VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

QT PROGRAMÁTORSKÝ EDITOR PRO LINUX

QT BASED PROGRAMMING EDITOR FOR LINUX SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RICHARD HAUERLAND**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN KRČMA**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Hauerland Richard**
Obor: Informační technologie
Téma: **Qt programátorský editor pro Linux**
Qt Based Programming Editor for Linux Systems
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte nabídku a vlastnosti programátorských editorů dostupných na platformě Linux.
2. Porovnejte editory placené s editory volně dostupnými a vyčleňte společné vlastnosti a principy a dále pak ty, které volně dostupné editory vůči těm placeným postrádají.
3. Na základě této analýzy navrhněte editor, který bude kromě běžných vlastností vybaven některými vlastnostmi kterými disponují placené editory.
4. Editor implementujte s použitím technologií C++ a Qt.
5. Proveďte uživatelské testování a vyvoďte zpětnou vazbu od testerů.
6. Navrhněte další možné směřování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Krčma Martin, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Cílem této práce je návrh a následná implementace svobodného textového editoru určeného pro operační systém Linux. Před návrhem textového editoru je provedeno srovnání nejvýznamějších existujících řešení a shrnutí tohoto srovnání je podkladem pro samotný návrh. V rámci návrhu editoru je zohledněna funkcionalita, kterou existující řešení buďto neposkytují, nebo ji poskytují jen některá řešení, obzvláště pak ta proprietární. Pro poskytnutí manipulace s projekty je zohledněna i funkcionalita z vývojových prostředí. Pro implementaci navrženého textového editoru je využito frameworku Qt v kombinaci s programovacím jazykem C++.

Abstract

Objective of this Bachelor's Thesis is about design and implementation of free text editor designed for Linux. Comparison of most popular existing solutions was performed before the design and it's output serves as basis for design process. Functionalities not provided and behaviors provided only by some, mostly proprietary, solutions, were taken in account in editor design. As well functionalities for projects manipulation used in development environments were considered. Editor is implemented in C++ language with use of Qt Framework.

Klíčová slova

textový editor, C++, Qt Framework, Linux, svobodná aplikace, projekt

Keywords

text editor, C++, Qt Framework, Linux, free application, project

Citace

HAUERLAND, Richard. *Qt programátorský editor pro Linux*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Krčma

Qt programátorský editor pro Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Krčmy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Richard Hauerland
11. května 2018

Poděkování

Chtěl bych poděkovat panu Ing. Martinu Krčmovi za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat.

Obsah

Kapitola 1

Úvod

Textový editor je aplikace sloužící k úpravě prostého textu bez formátování, která je využívána především vývojáři softwaru za účelem úpravy zdrojových kódů. Některý z textových editorů je tedy esenciální aplikací pro každého vývojáře softwaru. Textový editor určený pro úpravu zdrojových kódů je také možno nazvat programátorským editorem. Textových editorů existuje celá řada a některé z nich jsou dokonce základní součástí operačního systému. Prakticky každá distribuce operačního systému Linux obsahuje výchozí textový editor, který umožňuje editaci zdrojových kódů. Kromě výchozích textových editorů na Linuxových systémech existují další jak svobodná, tak proprietární řešení. Systém Linux je navíc nejpřívětivějším systémem pro vývojáře, nejen díky široké škále dostupných textových editorů, ale také díky velmi dobré dostupnosti překladačů a interpretů pro naprostou většinu programovacích, skriptovacích a značkovacích jazyků. [10]

Mezi hlavní funkce programátorských editorů patří samozřejmě editace zdrojových souborů, která zahrnuje pohodlné označování konkrétních částí textu a případné kopírování a vkládání takto označených bloků kódu. Naprostá většina dnešních textových editorů umožňuje editaci zdrojového kódu pomocí klávesnice, doplněné o práci s kurzorem myši pro snadné označování konkrétních bloků kódu. Umožněn je také návrat provedených změn a vyhledávání slov v textu. Vyhledávány jsou obvykle konkrétní textové vzory, nicméně některé editory umožňují použití regulárních výrazů pro pokročilé vyhledávání vzorů v kódu. Vyhledané vzory v textu bývá obvykle možné rovnou nahrazovat slovy jinými. Otevření vícero souborů zároveň v separátních záložkách je v dnešní době taktéž běžnou funkcionalitou. Stěžejní funkcí programátorských editorů musí být zvýrazňování syntaxe konkrétního jazyka v textu, kdy každý prvek jazyka bývá zvýrazňován odlišnou barvou z důvodu lepší čitelnosti zdrojového kódu.

Predevším z důvodu jednoduchého sestavení projektů vznikla vývojová prostředí, která vycházejí z programátorských editorů, ale jejich rozsah funkcionality bývá mnohem větší. Vývojová prostředí poskytují snadnou správu projektů včetně jejich sestavování, ladění a taktéž snažší editace zdrojového kódu za pomocí našeptávání. Některá vývojová prostředí umožňují například zobrazení stromové struktury popisující hierarchii tříd v daném projektu. Vývojová prostředí nicméně nejsou dokonalá a mezi jejich hlavní nevýhodu patří především omezení protředí pro konkrétní programovací jazyk. V některých případech může být podporováno i více jazyků, nicméně počet podporovaných jazyků bývá nesrovnatelně nižší než v případě klasických textových editorů.

Jak textové editory, tak vývojová prostředí mají své výhody i nevýhody. Nicméně by bylo vhodné porovnat vlastnosti nejpopulárnějších textových editorů a vývojových prostředí běžících na systému Linux a následně navrhnout a implementovat svobodný textový editor,

který by obsahoval potřebnou funkcionalitu svobodných textových editorů, ale podporoval by i některé nadstardardní funkce z proprietárních řešení. Výsledný textový editor by měl taktéž usnadnit manipulaci s projekty, a tudíž by měl obsahovat i funkcionalitu z vývojových prostředí s tím, že nebude omezen z hlediska počtu podporovaných programovacích jazyků. [3]

Kapitola 2

Analýza existujících řešení

Pro operační systém Linux je dostupných hned několik textových editorů a vývojových prostředí. Některá existující řešení jsou svobodná a lze je provozovat bez omezení, nicméně jsou i řešení proprietární, jejichž dlouhodobé používání vyžaduje zaplacení poplatku. V této kapitole budou popsány a analyzovány nejznámější exitující řešení a následně bude na základě výhod a nevýhod existujících řešení navrhnut textový editor poskytující jak nutnou funkcionalitu obsaženou ve všech zmíněných řešeních, tak některé funkce z placených editorů a vývojových prostředí.

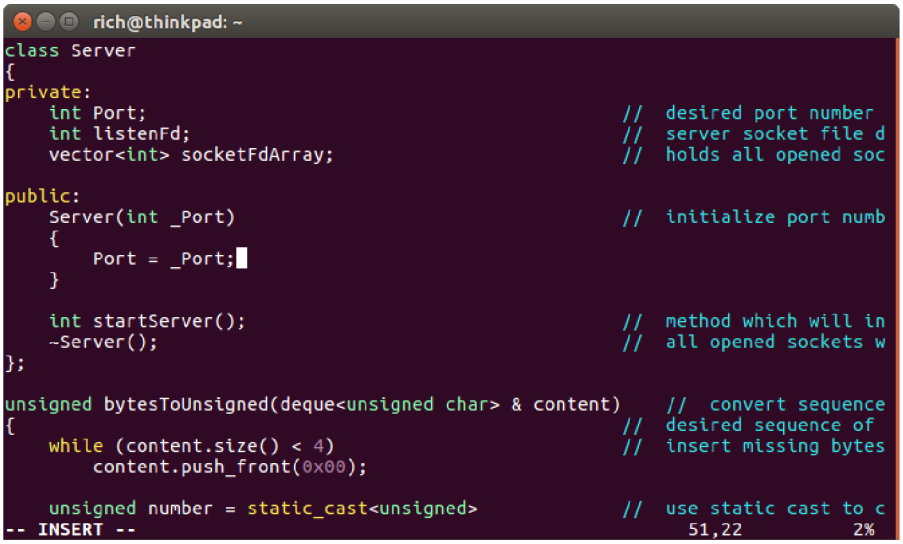
2.1 Vim

Vim je svobodný textový editor určený pro běh v rozhraní příkazové řádky. Velkou výhodou Vimu je jeho výkonnost při editaci textových souborů za pomocí klávesových zkratek reprezentujících příkazy Vimu. Vim lze bez problémů spustit na vzdáleném počítači, protože jeho ovládání je přizpůsobeno příkazové řádce a samozřejmě není potřeba přítomnost kurzoru myši. Jedná se o jeden z nejstarších a také nejspolehlivějších textových editorů pro Linux a je součástí drtivé většiny distribucí. I přes jeho stáří se stále jedná o jeden z nejpoužívanějších programátorských editorů. Dle aktuálních statistik používá Vim pro vývoj webových aplikací zhruba 27% uživatelů. [21] Dále Vim používá zhruba 17% uživatelů jako své primární C++ vývojové prostředí, což je velmi dobrý výsledek vzhledem k tomu, že je Vim pouhý textový editor. [5] Vim podporuje zvýrazňování syntaxe pro širokou škálu programovacích, skriptovacích a značkovacích jazyků, čímž by měl vyhovět naprosté většině uživatelů.

Ovládání Vimu je rozděleno hned do čtyř různých módů, které jsou určeny pro odlišné druhy operací. Pro klasickou editaci textu je přítomen vkládací režim, který je doplňován vizuálním režimem, který umožňuje snadné označování libovolných částí textu. Dále je ve Vimu přítomen nahrazovací režim sloužící k pohodlnému nahrazování znaků v textu. Zbylým režimem je příkazový režim, díky němuž je Vim tak výkonným textovým editorem. V příkazovém režimu je většina příkazů prováděna pouhým stiskem konkrétní klávesy, tudíž je uživatel rychle schopen upravit text dle jeho libosti. V příkazovém režimu je samozřejmě umožněno snadné vyhledávání vzorů v textu a jejich případné nahrazení jinými vzory. Jednotlivé režimy lze snadno a rychle přepínat pomocí jediné klávesy.

Ačkoliv je Vim kvalitní a léty prověřený textový editor, tak mezi znatelnou částí uživatelů nemusí být oblíbený, protože jeho rafinové ovládání vyžaduje značnou trpělivost, aby si na něj uživatel zvykl. Málokterý uživatel si Vim oblíbil hned po jeho první zkušenosti

s ním, protože jeho schopnosti uživatel ocení až po delší době jeho používání. Vzhledem k jeho uživatelskému rozhraní má Vim i některá omezení, která neumožňují například přímou editaci vícero souborů zároveň a také absence ovládání editoru pomocí kurzoru myši je jisté omezení. Pokročilejší textové editory by měly podporovat ovládání ve stylu Vimu, nicméně tomu tak v drtivé většině případů není. Z tohoto důvodu by nově vzniklé textové editory měly tuto situaci napravit. [22]



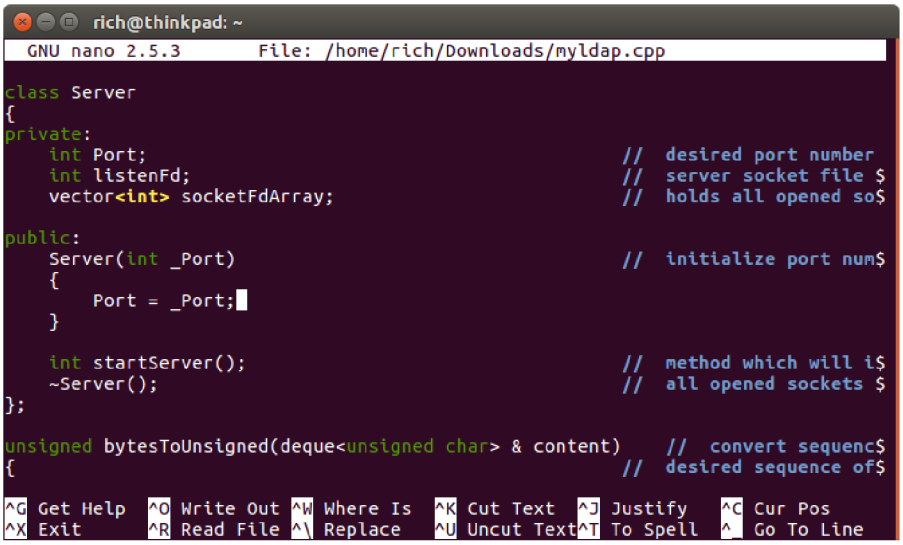
Obrázek 2.1: Textový editor Vim

2.2 nano

Svobodný textový editor nano je stejně jako editor Vim určen pro běh v příkazové řádce, nicméně jeho ovládání je od Vimu naprosto odlišné. Na rozdíl od Vimu zde není přítomno více režimů a přímá editace textu je aktivní neustále. Editace textu je doplněna o množinu klávesových zkratek, které jsou graficky zvýrazněny ve spodní části okna aplikace. Samotné ovládání aplikace je na rozdíl od Vimu velmi snadné a editor může bez potíží používat i nezkušený uživatel. Bohužel editor nenabízí tak efektivní editaci taxtu jako Vim a z hlediska ovládání se jedná o klasický textový editor. Vzhledem k uživatelskému rozhraní není umožněno využití kurzoru myši, a taktéž nelze přímo otevřít vícero textových souborů. Součástí je samozřejmě zvýrazňování syntaxe pro širokou škálu programovacích, skriptovacích a značkovacích jazyků.

Jednotlivé klávesové zkratky jsou v mřížce zvýrazněny ve spodní části okna s tím, že ke každé klávesové zkratce je upřesněn i její význam. Klávesové zkratky jsou prováděny souběžným stiskem klávesy Ctrl a klávesy některého písmena. Pomocí klávesových zkratek lze snadno kopírovat a vkládat označený text. Dále jsou přítomné klávesové zkratky pro vyhledávání v textu a pro nahrazování konkrétních vzorů v textu. Ukládání změn a ukončování aplikace je taktéž prováděno pomocí klávesových zkratek.

Aplikace nano lze nazvat editorem, který má uživatelsky přívětivé ovládání a je vhodný pro editaci souborů umístěných na vzdáleném počítači. Editor nano neobsahuje žádné nadstandardní funkce oproti konkurenčním řešením a z hlediska návrhu nového textového editoru je zohlednění jeho funkcionality zanedbatelné. [12]



Obrázek 2.2: Textový editor nano

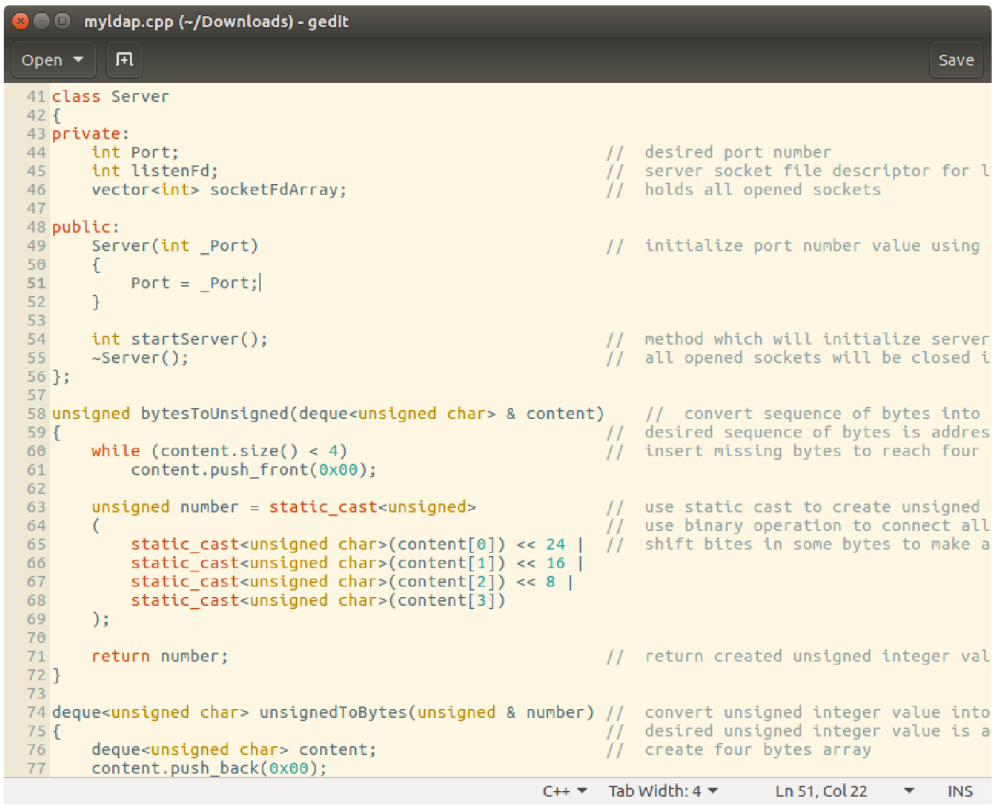
2.3 gedit

Jedná se o svobodný textový editor, který je výchozím editorem pro Linuxové distribuce využívající grafické prostředí GNOME. Editor využívá grafické uživatelské rozhraní a jeho funkcionalita je poměrně rozsáhlá a umožňuje otevření hned několika textových souborů zároveň a to jak v oddělených záložkách, tak s využitím rozděleného okna aplikace do více textových editorů. Editor samozřejmě podporuje zvýrazňování syntaxe pro širokou škálu jazyků a nabízí několik barevných motivů pro úpravu jeho vzhledu. Uživatelské rozhraní aplikace je jednoduché a bez potíží ho pochopí i začátečníci. Samotný vývoj editoru se přímo odvíjí od vývoje grafického prostředí GNOME. Nespornou výhodou tohoto editoru je jeho plynulost a jeho velmi dobrá optimalizace, což rozhodně neplatí pro všechny textové editory s grafickým uživatelským rozhraním. Navíc samotná doba jeho spouštění je krátká i na starších počítačích.

Ačkoliv se jedná o textový editor s grafickým uživatelským rozhraním, tak pro překlad zdrojového kódu je potřeba instalace pluginu[1]. Jakákoliv manipulace s projekty není přímo podporována a nejedná se o editor vhodný pro práci s většími projekty. Dále není možné přibližování a oddalování textu, bez závislosti na velikosti písma editoru a také není možné mít nastavenou různou velikost písma v různých záložkách. Editor dále neumožňuje zobrazení seznamu funkcí, a zobrazení stromu tříd také není možné, ačkoliv konkurenční řešení tuto funkcionalitu alespoň v omezené míře podporují. Editor také nepodporuje zabalování bloků kódu uvnitř blokových závorek, což pravděpodobně nevyužívá každý uživatel, nicméně konkurence tuto funkcionalitu obvykle podporuje.

Ve výsledku se jedná pouze o základní textový editor se zvýrazňováním syntaxe. Editor nenabízí žádné nadstandardní funkce, a z pohledu návrhu nového textového editoru je vhodné zahrnout jeho veškerou podstatnou funkcionalitu. [7]

[1] Software umožňující rozšíření funkcionality aplikace



Obrázek 2.3: Textový editor gedit

2.4 Kate

Kate je velmi oblíbený svobodný textový editor, který je výchozím editorem pro grafické prostředí KDE[2] a samozřejmě má grafické uživatelské rozhraní. Oproti svému přímému konkurentovi z prostředí GNOME[3] se jedná o pokročilejší aplikaci s rozsáhlejší funkcionalitou. Kate na rozdíl od editoru gedit umožňuje snadné zabalování bloků kódu v blokových závorkách, a taktéž je umožněno libovolné přibližování a oddalování textu bez závislosti na velikosti písma. Kate navíc umožňuje nastavení odlišných barevných motivů pro různé otevřené záložky. Samozřejmostí je podpora velkého množství programovacích, skriptovacích a značkovacích jazyků. Na poměry textového editoru jako jediný umožňuje zobrazení stromu tříd na základě zdrojového kódu, a to včetně skoku do zdrojového kódu přímo na definici zvoleného uzlu stromu, což je funkcionalita, kterou disponují jen některá vývojová prostředí.

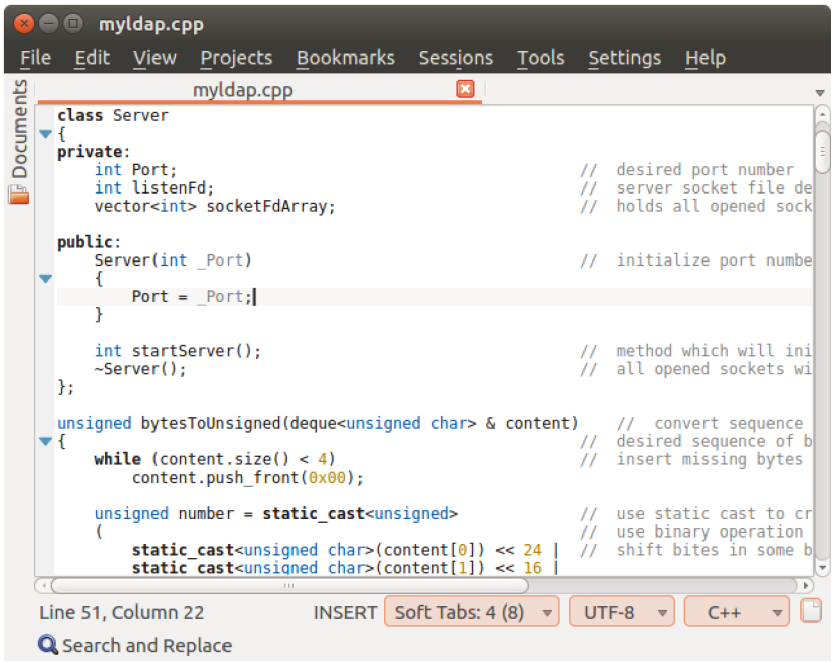
Další velkou výhodou editoru Kate je možnost postupného posouvání označeného úseku textu po řádcích, což nabízejí jen některé textové editory, mezi které konkurenční gedit nepatří. Mezi další zajímavé funkce aplikace patří automatické zvýraznění všech stejných identifikátorů v případě, že daný identifikátor uživatel ručně označí, což může být velmi užitečné v případě, kdy chce uživatel vyhledat umístění daného slova v textu. Kate se rozvíjí postupně s grafickým prostředím KDE a postupně se rozšiřuje jak jeho funkcionalita, tak

[2]Grafické prostředí založené na knihovně Qt
[3]Grafické prostředí založené na knihovně GTK/GTK+

jeho stabilita a spolehlivost. Samotný textový editor Kate je velmi plynulý a je poměrně nenáročný z hlediska využití systémových zdrojů.

Ačkoliv Kate umožňuje pohodlný překlad zdrojového kódu, tak pokročilá manipulace s projekty není možná ve srovnání s vývojovými prostředími. Mírnou nevýhodou této aplikace je delší časová prodleva při jejím spouštění. Doba spouštění je rozhodně kratší, než v případě vývojových prostředí, nicméně je rozhodně delší, než v případně výchozího editoru pro grafické prostředí GNOME. Zobrazení stromu tříd má samozřejmě jistá omezení, mezi která patří především absence zobrazení viditenosti třídních atributů a metod. Uživatel tedy přímo ze stromu nerozezná, zda je daná metoda soukromá, veřejná či chráněná.

Editor Kate je velmi kvalitní aplikací a při návrhu nového textového editoru je potřeba zohlednit velkou část jeho funkcionality. Především zobrazení stromu tříd je na poměry textových editorů nevídaná funkcionalita a nově vzniklý editor by měl taktéž umožnit zobrazení stromu tříd. [9]



Obrázek 2.4: Textový editor Kate

2.5 Atom

Atom je volně dostupný programátorský editor s grafickým uživatelským rozhraním, jehož popularita v poslední době výrazně stoupá. Atom se soustředí na podporu velkého množství pluginů, jejichž implementace má být oproti konkurenci znatelně jednodušší a rychlejší. Samozřejmě je uživatelům umožněno vytvářet, používat a samozřejmě sdílet vlastní pluginy, které mohou rozšířit funkcionalitu aplikace. Další velkou výhodou Atomu je vestavěná podpora manipulace s Git[4] repozitáři, což konkurenční programátorské editory neumožňují. Z hlediska možností editace zdrojového kódu jsou možnosti Atomu na srovnatelné úrovni s textovým editorem Kate, což z Atomu dělá důstojný textový editor. Mezi další přednosti Atomu patří možnost úprav zvýrazňování syntaxe dle libosti uživatele, což konku-

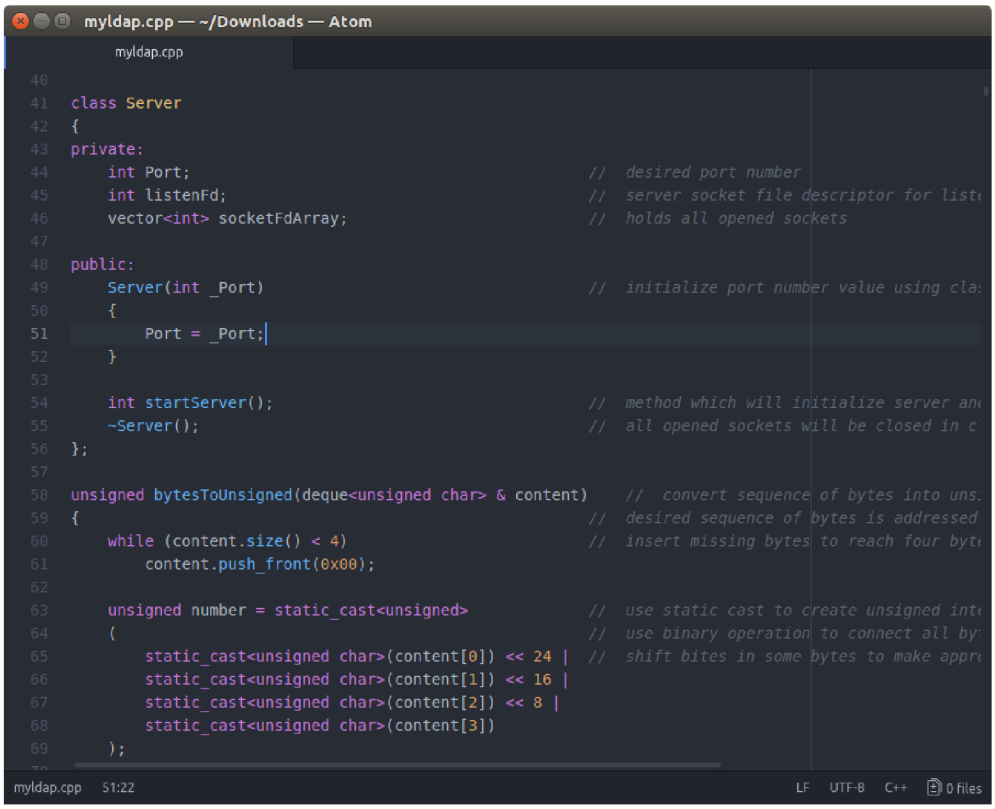
[4]Nástroj umožňující podporu verzování v repozitářích

rence přímo neumožňuje. Atom dále podporuje našeptávání při psaní textu, což je funkce, která může uživatelům výrazně ušetřit práci nutnou pro vložení například těla cyklu v jazyce C++[5], kdy editor rozezná první klíčové slovo výrazu a pomocí vyskakovací nabídky umožní uživateli automatické doplnění těla cyklu. Našeptávání při psaní textu je záležitost, která je běžná pouze pro vývojová prostředí, tudíž je její přítomnost v textovém editoru velmi dobrou zprávou. Atom umožňuje snadnou správu projektů, v rámcí níž si uživatel snadno otevře konkrétní adresář projektu, a po instalaci příslušného pluginu je následně možné sestavení projektu. Oproti výše zmíněné konkurenci přináší Atom automatické uložení stavu aplikace po jejím ukončení, což zajistí, že aplikace bude při dalším spuštění ve stejném stavu jako před ukončením. Dle aktuálních statistik používá Atom zhruba pětina uživatelů pro vývoj webových aplikací, čímž se editor přibližuje oblíbenému Vimu. [21] Samozřejmostí je podpora většího množství programovacích, skriptovacích a značkovacích jazyků, nicméně oproti výše zmíněné konkurenci se jedná o znatelně nižší počet.

Bohužel je Atom doprovázen také několika nevýhodami, které mohou některé uživatele odradit, nicméně vždy závisí na prioritách daného uživatele. Stěžejní nevýhoda Atomu přímo souvisí s jeho implementací. Výše zmíněné editory jsou buďto implementovány v jazyce C++, což je činí skromnými z hlediska náročnosti na systémové zdroje, a také doba jejich spouštění je obvykle krátká. Pro implementaci Atomu bylo naopak využito jazyků a nástrojů pro vývoj webových aplikací. Tato skutečnost má nicméně negativní dopad na provozní vlastnosti editoru. Atom je na poměry textového editoru hodně robustní aplikací a jeho běžná spotřeba operační paměti je opravdu vysoká a překonává i některá vývojová prostředí. Spouštěcí doba Atomu je nepříjemně dlouhá a je delší, než v případě některých vývojových prostředí. Plynulost uživatelského rozhraní Atomu je také na poměrně špatné úrovni a například při posuvu náhledu textu pomocí posuvné lišty, lze vypozorovat zpoždění, které není vyloženě nepříjemné, ale v porovnání s editorem Kate je vidět obrovský rozdíl, kdy posuv náhledu textu v editoru Kate je velmi rychlý a svižný. Pro některé uživatele nemusí být náročnost na sytémové zdroje překážkou, ale rozhodně se jedná o věc, která může být v porovnání s konkurencí nepříjemná a navíc je potřeba zohlednit skutečnost, že hardwarová náročnost editorů byla srovnávána na aktuálním počítači s nadprůměrným výkonem, tudíž na starších počítačích by bylo možné vypozorovat ještě výraznější rozdíly.

Dále je také vhodné zmínit, že oproti textovému editoru Kate zde není přítomná možnost zobrazení stromu tříd, což není vyloženě nevýhoda, nicméně konkurence je v tomto ohledu popředu. Další funkcí, kterou editor Kate nabízí, ale v Atomu není k dispozici, je automatické zvýraznění všech stejných identifikátorů při ručním označení některého z nich. I přes zmíněná negativa je Atom vydařeným programátorským editorem, z jehož rozsáhlé funkcionality se dá rozhodně inspirovat při návrhu nového textového editoru. Navržený textový editor by se také měl vyvarovat vysokým provozním nárokům, protože dobře optimalizovaná aplikace šetří nejen operační paměť, ale potenciálně se i rychleji spouští. [1]

[5]Kompilovaný a multiparadigmatický programovací jazyk, který je primárním jazykem Qt frameworku



Obrázek 2.5: Textový editor Atom

2.6 Sublime Text

Sublime Text je programátorský editor s grafickým uživatelským rozhraním, který svým uživatelům nabízí velké množství užitečných funkcí. Díky své funkcionalitě se bezpochyby jedná o nejpokročilejší textový editor. Sublime Text obsahuje vše z hlediska funkcionality, kterou obsahují výše zmíněná řešení. Výjimkou je pouze integrace podpory práce s Gitem. Editor obsahuje velké množství funkcí, díky kterým výrazně převyšuje svou konkurenci. Sublime Text je poměrně novou aplikací, nicméně její schopnosti se postupem času vylepšují a taktéž její spolehlivost je na velmi dobré úrovni. Aktuálně zhruba 32% uživatelů používá Sublime Text pro vývoj webových aplikací, čímž editor v tomto ohledu překonává populární Vim. [21] Sublime Text využívají jako C++ vývojové prostředí pouze 2% uživatelů, což je výrazně menší podíl v porovnání s Vimem. [5]

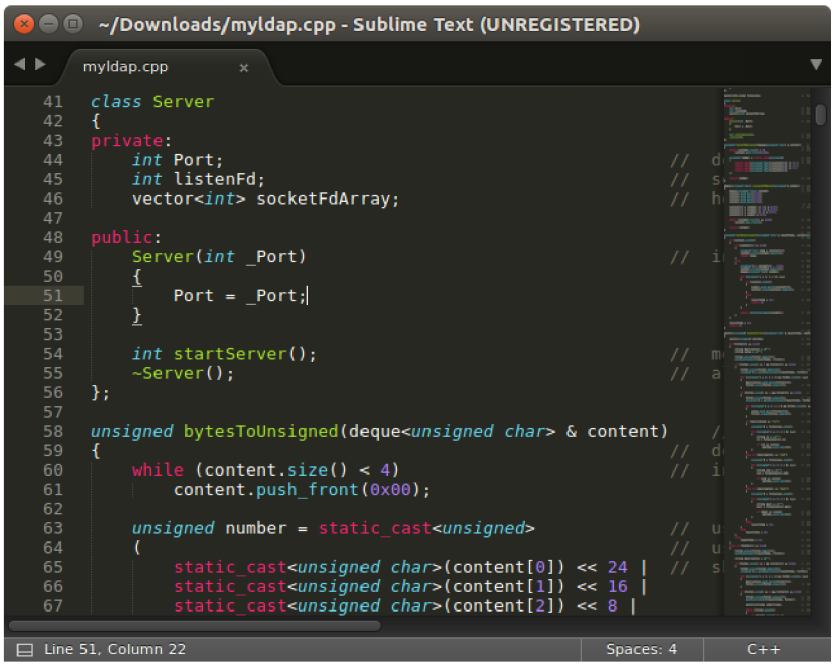
Stěžejní funkcí Sublime Textu je možnost editace textu pomocí vícero textových kurzorů. Počet aktivních kurzorů není omezen, což umožňuje rychle a pohodlně upravovat více částí textu zároveň. Uživatel má možnost si postupně aktivovat nové textové kurzory na libovolných pozicích v textu. Veškeré operace, které uživatel provede pomocí klávesnice, se projeví v rámci všech aktivních kurzorů. Editor taktéž umožňuje rychlé přidávání nových kurzorů ve směru následujích řádků textu s tím, že pozice kurzoru na nově vzniklém řádku zůstává stejná. Vícero textových kurzorů je automaticky vytvořeno i v případě, že uživatel chce označit všechny výskyty zvoleného vzoru při vyhledávání, čímž je umožněna pohodlná editace textu na nově nalezených pozicích.

Další věc, která uživatele rozhodně potěší, je zvýrazňování identifikátorů funkcí a metod, což kromě Atomu neumožňuje žádný konkurenční editor a dokonce ani vývojové prostředí. Zvýrazňování identifikátorů funkcí je velmi užitečné v případě editace zdrojového kódu v některém z objektově orientovaných jazyků, kdy zvýrazněné názvy volaných metod výrazně napomáhají přehlednosti zdrojového kódu. V porovnání s editorem Kate není zabudováno zobrazení stromu tříd, ale je zde přítomnost rychlého přehledu funkcí a metod ve zdrojovém kódu, včetně možnosti přímého skoku na odpovídající pozici daného prvku. Snadný a rychlý přesun na odpovídající pozici ve zdrojovém kódu je doplněn o možnost skákání mezi deklarací a definicí některého z elementů daného aktivního jazyka. Pokud se chce uživatel například přesunout na deklaraci funkce, kterou momentálně edituje, tak stačí pouze najet kurzorem myši na identifikátor dané funkce a aplikace nabídne uživateli možnost přímého skoku na její deklaraci. Samozřejmě i v případě, že se deklarace nachází v jiném otevřeném souboru.

V rámci manipulace s projekty je editor na velmi dobré úrovni a umožňuje uživateli jejich pohodlnou správu. Stačí pouhé zvolení seznamu souborů, které editor reprezentuje jako celý projekt. Sublime Text umožňuje sestavování projektů, nicméně je potřeba nastavit překlad v konfiguračním souboru, což je obtížnější činnost, nicméně je tímto způsobem uživatelům umožňeno si dle své libosti nastavit parametry překladu projektu. Samotné sestavení projektu je po jeho správném nastavení již snadné. Aplikace je při spuštění ve stejném stavu, v jakém byla před posledním ukončením, a to včetně uložení všech prázdných záložek, což umožňuje snadnou editaci textu bez nutnosti cokoliv ukládat na disk počítače.

Editor samozřejmě poskytuje zvýrazňování syntaxe pro velké množství jazyků, které je srovnatelné s Vimem, což je velmi dobrý výsledek. Z výkonnostního hlediska je Sublime Text na vynikající úrovni a i přes svou bohatou funkcionalitu je velmi skromný z pohledu systémových nároků a na jeho provoz postačí téměř libovolný počítač. Spotřeba operační paměti je při běžném použití relativně nízká a plynulost grafického prostředí je ze všech doposud zmíněných editorů na nejlepší úrovni a uživatelům je dokonce umožněno vypnutí animací pro snížení provozních nároků.

Ačkoliv by se Sublime Text mohl na první pohled jevit jako téměř dokonalý programátorský editor, který přináší oproti své konkurci opravdu hodně možností navíc, nicméně skutečnost je mírně odlišná. Sublime Text je opravdu špičková aplikace, jenomže oproti své konkurenci není volně dostupná. Pro její dlouhodobé používání je nutné zaplatit částku 80 USD jeho vývojářům. Tato částka rozhodně není malá, ale naštěstí zakoupení licence neomezuje její použití na více počítačích zároveň. Z pohledu návrhu svobodného textového editoru je Sublime Text ideálním vzorem pro inspiraci. Je vhodné zahrnout některé jeho vlastnosti do svobodné aplikace a nabídnout tak placenou funkcionalitu úplně zdarma. [20]



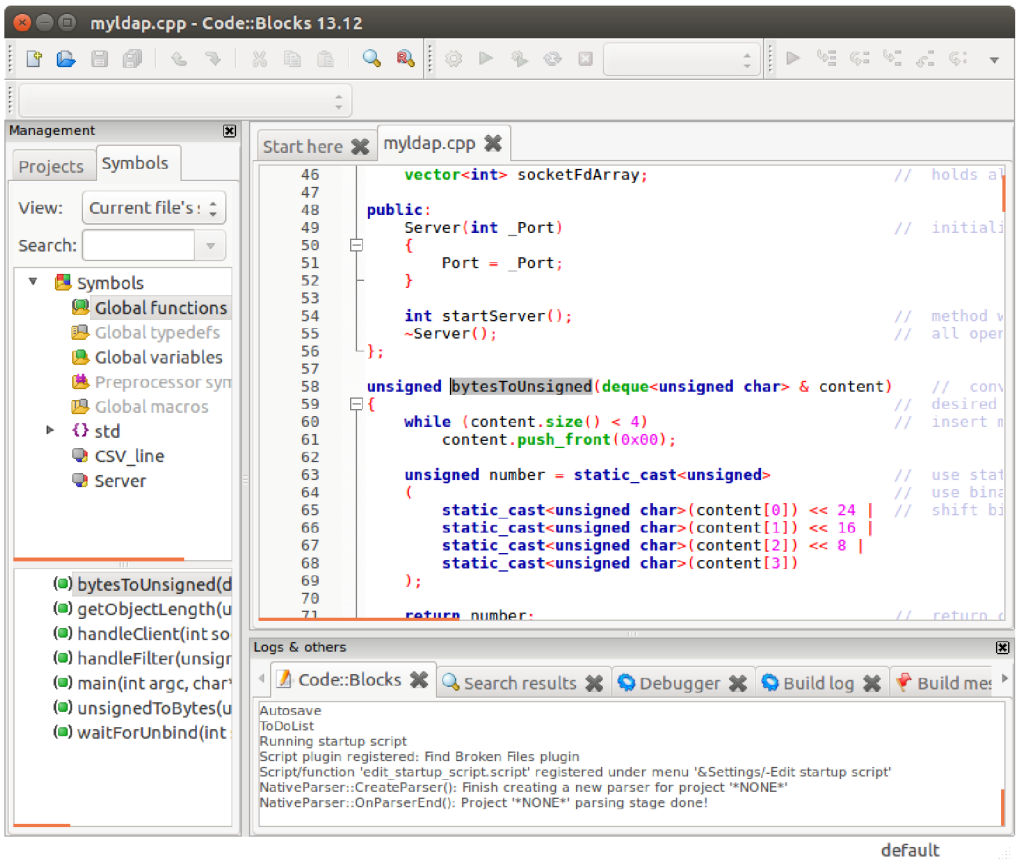
Obrázek 2.6: Textový editor Sublime Text

2.7 Code::Blocks

Code::Blocks je svobodné vývojové prostředí určené pro pohodlný vývoj aplikací v jazycích C a C++. [2] Prostředí umožňuje komfortní manipulaci s C++ projekty a je umožněno okamžité sestavení projektu včetně spuštění samotné aplikace. Samozřejmostí je také podpora ladění aplikace pomocí vestavěného ladícího nástroje a při editaci textu lze vkládat zarážky na libovolné řádky kódu. Vývojové prostředí Code::Blocks je z hlediska editace zdrojového kódu na velmi dobré úrovni, což zhruba odpovídá společné funkcionalitě výše zmíněných řešení. Na rozdíl od všech výše zmíněných editorů je zde navíc přítomna barevná indikace změn na konkrétních řádcích zdrojového kódu, což umožňuje uživateli snadno nalézt úseky kódu, na nichž v minulosti prováděl změny. Indikace změn v kódu je velmi užitečnou funkcionalitou, jejíž přítomnost by byla přínosná i u klasických textových editorů, čímž se řadí mezi další potenciální funkci z hlediska návrhu svobodného textového editoru. Code::Blocks dále umožňuje inteligentní našeptávání kódu při psaní, které oproti Sublime Textu, umožňuje našeptávání na základě syntaxe jazyků C a C++ a to včetně našeptávání názvů vestavěných funkcí. Aktuálně necelá 3% uživatelů využívájí Code::Blocks jako své hlavní C++ vývojové prostředí. [5]

Code::Blocks nicméně není dokonalou aplikací a má i své nevýhody. Poměrně nepříjemnou věcí je neumožnění pohodlného skrytí uživatelského prostředí a zobrazení pouze textového editoru, což pro některé uživatele nemusí být potíž, nicméně vzhledem k propracovanosti uživatelského rozhraní by snadné skrytí rozhraní mohli ocenit především majitelé přenosných počítačů, kterým velké množství ovládacích prvků kolem samotného textového editoru, může někdy překážet. Dalším negativem je obtížná úprava barevného motivu aplikace. Opět se může jednat o věc, kterou část uživatelů přehlédne, ale v porovnání s výše zmíněnými řešeními, se rozhodně jedná o znatelné omezení. Z pohledu návrhu svobodného textového editoru by bylo vhodné snadné schování uživatelského rozhraní zahrnout, aby

byla uživateli umožněna maximální koncentrace na zdrojový kód. Samozřejmostí je také omezení prostředí na práci s jazyky C a C++, s tím, že zde není podpora dalších programovacích jazyků, což je pro vývojová prostředí běžné a nelze to přímo považovat za negativum. [4]



Obrázek 2.7: Vývojové prostředí Code::Blocks

2.8 Qt Creator

Qt Creator je vývojové prostředí určené pro práci s projekty, které využívají framework[6] Qt v rámci své implementace. Qt Creator je velmi pokročilé vývojové prostředí umožňující také grafický návrh aplikací založených na frameworku Qt a samozřejmostí je také propracované ladění aplikací. Qt Creator nabízí pohodlnou manipulaci s projekty v programovacím jazyce C++ a deklarativním jazyce QML[7]. Aktuálně zhruba 12% uživatelů používá Qt Creator jako své primární C++ vývojové prostředí, čímž se řadí až za populární Vim. [5]

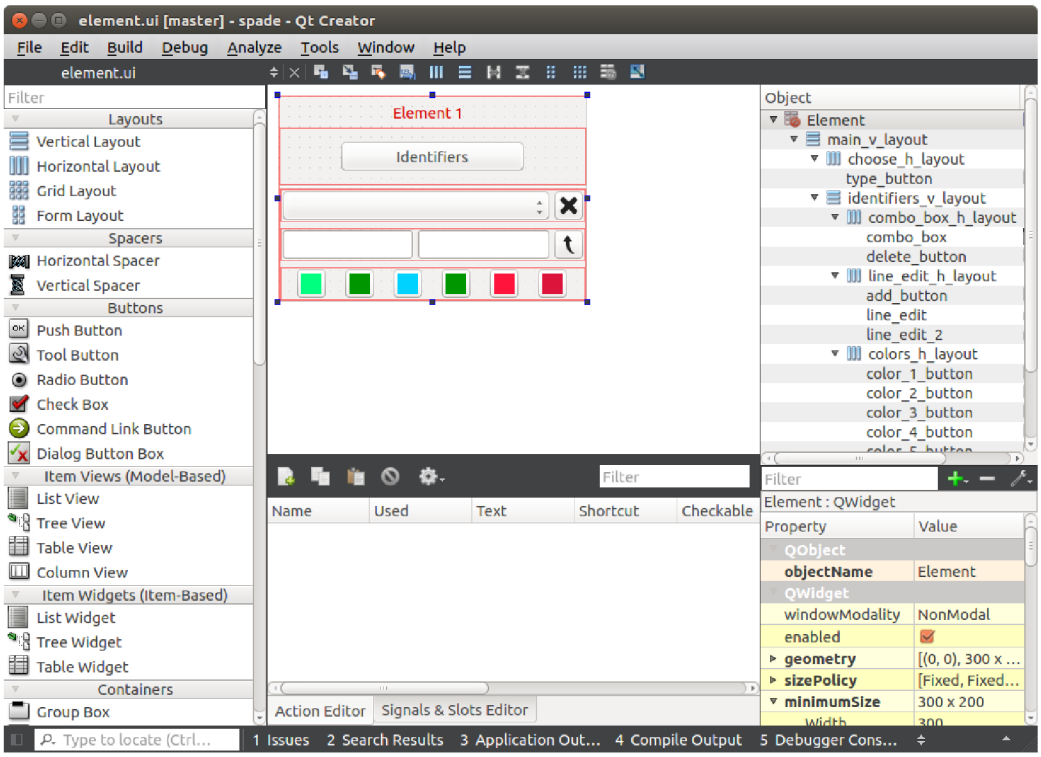
Qt Creator obsahuje vestavěný textový editor, který nabízí na poměry textových editorů nevídanou funkcionalitu. Našeptávání zdrojového kódu je zde posunuto ještě o úroveň výše, než u konkurence, protože našeptávání je prováděno i v rámci Qt knihoven a uživatel má možnost v rámci našeptávání přepínat například mezi možnými tvary atributů u přetížených metod, což velmi usnadňuje uživateli práci a navíc uživatel nemusí studovat

[6] Komplexní sada knihovních modulů s dalšími podpůrnými nástroji a programy usnadňujícími a urychlujícími vývoj aplikací

[7] Značkovací jazyk umožňující daklarativní způsob programování a tvorbu uživatelských rozhraní

dokumentaci, protože aplikace mu vše potřebné našeptá při psaní. Zásadní výhodou Qt Creatoru je možnost zobrazení stromu celého projektu, v němž jsou obsaženy třídy, struktury, výčtové typy, funkce a proměnné, které se nacházejí v projektu. Uvnitř každé třídy jsou samozřejmě zobrazeny i její atributy a metody. Strom prvků je velmi propracovaný a po klepnutí na libovolný uzel stromu dojde k přímému skoku do zdrojového kódu na deklaraci daného prvku. Zohledněny jsou také viditelnosti konkrétních prvků a například soukromé atributy dané třídy jsou zvýrazněny odlišně, než atributy veřejné. Vestavěný editor dále umožňuje snadný přesun na deklaraci některého prvku ve zdrojovém kódu za pomocí stisku tlačítka Ctrl a klepnutím na daný prvek. Samozřejmostí je také barevná indikace změn na konkrétních řádcích zdrojového kódu. Qt Creator dále nabízí automatické zvýrazňování stejných klíčových slov v textu, které na rozdíl od Sublime Textu, nevyžaduje označení všech znaků daného slova, ale pouhé najetí textového kurzoru na dané slovo. Ačkoliv je Qt Creator rozsáhlé vývojové prostředí s mnoha ovládacími prvky, tak umožňuje snadné a pohodlné skrytí uživatelského rozhraní a nechá zobrazen pouze textový editor, aby se uživatel mohl maximálně soustředit na zdrojový kód.

Mezi hlavní nevýhodu Qt Creatoru patří podpora vývoje aplikací pouze ve dvou jazycích, kterými jsou programovací jazyk C++ a deklarativní jazyk QML. Dále je nutné zmínit, že Qt Creator je proprietární vývojové prostředí, nicméně pro tvorbu svobodné aplikace je možné jej využít zdarma včetně celého frameworku. [3]



Obrázek 2.8: Vývojové prostředí Qt Creator

2.9 Shrnutí

Cílem analýzy bylo získání přehledu o schopnostech nejznámějších existujících řešení na platformě Linux. Na základě získaného přehledu bylo potřeba navrhnout svobodný textový

editor, který bude zahrnovat nutnou funkcionalitu společnou pro existující řešení, a dále bude poskytovat funkcionalitu, která je přítomna jen v některých textových editorech a vývojových prostředích, nebo v řešeních, která jsou proprietární.

Pro nabídnutí co nejvíce funkcí uživateli, by měl navržený textový editor poskytovat grafické uživatelské rozhraní. Textový editor by tedy měl umožnit ovládání jak pomocí kurzoru myši, tak pomocí klávesnice. Dále je vhodné umožnit uživateli manipulaci s více soubory zároveň pomocí záložek. Uživateli by mělo být umožněno změnit velikost a styl písma s tím, že aplikace musí nabízet výběr z několika barevných motivů. Doporučená je přítomnost vyhledávání vzorů v textu, včetně vyhledávání za pomoci regulárních výrazů. Zvýrazňování syntaxe by mělo zahrnovat co nejvíce programovacích, skriptovacích a značkovacích jazyků. Počet podporovaných jazyků by potenciálně mohl být rozšiřitelný za pomoci vestavěného editoru zvýrazňování syntaxe pro vlastní jazyky, díky čemuž by se aplikace odlišila od konkurence, která neumožňuje rozšíření počtu podporovaných jazyků. Editor zvýrazňování syntaxe, by měl v ideálním případě využívat grafického uživatelského rozhraní a měl by poskytovat pohodlný návrh zvýrazňování syntaxe za pomoci množiny regulárních výrazů. Všechny návrhy na implementaci doposud nevídané funkcionality, by měly být do nově navrženého editoru zahrnuty, s cílem dosažení co nejvíce inovací. Výsledná aplikace je samozřejmě cílena na platformu Linux, nicméně zajištění kompatibility zdrojových kódů i pro ostatní platformy, je vítaným krokem ze strany vývojáře aplikace.

Mezi pokročilé vlastnosti navrženého textového editoru, by mělo spadat automatické odsazování slov při editaci a automatické doplňování uzavíracích závorek. Oblíbenou funkcí z existujících řešení je taktéž snadná konverze označených znaků na malá či velká písmena nebo převrácení velikosti písmen. Je vhodné nevynechat možnost posunu označených řádků vzhůru či níže po jednotlivých řádcích. Navržený textový editor by měl uživateli umožnit snadné a pohodlné skrytí většiny ovládacích prvků a poskytnout mu maximální koncentraci na editaci zdrojového kódu. Doporučená je přítomnost číslování řádků doplněná o možnost zabalení bloku kódu. Na poměry textového editoru je také vítána barevná indikace změn textu na konkrétních řádcích. Dále by nemělo chybět zvýrazňování názvů funkcí a třídních metod ve zdrojovém kódu. V rámci zvýrazňování syntaxe je doporučeno zahrnout zvýraznění všech stejných slov ve zdrojovém kódu v případě, že je na jedno z daných slov najeto textovým kurzorem nebo je celé slovo označeno. Pro maximální komfort uživatele je vhodné implementovat našeptávání při psaní, což je funkcionalita, díky které by měl mít uživatel možnost snadno vkládat obecné konstrukce daného jazyka, aby si ušetřil práci při časté tvorbě stejných konstrukcí. V rámci ušetření uživatelovi práce je vhodné zahrnout automatické ukládání stavu editoru při jeho ukončení, což by uživateli ušetřilo práci s otevíráním souborů při opětovném spuštění aplikace.

Nově navržený textový editor by měl v ideálním případě poskytovat i jistou část funkcionality poskytované vývojovými prostředími. Mezi tuto funkcionality patří manipulace s projekty, čímž by uživateli bylo umožněno pracovat s množinou zdrojových souborů jako s celkem. Manipulace s projekty by taktéž měla být doplněna o možnost překladu zdrojových kódů. Překlad musí být umožněn jak v rámci jediného zdrojového souboru, tak v rámci celého projektu. Práce s velkým množstvím zdrojových souborů by bylo vhodné udělat pohodlnější za pomoci rozšíření okna aplikace do několika samostatných editorů, což ocení především majitelé obrazovek s vysokým rozlišením. Manipulace s projekty by měla být doplněna o možnost zobrazení stromové struktury obsahující všechny třídy, struktury, výčtové typy, funkce a proměnné, které se nacházejí v daném projektu. Pro pohodlný průchod zdrojovým souborem je vhodné zařadit možnost přímého skoku do zdrojového kódu na definici daného uzlu stromové struktury.

Všem uživatelům textového editoru Vim, by měla posloužit vestavěná možnost editace textu za pomocí příkazů editoru Vim, což je nevídaná funkcionalita na poměry textových editorů. Velmi vítanou funkcionalitou, kterou nabízí pouze konkurenčí editor Sublime Text, je možnost souběžné editace textu za pomocí vícero textových kurzorů, nicméně implememntace této funkcionality patří mezi náročnější úkoly. Vestavěná podpora práce s aplikací Git pro práci s repozitáři, je taktéž vítanou a téměř nevídanou funkcionalitou, nicméně její implementace rozhodně není jednoduchá.

V rámci návrhu svobodného textového editoru byla taktéž aktivně vyhledávána doposud nevídaná funkcionalita, kterou neposkytuje žádné z existujících řešení. Především na diskuzních fórech určených pro vývojáře, se poměrně často nacházejí dotazy a návrhy na téma doposud nevídané funkcionality v textových editorech. Mezi často dotazovanou funkcionalitu patří možnost dočasného skrytí všech komentářů ve zdrojovém kódu, což samozřejmě může přispět k lepší čitelnosti zdrojového kódu v době, kdy jsou komentáře skryté. Mezi další často dotazovanou funkcionalitu textového editoru patří manuální zvýrazňování libovolných částí zdrojového kódu, kdy manuálně bude zýrazněno pouze pozadí konkrétních znaků, aby nedošlo ke konfliktu s výchozím zvýrazňováním syntaxe pro daný jazyk. Je potřeba zmínit, že manuální zvýrazňování musí být k dispozici i v případě editace prostého textu.

Kapitola 3

Návrh řešení

Cílem této kapitoly je provést návrh stěžejních komponent textového editoru. V následující kapitole bude na základě tohoto návrhu popsána implementace výsledného textového editoru. Návrh komponent přímo vychází ze shrnutí existujících řešení z předcházející kapitoly, v němž byl popsán seznam nutné a doporučené funkcionality, kterou by měl nově navržený textový editor poskytovat. Před popisem návrhu jednotlivých komponent, je nutné zmínit, že implementace některé doporučené funkcionality, je buďto velmi náročná nebo špatně realizovatelná v nástrojích použitých pro implementaci, a proto byla vyřazena z návrhu.

3.1 Uživatelské rozhraní

Jak již bylo zmíněno v předcházející kapitole, tak je ideální, aby textový editor poskytoval grafické uživatelské rozhraní. Cílem je, aby prostředí poskytovalo co nejvíce funkcí a možností uživateli, nicméně je vítána také co největší kompaktnost prostředí. Možnost skrytí všech svých ovládacích prvků je doporučená. Styl grafického prostředí by se měl automaticky přizpůsobit grafickému prostředí operačního systému, což zahrnuje jak tvar tlačítek, tak posuvných lišt a ostatních ovládacích prvků. Styl písma v textovém editoru by taktéž měl odpovídat výchozímu neproporcionálnímu písmu daného operačního systému. Jak pozadí textového editoru, tak výchozí barva písma by měla být odvozena z aktuálně nastaveného barevného motivu. Důležitou vlastností musí být možnost libovolné manipulace s velikostí okna aplikace s tím, že musí být přítomen způsob aktivace režimu zobrazení aplikace na celou obrazovku.

Pro dosažení co nejvyšší kompaktnosti prostředí je vhodné se odlišit od konkurence, která využívá kombinace navigačního menu a několika dalších ovládacích prvků pro volbu zvýrazňovaného jazyka, nastavení míry odsazení a samostatného řádku se záložkami. Prostředí je vhodné řešit pomocí jediné navigační lišty, v níž budou uloženy všechny potřebné ovládací prvky. Pro pokrytí všech potřebných ovládacích prvků by měla být navigační lišta hierarchická. Kořenovou lištou musí být výchozí lišta určena pro manipulaci s konkrétními soubory, z níž by se mělo dát dostat do ostatních lišt se zbylými ovládacími prvky. Mezi zbylé navigační lišty by měla patřit především lišta navigační, lišta vyhledávací, lišta s pokročilými funkcemi, lišta projektů a lišta pro práci se soubory. Velkou výhodou toho systému je jeho kompaktnost, kdy uživateli pro přístup ke všem funkcím stačí jediná lišta v okně editoru a navíc samotná lišta zabírá pouze minimum místa v okně aplikace. Dále je možné přímo mapovat každý ovládací prvek v liště na jednu funkční klávesu na klávesnici ve směru zleva doprava, což umožňuje velmi pohodlný průchod skrze hierarchii lišt. Navigační lištu

by taktéž mělo být možné schovat pomocí některé z kláves na klávesnici. Kromě samotného editoru a navigační lišty již tím pádem není v okně aplikace potřeba žádný jiný ovládací prvek.

3.2 Vyhledávání

Jednou z navigačních lišt by měla být lišta určená pro vyhledávaní v textu. Vyhledávání by mělo být možné jak pomocí textového vzoru, tak pomocí regulárního výrazu. Nalezenými výsledky musí být možné procházet jak směrem níže v textu, tak směrem vzhůru k předcházejících výsledkům. Pro snadné nahrazování slov je vhodné, aby bylo ve vyhledávací liště možno zadat jak vyhledávací vzor, tak text, kterým bude vyhledaný text nahrazen. Nahrazení slova by mělo samozřejmě proběhnout na základě stisku tlačítka pro nahrazení s tím, že je také nutná přítomnost tlačítka pro nahrazení všech výsledků. Dále je vhodné do lišty zařadit i tlačítko na aktivaci citlivosti na malá a velká písmena a taktéž aktivace vyhledávání pouze celých slov.

3.3 Editace textu

Samotný textový editor samozřejmě musí podporovat nutnou funkcionalitu z hlediska samotné editace textu, mezi kterou patří podpora označování textu jak pomocí kurzoru myši, tak za pomocí klávesnice. Nutností je taktéž podpora pohybu textového kurzoru jak po znacích, tak po slovech a celých větách. Samozřejmostí je možnost zkopírování označeného textu do schránky a možnost vložení textu na libovolnou pozici v textu. Nutnou funkcí textového editoru je možnost vrácení předchozích změn, ať už pomocí klávesové zkratky nebo tlačítka v navigační liště. Dopředný návrat na novější změny pak následně přímo doplňuje návrat zpětný. Textový editor také musí správně zobrazovat znaky různých národních abeced.

Programátorský editor by měl taktéž umožňovat automatické odsazování textu dle aktuálního počtu bílých znaků, které předcházejí samotnému kódu. Začátek psaní textu po odřádkování, by tedy měl být zarovnán se zbytkem zdrojového kódu. Dále je vhodné, aby textový editor automaticky doplňoval uzavírací závorky, což zahrnuje všechny typy závorek včetně blokových a ostrých. Automatické doplnění zdrojového kódu by se dále mělo lišit podle typu nastaveného jazyku. Například, při odřádkování za uzavírací závorkou hlavičky některého cyklu v jazyce C++, by došlo automatickému doplnění blokových závorek a taktéž k posunutí odsazení uvnitř těla cyklu. V předcházející kapitole byla doporučena implementace našeptávání syntaxe s tím, že našeptané výrazy by byly zobrazeny ve vyskakovací nabídce hned vedle textového kurzoru. Implementace samotné logiky našeptávání velmi náročná a z časových důvodů nebyla do navrženého textového editoru zahrnuta. Nicméně se jedná o funkcionalitu, která je typická pro vývojová prostředí a její absence v textovém editoru, nečiní velké omezení. Částečnou kompenzací našeptávání syntaxe může samostatná lišta, v níž byly k dispozici tlačítka, která by umožňovala přímé vložení šablony dané konstrukce jazyka do zdrojového kódu, což je z hlediska vkládání šablon velmi pohodlný způsob. Pro každý z podporovaných jazyků musí být samozřejmě k dispozici odlišné šablony.

Jak pomocí klávesové zkratky, tak pomocí tlačítek v liště s pokročilými funkcemi, by měl být možný posun označeného úseku textu výše nebo níže po jednotlivých řádcích. Každý přeskočený řádek by se v takovém případě přesunul buďto nad nebo pod daný úsek textu.

Je vhodné, aby lišta s pokročilými funkcemi také obsahovala tlačítka určená pro převod všech označených písmen na velká nebo na malá písmena nebo případné prohození velikosti písmen.

3.4 Číslování řádků

Nutnou součástí textového editoru je taktéž sloupec s čísly řádků, umístěný v levé části okna aplikace. Barva čísel řádků by měla být stejná jako barva komentářů ve zrojovém kódů, aby jednotlivá čísla řádků co nejméně rušila při editaci textu. Sloupec s čísly řádků by měl být pro nejlepší čitelnost mírně odsazen od samotného textu a také by neměl obsahovat ohraničení. Čísla konkrétních řádků by měla změnit svou barvu na červenou v případě, že na daném řádku došlo k úpravě. Po uložení souboru dojde k následné změně barvy všech čísel řádků na barvu původní. Vzhledem k přítomnosti sloupce s čísly řádků, není potřeba jiným způsobem zobrazovat aktuální číslo řádku, nicméně například po najetí na sloupec s čísly řádků, by mělo být zobrazeno aktuální číslo sloupce. Zabalení bloku kódu je funkcionalita, která se týká taktéž sloupce s čísly řádků, nicméně její implementace je špatně realizovatelná za pomocí použitých nástrojů. Právě z tohoto důvodu bylo zabalování bloků kódu vyřazeno z návrhu editoru.

3.5 Zvýrazňování syntaxe

Stěžejní komponentou každého programátorského editoru je podpora zvýrazňování syntaxe pro širokou škálu programovacích, skriptovacích a značkovacích jazyků. Z časového hlediska je možné zanedbat implementaci zvýrazňování pro některé jazyky. Podporovány by měly být především objektově orientované jazyky, protože jak manipulace s projekty, tak analýza tříd, se týká právě těchto jazyků. Podpora objektově orientovaných jazyků má tedy nejvyšší prioritu. Nutností je taktéž zvýrazňování identifikátorů funkcí a třídních metod unikátní barvou, což výrazně přispívá k čitelnosti zdrojového kódu. Ve výchozí navigační liště by se mělo nacházet tlačítko umožňující skrytí všech komentářů ve zdrojovém kódu. Zajisté je potřeba implementovat automatické podtržení všech stejných identifikátorů ve zdrojovém kódu v případě, že se na některém z nich nachází textový kurzor. Z výkonnostního hlediska je vhodné zařadit krátký časový interval před aktualizací všech podtržených identifikátorů, protože i samotné podtržení vetšího množství slov, může vyžadovat znatelnou výpočetní režii.

Pro poskytnutí manuálního zvýrazňování textu uživateli, je vhodné přidat další navigační lištu, obsahující potřebné nástroje pro vložení uživatelského zvýraznění. Zajisté je potřeba zařadit možnost zvolení barvy a průhlednosti zvýraznění, například pomocí dialogu pro volbu barvy. Vlastní zvýraznění by mělo být aplikováno v rozsahu označeného textu s tím, že překrývání několika různých zvýraznění je povoleno. Každé aplikované zvýraznění se týká pouze pozadí daného textu a pro odstranění konkrétního zvýraznění stačí pouze umístit textový kurzor kdekoliv do rozsahu zvýraznění. Následným stiskem tlačítka pro odstranění dojde k odstranění daného zvýraznění. Možnost postupného vrácení změn v rámci manuálního zvýrazňování, by mělo být taktéž umožněno pomocí tlačítka v navigační liště pro zvýrazňování.

3.6 Projekty

Zabudování správy projektů do textového editoru vyžaduje použití samostatné lišty, určené pro samotnou manipulaci s projekty a se soubory, které tvoří daný projekt. Pro dosažení nezávislosti jednotlivých projektů na textovém editoru, je vhodné spojit každý projekt s projektovým souborem uloženým v některém z adresářů na disku. Daný projektový soubor by měl obsahovat všechny potřebné informace o struktuře projektu. Díky tomuto návrhu jsou jednotlivé projekty snadno přenositelné mezi různými počítači.

Textový editor by měl uživateli umožnit snadné vytvoření projektu ve zvoleném adresáři a následně nechat uživatele zařadit konkrétní seznam souborů do projektu. Textový editor by měl automaticky vyhodnotit, zda se v adresáři projektu již nachází uživatelský soubor Makefile. V případě, že se v adresáři projektu soubor Makefile nenachází, tak dojde k automatickému vytvoření souboru Makefile na základě struktury projektu. Za pomoci tlačítek určených pro překlad projektu, by mělo být možné rovnou sestavit projekt a spustit výslednou aplikaci. Uživateli je vhodné poskytnout možnost nastavit jak spouštěcí parametry aplikace, tak parametry překladače a parametry aplikace make. Ideálním způsobem zobrazení výstupu překladu, je využití samostatného okna textového terminálu. Důvodem je především přehledné a především barevné zvýraznění varování a chyb v průběhu překladu. Ke spuštění samotné aplikace dojde ihned po dokončení překladu ve stejném okně terminálu. Typ použité terminálové aplikace si samozřejmě zvolí sám uživatel. Textový editor nesmí v žádném případě uživatele nutit do použití projektu při práci. Uživatel si může kdykoliv aktivovat výchozí možnost, kterou je práce pouze s jediným souborem. Samozřejmostí je podpora překladu nebo přímé interpretace samostatného zdrojového souboru. [11]

Lišta určená pro manipulaci s projekty, by měla obsahovat tlačítko, jehož stiskem dojde k analýze celého projektu a následně bude automaticky sestaven a zobrazen strom všech tříd, které jsou přítomny v daných souborech projektu. Kromě tříd by měl výsledný diagram obsahovat i množinu všech globálních proměnných, funkcí, struktur a výčtových typů. Součástí tříd musí samozřejmě být i třídní atributy a metody. Všechny nalezené elementy by měly být umístěné v hierarchické stromové struktuře, kde součástí každého uzlu je kromě samotného identifikátoru i ikona reprezentující konkrétní typ daného elementu. Přesný typ uzlu bude samozřejmě zobrazen i po najetí kurzorem na daný uzel. U třídních atributů a metod bude mít daná ikona odlišnou barvu, aby šlo snadno rozpoznat viditelnost daného elementu z pohledu zděděné třídy. Po klepnutí na daný element dojde k přímému skoku na deklaraci daného elementu ve zdrojovém kódu. Za pomoci klepnutí pravým tlačítkem myši, dojde k přednostnímu skoku na definici elementu. Cílový zdrojový soubor bude automaticky otevřen v editoru, pokud se tak nestalo v minulosti. V případě, že nebude zvolen žádný konkrétní projekt, tak dojde k vytvoření stromu tříd z aktuálně otevřeného zdrojového souboru.

3.7 Rozvržení okna

Navržená aplikace musí vycházet vstříc uživatelům obrazovek s vysokým rozlišením tím, že jim poskytne snadné otevření a editace více zdrojových souborů najednou. Uvnitř navigační lišty pro práci se soubory je potřeba přidat možnost výběru rozvržení aplikace. Mezi podporovaná rozvržení okna by mělo spadat řádkové, sloupcové a také maticové rozvržení, kdy by celé okno bylo pokryto vícero textovými editory. Uživateli musí být při rozdělené obrazovce umožněno upravovat velikost jednotlivých textových editorů za pomoci posunu rozdělovače umístěného mezi jednotlivými editory. Každý textový editor by měl mít svůj

vlastní seznam jak otevřených textových souborů, tak seznam otevřených projektů, aby bylo dosaženo úplné nezávislosti editorů.

3.8 Podpora editoru Vim

Možnost editace zdrojového kódu ve stylu editoru Vim, by měla být uživateli poskytnuta, nicméně uživateli nesmí být žádným způsobem vnucena. Výhodným řešením je umístění tlačítka do lišty s pokročilými funkcemi, které by aktivovalo editaci textu ve stylu editoru Vim. Po aktivaci této možnosti by se aplikace měla tvářit podobně, jako skutečný editor Vim. Navigační lišta by měla být upravena do podoby, která odpovídá stavové liště přítomné ve Vimu. Uživateli by měla být poskytnuta co největší část funkcionality, kterou Vim umožňuje, nicméně implementace veškeré funkcionality je velmi náročným úkolem. Zajisté je potřeba podporovat všechny režimy editoru Vim, mezi které patří vkládací, vizuální, nahrazovací a příkazový režim. Z hlediska pohybu skrze text je nutné podporovat jak pohyb po jednom znaku ve všech směrech, tak pohyb skrze celá slova, řádky, věty, odstavce a nesmí chybět také skok na nejbližší výskyt zvoleného znaku. Samozřejmostí je také podpora příkazu pro substituci a vyhledávání vzorů v textu. Každý z příkazů pohybu může uživatel kdykoliv doplnit o příkaz k mazání obsahu nebo převodu velikosti písmen. Nutností je také podpora násobení počtu provedených za pomocí uživatelem vložené číslovky. Také je potřeba, aby veškeré klávesové zkratky byly shodné se zkratkami z editoru Vim a všechny byly přehledně popsány v nápovědě k aplikaci.

3.9 Správa verzí

Zahrnutí podpory manipulace s Git repozitářem do výsledného editoru, bylo vynecháno vzhledem k obtížnosti implementace této funkcionality. V rámci implementace by totiž bylo potřeba zajistit detekci Git repozitáře v adresáři, kde se nachází daný zdrojový soubor nebo celý projekt. Dále by bylo nutné rozpoznat aktivní větev repozitáře a poskytnout uživateli provedení většiny operací, které Git uživateli umožňuje s tím, že uživatel by si jen zvolil konkrétní operaci a nemusel psát přesný tvar dané operace tak, jak ho vyžaduje Git. Mezi tyto operace patří aktualizace souborů jak v lokálním, tak ve vzdáleném Git repozitáři. Vynechání implementace této funkcionality nečiní problém vzhledem k tomu, že se jedná o funkcionalitu, která má větší význam především při práci s rozsáhlejšími projekty ve vývojových prostředích.

3.10 Více textových kurzorů

Editace zdrojového souboru pomocí více textových kurzorů najednou, je funkcionalita, kterou nabízí pouze proprietární textový editor Sublime Text. Implementace této funkcionality je velmi problematická za pomoci běžných vývojových nástrojů, jejichž cílem je poskytnout pohodlnou implementaci programátorského editoru. Jádro problému spočívá ve skutečnosti, že dostupné vývojové nástroje poskytují uživateli šablonu textového editoru, do níž nelze zakomponovat podporu více textových kurzorů. Právě z důvodu problematické implementace byla podpora více textových kurzorů vyřazena z návrhu textového editoru.

3.11 Jazyky

Navržený programátorský editor by měl samozřejmě podporovat dostatečný počet jazyků, aby vyhověl potřebám většiny uživatelů. Aby navržená aplikace vyhověla i těm uživatelům, kteří si chtějí nadefinovat zvýrazňování syntaxe pro vlastní jazyk, by bylo vhodné integrovat komponentu, která těmto lidem vyhoví. Tato komponenta by se dala nazvat editorem zvýrazňování pro vlastní jazyk a její spuštění je ideální řešit pomocí tlačítka v liště s pokročilými funkcemi. V rámci editoru vlastního zvýrazňování, je vhodné vytvořit další lištu určenou pro správu uživatelských zvýrazňování. Počet uživatelských zvýrazňování by neměl být omezen a vytvořená zvýrazňování je nutné automaticky zařadit do seznamu vestavěných jazyků.

V případě, že si uživatel zvolí možnost vytvoření nového zvýrazňování, tak se obsah okna aplikace nahradí přehledným editorem uživatelských zvýrazňování. Obsah okna bude nabízet dostatečné množství prvků jazyka, kdy pro každý z nich bude mít uživatel možnost vložení libovolného regulárního výrazu. Kromě regulárního výrazu bude mít uživatel možnost vložit seznam klíčových slov nebo šablonu komentáře pro daný jazyk. Pro usnadnění práce bude pro každý prvek jazyka dostupných hned několik šablon, které jsou společné pro většinu známých jazyků. Patří zde například šablona pro zvýrazňování hlavičky funkce, klíčového slova, desetinného čísla a samozřejmě řetězcového literálu. Díky zabudovaným šablonám je uživateli umožněno navrhnout vlastní zvýrazňování rychle a pohodlně. Je vhodné zajistit možnost vložení zvýrazňování víceřádkových komentářů. V rámci každého prvku jazyka bude přítomna možnost výběru barvy zvýrazňování a to pro každý z dostupných motivů. Pro lepší přehlednost bude ve vrchní části okna zobrazena barva pozadí všech motivů včetně možnosti nastavení filtru pouze pro jeden konkrétní motiv, aby měl uživatel jistotu, že omylem nepřiřadil zvolenou barvu jinému motivu. V rámci editace vlastního zvýrazňování bude samozřejmě nutnost si zvýrazňování pojmenovat názvem jazyka, kterého se týká.

3.12 Automatické ukládání

Navržený editor by se měl pokaždé spustit ve stejném stavu, v jakém byl při ukončení, samozřejmě s výjimkou prvního spuštění. Týká se to jak samotného obsahu editoru včetně pozice kurzoru a označení textu, tak otevřených projektů. Samozřejmostí je zachování nastavení týkajících se každého otevřeného souboru. Stav aplikace se týká také všech jednotlivých textových editorů v případě, že bylo okno aplikace v minulosti rozděleno. Veškerá manuální zvýrazňování musejí být taktéž ve stejném stavu, jako při ukončení aplikace. Pro uložení všech nutných informací bude využito vícero konfiguračních souborů, kdy bude přítomen konfiguračních soubor pro textový editor, projekty a uživatelská zvýrazňování.

Kapitola 4

Implementace

V rámci implementace navrženého textové editoru je potřeba zajistit, aby výsledná aplikace bezproblémově běžela na operačním systému Linux. Znatelnou výhodou by také bylo zajištění podpory běhu výsledné aplikace i na dalších operačních systémech. V tomto ohledu je ideální využít multiplatformní framework Qt, který poskytuje velké množství dostupných nástrojů a knihoven pro implementaci navrženého programátorského editoru. Framework navíc umožňuje běh výsledné aplikace i na dalších operačních systémech. Framework Qt umožňuje pohodlnou implementaci aplikací za pomoci programovacího jazyka C++ s tím, že samotný framework je v C++ naimplementován. V důsledku použití jazyka C++ by měla být výsledná aplikace na velmi dobré úrovni z hlediska výkonu a spotřeby operační paměti. Na základě zmíněných výhod byl pro implementaci zvolen právě framework Qt. Rychlost vývoje byla pozitivně podpořena taktéž využitím aplikace Qt Creator, která umožňuje rychlý a komfortní přístup k frameworku Qt a poskytuje velké množství nástrojů pro usnadnění implementace výsledné aplikace. Qt Creator taktéž poskytuje nástroje pro intuitivní návrh aplikací za pomocí grafických prvků. Pro implementaci svobodné aplikace je navíc možné používat celý Qt framework i potřebné nástroje úplně zdarma. Kompletní návod na sestavení aplikace ze zdrojových textů lze nalézt v příloze B. Veškeré informace o třídách použitých v rámci implementace, byly získány buďto z oficiální dokumentace k dané třídě, nebo z knihy [3].

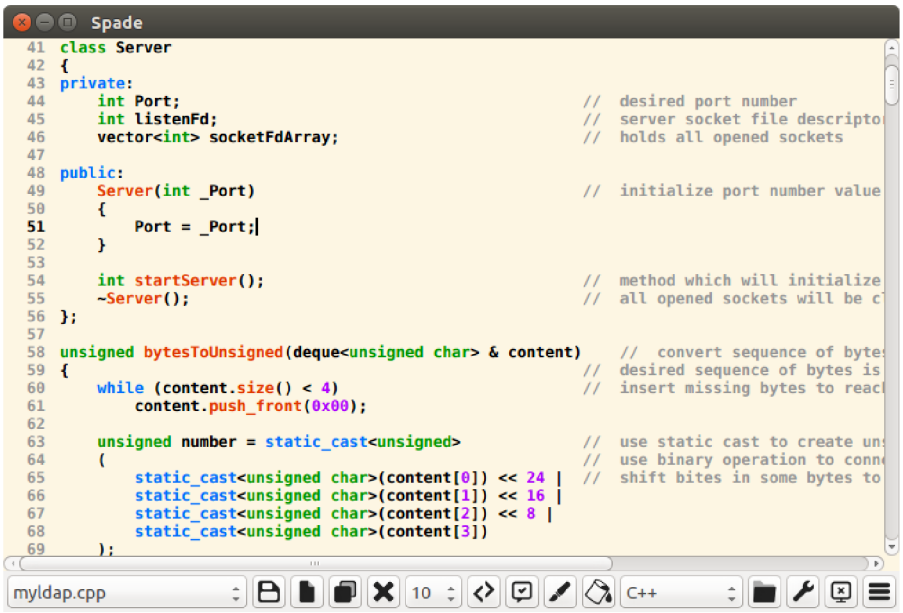
4.1 Uživatelské rozhraní

Pro vytvoření samotného okna aplikace bylo využito třídy `QDialog`, které implementuje nutné operace týkající se manipulace s oknem, což zahrnuje změnu velikosti okna, změnu pozice okna, reakci na podněty z okolí aplikace a samozřejmě je také rodičem veškerých grafických prvků a jiných komponent, které jsou součástí výsledné aplikace. Třída `QDialog` byla v rámci implementace zděděna a byl vytvořen její potomek zapouzdřující komponenty tvořící obsah okna. Qt framework využívá v rámci vzhledu grafických prvků motiv, který odpovídá grafickému prostředí daného operačního systému. Celkový vzhled výsledné aplikace je proto podobný výchozím aplikacím pro daný operační systém.

Pro manipulaci s pozicemi a zarovnáním prvků uvnitř okna aplikace, bylo využito tříd `QHBoxLayout`, `QVBoxLayout` a `QGridLayout`. Tyto třídy zajišťují korektní zarovnání obsahu i při změně velikosti okna. Rozdíl v těchto třídách spočívá v tom, zda uživatel potřebuje zarovnat prvky horizontálně, vertikálně nebo do matice. Tyto třídy byly taktéž využity při implementaci navigačních lišt, kdy se použilo horizontální zarovnání množiny prvků dané

lišty v kombinaci s případným skrýváním a odkrýváním prvků jiných lišt při přepnutí na jinou lištu. Pevná výška lišty zajistila stálý tvar a konzistenci prvků v navigační liště. Pro rozeznání aktivní lišty byla zavedena množina příznaků, kdy vždy jen jeden z nich měl pravdivou hodnotu, protože v daný moment je vždy viditelná pouze jediná lišta.

Pro vytváření tlačítek v navigačních lištách se využilo třídy `QPushButton` s tím, že použitá tlačítka v lištách vždy obsahují ikonu typu `QIcon` a mají čtvercový tvar. Velikost každé ikony byla specifikována za pomoci třídy `QSize`, která zajišťuje zobrazení zvoleného prvku ve zvolené velikosti s tím, že jeho rozměry neovlivní ani budoucí pozicování. [15] Volba aktivního zdrojového souboru je implementována za pomoci třídy `QComboBox` implementující výsuvný seznam otevřených souborů. Všechny grafické prvky v okně aplikace využívají atribut `toolTip` pro zobrazení informačního textu v bublině v případě najetí kurzorem myši na daný prvek.



Obrázek 4.1: Uživatelské rozhraní výsledného editoru

4.2 Editace textu

V rámci implementace komponenty pro samotnou editaci textu se využilo možností třídy `QPlainTextEdit`, která již ve své základní podobě implementuje relativně pokročilý textový editor, který více než postačuje pro implementaci poznámkového bloku. Tato třída je taktéž ideálním stavebním kamenem pro implementaci pokročilého programátorského editoru, čehož bylo samozřejmě využito. Instance této třídy má ve své výchozí podobě implementováno veškerou manipulaci s textovým kurzorem a implementována je taktéž logika, která umožňuje návrat provedených změn. Podporováno je samozřejmě i kopírování a vkládání textu. Instance třídy `QPlainTextEdit` ukládá obsažený text ve formátu definovaném třídou `QString`, která zapouzdřuje podporu velkého množství kódování řetězců s tím, že v rámci vloženého textu provádí převod do kódování UTF-8[1], ve kterém jsou výsledné textové sou-

[1] Univerzální, multiplatformní a široce podporovaná znaková sada

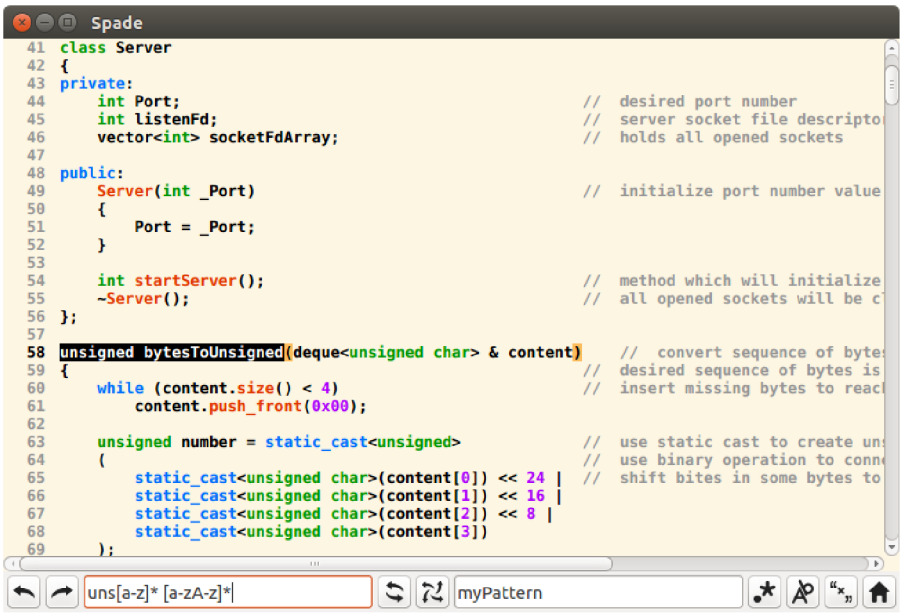
bory také ukládány. Výchozím písmem textového editoru je neprociální písmo, které je implicitní pro grafické prostředí daného operačního systému.

Z důvodu implementace automatického odsazování a doplňování závorek, bylo nutné předefinovat reakci na stisk některých kláves. Například místo klasického přechodu na nový řádek, byla vypočtena míra odsazení na aktuálním řádku s tím, že přechod na nový řádek již zahrnoval shodné odsazení. Automatické doplňování závorek a obecných úseků kódů vyžadoval rozpoznání, který prvek daného jazyka je právě editován. Například v případě přechodu z hlavičky cyklu na nový řádek bylo automaticky doplněno tělo cyklu. V případě volby šablony konkrétního prvku z lišty se šablonami prvků, dochází pouze k vložení obecného textu do zdrojového kódu. Implementace posunu označeného textu po řádcích směrem vzhůru nebo dolů, zahrnovalo označení řádku pod nebo nad zvoleným textem přímo ve zdrojovém kódu a jeho následné přesunutí na cílové místo. Převod velikosti písmen byl proveden za pomoci třídy `QString`, která má implementovány metody na převod velikosti písmen a případné převrácení velikosti písmen. Označený text byl následně přepsán textem s již změněnou velikostí písmen.

Obrázek 4.2: Výsledná lišta se šablonami prvků

4.3 Vyhledávání

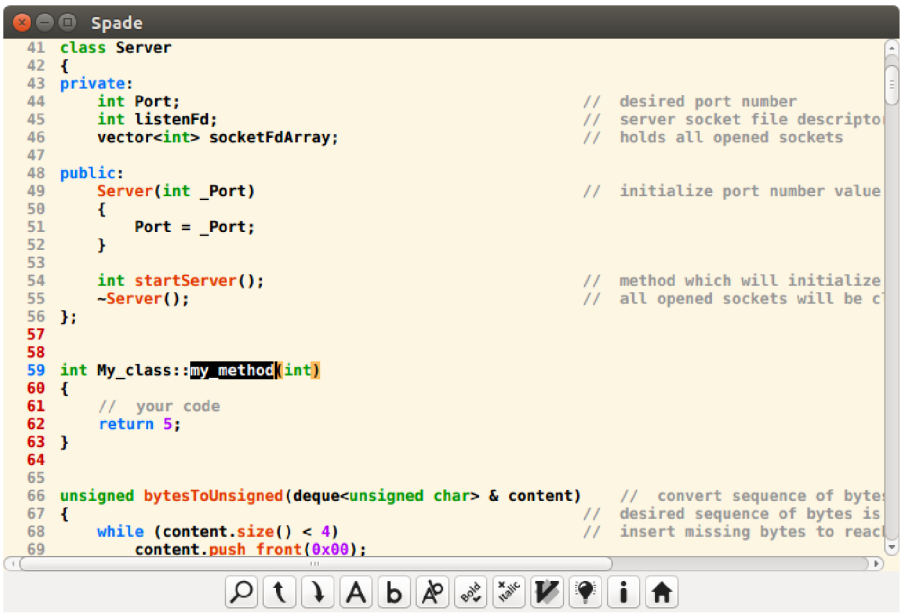
Implementace vyhledávání textových vzorů se provedla za pomoci tříd `QRegularExpression` a `QRegularExpressionMatch`. [14] Jak vyhledávání textových vzorů, tak vyhledávání pomocí regulárního výrazu, bylo implementováno stejným způsobem s tím, že v prvním případě byla využita metoda escape třídy `QRegularExpression`, která zrušila význam všech speciálních znaků v textu. Výsledná vyhledaná pozice byla odvozena od pozice textového kurzoru, kdy za vyhledaný vzor byl považován vzor umístěný nejblíže kurzoru. Vyhledaný vzor byl následně zvýrazněn za pomocí jeho označení, což samozřejmě změnilo pozici textového kurzoru, nicméně tímto způsobem byl zajištěn pohodlný sekvenční průchod vyhledanými vzory. Stejným přístupem jako v případě vyhledávání, bylo řešeno zvýraznění všech stejných slov v textu, kdy bylo vyhledáno slovo, na němž se právě nachází textový kurzor.



Obrázek 4.3: Ukázka vyhledávání pomocí regulárního výrazu

4.4 Číslování řádků

Pro vytvoření bloku s číslováním řádků, bylo potřeba vytvořit úplně novou třídu a nadefinovat chování prvků této třídy. Tato třída byla pojmenována jako `LineNumberArea` a byla potomkem třídy `QWidget`. Rodičovská třída poskytla nutnou funkcionalitu ke správnému zobrazení prvku a jeho korektní zarovnání v okně aplikace. Konstruktor této třídy obsahuje rodičovský objekt použitého textového editoru ve svém jediném parametru, protože blok s číslováním je komponentou textového editoru. Pro správné nastavení velikosti řádku s číslováním, byla nutná implementace metody `sizeHint`, jejímž účelem bylo získat hodnotu doporučených rozměrů bloku s číslováním, protože šířka bloku se mění s počtem řádků zdrojového kódu. Šířka bloku s číslováním je také ovlivněna velikostí a stylem písma, a proto je velikost vypočtena na základě těchto informací. Dále bylo zapotřebí implementovat metodu `paintEvent`, jejímž účelem je reakce na signál, který je zaslán textovým editorem v případě, že došlo ke změně vyžadující opětovné vykreslení bloku s číslováním. Uvnitř této metody je implemetováno samotné vykreslení čísel řádků s tím, že barva číslic je ovlivněna jak aktivním barevným motivem, tak skutečností, zda na daném řádku došlo v minulosti ke změně, což způsobuje zčervenání daného čísla řádku.



Obrázek 4.4: Reakce editoru na změnu textu

4.5 Zvýrazňování syntaxe

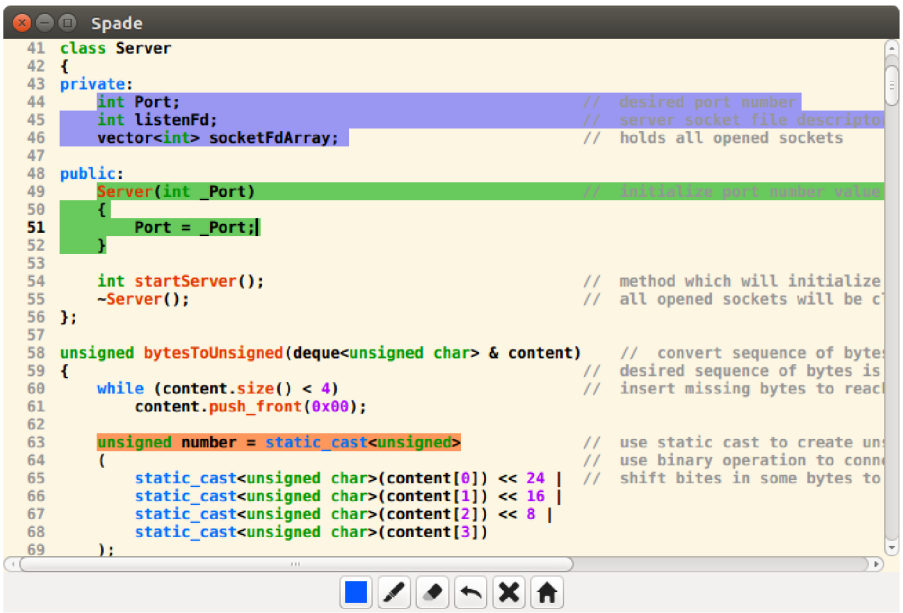
Pro implementaci programátorského editoru je nutné zvýrazňování syntaxe některého z podporovaných jazyků. Qt framework pro tyto účely nabízí třídu `QSyntaxHighlighter`, která umožňuje nadefinovat zvýrazňování syntaxe v objektu `QPlainTextEdit`, za pomocí množiny regulárních výrazů. [17] V rámci manipulace s regulárními výrazy byla využita funkcionalita poskytována třídou `QRegularExpression`, jejíž výhodou je především velká rychlost při hledání shody zvoleného regulárního výrazu s textovým řetězcem. Pro aktivaci zvýrazňování syntaxe zvoleného jazyka, bylo zapotřebí vytvořit instanci třídy, která dědila funkcionalitu z výše zmíněné třídy `QSyntaxHighlighter`. Součástí implementace ve zděděné třídě je načtení množiny regulárních výrazů zvoleného jazyka do množiny zvýrazňovacích pravidel, kde je kromě samotného regulárního výrazu definován předpis pro barevné stylování daného prvku jazyka pro podporované motivy. Při každé změně zdrojového kódu na některém řádku, je automaticky zavolána obslužná metoda, která zajistí aplikaci zvoleného zvýrazňování. Definice této metody je samozřejmě přítomna ve výše zmíněné zděděné třídě. Důležitým aspektem aplikace zvýrazňování, je pořadí aplikace jednotlivých vyhovujících pravidel. Pravidla, která byla definována dříve z hlediska naplnění množiny pravidel, mají v rámci aplikace zvýrazňování vyšší prioritu.

Výjimku v rámci zvýrazňování syntaxe, tvoří aplikace stylovacích pravidel na víceřádkové komentáře, protože výchozí styl aplikace je aplikovatelný pouze na změny týkající se jediného řádku textu. V rámci zvýraznění víceřádkového komentáře byla nutná implementace průchodu stavy jednotlivých řádků v případě, že komentář začínal na právě změněném řádku, což způsobí nastavení aktuálního stavu na daný řádek. Následně dojde k vyhledání ukončujících znaků víceřádkového komentáře. Zvýraznění bude aplikováno pouze v případě nalezení konce komentáře, jinak dojde k aktualizaci nastaveného stavu, který byl ovlivněn vyhledáváním konce komentáře. Nastavený stav bude dále nastavený na daný řádek i pro případ budoucích změn na jiných řádcích v textu. Skrytí komentářů ve zdrojovém kódu je možné implementovat různými způsoby, nicméně skrytí bylo dosaženo pouhým nastavením

barvy zvýrazňování komentářů na barvu pozadí editoru. Tento přístup má samozřejmě nevýhodu v tom, že v případě označení textu dojde v místě označení k opětovnému zvýraznění komentáře, nicméně tento přístup nevyžaduje žádnou analýzu zdrojového kódu navíc, což je pozitivní zpráva z hlediska výkonu aplikace.

Pro zjištění, kterých řádků se týkala právě provedaná změna textu, byla využita metoda, která byla automaticky zavolána v případě každé změny textu. Pro výpočet rozsahu změn bylo nutné uložit předcházející pozici textového kurzoru. Uvnitř metody byla následně porovnána nová pozice textového kurzoru s tou předcházejcí. Zjištění, kterých řádků se změna textu týkala, bylo dosaženo za pomocí průchodu zdrojového kódu po samostatných řádcích. V případě, že se text umístěný na pocici mezi aktuálním a předcházejícím kurzorem, nacházel na úrovni daného řádku, byl daný řádek označen jako změněný. Veškerá historie změněných řádků byla ukládána, z důvodu použití starší verze historie změn v případě vrácení změn v editoru. Nové změny na řádcích tedy byly detekovány pouze v případě úpravy historie změn. V případě vrácení změn bylo využito stavu daného minulého kroku z historie změn řádků.

V rámci manuálního zvýrazňování zdrojového kódu, bylo nutné umožnit uživateli volbu barev pro jednotlivá zvýraznění. Komfortní volbu barvy umožnila třída `QColorDialog`, která implementuje samostatný dialog, v němž má uživatel možnost zvolení přesného odstínu barvy s tím, že je možná také volba průhlednosti dané barvy. Aplikace manuálního zvýrazňování byla imlementována v rámci atributu `ExtraSelection` třídy `QTextEdit`, který uchovává nadstandardní zvýraznění v textu. Následné změny zvýrazňování vždy vyžadovaly úpravy zmíněného atributu. Samotné vložení zvýraznění do textu umožnila třída `QTextCursor`, jejímž účelem je právě manipulace s textovým kurzorem přímo ze zdrojového kódu aplikace.



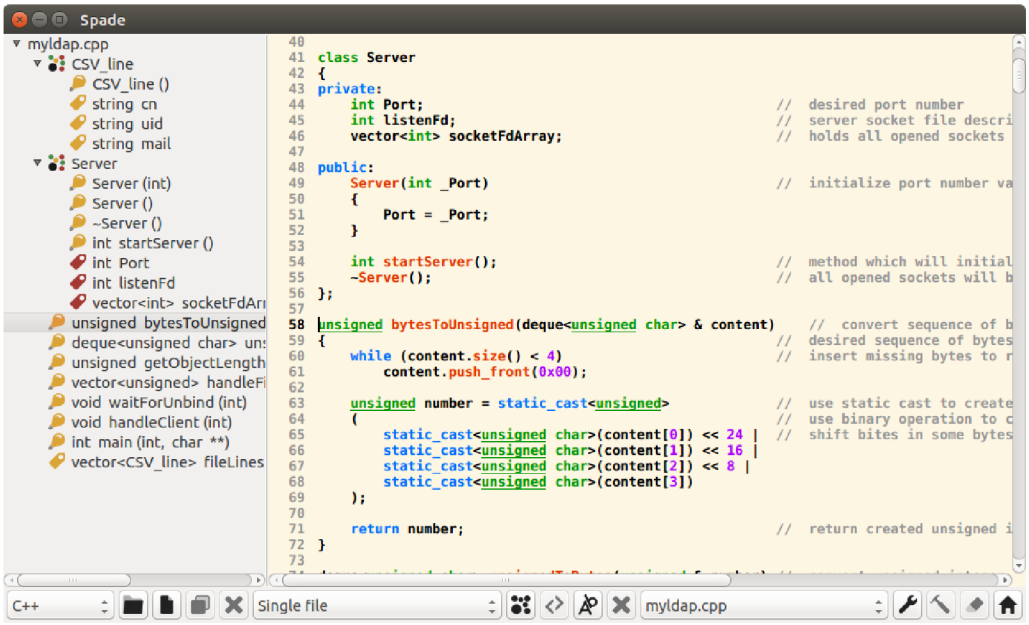
Obrázek 4.5: Ukázka manuálního zvýrazňování zdrojového kódu

4.6 Projekty

Pro manipulaci s projektovými soubory byla využita třída `QFile`, která umožnila jak vytváření projektových souborů, tak zápis a čtení údajů v rámci těchto souborů. Pro snadný zápis textových dat do projektových souborů, byla využita třída `QTextStream`, která je určena pro práci s textovým proudem dat. [18] Volba cílového adresáře byla umožněna pomocí třídy `QFileDialog`, která implementuje samostatné dialogové okno, v němž si uživatel může pohodlně otevřít existující soubory nebo vytvořit nový soubor či adresář. Pro ukládání projektů v rámci aplikace, byla vytvořena třída `Project_details`, která obsahuje všechny potřebné informace načtené z projektového souboru. Každého projektu se samozřejmě týkalo i vytváření souboru Makefile, které vyžadovalo pečlivé oddělení hlavičkových a zdrojových souborů. Následně byl pro každý zdrojový soubor vytvořen objekt, který zahrnoval překlad daného souboru. Názvy jednotlivých objektů měly obecné názvy, které se odlišovaly jenom číslem objektu na konci názvu. V rámci každého modulu byl vložen příkaz pro překlad včetně zahrnutí parametrů překladače. Veškeré objekty byly v rámci vytvoření spustitelného souboru slinkovány opět s použitím překladače. V případě spuštění překladu projektu je nutné, aby byly automaticky uloženy všechny neuložené změny v rámci souborů, které jsou zařazeny do projektu. Následně dochází k vygenerování souboru Makefile nutného pro překlad, nicméně pouze za předpokladu, že se soubor Makefile v adresáři projektu nenachází, což umožňuje použití uživatelských Makefile souborů. Jak sestavení, tak spuštění výsledné aplikace, je provedeno v samostatném procesu, jímž je aplikace terminálu. Pro vytvoření nového procesu byla využita třída `QProcess` umožňující pohodlné vytváření nových procesů přímo ze zdrojového kódu aplikace. Pro zajištění automatického zahájení překladu a spuštění aplikace v novém procesu, bylo využito aplikace bash, která se postarala o činnost provedenou po vytvoření nového procesu. V případě, že byl již projekt v minulosti sestaven, došlo k okamžitému spuštění aplikace bez nutnosti sestavování, o což se již ze svého principu fungování postaral soubor Makefile v kombinaci s aplikací make.

Zobrazení stromu tříd bylo implementováno za pomoci třídy `QTreeWidget`, která poskytuje propracovaný přístup k vytvoření stromové struktury na základě preferencí uživatele. [19] Získání dat pro naplnění uzlů stromu daty, vyžadovalo kombinaci syntaktické analýzy zdrojového kódu a vyhledávání v textu za pomoci regulárních výrazů. Důvodem kombinace těchto dvou odlišných přístupů, bylo dosažení co nejvyšší rychlosti z hlediska získání obsahu pro výsledný strom tříd. Stěžejní myšlenka postupu získání potřebných dat byla taková, že obsah zdrojového kódu byl rozdělen na pomyslné úseky textu, které reprezentovaly buďto bloky zdrojového kódu pro jazyk C++ nebo úseky s odlišnou hloubkou odsazení, což odpovídá jazykům, které nepoužívají blokové závorky k vyznačení úseků textu, mezi které patří například jazyk Python. [13] Právě v rámci rozdělení kódu na samostatné bloky se využilo syntaktické analýzy, protože dvojice blokových závorek můžou tvořit hierarchii a zanoření, což nečiní syntaktické analýze potíže, nicméně regulární výrazy nejsou vhodné na ověření shody textu, který obsahuje libovolné zanoření. Po získání pozic všech samostaných úseků textu, byla prověřena hlavička, která náleží danému úseku textu. Tímto způsobem bylo dosaženo získání všech definic v textu, což zahrnovalo funkce, metody, struktury a samozřejmě také třídy a výčtové typy. Jednotlivé nalezené úseky textu mohly být umístěny uvnitř bloků jiných, čímž byly rozeznány například metody definované uvnitř těla třídy. Následným krokem bylo vyhledání deklarací prvků nalezených v rámci úseků textu, což vyžadovalo porovnání identifikátoru daného prvku a případných parametrů u funkcí a metod. Definice metod nacházejících se mimo tělo třídy, byly nalezeny opět za pomoci porovnávání identifikátoru metody, parametrů metody a specifikace názvu

třídy. Veškeré vyhledávání bylo prováděno za pomocí regulárních výrazů, s cílem rychlejšího získání výsledku. Nalezené prvky v rámci stromu tříd, byly uchovávány za použití tříd `Class_tree_parent` a `Class_tree_child`, které uchovávaly jak přesnou lokaci prvku, tak například názvy rodičovských tříd, v případě, že prvek byl třídou. Uzly skutečného stromu tříd pak byly naplněny získanými hodnotami s tím, že pro dosažení co nejlepší čitelnosti stromu tříd, obsahoval každý uzel stromu ikonu korespondující s typem uzlu. Jednotlivé ikony se lišily nejen svým tvarem, ale také svou barvou, aby bylo možné okamžitě rozlišit například soukromé a veřejné atributy tříd.



Obrázek 4.6: Zobrazení stromu tříd

4.7 Rozvržení okna

Rozdělení okna aplikace na více samostatných textových editorů, bylo implementováno tím způsobem, že byla vytvořena nadřazená třída `Window`, která v sobě zahrnuje více objektů textového editoru s tím, že při spuštění aplikace je viditelný pouze jediný textový editor, který zabírá společnou plochu. Každý ze zahrnutých textových editorů je reprezentován samostatnou instancí, což poskytuje vzájemnou nezávislost jednotlivých editorů. Po změně rozvržení okna aplikace, byly aktivovány odpovídající textového editory a plocha okna byla upravena do požadovaného tvaru. Úprava poměru vzájemných velikostí textových editorů, byla umožněna díky třídě `QSplitter`, která implementuje posuvník v podobě horizontální nebo vertikální čáry mezi editory. [16] Nezávislost textových editorů byla pro maximální pohodlí uživatele doplněna o čtyři nová tlačítka do lišty se zvýrazňováním pro vlastní jazyky, která umožnila vložení zvoleného zvýrazňování do některého z vedlejších textových editorů, čímž uživateli ušetřila práci tím, že nebyl nucen znovu navrhnout stejné zvýraznění v jiném editoru.

Obrázek 4.7: Ukázka rozdělení okna aplikace na více samostatných textových editorů

4.8 Podpora editoru Vim

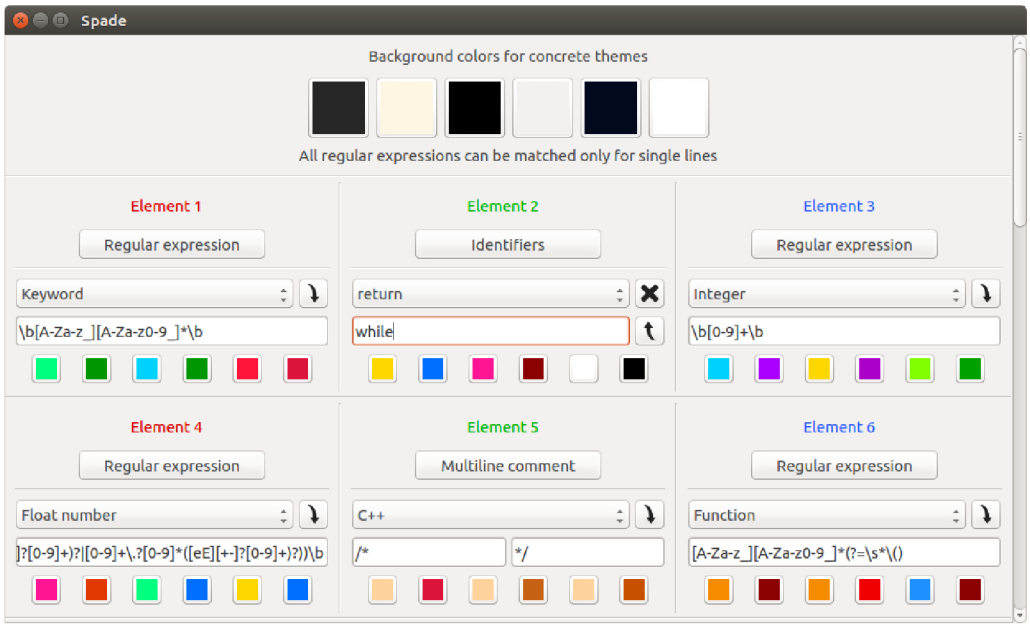
V případě, že uživatel aktivoval možnost editace ve stylu editoru Vim, bylo nutné upravit chování reakce na stisk většiny kláves, aby správně fungovaly podporované příkazy Vimu. Režim editoru byl rozlišován pomocí množiny příznaků, kdy kladné hodnoty nabýval pouze příznak odpovídající aktivnímu režimu. V rámci stisku kláves reprezentujících číslice, byla uchovávána výsledná hodnota, která byla vytvořena sekvencí uživatelem zvolených číslic. Uchovaná číselná hodnota se použije jako násobič zvoleného příkazu, kdy daná operace bude provedena vícekrát. Kromě číselných hodnot byly uchovávány také příkazy modifikující text, mezi které patří například smazání textu nebo změna velikosti písmen. Tyto příkazy byly následně zohledněny při použití příkazu na posun textového kurzoru, čímž mohlo být dosaženo například odstranění pěti následujících slov. Dále bylo potřeba využít

syntaktické analýzy v rámci použití příkazu pro substituci textu, kdy bylo nutné správně určit parametry substituce. Mírnou odlišností v porovnání s editorem Vim, je chování příkazu pro návrat změn textu. Editor Vim využívá návrat změn pouze v rámci jediného řádku, čímž se odlišuje od použité implementace, kdy se návrat změn týká celého textu, aby byla zajištěna kompatibilita s číslováním řádků.

Obrázek 4.8: Editace ve stylu editoru Vim

4.9 Jazyky

V rámci poskytnutí návrhu vlastního zvýrazňování uživateli, bylo nutné vytvořit třídu `Element` reprezentující jeden prvek uživatelského jazyka. Tato třída definovala potřebné atributy a operace v rámci volby barvy zvýrazňování, typu prvku, vložení výchozí šablony a nastavení víceřádkového komentáře. Uživatelské rozhraní všech prvků bylo vytvořeno za pomoci grafických nástrojů v aplikaci QtCreator. Třída `Element` je potomkem třídy `QWidget` a její instanci lze samozřejmě používat jako samostatnou komponentu, čehož se také využilo. Následně byla vytvořena třída `Elements`, která přistupuje ke všem vytvořeným prvkům jazyka jako k jednomu prvku a definuje společné chování pro zahrnuté použité prvky. Tato třída má v sobě zahrnuto i uložení nového zvýrazňování či zrušení všech provedených změn. Objekt třídy `Elements` byl použit v případě zvolení tvorby nového zvýrazňování, kdy v době jeho viditelnosti zabral celé okno aplikace z důvodu lepší čitelnosti.



Obrázek 4.9: Návrh vlastního zvýrazňování syntaxe

4.10 Automatické ukládání

Před každým ukončením aplikace je vždy zavolána metoda, která zajistí uložení stavu aplikace do odpovídajících konfiguračních souborů. Pro manipulaci s konfiguračními soubory, byly jako v případě projektových souborů využity třídy `QFile` a `QTextStream`. V rámci otevřených zdrojových souborů v textovém editoru, byl do konfiguračního souboru uložen jak obsah textového editoru, tak veškerá nastavení, která ovlivnila daný otevřený dokument. Ukládány byly jak pozice posuvníků editoru, tak pozice textového kurzoru a také byla uložena všechna uživatelská zvýraznění včetně barev i přesných pozic. Dále bylo nutné uložit informace týkající se zvýraznění vlastních jazyků, což zahrnovalo uložení jak regulárních výrazů všech prvků jazyka, tak také jejich barvy zvýraznění pro všechny barevné motivy.

Kapitola 5

Testování

Po dokončení implementace textového editoru bylo potřeba provést uživatelské testování. Cílem testování je prověření jak uživatelského rozhraní implementovaného textového editoru, tak funkcionality, kterou editor poskytuje. V rámci testování bylo nutné získat zpětnou vazbu od skupiny testerů. Zpětná vazba od testerů měla poskytnout přehled o tom, které části textového editoru jsou v pořádku, a u kterých by bylo vhodné některé věci upravit. Pro uživatelské testování bylo vybráno právě devět testerů, kterým byla v rámci procesu testování poskytnuta výsledná aplikace. Následně měli testeři příležitost vyzkoušet si veškeré aspekty aplikace. Zvolený počet devíti testerů je vhodným kompromisem mezi časovou náročností testování a získáním dostatečného počtu názorů na aplikaci. Obecná charakteristika testera je v tomto případě taková, že se jedná o vývojáře softwaru, který má dostatečné množství zkušeností na to, aby měl jeho názor dostatečnou váhu z hlediska získání zpětné vazby.

Skupinu zvolených devíti testerů lze rozdělit do tří odlišných částí. První částí testerů je tříčlenná skupina profesionálních vývojářů, které lze dále rozdělit na dva webové vývojáře a jednoho vývojáře aplikací. Tito lidé mají spoustu zkušeností z hlediska praktického vývoje softwaru s tím, že od textového editoru budou očekávat jak jeho spolehlivost, tak jeho vliv na produktivitu programátora, a v neposlední řadě je také bude zajímat i pořizovací cena aplikace. Druhou částí testerů je čtyřčlenná skupina vysokoškolských studentů oboru informačních technologií, které bude zajímat nejenom bohatá funkcionalita editoru, ale také moderní vzhled aplikace. Poslední částí testerů jsou vývojáři rekreační, kteří programují ve svém volném čase a programování je baví. Pro tuto skupinu lidí je textový editor aplikací, kterou rádi používají a která jim dobře slouží, a proto je vhodné získat zpětnou vazbu i od této skupiny lidí.

5.1 Návrh testování

Testovací procedura byla rozdělena do tří navazujících částí, které se lišily hlavně množstvím informací, které byly ohledně aplikace testerovi sděleny. První část testování spočívala ve spuštění aplikace a následném testování aplikace testerem, aniž by mu bylo cokoliv řečeno ohledně samotného rozhraní a ovládání aplikace. Tester se musel sám vypořádat s rozhraním aplikace a bylo pouze na něm, aby jejímu ovládání porozuměl a naučil se ho efektivně používat. Druhá část testování byla v zásadě shodná s fází předcházející. Jediným rozdílem bylo podrobné popsání uživatelského rozhraní testerovi, a to včetně vysvětlení principu ovládání aplikace. Poté měl tester prostor pro opětovné vyzkoušení rozhraní, nicméně už vě-

děl, jak rozhraní aplikace opravdu funguje. Poslední částí testování byla diskuze, jejíž náplní bylo pokládání otázek testerovi. Otázky se týkaly důležitých aspektů uživatelského rozhraní aplikace. Jednalo se zejména o to, zda je rozhraní přehledné, dostatečně pokročilé, vhodně vizuálně zpracované, intuitivní, snadno pochopitelné, rychle naučitelné a především snadno zapamatovatelné. Uživatelské rozhraní bylo vytvořeno právě proto, aby pomocí něj mohl uživatel dosáhnout toho, k čemu je aplikace určena a dokázal využít hlavní funkcionalitu aplikace. Proto bylo v neposlední řadě třeba od testera zjistit, zda podle něj uživatelské rozhraní plní své cíle. Na závěr byla s testerem prodiskutována samotná funkcionalita aplikace. Bylo důležité zjistit, které funkce editoru tester postrádal v porovnání s jinými textovými editory, a které schopnosti editoru ho příjemně překvapily a zaujaly ho. V rámci třetí fáze testování byl vytvořen testovací protokol obsahující otázky, které byly testerovi postupně kladeny s cílem získání potřebných informací. Otázek bylo celkem 24 a jejich sekvence postupně zahrnula téměř všechny stěžejní aspekty uživatelského rozhraní aplikace doplněné o funkcionalitu daného prvku. Otázky téměř korespondovaly s body popsanými v návrhu textového editoru, a odpovídalo i jejich pořadí. Poslední otázka dávala testerovi prostor k jeho zbylým připomínkám a doporučením, která by podle něj mohla zdokonalit aplikaci. Seznam všech otázek obsažených v testovacím protokolu lze nalézt v příloze C.

5.2 Průběh testování

Součástí první fáze testování tedy bylo vyzkoušení aplikace daným testerem. Jediná informace jemu sdělena byla taková, že se jedná o programátorský editor. Reakce všech testerů na uživatelské rozhraní editoru byly podobné. V zásadě každý z testerů dokázal editovat zdrojový kód včetně změny velikosti písma a aktivního jazyka. Testeři bez problému dokázali uložit soubor již v rámci první fáze testování. Následným zmatením testerů byla hierarchická navigační lišta, kdy po přechodu na její nižší úroveň úplně nepochopili, co daným úkonem provedli. I přes počáteční potíže s navigací se testeři dokázali za pomoci domovského tlačítka vrátit na výchozí navigační lištu. Dle jejich slov byla navigace komplikována také nezvyklým vzhledem ikon na tlačítkách v liště, kdy u některých tlačítek na první pohled nerozpoznali význam ikony. Lišta určená pro manipulaci s projekty byla na základě prvního dojmu testerů taktéž zdrojem nejasností, což bylo dle jejich slov způsobeno především odlišným přístupem k projektům narozdíl od konkurenčních editorů. Další mírnou nejasností testerů bylo, kterých souborů se aplikace motivu týká, a který motiv bude zvolen v případě otevření nového souboru. Samotný první dojem na vzhled uživatelského prostředí, byl u všech testerů pozitivní a z jejich úst zazněla také slovní pochvala vzhledu aplikace, a to včetně vzhledu ikon i přes občasnou nejednoznačnost jejich významu.

Počátkem druhé fáze testování bylo následné objasnění funkcionality a uživatelského rozhraní aplikace testerům. Cílem bylo dostatečně objasnit veškeré aspekty týkající se editoru s tím, že více detailně byly vysvětleny ty části editoru, které danému testerovi činily potíže v předchozí testovací fázi. Detailní vysvětlení se u všech testerů týkalo především navigační lišty a manipulace s projekty. Po vysvětlení následovalo opětovné vyzkoušení aplikace testery. Následná práce testerů s editorem již probíhala bez potíží a testeři bez problémů používali navigační lištu včetně využití klávesových zkratek. Testeři neměli potíže s vytvořením nového projektu včetně vložení zdrojových souborů do projektu. Následné sestavování projektů již bylo pro testery triviální záležitostí. Pět testerů zajímalo, jakým způsobem probíhá vytvoření nového souboru Makefile. Samotný princip tvorby objektů pro jednotlivé soubory v kombinaci s následným slinkováním objektů, byl těmto testerům vysvětlen. Vytvoření zvýrazňování pro vlastní jazyk si všichni testeři vyzkoušeli již v první

fázi testování s tím, že ve druhé fázi jim byly popsány veškeré aspekty tvorby zvýrazňování, což způsobilo tvorbu pokročilejších zvýrazňování ze strany testerů ve druhé fázi, kdy po upresnění pojmů dokázali především lépe využívat priorit regulárních výrazu pro prvky jazyka.

Cílem poslední fáze testování bylo získání statistiky testování za pomocí vytvořeného testovacího protokolu obsahujícího otázky pokládané testerům. Některé otázky se týkaly vzhledu aplikace a jiné se naopak týkaly její funkcionality. Otázky byly vytvořeny tak, aby do jejich odpovědí šly přímo zahrnout i výsledky a poznatky z prvních dvou fází testování. Poslední otázka byla věnována právě funkcionalitě, kterou testeři postrádali vůči existujícím řešením. Celkem bylo testerovi položeno 24 otázek s tím, že byl záměrně zvolen velký počet otázek, aby bylo možné získat co nejvíce potřebných informací od testerů. Statistika testování reprezentuje množinu odpovědí na otázky z testovacího protokolu. Odpovědi na otázky byly někdy velmi různorodé a hned v několika případech měl jediný tester velmi odlišný názor než zbytek testerů. V několika případech se také dva různé názory zhruba dělily mezi dvě poloviny počtu testerů. Následující kapitola se zabývá výsledným shrnutím získané statistiky testování. Kompletní statistiku testování lze nalézt v příloze D. Zde je zobrazen stručný seznam nejčastějších výhrad a přípomínek:

- Nezvykle působící hierarchická navigační lišta a vzhled jednotlivých ikon.
- Chybějící zástupný text v malých řádkových editorech, do kterých se vkládají parametry týkající se překladu.
- Sloupec s čísly řádků je umístěn poměrně blízko samotnému textu.
- Nejasnost, jakých souborů se aplikace motivu týká.
- Chybějící možnost změny motivu globálně pro všechny otevřené soubory.
- Chybějící možnost změny přítomných klávesových zkratek.
- Absence vyhledávání textových vzorů v rámci celého projektu.
- Chybějící možnost refaktorizace zdrojového kódu.
- Absence porovnání dvou zdrojových souborů přímo v editoru.

5.3 Výsledky testování

Po úspěšném dokončení testování byla získána potřebná zpětná vazba od testerů. Názory testerů na některé aspekty editoru byly téměř stejné, nicméně v některých názorech se testeři znatelně odlišovali, což napovídá tomu, že každý tester má odlišné požadavky a zvyky v rámci vývoje softwaru. Z výsledků testování by se obecně dalo dojít k závěru, že vytvořená aplikace nemá téměř žádné nedostatky z hlediska jeho funkcionality, nicméně z hlediska jeho uživatelského rozhraní, by bylo vhodné provést mírné úpravy, které by potenciálně vylepšily celkový dojem v rámci používání aplikace. Velmi dobrou zprávou je, že celkový počet výhrad testerů vůči editoru byl nízký a pozitivní ohlasy naprosto převažovaly. Dobrou zprávou je také skutečnost, že potřebné úpravy editoru, které by reagovaly na výhrady testerů, nejsou ve většině případů nikterak komplikované a nezabraly by mnoho času.

V rámci získávání zpětné vazby na funkcionalitu editoru, zazněla z úst testerů chvála a překvapení, jakožto reakce na funkcionalitu, kterou konkurenční textové editory nenabízejí. Reakce testerů na přítomnost manuálního zvýrazňování úseků textu, byla ve většině případů taková, že to je funkce, která je velmi zajímavá a užitečná, ale doposud se s ní u existujících řešení nesetkali. Dále byly zaznamenány velmi pozitivní ohlasy na možnost definice zvýrazňování syntaxe pro vlastní jazyk. Manipulace s projekty byla testery shledána jako snadno pochopitelná, intuitivní a pohodlná. Z hlediska chybějící funkcionality byly testery zmíněny jen dvě věci, jejichž absence údajně nečila problém, ale jednalo se spíše o návrh do budoucna. Prvním návrhem bylo zařazení refaktorizace zdrojového kódu do editoru. Tato funkce by umožnila automatické zarovnání a modifikaci zdrojového kódu do takové podoby, aby byl výsledný zdrojový kód čitelnější a čistší. Druhý návrh se týkal porovnání dvou zdrojových souborů přímo v editoru. Touto funkcí by bylo umožněno zjištění rozdílů mezi dvěma různými soubory. Velmi pozitivní zprávou je skutečnost, že žádný z testerů nepostrádal funkcionalitu, která byla v rámci návrhu aplikace vynechána z důvodu její náročnosti na implementaci.

Obecný názor testerů na uživatelské rozhraní editoru byl pozitivní, nicméně bylo vytknuto několik drobností, které by jej mohly zdokonalit. Pozitivní ohlasy testerů si editor vysloužil především díky přehlednému a nepřekombinovanému rozhraní. Také styl uživatelského prostředí je dle testerů vydařený. Nápad využití pouze navigační lišty pro ovládání celého editoru, byl testery shledán jako správný, nicméně některé drobnosti údajně komplikovaly manipulaci s navigační lištou. Výhrada, která byla zmíněna všemi testery se týkala vzhledu jednotlivých ikon umístěných na tlačítkách v navigační liště. Bylo řečeno, že vzhled ikon je vydařený, nicméně by údajně trvalo zhruba jeden den používání aplikace, aby si tester na vzhled ikon zvykl a pamatoval si, co každá ikona vyjadřuje. Navigační lišty se také týkala výtka, která poukazovala na chybějící zástupný text v malých řádkových editorech, do kterých se vkládájí parametry týkající se překladu. Dále by testeři také ocenili možnost okamžité aplikace zvoleného barevného motivu na všechny aktivní záložky v editoru, čehož by mohlo být využito především v noci, kdy by chtěl uživatel globálně aktivovat tmavý motiv. V rámci vyhledávání v textu bylo testery navrženo, aby přibyla podpora vyhledávání v rámci celého projektu a také, aby byl nějakým způsobem zobrazen celkový počet nalezených výsledků. Posledním návrhem testerů bylo umožnění předefinovat zabudované klávesové zkratky, protože dle jejich slov jsou někteří testeři zvyklí používat vlastní kombinace kláves.

Kapitola 6

Závěr

Cílem této práce byl návrh a následná implementace svobodného textového editoru určeného pro operační systém Linux. Před návrhem textového editoru bylo provedeno srovnání nejvýznamějších existujících řešení a shrnutí tohoto srovnání bylo podkladem pro samotný návrh. V rámci návrhu bylo potřeba zohlednit funkcionalitu, kterou existující řešení buďto neposkytují, nebo ji poskytují jen některá řešení, obzvláště pak ta proprietární. Pro implementaci navrženého textového editoru bylo využito frameworku Qt v kombinaci s programovacím jazykem C++. Hlavní výhodou této kombinace je podpora jak operačního systému Linux, tak dalších operačních systémů určených pro osobní počítače.

Implementace navrženého textového editoru byla úspěšně dokončena a aplikace je plně použitelná pro vývojáře. Pro získání zpětné vazby bylo provedeno uživatelské testování, kdy byla výsledná aplikace představena skupině testerů, kteří měli příležitost si aplikaci vyzkoušet a následně sdělit svůj názor jak na uživatelské rozhraní aplikace, tak na její funkcionalitu. Získání zpětné vazby od testerů bylo úspěšné, díky čemuž bylo možné určit, které části textového editoru se dle slov testerů vydařily, a které části by měly být vylepšeny či upraveny.

Z hlediska vyhlídek do budoucna je velkou výhodou navrženého editoru jeho rozšiřitelnost. Díky objektovému návrhu aplikace, je možné snadno přidat do editoru novou komponentu, která může rozšířit jeho možnosti. Pokud by se naskytla příležitost rozšíření funkcionality aplikace, tak by bylo vhodné implementovat funkcionalitu, která byla navrhnuta testery při testování. Mezi tuto funkcionalitu patří refaktorizace zdrojového kódu a možnost porovnání dvou zdrojových souborů přímo v editoru. Dále by bylo možné rozšířit schopnosti editoru za pomocí funkcionality, která byla vyřazena z návrhu editoru. Mezi tuto funkcionalitu patří podpora manipulace s Git repozitářem, možnost editace textu za pomocí více textových kurzorů a také zabalování bloků kódu. Textový editor lze samozřejmě rozšířit také z hlediska počtu podporovaných jazyků v rámci zvýrazňování syntaxe. Z výkonnostního hlediska lze také aplikaci vylepšit, což by šlo provést například využítím vícevláknového výpočtu při analýze zdrojového souboru v rámci konstrukce stromu tříd. Zdokonalení aplikace lze provést také vylepšením editace textu ve stylu editoru Vim. Výsledná aplikace již v základu podporuje širokou škálu příkazů Vimu, nicméně je v tomto směru pořád prostor pro mírná vylepšení. Vzhledem k použitému frameworku by bylo dále možné zajistit běh výsledné aplikace na více operačních systémech. Ačkoliv je zdrojový kód kompatibilní napříč operačními systémy pro osobní počítače, tak je potřeba pozměnit umístění konfiguračních souborů, a také je potřeba zajistit správné spuštění překladače při sestavování projektu, protože napříč různými systémy můžou být rozdíly v překladu.

Literatura

[1] *Atom - A hackable text editor for the 21st Century.* [online]. [cit. 2018-05-11]. Dostupné z: https://atom.io/.

[2] *C Language - Overview.* [online]. [cit. 2018-05-11]. Dostupné z: https://www.tutorialspoint.com/cprogramming/c_overview.htm.

[3] Chroboczek, M.: *Grafická uživatelská rozhraní v Qt a C++.* Computer Press, 2013, ISBN 978-80-251-4124-3.

[4] *Code::Blocks - The open source, cross platform, free C, C++ and Fortran IDE.* [online]. [cit. 2018-05-11]. Dostupné z: http://www.codeblocks.org/.

[5] Coppola, D.: *Market share of the most used C/C++ IDEs in 2018, statistics and estimates.* [online]. [cit. 2018-05-11]. Dostupné z: http://blog.davidecoppola.com/2018/02/market-share-most-used-c-cpp-ides-in-2018-statistics-estimates/.

[6] *Doxygen - Generate documentation from source code.* [online]. [cit. 2018-05-11]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/.

[7] *The GNOME text editor.* [online]. [cit. 2018-05-11]. Dostupné z: https://wiki.gnome.org/Apps/Gedit.

[8] *HTML Introduction.* [online]. [cit. 2018-05-11]. Dostupné z: https://www.w3schools.com/html/html_intro.asp.

[9] *Kate - Get an Edge in Editing.* [online]. [cit. 2018-05-11]. Dostupné z: https://kate-editor.org/about-kate/.

[10] *What is Linux?* [online]. [cit. 2018-05-11]. Dostupné z: https://www.linux.com/what-is-linux.

[11] *An Introduction to Makefiles.* [online]. [cit. 2018-05-11]. Dostupné z: https://www.gnu.org/software/make/manual/html_node/Introduction.html.

[12] *The GNU nano.* [online]. [cit. 2018-05-11]. Dostupné z: https://www.nano-editor.org/.

[13] *The official home of the Python Programming Language.* [online]. [cit. 2018-05-11]. Dostupné z: https://www.python.org/.

[14] *QRegularExpression Class.* [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/qt-5/qregularexpression.html.

[15] *QSize Class*. [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qsize.html.

[16] *QSplitter Class*. [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/qt-5/qsplitter.html.

[17] *QSyntaxHighlighter Class*. [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qsyntaxhighlighter.html.

[18] *QTextStream Class*. [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qtextstream.html.

[19] *QTreeWidget Class*. [online]. [cit. 2018-05-11]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qtreewidget.html.

[20] *Sublime Text - A sophisticated text editor for code, markup and prose*. [online]. [cit. 2018-05-11]. Dostupné z: https://www.sublimetext.com/.

[21] *Developer Survey Results 2017*. [online]. [cit. 2018-05-11]. Dostupné z: https://insights.stackoverflow.com/survey/2017.

[22] *Vim - The ubiquitous text editor*. [online]. [cit. 2018-05-11]. Dostupné z: https://www.vim.org/.

Příloha A

Obsah DVD

V této kapitole je uveden popis jednotlivých částí obsahu přiloženého DVD disku. Na disku jsou uloženy jak zdrojové texty aplikace, tak zdrojový text technické zprávy. Dále je zde přítomna také úplná programová dokumentace.

V adresáři *text* je umístěn zdrojový text technické zprávy. Zdrojové texty výsledné aplikace jsou umístěny v adresáři *src*, který dále zahrnuje adresář *resources*, v němž jsou uloženy veškeré použité ikony. Adresář *doc* zahrnuje programovou dokumentaci s výstupem ve formátu HTML [8]. Programová dokumentace byla vytvořena za pomoci aplikace Doxygen [6].

Příloha B

Návod na sestavení aplikace

V této kapitole je uveden návod na sestavení aplikace ze zdrojových textů. Návod se vztahuje k operačnímu systému Linux, a pro úspěšné sestavení aplikace je nutné mít na daném počítači nainstalován Qt Creator. Samotné sestavení lze provést dvěma odlišnými způsoby, které jsou popsány níže.

První možností je přímé využití aplikace Qt Creator. Počátečním krokem je spuštění Qt Creatoru s následným zvolením otevření existujícího projektu, a to přímo z úvodní obrazovky aplikace. Dále je nutné zvolit v dialogovém okně projektový soubor aplikace, který se nachází přímo v adresáři *src*. Po otevření projektu je nutné přepnout sestavovací režim z výchozího ladícího režimu, do režimu určeného k vydání aplikace. Následně jen stačí spustit překlad projektu za pomocí odpovídajícího tlačítka. Veškerá tlačítka jsou přehledně zobrazena v okně aplikace, a tudíž by neměl být problém s jejich nalezením.

Druhá možnost vyžaduje otevření adresáře *src* v aplikaci terminálu. Následně je nutné spustit aplikaci qmake[1], po jejímž spuštění dojde k vytvoření odpovídajícího souboru Makefile na základě struktury projektu. Zadáním příkazu make lze projekt následně sestavit.

[1] Multiplatformní sestavovací nástroj, který je obecně použitelný k sestavení libovolných projektů napsaných v jazyce C++

Příloha C

Testovací protokol

1. Lze uživatelské rozhraní považovat za přehledné a intuitivní?
2. Je uživatelské rozhraní dostatečně propracované?
3. Je uživatelské rozhraní vhodně vizuálně zpracované?
4. Nepůsobí editor spíše jako vývojové prostředí?
5. Poskytuje editor dostatečné množství barevných motivů?
6. Je editor použitelný i jako poznámkový blok?
7. Jak pohodlné je vytváření a sestavování projektů?
8. Bylo zvolení projektových souborů dobrým rozhodnutím?
9. Jak přehledný a především užitečný je strom tříd?
10. Nečiní potíže vkládání dalších souborů do projektu?
11. Je přepínání otevřených zdrojových souborů pohodlné a dostatečně rychlé?
12. Jak komfortní je změna velikosti písma?
13. Může vkládání šablon prvků jazyka zvýšit produktivitu?
14. Je skrytí komentářů užitečnou funkcí?
15. Má vlastní zvýrazňování zdrojového kódu praktické využití?
16. Poskytuje editor dostatečné množství klávesových zkratek?
17. Jak rychlé a užitečné je skrytí ovládacích prvků uživatelského rozhraní?
18. Je vyhledávání v textu dostatečně propracované?
19. Lze považovat možnost editace textu ve stylu editoru Vim jako výhodu?
20. Jak užitečný je vestavěný editor zvýrazňování syntaxe pro vlastní jazyky?
21. Nepůsobí tvorba zvýrazňování syntaxe pro vlastní jazyk komplikovaně?

22. Je automatické ukládání stavu editoru vítanou funkcí?
23. Je možnost rozdělení okna aplikace do více editorů nutnou funkcí?
24. Které funkce editor postrádá v porovnání s jinými textovými editory?

Příloha D

Statistika testování

1. Lze uživatelské rozhraní považovat za přehledné a intuitivní?

   - Nejdelší diskuze v rámci uživatelského testování náležela ve většině případů odpovědi právě na tuto otázku. Jisté výhrady vůči přehlednosti uživatelského rozhraní mělo všech devět testerů. Výhrady se týkaly především odlišnosti uživatelského rozhraní editoru od existujících řešení, kdy testerům jistou dobu trvalo, než si na ovládání aplikace zvykli. Na testery na první pohled nezvykle působila především hierarchická navigační lišta, která je zmátla hlavně při přechodu na její nižší úroveň, kdy úplně nepochopili, co přesně přechod způsobil. Testerům dále dělalo potíže si zvyknout na vzhled jednotlivých ikon umístěných na tlačítkách v navigační liště s tím, že by dle jejich slov údajně trvalo zhruba jeden den používání aplikace, aby si tester na vzhled ikon zvykl a pamatoval si, co každá ikona vyjadřuje. Pět testerů dále postrádalo chybějící zástupný text v malých řádkových editorech, do kterých se vkládají parametry týkající se překladu. Po pochopení principu rozhraní již testeři dále neměli potíže s ovládáním aplikace a samotné rozhraní následně označili jako vydařené a úsporné z hlediska potřebného prostoru v okně aplikace.

2. Je uživatelské rozhraní dostatečně propracované?

   - Testeři neměli výhrady vůči propracovanosti uživatelského rozhraní editoru. Například manipulace s projekty byla sedmi testery shledána jako velmi dobře propracovaná s tím, že provedení stromu tříd je dle nich též na velmi dobré úrovni.

3. Je uživatelské rozhraní vhodně vizuálně zpracované?

   - Vzhled aplikace byl testery shledán jako vydařený, s tím, že dva z nich měli výhradu vůči sloupci s čísly řádků, který je dle jejich názoru umístěn poměrně blízko samotnému textu a mohl by být více vzdálen. Pochvalu obdrželo i barevné provedení ikon, které dle testerů dodává editoru estetiku vzhledem k použitému pozadí editoru a navigační lišty.

4. Nepůsobí editor spíše jako vývojové prostředí?

   - Dle všech testerů editor rozhodně nepůsobí jako vývojové prostředí. Díky jediné navigační liště údajně aplikace působí kompaktně.

5. Poskytuje editor dostatečné množství barevných motivů?

- Všech devět testerů potvrdilo, že šest barevných motivů je naprosto dostačující počet s tím, že zvolené motivy jsou dobře navržené a použité barvy zvýrazňování jsou dobře čitelné na daném pozadí. Větší polovina testerů měla počáteční potíže pochopit, jakých souborů se aplikace motivu týká, a který motiv bude zvolen v případě otevření nového souboru. Této skupině bylo potřeba upřesnit chování aplikace při změně motivu. Po upřesnění chování již testeři neměli výhrady s tím, že dva z nich postrádali možnost změnit motiv globálně pro všechny otevřené soubory, čehož by se dle nich dalo využít především v noci při aktivaci tmavého motivu.

6. Je editor použitelný i jako poznámkový blok?

   - Všichni testeři potvrdili, že lze editor ihned po jeho spuštění používat jako klasický poznámkový blok. Dle čtyř testerů aplikace rozšiřuje schopnosti poznámkového bloku díky přítomnosti manuálního zvýrazňování zdrojového kódu, protože například zvýraznění nadpisů a některých slov může dle nich přispět k čitelnosti daného dokumentu.

7. Jak pohodlné je vytváření a sestavování projektů?

   - Samotné vytváření i sestavování projektů je dle všech testerů pohodlné a intuitivní, nicméně dle šesti z nich je použitý způsob manipulace s projekty odlišný od způsobu, kterého využívají existující řešení s tím, že je nutné si na odlišnosti zvyknout. Využití okna textového terminálu pro zobrazení výstupu překladu, je dle všech testerů vhodným řešením, které ušetří místo v okně aplikace.

8. Bylo zvolení projektových souborů dobrým rozhodnutím?

   - Názorem tří testerů bylo, že manipulace s projekty by mohla být řešena i bez použití projektových souborů s tím, že veškerá nastavení by byla ukládána pouze do konfiguračních souborů aplikace. Tito lidé posléze zmínili, že díky projektovým souborům je naopak možné snadno přenášet projekty mezi různými počítači. Zbytek testerů neměl vůči projektovým souborům výhrady.

9. Jak přehledný a především užitečný je strom tříd?

   - Všichni testeři označili strom tříd za zajímavou funkcionalitu, která není u konkurence obvyklá. Dva z nich i přes pochvalu této funkce sdělili, že by tuto funkci v praxi nevyužili. Dle zbytku testerů se jedná o velmi dobrý způsob, jak například rychle nalézt odpovídající deklaraci konkrétní funkce, která je právě editována. Testeři neměli výhrady vůči samotnému vzhledu stromu s tím, že zvolené ikony v uzlech rozhodně přispívají k lepší čitelnosti obsahu stromu.

10. Nečiní potíže vkládání dalších souborů do projektu?

    - Dva testeři byli mírně zmateni ze způsobu vložení zdrojových souborů do projektu, kdy očekávali, že výběr souborů bude automaticky nabídnut již při vytvoření nového projektu, narozdíl od explicitního vložení souborů po vytvoření. Zbytek testerů neměl výhrady proti vkládání souborů do projektu.

11. Je přepínání otevřených zdrojových souborů pohodlné a dostatečně rychlé?

- Všech devět testerů ocenilo způsob přepínání otevřených souborů, který je podle nich oproti konkurenci velmi rychlý, a navíc umožňuje svižné zobrazení náhledu daného souboru pouhým najetím kurzoru myši.

12. Jak komfortní je změna velikosti písma?

    - Změna velikosti písma jak pomocí klávesové zkratky, tak pomocí odpovídajícího nástroje v navigační liště, je dle všech testerů komfortní a především rychlá.

13. Může vkládání šablon prvků jazyka zvýšit produktivitu?

    - Vložení šablon prvků daného jazyka má dle osmi testerů pozitivní vliv na jejich produktivitu, kdy například vložení těla cyklu je poměrně pracnou záležitostí s tím, že šablona jim může ušetřit čas. Jeden tester nevidí v šablonách prvků smysl s tím, že dle jeho slov je schopen napsat tělo daného prvku velmi rychle a vložení šablony by pro něj bylo zdržením.

14. Je skrytí komentářů užitečnou funkcí?

    - Dle šesti testerů se jedná o hodně užitečnou funkci především v situaci, kdy je potřeba potřeba pochopit přesný význam některé části zdrojového kódu. Zbylí tři testeři neviděli v této funkci smysl a připadala jim zbytečná.

15. Má vlastní zvýrazňování zdrojového kódu praktické využití?

    - Každý z devíti testerů ocenil přítomnost manuálního zvýrazňování zdrojového kódu. Tato funkce jimi byla také nazvána jako nevídaná. Jeden tester navrhl, že by bylo vhodné přidat možnost postupného průchodu všemi zvýrazněními pomocí odpovídajících tlačítek v navigační liště.

16. Poskytuje editor dostatečné množství klávesových zkratek?

    - Všem testerům postačovalo dostupné množství klávesových zkratek s tím, že tři testeři by ocenili možnost změny přítomných klávesových zkratek, protože z jiných editorů jsou zvyklí používat i jiné kombinace kláves.

17. Jak rychlé a užitečné je skrytí ovládacích prvků uživatelského rozhraní?

    - Vůči rychlosti skrytí ovládacích prvků neměli testeři výhrady vzhledem k tomu, že ke skrytí lze provést pomocí jediné klávesy. Dle testerů je skrytí ovládacích prvků výhodnou funkcí především pro majitele přenosných počítačů.

18. Je vyhledávání v textu dostatečně propracované?

    - Názor všech testerů byl takový, že vyhledávání v textu je na velmi dobré úrovni s tím, že přítomnost vyhledávání pomocí regulárních výrazů je vítaná. Necelé polovině testerů chyběla přítomnost zobrazení počtu nalezených shod v textu s tím, že u konkurence je přítomnost této funkce obvyklá a neměla by chybět. Tři testeři postrádali možnost vyhledávat textové vzory v rámci celého projektu, což je dle nich mírný nedostatek vzhledem k tomu, že editor umožňuje pokročilou práci s projekty.

19. Lze považovat možnost editace textu ve stylu editoru Vim jako výhodu?

    - Dle šesti testerů je možnost editace textu ve stylu editoru Vim vítanou funkcí a hlavně nevídanou funkcí, kterou spousta uživatelů ocení. Tři testeři naopak tvrdili, že přítomnost této funkce je poměrně zbytečná vzhledem k tomu, že editor nabízí grafické uživatelské rozhraní.

20. Jak užitečný je vestavěný editor zvýrazňování syntaxe pro vlastní jazyky?

    - Všichni testeři označili tuto funkci jako užitečnou a vítanou. Dle sedmi testorů je tato funkce na poměry textových editorů nevídaná a má velký potenciál. Dle slov dvou z nich se jedná o funkci, kterou sami rozhodně využijí, neboť by si zvýrazňování syntaxe u některých jazyků chtěli mírně modifikovat.

21. Nepůsobí tvorba zvýrazňování syntaxe pro vlastní jazyk komplikovaně?

    - Na základě názoru všech testerů nelze tvorbu zvýrazňování považovat za komplikovanou. Naopak kombinace regulárních výrazů a přepínání barev daných prvků je podle nich velmi intuitivní.

22. Je automatické ukládání stavu editoru vítanou funkcí?

    - Dle pěti testerů je tato funkce nutností s tím, že hlavním důvodem má být konkurence, která automatické ukládání stavu editoru již delší dobu poskytuje.

23. Je možnost rozdělení okna aplikace do více editorů nezbytnou funkcí?

    - Podle názoru šesti testerů je tato funkcionalita v dnešní době naprosto nezbytná s tím, že důvodem je přítomnost obrazovek s vysokým rozlišením. Dle zbylých tří testerů se jedná o užitečnou funkci, nicméně nepovažují ji za nezbytnou.

24. Které funkce editor postrádá v porovnání s jinými textovými editory?

    - Odpověď testerů na poslední otázku byla poměrně různorodá. Jeden z testerů navrhnul zařazení refaktorizace zdrojového kódu mezi funkcionalitu editoru. Tato funkce by umožnila automatické zarovnání a modifikaci zdrojového kódu do takové podoby, aby byl výsledný zdrojový kód čitelnější a čistší. Refaktorizace kódu je funkcionalita typická výhradně pro vývojová protředí, nicméně dle slov testera by bylo vhodné ji do editoru zařadit nejenom z důvodu odlišení se od konkurence, ale samozřejmě také z důvodu její užitečnosti. Tři testeři by dle jejich slov ocenili přítomnost další funkcionality, která by umožnila porovnání dvou zdrojových souborů přímo v editoru s cílem přesného zjištění rozdílů mezi danými soubory. Žádný z testerů nezmínil v této otázce funkcionalitu, která byla v rámci návrhu aplikace vynechána z důvodu její náročnosti na implementaci.