Z umetno inteligenco podprt razvoj programske opreme

Mitja Gradišnik, Tina Beranič, Sašo Karakatič
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor
mitja.gradisnik@um.si; tina.beranic@um.si; saso.karakatic@um.si

Izvleček

Številni izzivi, na katere naletimo pri razvoju programskih rešitev, nas silijo, da neprestano iščemo nove pristope in prakse, s katerimi bi IT projekte realizirali boljše, hitreje in predvsem z nižjimi stroški. Želja po hitri in cenovno ugodni realizaciji IT projektov, višji stopnji njihove kakovosti ter nenazadnje v zadnjem času že kroničnem pomanjkanju usposobljenih IT strokovnjakov, so samo nekateri izmed izzivov, s katerimi se srečujemo v programskem inženirstvu. Pri naslavljanju omenjenih izzivov si v zadnjem času veliko obetamo od vpeljave umetne inteligence v proces razvoja programske opreme. Možnosti se kažejo predvsem v vpeljavi z naprednimi metodami umetne inteligence podprtih orodij, ki razvojno skupino razvijalcev aktivno podpirajo pri razvoju. Z umetno inteligenco podprta orodja odpirajo vrata odmiku od avtomatizacije ponavljajočih se trivialnih opravil in obljubljajo možnost avtomatizacije intelektualno zahtevnejših in kompleksnih opravil, kar bi občutno razbremenilo razvijalce informacijskih rešitev.

Ključne besede: proces razvoja programske opreme, orodja razvoja programske opreme, umetna inteligenca

Abstract

In software engineering, many of the challenges we face during software development force us to constantly look for new approaches and practices to help deliver better and faster IT projects at lower costs. The desire for a fast and affordable implementation of IT projects, a higher level of quality of solutions and the chronic shortage of skilled IT developers are just some of the challenges we face. Recently, in addressing these challenges, the introduction of artificial intelligence into the software development process has shown a lot of promise especially in the form of advanced artificial intelligence-supported tools that actively support the development team in the development process. Artificial intelligence-enabled tools offer abilities beyond the automation of trivial repetitive tasks. Primarily, they introduce the automation of more complex tasks, the execution of which was exclusively in the domain of skilled IT professionals until recently.

Keywords: Software development process, software development tools, artificial intelligence, intelligent assistants.

1 UVOD

Podjetja, ki delujejo v sodobnem poslovnem okolju, se morajo skozi proces digitalne preobrazbe neprestano prilagoditi spremenjenim pričakovanjem digitalno vedno bolj ozaveščenih strank. Te postajajo iz leta v leto bolj naklonjene interakciji preko digitalnih medijev, vse pogosteje posegajo po spletnih nakupih, prav tako dostopajo do vrste storitev preko digitalnih kanalov (Cuesta, Ruesta, Tuesta, & Urbiola, 2015). Za številne banke po svetu postaja spletno in mobilno bančništvo vsaj toliko pomembno, če že ne pomembnejše, kot poslovanje preko mreže lastnih izpostav in bančnih avtomatov (Ross & Srinivas, 2018).

Pravočasno in ustrezno digitalno preoblikovanje podjetij postaja posledično eden izmed ključnih strateških ciljev, kateremu so podjetja v zadnjih letih predana. Nepravočasno prilagajanje potrebam in pričakovanjem strank bi se lahko tako hitro prelilo v zamujene poslovne priložnosti. Kljub občutnemu premiku v smeri digitalizacije storitev opravljenem v zadnjih letih, gospodarstvo še zmeraj ne izkoriščajo polnega potenciala, ki ga prinaša vpeljava sodobnih informacijsko-komunikacijskih tehnologij. Po trenutnih ocenah izkoriščajo Združene države Amerike 18 % potenciala, ki ga prinaša digitalizacija, evropske države v povprečju le 12 % (Bughin idr., 2016). Di-

gitalna preobrazba podjetja prav tako naslavlja procese notranje optimizacije delovanja podjetja, v okviru katerega se morajo podjetja priučiti veščin, kako izkoristiti množico razpoložljivih podatkov ter na kakšen način bi te podatke s pomočjo sodobnih informacijsko-komunikacijskih rešitev uporabiti, da bi z njihovo pomočjo bolje razumeli potrebe strank, trgov, konkurence in aktualne trende na trgu (Cagle, 2019).

Četudi podjetja uspejo pripraviti strategije digitalne preobrazbe, ki zmorejo učinkovito nasloviti aktualne potrebe na trgu, pogosto naletijo na izzive pri njeni realizaciji. Izkoriščanje potencialov digitalne preobrazbe čedalje bolj ovira pomanjkanje ustrezno usposobljenih kadrov inženirskih smeri, ki so ključni za uspešno realizacijo tovrstnih IT projektov. Kot kažejo raziskave analitičnih hiš Gartner (Griffin, Mok, Struckman, & Berry, 2018) in Deloitte (Giffi idr., 2018), usposabljanje potrebnih IT kadrov trenutno ne sledi potrebam gospodarstva. Problem pomanjkanja ustrezno usposobljenega inženirskega kadra je sicer mogoče zaznati dalj časa, projekcije trendov pa nakazujejo, da se bo stanje z leti še dodatno poslabšalo (Deming & Noray, 2018). Na izzive pomanjkanja usposobljenega inženirskega kadra je sicer potrebno gledati kot na širši družbeni problem, katerega morajo ustrezno nasloviti tako državni organi preko izobraževalnih programov kot privatni sektor s spodbujanjem vseživljenjskega učenja zaposlenih. A vendar enostavnih rešitev, ki bi čez noč odpravile težave pomanjkanja usposobljenega IT kadra, kratkoročno ne gre pričakovati. Ključni izziv tako ostaja, kako z obstoječimi kadri opraviti večjo količino dela.

V zadnjih letih si pri reševanju predstavljene problematike posebej veliko obetamo od razvoja tehnologij umetne inteligence. Te so v zadnjih letih dosegle zrelostni nivo, pri katerem tehnologije niso uporabne zgolj v izoliranih laboratorijskih okoljih, temveč jih je mogoče aplicirati v konkretne primere iz realnega življenja. Z razvojnimi orodji podprtimi s tehnologijami umetne inteligence je mogoče uspešno nasloviti aktualne probleme, ki spremljajo proces razvoja programske opreme. Tega namreč v veliki meri sestavljajo intelektualno zahtevna opravila, ki v veliki meri terjajo od razvijalcev mentalni napor. Z razvojnimi orodji podprtimi z metodami umetne inteligence bi lahko presegli nivo trenutne avtomatizacije mentalno nezahtevnih in pogosto ponavljajočih se opravil. V razvojni proces IT bi tako vpeljali višji nivo avtomatizacije, s katero bi bilo mogoče razvijalce pri razvoju IT rešitev razbremeniti tudi intelektualno zahtevnejših opravil. Slednje bi imelo za posledico učinkovitejši in bolj optimalni razvojni proces informacijskih rešitev, s katerim bi lažje naslavljali izzive, ki jih prinaša pospešena digitalna transformacija podjetji, kateri smo priča v zadnjih letih.

2 UMETNA INTELIGENCA V ORODJIH ZA RAZVOJ PROGRAMSKIH REŠITEV

2.1 Pristopi in metode umetne inteligence

Umetna inteligenca je v zadnjih letih na polno vstopila v naša vsakodnevna življenja. Napovedani avtonomni avtomobili in pametni asistenti, kot so Siri, Alexa in Google, so zgolj najvidnejši predstavniki tehnologij, ki jih najpogosteje povezujemo z umetno inteligenco. Poleg navedenih izpostavljenih primerov je mogoče umetno inteligenco aplicirati na številna druga področja. Termin umetna inteligenca je sicer uporablja zgolj krovni in zelo posplošen izraz, pod katerim je združenih več družin med seboj zelo različnih pristopov in metod, ki so namenjeni reševanju najrazličnejših vsakodnevnih izzivov (Gradišnik, Karakatič, Mauša, Beranič, & Heričko, 2019).

Na področju programskega inženirstva je mogoče vključiti najrazličnejše metode umetne inteligence in jih uporabiti za reševanje širokega spektra aktualnih izzivov znotraj razvojnega cikla programske opreme. Področje umetne inteligence aktualno za programsko inženirstvo v grobem zajema različne pristope stojnega in globokega učenja, predstavitve znanja z grafi, semantičnimi mrežami in poslovnimi pravili, obdelavi naravnega jezika in avtomatizirani razpoznavi elementov v slikovnem ter avdio in video materialu (Lo Giudice, 2016). V bližnji prihodnosti se pričakuje se, da bo kombinacija navedenih metod umetne inteligence imela močan vpliv na vse stopnje razvojnega cikla programske opreme, in sicer tako, da bo razvijalcem omogočen razvoj boljše programske opreme v krajšem času (Lo Giudice, 2016). Metode umetne inteligence se na področje programskega inženirstva aplicirajo preko naprednih razvojih orodij, za katera je značilno, da imajo karakteristike intelektualnega procesa značilnega za ljudi, kot je na primer zmožnost sklepanja, odkrivanja pomena, generalizacije in učenja iz preteklih izkušenj (Copeland, 2019).

2.2 Repozitoriji projektov programskih produktov

Kljub čedalje večji raznovrstnosti in dovršenosti algoritmov umetne inteligence ti sami po sebi niso

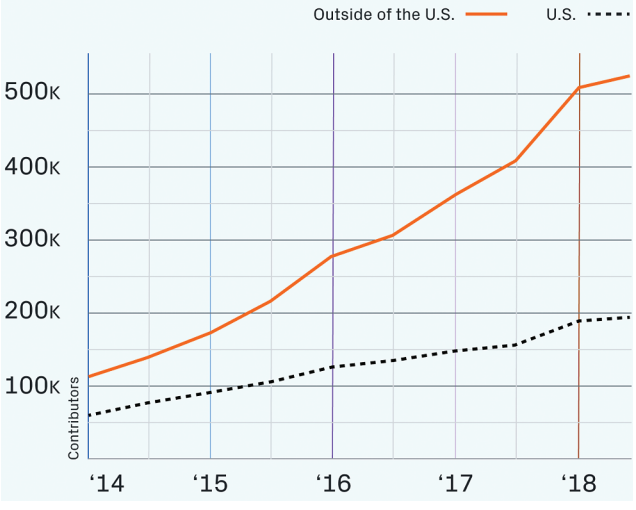
dovolj, da bi z njimi uspešno reševali predhodno izpostavljene izzive programskega inženirstva. Da izdelamo z umetno inteligenco podprta razvojna orodja, so poleg dodelanih algoritmov umetne inteligence potrebna zbirka kakovostnih učnih podatkov, nad katerimi je algoritme umetne inteligence mogoče učinkovito učiti. V praksi se dostopni repozitoriji programskih projektov, kot so na primer GitHub (»GitHub«, 2019), GitLab (»GitLab«, 2019), Bitbucket (»Bitbucket«, 2019a) in SourceForge (»SourceForge«, 2019), izkažejo za neprecenljiv vir znanja s področja programskega inženirstva. Repozitoriji programskih projektov so v osnovi virtualna okolja, znotraj katerih razvojne skupine v okviru aktivnosti razvoja programske opreme soustvarjajo raznovrstne artefakte potrebne za realizacijo programskih produktov (Güemes-Peña, López-Nozal, Marticorena-Sánchez, & Maudes-Raedo, 2018). Repozitoriji programskih projektov so po več desetletij splošne uporabe prerasli v ogromne zbirke najrazličnejših programskih projektov, ki jih lahko uporabimo kot temelj za izgradnjo s strojnim učenjem podprtih razvojnih orodij.

Poleg repozitorijev programskih projektov so pomemben vir domenskega znanja prav gotovo tudi spletne skupnosti za pomoč med razvijalci. Pri vidnejših predstavnikih tovrstnih skupnosti zagotovo ne moremo mimo spletne skupnosti razvijalcev StackOverflow (»StackOverflow«, 2019). Ta namreč ponuja ogromno bazo relevantnih razvojnih izzivov in njihovih rešitev, ki jih je mogoče učinkovito uporabiti pri izgradnji modelov v strojnem učenju.

Kot odziv na možnosti, ki jih prinaša analiza navedenih spletnih skupnosti in orodij, je vzniknila relativno mlada raziskovalna veja rudarjenja programskih repozitorijev (angl. Software Repository Mining), ki se ukvarja s sistematičnimi pristopi pridobivanja in analiziranja podatkov dostopnih v programskih repozitorijih (Kaur, Kaur, Chopra, & Kaur, 2020). V ospredju rudarjenja programskih repozitorijev ni zgolj izvorna koda programskih projektov, temveč vsi artefakti, ki nastajajo v okviru procesa razvoja programske opreme. Torej tudi zapisi v repozitoriju za sledenje programskim hroščem, sistemu za verzioniranje programske kode in različnih kanalih komunikacije med razvijalci, na primer v dopisnih seznamih (angl. mailing lists). Na podlagi relevantnih podatkov, izluščenih iz repozitorijev programskih projektov je mogoče razpoložljive algoritme umetne inteligence naučiti, da so ti zmožni samostojnega reševanja problemov, na katere je moč naleteti med razvojem programske opreme, kar pa predstavlja temelj novodobnih s strani umetne inteligence podprtih razvojnih orodij.

O možnostih, ki jih rudarjenje repozitorijev projektov programskih produktov za razvoj inteligentnih razvojnih orodij prinaša, zgovorno povedo številke o uporabnikih in gostujočih repozitorijih, ki so jih uporabnikih tovrstnih skupnosti ustvarili v zadnjih desetletjih. GitHub, z naskokom največji med navedenimi, je v avgustu 2019 poročal o več kot 50 milijonih uporabnikov, ki se učijo, si med seboj delijo in soustvarjajo več kot 100 milijonov repozitorijev programskih produktov (GitHub, 2020a, 2020b). Intenzivna je tudi rast uporabnikov, ki prispevajo programsko kodo. Graf rasti uporabnikov sicer med leti 2014 – 2018 prikazuje Slika 1. Za razliko, nekoliko manj uporabnikov beležijo pri konkurenčnih spletnih skupnostih Bitbucket. Njihova oblačna storitev je v aprilu 2019 poročala o 10 milijonih registriranih uporabnikov, ki so takrat upravljali več kot 28 milijonov repozitorijev projektov programskih produktov (Bitbucket, 2019b).

Programski projekti, ki se nahajajo v omenjenih repozitorijih, so strukturno izredno heterogeni. Temeljijo namreč na različnih programskih jezikih, uporabljajo najrazličnejša razvojna ogrodja in so spisana za široko paleto ciljnih naprav. Slednje potrjujte analiza spletne aplikacije GitHut 2.0 (Zapponi, 2020, ki kot najpogosteje uporabljen programski jezik na platformi GitHub navaja programski jezik JavaScript,



Slika 1: **Graf rasti novih uporabnikov na platformi GitHub v letih 2014–2018 (GitHub, 2020a)**

ki mu sledita Python in Java. Razvrstitev z deležem in njegovo spremembo glede na prejšnje četrtletje desetih najpogostejših programskih jezikov platforme GitHub za obdobje prvega četrtletja 2020 prikazuje Tabela 1. Heterogenost projektov sicer pripomore k lažjemu prilagajanju razvojnih orodij različnim okoljem in razvojnim procesom, v okviru katerih se ta uporabljajo.

Tabela 1: **Deset najpogostejših programskih jezikov na platformi GitHub za obdobje prvega četrtletja 2020 z navedenimi deleži programskih jezikov in spremembo deleža glede na prejšnje četrtletje**

mesto uvrstitve	programski jezik	delež	sprememba
1	JavaScript	18,7 %	–1,41
2	Python	16,2 %	–1,65
3	Java	10,9 %	+0,53
4	Go	9,0 %	+0,98
5	C++	7,4 %	+0,04
6	Ruby	6,8 %	+0,34
7	TypeScript	6,7 %	+1,52
8	PHP	5,1 %	–0,46
9	C#	3,8 %	+1,14
10	C	3,2 %	–0,20

2.3 Samoučeča razvojna orodja

Z uporabo strojnega učenja in podatkov, pridobljenih z rudarjenjem razpoložljivih baz znanja, ki jih soustvarjamo v sklopu omenjenih spletnih skupnosti in repozitorijev, je mogoče zgraditi razvojna orodja, ki predstavljajo močan odmik od trenutno uveljavljenih razvojnih orodij. Ta namreč temeljijo na vnaprej kodiranih pravilih obnašanja in strategijah reševanja problemov. Za zagotavljanje skladnosti obnašanja razvojnih orodij s pričakovanji uporabnikov skrbijo razvijalci razvojnih orodij, ki morajo vsako nadgradnjo ali spremembo obnašanja programski kodi popraviti ročno. Razvojna orodja, ki temeljijo na pristopu strojnega učenja se temu izognejo. Izpeljava potrebnih pravil, ki pripeljejo do rešitve zastavljenega cilja, je v celoti prepuščena procesu strojnega učenja, ki je pri tem avtonomen. Od orodij za razvoj programske opreme naslednje generacije si obetamo odpravo deterministčnega in vnaprej programiranega obnašanja. Od tovrstnih orodij pričakujemo zmožnost samodejnega učenja reševanja razvojnih izzivov in samodejno prilagajanje okoliščinam dela.

3 PODPORA UMETNE INTELIGENCE RAZVOJNEMU CIKLU PROGRAMSKE OPREME

Razvoj programske opreme velja za kompleksen proces, uspešnost katerega je odvisna od različnih faktorjev, pomembnejši med njimi so zagotovo razvojna skupina, vrsta programskega produkta v razvoju, stabilnosti funkcionalnostnih zahtev, izbira programskega jezika in arhitekture programske rešitve (Güemes-Peña idr., 2018). Realizacija projektov programskih produktov torej terja skupino dobro izurjenih strokovnjakov iz področja programskega inženirstva, od katerih se ob dobro razvitih komunikacijskih sposobnostih in obvladovanju dela v skupini pričakujejo predvsem močne razvite sposobnosti analitičnega razmišljanja ter zmožnost reševanja problemov, na katere naletijo med razvojem programskih produktov. Potreba po navedenih veščinah izhaja iz narave dela, ki terja od razvijalcev programske opreme v prvi vrsti vložen miselni napor. In ravno razbremenitev stopnje miselnega napora, ki ga morajo razvijalci vložiti v svoje delo, ostaja osrednja točka razvoja naprednih z umetno inteligenco podprtih razvojnih orodij.

Zaradi miselno intenzivne narave dela tekom razvoja programske opreme so možnosti uporabe orodij dokaj omejene, običajno na trivialna in pogosto ponavljajoča se opravila. Večino miselno zahtevnih opravil tako še vedno opravijo razvijalci. Čez leta smo bili sicer priča postopni evoluciji pristopov in orodij, sami temelji le-teh pa se čez leta niso kaj dosti spreminjali. Posledično lahko opazimo, da se pristopi programskega inženirstva in orodja, ki se pri tem uporabljajo, od samih začetkov programskega inženirstva, torej vse od vpeljave programskih jezikov Fortran in Lisp sredi petdesetih let prejšnjega stoletja, konceptualno niso korenito spremenili (Lorica & Loukides, 2018). Z večjimi ali manjšimi variacijami razvijalci povečini uporabljajo urejevalnike, v katerih urejajo izvorno kodo programov.

3.1 Podpora orodij razvoju programskih rešitev

Ker osrednji del celotnega procesa razvoja programskih rešitev še vedno predstavlja pisanje in urejanje programske kode, je to točka razvojnega procesa programske opreme, ki ponuja največ možnosti za povečanje učinkovitosti razvoja. Skrajšanje časa, v katerem razvijalci pridejo do potrebne programske kode, predstavlja enega izmed ključnih prispevkov k optimizaciji razvojnega procesa. Ključni izziv, ki jih

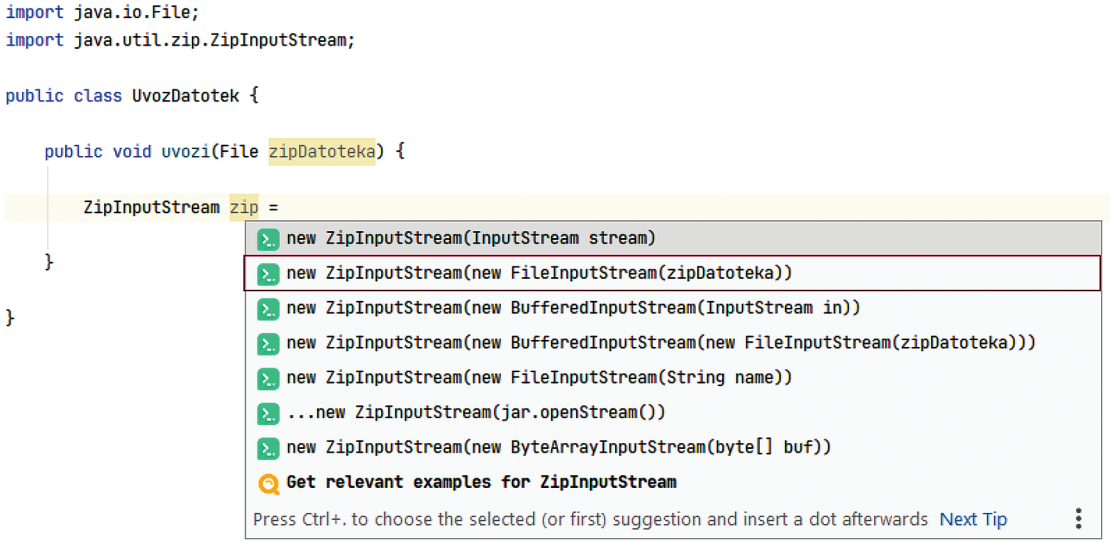
na tem mestu rešujejo z umetno inteligenco podprta orodja, je, kako čim hitreje do relevantne programske kode, ki je prilagojena kontekstu zahtev uporabnikov in potreb naročnikov (Gradišnik, Karakatič, idr., 2019).

3.1.1 Inteligentni asistenti

Pomembno skupino z umetno inteligenco podprtih razvojnih orodij, ki so iz idejnih zasnov že prešle v konkretne rešitve, predstavljajo inteligentni asistenti razvijalca. Njihova primarna naloga je, da v realnem času glede na kontekst programskega problema razvijalcu nudijo relevantne predloge blokov programske kode, priporočila in dobre prakse (Yao, 2018). Tovrstna razvojna orodja presegajo že uveljavljene rešitve predlagalnikov sledečega žetona programske kode. Njihovi predlogi zajemajo celostno rešitev v obliki sintaktično pravilnega bloka programske kode, ki ga je v danem kontekstu mogoče uporabiti.

Inteligentni asistenti so v pomoč predvsem, ko se razvijalci znajdejo na obsežnih in kompleksnih projektih, na katerih se morajo za nadgradnjo ali popravek posameznih funkcionalnosti prebiti čez številne vrstice programske kode. Te po možnosti niti niso napisane v programskem jeziku, ki bi jim bil povsem domač. Brskanje za rešitvami razvojnih problemov, dostopnih v najrazličnejših virih, je torej del dnevne rutine razvijalcev (Nguyen, Di Rocco, & Di Ruscio, 2018)software developers frequently look up external sources for related information. Consulting data available at open source software (OSS. Inteligentni asistenti torej razvijalcem v takšnih situacijah nudijo asistenco na mestu, ko bi ta moral prekiniti programiranje z namenom, da v dokumentaciji ali katerem izmed spletnih virov poišče rešitev problema, ki bi mu omogočila nadaljevanje dela. Ključna naloga inteligentnega asistenta je torej ponuditi izsek programske kode, ki razvijalcu omogoči takojšnje nadaljevanje dela (Gradišnik, Tina, & Karakatič, 2019). Ponujena rešitev temelji na analizi podobnih blokov programske kode učnih projektov, pridobljenih iz analiziranih repozitorijev projektov programske opreme.

Pristop razvoja programske kode, pri katerem razvijalcu z relevantnimi predlogi stoji ob strani inteligentni asistent, bi lahko primerjali s predvsem v agilnih metodah razvoja dobro uveljavljenim principom programiranja v paru. Pri razvoju programske kode nad skupnim programskim problemom delata dva razvijalca, od katerih prevzame eden aktivno vlogo pisanja programske kode, drugi razvijalcev v paru pa spremlja delo prvega ter z aktivnim spremljanjem in predlogi skrbi za učinkovitejši in bolj kakovosten potek dela. Lahko bi rekli, da s tem ohranimo učinkovitost in raven kakovosti razvite kode, hkrati pa se izognili neekonomičnosti programiranja v paru z dvema razvijalcem, kar je eden ključnih pomislekov pri njegovi vpeljavi v praksi (Handy, 2018). Slika 2 prikazuje primer predlogov inteligentnega asistenta Codota. Ta razvijalcu predlaga pomensko pravilne in popolne bloke programske kode, ki povežejo dva v kodi s strani razvijalca uporabljena razreda za delo s tokovi.



Slika 2: **Predlogi blokov kode pametnega asistenta Codota nameščenem v razvojnem okolju IntelliJ IDEA**

Trenutno ima namreč vsak izmed razpoložljivih inteligentnih asistentov platformo, v kateri je dominanten. Inteligentni asistent Codota (»Codota«, 2019) nudi asistenco razvijalcem, ki razvijajo rešitve s pomočjo programskega jezika Java in njemu sorodnem jeziku Kotlinu. Ob podpori programskemu jeziku Java se lahko uporabniki inteligentnega asistenta Codota nadejajo podpore programskemu jeziku JavaScript, kateremu podpora se nahaja v beta fazi razvoja in bo uporabnikom na voljo kmalu. Za razvijalce, ki prisegajo na tehnološke rešitve iz Microsoftovega tabora, bo najprimernejši inteligentni asistent IntelliCode (Microsoft, 2019). Microsoftovo rešitev IntelliCode odlikuje široka podpora programskim jezikom s poudarkom na programskem jeziku C# ter odlična integracija v Microsoftovi integrirani razvojni okolji Visual Studio in Visual Studio Code. Kite (Kite, 2019), kot tretji izmed vidnejših predstavnikov inteligentnih asistentov, cilja na iz leta v leto številčnejšo populacijo Python razvijalcev (Tiobe, 2020). Kljub trenutni specializiranosti inteligentnih asistentov v posamezne programske jezike se v prihodnje kaže jasen trend, da inteligentni asistenti presežejo specializacijo v posamezne programske jezike in postanejo bolj univerzalni s tem, da ponudijo podporo širši paleti programskih jezikov. Strnjena primerjava navedenih inteligentnih asistentov prikazuje Tabela 2.

Vse navedene inteligentne asistente je možno namestiti kot vtičnike integriranih razvojnih okolij za platforme programskih jezikov, na katere ti ciljajo. S tem inteligentni asistenti dobijo vpogled v delo razvijalca in dosežejo, da so razvijalcu tekom razvoja programske kode res ves čas razvoja na voljo. V trenutni fazi zrelosti inteligentnih asistentov si lahko razvijalci od le-teh obetajo predloge blokov programske kode, ki bi jih lahko v neki točki pisanja programske kode uporabili. Predlogi, ki jih dajejo inteligentni asistenti, presegajo enostavno dopolnjevanje kode. Glede na kontekst programskega problema so predlogi inteligentnih asistentov kompleksnejše strukture programske kode, do katerih so se predstavljeni inteligentni asistenti dokopali s strojnim učenjem bodisi nad programsko kodo odprtokodnih projektov repozitorijev projektov programske opreme GitHub in Bitbucket, bodisi kot v primeru inteligentnega asistenta Codota nad programsko kodo objavljeno v razpravah med razvijalci v spletni skupnosti StackOverflow.

3.1.2 Semantični iskalniki po programski kodi

Dopolnjevanje in izpopolnjevanje obstoječih sistemov terja veliko iskanja po programski kodi, pri čemer je popolnoma običajno, da večji programski produkti obsegajo več kot 100.000 vrstic programske kode. Pri tem, ko razvijalci razmišljajo v naravnem jeziku, je programska koda, po kateri iščejo, napisana v programskem jeziku. Razvijalec mora torej v iskalnik vpisati konkretni izsek iskalne programske kode, kar pa zna biti problematično, če se z iskano programsko kodo predhodno še ni srečal. Razkorak med naravnim in programskim jezikom pri iskanju po programski kodi premošča rešitev Microsoftove podružnice GitHub poimenovan Semantic Search (Husain & Wo, 2018). Rešitev stavke programske kode pretvori s pomočjo vektorjev preslikav v naravni jezik. Na podlagi primerjave sorodnosti vektorjev programske kode in iskalnih nizov je nato mogoče najti tisto programsko kodo hranjeno v javnih repozitorijih GitHub portala, ki najbolj ustreza vnesenemu iskalnemu nizu.

Tabela 2: **Strnjena primerjava inteligentnih asistentov**

Inteligentni asistent	Podprti programski jeziki	Podpora v prihodnje	Podprta razvojna orodja	Viri strojnega učenja
Codota	Java Kotlin	JavaScript *(beta)*	Intellij IDEA, Android Studio, Eclipse IDE	GitHub, Bitbucket, StackOverflow, Iskalnik Google
Kite	Python	*V fazi izbire podpore naslednjega programskega jezika*	IntelliJ, PyCharm, VS Code, Atom, Vim, Sublime, Spyder	Javni repozitoriji na GitHub-u
Microsoft IntelliCode	C#, C++, Python, JavaScript, TypeScript, XAML, Java.	/	Visual Studio 2019 (C#, C++, JavaScript, XAML), Visual Studio Code (Java, Python)	GihHub (za C# več kot 3000 najboljših odprtokodnih repozitorijev), sicer na zahtevo uporabnikov tudi privatni repozitoriji uporabnika

3.1.3 Generiranje programske kode na podlagi skic uporabniškega vmesnika

Razvijalci pogosto na podlagi predhodno pripravljenih skic uporabniškega vmesnika, ki jih pripravijo grafični oblikovalci v procesu zajema zahtev stranke, sprogramirajo ustrezno izvršljivo programsko kodo. Prihranek pri porabljenem času za razvoj obljublja orodje Sketch2Code (»Sketch2Code«, 2019), ki je zmožno iz ročno skicirane risbe uporabniškega vmesnika generirati ustrezno HTML programsko kodo vključno s pripadajočimi kaskadnimi stilskimi predlogami (CSS). Ključna prednost orodja je, da ne potrebuje modela ali natančno izrisane slike prototipa, temveč zadostuje le njegova skica (Kumar, 2018). Primer pretvorbe prikazuje Slika 3. Tako generirana programska koda lahko razvijalce pripelje bodisi do hitrega prototipa rešitve bodisi je uporabljena kot skelet programske kode končne rešitve. Opisan pristop generiranja uporabniškega vmesnika so nadgradili s pomočjo tri-slojnega sistema umetne inteligence (Moran, Bernal-C'Ardenas, Curcio, Bonett, & Poshyvanyk, 2018). Pri tem za učenje orodja seveda niso uporabili znanja ekspertov, temveč javno dostopne repozitorije programske kode.

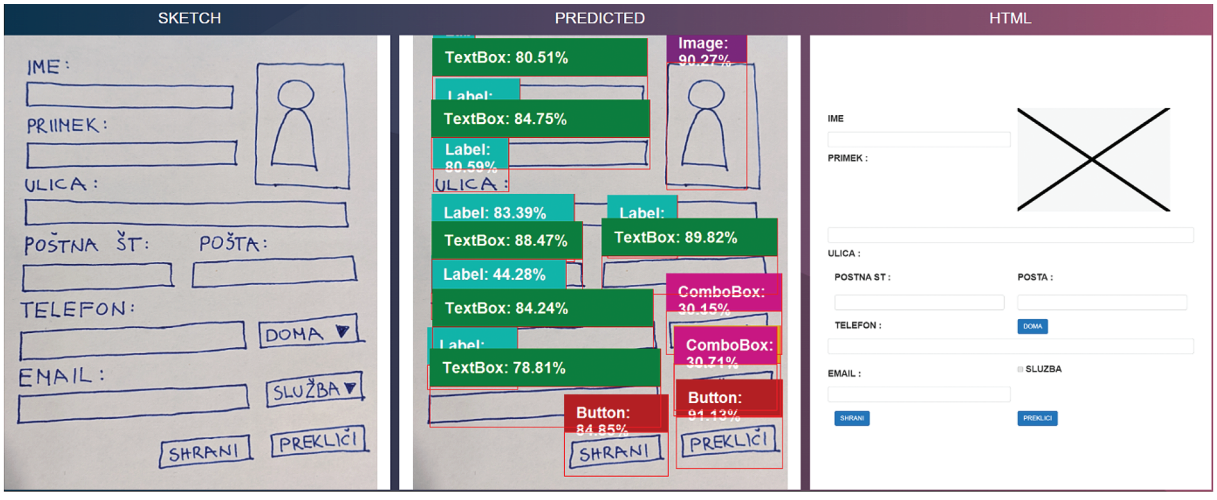
4 IZBIRA USTREZNEGA RAZVOJNEGA ORODJA

4.1 Izbira orodij

Čeprav se na prvi pogled zdi, da je trenutno na trgu več alternativnih inteligentnih asistentov, podrobnejši pregled razkrije, da so razvijalci pri izboru razpoložljivih razvojnih orodij relativno omejeni. Predstavljena orodja v tem trenutku še ne nudijo prav visoke stopnje univerzalnosti in so bolj ali manj vezana na ozko usmerjena opravila, ki so dodatno vezana na izbrane programske jezike oz. programske platforme.

Kot primer vzemimo orodje Sketch2Code, ki pretvarja skice uporabniškega vmesnika v izvršljivo programsko kodo, zapisano v obliki označevalnega jezika HTML. Čeprav je lahko vhod v orodje poljubna ročno narisana skica uporabniškega vmesnika, je izhod orodja ozko usmerjen v razvijalce spletnih rešitev. Razvijalci namiznih ali mobilnih aplikacij si z orodjem ne bi mogli veliko pomagati, poiskati bi morali katero izmed alternativ.

Podobno omejeni so uporabniki inteligentnih asistentov. Podroben pogled na področje razkrije, da ključni igralci na področju naslavljajo določene skupine razvijalcev. In sicer, rešitev Codota v prvi vrsti naslavlja razvijalce programskih jezikov Java in sorodni Kotlin, s katerimi ti razvijajo zelo širok spekter javanskih in android aplikacij. Rešitev Kite primarno naslavlja iz leta v leto večjo skupnost Python razvijalcev, medtem ko IntelliCode naslavlja predvsem razvijalce, ki svoje rešitve temeljijo na Microsoftovem tehnološkem skladu in razvojnem okolju Visual Studio. Kljub začetni ozki usmerjenosti inteligentnih asistentov na ciljano skupino razvijalcev se za prihodnost kažejo njihove močne tendence, da bi postali univerzalni in primernejši za širši krog razvijalcev. Trenutno razpoložljivi inteligentni asistenti se torej primarno osredotočajo na programske jezike, razvijalci pa ostajajo zaprti znotraj okvirjev razpoložljivih rešitev, ki so za njihovo razvojno platformo na voljo.



Slika 3: **SketchCode iz skice grafičnega vmesnika prepozna elemente vmesnika in na podlagi prepoznave generira HTML kodo**

Pri tem je popolnoma vseeno, za kakšen tip aplikacij jih bodo razvijalci uporabili. Iz vidika uporabe inteligentnih asistentov je sicer popolnoma vseeno, ali razvijalec razvija namizno, spletno ali mobilno aplikacijo, dokler pri tem uporablja s strani inteligentnega asistenta podprt programski jezik.

4.2 Omejitve z umetno inteligenco podprtih orodij

V raziskovalni skupnosti na področju umetne inteligence že desetletja sledijo viziji, da bi s pomočjo umetne inteligence izgradili sisteme, ki bi bili zmožni avtonomno spisati računalniške programe (Balog, Gaunt, Brockschmidt, Nowozin, & Tarlow, 2019). Čeprav se v povezavi s prehodno predstavljenimi orodji pogosto uporablja besedna zveza »umetna inteligenca«, praktični preizkus razpoložljivih orodij pokaže, da so njihove zmožnosti še zmeraj zelo omejene.

Inteligentni asistenti na primer pogosto ponudijo več alternativnih rešitev. Razvijalcem so torej ponujene možne alternative rešitve in ne optimalne rešitve programskega problema v konkretnem kontekstu programskega produkta. Razumevanje programske kode se med razvijalcem in inteligentnim asistentom pomembno razlikuje. Inteligentni asistenti so v trenutni fazi razvoja zmožni razumeti zgolj ozki okvir pisanja programske kode v dometu nekaj vrstic predhodno napisane programske kode. Posledično ponudi tiste bloke programske kode, za katere obstaja glede na projekte iz učnih primerov statistično gledano največja verjetnost, da se bodo v nadaljevanju uporabili v programski kodi. Razvijalec mora na drugi strani, da je sploh sposoben realizirati programsko rešitev v skladu s pričakovanji naročnika, razumeti popolni kontekst programskega produkta, vključno z namenom uporabe programske rešitve, pričakovanim obnašanjem funkcionalnosti in željami in pričakovanji uporabnikov. Za razliko od inteligentnih asistentov razvijalec torej opremljen z razumevanjem celotnega konteksta razvoja programske rešitve razume končni cilj procesa, zaradi česar se je med možnimi alternativnimi zmožen odločiti za tiste rešitve, ki so za dani kontekst najbolj optimalne.

Kljub temu prestavlja sinteza znanja iz razpoložljivih baz znanja in priprava alternativ rešitev, ki jo opravijo inteligentni asistenti, velik prispevek k optimizaciji razvoja programske kode in je ne gre kar zanemariti. Že en sam klic programskega vmesnika, ki razvijalcu ni popolnoma poznan, pomeni prekinitev pisanja kode in posledično njegovo brskanje po virih. Z uporabo inteligentnih asistentov je takšnega brskanja za informacijami pri programskega jezika ne popolnoma veščih razvijalcih občutno manj, vnaprej pripravljeni bloki kode pa tudi prihranijo marsikateri pritisk na tipkovnico.

5 SKLEP

Razvoj spletne tehnologij v preteklih desetletjih na čelu s spletnimi forumi, blogi in drugimi spletnimi skupnostmi je pospešil pretok znanja med razvijalci. Od naslednje generacije z umetno inteligenco podprtih orodij se pričakuje, da bodo razvijalcu v tej poplavi informacij, ki so jih spletne skupnosti akumulirale tekom več desetletij, pomagale v skladu s kontekstom problema narediti učinkovito agregacijo informacij, pridobljenih iz najrazličnejših virov v konkretno rešitev programskega problema.

Hiter razvoj algoritmov umetne inteligence in pristopov učinkovitega rudarjenja repozitorijev programskih rešitev odpirata čedalje več možnosti vpeljave tovrstnih tehnologij v področje programskega inženirstva. Želja po hitrejšem in cenejšem razvoju programskih rešitev, višja kakovost programskih produktov in pomanjkanje razvijalcev s potrebnim naborom veščin so samo nekateri izmed izzivov, za katere si obetamo, da jih bomo lahko z vpeljavo umetne inteligence uspešneje obvladovali. Prav tako kot v drugih inženirskih vejah, so poleg ustrezno usposobljenih razvijalcev tudi v programskem inženirstvu za uspešnost projektov ključna učinkovita orodja. Ker je razvoj informacijskih rešitev področje, ki od razvijalcev terja predvsem vložek intelektualnega dela, si veliko obetamo prav od vpeljave umetne inteligence v razvojna orodja. Orodja, podprta z metodami umetne inteligence so namreč eden izmed načinov, ki učinkovito izboljšati učinkovitost razvijalce. Na tehnologijah in pristopih umetne inteligence je mogoče zasnovati novo generacijo razvojnih orodij, ki v proces razvoja informacijskih rešitev vpeljejo višjo stopnjo avtonomnosti in napredno avtomatizacijo opravil.

Vpeljava umetne inteligence v orodja za razvoj programskih produktov na tej stopnji razvoja ne bo razrešila problema pomanjkanja kakovostnih in dobro izurjenih IT strokovnjakov v splošnem. Ti ostajajo osrednji in nepogrešljiv člen v procesu razvoja programske opreme, na katerih ostaja ključne odločitve v procesu. Nepoznavanja programskega jezika ali principov razvoja programske opreme s tovrstnimi

orodji ni moč nadomestiti. Vloge sodobnih z umetno inteligenco podprtih razvojnih orodij je, da razvijalce pri njihovem delu učinkovito podprejo in jih čim bolj razbremenijo. Razvijalci lahko tako z njihovo pomočjo v krajšem času ustvarijo več. Da je ta podpora tovrstnih orodij učinkovita, postaja vedno bolj nujno, da zmorejo ponuditi rešitve ne le preprostih in ponavljajočih se opravil, temveč ponudijo rešitve za opravila, ki od razvijalca terjajo miselni napor.

LITERATURA

[1] Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2019). DeepCoder: Learning to write programs. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.

[2] Bitbucket. (2019a). Pridobljeno 17. avgusta 2019 od https://bitbucket.org/

[3] Bitbucket. (2019b). Celebrating 10 million Bitbucket Cloud registered users. Pridobljeno 2. junija 2020 od https://bitbucket.org/blog/celebrating-10-million-bitbucket-cloud-registered-users

[4] Bughin, J., Hazan, E., Labaye, E., Manyika, J., Dahlström, P., Ramaswamy, S., & Cochin de Billy, C. (2016). Digital Europe: Pushing the Frontier, Capturing the Benefits. Pridobljeno 17. marca 2019 od www.mckinsey.com/mgi.

[5] Cagle, K. (2019). Why Most Digital Transformations Will Fail. Pridobljeno 8. avgusta 2019 od Forbes website: https://www.forbes.com/sites/cognitiveworld/2019/01/07/why-most-digital-transformations-will-fail/#4c2a2357520e

[6] Codota. (2019). Pridobljeno 21. januarja 2019 od https://www.codota.com/

[7] Copeland, B. J. (2019). Artificial intelligence. Pridobljeno 8. avgusta 2019 od https://www.britannica.com/technology/artificial-intelligence

[8] Cuesta, C., Ruesta, M., Tuesta, D., & Urbiola, P. (2015). The digital transformation of the banking industry. Pridobljeno 17. marca 2019 od http://www.bbvaresearch.com

[9] Deming, D. J., & Noray, K. (2018). STEM Careers and Technological Change. Pridobljeno 17. avgusta 2019 od https://scholar.harvard.edu/files/kadeem/files/demingnoray_stem_sept2018_final.pdf

[10] Giffi, C., Wellener, P., Dollar, B., Manolian, H. A., Monck, L., & Moutray, C. (2018). 2018 Deloitte and The Manufacturing Institute skills gap and future of work study.

[11] GitHub. (2019). Pridobljeno 17. avgusta 2019 od https://github.com/

[12] GitHub. (2020a). Thank you for 100 million repositories. Pridobljeno 1. junija 2020 od https://github.blog/2018-11-08-100m-repos/

[13] GitHub, I. (2020b). GitHub is how people build software. Pridobljeno 2. junija 2020 od https://github.com/about

[14] GitLab. (2019). Pridobljeno 17. avgusta 2019 od https://about.gitlab.com/

[15] Gradišnik, M., Karakatič, S., Mauša, G., Beranič, T., & Heričko, M. (2019). Možnost vpeljave umetne inteligence v proces razvoja programske opreme. V Š. Urh Popovič (Ur.), Slovenija 4.0 : zbornik. 26. konferenca Dnevi slovenske informatike, 16. in 17. april 2019, Portorož. Ljubljana: Slovensko društvo Informatika.

[16] Gradišnik, M., Tina, B., & Karakatič, S. (2019). Implementacija programskih rešitev s pomočjo inteligentnih asistentov. V M. HERIČKO & K. KOUS (Ur.), Sodobne informacijske tehnologije in storitve : OTS 2019 : zbornik štiriindvajsete konference, Maribor, 18. in 19. junij 2019 (str. 138–148). Univerzitetna založba Univerze.

[17] Griffin, D., Mok, L., Struckman, C., & Berry, D. (2018). Tackle the Talent Problem : Invest in Growing Your Own Employees. Gartner.

[18] Güemes-Peña, D., López-Nozal, C., Marticorena-Sánchez, R., & Maudes-Raedo, J. (2018). Emerging topics in mining software repositories: Machine learning in software repositories and datasets. Progress in Artificial Intelligence, 7(3), 237–247. https://doi.org/10.1007/s13748-018-0147-7

[19] Handy, A. (2018). Codota Offers Pair Programming with Artificial Intelligence. Pridobljeno 14. avgusta 2019 od https://thenewstack.io/codota-offers-ai-pair-programming/

[20] Husain, H., & Wo, H.-H. (2018). Towards Natural Language Semantic Code Search. Pridobljeno 18. avgusta 2019 od https://githubengineering.com/towards-natural-language-semantic-code-search/

[21] Kaur, A., Kaur, K., Chopra, D., & Kaur, H. (2020). Systematic Literature Review on Mining Software Repositories. 7(1), 195–231.

[22] Kite. (2019). Kite. Pridobljeno 17. avgusta 2019 od https://www.kite.com/

[23] Kumar, A. (2018). Automated front-end development using deep learning. Pridobljeno 21. maja 2019., od https://blog.insightdatascience.com/automated-front-end-development-using-deep-learning-3169dd086e82

[24] Lo Giudice, D. (2016). How AI Will Change Software Development And Applications.

[25] Lorica, B., & Loukides, M. (2018). What machine learning means for software development. Pridobljeno 8. avgusta 2019 od https://www.oreilly.com/ideas/what-machine-learning-means-for-software-development

[26] Microsoft. (2019). Visual Studio IntelliCode. Pridobljeno 18. avgusta 2019 od https://visualstudio.microsoft.com/services/intellicode/

[27] Moran, K., Bernal-C'Ardenas, C., Curcio, M., Bonett, R., & Poshyvanyk, D. (2018). Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. V IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

[28] Nguyen, P. T., Di Rocco, J., & Di Ruscio, D. (2018). Mining software repositories to support OSS developers: A recommender systems approach. CEUR Workshop Proceedings, 2140.

[29] Ross, A., & Srinivas, V. (2018). Accelerating digital transformation in banking Findings from the global consumer survey on digital banking. Pridobljeno 8. avgusta 2019 od https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-accelerating-digital-transformation-in-banking.pdf

[30] Sketch2Code. (2019). Pridobljeno 17. avgusta 2019 od https://sketch2code.azurewebsites.net/

[31] SourceForge. (2019). Pridobljeno 17. avgusta 2019 od https://sourceforge.net/

[32] StackOverflow. (2019). Pridobljeno 17. avgusta 2019 od https://stackoverflow.com/

[33] Tiobe. (2020). Tiobe Index. Pridobljeno 15. marca 2020 od https://www.tiobe.com/tiobe-index/

[34] Yao, M. (2018). 6 Ways AI Transforms How We Develop Software. Pridobljeno 20. marca 2019 od https://www.forbes.com/sites/mariyayao/2018/04/18/6-ways-ai-transforms-how-we-develop-software/#21347e7d26cf

[35] Zapponi, C. (2020). GitHut 2.0 - A small place to discover languages in GitHub. Pridobljeno 8. junija 2020 od https://madnight.github.io/githut/#/pull_requests/2019/1

▪

Mitja Gradišnik je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Raziskovalno se ukvarja s sodobnimi pristopi pri razvoju programskih rešitev, kakovostjo in obvladovanjem staranja programskih produktov ter praktično uporabo metod podatkovnega rudarjenja v programskem inženirstvu. Raziskovalne in aplikativno sodeluje na več projektih, ki se odvijajo v okviru Inštituta za informatiko.

▪

Tina Beranič je asistentka 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).

▪

Sašo Karakatič je docent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Raziskovalno se ukvarja s področjem umetne inteligence in strojnega učenja ter aplikacijo optimizacijskih pristopov po vzoru narave na področjih transporta in rudarjenja podatkov.