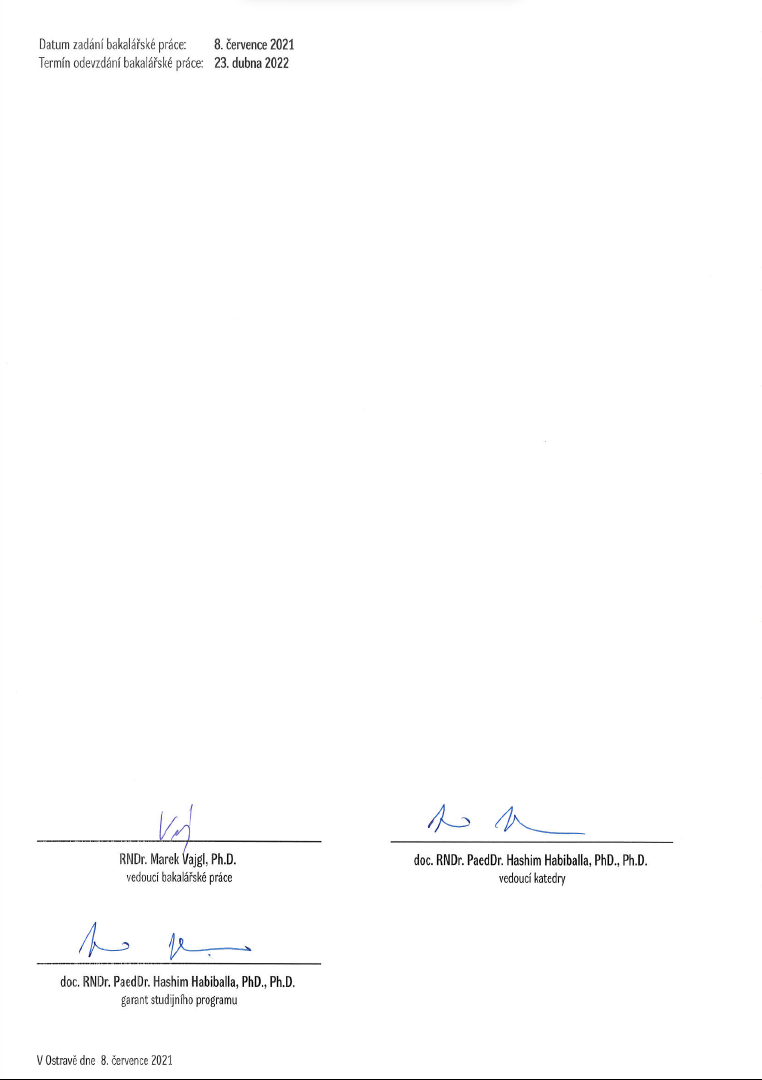
|  |
| --- |
| OSTRAVSKÁ UNIVERZITA  PŘÍRODOVĚDECKÁ FAKULTA  KATEDRA INFORMATIKY A POČÍTAČŮ |
| Plugin pro IDEA pro generování diagramu tříd do PlantUML  BAKALÁŘSKÁ PRÁCE |
| Autor práce: Marie Šrámková  Vedoucí práce: RNDr. Marek Vajgl, Ph.D. |
| 2022 |

|  |
| --- |
| UNIVERSITY OF OSTRAVA  FACULTY OF SCIENCE  DEPARTMENT OF INFORMATICS AND COMPUTERS |
| A plugin for IDEA to generate class diagram into PlantUML  BACHELOR THESIS |
| Author:  Marie Šrámková  Supervisor:  RNDr. Marek Vajgl, Ph.D. |
| 2022 |





Cílem práce je realizovat konfigurovatelný plugin pro vývojové prostředí IDEA, který bude umět převádět třídy definované v projektu do diagramu tříd jazyka PlantUML.

Řešení bude moci u otevřeného projektu vybrat oblast, která se má zahrnout do generování, a další parametry generování (vybraní členové, detailnost atd). Následně bude umět nástroj vygenerovat diagram do jazyka PlantUML. Konfigurace generování bude možno uložit.

Přínosem práce je vytvoření vlastního generovacího přístupu ze zdrojových kódu Java s možností poměrně podrobné specifikace generovaného obsahu.

Postup práce:

1) Definice cílů a analýza současného stavu

2) Definice metodiky řešení práce

3) Vytvoření vlastního řešení

4) Shrnutí vytvořeného řešení, přínosy

5) Shrnutí cílů práce, závěr

ABSTRAKT

Cílem této bakalářské práce je vytvořit konfigurovatelný plugin v prostředí IntelliJ IDEA, který bude generovat PlantUML kód pro diagram tříd jazyka UML podle zvolených parametrů. Uživatel bude moci výsledek generování uložit do souboru. Plugin bude také nabízet možnost vygenerovat a uložit konfiguraci generování, tj. nastavení generování, parametry generování. Plugin umožní uživateli zvolit umístění souboru, název souboru, balíčky, třídy a rozhraní, vnitřní třídy, atributy, metody a další parametry pro generování PlantUML kódu pro diagram tříd jazyka UML. Zvolené parametry generování si bude uživatel moci uložit pro příští generování a při dalším generování bude moci opětovně vygenerovat PlantUML kód pro diagram tříd jazyka UML z uložené konfigurace. Plugin také umožní uživateli spravovat již vzniklé konfigurace (zobrazení, smazání, změna parametrů).

*Klíčová slova:*

*(klíčová slova vypsaná na řádku, oddělená od sebe čárkami)*

**ABSTRACT**

The text of the abstract.

*Keywords:*

čestné prohlášení

Já, níže podepsaný/á student/ka, tímto čestně prohlašuji, že text mnou odevzdané závěrečné práce v písemné podobě je totožný s textem závěrečné práce vloženým v databázi DIPL2.

Ostrava dne

………………………………

podpis studenta/ky

|  |
| --- |
| Poděkování |
| Prohlašuji, že předložená práce je mým původním autorským dílem, které jsem vypracoval/a samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal/a, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.  V Ostravě dne . . . . . . . . . . . .  . . . . . . . . . . . . . . . . . .  (podpis) |

OBSAH

[ÚVOD 8](#_Toc103134815)

[1 Definice základních pojmů 9](#_Toc103134816)

[1.1 Co je to plugin 9](#_Toc103134817)

[1.2 Co znamená výraz „konfigurovatelný“ 9](#_Toc103134818)

[1.3 Prostředí IntelliJ IDEA 9](#_Toc103134819)

[2 Cíle práce a analýza současného stavu 10](#_Toc103134820)

[2.1 Cíle práce 10](#_Toc103134821)

[2.2 Analýza současného stavu 10](#_Toc103134822)

[3 Metodika řešení práce 14](#_Toc103134823)

[4 Realizace řešení práce 15](#_Toc103134824)

[4.1 Analýza uživatelů 15](#_Toc103134825)

[4.2 Požadavky – user stories 15](#_Toc103134826)

[4.3 Definice rizik 16](#_Toc103134827)

[4.4 Implementace řešení 18](#_Toc103134828)

[4.4.1 Tvorba pluginu 18](#_Toc103134829)

[4.4.2 Tvorba formulářů 18](#_Toc103134830)

[4.4.3 Generování konfiguračního souboru 18](#_Toc103134831)

[4.4.4 Generování UML diagramu podle parametrů zvolených ve formuláři 18](#_Toc103134832)

[5 Představení vytvořeného řešení 19](#_Toc103134833)

[RESUMÉ 20](#_Toc103134834)

[SUMMARY 21](#_Toc103134835)

[SEZNAM POUŽITÉ LITERATURY 22](#_Toc103134836)

[SEZNAM POUŽITÝCH SYMBOLŮ 23](#_Toc103134837)

[SEZNAM OBRÁZKŮ 24](#_Toc103134838)

[SEZNAM TABULEK 25](#_Toc103134839)

[SEZNAM PŘÍLOH 26](#_Toc103134840)

ÚVOD

Technologie se již několik let šíří tak, že zasahují téměř do každého odvětví. S rostoucím počtem použití roste také jejich komplexita. S problémem orientace v komplexních projektech se setkává většina vývojářů. Popis projektu může napomoci jak vývojářům vyznat se ve svém vlastním projektu a předejít tak možným chybám, tak i velkým firmám, uvnitř kterých kolaboruje více lidí na jednom projektu. V tomto případě může vhodný diagram ulehčit workflow mezi jednotlivými zaměstnanci, odděleními apod., a tak zlepšit chod firmy a její prosperitu. Aby se vývojáři či jiní uživatelé v tak komplexních technologiích vyznali, používají se různé způsoby, jak popsat problematiku (textové znázornění, grafické znázornění a další způsoby). Vhodný a jednoduchý způsob vyjádření problematiky je pomocí grafického znázornění – diagramů neboli grafů. Diagramy jsou kombinací textu a obrázků, což nám pomáhá se v popisu určité problematiky lépe vyznat.

Pro řešení problému vizualizace různých procesů ve vývoji softwaru byl vytvořen standardní modelovací jazyk, UML (Unified Modeling Language). Jako všechny jazyky, i tento nám pouze předepisuje syntaxi, které bychom se při práci s jazykem měli držet. Jazyk nabízí syntaxi pro tvorbu diagramů a každý z těchto diagramů má své specifické využití. Plugin je určen především pro vývojáře, kteří vyvíjí své projekty v prostředí IntelliJ IDEA v jazyce Java. Jelikož je jazyk Java objektově orientovaný jazyk, tzn. že základní entitou je třída, je diagram tříd, jež je jedním z diagramů UML, vhodný pro popsání projektů psaných v tomto jazyce.

Tato práce se tedy zabývá problematikou generování PlantUML kódu pro diagram tříd jazyka UML ze zdrojových kódů v jazyce Java pomocí konfigurovatelného pluginu v prostředí IntelliJ IDEA. Tento plugin by měl napomoci řešení výše zmíněného problému orientace v komplexním projektu tak, že si uživatel vygeneruje vlastní PlantUML kód, jež lze zobrazit ve vizuální formě jako diagram tříd a může tak vidět přehlednější grafické zobrazení svého projektu. Pro usnadnění generování je plugin konfigurovatelný, což znamená, že je možné si uložit parametry generování a jednoduše je spravovat.

1. Definice základních pojmů

Pro pochopení této bakalářské práce je potřeba si předem ujasnit pár základních pojmů, abychom se v problematice orientovali.

* 1. Co je to plugin

Plugin, nebo také plug-in, je označení pro zásuvný modul, což je software, který přidáváme k jinému, tzv. základnímu, softwaru za účelem rozšíření funkcionalit tohoto softwaru. Plugin nemůže fungovat sám o sobě, ale je vždy vyvíjen pro základní software, pro který je určen a je do něj doinstalován. Prostředí IntelliJ IDEA dává na výběr z pluginů dostupných na webových stránkách https://plugins.jetbrains.com/ [10]. Ty je možné doinstalovat přímo ve vývojovém prostředí.

* 1. Co znamená výraz „konfigurovatelný“ v kontextu práce

O konfiguraci v této práci budeme hovořit jako o nastavení. V kontextu práce je to nastavení parametrů pro generování PlantUML kódu pro diagram tříd jazyka UML, jako je např. parametr umístění souboru a parametr pojmenování souboru.

Mezi základními požadavky této práce je konfigurovatelnost za účelem:

* usnadnění opětovného generování PlantUML kódu se stejnými parametry daného projektu
* generování více PlantUML kódů daného projektu naráz

Pro tyto operace je zapotřebí konfiguraci uložit, aby bylo možné s ní dále pracovat a využít výše zmíněných funkcionalit.

Mezi dalšími výhodami bude zobrazení a správa existujících uložených konfigurací projektu, jako je jejich odstranění, nebo úprava parametrů konfigurace.

Podrobnější vysvětlení práce s pluginem v kapitole č. 5 - Představení vytvořeného řešení.

* 1. Prostředí IntelliJ IDEA

Vývojové prostředí IntelliJ IDEA vydané společností JetBrains slouží k vývoji projektů primárně v programovacím jazyce Java. Nabízí však možnost programování také v mnoha dalších programovacích jazycích jako je Kotlin, Groovy, SQL, HTML, JavaScrip a další. Prostředí programátorům pomáhá při vývoji různými nástroji, podporou jazyků a aktuálních technologií a podporou populárních framewofků, např. Spring, Spring Boot apod. Prostředí lze rozšířit o další pluginy, které umožní použití dalších přídavných funkcionalit.

Prostředí je distribuováno ve dvou verzích:

* Ultimate – placená verze, nabízí více funkcí než edice Community
* Community – zdarma k použití pro osobní rozvoj

JetBrains vydává každý rok novou verzi platformy, čímž se udržuje aktuální a nabízí programátorům podporu při potížích. Produkt IntelliJ IDEA je využíván velkým počtem uživatelů a je tedy na internetu k dispozici řada návodů, jak s prostředím pracovat.

Více informací je dostupných na webových stránkách https://www.jetbrains.com/idea/ [9].

1. Cíle práce a analýza současného stavu
   1. Cíle práce

Cílem této bakalářské práce je vytvořit konfigurovatelný plugin pro prostředí IntelliJ IDEA, který bude generovat PlantUML kód pro diagram tříd jazyka UML podle zvolených parametrů a umožní uložit parametry generování (konfiguraci) pro daný projekt. Plugin po kliknutí na složku projektu zobrazí vstupní formulář, který umožní uživateli zvolit v otevřeném projektu parametry pro generování PlantUML diagramu tříd jazyka UML a uložení konfigurace. Při uložení konfigurace generování se zvolené hodnoty uloží do souboru. Uživatel si sám zvolí, zda bude chtít vygenerovat pouze diagram nebo bude chtít také uložit parametry generování (dále jen konfiguraci). Plugin také umožní uživateli spravovat již vzniklé konfigurace (smazání, změna parametrů).

* 1. Analýza současného stavu

V úvodu jsme si vysvětlili, co je to diagram a k čemu nám slouží. Michal Obluk se problematikou diagramů okrajově zabývá v práci [1].

Pro práci bylo zvoleno vytvoření generátoru diagramu tříd PlantUML pro programovací jazyk Java v prostředí IntelliJ IDEA. Programovací jazyk Java je objektově orientovaný (více o OOP např. v práci [2]), tzn. že všechny objekty reálného světa můžeme reprezentovat jako objekty (instance třídy) a seskupovat je do tříd (entit). K popisu projektu můžeme využít více typů diagramů (diagram aktivit, diagram tříd, které jsou popsány v práci [1], sekvenční diagram a další typy diagramů). Pro práci byl zvolen diagram tříd [3], který popisuje základní stavební prvky objektů programovacího jazyka Java.

V rámci průzkumu aktuálního stavu v problematice generování diagramu tříd do PlantUML jsem se zaměřila na dva základní okruhy – analýzu již existujícího komplexního řešení a analýzu obecných metod, přístupů a nástrojů.

Z první části (analýzy existujících řešení) jsem zjistila, že již existuje nástroj pro zobrazení diagramu PlantUML ve vývojovém prostředí IntelliJ IDEA. Plugin PlantUML integration [4] umožňuje zobrazit textový soubor puml v diagramu PlantUML, tzn. v grafické podobě. Práce tento plugin nezahrnuje z důvodu ztráty přehlednosti. Pokud bychom se rozhodli vygenerovat více diagramů najednou a chtěli bychom je rovnou zobrazit, mohlo by nám to v liště otevřených souborů otevřít všechny, což v některých případech může znamenat i desítky nových záložek či i větší počet. Nevhodné by také bylo se pro každý diagram dotazovat, zda jej chceme také zobrazit. Pro větší počet generování by to znamenalo zdlouhavé zamítání či povolování. Jelikož již tento plugin existuje a je volně dostupný ve vývojovém prostředí IntelliJ IDEA a v práci není použit z výše zmíněných důvodů, poukazuji na jeho možné využití pro zobrazení námi vygenerovaného diagramu tříd.

V roce 2020 byl navržen a vydán plugin PlantUML Parser [5], jenž umožňuje generovat diagram PlantUML za pomoci výše zmíněného pluginu. Nyní už však ne na základě manuálně napsaného puml souboru, ale už přímo ve vývojovém prostředí. Plugin lze najít ve vývojovém prostředí při kliknutí na projekt. Zde najdeme možnost PluntUML Parser, jež nám otevře dialogové okno, ve kterém konkretizujeme podmínky generování. Tento plugin však nenabízí potřebné funkcionality a je nedostačující. Chybějícími funkcionalitami (chybějící funkcionality) z výše zmíněného pluginu, které tvoří také požadavky, jsou:

* volba umístění vygenerovaného PlantUML kódu pro diagram tříd a parametrů generování (dále jen konfigurací) a jejich následné uložení
* volba názvu vygenerovaného diagramu a konfiguračního souboru
* uložení konfigurace generování (parametry generování) pro jednotlivé složky/soubory/moduly/projekty
* načtení uložených konfigurací
* možnost výběru generování tříd/rozhraní
* možnost výběru generování vnitřních tříd
* definice, v jakém jazyce bude výsledný soubor vygenerován (formát „Java“, nebo formát diagramu tříd UML)
* možnost generování všech vybraných projektů najednou
* možnost generování všech projektů pro jednotlivé moduly
* možnost definovat komentář
* možnost definovat orientaci zobrazení tříd (levá/pravá)

V tabulce č. 1 jsem vybrala pár základních parametrů, které budu porovávat.

|  |  |  |
| --- | --- | --- |
| Parametry | PlantUML Parser | Vlastní plugin |
| 1. Generování PlantUML kódu pro diagramu tříd jazyka UML | Ano | Ano |
| 1. Uložení konfigurací | Ne | Ano |
| 1. Vlastní volba umístění a názvu pro diagram tříd | Ne | Ano |
| 1. Vlastní volba umístění a názvu pro soubor obsahující konfiguraci | Ne | Ano |
| 1. Zobrazení uložených konfigurací | Ne | Ano |
| 1. Odstranění uložených konfigurací | Ne | Ano |
| 1. Změna parametrů uloženého generování (konfigurace) | Ne | Ano |
| 1. Opětovné generování PlantUML kódu pro diagram tříd jazyka UML podle uložené konfigurace | Ne | Ano |

Tabulka - srovnání PluntUML Parseru s vlastním řešením

Z tabulky č. 1 vidíme, že náš plugin bude na rozdíl od existujícího řešení konfigurovatelný. Do tabulky jsem nezanášela parametry generování, ve kterých se naše řešení bude také lišit. Ty jsou uvedený v tabulce č. 2.

|  |  |  |
| --- | --- | --- |
| Parametry generování | PlantUML Parser | Vlastní plugin |
| 1. Třídy | Ne[[1]](#footnote-1) | Ano |
| 1. Rozhraní | Ne1 | Ano |
| 1. Zahrnuje balíčky | Ano | Ano |
| 1. Výběr z balíčků | Ne | Ano |
| 1. Třídní atributy | Ano1 | Ano |
| 1. Třídní metody | Ano1 | Ano |
| 1. Vnitřní třídy | Ne | Ano |
| 1. Atributy pro rozhraní | Ano1 | Ano |
| 1. Metody pro rozhraní | Ano1 | Ano |
| 1. Modifikátory pro třídy | Ne | Ano |
| 1. Modifikátory pro rozhraní | Ne | Ano |
| 1. Modifikátory pro atributy třídy | Ano | Ano |
| 1. Modifikátory pro položky rozhraní | Ano | Ano |

Tabulka - srovnání PluntUML Parseru s vlastním řešením (parametry generování)

Z tabulky č. 2 vidíme, že náš plugin je detailnější a umožňuje uživateli podrobnější výběr. V takovém případě však musíme počítat s možnými vyššími výkonnostními a také paměťovými nároky.

1. Metodika řešení práce

V této kapitole se seznámíme s teoretickou částí vývoje, kde si určíme, jak budeme postupovat při vývoji softwaru a specifikujeme si veškeré výhody a nevýhody tohoto postupu a jednotlivých kroků. Metodika řešení je tedy základní stavební jednotkou, podle které se bude práce dále odvíjet a jež nás povede po celou dobu vývoje.

Pro potřebu nalezení metodiky byl vyvinut vývojový rámec nazývaný Unified Process (jednotný proces, dále jen UP). V takovém případě se jedná o vývoj softwaru iterativně, což znamená, že je vyvíjen v cyklech (iteracích), které jsou řízeny riziky a případy užití (Use Case), jsou zaměřeny na modelování pomocí UML a v průběhu času se opakují. Mezi základní varianty UP patří RUP (Rational Unified Process = racionální jednotný proces) a OpenUP (Open Unified Process = otevřený jednotný proces).

RUP je velmi propracovaný, a tedy často komplexní UP. Zahrnuje tedy rozsáhlé plánování, analýzu a dokumentaci. Na projektu pracuje často celý tým lidí. Tento UP může být tedy pro projekty menšího rozsahu zbytečný.

Druhou možností je jednodušší OpenUP. Vychází z RUP, avšak byl zredukován sloučením několika částí RUP či jejich úplným odstraněním.[[2]](#footnote-2) Pro naše řešení postačí tato varianta pro svůj minimalismus.

Proces vývoje je podle UP rozdělen do 4 fází:

* Zahájení
* Rozpracování
* Konstrukce
* Zavedení

Fázemi lze chápat časový úsek vývoje, který se specifikuje na určitý aspekt vývoje. Každá fáze se může skládat z jedné iterace, či z více iterací. Iterace se snažíme naplánovat co nejmenší, abychom se vyvarovali vodopádovému principu vývoje a zůstali u iterativního (agilního) principu vývoje. Každá iterace obsahuje předem definované činnosti (disciplíny), kterým v každé iteraci dáváme novou váhu důležitosti a časového úsilí. Výstupem každé iterace je spustitelný a otestovaný build. Každá fáze je ukončená milníkem, který nám definuje podmínky, které musí být splněny pro přechod do následující fáze vývoje.[[3]](#footnote-3)

Pokračovat – Aplikace metodik v práci

1. Realizace řešení práce

Tato kapitola je zaměřena na počáteční části vývoje, podle kterých se budeme řídit v části implementace řešení. Počátečními částmi vývoje jsou myšleny kroky, které činíme, abychom předešly chybám a nemuseli celé řešení zahazovat a sestavovat od začátku. Tato část má v některých zdrojích svá pojmenování, v unified process (jednotý proces) se setkáme s fázemi. Počáteční části vývoje bychom mohli tedy přirovnat počáteční fázi (inception phase) a dále fázi zpracování (elaboration phase), které jsou definovány v tzv. jednotném procesu.

* 1. Analýza uživatelů

Uživateli používajícími náš plugin budou lidé, kteří budou pracovat v prostředí IntelliJ IDEA a budou potřebovat převést svůj projekt psaný v jazyce Java do vizuální formy v podobě PlantUML diagramu tříd. Ať už uživatelé pracují s prostředím na úrovni začátečníka, či profesionála, práce s pluginem by měla být jednoduchá a intuitivní, avšak plugin by měl být dostatečně komplexní, aby dokázal uspokojit požadavky uživatelů.

* 1. Požadavky – user stories

Požadavky chápeme jako specifikaci bodů, kterých se budeme snažit dosáhnout ve vývoji. Určují nám, co se od řešení očekává a stanovují, od čeho se ve vývoji budeme dále odrážet. Nedostatečná specifikace požadavků je častým důvodem neúspěchu.

//shrnout cíle – rozlišit text uml od grafického zobrazení a konkretizovat konfig. soubor

Požadavky lze specifikovat různými způsoby, např. use case diagramem, scénáři, pomocí user stories. User stories popisují požadavek z pohledu zákazníka způsobem, jako by si jej psal on sám. Požadavek však musí být minimalistický.

|  |
| --- |
| 1. Jako uživatel chci vytvořit PlantUML kód pro diagram tříd jazyka UML, abych mohl svůj projekt popsat pomocí jazyka UML a pomocí dalších nástrojů jej kdykoliv zobrazit v grafické podobě. |
| 1. Jako uživatel chci uložit PlantUML kód pro diagram tříd jazyka UML, abych s ním mohl pracovat později. |
| 1. Jako uživatel si chci zvolit umístění generovaného PlantUML kódu pro diagram tříd jazyka UML, abych mohl soubor následně separátně použít či jej přidat do projektového repozitáře. |
| 1. Jako uživatel chci zvolit, jak se bude generovaný PlantUML kód diagramu tříd jazyka UML jmenovat, abych i podle názvu souboru mohl poznat, o jaký soubor se jedná. |
| 1. Jako uživatel chci zvolit, co má PlantUML kód pro diagram tříd jazyka UML zahrnovat, abych mohl přizpůsobit obsah vygenerovaného diagramu různým situacím. |
| 1. Jako uživatel si chci uložit zvolené parametry generování (konfiguraci), abych si mohl PlantUML kód pro diagram tříd jazyka UML vygenerovat opakovaně bez opětovného volení stejných parametrů. |
| 1. Jako uživatel chci vidět výpis všech existujících uložených konfigurací daného projektu, abych je nemusel hledat v adresářové struktuře. |
| 1. Jako uživatel chci mít možnost odstranit již vytvořené konfigurace ve vývojovém prostředí, abych si udělal pořádek v adresářovém prostoru při jejich nevyužití a abych nemusel pro tuto operaci procházet adresářový prostor a hledat požadované konfigurace. |
| 1. Jako uživatel chci mít možnost znovu zobrazit parametry generování podle uložené konfigurace ve vývojovém prostředí, abych mohl provést opětovné generování bez nutnosti opakovaného vyplňování. |

* 1. Definice rizik

Při vývoji projektu je zapotřebí brát v úvahu rizika, která mohou ohrozit jeho úspěšné dokončení. Tato rizika je nutné si uvědomit již na počátku a pokusit se je redukovat. Pozdní zjištění, že se vyskytuje riziko, jež nejsme schopni vyřešit, může vést k prodloužení vývoje, či k úplnému selhání. Ve svém vývoji jsem nalezla rizika, která jsem zobrazila v tzn. risk-listu. Ten obsahuje mimo jiné také úroveň složitosti, dopad, který by riziko mohlo způsobit při jeho nevyřešení a také způsob řešení.

|  |  |  |  |
| --- | --- | --- | --- |
| Popis | Úroveň složitosti | Dopad | Řešení |
| 1. Neznalost tvorby pluginů, nedostatečné zkušenosti. | Vysoká | Velký | 1. Nalezení metody pro vytvoření pluginu. 2. Vytvoření prototypu. 3. Vytvoření finálního pluginu a následně na něm pracovat. |
| 1. Komplikované vytvoření komplexního vstupního formuláře. | Vysoká | Velký | Vytvoření prototypu vstupního formuláře se základními funkcionalitami, jenž se dále rozšíří o potřebné funkcionality až do finální podoby. |
| 1. Složité vytvoření algoritmu pro procházení adresářové struktury a vybírání .java souborů v projektu – neznalost algoritmu. | Vysoká | Velký | Navrhnout prototyp algoritmu pro procházení adresářové struktury a vybírání .java souborů z dané adresářové struktury. Rozvinout prototyp do finální podoby. |
| 1. Komplikované vytvoření algoritmu pro výběr požadovaných (ve formuláři zvolených) parametrů – princip generování. | Vysoká | Velký | 1. Analýza vstupů. 2. Filtrování vstupů. 3. Navrhnout prototyp algoritmu pro získávání všech dat souboru do formátu, se kterým se bude jednoduše pracovat. 4. Dovést prototyp do finální podoby. 5. Vybrat z dat ty, které vyhovují požadovaným parametrům. |
| 1. Velké množství možných komplikací při práci se soubory – nutné uložení a čtení souborů. | Vysoká | Velký | 1. Zjistit možné komplikace. 2. Najít řešení těchto komplikací. 3. Otestovat. |
| 1. Složitá orientace při hledání chyb a snaze navázat na řešení. | Nízká | Nízký | 1. Definovat si pravidla kódu. 2. Průběžně kontrolovat čitelnost kódu. 3. Refaktorovat nečitelný kód. |
| 1. Nefunkčnost na odlišných verzích operačních systémů či jiných verzích vývojového prostředí. | Nízká | Střední | Otestovat na různých verzích vývojového prostředí a různých zařízeních.  Otestovat vytíženost, nároky na paměť, apod. //specifikovat víc, ne apod. |
| 1. Dlouhá doba odezvy. |  |  | Otestovat časovou náročnost.  Optimalizovat. |
| 1. Uživatelé nebudou vědět, jak s pluginem pracovat a přestanou jej používat. | Nízká | Nízký | Nechat plugin otestovat uživateli, zda je práce s pluginem intuitivní (zda splňuje požadavky na UX (user experience)).  Vyslechnout a sepsat si jejich názor.  Upravit UI podle zkušeností uživatelů. |

* 1. Implementace řešení

Při implementaci řešení jsem vycházela z požadavků zákazníka a také z risk-listu, který je znázorněn v kapitole č. 4.3- Definice rizik. Po definování rizik jsem začala jednotlivá rizika minimalizovat v pořadí podle dopadu a následně dále rozvíjet řešení do finální podoby.

* + 1. Tvorba pluginu

Při tvorbě pluginu jsem vycházela z návodu popsaném na webových stránkách   
IntelliJ Platform Plugin SDK [8].

* + 1. Tvorba formulářů
    2. Generování konfiguračního souboru
    3. Generování UML diagramu podle parametrů zvolených ve formuláři

1. Představení vytvořeného řešení

//dělá to tohle takhle

RESUMÉ

//metodika funguje, aplikace funguje

SUMMARY

//zda jsem splnila požadavky

SEZNAM POUŽITÉ LITERATURY

1. **Příjmení, Jméno.** *Název knihy.* Město vydání: Vydavatelství, 2003. 123-4-56-789123-4.

2. **Příjmení1, Jméno1 a Příjmení2, Jméno2.** Název webové stránky. *Název webu.* [Online] Produkční společnost, 23. Září 2006. [Citace: 19. Září 2008.] http://www.urladresa.cz. 12-3456-789-12.

[1] OBLUK, Michal. *Modelování Business procesů s pomocí UML: BUSINESS PROCESS MODELING WITH UML*. Ostrava, 2006. Bakalářská. Ostravská univerzita. Vedoucí práce Lukasík Petr.

[2] Stuart Hirshfield and Raimund K. Ege. 1996. Object-oriented programming. ACM Comput. Surv. 28, 1 (March 1996), 253–255. DOI:https://doi.org/10.1145/234313.234415

[3] Berardi, Daniela & Calvanese, Diego & De Giacomo, Giuseppe. (2005). Reasoning on UML class diagrams. Artificial Intelligence. 168. 70-118. 10.1016/j.artint.2005.05.003.

[4] STEINBERG, Eugene, MAMONTOV, Ivan, Henady ZAKALUSKY, Max GORBUNOV, Vojtěch KRÁSA a Andrew KOROLEV, ed. PlantUML integration [online]. In: . [cit. 2021-11-16]. Dostupné z: https://plugins.jetbrains.com/plugin/7017-plantuml-integration/

[5] SHUZIJUN. *PlantUML Parser* [online]. In: . [cit. 2021-11-16]. Dostupné z: https://plugins.jetbrains.com/plugin/15524-plantuml-parser

[6] A GroupLayout Example. *Oracle Java Documentation: The Java™ Tutorials* [online]. [cit. 2022-01-03]. Dostupné z: <https://docs.oracle.com/javase/tutorial/uiswing/layout/groupExample.html>

[7] How to Use GroupLayout. *Oracle Java Documentation: The Java™ Tutorials* [online]. [cit. 2022-01-03]. Dostupné z: <https://docs.oracle.com/javase/tutorial/uiswing/layout/group.html>

[8] Creating Your First Plugin. *IntelliJ Platform Plugin SDK* [online]. [cit. 2022-04-06]. Dostupné z: <https://plugins.jetbrains.com/docs/intellij/getting-started.html>

[9] *IntelliJ IDEA: Capable and Ergonomic IDE for JVM* [online]. Na Hřebenech II 1718/10, 140 00 Prague, Czech Republic: JetBrains, 2022 [cit. 2022-05-12]. Dostupné z: <https://www.jetbrains.com/idea/>

[10] *Explore plugins for JetBrains Products* [online]. Na hřebenech II 1718/10, Prague, 14000, Czech Republic: JetBrains, 2022 [cit. 2022-05-12]. Dostupné z: https://plugins.jetbrains.com/

[11] *Lekce 1 - Úvod do metodologie vývoje softwaru* [online]. itnetwork.cz, 2022 [cit. 2022-05-12]. Dostupné z: https://www.itnetwork.cz/navrh/metodiky/uvod-do-metodologie-vyvoje-softwaru

[12] ŽÁČEK, Jaroslav. *Životní cyklus vývoje SW* [online]. [cit. 2022-05-12]. Dostupné z: https://docplayer.cz/243427-Zivotni-cyklus-vyvoje-sw-jaroslav-zacek-jaroslav-zacek-osu-cz-http-www1-osu-cz-zacek.html. Ostravská univerzita.

[13] OSIS, Janis a Uldis DONINS. Unified Process: Software Designing With Unified Modeling Language Driven Approaches. *ScienceDirect* [online]. United Kingdom: Elsevier, 2022, 2017 [cit. 2022-05-13]. Dostupné z: https://www.sciencedirect.com/topics/computer-science/unified-process

SEZNAM POUŽITÝCH SYMBOLŮ

|  |  |  |
| --- | --- | --- |
| ABC |  | Význam první zkratky. |
| B |  | Význam druhé zkratky. |
| C |  | Význam třetí zkratky. |
|  |  |  |

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

1. Nerozlišuje třídy a rozhraní a neumožňuje jejich oddělený výběr. [↑](#footnote-ref-1)
2. Více o UP na https://www.itnetwork.cz/navrh/metodiky [11], nebo v práci RNDr. Jaroslava Žáčka, Ph.D. [12]. [↑](#footnote-ref-2)
3. Více o jednotlivých fázích v práci RNDr. Jaroslava Žáčka, Ph.D. [12] a na webové stránce https://www.sciencedirect.com/topics/computer-science/unified-process [13]. [↑](#footnote-ref-3)