Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Samuel Braniša

Vývojové prostredie pre jazyk Robot Framework v editore Vim

Priebežná správa o riešení BP1

Študijný program: Informatika

Študijný odbor: 9.2.1 Informatika

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava

Vedúci práce: Ing. Martin Vojtko PhD.

Január 2021

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

ZADANIE BAKALÁRSKEHO PROJEKTU

Meno študenta:	**Braniša Samuel**
Študijný odbor:	Informatika
Študijný program:	Informatika
Názov projektu:	**Vývojové prostredie pre jazyk Robot Framework v editore Vim**

Zadanie:
Programovaciemu jazyku Robot Framework, ako nástroju primárne určenému na automatizáciu, dlhodobo narastá popularita predovšetkým v oblasti testovania softvéru. K Robot Frameworku existuje významná a rozsiahla dokumentácia a používateľská komunita. Jazyk má jednoduchú syntax a umožňuje generovanie prehľadných výstupov (logov). Kvalitné vývojové prostredie je v súčasnosti nevyhnutným komponentom akéhokoľvek vývoja, najmä z hľadiska jednoduchosti a efektivity práce softvérových inžinierov. Vim je špecifickým vývojovým prostredím, ktorý nedisponuje adekvátnou podporou Robot Frameworku.
Analyzujte existujúce vývojové prostredia pre Robot Framework mimo editoru Vim. Zistite ako fungujú vývojové prostredia v editore Vim pre iné jazyky, spôsob akým sa rozširujú a vytvárajú pluginy pre Vim. Skúmajte jazyk, syntax, sémantiku a typický štýl písania kódu pre Robot Framework. Na základe analýzy navrhnite a implementujte riešenie, ktoré rozšíri existujúce pluginy pre editor Vim o zvýrazňovanie syntaxe textu, zvýrazňovanie chýb, podporu pohybu pomocou štítkov (tag-ov). Súčasťou práce bude vytvorenie pluginu, ktorý doplní Vim o chýbajúcu funkcionalitu a spraví z neho plnohodnotné vývojové prostredie pre vývojára pracujúceho s Robot Frameworkom.

Práca musí obsahovať:

Anotáciu v slovenskom a anglickom jazyku
Analýzu problému
Opis riešenia
Zhodnotenie
Technickú dokumentáciu
Zoznam použitej literatúry
Elektronické médium obsahujúce vytvorený produkt spolu s dokumentáciou

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava
Vedúci projektu: Ing. Martin Vojtko, PhD.

Termín odovzdania práce v zimnom semestri : 7.12.2020
Termín odovzdania práce v letnom semestri : 11.5.2021

Bratislava 21.9.2020

doc. Ing. Valentino Vranić, PhD.
poverený riadením ÚISI

Čestne vyhlasujem, že som túto prácu vypracoval(a) samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, Január 2021

Samuel Braniša

Poďakovanie

Chcel by som sa poďakovať vedúcemu mojej záverečnej práce, Ing. Martinovi Vojtkovi PhD. za všetky užitočné rady, návrhy a pripomienky, ktoré mi veľmi pomohli pri písaní mojej bakalárskej práce.

Annotation

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies
Degree Course: Informatika

Author: Samuel Braniša
Bachelor's Thesis: Vývojové prostredie pre jazyk Robot Framework v editore Vim
Supervisor: Ing. Martin Vojtko PhD.
Január 2021

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Autor: Samuel Braniša
Bakalárka práca: Vývojové prostredie pre jazyk Robot Framework v editore Vim
Vedúci bakalárskej práce: Ing. Martin Vojtko PhD.
Január 2021

Cieľom tejto práce je tvorba rozšírenia pre editor Vim, ktoré pretvorí editor Vim na plnohodnotné vývojárske prostredie pre jazyk Robot Framework. Vďaka tomuto rozšíreniu bude Vim zvýrazňovať syntax jazyku Robot Framework. Spúšťat nad zdrojovým kódom statickú analýzu, na zisťovanie chýb, nedostatkov a porušení konvencii pri písaní zdrojového kódu. Umožňovať presun v zdrojovom kóde pomocou štítkou. Umožňovať spúštanie samostaných testov alebo sád testov priamo z editoru Vim. Podporovať automatické dopĺňanie slov. Súčasťou riešenia je tvorba Docker kontajneru, v ktorom je možné spúšťať editor Vim s plnou podporou jazyka Robot Framework vďaka vopred spomínanému rozšíreniu. Samotné rozšírenie je open source projekt, ktorý je možné stiahnuť a nainštalovať do editoru Vim a Neovim zo stránky github.com manuálne alebo za pomoci manažérov balíčkov.

Obsah

Úvod

Robot Framework sa stal jedným z najpoužívanejších jazykov, ktoré sa momentálne používajú pri integračnom testovaní a automatizácii. Ako jeho občasný používateľ som bol prekvapený, že nie všetky aktuálne populárne editory, zjednodušujú vývoj v tomto jazyku. Najviac ma prekvapilo, že rozšírenie na editor Vim, ktorý najčastejšie používam, nieje aktuálne a podporuje iba zvýrazňovanie syntaxe starej verzie Robot Framework. Keďže ma editor Vim veľmi zaujíma, chcem lepšie pochopiť ako funguje na pozadí a mám záujem rozšíriť svoje vedomosti o tvorbu nových rozšírení pre tento editor, tak som sa rozhodol, že by toto mohla byť dobrá téma na bakalársku prácu. Mojim cieľom je tvorba balíčka, ktorý by pretvoril editor Vim na plnohodnotné vývojárske prostredie pre jazyk Robot Framework. V časti analýza rozoberám aké iné vývojárske prostredia pre jazyk Robot Framework aktuálne existujú, akým spôsobom uľahčujú vývojárom prácu a v čom sa odlišujú. V neskoršej časti analýzy opisujem rôzne spôsoby, ktorými je možné editor Vim rozšíriť o novú funkcionalitu a porovnávam jednotlivé prístupy. Na záver popisujem akým spôsobom je možné vytvoriť nové rozšírenie a balíček pre editor Vim.

Úvod k technológiám

Táto kapitola predstavuje hlavné technológie, ktorými sa táto Bakalárska práca zaoberá. Kapitola pozostáva z úvodu k Robot Frameworku, ktorý jednoduchým spôsobom definuje potrebu, využitie a históriu Robot Frameworku. Neskôr sa kapitola zaoberá editormi Vim a Neovim, a ich rozdielmi.

2.1 Robot Framework

Robot Framework je open source framework automatizácie všeobecných testov, ktorý sa používa na akceptačné testy a približuje ich bežnému vývoju, vďaka čomu sa spopularizoval koncept programovania riadeným akceptačnými testami. Prvá verzia Robot Frameworku vznikla v roku 2005 v dielňach Nokia Siemens Networks ako diplomová práca študenta Pekka Klärck. Pred vytvorením Robot Frameworku, sa na testovanie používali rôzne nástroje, ktoré boli zväčša proprietárne a trpeli nedostatkom flexibility. Keďže rozličné projekty potrebujú akceptačné testy napísané odlišným spôsobom vzniká potreba flexibility pri písaní testov. Robot Framework je dostatočne flexibilný a rozšíriteľný nato, aby zvládol tieto scenáre. [2]

2.2 Vim

Vim je open source textový editor, ktorého prvé verzia bola vytvorená v roku 1988 holandským vývojárom menom Bram Moolenaar. Vim je od svojho vytvorenia až po dnes vyvíjaný a udržiavaný prevažne svojím tvorcom. Od svojho vytvorenia bol Vim silno inšpirovaný svojím predchodcom Vi, ktorý napísal Bill Joy pre systémy

typu unix. Pôvodným účelom Vimu bolo vytvorenie napodobneniny vopred spomínaného editoru Vi pre iné platformy ako Unix. Postupom času sa z Vimu stal jeden z najpoužívanejších a najobľúbenejších editorov medzi softvérovými vývojármi. Jeho hlavnou výhodou dodnes ostáva, že je podporovaný na väčšine zariadení. Keďže zdrojový kód Vimu je otvorený a tým pádom má k nemu prístup široká verejnosť, tak je možné ho používať zadarmo. Bram Moolenaar nemá z neho priamo žiaden zisk a prosí užívateľov o registráciu cez oficiálnu stránku Vimu. Licencia na využívanie Vimu stojí 10 euro a všetky peniaze sú použité na pomoc deťom v Ugande. Toto robí z Vimu tzv. charitatívny softvér. [18] [19] [24] [15]

2.3 Neovim

Neovim je open source textový editor, ktorý vznikol v roku 2014. Hlavným dôvodom jeho vzniku boli rozpory v komunite Vim-u. Dlhšiu dobu chceli aktívny používatelia pomôcť pri vývoji Vimu, sprehľadniť zdrojový kód a spraviť samotný program rýchlejší, spoľahlivejší a zároveň aj zlepšiť responzivitu. Bram Moolenaar má tendenciu väčšinu návrhov zamietnuť a neskôr ich implementovať vlastným spôsobom. Preto sa skupina programátorov rozhodla zobrať zdrojový kód Vimu a upraviť ho. Tento nový projekt pomenovali Neovim, čiže Vim budúcnosti. Nové funkcie, ktoré vývojári vymyslia, pridajú a otestujú v Neovime sa časom presúvajú v obdobnej forme aj do Vimu. Konkurencia, ktorú Neovim vytvára pre Vim motivuje aj po toľkých rokoch pridávanie novej funkcionality do Vimu. [18] [10]

2.4 Rozdiely medzi Vim a Neovim

Väčšina novej funkcionality, s ktorou prišiel Neovim je už aj v najaktuálnejšej verzii Vimu. Stále sú niektoré veci, v ktorých Neovim Vim prevyšuje.

Na začiatok by som uviedol, že Neovim spĺňa špecifikáciu XDG Base Directory Specification, ktorá bola navrhnutá skupinou Free Deskop Group. Táto špecifikácia zatrieďuje súbory rôznych typov a funkcionality do rôznych priečinkov. Jedna časť tejto špecifikácie definuje miesto pre konfiguračné súbory

$XDG_DATA_HOME/.config. Vďaka tomuto je domovský adresár používateľov prehľadnejší, čistejší a roztriedený. [29]

Neovim podporuje asynchrónne spúšťanie externých rozšírení. Vďaka tejto funkcionalite je Neovim rýchlejší a responzívnejší, zároveň niesu jeho knižnice spomaľované procesmi ako napríklad čakanie na vstup.

Pre pridanie podpory knižníc pre Vim napísané v jazyku Python je potrebné znova skompilovať Vim a špecifikovať podporu pre jazyk Python. Pre Neovim stačí dodatočne nainštalovať podporu a nieje potrebná nová kompilácia.

Neovim má v sebe zabudovanú podporu pre jazyk Lua, ktorá umožňuje jednoduchšie dodatočné upravovanie editoru. [18]

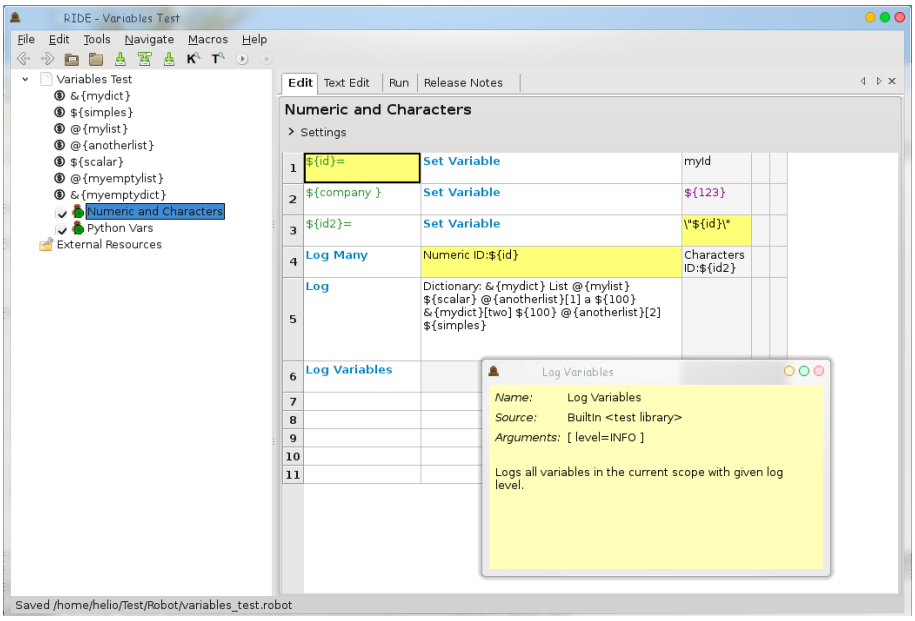
Porovnanie vývojových prostredí pre Robot Framework

Táto kapitola prechádza najdôležitejšími a najpoužívanejšími vývojovými prostrediami, ktoré vývojári používajú na vývoj v Robot Frameworku. Každe vývojové prostredie je najskôr predstavené a neskôr je opísané rozšírenie, ktoré dopĺňa novú funkcionalitu pre vývoj v Robot Frameworku. Na konci kapitoly je tabuľka, ktorá zvýrazňuje rozdiely medzi jednotlivými vývojovými prostrediami.

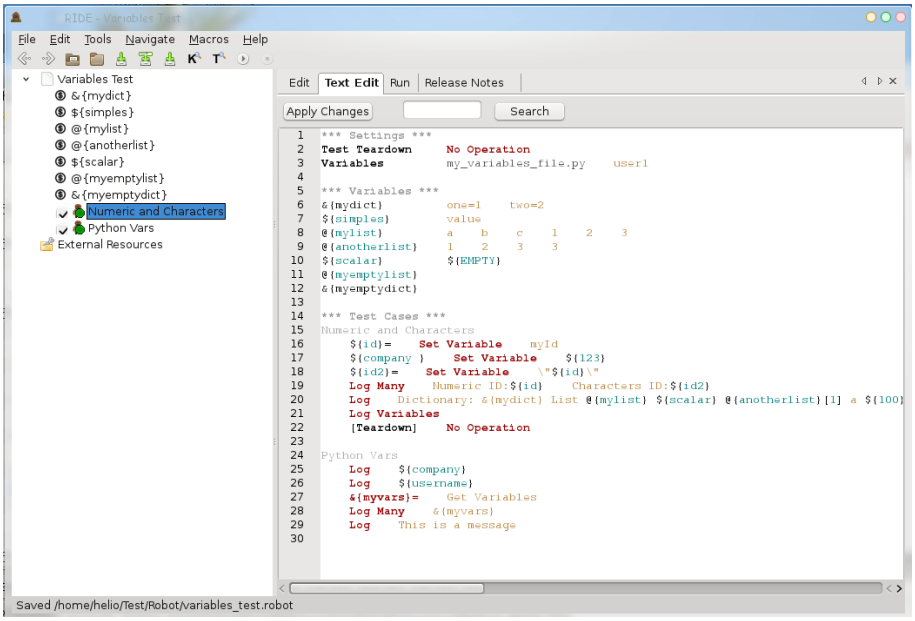
3.1 RIDE

RIDE je vývojové prostredie napísané v jazyku Python, ktorý sa na verejnosť prvý krát dostal v roku 2009. Na rozdiel od iných editorov sa nedá použiť pre vývoj v iných jazykoch ako Robot Framework. Jeho štruktúra pozostáva zo zoznamu súborov, zoznamu testovacích sád a samotného vývojového prostredia. Samotné vývojové prostredie sa skladá z troch častí. [7]

- Editora Testov, ktorý je určený hlavne na písanie dokumentácie k testom. Okrem písania dokumentácie je možná špecifikácia kľúčových slov, ktoré sa majú spustiť pred a po zbehnutí jednotlivých testov testovacej sady.

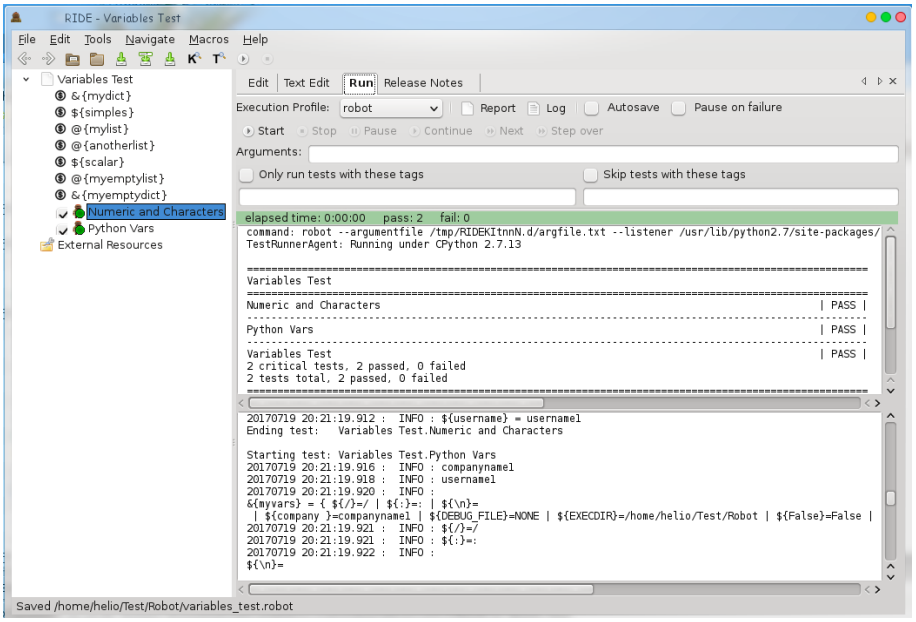


- Editor Textu, ktorý je určený na úpravu zdrojového kódu testov. Syntax zdrojového kódu je zvýrazňovaná a editor textu podporuje aj jednoduchý spôsob automatického doplňovania. Pre zobrazenie možností doplnenia je potrebné stlačenie klávesovej skratky, ktorá spustí proces navrhovania slov podľa aktuálneho začiatku slova.



- V záložke testov je možné spúšťanie sád testov a aj odstraňovanie chýb pomo-

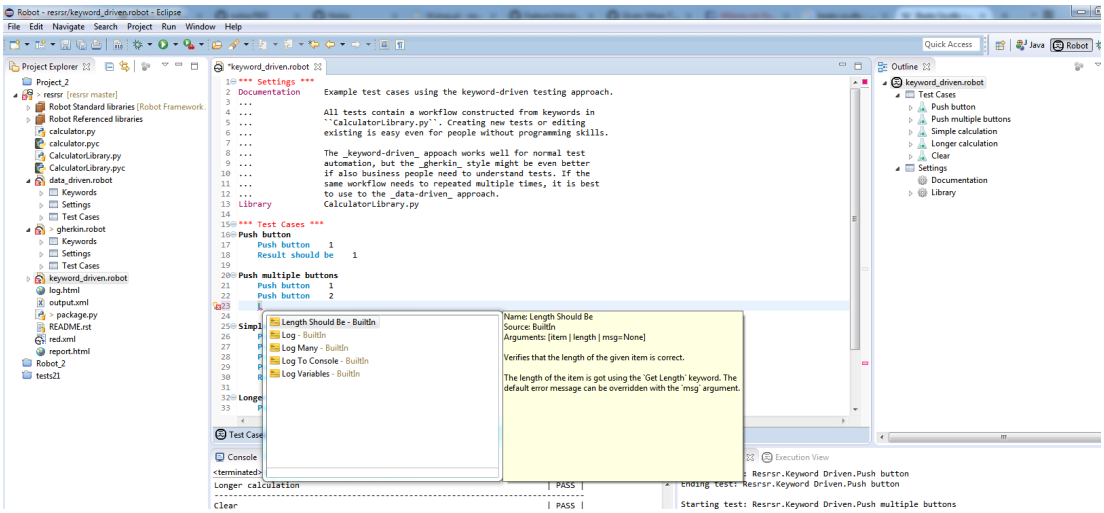
cou funkcie debug. Po spustení testov je možné podrobné prezeranie zápisov z ukončených testov.



3.2 Eclipse

Eclipse je jeden z najpoužívanejších vývojových prostredí. Jeho prvá verzia vyšla v roku 2001. Hlavným zámerom Eclipsu bolo vytvorenie rozsiahleho vývojového prostredia pre jazyk Java, o ktorý v danom čase veľmi rýchlo rástol záujem. Postupom času začal Eclipse podporovať vývoj v čoraz viac programovacích jazykoch. [4]

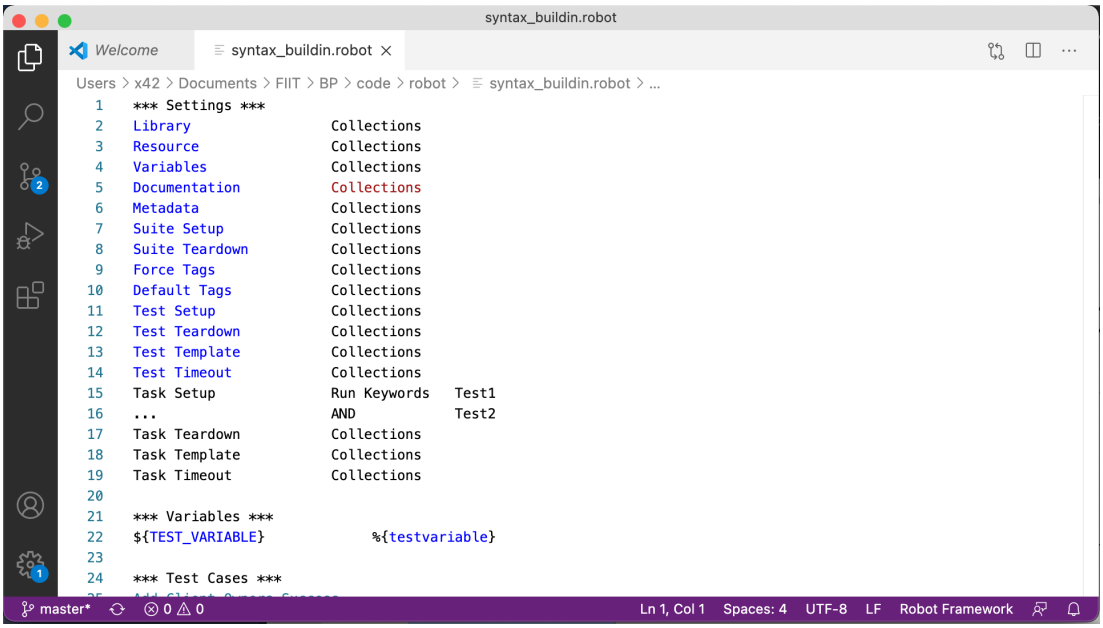
- RobotFramework-EclipseIDE je rozšírenie, ktoré premení Eclipse na vývojové prostredie pre jazyk Robot Framework. Podporuje zvýraznovanie syntaxe, prácu s tagmi(štítkami) automatické dopĺňanie. Toto rozšírenie už dlhšiu dobu nieje používané a v čase písanie tejto práce najnovšia verzie tohto rozšírenia nemala podporu pre najnovšiu verziu Eclipse. [3]
- RED je vývojové prostredie, ktoré je založené na Eclipse. Vytvorila ho Nokia a podporuje zvýraznovanie syntaxe zdrojového kódu a automatické dopĺňanie. Podobne ako vývojové prostredie RIDE, je rozdelený na viacero záložiek,



v ktorých je možné zobrazovanie sád testov, globálnych premenných, kľúčových slov a nastavení. [20]

3.3 Visual Studio Code

VS Code je open source textový editor vytvorený firmou Microsoft. Ide o momentálne najpoužívanejší editor, ktorý umožňuje vývoj vo väčšine programovacích jazykov vďaka LSP (language server protocol). Robot Framework Intellisense je rozšírenie, ktoré zjednodušuje vývoj zdrojového kódu v editore VS Code. Toto rozšírenie podporuje zvýrazňovanie syntaxe zdrojového kódu, prejdenie na definíciu, prácu so štítkami (tagmi), automatické dopĺňanie pre štandardnú knižnicu Robot Frameworku a zároveň aj mnohé knižnice vytvorené tretími stranami. [13][23]



3.4 Emacs

GNU Emacs je open source textový editor, ktorého autorom je zakladateľ projektu GNU Richard Stallman. Vďaka podpore jazyku elisp (Emacs Lisp) patrí Emacs medzi jeden z najviac rozšíriteľných a upravovateľných editorov. [5]

Robot-mode je rozšírenie, ktoré pretvára Emacs z textového editoru na vývojové prostredie pre jazyk Robot Framework. Podporuje zvýrazňovanie syntaxe zdrojového kódu, komentovanie riadkov, prehľadávanie kľúčových slov, automatické doplňovanie kľúčových slov, inteligentné odsadzovanie textu a prácu so štítkami (tagmi). Toto rozšírenie bolo naposledy upravené v roku 2010 a vzhľadom nato že k dnešnému dňu má iba 28 hviezdičiek na git-hube, nepatrí medzi najpoužívanejšie vývojové prostredia pre Robot Framework. [9]

3.5 Atom

Atom je open-source editor, ktorý je postavený na technológii Electron, to znamená že je to upravený internetový prehliadač Chromium, ktorý podporuje roz-

šírenia vďaka Node.js. Atom podporuje veľké množstvo rozšírení. Jedno z nich je language-robot-framework. Toto rozšírenie prináša podporu zvýrazňovania syntaxe a automatické dopĺňanie pri vývoji súborov s koncovkou robot. [6][8]

	RIDE	Eclipse	VS Code	Emacs	Atom
zvýrazňovanie syntaxe	+	+	+	+	+
práca so štítkami	-	+	+	+	-
automatické dopĺňanie	+	+	+	+	+
spúšťanie sád testov	+	+	-	-	-
prezeranie logov	+	+	-	-	-
debug mód	+	-	-	-	-
prejdenie na definíciu	-	+	+	+	-
komentovanie bloku textu	-	+	+	+	+
automatické odsadzovanie textu	-	-	-	+	+
písanie dokumentácie	+	-	-	-	+

Bežná podpora vývoja

V tejto kapitole budem popisovať rôzne typy bežnej podpory vývoja, ako je možné Vim rozšíriť o tieto jednotlivé typy funkcionalít. Na záver uvediem, pre aké rozšírenia som sa rozhodol ja.

4.1 Syntaktická analýza

Asi najzákladnejší spôsob podpory vývoje je narábanie so syntaxou. Existujú dve základné členenia, zvýrazňovanie syntaxe a syntaktická kontrola.

4.1.1 Zvýrazňovanie syntaxe

Zvýrazňovanie syntaxe zdrojového kódu je funkcia, ktorú si väčšina editorov implementujú vlastným spôsobom a Vim v tomto nieje výnimkou. Teoretický postup by mal vyzerať obdobne.

- Otvorí sa nový súbor.
- Editor zistí pomocou koncovky súboru typ novootvoreného súboru.
- Editor skúsi nájsť špecifikáciu syntaxe novootvoreného súboru.
- Ak editor našiel súbor so špecifikáciou zvýrazní syntax.
- Ak editor nenašiel súbor so špecifikáciou nezvýrazní syntax.

Súbor so špecifikáciou syntaxe by mal obsahovať pravidlá triedenia. Jednotlivé pravidlá by mali text zatriediť do rôznych skupín, ktoré sú následne zvýraznené

rôznymi farbami. Napr. konštanty budú zatriedené do odlišnej skupiny ako správy pre pre-procesor.

Implementácia vyhľadávania a zvýrazňovania syntaxe je pre Vim pomerne jednoduchá. Vim podporuje tri základné typy skupín pomocou ktorých je možné špecifikovať syntax.[16]

- Keywords - jednotlivé slová alebo slovné spojenia
- Matches - regex
- Regions - bloky zdrojového kódu, napr. medzi zátvorkami

Funkcionalitu definície zvýrazňovania je možné bližšie definovať pomocou rôznych prepínačov.

4.1.2 Syntaktická kontrola

Syntaktická kontrola je typ statickej analýzy zdrojového kódu. Je o niečo komplikovanejšia na implementáciu a preto si každý editor sám nevytvára vlastný program na statickú analýzu pre každý jazyk. Vytvorí sa väčšinou jeden program na statickú analýzu s ktorým programy, ktoré chcú využiť jeho funkcionalitu komunikujú prostredníctvom terminálového rozhrania. Pri spustení programu sa definuje súbor alebo skupina súborov nad ktorými má program vykonať kontrolu a na výstup program vráti výsledok syntaktickej analýzy. Výstup zväčša obsahuje nasledujúce údaje, ktoré reprezentujú každú nájdenú chybu alebo varovanie.

- názov súboru
- číslo riadku a znaku
- či ide o varovanie alebo chybu
- jednoslovné opísanie problému
- detailnejšie opísanie problému

4.1.3 Robotframework-lint

Robotframework-lint je jednoduchý nástroj na statickú analýzu kódu napísaného v jazyku Robot Framework. Robotframework-lint je napísaný v jazyku Python a je možné ho nainštalovať pomocou správcu balíčkou pip. Podporuje veľa prepínačov, ktoré uľahčújú prácu s ním. Vďaka týmto prepínačom je napríklad možné upraviť formát výstupného textu, ktorý tento nástroj vypisuje. Medzi ďalšiu funkcionalitu, ktorú tento program podporuje je rozširovanie množiny pravidiel, ktoré nástroj pozná pomocou jazyku Python alebo spôsob v akom poradí má prehľadávať súbory. [21]

4.1.4 Language Server Protocol

LSP (Language Server Protocol) je používaný medzi nástrojom (client) a znalcom jazyka (server) na integráciu funkcionality, akou je napríklad automatické dopĺňanie, prechod na definíciu, nájdenie všetkých referencií a podobne. LSP bolo vytvorené firmou Microsoft, neskôr sa pridali a pomohli pri vývoji LSP aj iné firmy, ako Codenvy, Red Hat a Sourcegraph. Počet editorov a komunít, ktoré podporujú tento protokol postupne rýchlo narastá.

Dôvodom vzniku LSP je absencia obdobného štandardu. Implementácia syntaktickej kontroly je veľmi zložitá činnosť. Ak by každý editor implementoval vlastné riešenie, akým sa syntax aktuálne používaného jazyka kontroluje, vznikol by problém rozsahu M * N, kde M predstavuje počet editorov a N počet jazykov. Pri použití štandardu ako je LSP sa tento problém zmenší na M + N z dôvodu, že stačí pridať podporu jazyka len raz a to len do LSP a pre každý editor napísať jedno rozšírenie, ktoré prepojí LSP a daný editor. Samozrejme sa týmto zmenšujú rozdiely medzi editormi. [12]

4.1.5 Asynchronous Lint Engine

ALE (Asynchronous Lint Engine) je rozšírenie na syntaktickú analýzu zdrojového kódu. Jeho výhoda je v tom, že okrem klasických externých nástrojov na kontrolu

syntaxe dokáže podporovať aj LSP. ALE podporuje veľké množstvo nástrojov na syntaktickú analýzu zdrojového kódu. Pri jeho spustení je možné špecifikovať, ktoré nástroje sú povolené alebo zakázané. Nato aby bol nástroj použitý na kontrolu zdrojového kódu musí byť v skupine povolených nástrojov a zároveň aj nainštalovaný na danom počítači. Pri spustení sa pretriedi skupina povolených nástrojov a ostanú v nej len tie, ktoré sú zároveň aj dostupné. ALE potom buď po každej úprave upravovaného zdrojového kódu alebo po uložení (záleží od nastavenia) skontroluje zásobník so zdrojovým kódom. Jednotlivé riadky s chybami alebo upozorneniami sú následne označené. Ak používateľ presunie kurzor na zvýraznený riadok, zobrazí sa jednoslovný dôvod zvýraznenia riadku, ak používateľ potrebuje obšírnejšie vysvetlenie, môže zadať príkaz ALEDetail, ktorý vypíše podrobnejší opis problému. Tento príkaz je v editore Vim možné jednoduchým spôsobom namapovať na skratku podľa voľby používateľa. [1]

4.1.6 Syntastic

Takisto ako ALE tak aj Syntastic je rozšírenie na syntaktickú analýzu zdrojového kódu. Tieto rozšírenia sú veľmi podobné a hlavný rozdiel je v tom, že Syntactic podporuje väčšie množstvo externých nástrojov na syntaktickú analýzu pre viacero jazykov ale nepodporuje LSP. Spôsob akým je chybný text označovaný je tiež veľmi podobný. [25]

4.2 Ctags

Úlohou Ctags je lepšie pochopenie zdrojového kódu a zrýchlenie navigácie. Tento štandard podporujú rôzne editory, ako napríklad Atom, Emacs, Vim a mnoho iných. Prvá implementácia Ctags vznikla v roku 1992 a podporovala okrem jazyku C aj Fortran a Pascal. Ctags prejde súborom a pomocou regexov, definovaných pre daný jazyk, vytiahne zo zdrojového kódu tie najviac zaujímavé vecí ako napríklad mená funkcii, tried alebo premenných. Tento spôsob výberu najdôležitejších informácii zo zdrojového kódu sa volá indexácia. Pôvodná verzia Ctags vytvárala jednotlivé indexy v nasledovnom formáte.

{meno_tagu}{meno_súboru}{odkaz_na_pozíciu}

Aktuálne najpoužívanejšia verzia Ctags sa volá Universal Ctags a jeho formát sa veľmi nezmenil, spoluautorom tohto nového formátu Universal Ctags bol sám Bram Moolenar, tvorca editoru Vim v roku 1998. Tento nový štandard je spätne kompatibilný a líši sa od svojho predchodcu len v tom, že na koniec je pridaný slovník. [28]

{pôvodný_c_tag}[;"{kľúč}:{hodnota}..]

4.2.1 Tag List

Tag List je open source rozšírenie pre Vim. Jeho hlavným účelom je vytvorenie vertikálneho zásobníku, v ktorom sú postupne zoskupené do skupín názvy premenných, funkcii, tried alebo iných dôležitých prvkov zdrojového kódu. Pomocou názvov jednotlivých indexov je možný presun na pozíciu v zdrojovom kóde. Indexy sa počas písania zdrojového kódu aktualizujú. Jedna z možností, vďaka ktorej je možné definovanie regexov na tvorbu indexov je pomocou Universal Ctags. Samotné Universal Ctags podporuje vyše 40 programovacích jazykov a syntax Robot Frameworku je takisto podporovaná. Dodatočné pravidlá sa dajú doplniť priamo pre toto rozšírenie. [27]

4.2.2 Tag Bar

Tag Bar tak isto ako Tag List je open source rozšírenie, ktoré Vim rozširuje o prácu so štítkami. Tieto rozšírenia sú si veľmi podobné. [26]

4.2.3 Porovnanie Tag Bar a Tag List

Tag Bar je novšie ako Tag List. Tag Bar na rozdiel od Tag List podporuje aj iné štandardy na indexáciu ako ctags. Tag Bar krajšie zoskupuje tie isté skupiny tagov, umožňuje zatrieďovanie do skupín s ohľadom na dosah. Tag List má takisto svoje výhody, je rýchlejší na spustenie a podporuje zobrazovanie tagov pre viacero zásobníkov naraz.

4.3 Automatické dopĺňanie textu

Automatické dopĺňanie je veľmi dôležitá funkcia, ktorá v obdobnom spôsobe funguje v každom editore. Vim nieje v tomto výnimkou. Existuje viacero rôznych spôsobov akým je možné dosiahnuť doĺňanie textu.

4.3.1 Omni Completion

Vim disponuje dvomi typmi doplňovania textu. Prvé je doplňovanie definované používateľom a druhé je doplňovanie v závislosti od typu jazyka. Tento druhý spôsob doplňovania je známy ako Omni doplňovanie. Pre jednotlivý typ jazyka je vopred potrebné definovať aký typ Omni funkcie má editor používať na ponúkanie slov. Jednou z možností je použiť súbor, pomocou ktorého je definovaná syntax daného jazyka. Samotná funkcia Omni sa potom po stlačení vopred definovanej skratky zavolá dva-krát a obsahuje dva vstupné argumenty.

- boolean, ktorý definuje či sa volá funkcia prvý alebo druhý krát
- string, ktorý definuje slovo ktoré má byť rozšírené, môže byť prázdne

Prvé volanie funkcie Omni zistí pozíciu stĺpca, v riadku v ktorom sa nachádza kurzor odkiaľ začína aktuálne slovo, ktoré chce používateľ doplniť. Prvé volanie funkcie pri správnom ukončení vracia kladné číslo, ktorého hodnota je menšia ako pozícia stĺpca kurzoru v riadku. Druhé zavolanie funkcie Omni zadá ako parameter slovo, ktoré chce používateľ rozšíriť. Druhé volanie funkcie pri správnom ukončení vracia pole slov, ktoré funkcia považuje za adekvátny výber pre používateľa a zobrazí ho v stĺpci ktorým môže používateľ následne prechádzať. [22]

4.3.2 YouCompleteMe

YCM (YouCompleteMe) je rozšírenie na automatické dopĺňanie pre Vim. YCM je rýchly, vyhľadáva aktuálne návrhy počas toho ako používateľ píše, nové návrhy sa vytvárajú na princípe neostrej rovnosti. Samotné YCM má vstavané nástroje

na dopĺňanie textu pre najpoužívanejšie jazyky ako Java, C, C++, Python, Javascript a okrem toho disponuje klientom pre LSP štandard. Slovnú zásobu YCM je možné rozšíriť aj pomocou slovníka, ktorým disponuje Omni dopĺňanie priamo z editoru Vim. YCM automaticky volá funkciu na zisťovanie možných dokončení slova, čím zaniká potreba manuálneho zadávania skratky ako v prípade funkcie Omni. Samotné rozšírenie používateľsky príjemnejšie ako vstavaná funkcionalita editoru Vim. [30]

4.3.3 Deoplete

Deoplete je rozšírenie veľmi podobné YCM. Hlavný rozdiel je v jeho implementácii. YCM potrebuje server, YCMD(YouCompleteMe daemon), v ktorom prebieha všetok výpočet hladania možností doplnení slova. Deoplete je napísaný v jazyku Python a ak daná verzia editoru Vim podporuje jazyk Python tak zaniká potreba externého daemona, ktorý beží v pozadí. Vďaka asynchrónnej implementácii je deoplete dostatočne rýchli na bežné používanie. Spôsob nastavovania Deoplete a YCM je odlišný a zároveň aj spôsob inštalácie (YCM treba kompilovať). [11]

Vim rozšírenia

Vim script je skriptovací jazyk, vďaka ktorému funguje celý Vim. Pomocou Vim scriptu je možné rozšíriť Vim o novú funkcionalitu. Po vložení scriptu do priečinku s rozšíreniami sa stáva zo skriptu nové rozšírenie. Existujú dva rozdielne typy rôzne typy rozšírení. [17]

- Globálne rozšírenia
- Rozšírenia na základe typu súboru

Rozšírenie na základe typu súboru je veľmi podobné globálnemu rozšíreniu, hlavný rozdiel je v tom, že globálne rozšírenie mení nastavenia a je použiteľné na globálnej úrovni ale rozšírenia na základe typu súboru upravujú platnosť nastavení a novej funkcionality iba na úrovni zásobníkov a okien.

5.1 Základne konvencie písania rozšírení

- **Meno**
  Každé rozšírenie v jazyku Vim script by malo mať svoje pomenovanie, ktoré by malo jasne a stručne vystihovať jeho poslanie a zároveň by nemalo presahovať dĺžku 8 znakov.
- **Hlavička**
  Súbor samotného rozšírenia by mal začínať 4mi riadkami, ktoré sú komentáre. Prvý riadok by mal obsahovať krátky opis, o tom čo dané rozšírenie robí. Druhý riadok by mal obsahovať dátum poslednej zmeny v aktuálnej verzii rozšírenia. Tretí riadok by mal obsahovať údaje o tvorcovi rozšírenia

a e-mailovú adresu, kam môžu používatelia rozšírenia písať pripomienky k rozšíreniu. Posledný štvrtý riadok by mal obsahovať typ licencie, ktorou je kryté dané rozšírenie.

```
"Vim filetype plugin for running Robot Framework tests
"Last Change: 2020 Dec 15
"Maintainer: Samuel Braniša <samuel.branisa@vim.org>
"License: This file is placed in the public domain.
```

- **Predchádzanie vedľajším účinkom**
  Kvôli globálnym nastaveniam Vim-u je možné, že u väčšej skupiny ľudí nebude rozšírenie správne fungovať. Toto sa dá opraviť jednoduchým spôsobom. Na začiatku súboru je potrebné si zapamätať aktuálne nastavenia editoru Vim a na konci súboru ich nastaviť späť na pôvodné hodnoty. Toto je možné docieliť s využitím premenných, ktoré dosah je iba v rámci skriptu.

```
“ začiatok skriptu
let s:save_cpo = &cpo
set cpo&vim
“ samotný skript
let &cpo = s:save_cpo
unlet s:save_cpo
“ koniec skriptu
```

- **Možnosť ne-načítania rozšírenia**
  Používateľ by mal mať možnosť zabrániť načítaniu rozšírenia pomocou nastavenia vopred určenej premennej. Tento koncept zároveň umožňuje zakázanie viacnásobného načítavania toho istého rozšírenia.

- **Možnosť definovania vlastných skratiek**
  Používateľ by mal mať možnosť si sám mapovať novú funkcionalitu na vlastné skratky, bez toho aby rozšírenie prepísalo už vopred existujúce skratky.

- **Rozšírenie by nemalo meniť správanie funkcii, ktoré mu nepatria**
  Všetky nové funkcie, ktoré rozšírenie definuje by mali byť platné iba v roz-

sahu daného rozšírenia. Vim poskytuje možnosť volať lokálne funkcie jednotlivých rozšírení pomocou identifikátora <SID>, ktorý zistí identifikátor rozšírenia a následne je možné pomocou tohto identifikátora volať lokálne funkcie rozšírenia.

- **Formát súboru rozšírenia**
  Všetky rozšírenia by maly byť písané vo formáte súboru Unix. Keby bol formát súboru nastavený na DOS, tak by dané rozšírenie nefungovalo na systémoch typu Unix. Tvorca rozšírenia by tým pádom mal nastaviť premennú fileformat na unix pomocou :set fileformat=unix pred písaním nového rozšírenia.

[17]

5.2 Vim manažment balíčkov

Jednotlivé rozšírenia je možné zabaliť do balíčkov pre zjednodušenie distribúcie. Balíček môže obsahovať viac rozšírení, ktoré od seba závisia. Balíček je možné jednoducho aktualizovať keďže môže byť vo forme git alebo mercurial repozitáru. Existuje viacero rôznych manažérov, ktoré zaobstarávajú sťahovanie, inštalovanie, aktualizovanie a mazanie balíčkov. Veľkú väčšinu rozšírení je možné nájsť na stránke `www.vim.org` a následne stiahnuť pomocou verziovacieho systému git vo forme balíčkov. Medzi najpoužívanejšie manažéri balíčkov patria vim-plug, vundle, pathogen a vimball ktoré sa medzi sebou líšia hlavne rozsahom funkcionality, rýchlosťou inštalácie nových rozšírení a použiteľnosťou.

Bežný proces inštalácie nového balíčku zo strany používateľa pozostáva z...

- Používateľ nájde rozšírenie.
- Používateľ zistí meno vlastníka repozitára, a meno projektu na stránke `github.com`.
- Používateľ záda ${meno_vlastníka}/${meno_repozitára} manažérovi balíčkov.

- Manažér rozšírení nainštaluje dané rozšírenie.

Inštalácia nového rozšírenia zo strany manažéra balíčkov.

- Stiahnutie nového balíčku pomocou nástroju git do vopred vybratého priečinku kam daný manažér inštaluje všetky nové rozšírenia.
- Pridanie cesty k novému rozšíreniu do premennej runtimepath.
- Rozšírenie indexu s manuálmi o nové manuály pre nové rozšírenie.

Na inštaláciu nových rozšírení a balíčkov nie je potrebné použitie manažérov. Vim pri štarte prechádza súbormi v priečinku /.vim/plugin a priečinkami v priečinku /.vim/pack. Ak rozšírenie pozostáva z jedného súboru, stačí aby sa nachádzalo v priečinku plugin. Ak rozšírenie je vo forme balíčku, je treba aby bolo umiestnené na miesto /.vim/pack. Manažér balíčkov inštaluje nové rozšírenia do do prednastaveného priečinku, ktorý môže používateľ zmeniť. Po stiahnutí nového balíčku pridá manažér cestu k štartovaciemu priečinku rozšírenia do premennej runtimepath. Podľa obsahu premennej runtimepath vie Vim, ktoré súbory a priečinky má pri svojom spustení načítať. Ak používateľ sťahuje nové rozšírenia manuálne sám, mal by po stiahnutí a inštalácii nového balíčku spustiť príkaz :helptag, pomocou ktorého sa pridajú nové manuály zo všetkých ciest, ktoré premenná runtimepath špecifikuje. [14]

Literatúra

[1] *Asynchronous Lint Engine*. URL: https://github.com/dense-analysis/ale.

[2] Sumit Bisht. *Robot Framework Test Automation*. 2013. ISBN: 9781783283033. URL: www.packtpub.com.

[3] Nitor Creations. *RobotFramework-EclipseIDE*. URL: https://github.com/NitorCreations/RobotFramework-EclipseIDE/wiki.

[4] Eclipse Foundation. *Eclipse IDE*. URL: https://www.eclipse.org/eclipseide/.

[5] GNU foundation. *Emacs*. URL: https://www.gnu.org/savannah-checkouts/gnu/emacs/emacs.html.

[6] Github. *Atom*. URL: https://flight-manual.atom.io.

[7] Johny Hyperion a Helio Guilherme. *RIDE*. URL: https://github.com/robotframework/RIDE.

[8] Thanabodee Charoenpiriyakij. *Language Robot Framework*. URL: https://atom.io/packages/language-robot-framework.

[9] Sakari Jokinen. *Robot Mode*. URL: https://github.com/sakari/robot-mode.

[10] Justin M. Keyes. *Neovim*. URL: https://neovim.io.

[11] Shougo Matsu. *deoplete.nvim*. URL: https://github.com/Shougo/deoplete.nvim.

[12] Microsoft. *Language Server protocol*. URL: https://langserver.org.

[13] Microsoft. *Visual Studio Code*. URL: https://code.visualstudio.com.

[14] Bram Moolenaar. *Plugin Support*. URL: https://vimhelp.org/version6.txt.html#new-plugins.

[15] Bram Moolenaar. *Vim*. URL: https://www.vim.org.

[16] Bram Moolenaar. *Vim syntax highlighting*. URL: https://vim.help/44-your-own-syntax-highlighted#.

[17] Bram Moolenaar. *Write a Vim script*. URL: https://vimhelp.org/usr_41.txt.html.

[18] D. Neil. *Modern Vim: Craft Your Development Environment with Vim 8 and Neovim*. Pragmatic Bookshelf, 2018. ISBN: 9781680506013. URL: https://books.google.sk/books?id=o-FdDwAAQBAJ.

[19] D. Neil. *Practical Vim: Edit Text at the Speed of Thought*. Pragmatic Bookshelf, 2015. ISBN: 9781680504101. URL: https://books.google.sk/books?id=LA9QDwAAQBAJ.

[20] Nokia. *RED Robot Editor*. URL: https://github.com/nokia/RED.

[21] Bryan Oakley. *Welcome to Robot Framework Lint*. URL: https://github.com/boakley/robotframework-lint.

[22] *Omni Completion*. URL: https://vimhelp.org/insert.txt.html#complete-functions.

[23] Robocorp. *Robot Framework LSP*. URL: https://marketplace.visualstudio.com/items?itemName=robocorp.robotframework-lsp.

[24] K. Schulz. *Hacking Vim 7. 2: Ready-to-use Hacks with Solutions for Common Situations Encountered by Users of the Vim Editor*. Community experience distilled. Packt Pub., 2010. ISBN: 9781849510516. URL: https://books.google.sk/books?id=BMUfbfuY3zEC.

[25] *Syntastic*. URL: https://github.com/vim-syntastic/syntastic.

[26] *Tag Bar*. URL: https://github.com/preservim/tagbar.

[27] *Tag List*. URL: https://www.vim.org/scripts/script.php?script_id=273.

[28] *Universal Ctags*. URL: https://docs.ctags.io/en/latest/.

[29] Lennart Poettering Waldo Bastian Ryan Lortie. *XDG Base Directory Specification*. URL: https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html.

[30] *YouCompleteMe*. URL: https://github.com/ycm-core/YouCompleteMe.