Generování dokumentace ze zdrojových kódu

Martin Feigl

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Feigl**
Osobní číslo:	**A15046**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Generování dokumentace ze zdrojových kódů**
Téma anglicky:	**Documentation Generation from Source Codes**

Zásady pro vypracování:

1. **Popište základní přístupy k vytváření dokumentace ke zdrojovým kódům programů.**
2. **Uveďte základní postupy používané pro automatizovanou tvorbu dokumentace zdrojových kódů**
3. **Vytvořte přehled dostupných generátorů dokumentace zdrojových kódů, včetně jejich podstatných vlastností.**
4. **Podrobně popište nejčastěji používané generátory, uveďte ukázky jejich použití.**
5. **Srovnejte vybrané generátory a vytvořte návod, který bude sloužit jako pomůcka pro výběr vhodného generátoru.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **GAROUSI, Golara, Vahid GAROUSI-YUSIFO?LU, Guenther RUHE, Junji ZHI, Mahmoud MOUSSAVI a Brian SMITH. Usage and usefulness of technical software documentation: An industrial case study.Information and Software Technology [online]. 2015, 57, 664-682 [cit. 2017-11-11]. DOI: 10.1016/j.infsof.2014.08.003. ISSN 09505849.**

2. **ZANONI, Julio Cezar, Milton Pires RAMOS, Cesar Augusto TACLA, Gilson Yukio SATO a Emerson Cabrera PARAISO. A semi-automatic source code documentation method for small software development teams. In: Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD) [online]. IEEE, 2011, s. 113-119 [cit. 2017-11-11]. DOI: 10.1109/CSCWD.2011.5960063. ISBN 978-1-4577-0386-7.**

3. **Comparison of documentation generators. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_documentation_generators**

4. **Doxygen [online]. [cit. 2017-11-11]. Dostupné z: http://www.stack.nl/ dimitri/doxygen/**

5. **TOOLS TO GENERATE BEAUTIFUL WEB API DOCUMENTATION. SILVERMAN, Matt. MATTSILVERMAN [online]. 2013 [cit. 2017-11-11]. Dostupné z: http://www.mattsilverman.com/2013/02/tools-to-generate-beautiful-api-documentation.html**

Vedoucí bakalářské práce: **Ing. Petr Chalupa, Ph.D.**
Ústav řízení procesů

Datum zadání bakalářské práce: **15. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.5.2018

.....
podpis diplomanta

ABSTRAKT

Cieľom tejto práce je priblížiť čitateľom potrebu dokumentácie a zlepšiť im povedomie ohľadom tejto problematiky. Táto práca približuje samotnú dokumentáciu, ako k nej pristupovať a postupovať pri jej tvorbe a ako efektívnejšie dokumentovať zdrojový kód programov pomocou generátorov dokumentácie. Okrem iného obsahuje prehľady a ukážky dostupných generátorov, ich porovnanie a odporúčanie pre výber vhodného generátoru.

Kľúčové slova: dokumentácia, zdrojový kód, generátory dokumentácie

ABSTRACT

The primary focus of this work is to inform readers about the necessity of documentation, and to improve their awareness of this issue. This work draws closer the documentation itself, how to access the documentation and proceed with its creation, and how to document the source code of programs more effectively using generators for documentation. Moreover, the documentation includes overviews and examples of available generators, their comparison, and recommendations for selecting a suitable generator.

Keywords: documentation, source code, documentation generators

Na tomto mieste by som rád poďakoval rodine, priateľom, profesorom, pracovníkom školy a všetkým ľudom, ktorí ma v priebehu celého štúdia podporovali. Osobitné poďakovanie patrí pánovi Ing. Petrovi Chalupovi Ph.D. za jeho vedenie pri tejto bakalárskej práci, za jeho trpezlivosť, rady a pripomienky.

Prehlasujem, že odovzdaná verzia bakalárskej/diplomové práce a verzia elektronická nahraná do IS/STAG sú totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 DOKUMENTÁCIA **11**
1.1 DRUHY DOKUMENTÁCIE 11
1.1.1 Dokumentácia požiadavkou 11
1.1.2 Dokumentácia architektúry a dizajnu 12
1.1.3 Užívateľská dokumentácia 12
1.1.4 Marketingová dokumentácia 13
1.1.5 Dokumentácia zdrojového kódu 13
1.2 PRÍSTUPY K DOKUMENTÁCII ZDROJOVÝCH KÓDOV PROGRAMOV 14
1.2.1 Autor dokumentácie a kódu 14
1.2.2 Doba tvorenia dokumentácie 16
1.2.3 Úrovne dokumentácie 17
1.2.4 Samo-dokumentujúci sa kód 17
1.2.5 Prečo (ne)dokumentovať? 18
1.3 POSTUPY PRI TVORBE DOKUMENTÁCIE ZDROJOVÝCH KÓDOV PROGRAMOV 21
1.3.1 Zásady dokumentácie 22
1.3.2 Komentáre 23
1.3.2.1 Zápisy komentárov 23
1.3.2.2 Druhy komentárov 26
2 GENERÁTORY DOKUMENTÁCIE ZDROJOVÝCH KÓDOV PROGRAMOV **27**
2.1 TYPY TAGOV 27
2.2 ZÁKLADNÉ TAGY 28
2.3 ROZDELENIE GENERÁTOROV 28
II PRAKTICKÁ ČÁST **33**
3 UKÁŽKY POUŽITIA GENERÁTOROV **34**
3.1 DOCUMENT! X 34
3.1.1 Vlastnosti nástroja Document! X 34
3.1.2 Práca s nástrojom Document! X 35
3.2 DOXYGEN 39
3.2.1 Vlastnosti nástroja Doxygen 39
3.2.2 Práca s nástrojom Doxygen 40
3.3 JAVADOC 43
3.3.1 Vlastnosti nástroja Javadoc 43
3.3.2 Práca s nástrojom Javadoc 44
3.4 ROBODOC 47
3.4.1 Vlastnosti nástroja ROBODOc 47
3.4.2 Práca s nástrojom ROBODoc 48
4 POROVNANIE A VÝBER VHODNÉHO GENERÁTORU **50**
ZÁVER **53**
ZOZNAM POUŽITEJ LITERATÚRY **54**

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK.....58
ZOZNAM OBRÁZKOV 60
ZOZNAM TABULIEK 61
ZOZNAM PRÍLOH.....62

ÚVOD

Určite sme sa už všetci v bežnom živote stretli s nejakou formou dokumentácie. Môže ísť napríklad o dokumentáciu vo forme návodu k zakúpenému produktu. Bez používania dokumentácie sme nútení zisťovať informácie metódou pokus-omyl, čo nemusí byť vždy najlepší spôsob.

Dokumentácia je potrebná a neoddeliteľná časť dodávaného produktu. Rovnako to platí o dokumentácii zdrojového kódu programu. Akokoľvek môže byť proces dokumentovania pre programátora neobľúbený a nepríjemný, softvér bez dokumentácie nie je kompletný. Tvorba dokumentácie nemôže byť braná ako strata času a vždy by sme si mali dať záležať na jej kvalite.

Podľa rozsahu zdrojového kódu si pri menších projektoch môžeme vystačiť s jednoduchými komentármi, no pri väčších projektoch je výhodné použiť špecializovaný nástroj pre tvorbu dokumentácie.

V teoretickej časti sú prebrané všetky základné typy softvérovej dokumentácie. Zameranie je predovšetkým na dokumentáciu zdrojových kódov programov. Na jej prínos, prečo je potrebná a na výhody a nevýhody samotnej dokumentácie. Rovnako sú tu opísané prístupy, ktorými môžeme dokumentáciu vnímať a zásady a postupy komentovania, ktoré by sme mali všeobecne dodržiavať. Záver teoretickej časti je venovaný najčastejšie používaným generátorom dokumentácie zdrojových kódov programov, na ich efektívne využitie pri tvorbe dokumentácie, príkazy a ich samotné rozdelenie z viacerých hľadísk.

Praktická časť je venovaná konkrétnym vybratým generátorom. Ide o generátory, ktoré podporujú prácu s programovacím jazykom Java. Sú tu uvedené ich vlastnosti, možnosti, použitie. Taktiež je v tejto práci opísaný postup práce pri tvorbe samotnej dokumentácie a ich výsledné ukážky. Okrem toho tu nájdeme ich podrobné porovnanie a následne vytvorený stručný návod, ktorý má za úlohu slúžiť ako pomôcka pre výber vhodného generátora.

I. TEORETICKÁ ČÁST

1 DOKUMENTÁCIA

Softvérová dokumentácia je dôležitá časť softvérového inžinierstva, ktorá nám popisuje proces tvorby daného softvéru, ako softvér pracuje alebo ako sa používa. Jej úlohou je nám podať čo najpresnejšie informácie za účelom lepšej práce s novým produktom alebo technológiou [1].

1.1 Druhy dokumentácie

Na vzniku softvéru sa okrem programátorov podieľajú viacerí ľudia, preto softvérová dokumentácia môže mať rôzne poňatie pre rozličných ľudí, podľa ich zamerania. Z toho dôvodu môžeme rozlišovať [2, 3]:

- dokumentáciu požiadavkou,
- dokumentáciu architektúry a dizajnu ,
- užívateľskú dokumentáciu,
- marketingovú dokumentáciu,
- technickú dokumentáciu.

Na každú časť dokumentácie sú kladené iné nároky a požiadavky, čo má celkovo vplyv na použité množstvo textu, obrázkov, diagramov, tabuliek alebo grafov. Spolu dostávame množstvo dôležitých informácií a poznatkov, ktoré vo finálnom dôsledku hrajú v softvérovom priemysle významnú úlohu. Každý produkt by mal mať rozhodne svoju dokumentáciu, dobrá dokumentácia robí produkt profesionálnejším, kvalitnejším, prívetivejším, čo má vplyv na samotnú rozšíriteľnosť daného softvéru [1, 3] .

1.1.1 Dokumentácia požiadavkou

Požiadavky sú vlastnosti softvéru, ktoré od nás požaduje používateľ alebo zákazník za účelom dosiahnutia stanoveného cieľu. Dokumentácia požiadavkou sa používa počas celého procesu vývoja a je to popis toho, ako má daný softvér fungovať, čo má robiť a čo všetko musí spĺňať. Tvorí základ pre to čo bolo alebo bude realizované. Potreba dokumentácie požiadavkou sa zvyčajne vzťahuje na zložitosť produktu, vplyv výrobku a dĺžku života softvéru [3].

Poznáme funkčné požiadavky, ktoré určujú chovanie a funkcie softvéru a nefunkčné požiadavky, ktoré špecifikujú vlastnosti a zahŕňajú obmedzujúce podmienky daného produktu. Sú to napríklad požiadavky na prevádzku systému (počet užívateľov), na softvér

(efektívnosť, bezpečnosť, hardvérové prostredie, operačný systém) a externé požiadavky (legislatívne a etické záležitosti) [3].

S dokumentáciou požiadavkou sa stretávajú všetci ľudia, ktorí sa podieľajú na vývoji a tvorbe softvéru. Správne zhrnutie požiadavkou je veľmi dôležité, ich zlá formulácia býva ako častá príčina chýb pri vývoji softvéru. V praxi sa veľmi často stretávame s tým, že zákazník nikdy nevie čo presne chce [3].

1.1.2 Dokumentácia architektúry a dizajnu

Architektúra softvéru nám podáva obraz o základných konštrukciách alebo vlastnostiach systému, ich vzájomných väzbách, princípoch a predpisoch, ktoré nás viedli počas návrhu a vývoja v priebehu času. Zaoberá sa problematikou štruktúry a organizácie softvérového systému. S rastúcou veľkosťou a zložitosťou systému sa môže stať návrh a špecifikácia celkovej štruktúry väčším problém ako voľba algoritmov a dátových štruktúr [4].

Návrh alebo dizajn softvéru je proces definovania softvérových metód, funkcií, objektov a celkovej štruktúry a interakcie kódu tak, aby výsledok našej práce splňoval zadané požiadavky. Dizajn by mal byť celkovo dobre premyslený a preskúmaný skôr, než sa začne s písaním kódu. Je jednoduchšie riešiť problémy v začiatku vývojového cyklu, než robiť zásadnú zmenu dizajnu po zapísaní veľkého množstva kódu [4].

Dizajn softvéru by mal teda opisovať napríklad všetky databázy, frameworky, nástroje, hardvér, mal by obsahovať popis celej architektúry. Všetky dokumenty týkajúce sa tejto problematiky môžeme zaradiť do dokumentácie architektúry a dizajnu, z ktorej vo výsledku dostávame predstavu o tom, čo sa deje, ako a kde sa budú všetky časti softvéru vzájomne ovplyvňovať [4].

1.1.3 Užívateľská dokumentácia

Užívateľská dokumentácia popisuje všetky funkcie a časti softvéru, je tvorená za účelom pomoci užívateľovi pri používaní a zoznamovaniu sa s novým produktom. Pre užívateľa predstavuje záchytný bod pri riešení otázok a problémov, do ktorých sa dostane počas práce [2, 3].

Obvykle je tvorená technicky orientovanými vývojármi, ktorí poznajú každý kúsok danej aplikácie alebo softvéru. Niekedy môže byť pre nich veľa vecí samozrejmých a nedokážu sa na softvér pozrieť očami užívateľa. Z tohto dôvodu sa odporúča spolupráca vývojárov s oddelením technickej podpory, ktorá ma lepšie vedomosti a skúsenosti o obvyklom

chovaní a najčastejších chybách užívateľov. Na základe častej a priamej komunikácie so zákazníkmi sa vedia lepšie vcítiť do ich potrieb a zmýšľania, vedia podať informáciu užívateľsky zrozumiteľnejším spôsobom [5].

Poznáme tematické používateľské dokumentácie, ktoré sú obsiahlejšie a dané kapitoly sú orientované na určitú konkrétnu oblasť, aby užívateľ pochopil princípy skryté za funkcionalitou. Ďalej poznáme dokumentácie písané formou tutoriálov, teda funkčnými príkladmi, kedy navigujeme užívateľa postupnými krokmi. Je to reálna ukážka využitia nášho softvéru, príklady by mali byť okomentované, ľahko kopírovateľné a prípadne v spustiteľnom tvare. Posledným typom dokumentácie je forma listu, odkazu alebo inej referencie. Tento typ je určený pre používateľov, ktorí vedia čo hľadajú. Ide o zoznam príkazov alebo úloh, ktoré sú abecedne alebo logicky usporiadané a následne odkazujú na bližšie informácie [2, 3].

1.1.4 Marketingová dokumentácia

Rovnako ako v iných odvetviach, tak aj pri softvérových produktoch, je marketing obchodná politika zaoberajúca sa dosiahnutím maximálneho ekonomického zisku. Marketing má správne rozpoznať a poznať súčasné a budúce potreby trhu a konkrétnych skupín zákazníkov. Zoznamuje potenciálnych zákazníkov s produktom, informuje ich o ňom a vnucuje myšlienku jeho potreby. Ide o všetky činnosti, procesy a metódy spojené s prezentovaním a predajom služieb a produktov. Zaraďujeme sem teda činnosti ako predpredajné aktivity, propagovanie, riadenie značky, marketingové prieskumy, komunikáciu, PR, reklamu a cenotvorbu. Celý tento strategický plán týkajúci sa obchodných záležitostí o marketingu nájdeme v marketingovej dokumentácii [6].

1.1.5 Dokumentácia zdrojového kódu

Technická dokumentácia, ako to už vyplýva zo samotného názvu, má za úlohu čo najlepšie opísať softvérový produkt z technického hľadiska. Preto do tejto kategórie môžeme zaradiť aj dokumentáciu zdrojového kódu. Samotný zdrojový kód býva ako zdroj informácii nedostatočný, preto spolu s kódom musí byť písaný sprievodný text, ktorý popisuje jednotlivé časti programu. Dokumentácia zdrojového kódu sa teda zaoberá logikou daného systému a jednotlivými algoritmami, použitými technológiami, funkciami, pripravenými triedami a ich metódami, popismi rozhraní a mnohými ďalšími prvkami. Podáva informácie o všetkých implementovaných stavebných prvkov programu. Okrem toho nemôžeme

pozabudnúť na informácie o autorovi, verzii, históriu revízii, dátume a na ďalšie rôzne iné doplňujúce informácie [1, 3].

S miliónmi riadkov kódu programu napísaných každý deň nemožno brať na ľahko dôležitosť dobrej dokumentácie. Dokumentovanie zdrojového kódu je dôležitá nevyhnutná činnosť, ktorá má zásadný vplyv na efektivitu a kvalitu vývoja softvéru. Táto dokumentácia je primárne určená pre ľudí, od ktorých sa všeobecne očakáva, že majú znalosti ohľadom programovania, nie pre laických koncových užívateľov. Je pre samotných vývojárov, testerov alebo iných ľudí, ktorí sa podieľajú na vývoji, rozširovaní, renovácií alebo inej údržbe daného softvéru. Dôkladná dokumentácia popisujúca softvér, nie je len nejaký manuál, ale predstavuje východiskový bod pre akúkoľvek zmenu vykonanú v programe [1, 3].

Pre každého tvorcu softvéru je v prvom rade prioritou funkčnosť výrobku, ktorý bude odovzdaný zákazníkovi. Okrem funkčnosti by sa mal zamerať na optimálnosť zdrojového kódu, snažiť sa pracovať efektívne a dokončiť svoju prácu v čo najkratšom časovom intervale [1, 3].

Preto častokrát ide potreba dokumentácie do úzadia, vnímaná ako strata času. Takéto vnímanie môžeme vidieť napríklad pri agilnom spôsobe vývoja, ktorého jedna z hlavných čŕt je, že fungujúci softvér prevláda nad rozsiahlou dokumentáciou [1, 3].

1.2 Prístupy k dokumentácii zdrojových kódov programov

K tvorbe dokumentácii zdrojových kódov programov môžeme pristupovať viacerými spôsobmi. Na tvorbu sa môžeme pozerať z viacerých hľadísk a kritérií. Môžeme pristupovať z pohľadu autorstva, doby kedy bola dokumentácia tvorená, môžeme skúmať úroveň dokumentácie, výhody a nevýhody dokumentácie alebo či je vôbec samotná dokumentácia potrebná.

1.2.1 Autor dokumentácie a kódu

Pokiaľ sa bavíme o autorovi dokumentácie, tak pravdepodobne najčastejším prípadom, s ktorým sa stretneme v praxi, hlavne pri menších tímoch a projektoch, alebo samozrejme pokiaľ pracujeme sami, tak bude spravidla platiť situácia, že autor zdrojového kódu je zároveň aj autorom dokumentácie daného kódu [7].

Pokiaľ je dokumentácia písaná rovnakou osobou, ktorá je zároveň zodpovedná za vznikom kódu, tak najväčšia výhoda tkvie v tom, že samotný programátor vie zo všetkých najlepšie, aký mal zámer. Tým pádom by mala byť dokumentácia predurčená k tomu, aby obsahovala čo najpresnejšie údaje, lebo neexistuje osoba, ktorá by videla viac do kódu ako programátor, ktorý ho písal. Rovnako pokiaľ ide o tú istú osobu, tak odpadáva možnosť, že nastane nejaká nejasná situácia alebo problém, ktorý by vyžadoval zháňanie a vypočúvanie pôvodného autora kódu alebo iného človeka s technickými znalosťami, ktorý vidí do problematiky a kódu. Daný autor, ktorý by nám musel pomáhať s takýmto problémom, by nemusel byť práve ani dostupný ani mať čas. Ešte zložitejšia situácia a komunikácia by mohla nastať pokiaľ by pracovníci neboli na jednom mieste a ide o nejaké externé skupiny [7].

Druhou možnosťou spôsobu dokumentovania z pohľadu tvorcov je, pokiaľ dokumentácia je písaná inou osobou, ako je tvorca kódu. Môže dokonca aj nastať situácia, že na dokumentácii pracuje viacero osôb. Niektoré firmy zamestnávajú vyslovene ľudí, ktorí sa zaoberajú návrhom, tvorbou alebo údržbou dokumentácie. Sú schopní ju písať aj pre bežných užívateľov, ale aj pre technicky zdatných ľudí podieľajúcich sa na vývoji softvéru. Môžu sa zaoberať viacerými druhmi dokumentácii, nie len dokumentáciou zdrojového kódu. Takého človeka môžeme nazvať dokumentaristom (technical writer). Či už je táto osoba dokumentarista, alebo niekto iný, jeho prácou je dokumentovanie, čiže by malo ísť o osobu, ktorá je technicky zdatná. Zároveň pokiaľ preferujeme tento spôsob písania dokumentácie, tak očakávame, že tieto osoby budú mať lepšie komunikačné schopnosti než programátor, lepšie formulovanie a zostavovanie viet, vyjadrovanie jadra veci a interpretačné vlohy, tým pádom by dokumentácia mala byť o niečo kvalitnejšia a prívetivejšia ako od programátora [8, 9].

Dokumentaristi môžu niekedy naraziť na nejasnosti alebo problematiku nad ich rámec znalostí, a preto sa môže stať, že aj napriek tomu budú potrebovať pomoc a spoluprácu autora kódu. S každou nejasnosťou sa predlžuje čas písania dokumentácie. Pokiaľ je program obsiahly a príliš zložitý, tak nemusí byť správne použiť na dokumentovanie inú osobu ako je tvorca kódu. Na druhej strane, za predpokladu, že je ich výsledná dokumentácia kvalitná, odbremeňujú programátorov, ktorí sa môžu viacej sústrediť iba na písanie kódu. Ujasnenie si stanoviska ohľadom tvorcu dokumentácie bude vždy v prvom rade na politike firmy, jej možnostiach, nakladaní s ľudskými zdrojmi, veľkosti projektu, tímu alebo zložitosti programu [8, 9].

1.2.2 Doba tvorenia dokumentácie

Máme tri časové možnosti ako môžeme začať písať dokumentáciu. Najprv napísať dokumentáciu a až podľa nej začať s písaním samotného kódu, písať dokumentáciu a zdrojový kód súčasne alebo najprv napísať kompletný kód a až potom začať pracovať na dokumentácii [10, 11].

Tvorba dokumentácie pred začatím programovania je najmenej vyskytujúci sa prípad. S touto možnosťou sa stretávame minimálne a väčšina ľudí s touto možnosťou nemusí prísť do styku ani raz počas svojej práce. Predtým ako začneme programovať, sme oboznámení s danými požiadavkami a návrhmi, čo by mal daný program splňovať. Nemali by sme však písať dokumentáciu pred samotným programovaním, lebo len ťažko zahrnieme do dokumentácie všetky nečakané komplikácie a úpravy, s ktorými sa počas vývoja stretneme. Programátori nemajú obvykle radi dokumentovanie a nadšencov, ktorí by písali dokumentáciu skôr ako kód, zrejme moc nenájdeme. Na takýto štýl dokumentovania by sme mohli naraziť teda len zriedka, napríklad pri veľmi dôležitých projektoch, ako je napríklad jadrová elektráreň. Pri takomto projekte by sme dostali vopred už presne zdokumentované špecifické nemenné požiadavky a návrh dizajnu, lebo pri takýchto rizikových projektoch neostáva pre programátorov voľnosť a priestor na zmeny a špekulácie pri implementácii [10, 11].

Dokumentovanie by malo byť kontinuálnym procesom a mali by sme dokumentáciu písať zároveň s písaním zdrojového kódu. Je to pravdepodobne najlepší spôsob dokumentovania. Pri súčasnom písaní dokumentácie a kódu máme stále čerstvo v hlave všetky naše myšlienky a pochody a vieme ich najlepšie zapísať, bez toho aby sme s odstupom času museli dlho uvažovať nad naším pôvodným zámerom. Navyše neskôr by sme si nemuseli už nájsť ani čas. Vzniká tu síce riziko, že teda nemusíme mať dokončené finálne vlastnosti a budeme musieť spraviť ešte nejaké zmeny, tým pádom aj vzniká potreba prepísať niektoré veci v dokumentácii. Stále nám však ostane základná kostra, do ktorej už ľahšie premietneme naše zmeny [10, 11].

Veľmi zlým prístupom k tvorbe dokumentácie je ju písať až po tom, ako napíšeme kód. Je to zároveň najčastejší spôsob. Veľkú rolu v tom hrá samozrejme veľkosť projektu, je rozdiel komentovať funkciu o desiatich riadkoch a vrátiť sa k písaniu dokumentácie, potom ako sme strávili týždeň programovaním niekoľkých tisícov riadkov kódu. Čakať s potrebou dokumentovať na poslednú minútu má tendenciu prinášať menej kvalitný výsledok. Nemali

by sme vnímať dokumentáciu ako úlohu, ktorú budeme robiť neskôr. Nie sme s úlohou hotoví skôr, ako sa dokument zdokumentuje. V niektorých spoločnostiach môže byť navyše programátor, po tom ako bola dosiahnutá potrebná funkcionalita aplikácie, nútený sa presunúť na ďalší projekt, a potom neostáva priestor na dokumentáciu [10, 11].

1.2.3 Úrovne dokumentácie

Podľa množstva dostupných alebo poskytnutých informácií môžeme rozdeliť dokumentáciu na tri základné úrovne. Prvá úroveň dokumentácie nám poskytuje iba nejaký brief, teda stručné jasné informácie. Pri metóde alebo funkcii to môže byť napríklad iba jedna výstižná veta, ktorá nám vysvetľuje základný popis, k čomu daná funkcia slúži. Táto úroveň je pre rýchle ozrejmenie a vhodný výber funkcie, ktorú budeme potrebovať [3].

Druhá úroveň je podrobnejšia, ale stále neopisuje použitú vnútornú logiku. Sú to podrobnejšie informácie, kde zahŕňame popis vstupov a výstupov. Napríklad teda opisujeme použité dátové typy vstupných premenných a výstupné hodnoty, ktoré môže funkcia nadobúdať. Stále je však celá funkcia pre nás ako čierna skrinka (black-box), do ktorej nevidíme a zaujíma nás iba jej použitie [3].

Posledná tretia úroveň je najpodrobnejšia a najobsiahlejšia. Je to dokumentácia, ktorá je pre lepšiu prehľadnosť členená na rôzne kapitoly. Nájdeme tu napríklad použitý syntax, detailný popis vnútornej implementácie a parametrov, obrázky, diagramy, príklady, zdroje z ktorých sa čerpalo a vychádzalo alebo odkazy na podobné funkcie. Dôležitá je dokumentácia funkcií, ale pri štúdiu kódu je veľmi vhodné mať okomentované aj jednotlivé problematické a zložité konštrukcie samotného algoritmu [3].

Pre každú úroveň môže byť iný spôsob, ako sa dostať ku dokumentácii. Jeden príkaz môže byť pre zobrazenie prvej úrovne a ďalší príkaz pre zobrazenie druhej úrovne. V prvej úrovni taktiež môžeme mať rozklikávacie menu pre zobrazenie druhej úrovne. Tretia úroveň býva často ako html stránka [3].

1.2.4 Samo-dokumentujúci sa kód

Niektorí ľudia zastávajú názor, že dokumentácia nie je potrebná a, že kód by mal mať sám o sebe výpovednú hodnotu postačujúcu ako dokumentácia, mal by hovoriť sám za seba. Ide o zástancov tzv. samo - dokumentujúceho (self-documenting/self-describing) sa kódu, ktorý vychádza iba z kvalitného programovacieho štýlu. Kvalitný programovací štýl sa vyznačuje dodržiavaním názvovej konvencie a štruktúrovaného programovania [12, 13].

Názvová konvencia je súbor odporúčaných pravidiel pre správne pomenovanie premenných, funkcií, tried, metód a ďalších používaných prvkov. Takýto kód je písaný pomocou mnemotechnických názvov, teda v čitateľsky prívetivých názvoch pre ľudí, obvykle teda meno alebo výraz písaný v ľudskom jazyku priamo odzrkadľuje daný význam. Zaoberá sa aj samotnou dĺžkou názvov, veľkosťou začiatočných písmen alebo pravidlami ako písať viacslovné názvy. Preto správne pomenovanie môže mať rovnaký účinok ako komentár, ktorým by sme objasňovali danú vec. Poznáme viacero názvových konvencií, výber pravidiel záleží podľa programovacieho jazyka alebo aj podľa samotných spoločností, ktoré si môžu zaviesť svoje vlastné interné pravidlá. Napríklad v jazyku Java, rovnako aj v iných jazykoch, by sme mali používať písanie štýlom Camel case, čo je štýl písania začiatočných slov a druh oddeľovania jednotlivých slov začiatočnými písmenami pri viacslovných výrazov. Napríklad názvy tried by mali byť podstatné mená začínajúce veľkým písmenom (class Customer), metódy by mali byť slovesá začínajúce malým písmenom (setKilometers), rovnako aj premenné (float myKilometers) a konštanty by mali byť písané veľkými písmenami oddelenými podtržníkmi (static final int MAX_USERS) [12, 13].

Štruktúrované programovanie je spôsob programovania kedy si problém rozdelíme na menšie úlohy a program píšeme v malých jednoduchých blokov(podprogramoch), z ktorého vyskladáme riešenie. Využívame tri základné riadiace štruktúry – sekvencie kódu, vetvenia a cykly, vyhýbame sa neobmedzenému používaniu skokov a návestiam, ktoré je ťažké sledovať a udržiavať. Ďalej by sme mali správne využívať odsadzovanie, zvislé zarovnanie, medzery a tabulátory [12, 13].

Všeobecne aj keď človek nemá záujem o vytváranie dokumentácie, mal by mať rozhodne správny programovací štýl. Správny programovací štýl robí kód jasnejším, čistejším, estetickejším, vzbudzuje väčšiu profesionalitu a znižuje úsilie potrebné na čítanie a pochopenie kódu a užívateľovi tak uľahčuje prácu aj bez predošlých špecifických znalostí. Spolu so samo - dokumentujúcim sa kódom sa vynára, ale aj otázka ako je takýto spôsob vhodný pre projekty väčšieho rozsahu [12, 13].

1.2.5 Prečo (ne)dokumentovať?

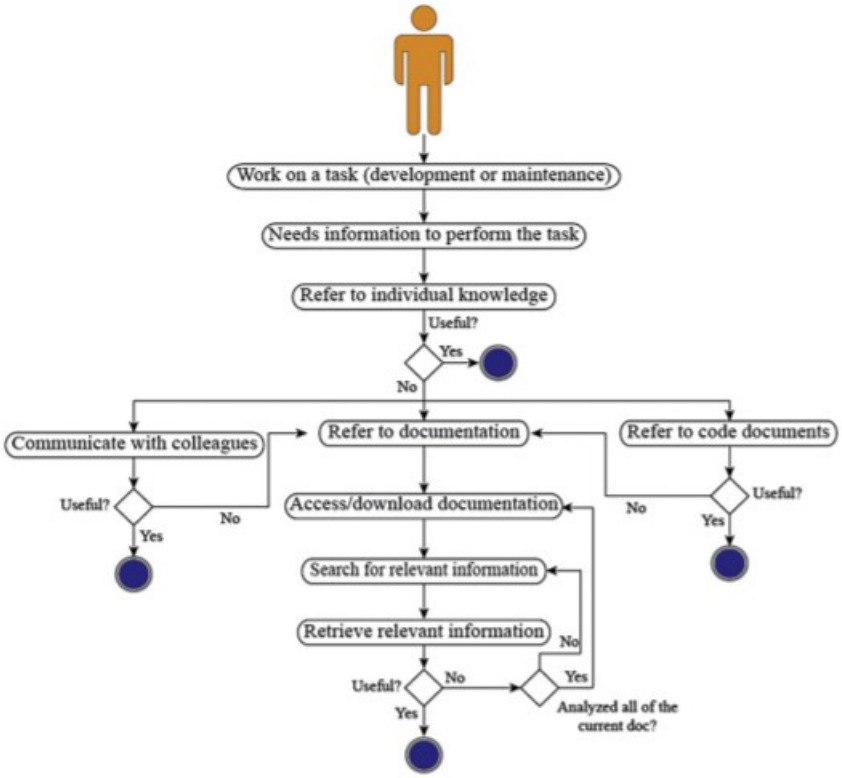
Akokoľvek nám môže pripadať dokumentovanie zdrojového kódu ako nezaujímavá a nepotrebná činnosť, musíme si uvedomiť, že s každou jednou vecou prichádzajú určité výhody a nevýhody. Samozrejme je rozdielna situácia, pokiaľ si píšeme doma sami pár riadkov kódu a pokiaľ pracujeme vo vývojárskom tíme na určitom projekte.

V prvom rade by nám mala dokumentácia poukázať na zámer, čo presne by mal daný úsek kódu robiť. Je veľmi dôležité si napísaný kód dôkladne okomentovať, aby sme aj do budúcna v prípade neskorších úprav vedeli, ako by mal program fungovať. Je veľmi nepravdepodobne, že si pri opätovnom pozretí na kód po dlhej dobe vybavíme ihneď náš pôvodný zámer. Pozerať kód, ktorý sme napísali napríklad pred mesiacom a celý si ho znova prezerať, aby sme ho pochopili je veľmi kontraproduktívne. Obvykle nás nezaujíma vnútorná logika, ktorú sme bez dokumentácie nútení prechádzať, ale iba funkcionalita [14, 15].

Softvér sa vo väčšine prípadov vyvíja v tímoch, preto musí existovať spolupráca nielen medzi programátormi, ale aj v rámci samotného kódu. Je potrebné, aby všetci vedeli, čo aká časť kódu robí. V prípade, že pracujeme v tíme alebo nami napísaný kód inak posielame napríklad externej skupine ďalej, je okomentovanie kódu dôležité, pretože iný vývojár by sa nemusel v našom kóde vyznať a nemusel by ho vedieť v prípade potreby upraviť [14, 15].

Taktiež môžeme byť napríklad preložený na rozrobenú prácu iného programátora a určite by sme neboli radi, keby sme v časovej tiesni a ešte by sme museli začínať od nuly, lebo by sme sa v danom kóde vôbec nevyznali. Môžeme byť v situácii, kedy budeme potrebovať využiť existujúci dostupný ovládač k nejakému zariadeniu, knižnicu vytvorených funkcií, ale nebude k nej dostupná dokumentácia. Sú dve možnosti ako sa z danej situácie dostať, zbytočne hodiny premýšľať nad tým, na čo slúži každá funkcia alebo znova začať úplne od začiatku [14, 15].

Bez dokumentácie sú budúci vývojári a ostatní ľudia podieľajúci sa na tvorbe softvéru nútení riskovať nebezpečné domnienky o zdrojovom kóde, musia prechádzať celú implementáciu alebo v najhoršom prípade musia vypočúvať samotného autora a jeho zámer. Tieto možnosti sú po väčšine neprijateľné, prípadne aj neuskutočniteľné a spomaľujú celý vývoj. Na obrázku dole (Obr. 1) môžeme vidieť pracovný postup získavania informácií a používania dokumentácie [1].



Obr. 1 Získavanie potrebných informácií [16].

Do dokumentácie môžeme zahrnúť aj upozornenia na potencionálne problémové oblasti. Komentáre môžu obsahovať dôležité faktory ako hraničné podmienky alebo platné rozsahy argumentov, čo je dobré spomenúť v prospech budúcich vývojárov a testerov [14].

Písanie dokumentácie nám zlepšuje a kráti čas na pochopenie kódu, ale zároveň samotné písanie zaberá čas. Tým pádom programátor miesto svojej primárnej práci je nútený písať komentáre, čo znižuje jeho produktivitu. Zároveň so zvýšenou časovou náročnosťou stúpajú aj financie vynaložené na vývoj produktu. Častokrát dokumentovanie je pre vývojára na obtiaž ako neobľúbená činnosť, ku ktorej má oproti písaniu kódu zníženú motiváciu. Niektorí ľudia majú problém aj so štylizáciou, vystihnutím podstaty dokumentácie alebo celkovo s vyjadrovacími schopnosťami [14, 15].

Pre veľa ľudí je ťažké si uvedomiť, ktoré časti kódu si vyžadujú dokumentáciu. Netreba dokumentovať tisícky riadkov kompletného kódu. Touto problematikou sa zaoberajú metódy vyhľadávania informácii v texte (information retrieval) alebo porovnávania vzorov (pattern matching), častokrát sa používa ako porovnávací vzor slovo. Jedným z nástrojov, ktorý sa to pokúša riešiť je nástroj Comente+ implementovaný v jazyku Java. Metóda tohto

nástroja prevádza extrakciu a analýzu informácii zo zdrojového kódu a všetkých priložených dokumentácii ku danému projektu. Táto metóda zhromažďuje informácie zo zdrojového kódu a porovnáva ich s komentármi a opismi obsiahnutými v priložených dokumentoch, ktoré sú písané prirodzeným jazykom. Týmto spôsobom je možné skontrolovať čo ide automaticky zdokumentovať, zdrojový kód by bol zdokumentovaný len v potrebných prípadoch a minimalizovala by sa redundancia dát. Tento nástroj sám o sebe nevytvára priamo dokumentáciu, slúži len pre lepšie kvalitnejšie vytvorenie dokumentácie konkrétnymi generátormi. Podobné analýzy sú veľmi prospešne pri práci v tímoch na veľkých projektoch, pri menších projektoch sú pravdepodobne nevýhodné kvôli značnému potrebnému vstupnému úsiliu [17].

Až správne zdokumentovaný kód sa stáva úplným. V praxi je častokrát kód viacej čítaný ako písaný. Dokumentovanie zdrojového kódu je nepopulárne, ale za to prináša viac pozitívnych dôvodov prečo dokumentovať než negatívnych. V konečnom dôsledku komentáre znižujú úsilie na pochopenie, údržbu a čitateľnosť kódu, zlepšujú rozšíriteľnosť a podporujú prenositeľnosť, teda priame opätovné využitie už vytvoreného kódu. Akokoľvek sa viacerí domnievajú, že kvalitný zdrojový kód a programovací štýl postačuje ako zdroj informácií, stále existuje viacero informácií o správaní kódu, ktoré sa ťažko vyjadrujú iba programovacím jazykom a preto vyžadujú silu a flexibilitu prirodzeného ľudského jazyka. Taktiež veľmi optimalizované algoritmy mávajú väčšinou v zdrojom kóde zložitú formu zápisu, ktorá je obvykle dosť ťažko čitateľná a preto sa u optimalizovaných algoritmov nevyhneme komentárom. Prioritou pred dokumentáciou bude pre vedenie vždy stihnúť zákazníkove záverečné termíny odovzdávania, prvoradá bude implementácia, funkcionalita, bezchybnosť a ostatné veci ktoré od produktu očakávame. Napriek tomu, ale na tvorbu dokumentácie sa stále veľmi často priradzuje príliš málo ľudských, finančných alebo časových zdrojov [1, 15].

1.3 Postupy pri tvorbe dokumentácie zdrojových kódov programov

Pri tvorbe dokumentácie zdrojových kódov programov by sme mali dodržiavať určité vecné zásady ohľadom toho ako by mala dokumentácia vyzerať a taktiež by sme mali dodržiavať pravidlá ohľadom zápisu a štýlu samotných komentárov.

1.3.1 Zásady dokumentácie

Je ťažké určiť ako by mala vyzerať kvalitná dokumentácia, každý človek vníma kvalitu rozlične a kvalita dokumentácie nevychádza z merateľných faktorov. Keby viacerí ľudia robili dokumentáciu na rovnakú vec, vždy by sme dostali o niečo rozdielny výsledok. Síce nie sú dané striktné pravidlá o tom ako písať dokumentáciu, ale všeobecne dobrá dokumentácia sa pridržiava niekoľkých základných pravidiel písania a vyvarovania sa chýb. Kvalitnú dokumentáciu spoznáme, keď ju čítame.

Najlepší spôsob ako písať dokumentáciu zdrojového kódu je písať ju priamo v zdrojovom kóde. Nemusíme si vytvárať žiadne nové textové dokumenty, ktoré by zhoršovali vyhľadávanie informácií alebo údržbu. Dokumentácia vznikajúca priamo vo vývojom prostredí bude ľahko dosiahnuteľná. Mala by byť vždy po ruke, oveľa lepšia dosiahnuteľnosť bude pri jednom kliknutí alebo pri klávesovej kombinácií než pri prechádzaní externých súborov či odkazov [18].

Ďalšou základnou vecou je chybovosť textu. Gramatické a iné chyby v jazyku nepridávajú na dobrom dojme. Ypsilony, čiarky, preklepy a iné chyby. Niekedy aj malý preklep môže spôsobiť dezorientáciu a nebezpečne zavádzať. Napríklad pri spracovaní dát môžu chybne podané informácie mať za následok fatálne následky. Chybám sa treba vyhýbať, kontrolovať si po sebe čo napíšeme a to je dôležitá vec pri každej jednej práci, pri dokumentácii detto. [13].

Veľa kráť sa stretávame s nekompletnou, v krajných prípadoch aj s úplne chýbajúcou dokumentáciou. Tomu by sme sa mali samozrejme pokúsiť vždy vyvarovať. Najčastejší dôvod je už viacero krát zmienený nedostatok času alebo pohodlnosť programátorov. Každý je hnaný za implementáciou vlastností programu. Dokumentácia žiadne nové vlastnosti neprináša a je to záťaž, ktorá nie vždy ihneď prináša benefity. Niekedy neúplnosť môže byť spôsobená písaním dokumentácie druhou osobou, ktorá nie je úplne zasvätená do danej problematiky. [13, 18].

Rovnako ako s nedostatočnou dokumentáciou, môžeme sa stretnúť aj s opačným prípadom a tým je príliš obsiahla dokumentácia. Všetkého veľa škodí a nie je to výnimka ani pri dokumentácii. S prebytočnou dokumentáciou vznikajú problémy ako dlhé hľadanie, nejasnosť, neprehľadnosť, horšia práca a údržba. Nemali by sme sa snažiť veci prílišne zaobaľovať. Mali by sme písať jasne a stručne, nemali by sme zbytočne vysvetľovať vetami to, čo ide opísať niekoľkými slovami. Nie je dôležité napísať príbeh, ale jednoducho podať

informácie, aby čitateľ nebol stratený. Pokiaľ je programátorovi dokumentácia nanútená, vytvára ju iba umelo, nezaoberá sa kvalitou, ale len preto aby ju mal čím skôr spravenú [13, 18].

Pri dokumentovaní je veľmi dôležitá aktuálnosť. Jeden dôvod neaktuálnosti je časový sklz pri nestíhaní záverečných termínov a spúšťaní produktu, kedy na reevidovanie dokumentácie neostáva čas. Druhý dôvod je neustály vývoj kódu. Kód môžeme časom meniť podľa potreby, dodatočne modifikovať, refaktorizovať. S refaktorizáciou často prichádzame o dokumentáciu a to môže viesť k dvom voľbám a to nefaktorizovať alebo nedokumentovať. Neaktuálnosť môže viesť k zmätenosti. Pokiaľ používame napríklad zložité funkcie, nezaujíma nás vnútorná logika a implementácia, ale funkcionalita. Preto ak dostaneme iné správanie, ako by sme očakávali, tak si pomyslíme, že nastala chyba v samotnom kóde, pritom celá situácia môže prameniť len z neaktuálnych informácii v dokumentácii [18].

Dokumentácia zdrojového kódu by mala byť vecne správna, dokumentovať by sme mali zrozumiteľne, aby sme informáciu správne pochopili a vedeli použiť. Proces dokumentácie by mal byť efektívny, jednoduchý a mal by byť schopný reagovať na zmeny. Vytváranie dokumentácie by sme mali plánovať od začiatku a nie až keď je to potrebné. Správna dokumentácia vie byť dobrým pomocníkom a vie poradiť, zle vypracovaná dokumentácia môže pomýliť a spraviť viac škody ako úžitku [1].

1.3.2 Komentáre

Komentáre sú špeciálne označené riadky textu v zdrojovom kóde, ktoré nie sú vykonané, ale majú len informačný účel. Sú to zlomky kódu, ktoré identifikuje kompilátor, ale úplne ich ignoruje, pretože nepredstavujú platnú akciu z hľadiska kompilácie kódu. Tým pádom nemajú žiadny vplyv na výsledný algoritmus. Používanie komentárov nie je povinné, ale je to najlepší dôvod ako tvoriť dokumentáciu zdrojového kódu. Pomáhajú človeku pochopiť program a môžeme ich dočasne používať vo fázach tvorenia a ladenia programu [17].

1.3.2.1 Zápisy komentárov

V jednotlivých programovacích jazykoch sa syntax a pravidlá ohľadom písania komentárov líšia a môžeme ich nájsť v špecifikáciách jednotlivých jazykov. Vo všeobecnosti komentáre rozdeľujeme na jednoriadkové a blokové. Niektoré programovacie jazyky umožňujú používanie obidvoch týchto druhov, ale niektoré jazyky napriek tomu podporujú len jednoriadkové komentáre [19].

V prípade jednoriadkového komentáru je každý znak od začiatočnej značky až do konca riadka interpretovaný ako komentár, na ktorý nebude braný ohľad pri kompilácii. Používajú sa na stručné a jednoduché opisy. Ukážky jednotlivých značiek môžeme vidieť v tabuľke nižšie (Tab. 1) [19].

Tab. 1 Začiatočné značky pre jednoriadkové komentáre [19].

Symbol	Jazyky
C	Fortran 77 a starší; znak 'C' musí byť v stĺpci 1 riadku, ktorý označuje komentár
REM, ::, :	BASIC, COMMAND.COM, cmd.exe, dávkové súbory
NB.	J (skratka z latinského výrazu Nota Bene – neprehliadni)
⍝	APL (mnemotechnický glyf pripomínajúci stolovú lampu, a preto "osvetľuje" vyššie uvedené)
#	Bourne shell a ostané UNIX shelly, Cobra, Perl, Python, Ruby, Seed7, Windows PowerShell, PHP, R, Make, Maple, Elixir, Nimrod
%	TeX, Prolog, MATLAB, Erlang, S-Lang, Visual Prolog
//	ActionScript, C (C99), C++, C#, D, Go, Java, JavaScript, Kotlin, Object Pascal (Delphi), Objective-C, PHP, Rust, Scala, SASS, Swift, Xojo
'	Monkey, Visual Basic, VBScript Small Basic, Gambas, Xojo
!	Fortran, Basic Plus, Inform, Pick Basic
;	AutoHotkey, AutoIt, Lisp, Common Lisp, Clojure, Rebol, Scheme, viaceré assemblery
--	Euphoria, Haskell, SQL, Ada, AppleScript, Eiffel, Lua, VHDL, SGML
*	COBOL (v prípade pevnej formy a * v stĺpci 7), PAW, viaceré assemblery, Fortran (v prípade pevnej formy a * v stĺpci 1), Pick Basic
\|\| | Curl |
" | Vimscript, ABAP |
\ | Forth |
*> | COBOL |

Blokové, teda viacriadkové komentáre používame tam kde píšeme podrobnejší opis, na ktorý nám jednoriadkové komentáre nestačia alebo nie sú efektívne. Používame ich napríklad na popisné bloky nad funkciami, cyklami alebo iných dlhších informácií. Obvykle je komentár umiestnený medzi dvojicou párových značiek. Pri viacriadkových komentároch by sme nemali nikdy zabúdať napísať druhú párovú značku, lebo komentár nebude ukončený a môže nám tým vzniknúť problém. Ukážky jednotlivých značiek môžeme vidieť v tabuľke nižšie (Tab. 2) [19].

Tab. 2 Párové značky pre blokové komentáre [19].

Symbol	Jazyky
¢ ~ ¢, # ~ #, co ~ co, comment ~ comment	ALGOL 68
/* ~ */	ActionScript, AutoHotkey, C, C++, C#, D, Go, Java, kotlin, Objective-C, PHP, PL/I, Rust, Scala, SASS, SQL, Swift, Visual, Prolog, CSS, JavaScript
#cs ~ #ce	AutoIt
/+ ~ +/	D
/# ~ #/	Cobra
<# ~ #>	Powershell
=begin ~ =cut	Perl
#`(~)	Perl6 (zátvorkové znaky môžu byť (), <>, {}, [], ľubovoľné Unicode znaky s BiDi zrkadlením, alebo Unicode znaky s Ps/Pe/Pi/Pf vlastnosťami)
=begin ~ =end	Ruby
{- ~ -}	Haskell
(* ~ *)	Object Pascal (Delphi),ML, Mathematica, Pascal, Seed7, Applescript, OCaml, Standard ML, Maple, Newspeak
{ ~ }	Object Pascal (Delphi), Pascal
|# ~ #|	Curl
%{ ~ %}	MATLAB (symboly musia byť v samostatnom riadku)
#| ~ |#	Lisp, Scheme, Racket
--[[~]], --[=[~]=]	Lua
" ~ "	Smalltalk
(comment ~)	Clojure

1.3.2.2 Druhy komentárov

Z významového hľadiska môžeme rozlišovať viaceré druhy alebo typy komentárov. Všeobecne rozlišujeme sedem základných druhov komentárov [20]:

- komentáre ohľadom autorských práv – obvykle sa nachádzajú na začiatku súboru, sú to informácie ohľadom licencie, názvu, dátume vydania, autorovi a ďalších vecí týkajúcich sa autorských práv,
- hlavičkové komentáre – sú to komentáre poskytujúce prehľad o funkcionalite tried, poskytujú informácie o autorovi, histórii revízií, obvykle sa nachádzajú pred deklaráciami tried,
- členské komentáre – popisujú funkcionalitu metód alebo polí, nachádzajú sa pred definíciou daného člena alebo v rovnakom riadku,
- vysvetľujúce komentáre – opisujú naše rozhodnutia týkajúce sa implementácie vnútri tela metódy,
- komentáre sekcií – sú to komentáre, ktorými opisujeme viaceré metódy alebo polia, ktoré majú rovnaký aspekt alebo význam, typický príklad je komentár ohľadom gettrov alebo settrov,
- pomocné komentáre – obvykle sú to iba dočasné komentáre, pomáhajú nám pri tvorení, ladení alebo údržbe,
- úlohové komentáre – informujú nás o zvyšných úlohách, ktoré sa majú implementovať, prípadných chybách, alebo o tom, ktoré časti sa majú pozmeniť.

2 GENERÁTORY DOKUMENTÁCIE ZDROJOVÝCH KÓDOV PROGRAMOV

Generátory dokumentácie zdrojového kódu sú softvérové nástroje, spadajúce do skupiny CASE nástrojov. Pomocou syntaktickej analýzy parsujú zdrojový kód a extrahujú z jeho komentárov potrebné informácie pre dokumentáciu. Taktiež vedia rozpoznať direktívy pre pripojenie iných súborov a tiež tieto súbory následne prechádzať a analyzovať. Okrem komentárov sa využíva samotný kód, ktorý slúži pre vytváranie väzieb medzi jednotlivými prvkami [21].

Generátory dokumentácie nám výrazne zjednodušujú a urýchľujú proces tvorby dokumentácie. Okrem ušetreného času nám pomáhajú ku komplexnosti, konzistentnosti dokumentácie a kódu, zlepšujú údržbu a aktuálnosť dokumentácie. Nástroje pre generovanie dokumentácii sú užitočné pre samotného autora, ale aj pre ostatných developerov, ktorých prídu do styku s daným programom a dokumentáciou [22].

2.1 Typy tagov

Generátory dokumentácie pri syntaktickej analýze venujú pozornosť komentárom, v ktorých si všímajú špecifické príkazy či značky zvané tagy. Musíme dodržiavať ich syntax, aby boli rozpoznané. Obvykle začínajú znakom ‚\' alebo ‚@' a všeobecne ich môžeme rozdeliť tagy do nasledujúcich kategórii [21]:

- indikátory štruktúry – slúžia pre popis celkovej štruktúry projektu, informujú o jednotlivých častiach projektu,
- indikátory sekcií – slúžia pre popis jednotlivých elementov, informujú napr. autorovi, dátume vytvorenia, bugoch, parametroch, návratových hodnotách,
- príkazy na vytváranie linkov – slúžia pre vytváranie odkazov a referencii na časti dokumentu,
- príkazy na zobrazovanie príkladov – slúžia pre pridávanie časti zdrojového kódu do dokumentácie,
- príkazy na vizuálne zlepšenia – slúžia pre zlepšenie vzhľadu dokumentácie.

2.2 Základné tagy

Každý generátor dokumentácie definuje svoje vlastné tagy. Počet dostupných tagov a ich význam sa môže mierne líšiť. Základné typy tagov, ktoré budú pri každom generátore veľmi podobné, pri niektorých generátoroch sú úplne totožné. Základné typy bez formátu atribútov môžeme vidieť v nasledujúcej tabuľke (Tab. 3) [21].

Tab. 3 Základné druhy tagov [21].

Tag	Popis
@author	meno autora daného elementu
@copyright	informácie o autorských právach
@version	číslo aktuálnej verzie
@class	meno triedy
@brief	stručný základný popis
@param	popis parametru funkcie
@return	návratová hodnota funkcie alebo metódy
@throws	popis výnimky
@todo	veci, ktoré treba ešte dokončiť

2.3 Rozdelenie generátorov

Na generátory dokumentácie sa môžeme pozerať z viacerých hľadísk. Môžeme ich rozdeliť z hľadiska podpory operačných systémov (Tab. 4), podpory programovacích jazykov (Tab. 5, Tab. 6, Tab.7), alebo z hľadiska výstupných formátov (Tab. 8). Pokiaľ je uvedená nejaká vlastnosť ako „čiastočne“, alebo „s pluginom“, tak o túto vlastnosť je generátor rozšírený až po stiahnutí nejakého doplnku alebo rozšírenia. Rovnako vlastnosť „nepriamo“, znamená, že napríklad Doxygen neumožňuje priamo mať výstupný formát PDF, ale má výstupný formát LaTeX, ktorý umožňuje prevedenie na formát PDF [21].

Tab. 4 Podpora operačných systémov [38].

	Windows	OS X	Linux	BSD	Unix
Ddoc [23]	áno	áno	áno	áno	nie
Document! X [22]	áno	nie	nie	nie	nie
Doxygen [21]	áno	áno	áno	áno	áno
Haddock [24]	áno	áno	áno	áno	áno
Imagix 4D [25]	áno	nie	áno	nie	áno
Javadoc [26]	áno	áno	áno	áno	áno
JSDoc [27]	áno	áno	áno	áno	áno
Natural Docs [28]	áno	áno	áno	áno	áno
perldoc [29]	áno	áno	áno	áno	áno
phpDocumentor [30]	áno	áno	áno	áno	áno
pydoc [31]	áno	áno	áno	áno	áno
RDoc [32]	áno	áno	áno	áno	áno
ROBODoc [33]	áno	áno	áno	áno	áno
Sphinx [34]	áno	áno	áno	áno	áno
Visual Expert [35]	áno	nie	nie	nie	nie
VSdocman [36]	áno	nie	nie	nie	nie
YARD [37]	áno	áno	áno	áno	áno

Tab. 5 Podpora programovacích jazykov [38].

	C, C++	Java	C#	VB, VBScript
Ddoc [23]	nie	nie	nie	nie
Document! X [22]	iba	áno	áno	áno
Doxygen [21]	áno	áno	áno	s pluginom
Haddock [24]	nie	nie	nie	nie
Imagix 4D [25]	áno	áno	nie	nie
Javadoc [26]	nie	áno	nie	nie
JSDoc [27]	nie	nie	nie	nie
Natural Docs [28]	čiastočne	čiastočne	áno	čiastočne
perldoc [29]	nie	nie	nie	nie
phpDocumentor [30]	nie	nie	nie	nie
pydoc [31]	nie	nie	nie	nie
RDoc [32]	čiastočne	nie	nie	nie
ROBODoc [33]	áno	áno	nie	áno
Sphinx [34]	áno	s pluginom	nie	nie
Visual Expert [35]	nie	nie	áno	nie
VSdocman [36]	nie	nie	áno	áno
YARD [37]	nie	nie	nie	nie

Tab. 6 Podpora programovacích jazykov [38].

	Delphi, Pascal	Ada	D	IDL
Ddoc [23]	nie	nie	áno	nie
Document! X [22]	nie	nie	nie	áno
Doxygen [21]	s pluginom	nie	čiastočne	áno
Haddock [24]	nie	nie	nie	nie
Imagix 4D [25]	nie	nie	nie	nie
Javadoc [26]	nie	nie	nie	nie
JSDoc [27]	nie	nie	nie	nie
Natural Docs [28]	čiastočne	čiastočne	nie	nie
perldoc [29]	nie	nie	nie	nie
phpDocumentor	nie	nie	nie	nie
pydoc [31]	nie	nie	nie	nie
RDoc [32]	nie	nie	nie	nie
ROBODoc [33]	áno	áno	áno	áno
Sphinx [34]	nie	s pluginom	nie	nie
Visual Expert [35]	nie	nie	nie	nie
VSdocman [36]	nie	nie	nie	nie
YARD [37]	nie	nie	nie	nie
	Access	Fortran	PHP	Perl
Ddoc [23]	nie	nie	nie	nie
Document! X [22]	áno	No	nie	nie
Doxygen [21]	No	áno	áno	nie
Haddock [24]	nie	nie	nie	nie
Imagix 4D [25]	nie	nie	nie	nie
Javadoc [26]	nie	nie	nie	nie
JSDoc [27]	nie	nie	nie	nie
Natural Docs [28]	nie	čiastočne	čiastočne	áno
perldoc [29]	nie	nie	nie	áno
phpDocumentor	nie	nie	áno	nie
pydoc [31]	nie	nie	nie	nie
RDoc [32]	nie	nie	nie	nie
ROBODoc [33]	nie	áno	áno	áno
Sphinx [34]	nie	s pluginom	s pluginom	nie
Visual Expert [35]	nie	nie	nie	nie
VSdocman [36]	nie	nie	nie	nie
YARD [37]	nie	nie	nie	nie

Tab. 7 Podpora programovacích jazykov [38].

	Python	Ruby	JavaScript	ActionScript
Ddoc [23]	nie	nie	nie	nie
Document! X [22]	nie	nie	nie	nie
Doxygen [21]	čiastočne	nie	nie	nie
Haddock [24]	nie	nie	nie	nie
Imagix 4D [25]	nie	nie	nie	nie
Javadoc [26]	nie	nie	nie	nie
JSDoc [27]	nie	nie	áno	nie
Natural Docs [28]	čiastočne	čiastočne	čiastočne	áno
perldoc [29]	nie	nie	nie	nie
phpDocumentor [30]	nie	nie	nie	nie
pydoc [31]	áno	nie	nie	nie
RDoc [32]	nie	áno	nie	nie
ROBODoc [33]	nie	áno	áno	čiastočne
Sphinx [34]	áno	s pluginom	áno	nie
Visual Expert [35]	nie	nie	nie	nie
VSdocman [36]	nie	nie	nie	nie
YARD [37]	nie	áno	nie	nie
	PL/SQL	Transact.SQL	Tcl	Haskell
Ddoc [23]	nie	nie	nie	nie
Document! X [22]	áno	nie	nie	nie
Doxygen [21]	nie	nie	áno	nie
Haddock [24]	nie	nie	nie	áno
Imagix 4D [25]	nie	nie	nie	nie
Javadoc [26]	nie	nie	nie	nie
JSDoc [27]	nie	nie	nie	nie
Natural Docs [28]	čiastočne	nie	čiastočne	nie
perldoc [29]	nie	nie	nie	nie
phpDocumentor [30]	nie	nie	nie	nie
pydoc [31]	nie	nie	nie	nie
RDoc [32]	nie	nie	nie	nie
ROBODoc [33]	áno	nie	áno	nie
Sphinx [34]	nie	nie	nie	nie
Visual Expert [35]	áno	áno	nie	nie
VSdocman [36]	nie	nie	nie	nie
YARD [37]	nie	nie	nie	nie

Tab. 8 Podpora výstupných formátov [38].

	HTML	CHM	PDF	LaTeX
Ddoc [23]	áno	nie	áno	áno
Document! X [22]	áno	nie	nie	nie
Doxygen [21]	áno	áno	nepriamo	áno
Haddock [24]	áno	nie	nie	nie
Imagix 4D [25]	áno	áno	nie	nie
Javadoc [26]	áno	nepriamo	nepriamo	nepriamo
JSDoc [27]	áno	nie	nie	nie
Natural Docs [28]	áno	nie	nie	nie
perldoc [29]	áno	nie	áno	nie
phpDocumentor [30]	áno	nie	nie	nie
pydoc [31]	áno	nie	nie	nie
RDoc [32]	áno	áno	nepriamo	áno
ROBODoc [33]	áno	nie	nepriamo	áno
Sphinx [34]	áno	nie	nie	nie
Visual Expert [35]	áno	nie	áno	nie
VSdocman [36]	áno	nie	nie	nie
YARD [37]	áno	nie	áno	áno
	PostScript	man-pages	DocBook	XML
Ddoc [23]	áno	áno	nie	áno
Document! X [22]	nie	nie	nie	nie
Doxygen [21]	nepriamo	áno	áno	áno
Haddock [24]	nie	nie	nepriamo	nie
Imagix 4D [25]	nie	nie	nie	nie
Javadoc [26]	nepriamo	nepriamo	nepriamo	nepriamo
JSDoc [27]	nie	nie	nie	nie
Natural Docs [28]	nie	nie	nie	nie
perldoc [29]	nie	nie	áno	áno
phpDocumentor [30]	nie	nie	nie	nie
pydoc [31]	nie	nepriamo	nie	áno
RDoc [32]	nepriamo	áno	áno	nie
ROBODoc [33]	nie	áno	nie	nie
Sphinx [34]	nie	nie	nie	nie
Visual Expert [35]	nie	nie	nie	áno
VSdocman [36]	nie	nie	nie	nie
YARD [37]	áno	áno	nie	áno

II. PRAKTICKÁ ČÁST

3 UKÁŽKY POUŽITIA GENERÁTOROV

Pre podrobnejší opis a následné ukážky použitia boli vybrané štyri generátory dokumentácie. Ide o generátory Document! X, Doxygen, Javadoc a ROBODoc. Všetky tieto generátory umožňujú tvorbu dokumentácie pre programovací jazyk Java, v ktorom bola tvorená vzorová ukážka pre dokumentovanie.

3.1 Document! X

Document! X je kombinácia automatizovaného dokumentačného nástroja a plného autorského prostredia, ktoré je možné použiť na vytváranie, publikovanie a udržiavanie presnej dokumentácie profesionálnej kvality [22].

Za vznikom Document! X stojí spoločnosť Innovasys, ktorá sa sformovala v roku 1997 a o rok neskôr vydala prvú verejnú verziu. Na Document! X sa sťahuje proprietárna licencia, teda ide o softvér, ktorý má obmedzenia na jeho používanie a kopírovanie. Ide o platený softvér, avšak Innovasys ponúka zdarma skúšobnú trial verziu [22].

3.1.1 Vlastnosti nástroja Document! X

Document! X je spustiteľný iba na operačnom systéme Windows. Tvorbu dokumentácie podporuje pre širokú škálu programovacích jazykov ako napríklad C++/CLI, Java, C#, VB, VBScript, IDL, COM (ActiveX) komponenty, webové služby (SOAP a REST), databázy (PL/SQL, Oracle, Access alebo iné databázy s ovládačmi OLEDB), projekty typu JavaScript a jQuery, XML schémy [22].

Výstupnými formátmi Document! X sú HTML stránky alebo kompilované Microsoft HTML Help stránky. Generovaný obsah zahŕňa napríklad tabuľku obsahu a indexu, full-textové vyhľadávanie, diagramy vzťahov medzi objektmi, alebo stromy dedičnosti [22].

Môžeme si vybrať či chceme tvoriť dokumentáciu priamo v zdrojovom kóde, alebo či využijeme dostupný Document!X Content File editor, ktorý ponúka možnosti rozšíreného obsahu a úprav. Tento editor nám zobrazuje zdrojové komentáre a môžeme vidieť, kde dokumentácia potrebuje ďalší opis alebo lepšie komentáre a je dobrý pre vykonávanie úprav po vývoji [22].

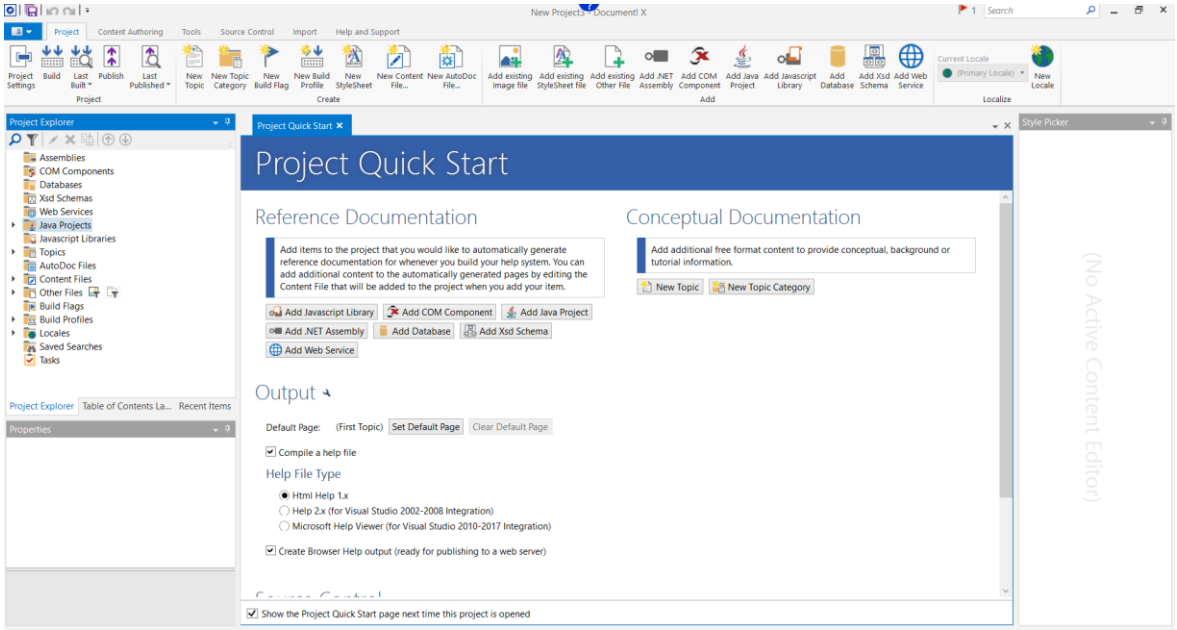
Document! X môže byť pridaný do integrovaného vývojového prostredia Microsoft Visual Studio. Toto spojenie umožňuje vytváranie, otvorenie, náhľad, udržovanie a spustenie

dokumentácie priamo z riešenia Visual Studia. Taktiež pri používaní Visual Studia získavame pomoc a referenciu pri zdokumentovaných veciach pri stlačení klávesy F1 [22].

Šablóny Dokument! X odrážajú súčasné štandard spoločnosti Microsoft pre technickú dokumentáciu, aby poskytli konzistentný a známy štýl a usporiadanie. Šablóny môžu byť plne prispôsobené (vrátane obsahu, typu a štýlu stránky, farebných schém atď.), bez potreby prepracovania obsahu [22].

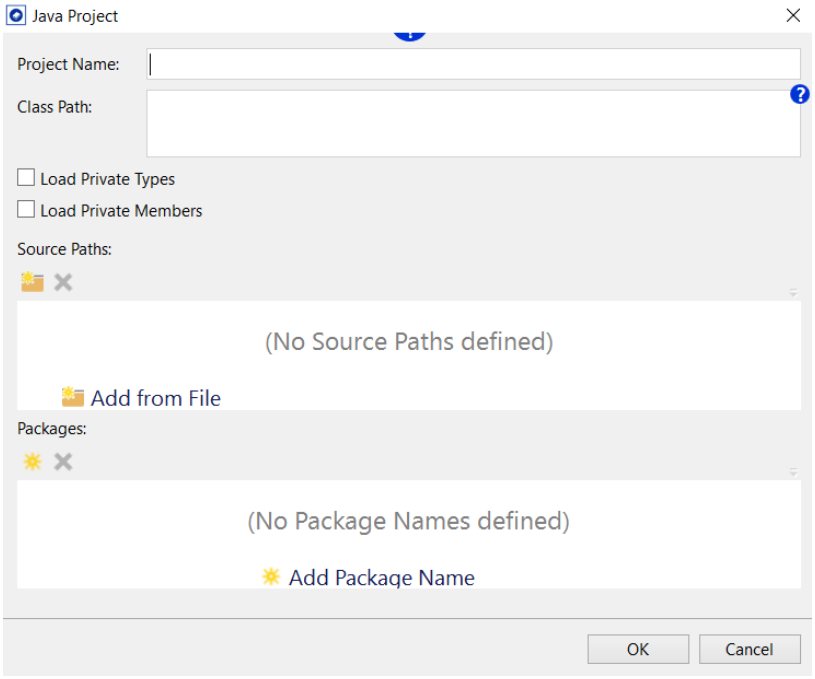
3.1.2 Práca s nástrojom Document! X

Inštalácia Document! X je veľmi jednoduchá a intuitívna. Na domovskej stránke si stačí vybrať či máme záujem o platenú alebo skúšobnú verziu. Počas inštalácie zvolíme miesto uloženia a súhlasíme s licenčnými podmienkami. Po úspešnej inštalácii vytvoríme nový projekt, ktorý pomenujeme a dostaneme sa na úvodnú stránku (Obr. 2).



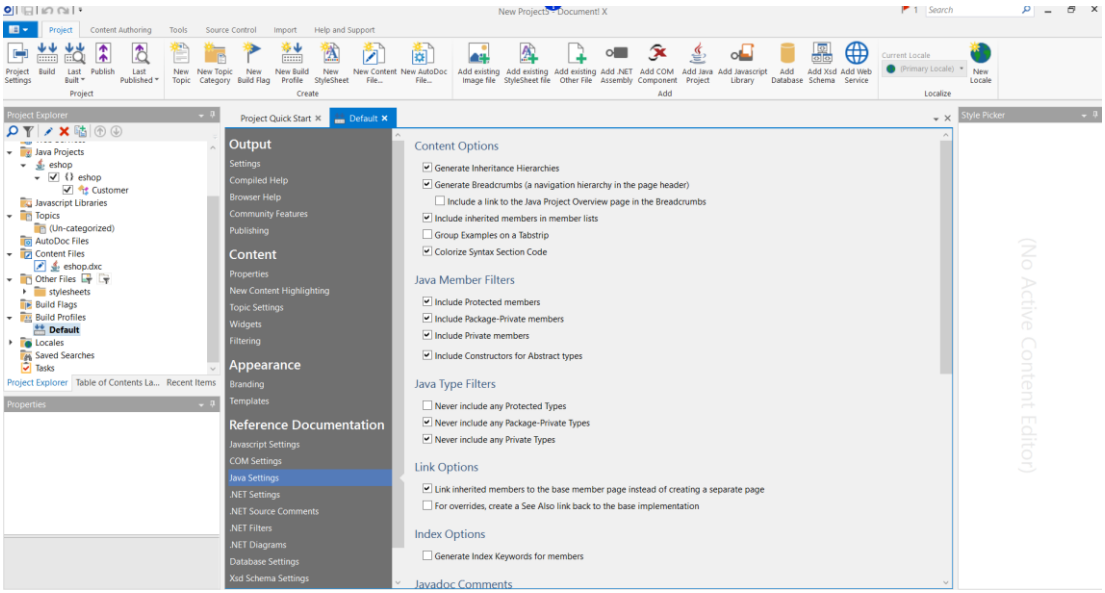
Obr. 2 Práca s nástrojom Document! X.

Na tejto úvodnej stránke klikneme na pridanie projektu Java , čo nám vyvolá nové okno (Obr. 3), kde zvolíme meno projektu, cestu k zdrojovému súboru Java a meno používaného balíčku.



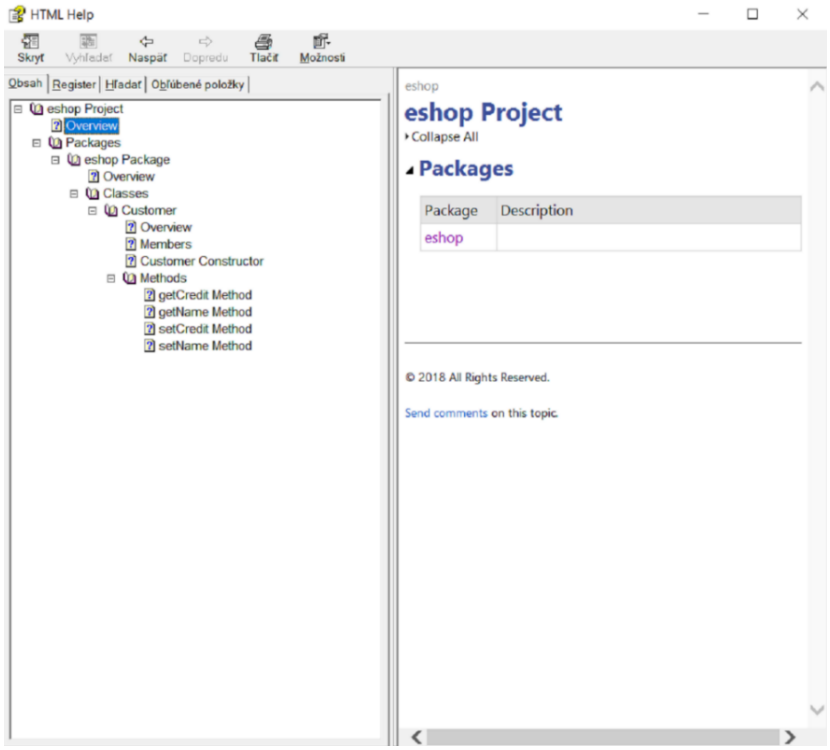
Obr. 3 Práca s nástrojom Document! X.

Na nasledujúcom obrázku (Obr. 4) môžeme v položke Java Projects vidieť, že sa nám pridal náš projekt a môžeme tu vybrať, ktoré časti majú byť obsiahnuté v dokumentácii. V položke Content Files môžeme vidieť náhľad dokumentácie a robiť v nej priamo úpravy. V položke Build Profiles môžeme meniť nastavenia pre náš projekt.

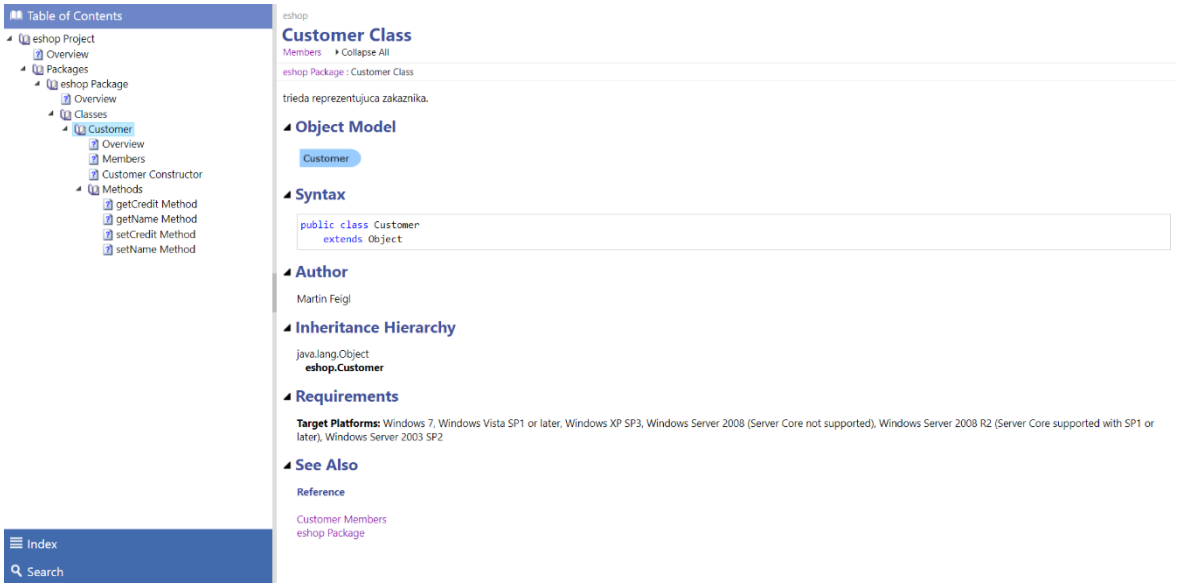


Obr. 4 Práca s nástrojom Document! X.

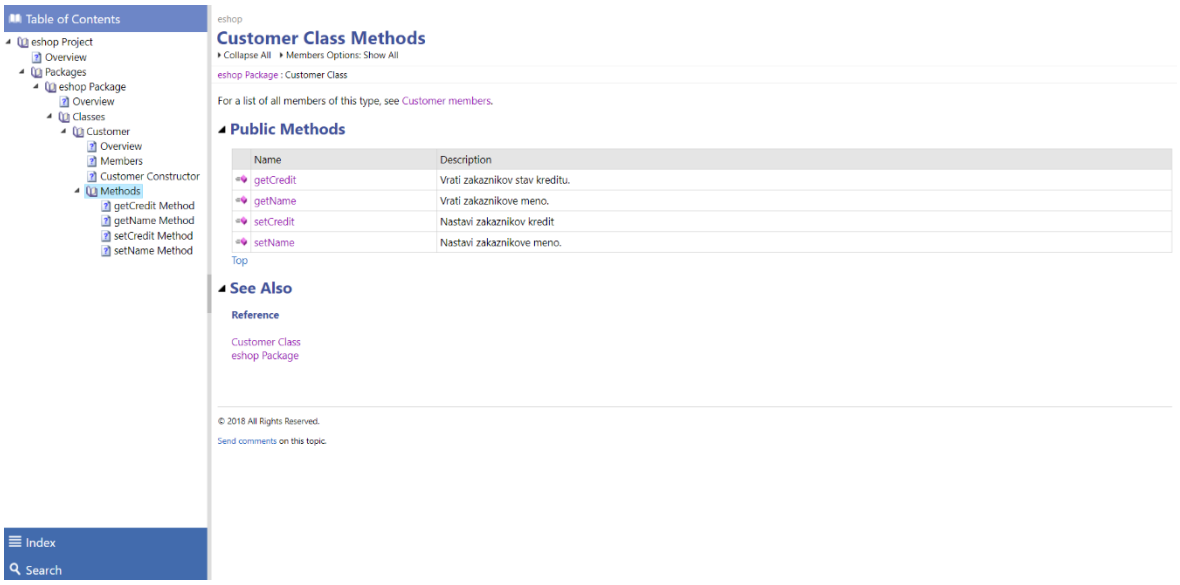
Potom stačí už len projekt zostaviť (build). Automaticky sa nám vygeneruje dokumentácia v obidvoch formátoch, HTML stránka (Obr. 6, Obr. 7, Obr. 8) a kompilovaná Microsoft Help stránka (Obr. 5). V obidvoch formátoch môžeme vidieť hierarchickú štruktúru a možnosť vyhľadávania. Na obrázku (Obr. 9) je znázornený zdrojový kód použitý pre tvorbu dokumentácie (Obr. 8).



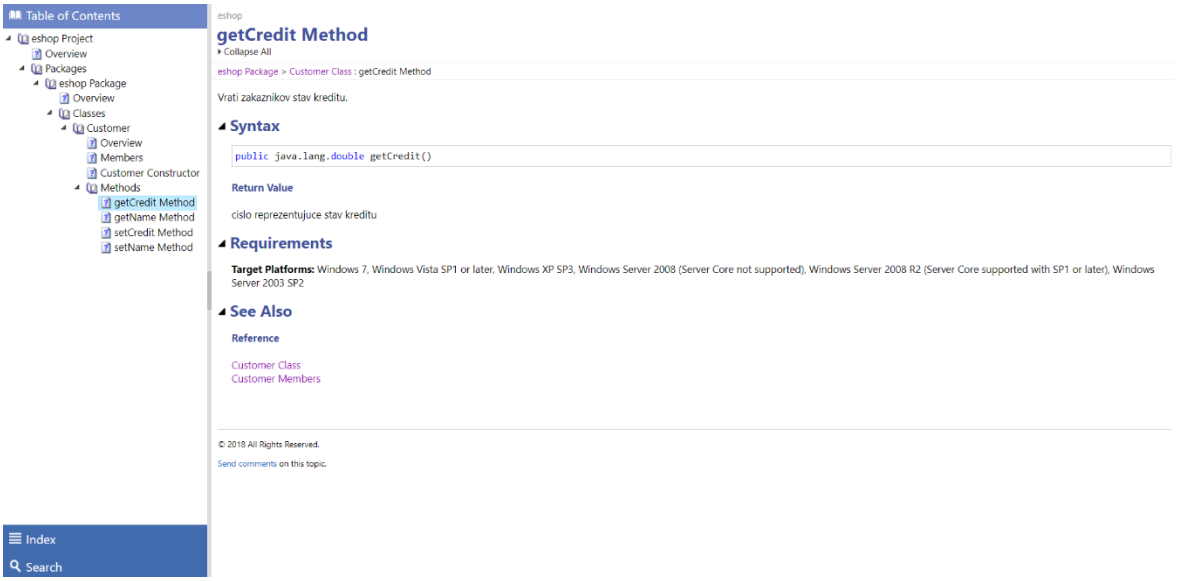
Obr. 5 Vygenerovaná CHM stránka pomocou nástroj Document! X.



Obr. 6 Vygenerovaná HTML stránka pomocou nástroja Document! X.



Obr. 7 Vygenerovaná HTML stránka pomocou nástroja Document! X.



Obr. 8 Vygenerovaná HTML stránka pomocou nástroja Document! X.

```
/** Vrati zakaznikov stav kreditu.
 * @return cislo reprezentujuce stav kreditu
 *
 */
public Double getCredit() {
    return credit;
}
```

Obr. 9 Ukážka zdrojového kódu.

3.2 Doxygen

Doxygen je veľmi rozšíreným a populárnym generátorom dokumentácie zo zdrojového kódu. Autorom Doxygenu je Dimitri van Heesh a prvú verziu tohto programu vydal v roku 1997. Doxygen má licenciu GNU, tým pádom ide o free software [21].

3.2.1 Vlastnosti nástroja Doxygen

Doxygen je multiplatformový program, môžeme ho spustiť na operačnom systéme Windows, OS X, Linux, BSD alebo Unix. Primárna podpora Doxygenu mala byť pre programovací jazyk C++, ale umožňuje tvoriť dokumentáciu aj pre jazyky C, C#, Java, IDL, VHDL, PHP, Fortran, Tcl, s pluginom pre VB a VB Script a čiastočne pre D a Python [21].

Výstupnými formátmi Doxygenu sú HTML stránky, CHM, LaTeX, man stránky, DocBook, XML alebo nepriamo PDF. Generovaný obsah má štyri časti – hlavnú stránku, stránku balíkov, tried a stránku súborov. Taktiež je k dispozícii textové okno pre vyhľadávanie [21].

Doxygen extrahuje dokumentáciu priamo zo zdrojových kódov, čo má vplyv na dobrú konzistenciu v súlade so zdrojovým kódom. Pre generovanie dokumentácie využíva samotnú štruktúry kódu, z ktorej vytvorí väzby medzi prvkami a zlepšuje sa tak rýchle hľadanie. Takého vytvorené väzby sú užitočné pri veľkých zdrojových distribúciách [21].

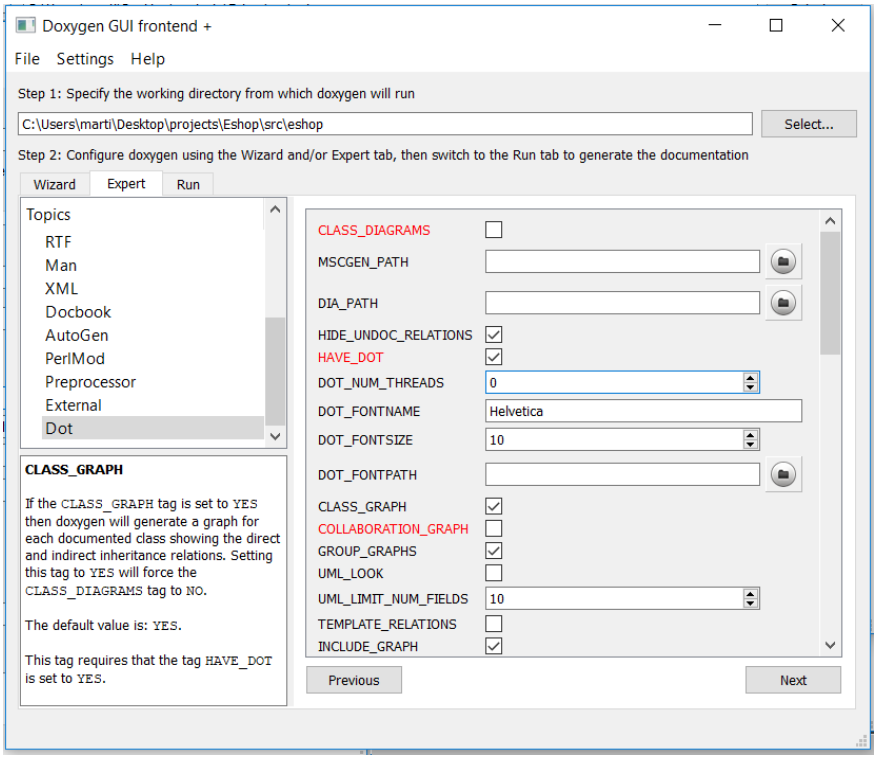
Pre vizualizáciu grafov a diagramov sa odporúča nainštalovať nástroj Graphviz. Tento nástroj vytvára volacie a volané grafy, grafy závislosti, dedičské diagramy alebo schémy väzieb medzi prvkami [21].

Doxygen ponúka široké spektrum editovateľnosti konfiguračného súboru, čo má za následok prispôsobenie dokumentácie podľa našich predstáv. Možnosti Doxygenu podčiarkuje aj fakt, že bol použitý pri tvorení jeho domovskej stránky a manuálových stránok. Taktiež umožňuje pridávanie referencií na rôzne časti dokumentácie, prípadne na dokumentáciu iného projektu. Tiež dovoľuje pridávať ukážky samotného kódu do dokumentácie a dokáže zvýrazňovať zdrojový kód rozličnými farbami podľa jeho štruktúry [21].

3.2.2 Práca s nástrojom Doxygen

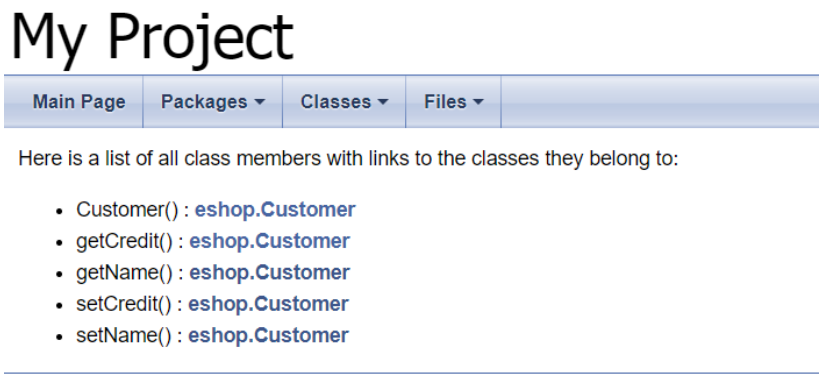
Inštalácia generátora Doxygen je veľmi jednoduchá. Na domovskej stránke si vyberieme verziu pre náš operačný systém. Počas inštalácie si môžeme zvoliť, či máme záujem o grafické rozhranie Doxywizard, manuálové stránky alebo vzorové príklady. Doxygen môžeme spúšťať pomocou príkazového riadku alebo s ním môžeme pracovať pomocou Doxywizardu.

V Doxywizarde (Obr. 10) si vyberieme zložky so zdrojovými súbormi. Taktiež tu máme možnosť editovať všetky nastavenia ohľadom projektu. Napríklad tu volíme programovací jazyk, výstupné formáty a ich úpravy, elementy, ktoré sa majú zobraziť a podobne.



Obr. 10 Práca s nástrojom Doxygen.

Vygenerovaná dokumentácia (Obr. 11, Obr. 12, Obr. 14) je hierarchicky rozdelená na štyri časti. Hlavná stránka obsahuje základný opis projektu, jeho verziu, čas vytvorenia a podobne. Stránka balíkov poskytuje zoznam balíkov a ich opis. Stránka tried zobrazuje všetky zdokumentované triedy, ich členské premenné a ich metódy. Posledná stránka súborov zobrazuje použité súbory. Na obrázku (Obr. 13) je znázornená časť zdrojového kódu použitého pre tvorbu dokumentácie (Obr. 12).



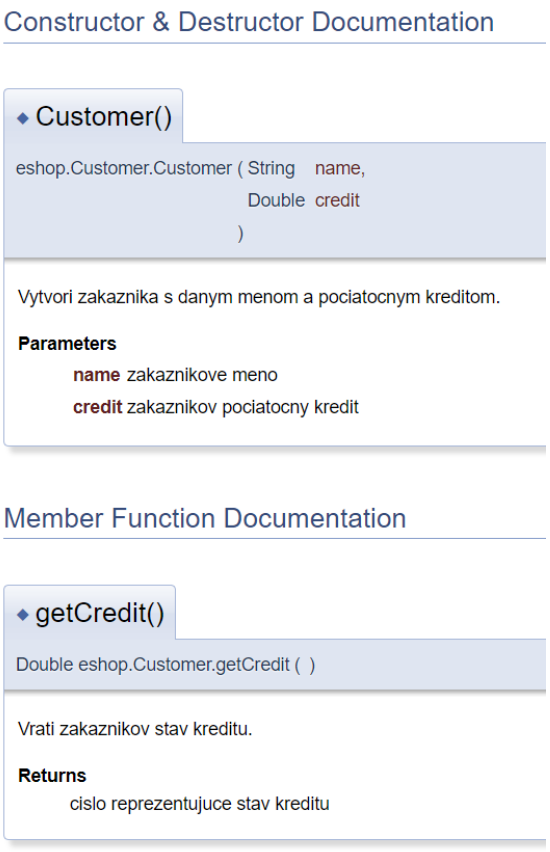
Obr. 11 Vygenerovaná HTML stránka pomocou nástroja Doxygen.

Obr. 12 Vygenerovaná HTML stránka pomocou nástroja Doxygen.

```
package eshop;

/** trieda reprezentujuca zakaznika.
 * @author Martin Feigl
 * @version 1.0
 *
*/
```

Obr. 13 Ukážka zdrojového kódu.



Obr. 14 Vygenerovaná HTML stránka pomocou nástroja Doxygen.

3.3 Javadoc

Pre generovanie dokumentácie zo zdrojového kódu programovacieho jazyka Java bol vytvorený nástroj Javadoc. Tento nástroj bol vytvorený spoločnosťou Sun Microsystems a jeho prvá verejná verzia vyšla v roku 1995. Javadoc je free softvér, na ktorý sa vzťahuje licencia GPL [26].

3.3.1 Vlastnosti nástroja Javadoc

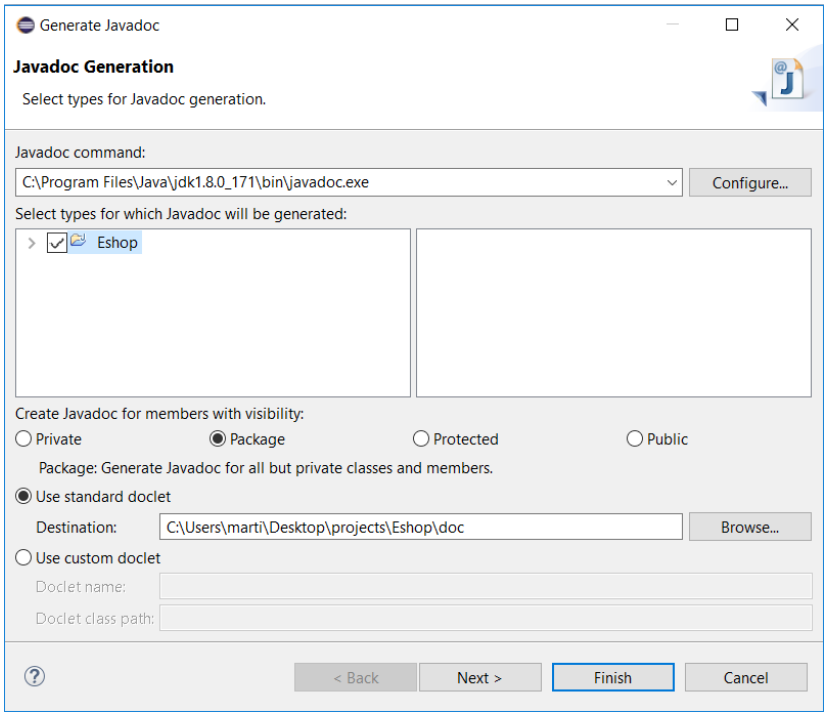
Javadoc je multiplatformový nástroj a spustíme ho na operačných systémoch ako sú Windows, OS X, Linux, BSD, Unix. Javadoc je vytvorený spoločnosťou, ktorá stojí za vznikom samotného programovacieho jazyka Java a je preto určený iba pre tento jediný konkrétny jazyk [26].

Javadoc môžeme rozdeliť na dve časti, Doclet a Taglet. Základným výstupným formátom pre Javadoc je HTML, avšak použitím Docletu môžeme získať aj formáty ako sú SGMl,

XML, RTF, MIF, LaTeX, man pages, DocBook, PDF, PostScript. Doclet okrem iného kontroluje komentáre a generuje správu obsahujúcu všetky chyby a nezrovnalosti, ktoré nájde. Taktiež dovoľuje vybrať, ktoré modifikátory prístupy (public, private, protected) majú byť zdokumentované. Doclet nám teda dovoľuje meniť výstupný formát, špecifikuje generovaný obsah a formát. Taglet je časť, ktorá nám umožňuje upravovať tagy, teda príkazy pre generátor. Okrem iného podporuje aj tvorbu vlastných tagov [26].

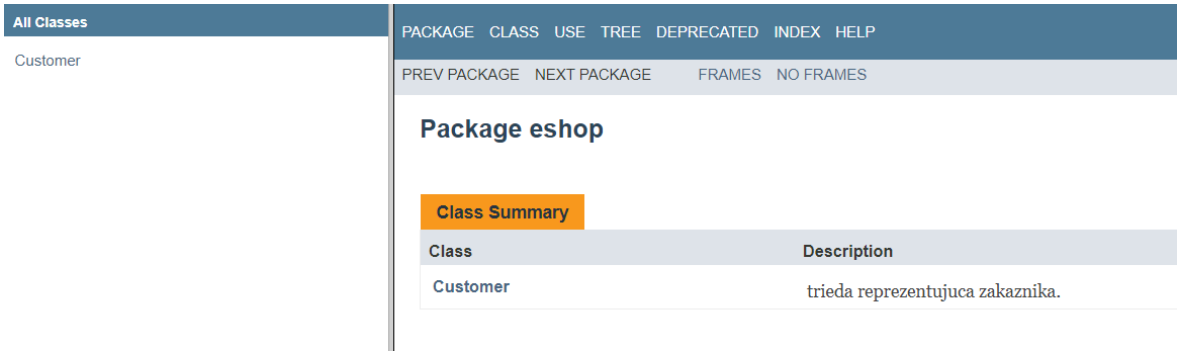
3.3.2 Práca s nástrojom Javadoc

Javadoc je obsiahnutý v balíku Java software development kit, ktorý je potrebný pre samotnú prácu a zostavovanie programov v jazyku Java. Preto ho nemusíme nijako samostatne sťahovať. Pre prácu s Javadocom si môžeme vybrať priamo nejaké integrované vývojové prostredie, napríklad Eclipse. V Eclipse si stačí len označiť súbory, ktoré chceme dokumentovať a spustiť Javadoc. Pre Javadoc (Obr. 15) určíme jeho umiestnenie a zvolíme, ktoré modifikátory prístupu majú byť zdokumentované.

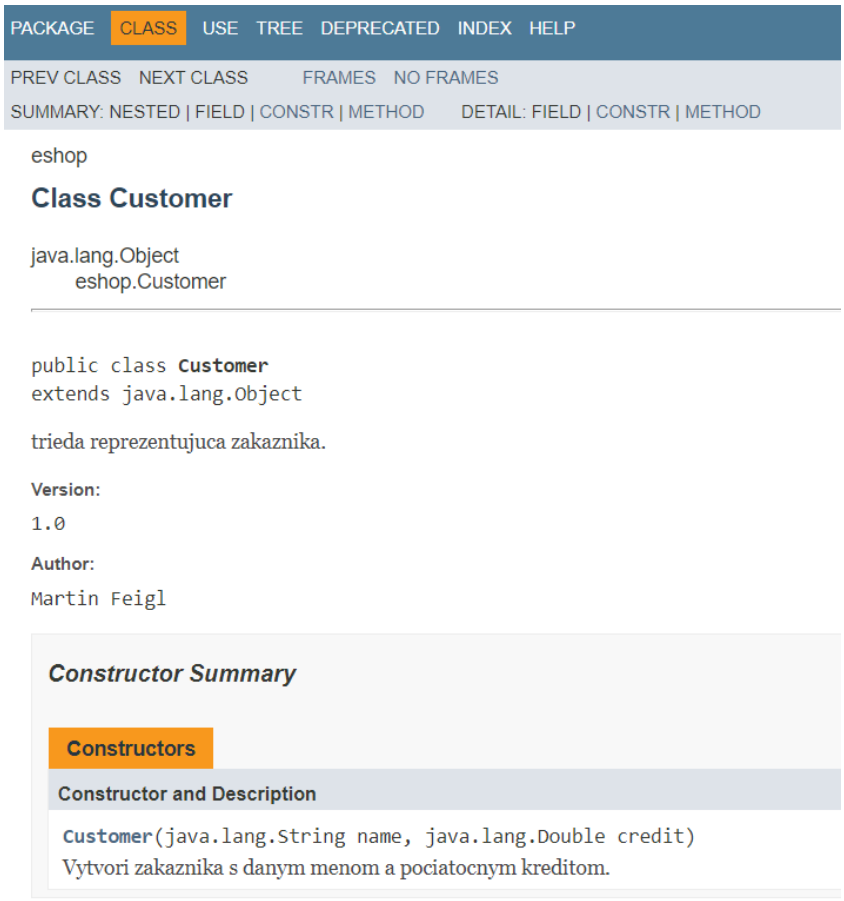


Obr. 15 Práca s nástrojom Javadoc.

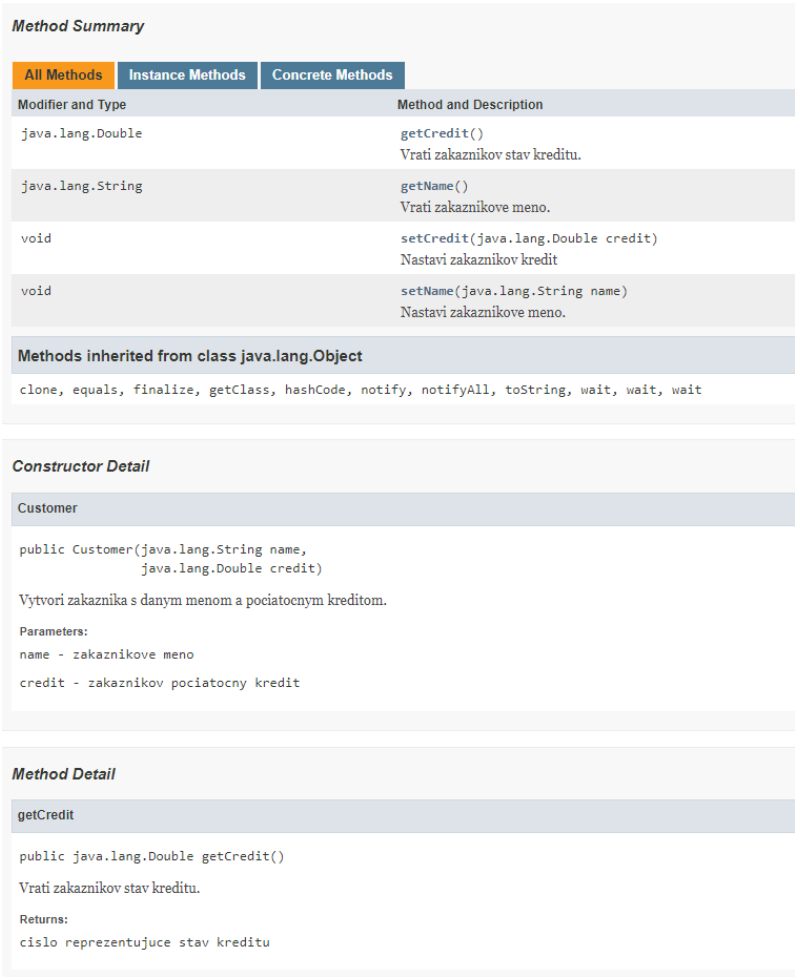
Vygenerovanú dokumentáciu si môžeme prezerať priamo v Eclipse alebo klasicky si otvoriť ako HTML stránku (Obr. 16, Obr. 17, Obr. 18). Bočný panel nám ponúka zobrazenie tried celého projektu alebo iba tried daného balíka. Ďalej tu môžeme vidieť menu, v ktorom sa dajú prezerať triedy, zoznam jej členov, metód a ich podrobný opis. Na obrázku (Obr. 19) je znázornená časť zdrojového kódu použitého pre tvorbu dokumentácie (Obr. 18).



Obr. 16 Vygenerovaná HTML stránka pomocou nástroja Javadoc.



Obr. 17 Vygenerovaná HTML stránka pomocou nástroja Javadoc.



Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
java.lang.Double	getCredit() Vrati zakaznikov stav kreditu.
java.lang.String	getName() Vrati zakaznikove meno.
void	setCredit(java.lang.Double credit) Nastavi zakaznikov kredit
void	setName(java.lang.String name) Nastavi zakaznikove meno.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Customer

```
public Customer(java.lang.String name,
                java.lang.Double credit)
```

Vytvori zakaznika s danym menom a pociatocnym kreditom.

Parameters:

name - zakaznikove meno

credit - zakaznikov pociatocny kredit

Method Detail

getCredit

```
public java.lang.Double getCredit()
```

Vrati zakaznikov stav kreditu.

Returns:

cislo reprezentujuce stav kreditu

Obr. 18 Vygenerovaná HTML stránka pomocou nástroja Javadoc.

```
/** Vytvori zakaznika s danym menom a pociatocnym kreditom.
 * @param name zakaznikove meno
 * @param credit zakaznikov pociatocny kredit
*/
public Customer(String name, Double credit) {
        this.name = name;
        this.credit = credit;
}
```

Obr. 19 Ukážka zdrojového kódu.

3.4 ROBODoc

ROBODoc je ďalším nástrojom pre vytváranie API dokumentácie. Za jeho vznikom stojí Frans Slothouber a jeho prvá verzia vznikla v roku 1995. ROBODoc z licenčného hľadiska patrí do skupiny GPL, teda môžeme hovoriť, že ide o free software [33].

3.4.1 Vlastnosti nástroja ROBODOc

ROBODoc je multiplatformový nástroj a spustíme ho napríklad na operačnom systéme Windows, OS X, Linux, BSD, Unix. Umožňuje prácu s mnohými programovacími jazykmi ako sú napríklad C, C++, Java, Assembler, Basic, Fortran, Tcl / Tk, LISP, Forth, Perl, Makefiles, Occam, COBOL, DCL, Visual Basic, PHP, Ruby, JavaScript, PL/SQL, Clarion, VB, VB Script, Delphi, Pascal, ADA, D, IDL a mnoho ďalších jazykov. Môže byť nastavený tak, aby pracoval s ľubovoľným jazykom, ktorý podporuje komentáre [33].

Ako výstupne formáty ponúka ROBODOc HTML, RTF, LaTeX, man pages, DocBook alebo nepriamo CHM, PDF a PostScript [33].

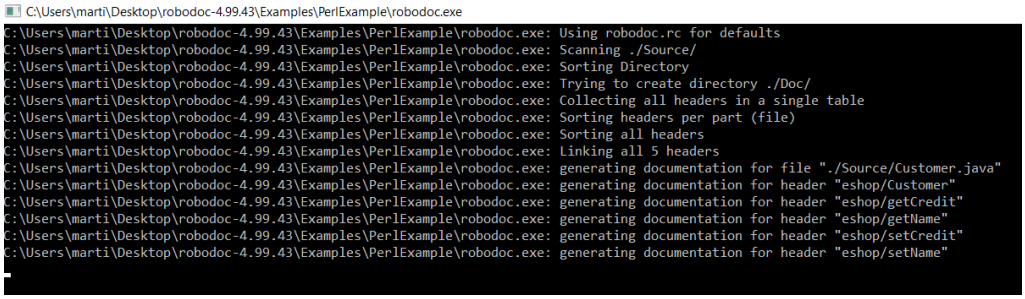
ROBODoc umožňuje kombinovať programovú dokumentáciu so zdrojovým kódom. Vyžaduje však, aby táto dokumentácia mala konkrétne rozloženie, aby mohla byť rozpoznaná. Existujú tri kľúčové pojmy: hlavičky (headers) , položky (items) a sekcie (sections). Tieto pojmy slúžia ako identifikátory pre generátor, sú to rozpoznané príkazy [33].

V konfiguračnom súbore môžeme definovať vlastné názvy týchto príkazov, ktoré príkazy sa majú ignorovať alebo vytvoriť preklady pre anglické názvy. Taktiež zmenou konfiguračného súboru sa môžeme vyhnúť problémom, ktoré by mohli vzniknúť, pokiaľ by pre nejaký programovací jazyk vznikal rozpor syntaxe s vnútornými značkami ROBODocu. Drobnou úpravou taktiež môžeme vkladať samotný kód do dokumentácie [33].

ROBODoc môže pracovať v troch režimoch – multidoc, singledoc a singlefile. V režime multidoc sú prezreté všetky zdrojové súbory v zdrojovom adresári a je vytvorený samostatný dokumentový súbor pre každý z nich. V režime singledoc sú prezreté všetky zdrojové súbory a je vytvorený jeden súbor dokumentácie, ktorý obsahuje všetku extrahovanú dokumentáciu. V režime singlefile je prezretý jeden zdrojový súbor a k tomu je vytvorený jeden súbor dokumentácie [33].

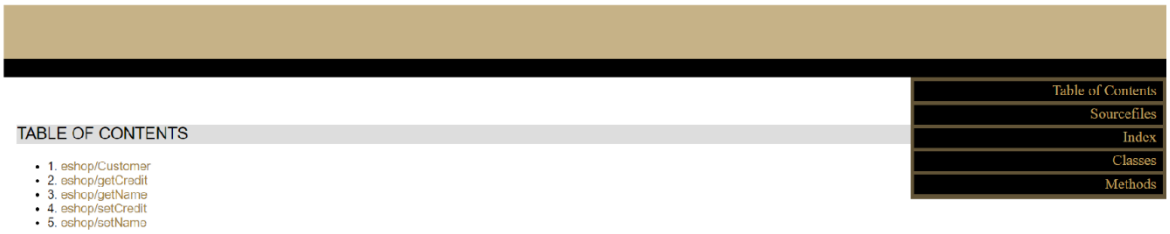
3.4.2 Práca s nástrojom ROBODoc

ROBODoc môžeme inštalovať pomocou príkazového riadku alebo pre operačný systém Windows môžeme priamo stiahnuť skompilovaný spustiteľný súbor. Tento súbor stačí vložiť do adresára so súbormi, z ktorých chceme extrahovať dokumentáciu a spustiť (Obr. 20).

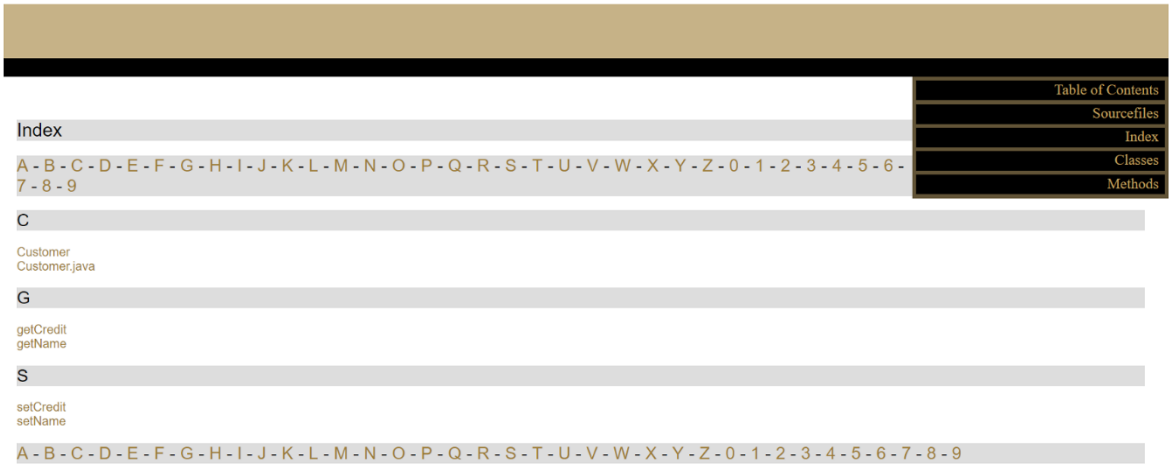


Obr. 20 Práca s nástrojom ROBODoc.

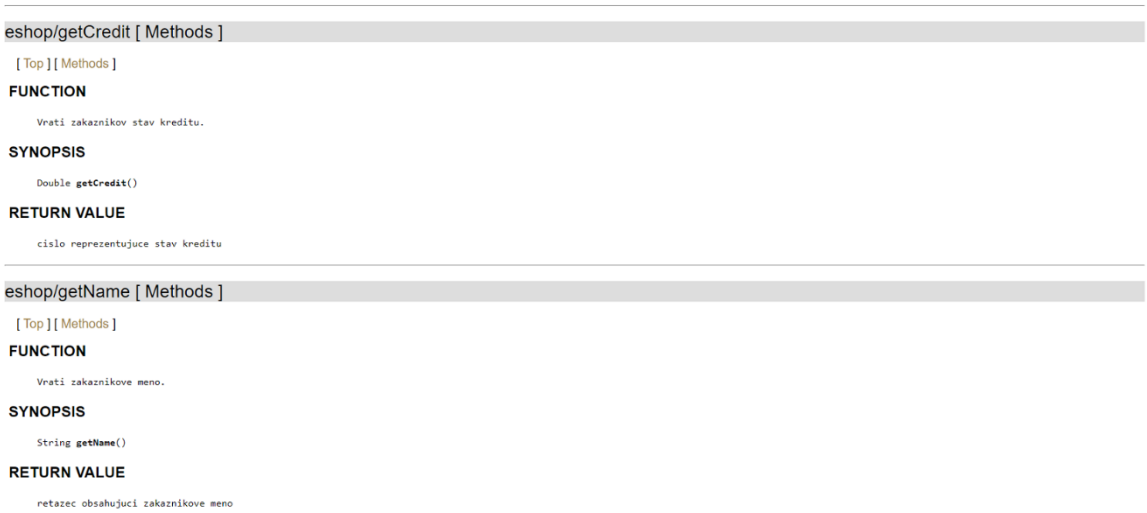
Po následnom spustení sa nám automaticky vytvorí priečinok s našou dokumentáciou. Po otvorení HTML stránky (Obr. 21, Obr. 22, Obr. 23) môžeme v menu vidieť tabuľku obsahu a indexu a taktiež záložky použitých hlavičiek a ďalších časti. Na obrázku (Obr. 24) je znázornený zdrojový kód použitý pre tvorbu dokumentácie (Obr. 23).



Obr. 21 Vygenerovaná HTML stránka pomocou nástroja ROBODoc.



Obr. 22 Vygenerovaná HTML stránka pomocou nástroja ROBODoc.



Obr. 23 Vygenerovaná HTML stránka pomocou nástroja ROBODoc.

```
/****m* eshop/setCredit
* FUNCTION
*   Nastavi zakaznikov kredit.
* SYNOPSIS
*   void setCredit(Double credit)
* INPUTS
*   nova hodnota zakaznikovho kreditu
*****
/** Nastavi zakaznikov kredit
 * @param nova hodnota zakaznikovho kreditu
*/
public void setCredit(Double credit) {
    this.credit = credit;
}
```

Obr. 24 Ukážka zdrojového kódu.

4 POROVNANIE A VÝBER VHODNÉHO GENERÁTORU

So všetkými vybratými generátormi dokumentácie zdrojového kódu sa pracovalo jednoducho a flexibilne. Pri práci ani pri inštalácii by nemali nastať žiadne nečakané komplikácie. Ku všetkým vybratým generátorom existuje dostačujúce množstvo dostupnej dokumentácie a manuálových stránok. Na domovských stránkach generátora Document! X môžeme nájsť aj vzorové videotutoriály pre všetky programovacie jazyky, ktoré tento generátor podporuje. Tieto videá sú veľmi nápomocné hneď z niekoľkých hľadísk. Poskytujú presné inštrukcie pre prácu s danými programovacími jazykmi a môžu byť skvelým pomocníkom ako pre pokročilých, tak pre úplných začiatočníkov. Môžu nám poskytnúť nové a dodatočné informácie, s ktorými sme sa do tej doby ešte nestreli.

Pokiaľ ide o dostupnosť z pohľadu licencií, tak všetky vybrané generátory sú zdarma dostupné softvéry až na generátor Document! X, ktorý je platený. Je dôležité vyzdvihnúť, že generátor Document! X ponúka bezplatnú skúšobnú trial verziu ktorá nám umožňuje sa s generátorom predovšetkým zoznámiť, vyskúšať základné funkcie a vlastnosti a následne učiniť rozhodnutie, či stojí za to investovať peniaze do kúpy tohto nástroja alebo nie. Ak sa napokon rozhodneme radšej pre zdarma dostupné softvéry, získame aspoň skúsenosti, ktoré sa nám môžu hodiť do budúcna.

Z pohľadu podpory operačných systémov, pre ktoré sú vybrané generátory dostupné, môžeme konštatovať, že všetky generátory, až na opäť výnimku Document! X, sú multiplatformové a spustiteľné na všetkých základných operačných systémoch ako sú Windows, OS X, Linux, Unix a ďalšie ich distribúcie. Zameranie sa nástroja Document! X na výhradne operačný systém Windows má predovšetkým výhodu v tom, že všetky dokumentačné šablóny, bez dodatočných fatálnych úprav, by mali splňovať všetky pravidlá ohľadom štandardov dokumentácie spoločnosti Microsoft. Druhou výhodou tohto zamerania je to, že Document! X môže byť priamo pridaný do integrovaného vývojového prostredia Microsoft Visual Studio, čo je jednoduchšie, efektívnejšie a rýchlejšie pre tvorbu dokumentácie.

Čo sa týka podpory programovacích jazykov, z ktorých môže byť dokumentácia tvorená, tak generátor Document! X a Doxygen ponúkajú podporu pre viaceré známe používané jazyky. JavaDoc je orientovaný iba pre jazyk Java. Toto obmedzenie sa nám môže zdať limitujúce, avšak má aj svoje výhody. Je vytvorený tvorcami Javy a táto špecializácia by mala napovedať, že najlepšia dokumentácia pre tento jazyk bude tvorená práve týmto

generátorom. Rovnako dokumentácia tvorená JavaDocom je dá sa povedať priemyselným štandardom pre dokumentáciu programov tvorených v Jave. Taktiež ako môžeme pridať Document! X do Microsoft Visual Studia, tak viaceré vývojové prostredia pre Javu, ako napríklad NetBeans, Eclipse a IntelliJ IDEA vedia automaticky generovať Javadoc HTML stránky. Toto spojenie generátora s vývojovým prostredím môže mať za výhodu lepšie možnosti náhľadu a udržateľnosti.

Presne opačným prípadom ako je Javadoc je ROBODoc. ROBODoc podporuje veľmi veľké množstvo rôznych programovacích jazykov. Je navrhnutý spôsobom, aby mohol byť rozšírený na podporu akéhokoľvek jazyka, ktorý má komentáre. Nemá znalosť syntaxe programovacích jazykov a má len určité vedomosti o tom, ako komentáre začínajú a končia vo viacerých programovacích jazykoch. Preto iné generátory podporujú iba niektoré jazyky. Iné generátory tieto znalosti používajú pre automatické zistenie viacerých informácii. To pri ROBODocu znamená, že musíme vynaložiť viac práce na tvorbu jednotlivých tagov, ktoré majú iný formát a zápis ako zvyšné vybraté generátory. Pri nástrojoch Document! X, Doxygen a Javadoc sme mohli používať ako vstup rovnaké komentáre a tagy, zatiaľ čo pri nástroji ROBODoc sa museli vykonávať zmeny v dokumentácii, bolo treba upraviť formát komentárov a tagov.

Všetky tieto opisované generátory poskytujú veľké množstvo tagov, teda príkazov pre generátor. Javadoc a ROBODoc umožňujú tvorbu aj vlastných tagov, čo má výhodu v prispôsobiteľnosti obsahu dokumentácie.

Výhodou nástrojov Doxygen, Javadoc a ROBODoc je schopnosť priamo alebo nepriamo spracovať dokumentáciu do viacerých výstupných formátov ako sú HTML, CHM, RTF, PDF, LaTeX, PostScript, man pages, Docbook alebo XML. Za to Document! X umožňuje prevedenie iba do formátu HTML a CHM.

Všetky spracované ukážky boli vo formáte HTML a môžeme vidieť, že všetky vybrané generátory sú schopné vytvoriť prehľadnú hierarchickú štruktúru. Môžeme konštatovať, že po estetickej vizuálnej stránke, prehľadnosťou a úrovňou detailov je na tom pravdepodobne najlepšia dokumentácia tvorená Doxygenom a Javadocom, a naopak najhoršie ROBODocom. Je to spravidla preto, lebo ROBODoc sa zameriava predovšetkým na všestrannosť a design preto nie je tak dôležitý. Document! X, Doxygen a Javadoc pridávajú okno pre vyhľadávanie a vedia pracovať s grafmi a diagramami, čo je ďalšie mínus pre

ROBODoc. Doxygen a Javadoc taktiež sú schopné upravovať výstup pomocou pridávania html a css značiek do komentárov dokumentácie.

Navyše ROBODoc predvolene vytvára predformátovaný text vo výstupnej dokumentácii pre celý text, ktorý nájde. To znamená, že formátovanie výstupu vyzerá rovnako ako formátovanie textu v komentároch. Teda napríklad zalomenie riadkov a odsadenie zostávajú rovnaké ako v komentároch, čo nie vždy vytvára najlepší výsledok.

Výber vhodného generátoru dokumentácie zdrojového kódu bude záležať v prvom rade na elementárnych veciach ako je napríklad podpora operačného systému, druh licencie, podpora programovacích jazykov a možnosť formátu výstupov. Okrem zmienených náležitostí samozrejme záleží na preferenciách daného užívateľa alebo firemných štandardov, a preto je vždy nutné brať v úvahu všetky okolnosti pri výbere toho správneho generátoru dokumentácie zdrojového kódu.

Zosumarizovaním všetkých testovaných štyroch nástrojov vychádza, že najvhodnejším generátorom dokumentácie zdrojového kódu pre väčšinu užívateľov je nástroj Doxygen. Nie nadarmo sa jedná o veľmi populárny, overený a rozšírený generátor, ktorý sa stále vylepšuje a na ktorom stále prebieha vývoj. Nástroj Doxygen skrýva široké spektrum výhod. Je multiplatformový, zdarma rozšíriteľný, čo môže byť rozhodujúce kritérium pre väčšinu užívateľov. Podporuje dostatok programovacích jazykov, výstupných formátov, má veľké množstvo tagov a širokú všeobecnú editovateľnosť. Ponúka konzolové aj grafické používateľské rozhranie. Jeho výstup má dobrú estetickú formu, ktorá sa dá upravovať, je detailný a prehľadný. Môžeme konštatovať, že jeho vážnou konkurenciou by bol Javadoc, ktorého hlavná nevýhoda tkvie v tom, že je obmedzený na jediný programovací jazyk.

ZÁVER

Cieľom tejto práce bolo priblížiť a vysvetliť čitateľom problematiku ohľadom dokumentácie zdrojových kódov programov. Dokumentácii zdrojového kódu sa väčšinou neprikladá veľká dôležitosť a nekladie sa jej dostatočná pozornosť. Ľudia často zabúdajú, že až správne zdokumentovaný softvér sa stáva kompletným. Dokumentácia má informatívny charakter a medzi jej výhody patrí prenositeľnosť kódu, jeho lepšia údržba a čitateľnosť. Dokumentácia by mala byť ľahko dostupná, aktuálna, stručná a vecne správna.

Základom práce bolo vytvoriť rešerš generátorov softvérových systémov, ktoré slúžia ku generovaniu dokumentácie na základe informácií v zdrojových kódoch. Pre dokumentáciu využívajú samotnú štruktúru programu a typicky využívajú informácie uvedené ako komentáre v špeciálnom formáte v zdrojovom kóde. Súčasťou práce sú ukážky použitia jednotlivých systémov.

Celkovým porovnaním vybratých generátorov sa dospelo k názoru, že najvhodnejším nástrojom pre tvorbu dokumentácie zdrojového kódu je Doxygen. Ide o rozšírený a populárny nástroj. Okrem dobrej podpory programovacích jazykov a výstupných formátov, medzi jeho hlavné výhody patrí profesionálna kvalita, prehľadnosť, jednoduchosť a široká škála prispôsobiteľnosti. Prvá verzia Doxygenu bola uvedená pred dvadsiatimi rokmi, ale stále na ňom prebieha aktívny vývoj a stále je nositeľom spoľahlivosti.

ZOZNAM POUŽITEJ LITERATÚRY

[1] KOTULA, Joffrey. *Source Code Documentation: An Engineering Deliverable. Proceedings of the Technology of Object-Oriented Languages and Systems*. Washington, IEEE Computer Society, 2000.

[2] Types of Software Documentation. *Tech Standards* [online]. [cit. 2018-05-15]. Dostupné z: http://www.techstandards.co.uk/types-of-software-documentation.html

[3] NEUMANN, Albrecht J. *Management Guide for Software Documentation* [online]. Washington, DC: National Institute of Standards and Technology, 1982 [cit. 2018-05-15]. Dostupné z: https://www.gpo.gov/fdsys/pkg/GOVPUB-C13-ddd47a2eca7877b72195f405e4a9fb8d/pdf/GOVPUB-C13-ddd47a2eca7877b72195f405e4a9fb8d.pdf

[4] PIERCE, R. *Software architecture in practice*. 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 978-032-1815-736.

[5] BASS, Len., Paul CLEMENTS a Rick. KAZMAN. *Optimizing Your Documentation with the Help of Technical Support*. 3rd ed. New York, N.Y: Association for Computing Machinery, 2003. ISBN 15-811-3696-X.

[6] Marketing. *Business Dictionary* [online]. [cit. 2018-05-15]. Dostupné z: http://www.businessdictionary.com/definition/marketing.html

[7] RIEVERA, T., A. TATE a S. WILL. *Deadly Sins of Technical Documentation*. Piscataway, N.J.: IEEE, 2004. ISBN 0-7803-8368-0.

[8] NESBIT, Wiliam F. Contracting with Technical Writers and Editors: Why, When, and How. *Asce Library* [online]. [cit. 2018-05-15]. Dostupné z: https://ascelibrary.org/doi/full/10.1061/%28ASCE%29LM.1943-5630.0000141?src=recsys&

[9] LEE, Marta F. a Brad MEHLENBACHER. *The Writer's Perspective,the Organizational Challenge. Reasearch Gate* [online]. 2000 [cit. 2018-05-15]. Dostupné z: https://www.researchgate.net/publication/2432399_The_Writer%27s_Perspective_the_Organizational_Challenge

[10] VAUGHAN-NICHOLS, Steven. *How to write documentation that's actually usefulge. Hewlet Packard Enterprise* [online]. 2017 [cit. 2018-05-15]. Dostupné z:

https://www.hpe.com/us/en/insights/articles/how-to-write-documentation-thats-actually-useful-1707.html

[11] CAMERLENGO., Terry a Andrew MONKHOUSE. *The Sun Certified Java Developer exam with J2SE 5*. 2nd ed. Berkeley, Calif: Apress, 2006. ISBN 14-302-0107-X.

[12] MCCONNELL, Steve. *Code complete*. 2nd ed. Redmond, Wash.: Microsoft Press, 2004. ISBN 07-356-1967-0.

[13] MARTIN, Robert C. *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009. ISBN 978-013-2350-884.

[14] KUNK, Joe. *To Comment or Not to Comment. Visual Studio Magazine* [online]. 2011 [cit. 2018-05-15]. Dostupné z: https://visualstudiomagazine.com/articles/2011/01/06/to-comment-or-not-to-comment.aspx

[15] *Comment Costs And Benefits* [online]. 2014 [cit. 2018-05-15]. Dostupné z: http://wiki.c2.com/?CommentCostsAndBenefits

[16] GAROUSI, Golara, Vahid GAROUSI-YUSIFOĞLU, Guenther RUHE, Junji ZHI, Mahmoud MOUSSAVI a Brian SMITH. *Usage and usefulness of technical software documentation: An industrial case study.Information and Software Technology*. 2015, ISSN 09505849.

[17] ZANONI, Julio Cezar, Milton Pires RAMOS, Cesar Augusto TACLA, Gilson Yukio SATO a Emerson Cabrera PARAISO. *A semi-automatic source code documentation method for small software development teams*. IEEE, 2011, ISBN 978-1-4577-0386-7.

[18] *Documentation Principles*. *Write The Docs* [online]. [cit. 2018-05-15]. Dostupné z: http://www.writethedocs.org/guide/writing/docs-principles/

[19] Comparison of programming languages (syntax). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-15]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_programming_languages_(syntax)#Commentsf

[20] STEIDL, Daniela, Benjamin HUMMEL a Elmar JUERGENS. *Quality Analysis of Source Code Comments* [online]. Garching b. Munchen, Germany: CQSE GmbHn,

2013, 2017 [cit. 2018-05-15]. Dostupné z: https://www.cqse.eu/publications/2013-quality-analysis-of-source-code-comments.pdf

[21] *Doxygen* [online]. Doxygen, ©2018 [cit. 2018-05-15]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/

[22] *Document! X* [online]. Innovasys Limited, ©2018 [cit. 2018-05-15]. Dostupné z: http://www.innovasys.com/product/dx/overview

[23] *D Programming Language* [online]. D Language Foundation, ©1999-2018 [cit. 2018-05-15]. Dostupné z: https://dlang.org/

[24] *Haddock: A Haskell Documentation Tool* [online]. Haddock, ©2018 [cit. 2018-05-15]. Dostupné z: https://www.haskell.org/haddock/

[25] Imagix [online]. Imagix Corp., ©2018 [cit. 2018-05-15]. Dostupné z: https://www.imagix.com/index.html

[26] *How to Write Doc Comments for the Javadoc Tool* [online]. Oracle, ©2018 [cit. 2018-05-15]. Dostupné z: http://www.oracle.com/technetwork/articles/java/index-137868.html

[27] *JSDoc* [online]. JSDoc, ©2011-2017 [cit. 2018-05-15]. Dostupné z: http://usejsdoc.org/

[28] *Natural Docs: Readable Source Code Documentation for 20 Programming Languages* [online]. Code Clear LLC., ©2018 [cit. 2018-05-15]. Dostupné z: http://www.naturaldocs.org/

[29] *Perl Programming Documentation* [online]. Perl 5 Porters, ©2018 [cit. 2018-05-15]. Dostupné z: https://perldoc.perl.org/index.html

[30] *PhpDocumentor* [online]. PhpDocumentor, ©2018 [cit. 2018-05-15]. Dostupné z: https://www.phpdoc.org/

[31] Python [online]. Python Software Foundation, ©2001-2018 [cit. 2018-05-15]. Dostupné z: https://www.python.org/doc/

[32] *Ruby programming language* [online]. Ruby-Doc, ©2018 [cit. 2018-05-15]. Dostupné z: http://ruby-doc.org/

[33] *ROBODoc: automating the software documentation process* [online]. Frans Slothouber, ©2010 [cit. 2018-05-15]. Dostupné z: https://rfsber.home.xs4all.nl/Robo/robodoc.html

[34] *Spinx: Python Documentation Generator* [online]. Georg Brandl and the Sphinx team, ©2007-2018 [cit. 2018-05-15]. Dostupné z: http://www.sphinx-doc.org/en/master/

[35] *Visual Expert: Explore, Analyze & Document your Code* [online]. Novalys, ©2015 [cit. 2018-05-15]. Dostupné z: http://www.visual-expert.com/EN/

[36] *VSdocman* [online]. Helixoft, ©2018 [cit. 2018-05-15]. Dostupné z: https://www.helixoft.com/

[37] *Yard: A Ruby Documentation Tool* [online]. Loren Segal, ©2007-2016 [cit. 2018-05-15]. Dostupné z: https://yardoc.org/

[38] Comparison of documentation generators In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-15]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_documentation_generators

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

BASIC	Beginner's All-purpose Symbolic Instruction Code
APL	A Programming Language
MATLAB	Matrix laboratory
PHP	Hypertext Preprocessor
SASS	Syntactically awesome style sheets
VB	Visual Basic
SQL	Structured Query Language
VHDL	VHSIC Hardware Description Language
SGML	Standard Generalized Markup Language
COBOL	Common Business Oriented Language
PAW	Physics Analysis Workstation
ABAP	Advanced Business Application Programming
ALGOL	Algorithmic Language
PL/I	Programming Language One
CSS	Cascading Style Sheets
BiDi	Bidirectional
Ps	Punctuation Open
Pe	Punctuation Close
Pi	Punctuation Initial quote
Pf	Punctuation Final quote
ML	Meta Language
OCaml	Objective Caml
CASE	Computer aided software engineering
PDF	Portable Document Format
LaTeX	Lamport TeX

OS X	Macintosh Operating System X
BCD	Berkeley Software Distribution
IDL	Interactive Data Language
PL	Procedural Language
HTML	Hypertext Markup Language
CHM	Microsoft Compiled HTML Help
XML	Extensible Markup Language
COM	Component Object Model
SOAP	Simple Object Access Protocol
REST	Representational state transfer
OLE DB	Object Linking and Embedding Database
GNU	General Public License
RTF	Rich Text Format
MIF	Maker Interchange Format
API	Application Programming Interface
LISP	List Processing Language
DCL	DIGITAL Command Language

ZOZNAM OBRÁZKOV

Obr. 1 Získavanie potrebných informácií [16]. 20
Obr. 2 Práca s nástrojom Document! X. 35
Obr. 3 Práca s nástrojom Document! X. 36
Obr. 4 Práca s nástrojom Document! X. 36
Obr. 5 Vygenerovaná CHM stránka pomocou nástroj Document! X. 37
Obr. 6 Vygenerovaná HTML stránka pomocou nástroja Document! X. 38
Obr. 7 Vygenerovaná HTML stránka pomocou nástroja Document! X. 38
Obr. 8 Vygenerovaná HTML stránka pomocou nástroja Document! X. 39
Obr. 9 Ukážka zdrojového kódu. 39
Obr. 10 Práca s nástrojom Doxygen. 41
Obr. 11 Vygenerovaná HTML stránka pomocou nástroja Doxygen. 41
Obr. 12 Vygenerovaná HTML stránka pomocou nástroja Doxygen. 42
Obr. 13 Ukážka zdrojového kódu. 42
Obr. 14 Vygenerovaná HTML stránka pomocou nástroja Doxygen. 43
Obr. 15 Práca s nástrojom Javadoc. 44
Obr. 16 Vygenerovaná HTML stránka pomocou nástroja Javadoc. 45
Obr. 17 Vygenerovaná HTML stránka pomocou nástroja Javadoc. 45
Obr. 18 Vygenerovaná HTML stránka pomocou nástroja Javadoc. 46
Obr. 19 Ukážka zdrojového kódu. 46
Obr. 20 Práca s nástrojom ROBODoc. 48
Obr. 21 Vygenerovaná HTML stránka pomocou nástroja ROBODoc. 48
Obr. 22 Vygenerovaná HTML stránka pomocou nástroja ROBODoc. 49
Obr. 23 Vygenerovaná HTML stránka pomocou nástroja ROBODoc. 49
Obr. 24 Ukážka zdrojového kódu. 49

ZOZNAM TABULIEK

Tab. 1 Začiatočné značky pre jednoriadkové komentáre [19].....24
Tab. 2 Párové značky pre blokové komentáre [19]. 25
Tab. 3 Základné druhy tagov [21]. 28
Tab. 4 Podpora operačných systémov [38]. 29
Tab. 5 Podpora programovacích jazykov [38]. 29
Tab. 6 Podpora programovacích jazykov [38]. 30
Tab. 7 Podpora programovacích jazykov [38]. 31
Tab. 8 Podpora výstupných formátov [38]. 32

ZOZNAM PRÍLOH

Práca neobsahuje prílohy.