni predmet obravnave.

Namen prispevka je proučiti posebnosti decentraliziranih aplikacij, ki jih je potrebno v okviru razvojnega procesa aplikacij ustrezno nasloviti, da bi se izognili težavam s kakovostjo nastalih programskih rešitev. Cilj prispevka je identificirati izzive, ki jih prinaša razvoj decentraliziranih aplikacij, in razpoložljive metode in ukrepe, s katerimi tovrstne razvojne izzive uspešno naslavljamo. V skladu s cilji raziskave smo izpeljali dve osnovni raziskovalni vprašanji:

RV1: Katere so posebnosti decentraliziranih napram klasičnim aplikacijam?

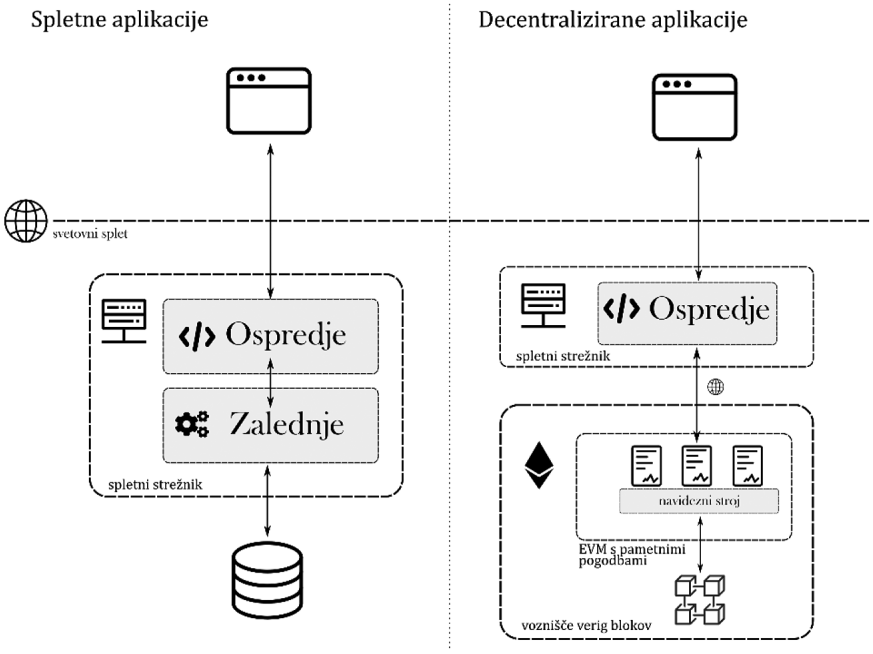
RV2: Katere razvojne metode in aktivnosti naslavljajo njihove posebnosti pri zagotavljanju kakovosti decentraliziranih aplikacij?

Raziskava se metodološko opira predvsem na pregled literature, pri čemer je bila pregledana relevantna aktualna literatura domačih in tujih avtorjev s področja programskega inženirstva. Za opazovanje, opisovanje in analiziranje identificiranih razvojnih izzivov in njihovih rešitev, bo v prispevku uporabljena metoda deskripcije. Metodološko je prispevek teoretične narave in služi kot uvod v nadaljnje raziskovalno delo na področju obvladovanja kakovosti pri razvoju decentraliziranih aplikacij. Poglavitni cilj raziskav je identificirati tiste posebnosti tehnologij veriženja blokov, ki jih je potrebno nasloviti pri gradnji kakovostnih decentraliziranih aplikacij.

2 DECENTRALIZIRANE APLIKACIJE

Arhitekturna zasnova decentraliziranih aplikacij je sicer podobna spletnim rešitvam, vendar z nekaterimi pomembnimi razlikami. V grobem so decentralizirane aplikacije sestavljene iz dveh ključnih komponent: ospredja z uporabniškim vmesnikom in zalednega dela s poslovno logiko aplikacije. Ospredje poskrbi za interakcijo uporabnikov z aplikacijo in je v veliki večini primerov realizirano kot spletna aplikacija. Poslovna logika v zaledju decentraliziranih aplikacij je običajno zapisana v pametnih pogodbah, ki predstavljajo jedro vsake decentralizirane aplikacije. Čeprav obstaja široka paleta različnih platform verig blokov s podporo pametnim pogodbam, se v praksi najpogosteje uporablja platforma Ethereum, pri čemer je za razvoj pametnih pogodb uporabljen programski jezik Solidity [7]. Pametne pogodbe implementirajo tako poslovno logiko aplikacije in hkrati poskrbijo za hrambo zapisov v verigah blokov [13]. Zaledni del aplikacije s poslovno logiko in podatkovna baza za hrambo podatkov pri decentraliziranih aplikacijah nista več potrebna, vendar sta še vedno lahko prisotna. Kadar sta prisotna, govorimo o hibridnih arhitekturah, kjer se del podatkov hrani v podatkovno bazo, del pa se jih zapiše v verige blokov [14]. Primerjavo med arhitekturama klasičnih spletnih aplikacij in decentraliziranih aplikacij [15] prikazuje Slika 1. Iz arhitekturne sheme je razvidno, da je ospredje decentraliziranih aplikacij enako ospredju spletnih aplikacij in se z razvojnega vidika ne razlikujeta veliko. Posebnosti razvoja decentraliziranih aplikacij izhajajo iz zaledja s pametnimi pogodbami. Večina posebnosti v razvoju decentraliziranih aplikacij izhaja ravno iz zalednega dela aplikacije in dejstva, da mora biti ospredje primerno usklajeno z zaledjem, ki je temelječe na pametnih pogodbah, kakor tudi povezljivo z uporabniškimi denarnicami tehnologije veriženja blokov. Pri vsem tem nam pomagajo namenske knjižnice, kot sta web3 [16] ali ethers [17]. Upoštevati je potrebno dodatno arhitekturno dejstvo. In sicer, v primeru potrebe po shranjevanju večjih količin podatkov ali datotek, se, z razliko od uporabe centraliziranih strežnikov, v dApp poslužujemo decentraliziranih sistemov shranjevanja datotek, kot je IPFS.

Osrednji del decentraliziranih aplikacij predstavljajo pametne pogodbe. Te sledijo v devetdesetih letih predstavljeni ideji samoizvršljivih pogodb, ki temeljijo na transakcijskem protokolu, ki ga avtono-



Slika 1 **Primerjava arhitekture spletnih in decentraliziranih aplikacij**

mno izvaja računalnik [4]. V osnovi gre za samoizvršljivo programsko kodo, ki se avtonomno izvede, kadar so za to izpolnjeni vnaprej definirani pogoji [18]. Pametne pogodbe bi lahko tako primerjali s konceptom klasičnih pogodb iz realnega sveta, pri čemer so pametne pogodbe povsem digitalizirane. Kot take so shranjene na vozliščih omrežja verig blokov.

2.1 Okolje izvajanja decentraliziranih aplikacij

Pomembna razlika med spletnimi in decentraliziranimi aplikacijami izhaja iz izvajalnega okolja, v katerem aplikacije tečejo. Spletne aplikacije običajno tečejo na spletnih strežnikih, ki zagotovi izvajalno okolje za njihovo izvajanje. Znotraj spletnega strežnika običajno teče celotna spletna aplikacija. Od aplikacije je običajno oddvojena le podatkovna baza. Nekatere zadolžitve prevzame sam spletni strežnik, in sicer skrb za usmerjanje prometa do aplikacije in predhodna obdelava HTTP zahtev, varnost aplikacij, upravljanje uporabnikov in dostopov ter poganjanje spletnih aplikacij [19]. Upoštevati je potrebno tudi skaliranje aplikacij, ki zajema tudi horizontalno skaliranje zalednih delov in samih spletnih strežnikov. Velikokrat se poslužujemo za ta namen avtomatiziranih pristopov, ki nam jih ponujajo ponudniki oblačnih storitev. Decentralizirane aplikacije močno zmanjšajo odvisnost od spletnih strežnikov, saj jih uporabljajo zgolj za poganjanje ospredja z uporabniškim vmesnikom. Pametne pogodbe, kot zaledna komponenta decentraliziranih aplikacij, tečejo ločeno od ospredja na vozliščih omrežja verig blokov v navideznih strojih sistemov v omrežju. Na primer, pametne pogodbe napisane za platformo Ethereum se izvajajo v Ethereum navideznih strojih [7]. Upoštevajoč zapisano, lahko naredimo primerjavo z oblačno skalirano infrastrukturo, vendar je ključna razlika ponovno decentralizacija, saj se v primeru klasične oblačne infrastrukture vsi procesi izvajajo in nadzorujejo s strani centralnega ponudnika takšnih storitev. Prav tako je okvir funkcionalnosti pri navideznih strojih v verigah blokov močno okrnjen, saj ti, za razliko od spletnih strežnikov, skrbijo izključno za izvajanje programske kode pametnih pogodb. Pomembna razlika okolij pametnih pogodb v primerjavi s vsebnikom spletnih strežnikov je v načinu zaračunavanja stroškov. Ker tečejo pametne pogodbe porazdeljeno po vozliščih omrežja verig blokov, je potrebno za uporabo infrastrukture plačati pristojbine za uporabo. Decentralizirane aplikacije so torej močno občutljive na količine podatkov, katere obdelujejo in zapisujejo.

3 POSEBNOSTI RAZVOJA DECENTRALIZIRANIH APLIKACIJ

Primerjava arhitekture decentraliziranih aplikacij s sorodnimi spletnimi rešitvami nakazuje na večjo kompleksnost decentraliziranih aplikacij [13]. V

praksi se izkaže, da je decentralizirane aplikacije posledično tudi težje razvijati. Arhitektura decentraliziranih aplikacij, posebnosti pametnih pogodb in lastnosti omrežij verig blokov torej zahtevajo prilagojene procese razvoja tovrstnih programskih rešitev.

3.1 Nespremenljivost pametnih pogodb

Spremembe so neizogiben del življenjskega cikla programskih produktov. Redki so programski produkti, ki po izdajni različici ne bi deležni nobenih sprememb ali popravkov. Ne glede na to, kako dobro so bile v začetni fazi razvoja programskega produkta zajete zahteve uporabnikov in kako dobro so bile kasneje v fazah razvoja te prenesene v programski produkt, sčasoma postaja razkorak med implementirano programsko rešitvijo in pričakovanji uporabnikov čedalje večji. Vzrokov za omenjen pojav je več. Na primer, delovni procesi v organizacijah, v okviru katerih se programska oprema uporablja, se neprestano spreminjajo in nadgrajujejo. Gonilo sprememb so bodisi nadgradnje bodisi razširitve delovnih procesov. Pričakovano je, da tem spremembam sledi tudi programska oprema in se na spremembe ustrezno prilagodi. Četudi bi delovni procesi ostali nespremenjeni, na spremembe programske opreme napeljuje kup zunanjih dejavnikov, kot so spremembe v zakonodaji, spremembe povezanih sistemov in spremenjeno okolje delovanja organizacije. Nenazadnje je potrebno upoštevati, da se običajno tekom uporabe programskega produkta odkrije nekaj hroščev, ki se jih v fazi testiranja ni zaznalo in jih je potrebno za nemoteno uporabo aplikacije odpraviti. Spremembe so torej sestavni del življenjskega cikla programskega produkta, ki ga sodobni razvojni procesi uspešno naslovijo.

Glede na sorodnost in podobnost med decentraliziranimi in spletnimi aplikacijami, bi se na prvi pogled zdelo smiselno, da se dobro preizkušeni in uveljavljeni agilni proces razvoja spletnih rešitev uporabi tudi pri razvoju decentraliziranih aplikacij. Ker pametnih pogodb, kot osrednje komponente decentraliziranih aplikacij, ko so te enkrat nameščene v okolje verig blokov, ne moremo enostavno spreminjati, decentralizirane aplikacije ne zahtevajo faze vzdrževanja v klasičnem pomenu besede [20]. Izkaže se, da agilnega procesa razvoja spletnih rešitev ni mogoče enostavno in brez prilagoditev prenesti v okvir razvoja decentraliziranih aplikacij.

Kratki iterativni razvojni cikli agilnih pristopov poskrbijo za odzivnost na spremembe v uporabniških zahtevah, s tem da prenesejo te spremembe v čim krajšem času v programski produkt. Pristopi in avtomatizacija procesa, ki jo vpelje DevOps poskrbijo, da je razvojni proces pri tem čim bolj učinkovit. Rezultat iterativnih razvojnih ciklov so torej vedno nove posodobljene različice programskih rešitev, ki se preko različnih mehanizmov avtomatizacije razvojnega procesa v čim krajšem možnem času prenesejo v produkcijsko okolje in predajo v uporabo. Naveden proces se uporablja pri različnih tipih programskih produktov, ne glede na platformo, programski jezik ali namen uporabe.

Čeprav za širok spekter programskih produktov opisan pristop v praksi deluje odlično in se izkaže za izjemno učinkovitega, tovrstni pristop k razvoju pri decentraliziranih aplikacijah trči na oviro. Če so običajne aplikacije odlično prilagojene neprestanemu spreminjanju in nadgrajevanju, so spremembe decentraliziranih aplikacij trši oreh. Pametnih pogodb, kot osrednje komponente decentraliziranih aplikacij, zaradi lastnosti tehnologij veriženja blokov ni mogoče enostavno spreminjati, ko so te enkrat nameščene v omrežje verig blokov. Navedeni varnostni mehanizem pametnih pogodb, ki preprečuje možnost njihovega kasnejšega zlonamernega spreminjanja, postavlja dodatne izzive pri procesu njihovega dolgoročnega vzdrževanja. V praksi to pomeni, da tudi manjše nadgradnje ali popravki hroščev v algoritmu pametnih pogodb, rezultirajo v ponovno nameščanje nove različice pametne pogodbe, ki prejšnjo nadomešča, kar povzroči izgubo podatkov, vezanih na predhodno različico. Razvojni cikel decentraliziranih aplikacij mora slediti načelu, da je potrebno pametne pogodbe spisati s popolnim naborom funkcionalnosti in vsa v teoriji brez programskih hroščev, zaradi katerih bi bili kasneje prisiljeni v izdajo posodobljenih različic. Ker pa bo do sprememb zagotovo prišlo, uporabljajo razvijalci decentraliziranih aplikacij različne nadgradljive arhitekture, da omogočijo posodabljanje funkcionalnosti. Najpogostejši pristop je uporaba **vzorca posrednika** [21], [22], kjer glavna pogodba preusmerja klice na ločeno, zamenljivo pogodbo z dejansko logiko. Naprednejši, a bolj kompleksen je **diamantni vzorec** [23], ki omogoča modularne nadgradnje posameznih funkcionalnosti. Alternativno se včasih uporabi preprosta **migracija na novo pogodbo**, kjer se decentralizirana aplikacija poveže na nov naslov, stanje pa se prenese ročno ali s skriptami. Dodatno fleksibilnost ponuja **vzorec register** [21],

kjer osrednja pogodba hrani naslove vseh aktivnih komponent. Izbira pravega mehanizma je odvisna od kompleksnosti sistema, potrebe po fleksibilnosti ter varnostnih in uporabniških zahtev.

3.2 Izpostavljenost med izvajanjem

Spletne aplikacije ne tečejo kot samostojni procesi na sistemu, temveč so nameščene v vsebnik spletnega strežnika. Spletni strežnik spletnim aplikacijam nudi varno izvajalno okolje in jih neposredno zaščiti pred zunanjim okoljem. Kadar poteka komunikacija s spletno rešitvijo, zahtevke sprejme in predhodno obdela spletni strežnik. Šele po predobdelavi so zahtevki posredovani spletnim aplikacijam. Spletni strežnik je torej zadolžen za kopico funkcionalnosti, performančnih in varnostih mehanizmov spletnih aplikacij, za katere razvijalcem spletnih rešitev ni potrebno neposredno skrbeti. Običajno je potrebna zgolj ustrezna konfiguracija spletnega strežnika.

Za razliko od spletnega strežnika, navidezni stroj na vozliščih omrežij verig blokov pametnim pogodbam ne nudi tako širokega nabora funkcionalnosti, temveč zgolj izvajalno okolje pametnih pogodb. Pametne pogodbe v okolju omrežja verig blokov delujejo kot povsem samostojne aplikacije, ki same sprejemajo zahtevke uporabnikov, jih izvršijo in klicatelju same vrnejo rezultat obdelave. Navidezni stroj pri tem ne nudi nobenih dodatnih funkcionalnosti, niti ne poskrbi za performančne in varnostne vidike decentraliziranih aplikacij, razen preverjanja celovitosti transakcij. Na razvijalcih pametnih pogodb je torej, da sami v programski kodi pametne pogodbe poskrbijo, da je izvajanje pogodbe performančno učinkovito, varno ter dostopno zgolj uporabnikom, ki do izbranih funkcionalnosti pametnih pogodb smejo dostopati. Razvijalci pametnih pogodb tudi nosijo breme odgovornosti, da je implementacija pametnih pogodb brezhibna in ne predstavlja nepotrebnega varnostnega tveganja za uporabnike decentraliziranih aplikacij.

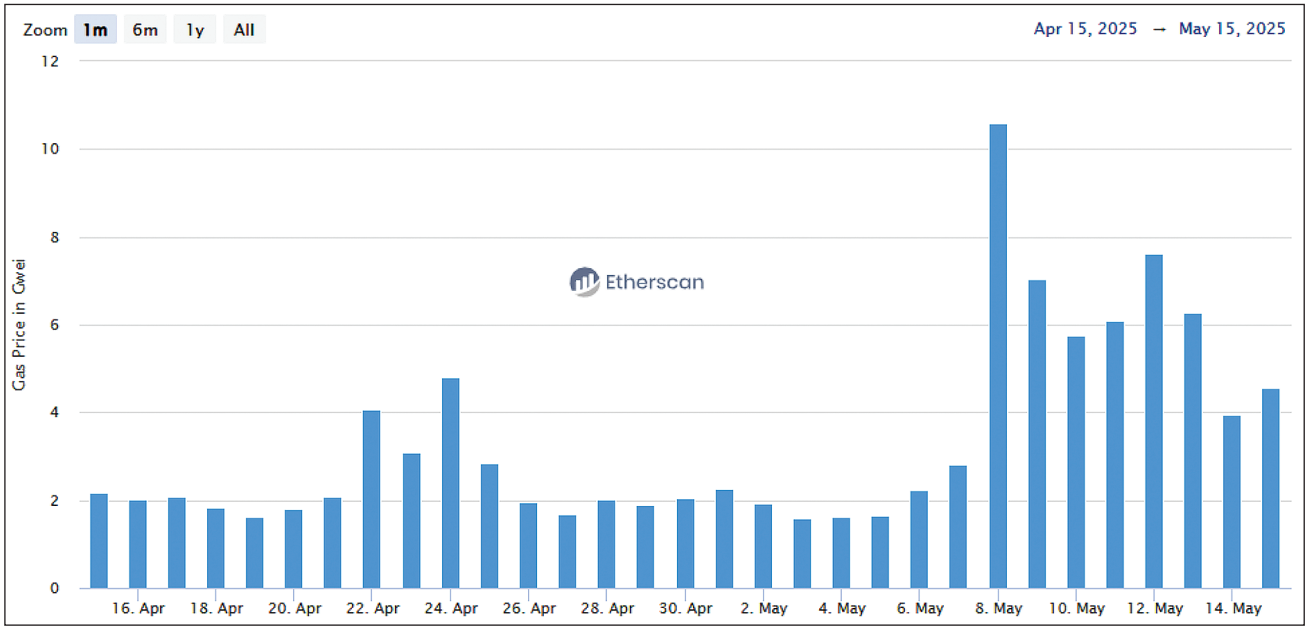
3.3 Obvladovanje stroškov uporabe infrastrukture

Performančna učinkovitost je pomemben atribut kakovosti programskih komponent, ki ga je potrebno ustrezno nasloviti pri razvoju programskih rešitev, ne glede na njihovo vrsto. Performančno neučinkovita programska oprema po nepotrebnem porablja sistemske vire in vpliva na stroške uporabljene infrastrukture ne glede na to, ali je ta najeta ali lastniška. Poudarek na stroškovni učinkovitosti je pri pametnih pogodbah še posebej izrazit.

Pametne pogodbe, nameščene na javnih verigah blokov, kot je Ethereum, se izvajajo na vozliščih, pri čemer je treba za uporabo infrastrukture plačati pristojbine za gorivo (angl. gas fees) [7]. Po prehodu Ethereuma na mehanizem soglasja ***Proof of Stake*** (PoS) so te pristojbine še vedno prisotne, čeprav se sedaj plačujejo overjevalcem (angl. validators) namesto rudarjem [24]. Cena pristojbine v posameznem omrežju je odvisna od kompleksnosti pametne pogodbe, zasedenosti omrežja in cene domorodne kriptovalute, v kateri se gorivo zaračunava [7], [25]. Gibanje cene pristojbine za omrežje Ethereum, izraženo v enoti GWei [7], za obdobje med 15. aprilom in 15. majem 2025 prikazuje Slika 2. GWei lahko razumemo kot manjšo enoto kripto kovanca, ki meri porabljeno računsko moč, potrebno za izvedbo operacij nad verigami blokov. Operacije na verigami blokov sicer definira algoritem zapisan v obliki pametnih pogodb. Temeljne operacije nad verigami blokov, za katere je potrebno plačevati pristojbine v omrežju Ethereum, na primer vključujejo zapisovanje podatkov v shrambo in računske operacije nad podatki. Pristojbine za gorivo so prav tako pomembne za ustvarjanje novih pogodb in pošiljanje žetonov. V omrežju verig blokov se izvajanje pametnih pogodb zaračunava po izvedenih operacijah zapisovanja podatkov. Praksa kaže, da stroški izvajanja pametnih pogodb v javnih omrežjih hitro narastejo [13], zaradi česar je njihovo obvladovanje ključno za vzdržnost dolgoročnega delovanja decentraliziranih aplikacij. V kolikor se za izvajanje decentraliziranih aplikacij uporabljajo javna omrežja, je pri njihovem razvoju torej potrebno vnaprej natančno proučiti predvidene stroške izvajanja pametnih pogodb in po potrebi izvesti njihovo optimizacijo.

3.4 Odprtokodnost programske kode

Javna objava programske kode pametnih pogodb je dobra praksa, ki ima številne pozitivne učinke. Odprtost kode zagotavlja transparentnost in prispeva k zaupanju med deležniki v omrežju. Vsak izmed deležnikov ima tako možnost vpogleda v algoritem pametne pogodbe in možnost preučitve njenega natančnega delovanja. S tem se lahko prepriča, da se pametna pogodba izvaja na pričakovan način. Transparentnost izvajanja pametnih pogodb je ključni mehanizem za preprečevanje morebitnih zlorab,



Slika 2: **Gibanje cene goriva za omrežje Ethereum [26]**

goljufij in manipulacij, kar pomembno pripomore k zaupanju v decentralizirane aplikacije [5]. Dodatno daje odprtokodnost priložnost vsem članom skupnosti, da prispevajo svoj delež k izboljšavam programske kode, na primer optimizaciji delovanja, odpravo najdenih napak ali dopolnitev funkcionalnosti. Pomemben vidik odprtokodnosti predstavlja tudi varnostna analiza pametnih pogodb, ki jo lahko opravi snovalec decentralizirane aplikacije ali skupnost.

Možnost vpogleda v izvorno kodo pametnih pogodb po drugi strani odpre vrata tveganjem, povezanim z izkoriščanjem opaženih pomanjkljivosti v programski kodi. Programska koda lahko vsebuje nezakrpane varnostne luknje, napake v logiki ali nepreverjene vhodne podatke. Te se v programski kodi kažejo v obliki logičnih napak ali uporabe slabih praks programiranja. Na tveganja, povezana z javno izpostavljenostjo izvorne kode pametnih pogodb, dodatno vpliva okolje, v katero so pametne pogodbe nameščene in na katerem se izvajajo. Navidezni stroji pametnim pogodbam ne nudijo nobenega dodatnega varnostnega ovoja ter jih navzven neposredno izpostavljajo. Pametne pogodbe lahko v javnih omrežjih pokliče kdorkoli, upoštevajoč vse morebitne implementirane modifikatorje in specifikacije vidnosti. Zato je toliko bolj pomembno, da je njihova programska koda brez morebitnih programskih pomanjkljivosti, saj so sicer nevarne za uporabo.

4 PRILAGODITVE RAZVOJNEGA PROCESA DECENTRALIZIRANIH APLIKACIJ

Nezmožnost enostavnega spreminjanja pametnih pogodb po namestitvi v okolje verig blokov, velika občutljivost, povezana z varnostnimi tveganji in relativno veliki stroški izvedenih operacij v omrežju nakazujejo na potrebo po prilagojenem razvojnem procesu. Sodobni agilni razvojni procesi s hitrim odzivanjem na uporabniške zahteve in avtomatizirano neprekinjeno integracijo in dostavo vedno novih različic programskega produkta se pri razvoju decentraliziranih aplikacij ne izkažejo za najučinkovitejši pristop k razvoju. Za kakovosten in stroškovno učinkovit razvoj pametnih pogodb je torej ključno, da se v fazi razvoja, pred namestitvijo v vozlišča verig blokov, zagotovi, da so te ustrezno načrtovane ter da ne vsebujejo programskih hroščev [7], ki bi razvijalce silile v izdajo popravljenih različic aplikacij.

4.1 Ključnost testiranja v zgodnjih fazah razvoja

Ker pametnih pogodb ni mogoče enostavno nadgrajevati, kot smo to sicer vajeni pri drugih družinah aplikacij, se je potrebno pred namestitvijo rešitve prepričati, da je rešitev funkcionalno ustrezna in ne vsebuje programskih hroščev, zaradi katerih bi bili prisiljeni v izdajo popravljene različice. V fazi načrtovanja je potrebno skrbno in celostno načrtovati tako funkcionalne kot arhitekturne vidike aplikacij. Prvi

v vrsti ukrepov za zagotavljanje kakovosti decentraliziranih aplikacij je obsežno in celovito testiranje. Sistematično testiranje v zgodnji fazah zagotovi, da se načrtovalske in implementacijske napake zaznajo in odpravijo še znotraj razvojnega cikla pred namestitvijo v produkcijsko okolje in predajo v uporabo. V praksi se izkaže, da zna biti odloženo odpravljanje programskih hroščev v kasnejših fazah stroškovno drago [27]. Praksa odloženega razreševanja razvojnih težav (angl. Delayed Issue Effect) sicer velja za slabo prakso tudi pri razvoju drugih družin aplikacij, vendar je težavnost odpravljanja napak zaradi nespremenljivosti pametnih pogodb po namestitvi na vozlišča verig blokov pri decentraliziranih aplikacijah toliko večja in ji je potrebno posvetiti več pozornosti [28]. Da bi preprečili nepotrebne stroške in škodo povezano z izgubo podatkov, se je potrebno izogniti napakam v produkcijskih različicah programskih rešitev.

Kljub temu, da se za razvoj pametnih pogodb uporabljajo manj uveljavljeni programski jeziki in platforme, ki so na voljo krajši čas in se uporabljajo v ožjem krogu razvijalcev, imajo razvijalci pametnih pogodb na voljo kopico testnih ogrodij in orodij, s katerimi je mogoče izvesti sistematično in temeljito testiranje. Slednje temelji na doslednem testiranju enot (angl. unit testing) in izvedbi testiranja fuzz. Pri testiranju enot se večinoma preverja pravilnost izračunov, delovanje nadzora dostopa in dovoljenja, upravljanje z dogodki itd. Testi fuzz predstavljajo tehniko avtomatiziranega testiranja, pri katerem se v vhode programov vnašajo obsežni, običajno naključno generirani in nepričakovani vhodni podatki, namen katerih je preveriti ustreznost delovanja aplikacije pod različnimi pogoji. Foundry [29] je primer orodja, ki omogoča takšno vrsto testiranja, pri čemer se lahko uporablja tudi za testiranje enot. Za slednje se sicer uporablja tudi Hardhat [30]. Del testiranja je lahko tudi avtomatski pregled izvorne kode za znane ranljivosti (npr. ponovni vstop, prelivanje, izvor transakcije, posredni klic). Tudi sicer je za testiranje decentraliziranih aplikacij potrebno ubrati učinkovitejši pristop, saj je strošek testiranja zaradi visokih stroškov decentraliziranega okolja bistveno višji, kot je strošek testiranja drugih družin aplikacij [5].

4.2 Testiranje porabe goriva

Izbira podatkovnih tipov in struktur podatkov, števila izvedenih ciklov in nabora ukazov, vrstnega reda inicializacij in ovrednotenj, ostaja povsem suverena odločitev razvijalcev pametnih pogodb [7]. Ne glede na razvijalčevo izbiro so nekateri zapisi algoritmov iz performančnega vidika učinkovitejši kot drugi, kar lahko pri izvajanju pametnih pogodb povzroči razlike v porabi goriva. Poglavitni cilj testiranja porabe goriva je preveriti, ali je poraba goriva izvajanja pametnih pogodb smotrna oz. ali v programski kodi pametnih pogodb obstajajo segmenti, ki bi jih bilo mogoče izvesti učinkoviteje. Naloga performančnega testiranja je torej preveriti, da programska koda na primer ne vsebuje nepotrebne programske kode, ne izvaja nepotrebnih iteracij zank in po nepotrebnem ne troši prostora za shranjevanje podatkov.

K testiranju porabe goriva pametnih pogodb pristopimo z dinamičnim testiranjem obnašanja aplikacije in statično analizo programske kode. Oceno porabe goriva lahko pridobimo z namestitvijo pametnih pogodb v simulirana omrežja, ki so lahko nameščena na lokalnem razvojnem ali zasebnem strežniku. Poganjanje in testiranje pametnih pogodb v lokalnem okolju omogoča oceno o sprejemljivosti stroškov izvajanja pametnih pogodb. Oceno stroškov porabe goriva pametnih pogodb je seveda mogoče izvesti neposredno na javnih omrežjih [31], vendar v tem primeru že nastanejo dejanski stroški. Če poraba goriva pri testiranju v simuliranem okolju ni zadovoljiva, je potrebno programsko kodo pametnih pogodb stroškovno optimizirati.

Stroškovna optimizacija programske kode pametnih pogodb temelji na statični analizi programske kode. Analiza zajema prepoznavo segmentov programske kode, ki veljajo za potratne. Takšne segmente programske kode označujemo kot smrdečo kodo. Sledi odstranitev nepotrebnih ali minimiziranje potratnih operacij na najmanjši možni obseg, potreben za realizacijo funkcionalnosti pametne pogodbe. V programski kodi se tako išče smrdečo kodo, na primer [7]:

- Podvojeni zapisi, kot so nepotrebna posodabljanja vrednosti spremenljivk znotraj zank.
- Neskončne zanke.
- Neučinkovitost pri inicializaciji spremenljivk, predvsem njihova inicializacija na njihove privzete vrednosti (npr. inicializacija tipa uint256 na vrednost 0).
- Neučinkovita raba podatkovnih tipov (če je le mogoče, je uporaba bytes32 bolj učinkovita od podatkovnega tipa niz).

- Neučinkovita uporaba indeksiranih parametrov.
- Neučinkovito delo z zunanjimi funkcijami.

Optimizirano različico pametnih pogodb se ponovno namesti v simulirano omrežje, kjer se ponovno oceni stroške izvajanja. Postopek se ponavlja, dokler rezultat stroškovne optimizacije ni zadovoljiv [7].

Identifikacija smrdeče kode preko analize programske kode velja za kompleksen in dolgotrajen postopek, ki terja za uspešno izvedbo opravila izkušenega strokovnjaka. Postopek identifikacije smrdeče kode pametnih pogodb, ki po nepotrebnem povečujejo stroške izvajanja, močno pospešijo orodja za optimizacijo porabe. Čeprav je ročno preverjanje kode mogoče, so avtomatizirana orodja pri tem učinkovitejša [32]. Orodje GasMet [7] opravi analizo programske kode pametnih pogodb napisanih v programskem jeziku Solidity, pri tem pa je sposobno zaznati 19 vzorcev, ki po nepotrebnem prispevajo k večji porabi goriva. Analiza, opravljena z orodjem, daje razvijalcem jasne indice, kako se lotiti optimizacije porabe pametnih pogodb. Na voljo sta tudi prototipa orodij GasChecker [33] in Gasper [34] , ki analizo pametnih pogodb opravita s simbolično izvedbo na nivoju zlogovne kode pametnih pogodb. Orodje GasChecker prepozna deset neučinkovitih programski vzorcev, Gasper jih prepozna sedem. Prav tako lahko za oceno stroškov goriva uporabljamo že omenjena orodja za testiranje, kot sta Hardhat in Foundry. Ne glede na pristop analize pametnih pogodb, orodja zgolj zaznajo potencialno problematično programsko kodo. Končna odločitev o ukrepih glede optimizacije porabe ostaja v pristojnosti razvijalcev.

4.3 Preverjanje varnosti

Pomemben vidik zagotavljanja kakovosti decentraliziranih aplikacij je tudi zagotavljanje varnosti pametnih pogodb. Ker so pametne pogodbe programske komponente, ki so nameščene na javno omrežje v verigah blokov in tako javno izpostavljene morebitnim napadalcem, je pravočasno odkrivanje ranljivosti ter preprečevanje vdorov nujno za njihovo dolgoročno in nemoteno delovanje.

Cilj varnostne analize je preveriti, ali so v programski kodi pametnih pogodb prisotne varnostne ranljivosti, ki predstavljajo tveganja za njihovo delovanje in bi lahko ogrozile dolgoročno vzdržnost izvajanja decentraliziranih aplikacij. Podobno kot optimizacija porabe pametnih pogodb, se tudi analiza varnostnih ranljivosti izvede s statično analizo programske kode in dinamično analizo obnašanja aplikacije med izvajanjem. V praksi je bilo prepoznanih več vzorcev, ki lahko ogrozijo nemoteno delovanje decentraliziranih aplikacij. Vidnejši med njimi so [8]:

- Ponovni vstop iste funkcije omogoča večkratno izvajanje, kar lahko pripelje do neveljavnih stanj podatkov.
- Neustrezna avtentikacija.
- Nepooblaščeno samouničenje pametnih pogodb-.
- Nezaščitene funkcije.
- Uporaba zastarelih funkcij.
- Nepreverjeni klici.
- Ranljivosti povezane z izgubo sredstev.

V okviru statične analize namenjene prepoznavi ranljivosti se v programski kodi pametnih pogodb identificira vzorce kodiranja, ki predstavljajo potencialne ranljivosti in bi jih napadalci lahko izkoristili v napadu [32]. Poleg prepoznavanja vzorcev povezanih z ranljivostmi, se statična analiza osredotoča na preučevanje strukture programske kode, na primer na analizo toka kontrole čez program, analizo podatkov, obravnavo napak in integracijo z uporabniškim vmesniki. Pri identifikaciji si je smiselno pomagati z orodji, ki olajšajo ročne preglede programske kode, saj jasno pokažejo na dele kode, ki prestavljajo morebiten vir varnostnih težav.

Dinamična analiza temelji na analizi programov z namenom odkrivanja ranljivosti, ki se razkrijejo šele med njihovim izvajanjem [32]. Tovrstna analiza običajno temelji na tehnikah testiranja, ki generirajo širok nabor naključno ali delno naključno ustvarjenih vhodnih podatkov [8]. Običajno se za to uporabi metoda testiranja fuzz. Generirani vhodni podatki so nato posredovani na vhode pametnih pogodb. Sledi natančno spremljanje obnašanja pametnih pogodb in detekcija morebitnega neželenega obnašanja, ki bi lahko predstavljalo varnostno tveganje, če bi se zgodilo v produkcijskem okolju med uporabo aplikacije. Takšno varnostno testiranja temelji na izvajanju pametnih pogodb v simuliranem okolju ali testnem omrežju z najrazličnejšim naborom vhodov, pri čemer se opazuje ustreznost delovanja pametnih pogodb med izvajanjem. Cilj metode je odkriti morebitne ranljivosti, izjeme ali nepričakovano obnašanje med izvajanjem. Pomemben pristop dinamične analize pametnih pogodb je prav tako analiza poti čez program pametnih pogodb. Ker temelji na dejanskem izvajanju program-

ske kode, dinamična analiza omogoča odkrivanje ranljivosti, ki so odvisne od specifičnih vhodnih podatkov, povezanih poti izvajanja in interakcij, ki nastanejo znotraj programa med delovanjem.

4.4 Revizije in presoje

Glavnina testiranja pri sodobnih pristopih razvoja programske opreme temelji na avtomatiziranih metodah. Orodja, ki jih za ta namen uporabljamo, so običajno specializirana in pokrivajo izbrane vidike kakovosti pametnih pogodb. V praksi pa se izkaže, da ta orodja ne zajemajo celotnega spektra ranljivosti in pomanjkljivosti programske kode. Pogosto zato niso sposobna zaznati tistih ranljivosti ali pomanjkljivosti, ki zahtevajo poznavanje širšega konteksta ali poglobljeno razumevanje delovanja aplikacije [5]. Presojevalci so zato pomemben steber zagotavljanja kakovosti decentraliziranih aplikacij, saj pomembno dopolnjujejo avtomatizirana orodja za testiranje in zapolnijo vrzeli, ki jih avtomatizirani postopki ne morejo pokriti, zlasti pri pregledu pametnih pogodb za prisotnost kompleksnejših ranljivosti.

Ključna razloga za izvedbo presoj programske kode pametnih pogodb sta dva. Prvič, s presojevalci se cilja predvsem na razkritje ranljivosti, ki jih orodja za avtomatizirano testiranje ne zaznajo. Presoje so pri razvoju decentraliziranih aplikacij razmeroma pogoste. Glede na študije o testiranju pametnih pogodb v odprtokodnih projektih kar 24,5 % preučenih projektov vključuje poročila o presojah, izvedenih s strani zunanjih presojevalcev [35]. Presojevalci, ki morajo biti izkušeni strokovnjaki za tehnologije veriženja blokov, izvedejo sistematičen ročni pregled programske kode, svoja dognanja strnejo v poročilu in podajo priporočila za izboljšave.

Drugi ključni razlog za izvedbo presoj izhaja iz dejstva, da pametnih pogodb ni mogoče enostavno popravljati, ko so te enkrat nameščene v omrežje verig blokov [36]. Pred namestitvijo se je namreč potrebno prepričati, da te delujejo brezhibno. Naloga presojevalcev je torej preveriti, ali pametne pogodbe izvajajo v skladu s pričakovanji uporabnikov brez nepričakovanih ali neželenih stranskih učinkov. Tovrstna presoja zajema celostno analizo ustreznosti izvajanja algoritmov, zapisanih v pogodbah, ter pravilno posodabljanje njihovih stanj. Presoje, zlasti tiste, ki jih izvedejo zunanji presojevalci, torej bistveno pripomorejo k zaupanju v kakovost implementacije pametnih pogodb.

5 SKLEP

Tehnologije veriženja blokov odpirajo možnosti inovacijam v poslovnih modelih na najrazličnejših domenah elektronskega poslovanja. Ključna prednost tehnologij veriženja blokov je možnost soustvarjanja deljenih zapisov, odpornih proti cenzuri, zlonamernim spremembam ali njihovem kasnejšem zanikanju. Z rastjo in razvojem tehnologij veriženja blokov se pojavlja potreba po programskih rešitvah, ki te ideje prenesejo v programsko kodo. V praksi se izkaže, da se pri razvoju decentraliziranih aplikacij pojavljajo posebnosti, ki jih je potrebno v okviru razvojnega procesa ustrezno nasloviti, da bi dosegli kakovostne in za uporabnika sprejemljive programske rešitve.

Posebno pozornost je potrebno nameniti aktivnostim zagotavljanja kakovosti, vključno s testiranjem. Razvijalci decentraliziranih aplikacij se v razvojnem procesu poslužujejo številnih metod in orodij, s katerimi premoščajo izzive razvoja tovrstnih aplikacij. Sistematično, temeljito in z avtomatizacijo podprto testiranje enot tekom razvoja zagotavlja, da se programska koda pametnih pogodb dodobra očisti programskih hroščev, preden je ta predana v produkcijsko okolje. Pri tem se zasleduje načelo programske kode brez hroščev. Orodja za identifikacijo slabih vzorcev in praks porabe goriva omogočajo optimizacijo programske kode pametnih pogodb, da ta pri izvajanju v javnih omrežjih porabi čim manj goriva in tako zniža operativne stroške distribuiranih aplikacij v sicer stroškovno dokaj dragem okolju. Orodja za prepoznavo varnostno ranljive programske kode predstavljajo steber zagotavljanja varnosti decentraliziranih aplikacij. Nenazadnje vseh izzivov, povezanih z zagotavljanjem kakovosti decentraliziranih aplikacij, ni mogoče nasloviti z uporabo orodij. Presoje programske kode pametnih pogodb, opravljene s strani usposobljenih strokovnjakov iz področja tehnologij veriženja blokov, dajejo dodatno zagotovilo, da so decentralizirane aplikacije grajene kakovostno in bodo v produkcijskem okolju upravičile pričakovanja uporabnikov.

Pri razvoju decentraliziranih aplikacij predstavlja uporaba sodobnih razvojnih metod, ki temeljijo na agilnih pristopih in DevOps, odprto raziskovalno področje. Agilni pristopi in DevOps temeljijo na hitrem razvoju, visoki stopnji avtomatizacije razvojnega procesa, inkrementalnem razvoju in sprotnem odzivanju na potrebe uporabnikov. Navedene lastnosti

sodobnih agilnih razvojnih procesov je pri razvoju decentraliziranih aplikacij težje doseči. Nespremenljivost pametnih pogodb, dodatna performančna testiranja in večja potreba po revizijah končnih različic programskih rešitev pred izdajami namigujejo na potrebo po prilagojenih razvojnih procesih. Ali je za razvoj kakovostnih decentraliziranih aplikacij primeren prilagojen agilni razvojni proces ali pa je v primeru razvoja tovrstnih rešitev najprimernejši slapovni procesni model, ostaja aktualna tema raziskav na tem področju. Kot prihodnje delo načrtujemo predstavitev v prispevku opisanih primerov iz prakse, kar bo dodatno dopolnilo teoretične koncepte.

LITERATURA

[1] M. J. Hossain Faruk *idr.*, »A Systematic Literature Review of Decentralized Applications in Web3: Identifying Challenges and Opportunities for Blockchain Developers«, *Proceedings - 2024 IEEE International Conference on Big Data, BigData 2024*, str. 6240–6249, 2024, doi: 10.1109/BIGDATA62323.2024.10826066.

[2] D. Macrinici, C. Cartofeanu, in S. Gao, »Smart contract applications within blockchain technology: A systematic mapping study«, *Telematics and Informatics*, let. 35, št. 8, str. 2337–2354, dec. 2018, doi: 10.1016/J.TELE.2018.10.004.

[3] »Hype Cycle for Web3 and Blockchain, 2024«. Pridobljeno: 15. maj 2025. [Na spletu]. Dostopno na: https://www.gartner.com/en/documents/5623191

[4] N. P. Imperius in A. D. Alahmar, »Systematic Mapping of Testing Smart Contracts for Blockchain Applications«, *IEEE Access*, let. 10, str. 112845–112857, 2022, doi: 10.1109/ACCESS.2022.3216874.

[5] R. Sujeetha in C. A. S. Deiva Preetha, »A Literature Survey on Smart Contract Testing and Analysis for Smart Contract Based Blockchain Application Development«, *Proceedings - 2nd International Conference on Smart Electronics and Communication, ICOSEC 2021*, str. 378–385, 2021, doi: 10.1109/ICOSEC51865.2021.9591750.

[6] L. Marchesi, M. Marchesi, R. Tonelli, in M. I. Lunesu, »A blockchain architecture for industrial applications«, *Blockchain: Research and Applications*, let. 3, št. 4, str. 100088, dec. 2022, doi: 10.1016/J.BCRA.2022.100088.

[7] A. Di Sorbo, S. Laudanna, A. Vacca, C. A. Visaggio, in G. Canfora, »Profiling gas consumption in solidity smart contracts«, *Journal of Systems and Software*, let. 186, str. 111193, apr. 2022, doi: 10.1016/J.JSS.2021.111193.

[8] Z. A. Khan in A. S. Namin, »A Survey of Vulnerability Detection Techniques by Smart Contract Tools«, *IEEE Access*, let. 12, str. 70870–70910, 2024, doi: 10.1109/ACCESS.2024.3401623.

[9] N. Six, N. Herbaut, in C. Salinesi, »Blockchain software patterns for the design of decentralized applications: A systematic literature review«, *Blockchain: Research and Applications*, let. 3, št. 2, str. 100061, jun. 2022, doi: 10.1016/J.BCRA.2022.100061.

[10] N. Sánchez-Gómez, L. Morales-Trujillo, J. J. Gutiérrez, in J. Torres-Valderrama, »The importance of testing in the early stages of smart contract development life cycle«, *Journal of Web Engineering*, let. 19, št. 2, str. 215–242, jun. 2020, doi: 10.13052/JWE1540-9589.1925.

[11] C. Castellon, S. Roy, P. Kreidl, A. Dutta, in L. Boloni, »Energy Efficient Merkle Trees for Blockchains«, *Proceedings - 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021*, str. 1093–1099, 2021, doi: 10.1109/TRUSTCOM53373.2021.00149.

[12] »Popoln vodnik za Ethereum«. Pridobljeno: 11. avgust 2025. [Na spletu]. Dostopno na: https://ethereum.org/sl/

[13] »Web3 Architecture and How It Compares to Traditional Web Apps - The New Stack«. Pridobljeno: 25. marec 2025. [Na spletu]. Dostopno na: https://thenewstack.io/web3-architecture-and-how-it-compares-to-traditional-web-apps/

[14] F. Wessling, C. Ehmke, M. Hesenius, in V. Gruhn, »How much blockchain do you need?: Towards a concept for building hybrid DApp architectures«, v *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, maj 2018, str. 44–47. doi: 10.1145/3194113.3194121.

[15] M. Farnaghi in A. Mansourian, »Blockchain, an enabling technology for transparent and accountable decentralized public participatory GIS«, *Cities*, let. 105, str. 102850, okt. 2020, doi: 10.1016/J.CITIES.2020.102850.

[16] »web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation«. Pridobljeno: 11. avgust 2025. [Na spletu]. Dostopno na: https://web3js.readthedocs.io/en/v1.10.0/

[17] »Ethers«. Pridobljeno: 11. avgust 2025. [Na spletu]. Dostopno na: https://docs.ethers.org/v5/

[18] H. Chu, P. Zhang, H. Dong, Y. Xiao, S. Ji, in W. Li, »A survey on smart contract vulnerabilities: Data sources, detection and repair«, *Inf Softw Technol*, let. 159, str. 107221, jul. 2023, doi: 10.1016/J.INFSOF.2023.107221.

[19] »GlassFish Wiki : GlassFishVsTomcat«. Pridobljeno: 11. april 2025. [Na spletu]. Dostopno na: https://javaee.github.io/glassfish/wiki-archive/GlassFishVsTomcat.html

[20] Y. Liu, F. R. Yu, X. Li, H. Ji, in V. C. M. Leung, »Blockchain and Machine Learning for Communications and Networking Systems«, *IEEE Communications Surveys and Tutorials*, let. 22, št. 2, str. 1392–1431, apr. 2020, doi: 10.1109/COMST.2020.2975911.

[21] V. Rajasekar, S. Sondhi, S. Saad, in S. Mohammed, »Emerging design patterns for blockchain applications«, *ICSOFT 2020 - Proceedings of the 15th International Conference on Software Technologies*, str. 242–249, 2020, doi: 10.5220/0009892702420249.

[22] X. Xu, C. Pautasso, L. Zhu, Q. Lu, in I. Weber, »A pattern collection for blockchain-based applications«, *ACM International Conference Proceeding Series*, jul. 2018, doi: 10.1145/3282308.3282312.

[23] P. van Vulpen, H. Heijnen, S. Mens, T. Kroon, in S. Jansen, »Upgradeable diamond smart contracts in decentralized autonomous organizations«, *Frontiers in Blockchain*, let. 7, str. 1481914, dec. 2024, doi: 10.3389/FBLOC.2024.1481914/BIBTEX.

[24] »Gas (Ethereum): How Gas Fees Work on the Ethereum Blockchain«. Pridobljeno: 8. avgust 2025. [Na spletu]. Dostopno na: https://www.investopedia.com/terms/g/gas-ethereum.asp

[25] »Understanding Ethereum Gas Fees in 2025: A Comprehensive Guide | KuCoin Learn«. Pridobljeno: 16. maj 2025. [Na spletu]. Dostopno na: https://www.kucoin.com/learn/web3/understanding-ethereum-gas-fees

[26] »Ethereum Average Gas Price Chart | Etherscan«. Pridobljeno: 16. maj 2025. [Na spletu]. Dostopno na: https://etherscan.io/chart/gasprice

[27] A. Vacca, A. Di Sorbo, C. A. Visaggio, in G. Canfora, »A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges«, *Journal of Systems and Software*, let. 174, str. 110891, apr. 2021, doi: 10.1016/J.JSS.2020.110891.

[28] N. Sánchez-Gómez, L. Morales-Trujillo, J. J. Gutiérrez, in J. Torres-Valderrama, »The importance of testing in the early stages of smart contract development life cycle«, *Journal of Web Engineering*, let. 19, št. 2, str. 215–242, jun. 2020, doi: 10.13052/JWE1540-9589.1925.

[29] »Foundry - Ethereum Development Framework«. Pridobljeno: 11. avgust 2025. [Na spletu]. Dostopno na: https://getfoundry.sh/forge/tests/overview/

[30] »Hardhat - Ethereum development environment«. Pridobljeno: 11. avgust 2025. [Na spletu]. Dostopno na: https://hardhat.org/hardhat-runner/docs/guides/test-contracts

[31] N. P. Imperius in A. D. Alahmar, »Systematic Mapping of Testing Smart Contracts for Blockchain Applications«, *IEEE Access*, let. 10, str. 112845–112857, 2022, doi: 10.1109/ACCESS.2022.3216874.

[32] A. Reyes, M. Jimeno, in R. Villanueva-Polanco, »Continuous and Secure Integration Framework for Smart Contracts«, *Sensors 2023, Vol. 23, Page 541*, let. 23, št. 1, str. 541, jan. 2023, doi: 10.3390/S23010541.

[33] T. Chen *idr.*, »GasChecker: Scalable Analysis for Discovering Gas-Inefficient Smart Contracts«, *IEEE Trans Emerg Top Comput*, let. 9, št. 3, str. 1433–1448, 2021, doi: 10.1109/TETC.2020.2979019.

[34] T. Chen, X. Li, X. Luo, in X. Zhang, »Under-Optimized Smart Contracts Devour Your Money«, *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, str. 442–446, mar. 2017, doi: 10.1109/SANER.2017.7884650.

[35] L. Palechor in C. P. Bezemer, »How are Solidity smart contracts tested in open source projects? An exploratory study«, *Proceedings - 3rd ACM/IEEE International Conference on Automation of Software Test, AST 2022*, str. 165–169, 2022, doi: 10.1145/3524481.3527228.

[36] Y. Huang, Y. Bian, R. Li, J. L. Zhao, in P. Shi, »Smart contract security: A software lifecycle perspective«, *IEEE Access*, let. 7, str. 150184–150202, 2019, doi: 10.1109/ACCESS.2019.2946988.

Mitja Gradišnik je asistent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Njegovo raziskovalno delo je usmerjeno v sodobne pristope k razvoju programskih rešitev, zagotavljanje kakovosti programskih produktov ter uporabo metod podatkovnega rudarjenja v programskem inženirstvu. Sodeluje pri številnih raziskovalnih in aplikativnih projektih, ki potekajo v okviru Inštituta za informatiko.

▪

Tina Beranič je docentka na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Doktorirala je leta 2018 iz tematike identifikacije pomanjkljive programske kode. Njeno raziskovalno delo obsega domeno kakovosti programske opreme, še posebej področje programskih metrik in mejnih vrednosti ter njihove uporabe za namen vrednotenja programske opreme. Ukvarja se tudi s področjem revizije informacijskih sistemov, pri čemer je leta 2017 pridobila certifikat CISA (Certified Information Systems Auditor) in leta 2020 naziv Preizkušena revizorka informacijskih sistemov na Slovenskem inštitutu za revizijo.

▪

Muhamed Turkanović je visokošolski učitelj, izredni profesor, na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Je vodja raziskovalne skupine Blockchain Lab:UM Inštituta za informatiko, namestnik predstojnika Inštituta za informatiko, vodja slovenskega EDIH-a DIGI-SI, vodja projektov H2020, Horizont Evropa, Interreg Alpine Space ter ARRS CRP. Njegovi trenutni raziskovalni interesi vključujejo področja tehnologij veriženja blokov, podatkovnih tehnologij ter digitalnih identitet.