
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# MULTIPLATFORMNÍ NÁSTROJ PRO GENEROVÁNÍ TECHNICKÉ DOKUMENTACE Z XML

**MULTI-PLATFORM TOOL FOR GENERATION OF TECHNICAL DOCUMENTATION FROM XML**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** — **SIMONA JÁNOŠÍKOVÁ**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. JOSEF STRNADEL, Ph.D.**
**SUPERVISOR**



**BRNO 2024**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Zadání bakalářské práce

156520

| | |
|---|---|
| Ústav: | Ústav počítačových systémů (UPSY) |
| Studentka: | **Jánošíková Simona** |
| Program: | Informační technologie |
| Název: | **Multiplatformní nástroj pro generování technické dokumentace z XML** |
| Kategorie: | Algoritmy a datové struktury |
| Akademický rok: | 2023/24 |

Zadání:

1. Proveďte rešerši v oblasti ukládání informací ve formátu XML pro různé případy užití; zaměřte se zejména na využití XML pro uložení projektů vývojových prostředí.
2. Proveďte rozbor požadavků kladených na technickou dokumentaci; identifikujte třídu požadavků, jejichž splnění je možno automatizovat a předpoklady pro tuto automatizaci.
3. Proveďte rešerši metod, nástrojů, technologií a přístupů v oblasti automatizovaného generování technické dokumentace ze zdrojových souborů, zejména pak z XML.
4. Navrhněte multiplatformní nástroj, který bude schopen maximálně automatizovaně generovat čitelnou technickou dokumentaci z XML vstupu, případně doplněného o uživatelskou anotaci, konfiguraci apod. před spuštěním generování.
5. Navržený nástroj implementujte, proveďte jeho základní testování a připravte podmínky pro jeho zhodnocení.
6. Na základě uživatelského, popř. dalšího, testování zhodnoťte vlastnosti implementovaného nástroje, především co se týká jeho schopnosti generovat čitelnou dokumentaci z XML (přinejmenším předpokládejte SVD vstup, XML vstup pro popis projektu nástroje UPPAAL a jeden vlastní vstup) a z případných doplňujících informací dodaných uživatelem nástroje.
7. Zhodnoťte vlastnosti implementovaného nástroje a výsledky pomocí něj dosažené; identifikujte silné a slabé stránky implementace, navrhněte možné směry jejího vylepšení a rozveďte ty, které považujete za nejvíce perspektivní.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

| | |
|---|---|
| Vedoucí práce: | **Strnadel Josef, Ing., Ph.D.** |
| Vedoucí ústavu: | Sekanina Lukáš, prof. Ing., Ph.D. |
| Datum zadání: | 1.11.2023 |
| Termín pro odevzdání: | 9.5.2024 |
| Datum schválení: | 30.10.2023 |

# Abstrakt

Táto bakalárska práca sa zaoberá výzvou v oblasti technickej dokumentácie vývojových prostredí a preskúmava možnosti automatizovanej generácie dokumentácie zo štruktúrovaných XML dát. Problém spočíva v potrebe efektívnej tvorby a udržiavania technickej dokumentácie projektov prostredníctvom automatizovaného procesu. Cieľom práce je navrhnúť, implementovať a zhodnotiť multiplatformný nástroj, ktorý bude schopný z XML vstupov generovať technickú dokumentáciu s dôrazom na praktickú použiteľnosť v reálnych projektoch. Práca je užitočná pre vývojárov, ktorí sa zaoberajú tvorbou a udržiavaním technickej dokumentácie vývojových prostredí. Poskytuje konkrétny nástroj, ktorý môže zefektívniť proces generovania dokumentácie a zlepšiť čitateľnosť dokumentácie v rámci projektov využívajúcich XML formát pre uloženie informácií.



# Abstract

This bachelor's thesis deals with the challenge in the field of technical documentation of development environments and explores the possibilities of automated generation of documentation from structured XML data. The problem lies in the need for efficient creation and maintenance of technical project documentation through an automated process. The goal of the work is to design, implement and evaluate a multi-platform tool that will be able to generate technical documentation from XML inputs with an emphasis on practical usability in real projects. The work is useful for developers who are involved in the creation and maintenance of technical documentation of development environments. It provides a specific tool that can streamline the documentation generation process and improve the readability of documentation within projects using the XML format for storing information.



# Kľúčové slová

XML, technická dokumentácia, Python, automatizované generovanie, nástroj, PDF, Latex, Markdown, UPPAAL, CMSIS-SVD

# Keywords

XML, technical documentation, Python, automated generation, tool, PDF, Latex, Markdown, UPPAAL, CMSIS-SVD


# Citácia

JÁNOŠÍKOVÁ, Simona. *Multiplatformní nástroj pro generování technické dokumentace z XML*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

# Multiplatformní nástroj pro generování technické dokumentace z XML

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Josefa Strnadela, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Simona Jánošíková
8. mája 2024


## Poďakovanie

Rada by som sa poďakovala vedúcemu práce, pánovi Ing. Josefovi Strnadelovi, Ph.D., za jeho pomoc, odborné rady, užitočné nápady a trpezlivosť pri riešení tejto práce. Rovnako by som sa chcela poďakovať dobrovoľníkom za pomoc pri testovaní výsledného nástroja. V neposlednom rade by som sa chcela poďakovať svojej rodine za podporu a trpezlivosť pri mojom štúdiu.

# Obsah

# Kapitola 1

# Úvod

Bakalárska práca, predstavujúca sa v nasledujúcich stranách, sa venuje tematike automatizovaného generovania technickej dokumentácie z XML dát v prostredí vývojových platforiem. V dnešnej dobe, kedy sa softvérové inžinierstvo vyvíja rýchlym tempom, je nevyhnutné zdôrazňovať potrebu efektívnej tvorby a udržiavania technickej dokumentácie. Vývojári a inžinieri sa denne stretávajú s výzvou správy a aktualizácie dokumentácie v rámci komplexných projektov, a preto je vhodné hľadať nové prístupy, ktoré im uľahčia túto náročnú úlohu.

Vzhľadom na rýchly vývoj technológií je nutné smerovať k automatizácii generovania technickej dokumentácie, čo je poháňané aj rastom komplexnosti softvérových projektov a narastajúcimi požiadavkami na kvalitnú dokumentáciu. Súčasné riešenia môžu byť časovo náročné, a niekedy náchylné na chyby, čo zdôrazňuje potrebu inovatívnych prístupov a nástrojov. Motiváciou tejto práce je vytvorenie nástroja, ktorý nielen ponúka rýchle a spoľahlivé riešenie pre generovanie dokumentácie z XML štruktúr, ale aj minimalizuje námahu vývojárov a znižuje riziko chýb spojených s manuálnou tvorbou dokumentácie.

Cieľom tejto práce je poskytnúť detailný pohľad na problémy spojené s generovaním technickej dokumentácie z XML dát a predstaviť nový nástroj, ktorý efektívne zvláda túto úlohu. Toto riešenie by malo zlepšiť čitateľnosť technickej dokumentácie, ale aj vytvoriť efektívny nástroj, ktorý šetrí čas vývojárom a redukuje riziko chýb spojených s manuálnou tvorbou dokumentácie.

Táto práca sa vyznačuje novým prístupom k automatizácií generovania technickej dokumentácie z XML, zohľadňujúc špecifiká vývojových prostredí. Práca je štrukturovaná do niekoľkých hlavných častí. Po úvode nasleduje Rozbor riešenej problematiky a realizačných možností, kde sa nachádza teoretický úvod do jazyka XML, a je popísaná analýza existujúcich problémov a prístupov k automatizácii generovania technickej dokumentácie. Ďalej je popísaný návrh konkrétneho postupu riešenia, ktorý podrobne charakterizuje navrhovaný multiplatformný nástroj. Realizácia a Testovanie sú ďalšie kapitoly, ktoré približujú spôsob riešenia nástroja a hodnotia jeho úspešnosť a prínos. Správa končí záverom, kde sú zhrnuté hlavné myšlienky a odporúčania.

# Kapitola 2

# Rozbor riešenej problematiky a realizačných možností

## 2.1 XML

### 2.1.1 Úvod do problematiky XML

Extensible Markup Language, ďalej XML, je typ značkovacieho jazyka a zároveň aj formátu súborov navrhnutý na ukladanie, prenos, rekonštrukciu, či prácu s rôznym druhom dátových údajov. Tento jazyk určuje pravidlá pre kódovanie dokumentov, ktoré sú zrozumiteľné pre počítače, ale zároveň sú pochopiteľné aj pre ľudí. Primárnym zámerom tohto jazyka je poskytnúť jednoduchý, ale na druhej strane rozšíriteľný spôsob na organizovanie a prenos dát medzi rôznymi systémami, aplikáciami a internetovými platformami.

Oficiálna dokumentácia XML[18] popisuje XML dokumenty takto: *Dokumenty XML sa skladajú z úložných jednotiek nazývaných entity, ktoré obsahujú buď analyzované alebo neanalyzované údaje. Analyzované údaje sa skladajú zo znakov, z ktorých niektoré tvoria znakové údaje a niektoré tvoria značky. Značky slúžia na kódovanie popisu rozloženia úložiska dokumentu a logickej štruktúry. XML poskytuje mechanizmus na stanovenie obmedzení týkajúcich sa rozmiestnenia údajov a logickej štruktúry dokumentu.*

História XML sa začala v roku 1996, kedy pracovná skupina XML Working Group pod záštitom World Wide Web Consoritium (W3C) začala pracovať na odľahčenej verzií SGML, Standard Generalized Markup Language. Na čele tejto výskumnej skupiny stál Jon Bosak a ďalej členovia Tim Bray, C. M. Sperberg-McQueen, James Clark a mnohí ďalší. Ako výsledok tejto práce bol vo februári roku 1998 predstavený jazyk XML 1.0 [17] a vďaka svojej flexibilite a rozsiahlemu použitiu zožal okamžitý úspech. V roku 2006 bola špecifikovaná najnovšia verzia XML 1.1 (2nd edition) [18] ktorá sa používa dodnes spolu s XML 1.0 (5th edition) vydanou v roku 2008.

V oblasti informatiky má XML význam v niekoľkých dôležitých aspektoch. Často je využívaný na proces výmeny údajov medzi systémami a aplikáciami, predovšetkým v sieťových službách alebo v prostrediach s distribuovanou architektúrou. Taktiež sa tento jazyk používa na ukladanie konfiguračných informácií v aplikáciach a iných rôznych systémoch, čo umožňuje jednoduché rozšírenie a úpravu konfigurácie. Aj webové služby ako SOAP a REST využívajú XML ako svoj základ. Tieto webové služby umožňujú komunikáciu medzi aplikáciami prostredníctvom štandardizovaných protokolov. Okrem toho sa dá XML použiť ako dokumentácia štruktúry a obsahu údajov, čo uľahčuje porozumenie a správu databáz a dátových súborov. Takto sa XML stáva neoddeliteľnou súčasťou moderných in-

formačných technológií a prispieva k efektívnejšej správe a výmene údajov v počítačových systémoch.

### 2.1.2 Dôvody pre používanie XML vývojových prostredí

Dôvody, prečo sa XML často používa vo vývojových prostrediach sú rôzne a odzrkadľujú jeho flexibilitu a schopnosť efektívne manipulovať a spracovávať údaje medzi systémami a aplikáciami. Informácie v tejto sekcií sú prevzaté z [7] a [6].

Vďaka svojmu formátu poskytuje XML štrukturované ukladanie dát pomocou značiek a elementov. Vďaka tejto hierarchickej štruktúre je umožnené vývojárom organizovať dáta do logických skupín a definovať ich vzájomné vzťahy medzi nimi. Napríklad XML súbor môže mať hierarchiu, kde jeden element obsahuje ďalšie elementy, ktoré popisujú detaily o rôznych aspektoch údajov.

XML je taktiež veľmi flexibilný formát, v ktorom je umožnené definovať štruktúru podľa potreby. To znamená, že je možné definovať vlastné značky, ľubovoľne ich pomenovávať, a tým pádom vieme vytvoriť štruktúru prispôsobenú našim potrebám. Táto flexibilita zohráva hlavnú úlohu pri adaptácií na rôzne typy dát a aplikácií. Zároveň je XML veľmi ľahko modifikovateľné, a preto ak nastane zmena v požiadavkách na dáta, je veľmi ľahké upraviť XML štruktúru a nie je potreba zmeny v celej aplikácií.

Použitie XML je jednoduché v rôznych vývojových prostrediach, pretože je nezávislý na platforme a programovacom jazyku. Práve pre to, sa XML používa ako prostriedok na výmenu dát medzi rôznymi vývojovými prostrediami bez ohľadu na použité technológie. Táto schopnosť interoperability je kľúčová, najmä pri integrácii rôznych systémov a služieb.

Vývojári dokážu XML veľmi ľahko integrovať do svojich projektov, pretože je XML podporovaný mnohými štandardami a technológiami. Veľmi často sa využíva napríklad XML Schema, ktorá poskytuje formálny popis štruktúry XML dokumentu, a tiež je poskytnutá validácia vstupých dát. Ďalej je veľmi znám XSLT(Extensible Stylesheet Language Transformations), vďaka ktorému je možné XML transformovať do rôznych formátov, čo padne na úžitok pri prezentácií dát v rôznych prostrediach.

Úzke prepojenie XML s modernými technológiami otvára široké možnosti pre vývojárov v rôznej škále aplikácií. Napríklad SOAP(Simple Object Access Protokol) je protokol pre výmenu štrukturovaných dát v distribuovaných prostrediach a ako formát reprezentácie dátových správ využíva XML. Vďaka tejto integrácií je umožnená komunikácia prostredníctvom štandardizovaných XML správ medzi rôznymi platformami. Ďalej sa tiež používa XHTML, čo je varianta klasického HTML založená na XML, ktorá ponúka prísnejšiu syntax pre tvorbu webových stránok. XML je taktiež základom pre SVG (Scalable Vector Graphics) pomocou ktorého sa vytvárajú grafické prvky pre web stránky.

V neposlednom rade sa XML veľmi často používa pre dokumentáciu dátových štruktúr a formátov v rámci vývojových projektov. Vývojári môžu popísať štruktúru údajov a ich vzťahy pomocou XML, čo výrazne uľahčuje porozumenie a správu dát v rámci projektu. Dokumentácia v XML dokáže slúžiť ako užitočný zdroj informácií pre celý tím vývojárov projektu a vo veľkej miere pomáha znížiť chybovosť, a aj zvýšiť efektivitu vývoja.

Vcelku sa dá povedať, že XML je populárnym nástrojom medzi vývojármi vďaka možnosti štruktúrovať, schopnosti ukladať a vymieňať údaje medzi rôznymi systémami a aplikáciami. Tiež netreba zabudnúť na flexibilitu, interoperabilitu a podporu moderných technológií, čo robí XML populárnym a nevyhnutným nástrojom pre veľké množstvo vývojových prostredí a aplikácií.

### 2.1.3 Zásady XML štruktúry

Pri písaní XML dokumentov je veľkým základom porozumieť prvkom ktoré tvoria jeho štruktúru. Informácie o XML štruktúre boli prevzaté [8], [18] a [7].

XML dokument je vlastne reťazec znakov, v ktorom sa môže nachádzať každý legálny Unicode znak okrem znaku `null`. *Každý dokument XML obsahuje jeden alebo viac elementov, ktorých hranice sú buď ohraničené počiatočnými a koncovými značkami, alebo v prípade prázdnych elementov značkou prázdnych elementov. Každý element má typ, identifikovaný názvom, niekedy nazývaný jeho „generický identifikátor, a môže mať súbor špecifikácií atribútu. Každá špecifikácia atribútu má názov a hodnotu*, preložené z [18]. Vzorový príklad XML dokumentu je vo výpise 2.1

#### Znaky a kódovanie v XML

Kódovanie XML dokumentu môže byť rôzne, ale najčastejšie sa používa UTF-8 alebo UTF-16. Taktiež môžu byť požité aj iné kódovania, ako napr.: ISO-8859-1 alebo ISO-8859-2.

V XML sa môže vyskytovať každý legálny znak spĺňajúci špecifikáciu ISO/IEC 10646. Avšak, ak je potreba napísať niektoré znaky ako ‘`&`‘, ‘`<`‘, ‘`>`‘, ‘`"`‘ a ‘`’`‘ v texte elementu XML, tak musia byť zapísané formou oddeľovacích značiek ‘`&`‘ pre ‘`&`‘, ‘`<`‘ pre ‘`<`‘, ‘`>`‘ pre ‘`>`‘, ‘`&quot;`‘ pre ‘`"`‘ a ‘`&apos;`‘ pre ‘`’`‘.

XML dokument je case-sensitive, teda citlivý na veľké a malé písmená. To znamená, že element ‘`<Person>`‘ sa líši od elementu ‘`<PERSON>`‘ a tiež od ‘`<person>`‘. Toto pravidlo platí pre všetky názvy elementov, atribútov a hodnôt atribútov XML. Toto pravidlo taktiež núti pisateľa XML písať konzistentne a dodržiavať jednotný štýl dokumentu.

#### Elementy a značky

Element je logickým prvkom XML dokumentu a jeho základom je otváracia a zatváracia značka, alebo pozostáva iba zo značiek označujúce prázdny prvok. Otváracia značka musí byť na začiatku každého neprázdneho XML elementu a obsahuje názov elementu. Zatváracia značka označuje koniec elementu a tiež obsahuje názov elementu totožný s otváracou značkou. Každý element čo začína otváracou značkou musí byť spravidla ukončený zatváracou značkou. Znaky, ktoré sa nachádzajú medzi začiatočnou a koncovou značkou sa nazývajú obsah elementu. Obsah elementu môže tvoriť ľubovoľná sekvencia Unicode znakov, ale aj značky iných elementov, ktorá sa nazývajú podradené elementy. Príklad XML elementu: ‘`<person> </person>`‘.

#### Atribút

Atribút je pár názov-dvojica, napr. `"name="value"`, ktorý sa môže pridať do vnútra otváracej značky za názov elementu. Každý atribút môže mať iba jednu hodnotu, a v rámci jedného konkrétneho elementu môže byť špecifikovaný maximálne jedenkrát. Používajú sa na poskytnutie dodatočných informácií o elemente alebo na spresnenie jeho parametrov. Atribúty sú voliteľným prvkom XML štruktúry, ale podľa vlastnej definície XML štruktúry môžu byť aj povinné. Názvy atribútov musia byť platné XML identifikátory, teda môžu začínať iba písmenom alebo podčiarovníkom, a ďalej obsahovať iba písmená, číslice alebo podčiarovníky. Hodnota atribútu môže nadobudnúť hocijakú hodnotu bez ohľadu na typ znakov a je ohraničená hornými úvodzovkami. Príklad XML elementu: ‘`<person name="Johnäge="30» </person>`‘.

### XML deklarácia

Začiatok každého XML dokumentu by mal tvoriť prolog, alebo aj XML deklarácia. Je to voliteľný prvok, ktorý sa musí nachádzať výlučne na prvom riadku dokumentu. Poskytuje konkrétne informácie o verzií XML, kódovaní, a tým napomáha k lepšiemu spracovaniu a interpretácií XML dokumentu aplikáciami. Deklarácia XML má špecifický formát a je to špeciálna značka `<?xml ?>`. Do vnútra tejto značky sa zadávajú atribúty špecifikujúce verziu, kódovanie, a ako voliteľný atribút sa uvádza atribút `standalone`. Povinný atribút `version` špecifikuje, ktorá verzia XML je použitá v dokumente. Väčšinou je to verzia `1.0`, ale môže sa uviesť aj iná verzia podľa potreby. Atribút kódovania, `encoding`, je nepovinným a jeho hodnota špecifikuje kódovanie znakov v dokumente. Najčastejšia hdnota, ktorú atribút `encoding` nadobúda, je `UTF-8`, ale jedna z ďalších možných hodnôt je napríklad aj `UTF-16`, či iné podľa potreby. Atribút `standalone` nadobúda hodnoty iba `yes` alebo `no`. Jeho význam je taký, že určuje, či je dokument spracovateľný sám alebo nie. Príklad korektnej deklarácie xml so všetkými atribútmi:

`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`.

### Komentáre

Komentáre sa v XML sa uvádzajú do špeciálnych značiek `<!-- -->` a medzi ne sa píše obsah komentára. Slúžia na poznámkovanie alebo dokumentáciu XML dokumentu. Príklad XML komentára:

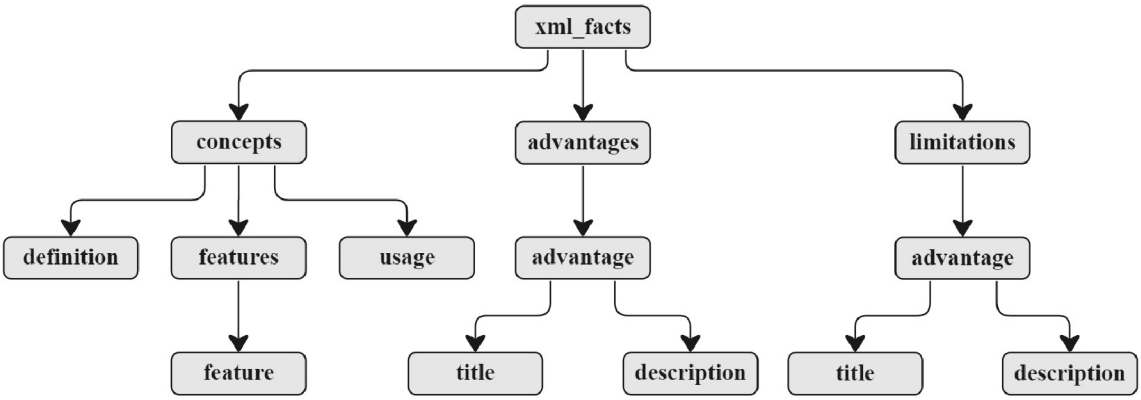
`<!-- This is comment in XML-->`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml_facts>
    <concepts>
        <concept id="c001" name="XML">
            <definition>
                Extensible Markup Language (XML) is a markup language that
                defines rules for encoding documents in a format that is
                both human-readable and machine-readable.
            </definition>
        </concept>
    </concepts>
    <advantages>
        <advantage id="a001">
            <title>Flexibility</title>
            <description>
                XML allows users to define their own tags, making it highly
                adaptable to various data formats and applications.
            </description>
        </advantage>
    </advantages>
</xml_facts>
```

Výpis 2.1: Vzorový príklad XML dokumentu.

### XML strom

Celý XML dokument je štrukturovaný ako hierarchický strom. To znamená, že dokument začína takzvaným **koreňovým** elementom a ďalšie nasledujúce elementy sú jemu podradené, teda sú jeho **listy**. Takto sa vytvára takzvaná štruktúru nadradenosti a podradenosti, kde elementy môžu byť nadradené iným elementom, ale zároveň aj byť podradený inému elementu.




Obr. 2.1: Príklad štruktúry XML stromu pre XML 2.1.

### XML Schema a DTD

XML Schema Definition Language: Structures (XSD) [19] a Document Type Definition (DTD), sú dve základné špecifikácie používané na definíciu štruktúry a validáciu XML dokumentov.

*XSD slúži na definíciu a popis triedy XML dokumentu pomocou schematických komponentov na obmedzenie a zdokumentovanie významu, použitia a vzťahov základných častí, ako sú dátové typy, elementy a ich obsah, a atribúty, a ich hodnoty. Schémy tiež môžu poskytnúť špecifikáciu dodatočných informácií o dokumente, ako je normalizácia a štandardné nastavenie hodnôt atribútov a prvkov. Schémy majú možnosť svojej vlastnej dokumentácie. XSD je teda možné použiť na definovanie, popis a katalogizáciu slovníkov XML pre triedy dokumentov*; preložené z XML[19]. Zjednodušene je XSD samostatný XML dokument, ktorý popisuje, ktoré elementy a atribúty sa môžu v XML dokumente nachádzať, ktoré elementy sú podelementy, ich poradie a počet, obsah elementu, dátový typ a východzie hodnoty elementov a atribútov.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="xml_facts">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="concepts" type="conceptsType"/>
                <xs:element name="advantages" type="advantagesType"/>
                <xs:element name="limitations" type="limitationsType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="conceptsType">
        <xs:sequence>
            <xs:element name="concept" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="definition" type="xs:string"/>
                        <xs:element name="features" type="featuresType"/>
                        <xs:element name="usage" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:string"/>
                    <xs:attribute name="name" type="xs:string"/>
                </xs:complexType>
          </xs:element>
        </xs:sequence>
    </xs:complexType>
    <!-- XSD for the rest of elements continues -->
</xs:schema>
```

Výpis 2.2: Vzorový príklad XSD pre XML 2.1.

DTD vznikla ako prvá štandardizovaná metóda pre definovanie štruktúry a validáciu XML a ostatných formátov ako HTML, založených na SGML. DTD tiež definuje štruktúru XML dokumentu podobne ako XSD, ale keďže vznikol omnoho skôr, tak prináša aj určité obmedzenia, napríklad v rámci deklarácie zložitejších vzťahov medzi elementami. DTD môže byť definovaný buď priamo v dokumente, alebo ako zvlášť súbor s príponou `.dtd`.

```xml
<!DOCTYPE xml_facts [
    <!ELEMENT xml_facts (concepts, advantages, limitations)>
    <!ELEMENT concepts (concept+)>
    <!ELEMENT concept (definition, features, usage)>
    <!ATTLIST concept id CDATA #REQUIRED name CDATA #REQUIRED>
    <!ELEMENT definition (#PCDATA)>
    <!ELEMENT features (feature+)>
    <!ELEMENT feature (#PCDATA)>
    <!ELEMENT usage (#PCDATA)>
    <!-- DTD for the rest of elements continues -->
]>
```

Výpis 2.3: Vzorový príklad DTD pre XML 2.1.

### 2.1.4 Prínosy a obmedzenia pri použití XML pre ukladanie dát

Prínosov pri ukladaní dát v XML je veľké množstvo. Najhlavnejším z nich je hierarchická štruktúra, ktorá umožňuje ukladať dáta organizovane do zložitej hierarchie pomocou značiek a elementov. Štruktúra formátov ako CSV alebo JSON zvyčajne nebýva až tak štrukturovaná ako XML, a tým pádom sú aj menej vhodné pre zložitú hierarchiu. Vďaka možnej definícií vlastných značiek a atribútov je možné XML prispôsobiť pre rôzne aplikácie, čo iné formáty, ako CSV nie úplne umožňujú. XML je široko podporovaný a nemá problém s kompatibilitou s rôznymi technológiami, väčšinou nie je potrebná konverzia dát do iného formátu, a práve pre to je ho možné integrovať takmer do každého projektu.

Obmedzení pri písaní a použití XML je niekoľko. Je potrebné dodržať korektnú štruktúru XML, musí obsahovať jeden jedinečný koreňový element, dodržanie zatváracích značiek pre každý element, zároveň každý element musí byť správne ukončený, teda nesmie nastať situácia, kedy napríklad začneme písať po elemente A element B, ale element A nie je ešte ukončený. Treba tiež brať do úvahy, že XML dokumenty sú case sensitive, a taktiež klásť dôraz na správne zapísanie hodnoty atribútov do úvodzoviek. Ďalšie možné problémy pri pracovaní s XML môžu nastať pri pracovaní s obsiahlymi a veľkými XML dokumentami, čo môže mať vplyv na kapacitu úložiska a taktiež aj na výkon zariadenia. Čas spracovania XML dokumentu tiež závisí na jeho veľkosti, čo pri obsiahlych XML môže trvať mnoho jednotiek času, a tým pádom je program menej efektívny a rýchly. Hierarchická štruktúra môže viesť ako následok komplikovanejšieho spracovania dát a manipulácií s nimi v porovnaní s inými formátmi, ako je napríklad JSON.

### 2.1.5 Príklady prípadov použitia XML vo vývojových prostrediach

Vďaka všestrannosti XML je možné ho použiť v širokej škále aplikácií v rôznych systémových odvetviach. Najčastejšie sa XML používa pre uchovávanie dát, výmenu dát alebo podklad pre vytváranie nových jazykov na základe XML.

Vo webových službách XML využíva SOAP (Single Object Access Protocol), ktorý využíva XML ako formát správ, ktoré obsahujú štrukturované informácie pri implementácií webových služieb v počítačových sieťach. XML využíva taktiež aj REST (Representational State Transfer) ako formát reprezentácie zdroja, ale využiť môže aj iný formát reprezentácie dát.

XML je využité tiež aj v systémoch pre správu dokumentov, DMS (Document Management System), na štruktúrovanie a ukladanie metadát dokumentov, hierarchiu adresárov, či prístupové práva. Medzi takéto aplikácie patria napríklad OpenText Documentum, Microsoft SharePoint, Alfresco, a mnohé iné. Taktiež je možné využiť XML pre písanie technickej dokumentácie či technických správ.

Ako základom pre nové značkovacie jazyky je často použité XML. Medzi najznámejšie patrí SVG, XHTML, MathML, Office Open XML, a iné. Zoznam značkovacích jazykov založených na XML je dostupný tu [16].

XML je často používaný ako typ konfiguračného súboru v rôznych IDE platformách. V Eclipse sú na XML založené konfiguračné súbory `.project` a `.classpath`. Taktiež IntelliJ IDEA je známe IDE pre Java, kde konfiguračný súbor `workspace.xml` je XML a obsahuje konfiguráciu nastavení projektu a modulov. Asi najznámejším konfiguračným súborom je `.csproj` vo Visual Stuio pre jazyk C#.

XML sa dá využiť aj na definíciu rôznych obvodov, stavových automatov, grafov a systémov v reálnom čase. Medzi takéto nástroje patrí Uppaal, PRISM, KRONOS. Ďalej sa

XML využíva v CMSIS ako `.svd` formát pre definovanie registrov, polí a funkcionality na rôznych periférіách pre ARM Cortex-M mikrokontroléry.

Použitie XML je veľmi časté aj vo finančných službách, geografických informačných systémoch GIS, zdravotníckych informačných systémoch, hardvérových aplikáciách, mobilných aplikáciách, na vývoj android aplikácií či rôznych UI, konfiguračných nástrojov, a mnohých iných. XML dokument má veľmi variabilné využitie a tieto príklady použitia sú iba zlomok toho, na čo je ho možné využiť.

## 2.2 Technická dokumentácia

### 2.2.1 Význam technickej dokumentácie v softvérovom vývoji

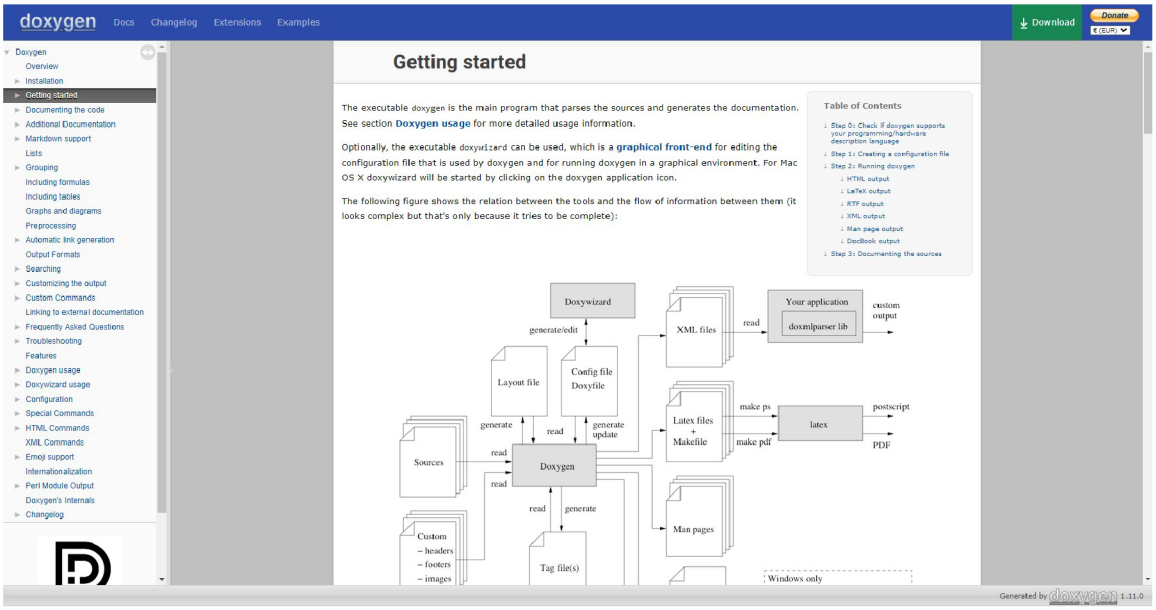
Technická dokumentácia predstavuje kľúčový prvok v softvérovom vývoji a informačných technológiách. Andrew Forward definuje softvérovú dokumentáciu ako *artefakt, ktorého účelom je komunikovať informácie o softvérovom systéme, ku ktorému patrí* [5]. Ide o súbor dokumentov a materiálov, ktoré detailne popisujú návrh, implementáciu, použitie a údržbu softvérových systémov. Takáto dokumentácia slúži na poskytnutie jasného a podrobného pohľadu na všetky aspekty projektu pre rôzne zainteresované strany. Niektoré údaje v tejto sekcií boli prebrané z [9] a [15].

Technická dokumentácia sa väčšinou píše z troch dôvodov, a to buď pomôcť konečnému užívateľovi, podporiť vývoj projektu alebo pomôcť svojej organizácií porozumieť danému projektu. V softvérovom vývoji sa môžeme stretnúť s typmi technickej dokumentácie, ako dokumentácia požiadaviek na vývoj, dokumentácia dizajnu a architektúry, dokumentácia testovania a poprípade aj dokumentácia o údržbe systému. Samozrejme, typov dokumentácií je veľmi veľa a medzi často používané patria aj užívateľská príručka, technická špecifikácia, technické správy či výskumné práce.

Kvalitná technická dokumentácia prispieva k lepšiemu a rýchlejšiemu porozumeniu projektu či už pre nových členov tímu, ale aj tých stálych. Tým, že členovia tímu dokážu vďaka kvalitnej technickej dokumentácií pochopiť projekt rýchlejšie, znižuje sa čas a úsilie vyhradené pre osvojenie si kódu projektu, a je možné tento čas venovať vývoju alebo iným potrebám projektu.

Obsahom technickej dokumentácie sú aj informácie o funkcionalite, rozhraniach či architektúre softvéru. Na základe týchto informácií je možné plánovať implementáciu nových funkcií a vylepšení, čo pomáha aj zladiť nové vlastnosti funkcionality s už existujúcou, a taktiež dodržiavať stanovené princípy a konvencie. Niekedy sa v technickej dokumentácií môžu vyskytnúť aj nejaké nové myšlienky či nápad novej funkcionality, ktorú by bolo dobré v budúcnosti implementovať, a to sa tiež dá zahrnúť pri spomínanom plánovaní novej implementácie či vylepšenej funkcionality. Avšak, obsahom môžu byť aj výsledky testovania systému, implementácia a návrh systému, z čoho je ľahké detekovať chyby bez skúmania zdrojových súborov a zahrnúť opravu chýb do vývoja. To, že technická dokumentácia poskytuje prehľad takýchto informácií znižuje čas potrebný na detekciu chýb a zvyšuje efektivitu práce v softvérovom tíme.

V niektorých prípadoch, dokumentácia obsahuje aj návod k používaniu, inštalácií, či iné detaily určené pre užívateľa. To uľahčuje porozumenie produktu užívateľovi, znižuje to pokladanie otázok na technickú podporu a slúži ako zrozumiteľná reprezentácia celého projektu, vďaka ktorej je možné produkt aj jednoduchšie predstaviť vrámci marketingu produktu.




Obr. 2.2: Príklad výzoru online dokumentácie, prevzaté z dokumentačných stránok Doxygen [4].

### 2.2.2 Požiadavky kladené na technickú dokumentáciu

Tvorca technickej dokumentácie by si mal pri jej písaní dávať pozor, akým spôsobom píše a štruktúruje technickú dokumentáciu, pre koho ju píše a akú má mať výsledná dokumentácia podobu. Niektoré informácie v tejto sekcií boli prevzaté z [3] a [12].

Ako je uvedené v knihe Handbook of Technical writing[1] pri efektívnom a úspešnom písaní technického dokumentu sa odporúča dodržiavať tento postup: príprava, rešerš, organizácia, písanie a revízia. V prípravnej fáze sa odporúča určiť si hlavný účel dokumentu, určenie cieľovej skupiny a stanovenie rozsahu pokrytia z hľadiska obsahu a určiť hlavné komunikačné médium. V rešeršnej časti je potreba nahromadiť zdroje informácií a porozumieť téme. Organizácia znamená vytvorenie najlepšej možnej štruktúry myšlienok a usporiadania informácií. Pri písaní práce je dôležité pretvárať myšlienky do textu tak, aby boli pochopiteľné čitateľovi. V revíznej časti je potreba skontrolovať text a opraviť chyby. Dôležité je pri tom byť kritický, ako keby je text čítaný prvý krát.

Obsah a štruktúra sú potrebné prispôsobiť typu dokumentácie. Z hľadiska obsahu je potrebné si uvedomiť, pre koho je dokumentácia určená a podľa toho navrhnúť štruktúru dokumentu. Obsah dokumentácie by mal byť jasný a úplný pre minimalizovanie nejasností a prípadného neporozumenia obsahu. Z hľadiska relevantnosti by mala byť dokumentácia vecná a obsahovať iba konkrétne informácie určené čitateľovi podľa typu dokumentácie, napr.: informácie o implementácií sú úplne nevhodné do užívateľskej príručky.

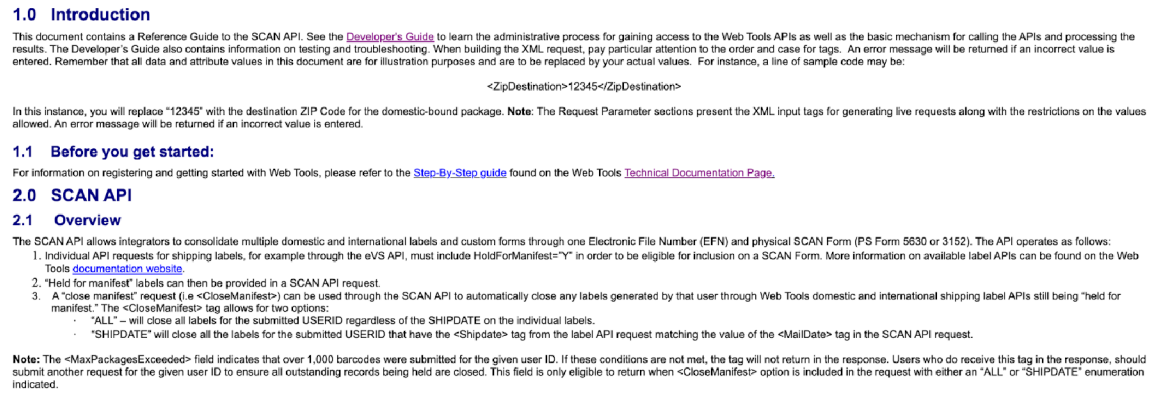
Z hľadiska štýlu je vhodné použiť jednotnú terminológiu a štýl písania. Technické termíny by mali byť použité tak, aby boli jednotné vrámci celého dokumentu a zároveň ľahko pochopiteľné pre čitateľa.

V neposlednom rade je dôležité aj formátovanie a grafické spracovanie. Pre jednotnosť štýlov je odporúčané používať predpripravené šablóny, ak sú k dispozícií. Ak šablóna k dispozícií nie je, odporúča sa predurčiť jednotný štýl textu používaný v celom dokumente. V rámci ľahkého pochopenia zložitejších konceptov je vhodné použiť obrázky, grafy či ta-

buľky vo vhodnej miere. Príklad nevhodne spracovanej dokumentácie z hľadiska štýlov a formátovania je na obrázku 2.3.

Vhodnými pomocníkmi pri tvorbe technickej dokumentácie je zvolenie vhodného dokumentačného nástroja na tvorbu a správu technickej dokumentácie. Treba brať do úvahy, či bude dokumentácia automaticky generovaná, alebo bude písaná ručne. Veľkým pozitívom je, keď je dokumentácia dostupná v online forme, a taktiež obsahuje veľké množstvo interakčných prostriedkov pre ešte lepšie pochopenie.

V neposlednom rade treba spomenúť, že každá dobrá dokumentácia potrebuje pravidelnú údržbu a aktualizáciu informácií, aby nebola pozadu oproti softvéru.



**1.0 Introduction**

This document contains a Reference Guide to the SCAN API. See the Developer's Guide to learn the administrative process for gaining access to the Web Tools APIs as well as the basic mechanism for calling the APIs and processing the results. The Developer's Guide also contains information on testing and troubleshooting. When building the XML request, pay particular attention to the order and case for tags. An error message will be returned if an incorrect value is entered. Remember that all data and attribute values in this document are for illustration purposes and are to be replaced by your actual values. For instance, a line of sample code may be:

<ZipDestination>12345</ZipDestination>

In this instance, you will replace "12345" with the destination ZIP Code for the domestic-bound package. **Note**: The Request Parameter sections present the XML input tags for generating live requests along with the restrictions on the values allowed. An error message will be returned if an incorrect value is entered.

**1.1 Before you get started:**

For information on registering and getting started with Web Tools, please refer to the Step-By-Step guide found on the Web Tools Technical Documentation Page.

**2.0 SCAN API**

**2.1 Overview**

The SCAN API allows integrators to consolidate multiple domestic and international labels and custom forms through one Electronic File Number (EFN) and physical SCAN Form (PS Form 5630 or 3152). The API operates as follows:

1. Individual API requests for shipping labels, for example through the eVS API, must include HoldForManifest="Y" in order to be eligible for inclusion on a SCAN Form. More information on available label APIs can be found on the Web Tools documentation website.
2. "Held for manifest" labels can then be provided in a SCAN API request.
3. A "close manifest" request (i.e <CloseManifest>) can be used through the SCAN API to automatically close any labels generated by that user through Web Tools domestic and international shipping label APIs still being "held for manifest." The <CloseManifest> tag allows for two options:
   - "ALL" – will close all labels for the submitted USERID regardless of the SHIPDATE on the individual labels.
   - "SHIPDATE" will close all the labels for the submitted USERID that have the <Shipdate> tag from the label API request matching the value of the <MailDate> tag in the SCAN API request.

**Note:** The <MaxPackagesExceeded> field indicates that over 1,000 barcodes were submitted for the given user ID. If these conditions are not met, the tag will not return in the response. Users who do receive this tag in the response, should submit another request for the given user ID to ensure all outstanding records being held are closed. This field is only eligible to return when <CloseManifest> option is included in the request with either an "ALL" or "SHIPDATE" enumeration indicated.

Obr. 2.3: Príklad zle štruktúrovanej a nevhodne naštýlovanej dokumentácie, prevzaté z [3].

### 2.2.3 Možnosti automatizácie pre splnenie niektorých požiadaviek

Automatizácia generovania dokumentácie sa dostáva čoraz viac do popredia. Medzi časti technickej dokumentácie, ktoré sa dajú generovať automaticky, patria rôzne grafy a diagramy, ktoré je možné generovať na základe vstupov. Užívateľ považuje za samozrejmé generovanie obsahu na základe štruktúry dokumentu, a tiež je možné generovať aj indexy a zoznamy na základe štandardných metód kategorizácie.

Generovať sa avšak dajú aj komplexnejšie časti dokumentácie. Veľmi známym spôsobom generovania dokumentácie je generovanie dokumentácie zo zdrojového kódu, čítaním a analyzovaním komentárov a ich následné spracovanie do upraviteľnej dokumentácie. Existujú aj možnosti, ako vygenerovať šablónu alebo štruktúru na základe zadaných vstupných údajov alebo poskytnutých súborov. Taktiež je možné generovať dokumentáciu na základe nejakého vstupného formátu dát. To však vyžaduje komplexnejšie riešenie a spôsob spracovania vstupného súboru.

Automatické generovanie dokumentácie je vhodným nástrojom, pokiaľ je kladený dôraz na dodržanie požiadaviek na štruktúru, formátovanie textu a grafické spracovanie. Zároveň generovanie technickej dokumentácie uľahčuje dodržiavať jednotný štýl a formuláciu dokumentácie, a jednoznačne skracuje čas na jej vytvorenie. Zároveň je možný výskyt aj nejakých nevýhod, ktoré môžu tvorbu dokumentácie spomaliť alebo znevýhodniť generovanie oproti ručnému písaniu.

## 2.3 Automatizované generovanie technickej dokumentácie

### 2.3.1 Metódy a prístupy generovania dokumentácie

Pre generovanie technickej dokumentácie priamo zo zdrojového kódu existuje viacero techník a nástrojov, ktoré umožňujú automatizované získavanie informácií a ich transformáciu do výsledného formátu dokumentácie. Tieto postupy sú kľúčové pre efektívne spravovanie a udržiavanie komplexných softvérových projektov.

Praktickým spôsobom pre generovanie dokumentácie zo zdrojového kódu je použitie nástroja navrhnutého na automatické generovanie. Tieto nástroje fungujú na princípe analýzy štruktúry zdrojového kódu a snažia sa analyzovať komentáre, a získať z nich informácie potrebné na tvorbu dokumentácie. Výstupným formát dokumentácie je rôzny, napríklad: HTML, PDF, Markdown, a závisí od podporovaných výstupných formátov použitého nástroja. Príklady takýchto nástrojov zahŕňajú Javadoc, Doxygen, Sphinx a mnohé ďalšie.

Najčastejším spôsobom je generovanie dokumentácie priamo z **komentárov** v zdrojových súboroch. Komentáre v kóde slúžia na dokumentovanie funkcionality a implementačných detailov jednotlivých úsekov kódu. Ich štruktúra zabezpečuje zrozumiteľné pochopenie funkcionality kódu pre ďalších vývojárov. Komentáre typicky obsahujú informácie o funkciách, triedach, metódach a premenných. Tieto komentáre môžu obsahovať popisy funkcií, vysvetlenia parametrov, návratových hodnôt a príklady použitia, či konkrétne vysvetlenia pokročilejších častí kódu. Výhodou tejto metódy je priama integrácia do procesu vývoja, čiže je možné vidieť dokumentáciu priamo pri čítaní kódu, a to uľahčuje jeho pochopeniu. Zároveň hrozí riziko pri údržbe a aktualizácií komentárov, keďže je potrebné ich aktualizovať pri každej zmene v kóde. Príkladom nástroja čo generuje dokumentáciu priamo z komentárov je napríklad Natural Docs.

V rámci komentárov sa v kóde používajú **špeciálne značky** alebo **anotácie**, ktoré majú špeciálny význam alebo sú určené pre generovanie dokumentácie. Takéto značky obsahujú rôzne metadáta, obvykle doplnkové informácie o kóde, ktoré majú byť zahrnuté do dokumentácie. Umiestnenie značiek je zvyčajne pred definíciou tried, metód či premenných a môžu obsahovať popis metód, vysvetlenie parametrov, návratových hodnôt, príkladov použitia či vysvetlenie všeobecnej funkcionality kódu. Tieto značky sú buď štandardizované, ako napríklad využíva nástroj Javadoc alebo Doxygen. Taktiež je možné vytvoriť aj vlastné značky, ktoré sa väčšinou vytvárajú v rámci projektu a prispôsobujú sa jeho potrebám.

Ďalšou možnosťou je využitie nástrojov na statickú analýzu kódu, ktoré môžu byť tiež využité na generovanie dokumentácie. Nástroje tohto typu analyzujú zdrojový kód a identifikujú vzory alebo štandardy v komentároch, závislosti a ďalšie relevantné informácie, ktoré môžu byť použité na generovanie dokumentácie. Na tomto princípe fungujú nástroje ako je Swimm a iné.

Niektoré nástroje (IDE) poskytujú možnosť **integrovania do vývojového prostredia**, čo ponúka automatické generovanie komentárov alebo častí dokumentácie automaticky. Výhodou je konzistentnosť a kvalita komentárov v celom zdrojov kóde, a taktiež možnosť rýchlejšieho dokumentovania kódu. Príkladom takéhoto spôsobu generovania je napríklad Visual Studio, ktoré dokáže automaticky generovať štruktúru komentárov typu XML pre jazyk C#.

Možnosťou pre generovanie dokumentácie je aj jej **generovanie zo súborov** obsahujúcich metadáta. Formát súboru je obvykle XML, JSON, YAML a mnohé iné, a obvykle obsahujú informácie o aplikácií či API. Písanie dokumentácie je v tomto prípade oddelené od zdrojového kódu a je to vhodné najmä pre projekty so špeciálnymi požiadavkami pre

dokumentáciu, alebo potrebujú flexibilnejší spôsob pri definícií informácií mimo zdrojového kódu. Nástroj ktorý funguje na takomto princípe je napríklad DocFX, Swagger pre dokumentáciu API a mnohé iné. Nástroje ponúkajú rôzne výstupné formáty, najčastejšie sú HTML, PDF či Markdown.

### 2.3.2 Nástroje a technológie pre automatizované generovanie dokumentácie

Existuje mnoho nástrojov, ktoré umožňujú automatizované generovanie technickej dokumentácie z XML dát. Medzi najčastejšie patrí Doxygen, Javadoc a Sphinx. Tieto nástroje poskytujú rôzne výhody, ako sú automatická generácia dokumentácie, šablóny pre formátovanie, interaktívne navigačné možnosti a podpora pre rôzne programovacie jazyky. Súčasne však čelia výzvam, ako napríklad správne analyzovanie komplexných štruktúr XML alebo správna interpretácia rôznorodých vstupných dát.

#### Doxygen

Doxygen[4] je nástroj na generovanie dokumentácie z komentárov v zdrojovom kóde. Podporuje rôzne programovacie jazyky vrátane C++, Java, Python, a mnohé ďalšie. Vstupom sú komentáre v zdrojovom jazyku a výstupom je HTML, PDF alebo LaTeX. Jeho výhodami sú automatické generovanie dokumentácie z komentárov, podpora rôznych výstupných formátov HTML, PDF, LaTeX, či schopnosť vytvárať interaktívne grafy tried a volania funkcií. Nevýhodou je, že nie vždy poskytuje vizuálne atraktívnu dokumentáciu, a pre nových užívateľov môže byť komplexný vzhľadom na bohatú možnosť konfigurácie.

#### Sphinx

Sphinx[14] je nástroj na generovanie dokumentácie pre projekty napísané v Python a iných jazykoch. Používa formát reStructuredText podobný XML a podporuje rôzne výstupné formáty ako HTML, PDF, ePub, a podobne. Jeho výhodami sú flexibilná štruktúra dokumentácie a možnosť generovania statických aj interaktívnych stránok. Nevýhodou môže byť reStructuredText syntax, ktorá môže byť pre niekoho náročná, a to, že tento nástroj nie je až tak univerzálny v porovnaní s inými nástrojmi.

#### JavaDoc

Javadoc[10] je nástroj na generovanie dokumentácie pre Java aplikácie. Využíva komentáre v zdrojovom kóde a produkuje HTML dokumentáciu. Výhodou je jednoduché a štandardizované použitie pre Java projekty, automatická generácia dokumentácie z anotácií a komentárov, a integrácia s mnohými vývojovými prostrediami. Nevýhodou je, že je špecializovaný priamo pre jazyk Java a to,že nie je natoľko flexibilný ako iné nástroje tohto typu.

#### Oxygen XML Editor

Oxygen XML Editor[13] je IDE určené pre prácu s XML dokumentami. Jeho základnými funkciami je možnosť editovania XML dokumentov na vysokej úrovni a zároveň aj možnosť publikácie dokumentu vo formátoch PDF, CHM, EPUB a ďalších. Podporuje tiež aj XPath, XQuery, XSD a XSLT pre prácu s XML dátami. Jeho hlavnou nevýhodou je to, že po uplynutí skúšobného obdobia je nutné za tento softvér platiť.

| Nástroj | Jazyk | Vstup | Výstup | Dostupnosť |
|---|---|---|---|---|
| Doxygen | C++, Java, Python, atď. | Komentáre v zdrojovom kóde | HTML, PDF, LaTeX | Open source |
| Sphinx | Python, atď. | reStructuredText | HTML, PDF, ePub, atď. | Open source |
| JavaDoc | Java | Komentáre v zdrojovom kóde | HTML | Open source |
| Oxygen | Nie je obmedzený na programovací jazyk | XML | PDF, CHM, EPUB, atď. | Komerčný |

Tabuľka 2.1: Porovnanie vlastností nástrojov.

### 2.3.3 Prínosy a výzvy spojené s automatizáciou generovania dokumentácie

Automatizácia generovania technickej dokumentácie sa dostáva čoraz viac do popredia v oblasti softvérového vývoja. Kvôli potrebe rýchlejšej a presnejšej dokumentácie začínajú hrať kľúčovú úlohu automatizované nástroje a procesy pre generovanie dokumentácie. Na druhej strane s automatizovaným generovaním dokumentácie sú spojené prínosy aj výzvy, ktorým treba čeliť a brať do úvahy. Niektoré údaje v tejto sekcií boli prebraté z [11] a [2].

Automatizácia dokumentácie ponúka veľké množstvo výhod, ktoré môžu zefektívniť dokumentačný proces a zároveň tým zvýšiť efektivitu v iných častiach projektu. Jednou z najhlavnejších výhod je **konzistencia**. Automatizované nástroje na generovanie dokumentácie zabezpečujú jednotné formátovanie, štýl a terminológiu vo všetkých súboroch výslednej dokumentácie, alebo aj možnosť definovania personalizovaného štýlu pre dokument. Použitím automatizovaného nástroje je rovno zabezpečený ucelený a jednotný vzhľad dokumentácie, a predchádza sa tým riziku chýb, ktoré by mohli vzniknúť v prípade nekonzistentnej dokumentácie. Jednotným formátom sa tiež zlepšuje **dostupnosť**, priehľadnosť a ľahká orientácia v dokumentácií. **Štandardizáciu** v dokumentačných postupoch zabezpečuje konzistentnosť dokumentácie, čo znamená dodržiavanie vopred definovaných pravidiel alebo firemných požiadaviek všetkými členmi tímu. Vhodne štruktúrovaná a aktuálna dokumentácia sa prejavuje aj na **užívateľskej skúsenosti**, keďže celková použiteľnosť kvalitnej dokumentácie predchádza zmätkom pri používaní dokumentácie, hľadaní informácií či porozumenia obsahu rôznymi členmi tímu.

Ďalšou, a asi najviac prínosnou výhodou je **úspora času**. Vďaka automatizácií dokumentácie je potrebné stráviť menej času nad jej tvorbou, a preto vývojári môžu venovať viac času dôležitejším úlohám na projekte, a tým pádom sa aj zvyšuje efektívnosť práce.

Nasledujúcou výhodou je **presnosť**, ktorú automatizovaná dokumentácie poskytuje. Niektoré nástroje dokážu získať informácie priamou extrakciou dát zo zdrojového kódu alebo iných dodatočných dokumentov, a tak je možné zaistiť jej aktuálnosť s najnovšími zmenami v projekte, a tým pádom odzrkadľuje jeho aktuálny stav. Aktuálnosť dokumentácie je zabezpečená aj vďaka integrácií so správou verzií. Týmto spôsobom je dokumentácia aktualizovaná pri každej zmene v kóde, a taktiež je udržiavaná synchronizácia medzi verziami kódu a dokumentácie. Automatická aktualizácia dokumentácia sa škáluje na základe rozsiahlosti projektu, vďaka čomu sa dá udržať aktuálna pri rozsiahlych projektoch. Vďaka

automatizovanej aktualizácií dokumentácie sa tiež znižuje réžia údržby a množstvo manuálnych zásahov potrebných k úpravám.

Jednou z kľúčových výziev pri automatickom generovaní technickej dokumentácie je potreba **kvalitných dát**, z ktorých sa bude dokumentácia generovať. Pre generovanie jednoznačnej a spoľahlivej dokumentácie s hodnotným obsahom je potrebné mať ako základ kvalitné zdrojové dáta, čo býva častým problémom pri generovaní dokumentácie, a môže tým byť ovplyvnená účinnosť procesu automatizovaného generovania. Problémy môžu nastať kvôli neaktuálnym alebo neúplným dátam, alebo kvôli chaotickému rozmiestneniu dát medzi viacerými súbormi. Práve pre tieto riziká je vhodné dáta kontrolovať a v prípade nedostatkov dáta aj aktualizovať.

S touto výzvou je tiež spojená **nedostatočná presnosť** alebo **nízka kvalita** dokumentácie. Tieto riziká môžu viesť k neúplnému či chybnému textu dokumentácie, čo môže vo veľkej miere ovplyvniť alebo zavádzať užívateľa. Ďalšími možným rizikami je aj neúplnosť informácií a podrobností, alebo nezrozumiteľný či príliš technický jazyk textu, čo vedie k riziku nejasnému pochopeniu dokumentácie.

Ďalší rizikom je kvalita nástroja využívaného na generovanie dokumentácie a tiež kontrola kvality výsledného dokumentu. Nástroje pre generovanie dokumentácie môžu mať určité chyby, ktoré môžu viesť k chybe vo vygenerovanej dokumentácií. Taktiež nástroje nemusia korektne zvládať neštandardné či hraničné situácie, a preto je veľmi dôležitá kontrola výslednej dokumentácie, ale tiež potreba správne vybrať alebo implementovať nástroj ktorým bude dokumentácia generovaná.

# Kapitola 3

# Návrh riešenia

Podľa rozboru problematiky ohľadom automatického generovania technickej dokumentácie je navrhnutý systém, ktorého úlohou je automatizovane generovať technickú dokumentáciu z XML vstupu. Oblasť automatizovaného generovania technickej dokumentácie poskytuje dostatočné množstvo nástrojov na generovanie technickej dokumentácie priamo zo zdrojového kódu a iných nástrojov, alebo editorov dokumentácie, avšak v tomto prostredí je nedostatok nástrojov, ktoré by generovali dokumentáciu z dodaného vstupného súboru. Takýto problém je evidovaný aj v problematike generovania dokumentácie z XML súborov. Existujúcich nástrojov je veľmi málo, a existujúce nástroje neposkytujú všetky realizačné možnosti, ako by bolo potrebné v tejto problematike. Navrhovaný systém sa teda zaoberá automatizovaným generovaním technickej dokumentácie priamo z XML vstupného súboru. Propozícia riešenia je zameraná na návrh spoľahlivého systému, ktorý bude zjednodušovať proces tvorby a udržiavania technickej dokumentácie.

V tejto kapitole bude postupne popísaný návrh nástroja generujúci technickú dokumentáciu. Bude popísaná štruktúra nástroja, typy vstupných a výstupných súborov podporovaných nástrojom, a tiež budú opísané požiadavky na nástroj a využité technologické prostriedky.

## 3.1 Požiadavky a realizačné prostriedky

Pre vytvorenie efektívneho a užívateľsky prívetivého nástroja bolo potrebné určiť presné požiadavky na jeho funkcionalitu, ale bolo treba vhodne vybrať aj vývojové prostredie pre zabezpečenie multiplatformnosti a tiež ľahkého používania.

### 3.1.1 Požiadavky

Od každého nástroja zameraného na generovanie technickej dokumentácie je potrebné aby boli dodržané jednak určité požiadavky na nástroj, a zároveň aby boli splnené pravidlá a normy pre správnosť formátu technickej dokumentácie.

#### Požiadavky na nástroj

Pri návrhu nástroja bolo zvážených množstvo požiadaviek na výsledný nástroj, avšak po ich prehodnotení boli vybrané tieto:

- **Jednoduché použitie** je kľúčovou požiadavkou na nástroj. Pre zabezpečenie ľahkej manipulácie by mali byť zdrojové súbory nástroja ľahko prenositeľné, a tiež rýchlo

pripravené k používaniu. V rámci jednoduchej manipulácií s nástrojom by malo byť zabezpečené ľahké a intuitívne ovládanie nástroja.

- **Rozšíriteľnosť** je požiadavka, vďaka ktorej by malo byť umožnené do nástroja ľahko dopĺňať funkcionalitu a prípadne nové metódy spracovávajúce iné typy XML súborov, alebo poskytnutie ďalších výstupných formátov dokumentácie.
- **Možnosť spracovania rôznych druhov XML súborov** pridáva nástroju multifunkčnosť v rámci spracovania rozličných typov XML dát a vytvára tak nástroj s rozličnými prípadmi použitia, čo je užívateľsky veľmi prívetivé, a priam až vyžadované.
- **Podpora rozličných výstupných formátov** poskytuje užívateľovi možnosť prispôsobiť výsledok dokumentácie priam jeho potrebám. Jednotlivé výstupné formáty poskytujú iné možnosti a benefity, v rámci výsledného vzhľadu alebo možnosti manipulácie s výsledným súborom. Každý výstupný formát sa hodí do iného prostredia alebo pre iné prípady použitia dokumentácie. Možnosť výberu výsledného formátu umožňuje užívateľovi personalizáciu výslednej dokumentáciu vzhľadom k jej budúcemu použitiu.

#### Požiadavky na výslednú dokumentáciu

Bežné požiadavky kladené na technickú dokumentáciu boli objasnené v časti 2.2.2. Po zhodnotení všetkých odporúčaných požiadaviek na technickú dokumentáciu je zrejmé, že v rámci tohto nástroja sa dá zabezpečiť iba **jednotný štýl a grafické spracovanie** dokumentu. Na obsah a kvalitu dát obsiahnutých vo vstupnom XML súbore nástroj nemá vplyv, takže to nie je možné nástrojom zabezpečiť.

Jednotné grafické spracovanie a štýl dokumentácie je zabezpečený vo všetkých výstupných formátoch na rovnakej úrovni. V každom výstupnom súbore sa používa jednotný font a veľkosť písma pre jednotlivé úrovne nadpisov a textu, a tiež je dodržiavané jednotné odsadenie sekcií. Spracovanie modelov vygenerovaných nástrojom UPPAAL je riešené jednotným sádzaním do dokumentov, a tiež kontrolou ich rozmerov, a prípadné následné prispôsobenie pre rozmery veľkosti dokumentácie.

### 3.1.2 Technologické prostriedky

Ako programovací jazyk pre implementáciu navrhnutého riešenia bol zvolený jazyk `Python`[1] verzie `3.11.3`. Tento jazyk bol vybratý vďaka jeho všestrannosti a tiež pre to, lebo sa jedná o vysokoúrovňový jazyk s dobre čitateľnou syntaxou. Jeho ďalšou výhodou je možnosť využitia objektovo-orientovaného programovania, čo vedie k písaniu prehľadného kódu. Medzi neposledné výhody patrí aj rozsiahly ekosystém knižníc a rámcov, ktoré uľahčujú programovanie. Knižnice použité v tomto programe sú:

- `sys`[2] - na spracovanie argumentov programu príkazového riadku,
- `os`[3] - na manipuláciu so súbormi,

[1] https://www.python.org/
[2] https://docs.python.org/3/library/sys.html
[3] https://docs.python.org/3/library/os.html

- `xml.etree.ElementTree`[4] - na prácu s XML formátom,
- `subprocess`[5] - na spúšťanie prostredia UPPAAL pre generovanie modelov,
- `cairosvg`[6] - na prevod modelu v reálnom čase vo formáte `.svg` do formátu `.png`,
- `reportlab.pdfgen.canvas`[7] - na vytvorenie nového PDF súboru,
- `reportlab.lib.pagesizes`[8] - na definovanie veľkosti stránky generovaného PDF,
- `PIL.Image`[9] - na zmenu ich rozmerov obrázkov vygenerovaných prostredím UPPAAL.

Na generovanie modelov v reálnom čase bolo použité prostredie UPPAAL[10]. **UPPAAL** je integrované prostredie, ktoré slúži na modeláciu, verifikáciu a validáciu systémov v reálnom čase modelovaných ako siete časových automatov, rozšírených o dátové typy. Toto prostredie prijíma ako vstupný formát XML súbor určitej štruktúry a následne je vygenerovaný model popisujúci systém v reálnom čase. V rámci tohto nástroja je možné tieto modely generovať v rámci zakombinovania do technickej dokumentácie.

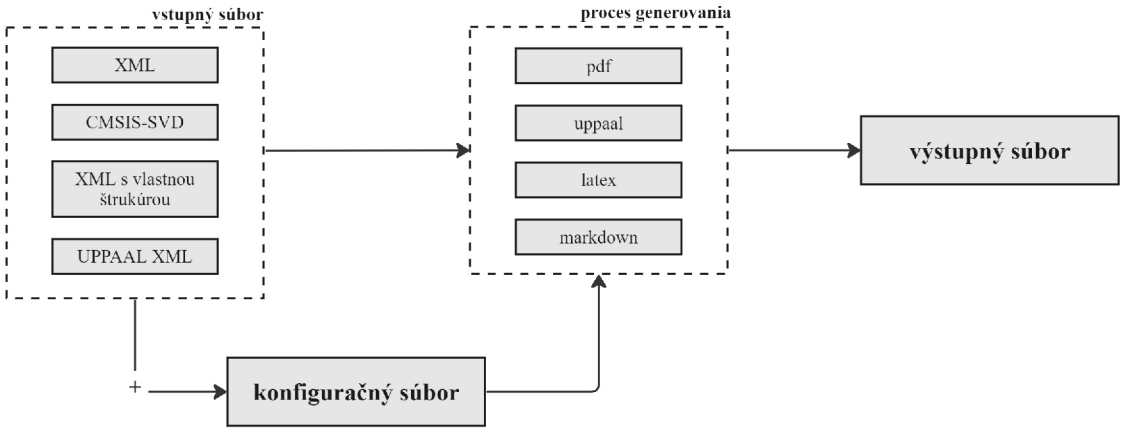
## 3.2 Špecifikácia nástroja

Táto sekcia opisuje navrhnutú architektúru nástroja spolu s prípadmi použitia nástroja. Dôkladne je tu popísaný prínos nástroja pre automatizované generovanie technickej dokumentácie spolu s jeho možnosťami využitia.

### 3.2.1 Architektúra nástroja

Architektúra nástroja je založená na procedurálnom programovaní s prvkami objektovo orientovaného programovania s možnosťou ľahkého rozšírenia do budúcna prípadnou implementáciou novej procedúry alebo funkcie. Nástroj akceptuje rôzne typy vstupných XML súborov, ako XML súbor popisujúci UPPAAL model, či SVD súbor popisujúci systém mikrokontrolérov definovaný CMSIS[11]. Nástroj taktiež dokáže spracovať navrhnutý XML súbor so špecifickými elementami prispôsobenými pre výsledný vzhľad celkovej dokumentácie. Každý zo vstupných súborov, z ktorých bude generovaná technická dokumentácia, má možnosť úpravy pomocou konfiguračného súboru. XML súbor je ďalej spracovaný procedúrami podľa požiadaviek na výstupný formát a spôsob spracovania. Výstup technickej dokumentácie je dostupný pre užívateľa v troch formátoch, ktoré nástroj podporuje, a to sú PDF, Latex a Markdown. Tieto výstupné súbory boli zvolené na základe možnosti vykonávania úprav po generovaní dokumentácie týmto nástrojom. Bloková schéma nástroja je možné vidieť na obrázku 3.1.

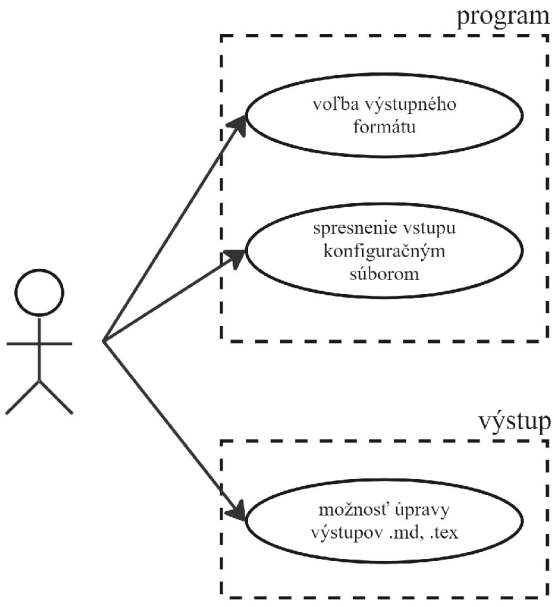
[4] https://docs.python.org/3/library/xml.etree.elementtree.html
[5] https://docs.python.org/3/library/subprocess.html
[6] https://cairosvg.org/documentation/
[7] https://docs.reportlab.com/reportlab/userguide/ch2_graphics/
[8] https://docs.reportlab.com/reportlab/userguide/ch1_intro/#useful-rl_config-variables
[9] https://pillow.readthedocs.io/en/stable/reference/Image.html
[10] https://uppaal.org/
[11] https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html




Obr. 3.1: Bloková schéma navrhnutého nástroja.

### 3.2.2 Prípady použitia nástroja

Pri spúšťaní programu je možné užívateľom zvoliť výstupný formát dokumentácie, a tiež špecifikovať ďalšie požiadavky na vstupný XML súbor prostredníctvom konfiguračného súboru. Navrhnutý nástroj je určený pre užívateľa s cieľom disponovať s kvalitnou technickou dokumentáciou, ktorá ponúka možnosti personalizácie a ľahkých úprav v budúcnosti. Diagram prípadov užitia je možné vidieť na obrázku 3.2.




Obr. 3.2: Diagram prípadov užitia nástroja.

## 3.3 Vstupy a výstupy

Vstupné a výstupné formáty hrajú dôležitú úlohu pre splnenie požiadavky o podpore rôznych formátov na vstupe a výstupe tak, aby boli prakticky použiteľné pre užívateľa. Z toho dôvodu boli vybrané ďalej opísané vstupné a výstupné formáty.

### 3.3.1 Vstupné formáty

Nástroj podporuje vstupné formáty typu XML a SVD. Pre konkrétnejšiu špecifikáciu sa jedná o tieto typy súborov:

- XML súbory,
- XML súbory popisujúce modely v prostredí UPPAAL,
- CMSIS-SVD súbor popisujúci mikrokontroléry,
- XML súbor so špecifickými elementami.

Tieto vstupné súbory typu XML boli vybrané tak, aby bolo možné nástrojom vygenerovať či už dokumentáciu obsahujúcu iba text zaznamenaný v XML súbore, alebo aj náročnejšie štruktúry popisujúce mikrokontroléry, alebo modely systémov v reálnom čase. Takýto nástroj vďaka svojej všestrannosti poskytuje nový prístup v rámci automatizovaného generovania technickej dokumentácie.

#### Vstupný súbor XML

Navrhnutý nástroj podporuje vo všeobecnosti všetky súbory XML na vstupe. V rámci obsahu XML súboru už záleží, akú bude mať výsledná vygenerovaná technická dokumentácia podobu. Ak nástroj nerozozná ani jeden element v XML súbore ako známy, teda taký, na základe ktorého dokáže určiť jeho štýl, tak sa bude XML súbor generovať čisto ako text. V opačnom prípade bude elementu priradená preddefinovaná štruktúra a štýl, v ktorom sa obsah XML elementu vygeneruje. Výsledný štýl je taktiež možné zmeniť po vygenerovaní užívateľom manuálne.

#### Vstupný súbor XML popisujúci systém v reálnom čase

Súbory XML popisujúce systémy v reálnom čase môžu byť tiež súčasťou technickej dokumentácie, alebo sa môže vyskytnúť prípad, kedy je potreba nejaký model okomentovať alebo ho vysvetliť. Navrhnutý nástroj dokáže vygenerovať UPPAAL model na základe poskytnutého XML vstupu. Taktiež je možné do XML dokumentu pridať vlastné elementy obsahujúce text, ktoré nástroj rozpoznáva, ktoré sa potom s modelmi vygenerujú ako celistvá dokumentácia. Túto dokumentáciu je možné po vygenerovaní upraviť podľa užívateľských preferencií.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nta PUBLIC "-//Uppaal Team//DTD Flat System 1.1//EN"
"http://www.it.uu.se/research/group/darts/uppaal/flat-1_2.dtd">
<nta>
    <declaration>
        // Global declarations (variables, constants, etc.)
    </declaration>
    <template>
        <name>Template1</name>
        <declaration> // Local declarations for Template1 </declaration>
        <location id="id0">
            <name>Location0</name>
            <!-- Location properties -->
        </location>
        <!-- Additional locations for Template1 -->
        <transition>
            <source ref="id0"/>
            <target ref="id1"/>
            <!-- Transition properties -->
        </transition>
        <!-- Additional transitions for Template1 -->
    </template>
    <!-- Additional templates -->
    <system>
        <!-- System declarations -->
        <component name="Template1"/>
        <!-- Additional components -->
    </system>
    <queries>
        <!-- Property queries -->
    </queries>
</nta>
```

Výpis 3.1: Kostra XML pre Uppaal model.

### Vstupný súbor CMSIS-SVD

CMSIS System View Description (CMSIS-SVD) formát formalizuje popis systému obsiahnutého v mikrokontroléroch založených na procesore Arm Cortex-M, najmä pamäť mapovanú v registroch rôznych periférií. Takýto popis je porovnateľný s údajmi v referenčných manuáloch mikrokontrolérov, avšak z tohto popisu je možné vytvoriť prehľadnejšiu dokumentáciu obsahujúcu iba tie najpotrebnejšie informácie. Navrhnutý nástroj dokáže rozpoznať takmer každý element, ktorý sa môže vyskytovať v CMSIS-SVD a na jeho základe vygeneruje prehľadnú dokumentáciu mikrokontroléra s možnosťou úpravy.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<device>
    <!-- Device information (vendor, name, series, version, etc.) -->
    <peripherals>
        <peripheral>
            <!-- Peripheral information (name, baseAddress, description,
            groupName, etc.) -->
            <registers>
                <register>
                    <!-- Register information (name, addressOffset, etc.) -->
                    <fields>
                        <field>
                            <!-- Field information (name, description,
                            bitOffset, bitWidth, access, etc.) -->
                        </field>
                        <!-- Additional fields -->
                    </fields>
                </register>
                <!-- Additional registers -->
            </registers>
        </peripheral>
    </peripherals>
</device>
```

Výpis 3.2: Kostra XML pre CMSIS-SVD.

### Vstupný súbor XML so špecifickými elementami

V rámci riešenia problematiky bolo potrebné navrhnúť vlastnú štruktúru XML súboru vhodnú pre generovanie technickej dokumentácie. Práve pre to, boli navrhnuté tieto elementy podporované navrhnutým nástrojom:

- `<chapter name="...">` generuje názov kapitoly,
- `<section>` generuje názov podkapitoly,
- `<subsection>` generuje názov podsekcie,
- `<text>` generuje text.

Tieto elementy sú rozpoznateľné v ktoromkoľvek XML vstupnom súbore a majú vopred preddefinovaný štýl, ktorý si držia vo všetkých dokumentoch. Jednotlivé elementy boli navrhnuté pre to, aby bolo možné týmto nástrojom generovať aj technickú dokumentáciu obsahujúcu text, ktorý je získaný a uložený v XML súbore. Takéto údaje je možné získať z akýchkoľvek zdrojov a pretransformovať do formátu XML, a následne vygenerovať ako technickú dokumentáciu ktorú je možné upravovať.

### 3.3.2 Výstupné formáty

Podporované výstupné súbory tohto nástroja boli vybrané na základe prieskumu v rámci najčastejšie používaných výstupných formátov pre technickú dokumentáciu. Práve pre to, boli vybrané tieto 3 výstupné súbory:

- PDF,
- Latex,
- Markdown.

Tieto súbory sú celkovo najpoužívanejšie a najbežnejšie v rámci textových dokumentov. Práve pre to, by mali byť známe pre každého užívateľa, a tým pádom aj zaručiť ľahkú manipuláciu s nimi.

#### PDF

Výstupný súbor typu PDF poskytuje užívateľovi pohodlný výstup, ktorý je navrhnutým nástrojom rýchlo spracovaný. Výstup je prehľadný a poskytuje užívateľovi rýchlo porozumieť dátam obsiahnutým v XML súbore. Avšak, výsledné PDF už nie je možné upravovať, a práve pre to je odporúčané klásť dôraz na kvalitu a korektnosť dát obsiahnutých v XML súbore vzhľadom na budúce použitie a manipuláciu s týmto typom výsledného formátu dokumentácie.

#### Latex

Výstupný súbor vygenerovaný nástrojom vo formáte Latex poskytuje možnosť úpravy dokumentu pred samotnou kompiláciou zdrojového Latex kódu do výsledného PDF formátu. Táto možnosť poskytuje generovaný dokument nástrojom ďalej upravovať a prispôsobiť ho podľa vlastných predstáv užívateľa. Veľká výhoda spočíva v možnosti predurčenia štruktúry už v XML vstupe prostredníctvom vlastných elementov nástroja s následnou možnosťou úpravy štýlu či obsahu. Úprava Latex zdrojového súboru môže spočívať aj pridaním popisu k modelom vygenerovaným z nástroja UPPAAL, ak tieto popisy neboli v XML štruktúre, pridaním obrázkov, tabuliek či v celku možnosti personalizovať dokument podľa vlastných predstáv užívateľa. Od užívateľa sa očakáva, že na úpravu Latex výstupu bude mať zabezpečené vlastné prostriedky, či už prostredníctvom online editora alebo pomocou prekladača v terminály.

#### Markdown

Výstupný súbor typu Markdown poskytuje užívateľovi vygenerovať prehľadnú a upraviteľnú dokumentáciu, ktorá môže byť určená ako popisy kódu alebo projektov v repozitároch ako známy README súbor. Markdown výstup poskytuje tiež možnosť úpravy výslednej dokumentácie, pridanie obrázkov, tabuliek alebo aj úpravy štýlu samotného textu. Na jeho úpravu nie je potrebný žiadny špeciálny nástroj, dá sa upraviť po otvorení v textovom dokumente.

### 3.3.3 Konfiguračný súbor

Pre možnosť upravenia vstupného XML súboru bol navrhnutý konfiguračný súbor, ktorý je možné zadať ako voliteľný argumenty pri spúšťaní nástroja. Tento konfiguračný súbor

umožňuje pomocou navrhnutých príkazov editovať XML súbor, alebo zadefinovať titulnú stranu dokumentácie. V rámci úprav XML dokumentu sa jedná o vymazanie sekvencie riadkov, alebo zmena mena elementu za jedno z mien navrhnutých elementov pre definíciu štýlu výslednej dokumentácie. Je potrebné poznamenať, že zmeny XML vstupu sa vykonávajú mimo tohto vstupného súboru, takže pôvodné XML zostane bez zmeny.

#### Štruktúra

Na zabezpečenie správnej funkcionality nástroja, je potrebné aby bola dodržaná presná štruktúra konfiguračného súbora.

- **Výstupný formát** je nutné definovať na prvom riadku konfiguračného súboru, tento riadok je v súbore povinný a musí to byť jeden z výstupných formátov podporovaným nástrojom.
- **Titulnú stranu** je možné definovať následne za uvedením výstupného formátu. Do titulnej strany je možné uviesť názov organizácie, názov a podnázov dokumentácie, meno autora a dátum vytvorenia.
- **Ostatné pravidlá** nasledujú po definícií titulnej strany. V rámci týchto pravidiel je možné zadať, ktoré riadky z dokumentu treba vymazať, alebo zmeniť názov niektorým elementom v XML súbore. Poradie pravidiel nie je nijako určené, požadované je aby bolo každé pravidlo na samostatnom riadku.

Ďalšie podrobnosti o presnej štruktúre jednotlivých pravidiel sú popísané v časti 4.2.1.

# Kapitola 4

# Realizácia

V tejto kapitole bude detailne popísaná implementácia navrhnutého nástroja. Bude popísaný spôsob ktorým bolo riešené generovanie výstupných súborov.

## 4.1 Štruktúra programu

Kvôli komplexnosti programu a požiadavke na jeho ľahké používanie bolo vhodnejším riešením kód štrukturovať do jedného zdrojového súboru, kde hlavná logika programu je vykonávaná prostredníctvom funkcií a procedúr. Rozhodnutie zjednotiť kód do jedného zdrojového súboru a systematicky ho rozdeliť do funkcií a procedúr, ako aj využiť prvky objektovo orientovaného programovania, bolo podmienené snahou o vytvorenie užívateľsky ľahko použiteľného nástroja, kde sa nemusí zapodievať veľkými zložkami so zdrojovým kódom a dbať na to, aby mal všetky potrebné moduly pri sebe. Program slúži pre koncového užívateľa, ktorého potrebou je dostať ako výstup kvalitnú technickú dokumentáciu. Užívateľa tohto programu nezaujíma obsah zdrojového súboru, a práve pre to bolo zvolené takéto riešenie.

V rámci členenia kódu do jedného zdrojového súbora bola tiež snaha o zvýšenie prehľadnosti, udržateľnosti a použiteľnosti programu, čo sa v kóde ktorý obsahuje veľký počet riadkov stáva čoraz náročnejším, a udržiavať prehľad o tom, ako jednotlivé časti vzájomne interagujú a aké sú ich funkcionálne súvislosti sa stáva problematické. Štruktúrovaním kódu do logických celkov, ktoré tvoria funkcie a procedúry, je umožnené ľahšej navigácií v rámci kódu, čomu pomáha aj dostatočné množstvo komentárov a záchytných bodov o ktoré je možné sa oprieť pri pochopení funkcionality. V prípade snahy o rozšírenie funkcionality je možné implementovať novú funkcionalitu do jednotlivých tried alebo funkcií, či aj možnosť vytvoriť úplne nových tried. Navyše, zvýšenie modularity kódu a využitie prvkov objektovo orientovaného programovania znamená, že užívatelia môžu ľahko prispôsobiť či rozšíriť funkcionalitu aplikácie podľa svojich potrieb, čím sa zvyšuje aj ich spokojnosť a efektivita práce s aplikáciou.

### Štruktúra funkcií programu

Logika programu je rozdelená do týchto funkcií, ktoré zabezpečujú celkovú funkcionalitu programu:

- `generate_pdf` - funkcia na generovanie PDF výstupu,
- `generate_uppaal` - funkcia na generovanie UPPAAL modelov,

- `generate_latex` - funkcia na generovanie Latex výstupu,
- `generate_makdown` - funkcia na generovanie Markdown výstupu,
- `apply_config_rules` - funkcia na aplikovanie príkazov zadaných v konfiguračnom súbore,
- `parse_config_file` - funkcia na spracovanie konfiguračného súboru,
- `check_arguments` - funkcia na spracovanie argumentov programu,
- `check_XML_input` - funkcia na kontrolu vstupného XML súboru.

## 4.2 Implementácia nástroja

Implementácia programu `xmlTechDocGen` bola realizovaná v programovacom jazyku Python. Zdrojový súbor sa nachádza v priečinku `code`. V priečinku source sa okrem zdrojového kódu nachádzajúce priečinky `other_examples` a `testing_examples`, v ktorých sa nachádzajú vzorové vstupné XML a SVD súbory a tiež testovacie prípady. Program sa spúšťa v príkazovom riadku zadaním argumentov.

### 4.2.1 Argumenty programu

Argumenty programu sú spracované vo funkcií `check_arguments`, ktorá postupne kontroluje počet zadaných argumentov a následne ich korektnú štruktúru. Ďalšou úlohou tejto funkcie je uložiť potrebné údaje pre ich ďalšie použitie.

Všetky argumenty potrebné pre spustenie programu sú povinné. Kostra volania programu je následovná:

```
python .\xmlTechDocGen.py <xml_file> <output_file> <config.txt>/
                          <outputFormath>
```

Za uvedené argumenty v kostre spúšťacieho príkazu je nutné uviesť následovné hodnoty:

- `<xml_file>` - názov, poprípade cesta k vstupnému XML súboru.
- `<output_file>` - požadovaný názov, poprípade cesta k výstupnému XML súboru.
- `<config.txt>/<outputFormath>` - cesta ku konfiguračnému súboru alebo formát výstupného súboru.

#### Argument `<outputFormath>`

Za argument `<outputFormath>` je potrebné dosadiť jednu z nasledujúcich hodnôt:

- `pdf` - pre výstup vo formáte PDF.
- `tex` - pre výstu vo formáte Latex.
- `md` - pre výstup vo formáte Markdown.
- `uppaal <outputFormath>` - pre súbory obsahujúce definíciu systémov v reálnom čase, ktoré je treba vygenerovať v prostredí UPPAAl. Za hodnotu `uppaal` je potrebné uviesť ešte výsledný výstupný formát (`pdf`, `tex`, `md`).

Príklady korektne zadaných argumentov:

```
python .\xmlTechDocGen.py vstup.xml vystup.pdf pdf
python .\xmlTechDocGen.py model.xml model.pdf uppaal pdf
python .\xmlTechDocGen.py vstup.xml vystup.pdf config.txt
```

### Príkazy konfiguračného súboru

V rámci argumentov ktoré spúšťajú program je možné zadať aj konfiguračný súbor. Tento súbor má špecifickú štruktúru, ktorá musí byť dodržaná.

Na prvom riadku je potreba uviesť výstupný formát dokumentácie rovnako ako pri zadávaní argumentov v príkazovom riadku. Jedná sa teda o možnosti:

- `pdf`,
- `tex`,
- `md`,
- `uppaal <outputFormath>`.

Následne je možné definovať titulnú stranu generovanej dokumentácie pomocou týchto značiek:

- `title: "..."` - definovanie názvu dokumentu,
- `subtitle: "..."` - definovanie podnázvu dokumentu,
- `organisation: "..."` - definovanie organizácie alebo nadpisu v hornej časti obrazovky,
- `date: "..."` - definovanie dátumu vytvorenia dokumentu,
- `author: "..."` - definovanie autora dokumentu.

Namiesto priestoru v úvodzovkách je potreba zadať vlastné údaje. Značky označujúce titulnú stranu nie sú povinné a je teda možné ľubovolne vybrať ktorú z nich je potreba využiť.

Ako poslednú časť konfiguračného súboru tvorí sekvencia príkazov. Jedná sa konkrétne o príkaz na vymazanie konkrétnych riadkov z XML dokumentu alebo príkaz na zmenu mena elementu za iné, za účelom určenia štýlu písma v elemente. Jedná sa o konkrétne tieto príkazy:

- `take <nazov_elemntu_XML> as <nazov_elemntu_struktury>` - príkaz slúžiaci na zmenu mena elementu vo vstupnom XML dokumente. Za `<nazov_elemntu_XML>` treba dosadiť meno, elementu ktoré bude menené, a za `<nazov_elemntu_struktury>` je potreba dosadiť nové meno elementu.
- `remove from <riadokXML> to <riadokXML>` - príkaz na odstránenie riadkov z pôvodného XML, v prípade, že nejaké nie je potreba zahrnúť do technickej dokumentácie. Za `<riadokXML>` je potrebné dosadiť číslo riadka tak, aby vzniklo rozmedzie ktoré treba vymazať.

Pre upresnenie, príkazmi zadanými v konfiguračnom súbore nebudú zmenené dáta v pôvodnom XML súbore. Program si vytvára nový XML súbor aby pôvodné XML ostalo zachované.

```
pdf

title: Nazov dokumentacie
subtitle: Podnazov dokumentacie
organisation: Nazov organizacie
date: 28.4.2024
author: Simona Janosikova

take kapitolka as chapter
remove from 9 to 13
```

Výpis 4.1: Príklad korektného konfiguračného súboru.

Po priblížení princípov ako spustiť program `xmlTechDocGen` je vhodné priblížiť aj celkovú implementáciu nástroja. V následujúcej časti tejto kapitoly bude vysvetlená implementácia celého programu, od kontroly vstupného súboru až po konečné generovanie výsledných súborov.

### 4.2.2 Kontrola XML vstupu

Kontrola vstupného XML alebo CMSIS-SVD súboru prebieha vo funkcií `check_XML_input`. Táto funkcia sa v prvom kroku pokúsi otvoriť XML súbor, a tým sa skontroluje validita jeho štruktúry, či má otváracia značka aj uzatváraciu, či sa zhodujú mená elementov v otváracej a zatváracej značke, a pod.. V prípade neúspechu je program ukončený a vypísaná prislúchajúca chybová značka.

V prípade úspechu ďalej prebieha kontrola elementov `chapter`, `section`, `subsection`, `root` a `text`, ktoré slúžia pre zabezpečenie formátu dokumentu. Kontroluje sa existencia atribútu `name` elementu `chapter`, pretože tento atribút je povinný pre každý element názvu `chapter`. Ďalej je kontrolované, či elementy nie sú prázdne alebo neobsahuje iba prázdne znaky.

Táto kontrola sa vykonáva dva krát za behu programu. Prvý krát sa kontroluje vstupný súbor pri jeho spracovaní argumentov programu a druhý krát po aplikovaní pravidiel zadaných v konfiguračnom súbore. Tieto kontroly sú dve, aby bola stopercentne zaistený validný vstupný súbor, a tak zaistená plynulá práca programu.

### 4.2.3 Spracovanie konfiguračného súboru a aplikácia jeho pravidiel

Spracovanie konfiguračného súboru zabezpečuje funkcia `parse_config_file`. Na začiatku je konfiguračný súbor zbavený prebytočných bielych znakov. Tento úkon vykonáva podfunkcia `clean_config_file`, ktorej funkcionalita spočíva v prejdení každého riadku konfiguračného súboru a odstránenie bielych znakov z riadka. Po spracovaní každého riadku sú do konfiguračného súboru zapísané všetky upravené riadky, a tým pádom je pripravený na ďalšie spracovanie. Biele znaky sú mazané z konfiguračného súboru z toho dôvodu, aby sa predišlo nechcenému chovaniu programu alebo neprívetivému vzhľadu výsledného dokumentu.

V ďalšom kroku kontroly konfiguračného súboru prebieha kontrola výstupného formátu podľa požadovaných kritérií opísaných v časti 4.2.1. Ďalej prebieha načítanie dát titulnej strany, ak boli zadané, a ich uloženie do štruktúry pre možnosť neskoršieho spracovania.

Ako posledný úkon tejto funkcie je načítanie pravidiel pre úpravu vstupného súboru. Konkrétna štruktúra pravidiel je popísaná v časti 4.2.1. Tieto pravidlá sú uložené do štruktúry `additional_rules` podľa ktorej sa ďalej upravuje vstupný súbor.

Aplikácia pravidiel zadaných pomocou konfiguračného súboru je implementované vo funkcií `apply_config_rules`. Logika tejto funkcie spočíva v spracovaní jednotlivých typov pravidiel v jednom celku. Najskôr sú spracované pravidlá pre odstraňovanie riadkov v prvom kroku selekciou pravidiel zo štruktúry `additional_rules`, definujúcich ktoré riadky treba vymazať. Na základe týchto pravidiel je vytvorená štruktúra `remove_lines`, do ktorej sú zaznamenané rozmedzia čísel riadkov na vymazanie. Na základe tejto štruktúry podfunkcia `remove_lines_from_xml` vytvorí nový vstupný súbor a pridáva doňho riadky, ktoré sa majú spracovať. Týmto spôsobom sa neporuší pôvodné XML a zároveň užívateľ si nemá šancu všimnúť, že nejaký pomocný vstupný súbor vôbec vznikol, keďže je na konci programu vymazaný. Podobným spôsobom sú ďalej spracované aj pravidlá na premenovanie elementov, a to selekciou pravidiel zo štruktúry `additional_rules` definujúcich premenovanie elementov, a následne premenovanie mien elementov v pomocnom XML súbore.

### 4.2.4 Implementácia výstupných formátov

Po vykonaní spomínaných úkonov v programe prichádza rad na generovanie výstupných súborov. Každý typ výstupného formátu je generovaný v zvlášť funkcií. Každá funkcia dokáže rozlíšiť o aký typ vstupného dokumentu sa jedná, a podľa toho generuje štruktúru a formátovanie výsledného dokumentu. V nasledujúcom texte budú priblížené spôsoby implementácie generovania jednotlivých výstupných formátov.

#### UPPAAL

Generovanie systémov v reálnom čase pomocou nástroja UPPAAL je implementované vo funkcií `export_uppaal_model`. Táto funkcia prechádza vstupný XML súbor a pri každom elemente s názvom `template` volá podfunkciu `export_uppaal_model`.

Funkcia `export_uppaal_model`, ktorej ukážka je vo výpise 4.2 sa pokúša spustiť prostredie UPPAAL a to volaním príkazu v tvare:

```
java -jar \path\uppaal.jar --export templateName filename.ext [FILENAME]
```

Za jednotlivé argumenty je potrebné zadať jednotlivé údaje, a to za `\path\uppaal.jar` je potrebné zadať úplnú cestu k súboru `uppaal.jar`. Následne za prepínač `-export` je potreba zadať meno uvedené v elemente s názvom `template`, zaň meno výstupu a ako posledný argument je súbor XML, z ktorého bude model generovaný. Informácie potrebné na spustenie prostredia UPPAAL pre generovanie modelov sú predné funkcií ako argumenty.

```
def export_uppaal_model(export_specification, output, input_xml):
    try:
        uppaal_jar_path = r"\Program Files\UPPAAL-5.0.0\app\uppaal.jar"
        command = [
            "java",
            "-jar",
            uppaal_jar_path,
            "--export",
            export_specification,
            output,
            input_xml,
        ]
        subprocess.run(command, check=True)
    except subprocess.CalledProcessError as e:
        ...
```

Výpis 4.2: Spúšťanie prostredia UPPAAL pomocou python funkcie pre generovanie modelu.

Modely vygenerované prostredím UPPAAL sú uložené ako súbory s príponou `.svg`. Tieto súbory sa podľa zadaného výstupného formátu generujú do výslednej technickej dokumentácie. Implementácia generovania dokumentácie obsahujúcej aj tieto modely bude popísaná v nasledujúcich častiach.

### PDF

Generovanie PDF súborov priamo pomocou nástrojov jazyka Python je implementované vo funkcií `generate_pdf`. Najprv sa inicializujú konštanty obsahujúce údaje, ktoré sa používajú na výpočty rozloženia stránok. Samotné generovanie PDF dokumentu začína pokusom vytvoriť plátno `c` pomocou modulu `canvas.Canvas` z knižnice `reportlab`. Ak sa táto operácia nepodarí kvôli chybe oprávnenia alebo inému výnimkovému stavu, program vypíše príslušnú chybovú správu a ukončí svoje vykonávanie.

Táto funkcia disponuje dvoma implementovanými podfunkciami, a to `wrap_text` a funkciou `draw_text`. Funkcia `wrap_text`, ktorej ukážka je vo výpise 4.3, slúži na rozdelenie textu elementu na šírku textovej plochy dokumentu. Jej funkcionalita spočíva v prechádzaní slov v texte, a podľa vypočítanej dĺžky slova sa rozhoduje, či sa slovo zmestí na riadok, následne sa buď pridá na aktuálny riadok, alebo pokračuje slovo na ďalšom riadku. V tejto funkcií sa vytvára zoznam riadkov `lines`, ktorý je aj návratovou hodnotou funkcie.

```
def wrap_text(text, width, font_size, font_style="Helvetica"):
    lines = []
    current_line = ""
    current_width = 0
    words = text.split()
    for word in words:
        word_width = c.stringWidth(word, font_style, font_size)
        if current_width + word_width < width:
            current_line += " " + word
            current_width += word_width
        else:
            lines.append(current_line.strip())
            current_line = word
            current_width = word_width
    if current_line:
        lines.append(current_line.strip())
    return lines
```

Výpis 4.3: Funkcia `wrap_text`.

Funkcia `draw_text` slúži na zakreslenie textu do plátna `c`. V prvom kroku je v nej volaná funkcia `wrap_text`, a následne je kreslený text rozdelený do riadkov touto funkciou vykresľovaný na plátno. Popri vykresľovaní je neustále overované zvyšné miesto na aktuálnej strane plátna, a v prípade nedostatku miesta je vytvorená nová strana.

V prípade dostupnosti údajov o titulnej strane v premennej `titlePageData` pokračuje nástroj generovaním titulnej strany PDF dokumentu. Postupne sa nastavuje štýl a veľkosť písma pre dané údaje titulnej strany, vypočítavajú sa pozície kam vykresliť text, a následne sa vykresľujú textové prvky, ako sú názov organizácie, názov, podnázov, autor dokumentácie a dátum.

Následne nastáva generovanie výslednej dokumentácie. V prípade že vstupný súbor je vo formáte CMSIS-SVD, je prechádzaný tento súbor element po elemente, a na základe ich mien je vykreslený text do plátna. Formát a štýl textu závisí od dôležitosti elementu, a je programom predurčený. Po úspešnom spracovaní celého vstupu je vygenerovaná technická dokumentácia opisujúca mikrokontrolér.

V prípade, že sa jedná o model UPPAAL, program prechádza vstupný súbor XML a zakomponováva obrázky do výslednej dokumentácie. K týmto modelom je možné generovať aj text zadaný v elementoch `chapter`, `section`, `subsection` a `text`. Tieto elementy majú predurčenú štruktúru a štýl písma, ktoré sa nadstavujú pri každom vykresľovaní textu do plátna. Obrázky z prostredia UPPAAL sa pred vykreslením musia prekonvertovať z `.svg` na `.png` formát z toho dôvodu, že knižnica `reportlab` nepodporuje obrázky tohto typu. Prekonvertovanie obrázka prebieha príkazom `cairosvg.svg2png`, ktorý umožňuje nadstaviť aj výsledné pozadie obrázka, a tiež vzorkovaciu frekvenciu pre konvertovanie. Následne sú tiež kontrolované rozmery obrázkov, a v prípade že sa nezmestia na plátno, sú ich rozmery zmenšené. Nové rozmery obrázka sú vypočítané jednoduchým algoritmom zachovávajúcim pomer šírky a výšky obrázka. Na zmenu rozmerov bola použitá funkcie `resize` z knižnice `Image`.

Ako posledná je spracovaná možnosť, že sa jedná o vstup XML s čisto textovým obsahom, alebo s elementami vymyslenými na definovanie štruktúry. V prípade, že sa jedná

o jeden z elementov `chapter`, `section`, `subsection` a `text` je nadstavené veľkosť písma a font písma podľa štýlu elementu. V prípade, že sa jedná o element s neznámym menom, tak je generovaný ako element `text`. Na určenie štýlu textu v plátne je v rámci generovania PDF používaný príkaz `c.setFont(font_style, font_size)`. Text elementov je vykresľovaný pomocou funkcie `c.drawString(x_position, y_position, line)`, do ktorej sa zadáva `x` a `y` pozícia, na ktorú sa vykreslí daný riadok `line`.

Posledným krokom je uloženie plátna do súboru príkazom `c.save()` a následne je užívateľovi poskytnutá výsledná dokumentácia.

### Latex

Generovanie súborov typu Latex je implementované vo funkcií `generate_latex`. Výsledná dokumentácia sa pred samotným zapísaním do výsledného súboru ukladá do premennej `latex_code`, ktorá sa v poslednom kroku prepíše do výstupného súboru. Všetky príkazy sa pridávajú do premennej `latex_code` v rovnakej štruktúre akou sa píšu príkazy v online editoroch, príklad generovania titulnej strany Latex dokumentu sa nachádza vo výpise 4.4.

```
1  if titlePageData:
2      latex_code += r"""
3      \begin{titlepage}
4          \begin{center}
5              \textsc{\Huge %(organisation)s}\\[0.5em]
6              \vspace{\stretch{0.382}}
7              {\LARGE \textbf{%(title)s}\\[0.5em]
8              %(subtitle)s}
9              \vspace{\stretch{0.618}}
10         \end{center}
11         {\Large %(date)s \hfill %(author)s}
12     \end{titlepage}
13     """
14     latex_code = latex_code % titlePageData
```

Výpis 4.4: Definovanie nového Latex dokumentu.

Pred samotným generovaním dokumentácie je potrebné pre správnosť `.tex` zdrojového kódu zadefinovať dokument Latex. Konkrétne treba definovať formát stránky, kódovanie a použité balíčky. Formát strany nástroj generuje ako `documentclass[a4paper, 12pt]{report}`, kódovanie dokumentu je nadstavené na `utf8` a ďalej je definovaný priestor textovej časti. Nástrojom sú tiež pridané základné balíčky `times` a `graphicx`.

V prípade, že boli definované dáta titulnej strany, je pridaná do dokumentu tiež titulná strana. Spôsob, akým je titulná strana generovaná je ukázaný vo výpise 4.4. Následne prebieha proces generovania dokumentu.

V tejto funkcií sú tiež implementované dve pomocné funkcie s názvom `modify_tree` a `sanitize_text`, ktoré slúžia na úpravu špeciálnych znakov ako `_`, `#`, `&` alebo `%` a mnohých ďalších. Tieto znaky je treba v Latex-e zadávať pomocou príkazov. Tieto funkcie slúžia na to, aby sa tieto znaky prepísali na príslušný príkaz, aby vo výslednom Latex dokumente nevznikli žiadne chyby.

V prípade, že sa jedná o vstupný dokument typu CMSIS-SVD, tak sa vstupný súbor prechádza postupne po jednotlivých elementoch a do premennej `latex_code` sú postupne pridávané riadky Latex kódu spolu s obsahom hodnôt elementov. Štýl textu je predurčený

a používajú sa príkazy z Latex ako `\chapter`, `\section`, `\subsection` a nečíslované zoznamy `itemize`. Po spracovaní všetkých elementov je vygenerovaný `.tex` súbor, ktorý je ďalej upraviteľný v ľubovolnom editore súborov Latex.

Ak sa jedná o súbory ostatných typov, tak je prechádzaný vstupný súbor element po elemente. Štýl elementu záleží od jeho mena. V prípade generovania Latex dokumentu sú rozlišované tieto textové elementy:

- `<chapter name="...">` je generovaný ako príkaz `\chapter`,
- `<section>` je generovaný ako príkaz `\section`,
- `<subsection>` je generovaný ako príkaz `\subsection`,
- `<text>` je generovaný ako klasický text.

Ak sa v XML dokumente vyskytne element neznámeho mena, tak je generovaný ako text bez žiadneho štýlu s úmyslom prenechať štýl na prispôsobenie užívateľom.

V prípade, že sa jedná o element `template` definujúci systém v UPPAAL, tak je generovaný príslušný obrázok modelu do Latex dokumentu. Obrázok modelu vygenerovaný ako `.svg` je tiež potrebné prekonvertovať do `.png`. To prebieha podobne ako pri generovaní PDF súboru, a to pomocou funkcie `.cairosvg.svg2png`. Následne je podľa potreby obrázok zmenšený tak, aby sa zmestil na stránku. Potom je obrázok generovaný ako `figure` s príslušným komentárom `caption`. Z hľadiska modelov v UPPAAL sú tiež rozpoznávané elementy `queries` s ich príslušnými podelementami, do ktorých sú obvykle zapisované testovacie prípady modelov. V súbore popisujúci systém v reálnom čase, je tiež možné definovať ľubovolný element z navrhnutých textových elementov, ktoré sú spolu s modelmi generované.

Posledný krok generovania je export premennej `latex_code` do výstupného súboru, ktorý je potom prístupný užívateľovi.

### Markdown

Generovanie súborov Markdown je implementované vo funkcií `generate_makdown`. V tejto funkcií sa riadky kódu Markdown zapisujú do premennej `markdown_code`. V prvom kroku tejto funkcie je generovaná titulná strana v prípade, že bola definovaná v konfiguračnom súbore. Následne je tiež potreba upraviť niektoré špeciálne znaky za príkaz, ktorým ich treba v Markdown súbore zapisovať. Tieto zmeny sú prevádzané prejdením celého XML vstupu a následnou výmenou znaku za prislúchajúci príkaz.

Potom je generovaný obsah Markdown dokumentu. V prípade, že sa jedná o súbor SVD sú spracované elementy podľa ich mien. Je im priradený predurčený štýl, ktorý je možné po vygenerovaní dokumentu upraviť. Jednotlivé údaje sú zapisované do premennej `generate_makdown` ako jednotlivé riadky Markdown kódu. Na štýlovanie textu elementov boli použité typické označenia nadpisov v Markdown-e, ako `#`, `##`, `###` a `####`. Na zabezpečenie prehľadnosti boli použité tiež nečíslované zoznamy, ktoré sa vytvárajú príkazom `-` a tiež tabuľky vytvárané pomocou `|`.

V prípade ostatných typov vstupných súborov je prechádzaný vstup pomocou cyklu, v ktorom sa element po elemente detekuje jeho meno. V Markdown súbore sa vypisujú textové s následovným priradeným štýlom:

- `<chapter name="...">` je generovaný ako nadpis prvej úrovne `#`,

- `<section>` je generovaný ako nadpis druhej úrovne `##`,
- `<subsection>` je generovaný ako nadpis tretej úrovne `###`,
- `<text>` je generovaný ako klasický text.

V prípade nerozpoznania mena elementu je text elementu generovaný ako text bez štýlu a úpravy. Možnosť úpravy takéhoto textu je možné po vygenerovaní finálneho dokumentu.

Zápis modelov vygenerovaných prostredím UPPAAL prebieha po detekcií elementu s názvom `template`. Vygenerovaný model typu `.svg` nie je potreba prekonvertovať, a tak je iba priradený do `generate_makdown` príkazom pre vkladanie do obrázkov v Markodowne `!["popis"]("cesta")`. Ostatné elementy súvisiace s modelami v reálnom zahrnuté do vstupu XML, ako element `queries` s jeho podelementami sú tiež generované s príslušným štýlom. V tomto type vstupného súboru je tiež možné zahrnúť hociktorý textový element navrhnutý v rámci nástroja, ktoré sú popri modeloch generované.

Posledným krokom pri generovaní súboru Markdown je zapísanie obsahu premennej `generate_makdown` do finálneho súboru. Pred samotným zápisom je výsledný Markdown kód zbavený prebytočných bielych znakov, ktoré mohli vzniknúť pri zapisovaní príkazov ako text. Výsledný súbor je dostupný užívateľovi s možnosťou ďalších úprav.

# Kapitola 5

# Testovanie a vyhodnotenie

Cieľom tejto kapitoly je poskytnúť prehľad o spôsobe, akým bol nástroj testovaný, a vyhodnotiť jeho výkonnosť a schopnosť splniť požiadavky stanovené v počiatočných špecifikáciách. V rámci tejto kapitoly budú popísané testovacie prístupy, metódy a techniky, ktoré boli použité na overenie správneho fungovania nástroja. Okrem toho budú vyhodnotené výsledky týchto testov a bude priblížené analýza, do akej miery nástroj spĺňa stanovené kritériá a požiadavky. Vyhodnotenie nástroja poskytne dôležitý náhľad na jeho silné a slabé stránky a umožní identifikovať oblasti, v ktorých sú potrebné prípadné ďalšie vylepšenie. Celkový cieľ tejto kapitoly je poskytnúť komplexnú analýzu testovania a výkonnosti nástroja, čo pomôže lepšie porozumieť jeho schopnostiam a prínosom pre užívateľov.

## 5.1 Testovacie scenáre

Testovanie prebiehalo na testovacej sade zloženej z XML súborov rôznych typov a veľkostí. Pre každý možný formát vstupného súboru boli pripravené minimálne 3 testovacie súbory rôznej veľkosti a rozličného obsahu. Testovacie súbory sú dostupné v zložke `testing_examples`, ktorá je členená ďalej do priečinkov `uppaal_examples`, `xml_examples` a `svd_examples`. Testovacie sady boli vo veľkej miere prebraté z repozitárov na GitHub-e, z oficiálnych stránok dokumentácie alebo v prípade textových elementov boli niektoré súbory vytvorené aj ručne. Malý prehľad generovaných súborov je dostupný v prílohe C.

Ďalším experimentom testovania bolo predvedenie aplikácie piatim dobrovoľníkom vo veku od 22 do 40 rokov. Ich úlohou bolo pochopiť význam nástroja s následnou možnosťou otestovania aplikácie, či už predpripravenými alebo vlastnými vstupmi. Následne im bolo položených pár otázok týkajúcich sa nástroja, ktoré boli následne vyhodnotené.

### 5.1.1 Testovanie pomocou dátových sád

Testovanie nástroja pomocou dátových sád bolo kľúčovým pre overenie funkcionality nástroja, ale aj na overenie kvality a udržateľnosti výslednej technickej dokumentácie. Toto testovanie bolo rozdelené na 3 hlavnejšie celky so zameraním na špecifický vstupný formát. Pre každý vstupný formát boli pripravené minimálne 3 testovacie sady, niektoré spolu aj s predpripraveným konfiguračným súborom potrebným pre testovanie. Každá testovacia sada bola vygenerovaná v každom výstupnom formáte, pre porovnanie vzhľadu rôznych výstupov dokumentácie.

### UPPAAL modely

Na testovanie generovania dokumentácie obsahujúcej modely v reálnom čase generované v prostredí UPPAAL boli prichystané tieto dátové sady:

- `covid19-ctmc.xml` - obsahujúci 2 modely,
- `ball.xml` - obsahujúci 4 modely,
- `bocdpFIXED.xml` - obsahujúci 9 modelov,
- `SHS-SPORADIC-2-tasks.xml` - obsahujúci 20 modelov.

Tieto dátové sady boli prebrané z repozitárov[1] na GitHub-e alebo z oficiálnych stránok nástroja[2] a následne upravené. Tieto súbory boli vybraté na otestovanie z toho dôvodu, že každý obsahuje iný počet systémov potrebných vygenerovať v prostredí UPPAAL. S väčším množstvom generovaných systémov pribúda aj čas potrebný na generovanie nástroja. Za toto trvanie nemôže program `xmlTechDocGen`, ale prostredie UPPAAL, ktoré je programom spúšťané. V nasledujúcej tabuľke 5.1 je možné vidieť prehľad, ako dlho trvá vygenerovanie dokumentácie vzhľadom na počet modelov vo vstupnom súbore.

| **Vstupný súbor** | **Počet modelov** | **Trvanie[s]** |
|---|---|---|
| `covid19-ctmc.xml` | 2 | 11.5 |
| `ball.xml` | 4 | 27 |
| `bocdpFIXED.xml` | 9 | 56 |
| `SHS-SPORADIC-2-tasks.xml` | 20 | 125 |

Tabuľka 5.1: Porovnanie trvania programu pri generovaní modelov z prostredia UPPAAL vzhľadom na počet modelov v súbore.

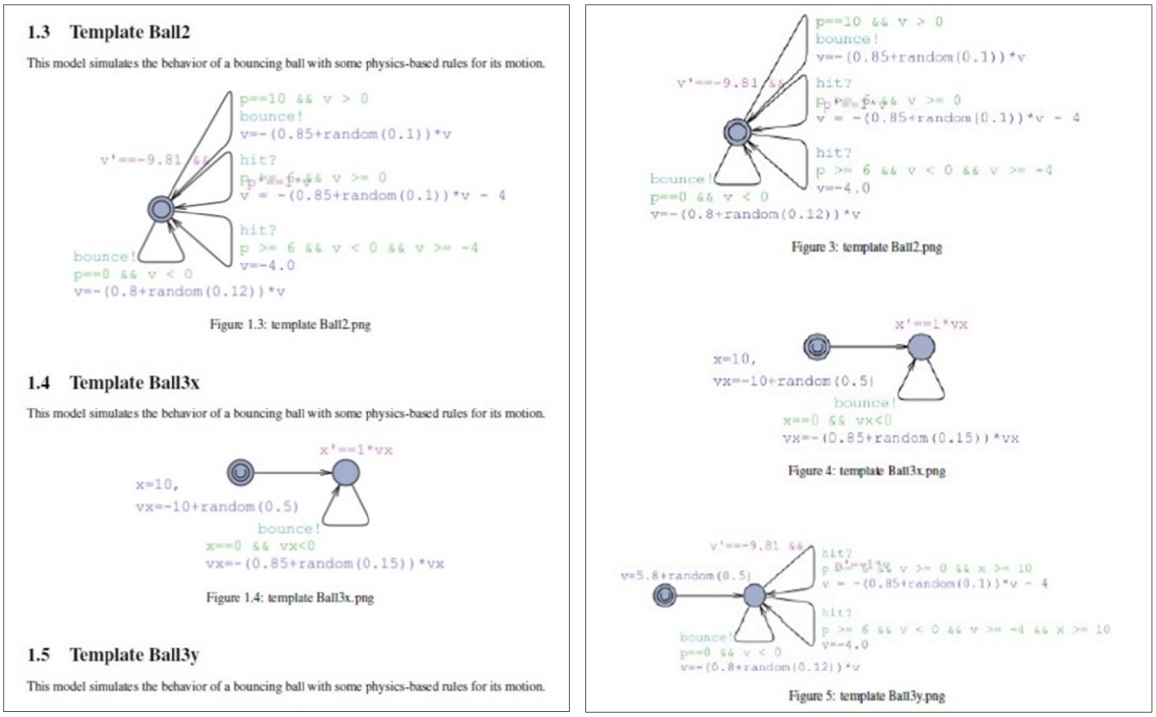
Podľa nameraných dát je zrejmé, že s pribúdajúcim množstvom modelov vzrastá aj čas potrebný na ich spracovanie. Dĺžka času ale neovplyvní chovanie zbytku programu.

Pri testovaní generovania modelov boli vytvorené aj upravené pôvodné súbory obsahujúce pridané popisy modelov pre demonštrovanie zakomponovania textových elementov do zdrojového súboru. K týmto súborom boli ručne vytvorené aj prislúchajúce konfiguračné súbory zakomponované do testovania. Dátová sada obsahujúca pridané textové elementy, ktorá vznikla na základe pôvodných modelov, je táto:

- `covid19-ctmc_text.xml`,
- `ball_text.xml`,
- `bocdpFIXED_text.xml`,
- `SHS-SPORADIC-2-tasks_text.xml`.

Pre porovnanie výzoru a kvality výslednej dokumentácia sú uvedené následovné obrázky. Je vidno, že dokumentácia s pridaným textom modelov vyzerá lepšie, úhľadnejšie a pôsobí lepším dojmom.

[1] https://github.com/DEIS-Tools/uppaal-models/tree/main/CaseStudies
[2] https://uppaal.org/




Obr. 5.1: Porovnanie pri generovaní súboru ball.xml bez pridaných popisov (vľavo), a s pridanými popismi (vpravo).

### SVD vstupy

Testovanie generovania dokumentácie, ktorá definuje systém popisujúci jadro mikrokontroléra prebiehalo bolo prevedené pomocou rôznych dátových sád. Testovacie sady boli prebrané z repozitárov[3,4] na GitHub-e a z oficiálnych stránok CMSIS-SVD[5]. Testovacie dáta sa odlišujú v množstve obsahu a zložitosti dát. Nástroj bol testovaný touto dátovou sadou skladajúca sa z týchto súborov:

- `ARMCM0.svd` o dĺžke 282 riadkov,
- `device.svd` o dĺžke 902 riadkov,
- `esp32.svd` o dĺžke 2009 riadkov.

V porovnaní s generovaním modelov prostredím UPPAAL je čas potrebný na generovanie súboru minimálny, a rozdiel pri generovaní súborov inej dĺžky je zanedbateľný, priam nepovšimnuteľný.

Počas testovania boli vygenerované všetky typy výstupných súborov a následne boli súbory generované ešte raz, ale tentokrát aj s konfiguračným súborom, v ktorom bola definovaná titulná strana.

[3] https://github.com/ARM-software/CMSIS_4/tree/master/Device/ARM/SVD
[4] https://github.com/espressif/svd/tree/main/svd
[5] https://www.keil.com/pack/doc/CMSIS/SVD/html/svd_Example_pg.html

#### XML vstupy

Testovanie vstupných súborov ostatných XML formátov prebiehalo formou vytvorenia troch rozličných s rozdielnym obsahom. Jedná sa o tieto súbory:

- `plain.xml` obsahujúci iba neznáme názvy elementov,
- `mixed.xml` obsahujúci známe aj neznáme mená elementov,
- `formath.xml` obsahujúci iba známe mená elementov.

Tieto súbory obsahujú ručne vymyslené dáta pre účely testovania. Z týchto súborov bola vygenerovaná technická dokumentácia vo formátoch všetkých výstupných súborov. Na porovnanie bol ku každému vstupnému XML súboru priradený aj konfiguračný súbor obsahujúci pravidlá upravujúce názvy elementov. Následne bola dokumentácia vygenerovaná s použitím konfiguračného súboru ako argument programu.

### 5.1.2 Užívateľské testovanie

Na otestovanie nástroja užívateľmi boli vybraný piati dobrovoľníci, ktorí boli oboznámení s funkcionalitou a spôsobom ovládania nástroja. Následne im bol prenechaní čas na spoznanie programu a možnosť skúsiť si vygenerovať technickú dokumentáciu, či už z vlastných vstupov alebo z predpripravených súborov. Po otestovaní programu užívateľmi im boli položené tieto otázky:

- *Aké sú vaše dojmy z používania programu? Čo sa vám najviac páčilo a prečo?*
- *S akými ťažkosťami ste sa stretli pri používaní programu, aké nedostatky ste identifikovali?*
- *Myslíte si, že program splnil vaše očakávania? Ak áno, prečo? Ak nie, čo by ste zmenili alebo vylepšili?*
- *Aké by ste navrhli zlepšenia alebo nové funkcie pre tento program?*
- *Ako by ste popísali svoju celkovú spokojnosť s programom na stupnici od 1 do 10?*

Po zodpovedaní všetkých otázok užívateľmi boli odpovede na otázky vyhodnotené. V nasledujúcich riadkoch bude priblížené, ako užívatelia odpovedali na tieto otázky.

**Otázka č. 1: *Aké sú vaše dojmy z používania programu? Čo sa vám najviac páčilo a prečo?***

Odpovede užívateľov na túto otázku boli kladné, nenašiel sa nikto, komu by sa nástroj nepáčil. Najčastejšie spomínané odpovede na otázku čo sa im najviac páčilo bola: efektivita a jednoduchosť používania nástroja a rýchle generovanie dokumentácie. Taktiež bola vyzdvihnutá možnosť prispôsobenia vstupu pomocou konfiguračného súboru a tiež možnosť úpravy výstupnej dokumentácie. Väčšine užívateľov sa páčilo, že generovaním dokumentácie boli extrahované dáta z XML súboru, čo im vlastne uľahčilo prácu a ďalej sa mohli zaoberať už iba úpravou štýlu dokumentácie.

**Otázka č. 2:** ***S akými ťažkosťami ste sa stretli pri používaní programu, aké nedostatky ste identifikovali?***

Problémy pri obsluhe nástroja nenastali žiadne. Užívatelia sa nástrojom oboznámili rýchlo a následné používanie bolo veľmi jednoduché. Jeden užívateľ mal pripomienku na ťažie naučenia sa štruktúry a príkazov v konfiguračnom súbore, avšak tento užívateľ si ich po chvíli osvojil a následná práca bola bezproblémová.

**Otázka č. 3:** ***Myslíte si, že program splnil vaše očakávania? Ak áno, prečo? Ak nie, čo by ste zmenili alebo vylepšili?***

Celá skupina užívateľov čo testovala tento nástroj odpovedala kladne. Páčilo sa im rýchla generácia dokumentácie, efektívnosť, jednoduchosť a spoľahlivosť nástroja. Viacerí povedali že by sa im takýto nástroj hodil aj v pracovnom prostredí.

**Otázka č. 4:** ***Aké by ste navrhli zlepšenia alebo nové funkcie pre tento program?***

V rámci funkcionality nástroj spĺňa všetko čo by mal. Návrhy užívateľov pozostávali prevažne ohľadom spracovania viac XML dokumentov súčasne, alebo spájať viacero vstupných súborov do jednej výslednej dokumentácie. Ďalším poznatkom bol zakomponovania viacerých štýlov textu pri generovaní dokumentácie, poprípade výber štýlu užívateľom.

**Otázka č. 5:** ***Ako by ste popísali svoju celkovú spokojnosť s programom na stupnici od 1 do 10?***

Z hľadiska celkovej spokojnosti nástroja uviedli dvaja užívatelia 8 bodov z 10, dvaja užívatelia 9 bodov z 10 a jeden 10 bodov z 10. To vytvára priemerné hodnotenie 8.8 bodov z 10. Z tohto výsledku plynie, že spokojnosť užívateľov je vynikajúca.

Po vyhodnotení výsledkov užívateľského testovania vyplýva, že užívatelia vnímajú tento nástroj pozitívne, vyhovuje im jeho funkcionalita a v celku ho ohodnotili ako funkčný a použiteľný.

## 5.2 Vyhodnotenie výsledkov

Na základe testovacích scenárov a užívateľského testovania boli zhromaždené dáta, ktoré umožnili posúdiť silné a slabé stránky nástroja a identifikovať oblasti, v ktorých je potrebné prípadné ďalšie vylepšenie. V rámci testovacích scenárov sa preukázala funkčnosť nástroja pri spracovaní rôznych typov vstupných súborov. Testovanie na základe XML súborov rôznych typov a veľkostí, ako aj testovanie na základe SVD vstupov a UPPAAL modelov poskytlo ucelený pohľad na schopnosť nástroja generovať technickú dokumentáciu podľa daných špecifikácií. Výsledky testovania ukázali efektivitu a spoľahlivosť nástroja pri generovaní dokumentácie a jeho schopnosť prispôsobiť sa rôznym vstupným formátom.

Výsledná dokumentácia generovaná nástrojom je dobre štruktúrovaná a prehľadná, čo bolo potvrdené nielen testovaním, ale aj užívateľským hodnotením. Každý generovaný dokument jasne a systematicky zobrazuje informácie obsiahnuté vo vstupných súboroch, čím uľahčuje ich interpretáciu a porozumenie. Štruktúra dokumentácie je logicky usporiadaná a umožňuje užívateľom rýchlo nájsť potrebné informácie.

Užívateľské testovanie tiež potvrdilo pozitívne vnímanie nástroja zo strany používateľov. Jednoduchosť použitia, rýchla generácia dokumentácie a možnosť prispôsobenia vstupu boli hlavnými pozitívnymi aspektmi, ktoré boli spomenuté. Užívatelia hodnotili nástroj ako funkčný a použiteľný vo svojom pracovnom prostredí.

Nástroj úspešne spĺňa všetky požiadavky stanovené voči nemu. Jeho schopnosť generovať technickú dokumentáciu podľa daných špecifikácií bola účinne demonštrovaná počas testovacích scenárov a užívateľského testovania. Okrem toho, výsledná dokumentácia je nielen funkčná, ale aj esteticky príjemná, čo zvyšuje jej hodnotu pre používateľov.

Celkovo je nástroj úspešným nástrojom na generovanie technickej dokumentácie, ktorý spĺňuje očakávania svojich užívateľov a poskytuje kvalitné výstupy. Jeho schopnosť produkovať dobre štrukturované a prehľadné dokumenty môže byť cenným prínosom pre vývojárske tímy a inžinierov.

Vyhodnotenie výsledkov testovania potvrdilo, že nástroj splnil stanovené očakávania a bol úspešne testovaný na rôznych typoch vstupných súborov. Identifikovalo tiež možné oblasti na ďalšie vylepšenie, ako je napríklad možnosť spracovania viacerých XML súborov súčasne alebo rozšírenie možností úpravy výstupnej dokumentácie. Vyhodnotenie výsledkov poskytuje celkovo ucelený pohľad na výkonnosť a použiteľnosť nástroja a jeho prínos pre užívateľov.

# Kapitola 6

# Záver

Cieľom tejto práce bolo navrhnúť a následne zostrojiť multiplatformný nástroj slúžiaci na generovanie technickej dokumentácie z XML vstupov. V rámci riešenia bolo implementované rozpoznanie viacero vstupných formátov XML. Okrem XML s ľubovoľným obsahom dokáže tento nástroj rozpoznať v XML popísané modely systémov v reálnom čase, CMSIS-SVD vstup, ale aj navrhnutý vlastný typ XML s vlastnými názvami elementov, na základe ktorých je text lepšie formátovaný.

Výsledkom produktom tejto práce je program xmlTechDocGen implementovaný v jazyku Python. Ako výstup tohto programu je prehľadná technická dokumentácia s jednotnou štruktúrou, ktorej automatické generovanie prostredníctvom tohto nástroja uľahčuje užívateľovi prácu. Za predpokladu, že užívateľ bude vyžadovať možnosť úpravy výslednej dokumentácie bol navrhnutý konfiguračný súbor, ale aj prispôsobené výstupné formáty tohto nástroja.

Prostredníctvo testovania neboli odhalené žiadne chyby a bolo preukázané, že nástrojom je možné vygenerovať kvalitnú technickú dokumentáciu, ktorá spĺňa potrebné požiadavky ako jednotnosť štýlu a prehľadnosť. Možnosť rozšírenia tohto programu by mohli spočívať v množstve vstupných súborov spracovávaných naraz, alebo širšej škále štýlov textu výslednej dokumentácie.

Touto prácou bola otvorená zaujímavá problematika ohľadom témy automatického generovania technickej dokumentácie, ktorá je z pohľadu tejto témy zaostalá a menej skúmaná. Preto by mohlo byť automatizované generovanie technickej dokumentácie z XML súborov viac preskúmané v iných výskumoch.

# Literatúra


[1] Alred, G. J., Brusaw, C. T. a Oliu, W. E. *Handbook of Technical Writing*. 9. vyd. St. Martin's Press, 2009. ISBN 978-0-312-47707-3.

[2] Archbee. *6 Business Benefits of Creating Technical Documentation* [online]. 2022. Revidované 18.03.2024 [cit. 2024-05-05]. Dostupné z: https://www.archbee.com/blog/benefits-of-technical-documentation.

[3] Archbee. *The Dos and Don'ts of Creating Technical Documentation* [online]. 2023. Revidované 18.03.2024 [cit. 2024-05-05]. Dostupné z: https://www.archbee.com/blog/technical-documentation-dos-donts.

[4] Doxygen. *Doxygen Documentation* [online]. 2023 [cit. 2024-04-16]. Dostupné z: https://www.doxygen.nl/manual/index.html.

[5] Forward, A. *Software Documentation – Building and Maintaining Artefacts of Communication*. 2002. Master of Science Thesis. University of Ottawa. Dostupné z: https://ruor.uottawa.ca/items/ba5a8cec-2cef-4cf4-9cf8-407fdb401da7.

[6] Goldberg, K. H. *XML: Visual QuickStart Guide*. 2. vyd. Peachpit Press, 2009. ISBN 978-0-321-55967-8.

[7] Harold, E. R. a Means, W. S. *XML in a Nutshell: A Desktop Quick Reference*. 4. vyd. O'Reilly, 2002. ISBN 0-596-00292-0.

[8] Hunter, D., Rafter, J., Fawcett, J. et al. *Beginning XML*. 4. vyd. Indiana: Wiley Publishing, Inc., 2007. ISBN 978-0-470-11487-2.

[9] Krosel, A. *What Is Technical Documentation? (And How To Create It)* [online]. 2023. Revidované 17.08.2023 [cit. 2024-05-05]. Dostupné z: https://www.indeed.com/career-advice/career-development/technical-documentation.

[10] Oracle. *Javadoc Guide* [online]. 2024 [cit. 2024-04-16]. Dostupné z: https://docs.oracle.com/en/java/javase/13/javadoc/javadoc.html.

[11] Linkedin. *How can you automate system documentation generation and maintenance?* [online]. 2023. Revidované 13.12.2023 [cit. 2024-05-05]. Dostupné z: https://www.linkedin.com/advice/0/how-can-you-automate-system-documentation-generation-tq3jc.

[12] Linkedin. *How do you avoid common pitfalls when writing technical documentation?* [online]. 2024. Revidované 12.04.2024 [cit. 2024-05-05]. Dostupné z: https://www.linkedin.com/advice/0/how-do-you-avoid-common-pitfalls-when-writing-1c.

[13] Soft, S. *Oxygen XML Editor User Guide* [online]. 2024 [cit. 2024-04-16]. Dostupné z: https://www.oxygenxml.com/doc/versions/26.1/ug-editor/.

[14] Sphinx. *Sphinx Documentation* [online]. 2024 [cit. 2024-04-16]. Dostupné z: https://www.sphinx-doc.org/en/master/.

[15] Olmstead, L. *12 Types of Technical Documentation +Examples (2024)* [online]. 2022. Revidované 26.12.2023 [cit. 2024-05-05]. Dostupné z: https://whatfix.com/blog/types-of-technical-documentation/.

[16] Wikipedia. *List of XML markup languages* [online]. 2024. Revidované 28.03.2024 [cit. 2024-03-16]. Dostupné z: https://en.wikipedia.org/wiki/List_of_XML_markup_languages.

[17] Consortium, W. W. W. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. 2008. Revidované 07.02.2017 [cit. 2024-03-13]. Dostupné z: https://www.w3.org/TR/REC-xml/.

[18] Consortium, W. W. W. *Extensible Markup Language (XML) 1.1 (Second Edition)* [online]. 2006. Revidované 29.09.2006 [cit. 2024-03-13]. Dostupné z: https://www.w3.org/TR/xml11/.

[19] Consortium, W. W. W. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures* [online]. 2012. Revidované 05.04.2012 [cit. 2024-04-15]. Dostupné z: https://www.w3.org/TR/xmlschema11-1/.

# Príloha A

# Obsah pamäťového média

```
text/                                   text tejto práce
|   ...
code/
|   other_examples/                     súbor obsahujúci ďalšie testovacie sady
|   |   ...
|   testing_examples/
|   |   svd_examples/                   tesovacia sada .svd vstupov
|   |   |   ARMCM0/
|   |   |   |   ...                     výsledok testovania súborom ARMCM0.svd
|   |   |   device/
|   |   |   |   ...                     výsledok testovania súborom device.svd
|   |   |   esp32/
|   |   |   |   ...                     výsledok testovania súborom esp32.svd
|   |   uppaal_examples/                tesovacia sada uppaal modelov
|   |   |   ball/
|   |   |   |   ...                     výsledok testovania súborom ball.xml
|   |   |   bocdpFIXED/
|   |   |   |                           výsledok testovania súborom
|   |   |   |   ...                         bocdpFIXED.xml
|   |   |   covid19-ctmc/
|   |   |   |                           výsledok testovania súborom
|   |   |   |   ...                         covid19-ctmc.xml
|   |   |   SHS-SPORADIC-2-tasks/
|   |   |   |                           výsledok testovania súborom
|   |   |   |   ...                         SHS-SPORADIC-2-tasks.xml
|   |   xml_examples/                   testovacia sada textových .xml vstupov
|   |   |   formath/
|   |   |   |   ...                     výsledok testovania súborom formath.xml
|   |   |   mixed/
|   |   |   |   ...                     výsledok testovania súborom mixed.xml
|   |   |   plain/
|   |   |   |   ...                     výsledok testovania súborom plain.xml
|   |   readme.md                       návod k testovaniu
|   README.md                           návod na použitie
|   xmlTechDocGen.py                    program generujúci dokumentáciu
```

# Príloha B

# Návod na inštaláciu

Ako prvé nainštalujte z oficiálnych stránok[1] Python verzie aspoň 3.11.3, ak ho ešte nemáte neinštalovaný.
Následne je potreba nainštalovať pomocou príkazov v príkazovom riadku tieto knižnice:

- **cairosvg** príkazom `pip install cairosvg`
- **reportlab** príkazom `pip install reportlab`
- **PIL** príkazom `pip install pillow`

Ďalším krokom je inštalácia prostredia UPPAAL z oficiálnych stránok[2] podľa nimi uvedeného návodu na používanie[3].
Po úspešnom nainštalovaní UPPAAL je potreba zažiadať o akademickú licenciu na tejto stránke[4]. Po získaní licencie zadajte licenčný kód do programu ako je uvedené v návode.
Ako posledné je treba skontrolovať cestu k súboru `uppaal.jar` či je zhodná s cestou použitou v programe `\Program Files\UPPAAL-5.0.0\app\uppaal.jar`. Ak sa cesta na vašom počítači nezhoduje s použitou, zmeňte na riadku 1700 použitú cestu za vašu.
Po splnení týchto krokov by mal byť nástroj pripravený k používaniu.

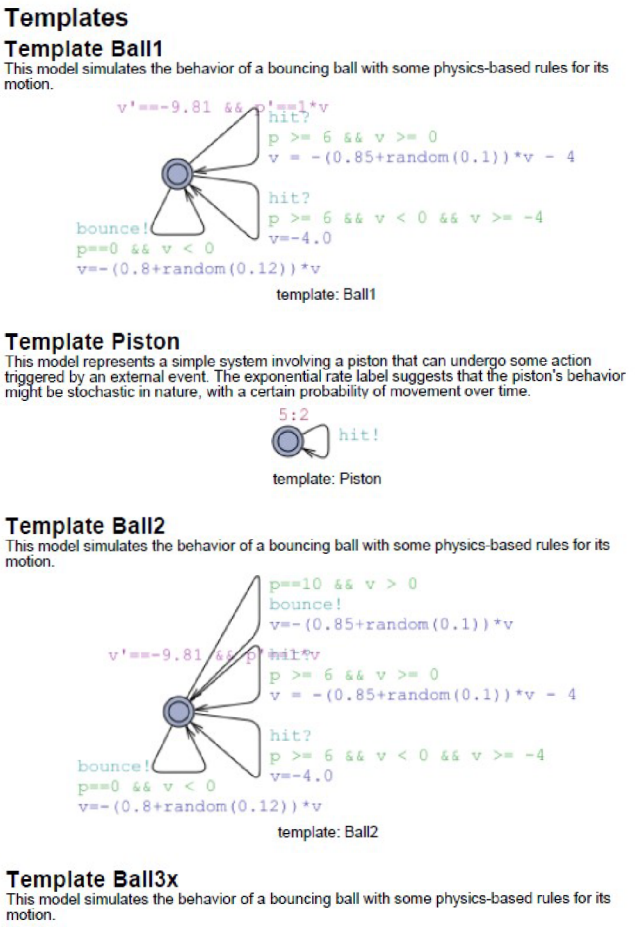
Návod na používanie bol podrobne popísaný v kapitole 4.2 spolu so správnym použitím argumentov programu a tiež štruktúrou konfiguračného súboru.

[1] https://www.python.org/downloads/
[2] https://uppaal.org/downloads/
[3] https://uppaal.org/downloads/#installation-instructions
[4] https://uppaal.veriaal.dk/academic.html

# Príloha C

# Galéria vygenerovaných súborov

## UPPAAL na .pdf




Obr. C.1: Výsledný .pdf súbor z XML vstupu popisujúci UPPAAL model.

## UPPAAL na .tex

# Chapter 1

# Templates

## 1.1 Template Ball1

This model simulates the behavior of a bouncing ball with some physics-based rules for its motion.

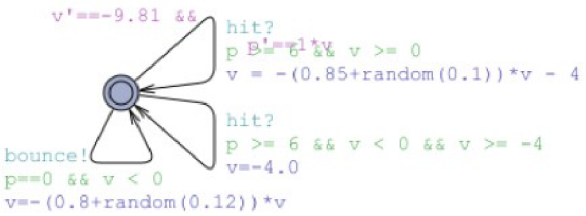



Figure 1.1: template Ball1.png

## 1.2 Template Piston

This model represents a simple system involving a piston that can undergo some action triggered by an external event. The exponential rate label suggests that the piston's behavior might be stochastic in nature, with a certain probability of movement over time.


Figure 1.2: template Piston.png

2

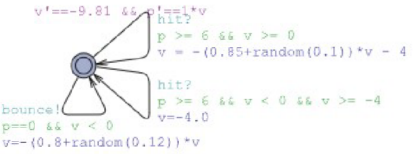
Obr. C.2: Výsledný .tex súbor z XML vstupu popisujúci UPPAAL model.

# UPPAAL na .md

## Templates

### Template Ball1

This model simulates the behavior of a bouncing ball with some physics-based rules for its motion.




template: Ball1.svg

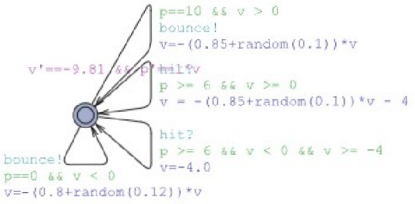
### Template Piston

This model represents a simple system involving a piston that can undergo some action triggered by an external event. The exponential rate label suggests that the piston's behavior might be stochastic in nature, with a certain probability of movement over time.


template: Piston.svg

### Template Ball2

This model simulates the behavior of a bouncing ball with some physics-based rules for its motion.




template: Ball2.svg

### Template Ball3x

Obr. C.3: Výsledný .md súbor z XML vstupu popisujúci UPPAAL model.

# CMSIS-SVD na .pdf

## ARMCM0 Peripheral Documentation

### Overview
ARM 32-bit Cortex-M3 Microcontroller based device, CPU clock up to 80MHz, etc.
Vendor: ARM Ltd.
VendorID: ARM
Series: ARMCM

### Other specifications
Address: 8 UintBits
Width: 32 bits
Size: 32 bits
Access: read-write
ResetValue: 0x00000000
ResetMask: 0xFFFFFFFF

### CPU
Name: CM0
Revision: r0p0
Endian: little
MPU Present: false
FPU Present: false
NVIC Priority Bits: 3
Vendor SysTick Config: false

### Peripherals
**Peripheral SysTick**
Description: 24Bit System Tick Timer for use in RTOS
Base Address: 0xE000E010
Address Block
Offset: 0
Size: 0x10
Usage: registers

**Registers**
**Register CSR**
Description: SysTick Control and Status Register
Address Offset: 0
Size: 32 bits
Reset Value: 0x4
Reset Mask: 0xFFFFFFFF
**Field ENABLE**
Enable SysTick Timer
Bit Offset: 0
Bit Width: 1
Access: read-write
Enumerated Values
Enumerated Value: 0
disabled
Value: 0
Enumerated Value: 1
enabled
Value: 1
**Field TICKINT**
Generate Tick Interrupt

Obr. C.4: Výsledný .pdf súbor z CMSIS-SVD vstupu.

## CMSIS-SVD na .tex

# Chapter 1

# ARMCM0 Peripheral Documentation

## 1.1 Overview

ARM 32-bit Cortex-M3 Microcontroller based device, CPU clock up to 80MHz, etc.

- Vendor: ARM Ltd.
- VendorID: ARM
- Series: ARMCM

## 1.2 Other specifications

- Address: 8 UintBits
- Width: 32 bits
- Size: 32 bits
- Access: read-write
- ResetValue: 0x00000000
- ResetMask: 0xFFFFFFFF

## 1.3 CPU

- Name: CM0
- Revision: r0p0
- Endian: little
- MPU Present: false
- FPU Present: false
- NVIC Priority Bits: 3
- Vendor SysTick Config: false

2

Obr. C.5: Výsledný .tex súbor z CMSIS-SVD vstupu.

# CMSIS-SVD na .md

## ARMCM0 Peripheral Documentation

### Overview

ARM 32-bit Cortex-M3 Microcontroller based device, CPU clock up to 80MHz, etc.

- Vendor: ARM Ltd.
- VendorID: ARM
- Series: ARMCM

#### Other specifications

- Address: 8 UintBits
- Width: 32 bits
- Size: 32 bits
- Access: read-write
- ResetValue: 0x00000000
- ResetMask: 0xFFFFFFFF

## CPU

- Name: CM0
- Revision: r0p0
- Endian: little
- MPU Present: false
- FPU Present: false
- NVIC Priority Bits: 3
- Vendor SysTick Config: false

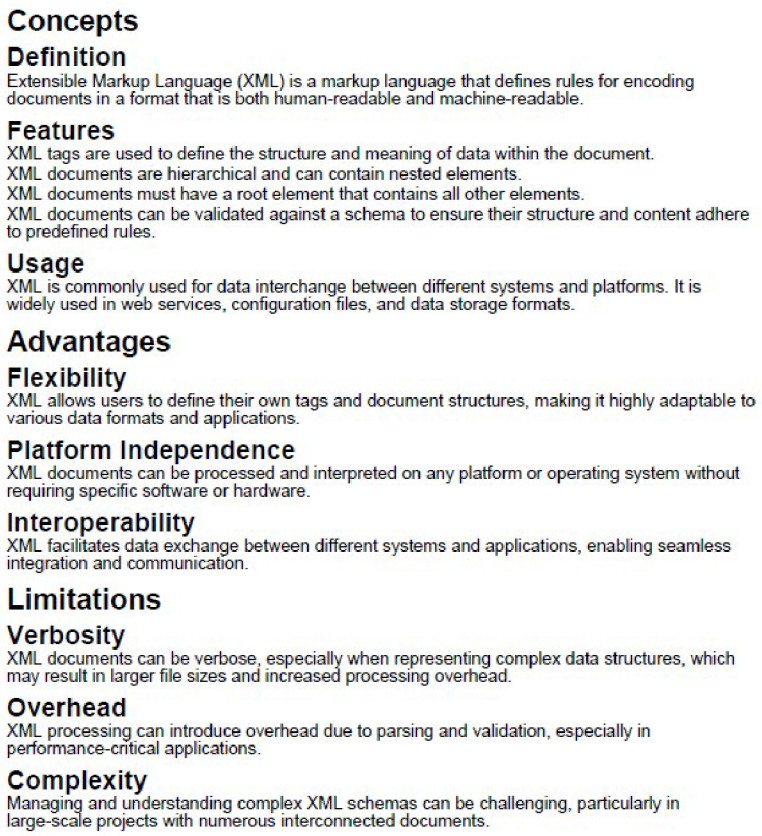
### Peripherals

#### Peripheral SysTick

- Description: 24Bit System Tick Timer for use in RTOS
- Base Address: 0xE000E010
- Address Block
    - Offset: 0
    - Size: 0x10
    - Usage: registers

Obr. C.6: Výsledný .md súbor z CMSIS-SVD vstupu.

# XML na .pdf



## Concepts

### Definition

Extensible Markup Language (XML) is a markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

### Features

XML tags are used to define the structure and meaning of data within the document.
XML documents are hierarchical and can contain nested elements.
XML documents must have a root element that contains all other elements.
XML documents can be validated against a schema to ensure their structure and content adhere to predefined rules.

### Usage

XML is commonly used for data interchange between different systems and platforms. It is widely used in web services, configuration files, and data storage formats.

## Advantages

### Flexibility

XML allows users to define their own tags and document structures, making it highly adaptable to various data formats and applications.

### Platform Independence

XML documents can be processed and interpreted on any platform or operating system without requiring specific software or hardware.

### Interoperability

XML facilitates data exchange between different systems and applications, enabling seamless integration and communication.

## Limitations

### Verbosity

XML documents can be verbose, especially when representing complex data structures, which may result in larger file sizes and increased processing overhead.

### Overhead

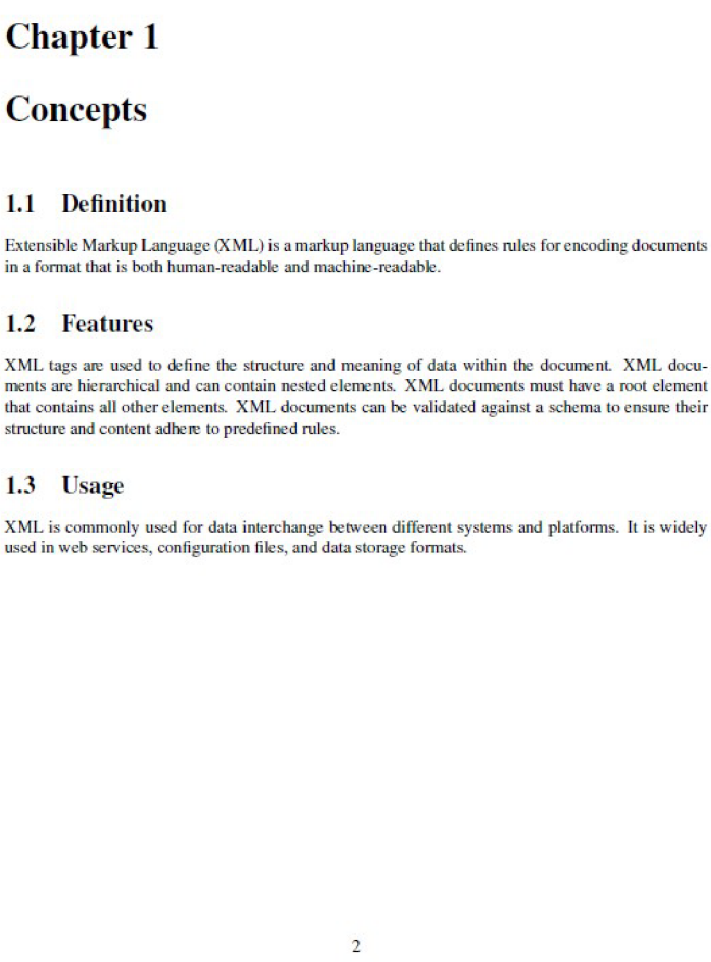
XML processing can introduce overhead due to parsing and validation, especially in performance-critical applications.

### Complexity

Managing and understanding complex XML schemas can be challenging, particularly in large-scale projects with numerous interconnected documents.

Obr. C.7: Výsledný .pdf súbor z XML vstupu.

## XML na .tex



# Chapter 1

# Concepts

## 1.1 Definition

Extensible Markup Language (XML) is a markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

## 1.2 Features

XML tags are used to define the structure and meaning of data within the document. XML documents are hierarchical and can contain nested elements. XML documents must have a root element that contains all other elements. XML documents can be validated against a schema to ensure their structure and content adhere to predefined rules.

## 1.3 Usage

XML is commonly used for data interchange between different systems and platforms. It is widely used in web services, configuration files, and data storage formats.

2

Obr. C.8: Výsledný .tex súbor z XML vstupu.

# XML na .md

## XML generation example

### with config file

Datum: 25.4.2024 Autor: Simona Janosikova

## Concepts

### Definition

Extensible Markup Language (XML) is a markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

### Features

XML tags are used to define the structure and meaning of data within the document.

XML documents are hierarchical and can contain nested elements.

XML documents must have a root element that contains all other elements.

XML documents can be validated against a schema to ensure their structure and content adhere to predefined rules.

### Usage

> XML is commonly used for data interchange between different systems and platforms. It is widely used in web services, configuration files, and data storage formats.

## Advantages

### Flexibility

XML allows users to define their own tags and document structures, making it highly adaptable to various data formats and applications.

### Platform Independence

XML documents can be processed and interpreted on any platform or operating system without requiring specific software or hardware.

### Interoperability

XML facilitates data exchange between different systems and applications, enabling seamless integration and communication.

Obr. C.9: Výsledný .md súbor z XML vstupu.