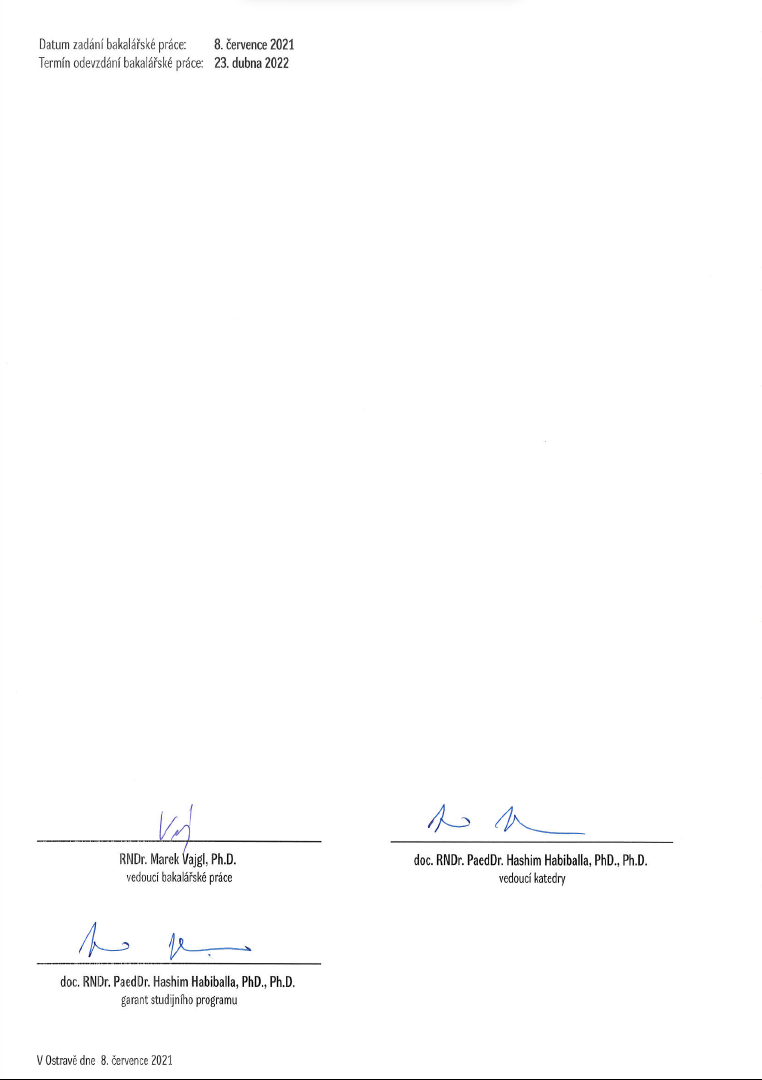
|  |
| --- |
| OSTRAVSKÁ UNIVERZITA  PŘÍRODOVĚDECKÁ FAKULTA  KATEDRA INFORMATIKY A POČÍTAČŮ |
| Plugin pro IDEA pro generování diagramu tříd do PlantUML  BAKALÁŘSKÁ PRÁCE |
| Autor práce: Marie Šrámková  Vedoucí práce: RNDr. Marek Vajgl, Ph.D. |
| 2022 |

|  |
| --- |
| 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 11](#_Toc103634089)

[1 Definice základních pojmů 12](#_Toc103634090)

[1.1 Co je to plugin 12](#_Toc103634091)

[1.2 Co znamená výraz „konfigurovatelný“ v kontextu práce 12](#_Toc103634092)

[1.3 Prostředí IntelliJ IDEA 13](#_Toc103634093)

[1.4 PlantUML 13](#_Toc103634094)

[1.4.1 Diagram tříd 14](#_Toc103634095)

[2 Cíle práce a analýza současného stavu 15](#_Toc103634096)

[2.1 Cíle práce 15](#_Toc103634097)

[2.2 Analýza současného stavu 15](#_Toc103634098)

[3 Metodika řešení práce 19](#_Toc103634099)

[3.1 Vybrané techniky fáze zahájení 20](#_Toc103634100)

[3.1 Vybrané techniky fáze rozpracování 21](#_Toc103634101)

[3.2 Vybrané techniky fáze konstrukce 22](#_Toc103634102)

[3.3 Vybrané techniky fáze zavedení 22](#_Toc103634103)

[4 Realizace řešení práce 23](#_Toc103634104)

[4.1 Vize 23](#_Toc103634105)

[4.2 Analýza uživatelů 24](#_Toc103634106)

[4.3 Požadavky – user stories 24](#_Toc103634107)

[4.4 Architektura 25](#_Toc103634108)

[4.5 Definice rizik 26](#_Toc103634109)

[4.6 Implementace řešení 29](#_Toc103634110)

[4.6.1 Prototypy 30](#_Toc103634111)

[4.6.2 Tvorba pluginu 30](#_Toc103634112)

[4.6.3 Tvorba formulářů 31](#_Toc103634113)

[4.6.1 Generování PlantUML kódu pro diagram tříd jazyka UML 33](#_Toc103634114)

[4.6.2 Generování konfiguračního souboru 34](#_Toc103634115)

[4.6.3 Nasazení 34](#_Toc103634116)

[5 Představení vytvořeného řešení 36](#_Toc103634117)

[RESUMÉ 37](#_Toc103634118)

[SUMMARY 38](#_Toc103634119)

[SEZNAM POUŽITÉ LITERATURY 39](#_Toc103634120)

[SEZNAM POUŽITÝCH SYMBOLŮ 41](#_Toc103634121)

[SEZNAM OBRÁZKŮ 42](#_Toc103634122)

[SEZNAM TABULEK 43](#_Toc103634123)

[SEZNAM PŘÍLOH 44](#_Toc103634124)

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 její vizualizace.

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. Pluginy jsou vyvíjeny vývojáři, či celými týmy vývojářů. Je možné si tedy navrhnout a vyvinout vlastní plugin dle potřeby. Postup vývoje pluginů v prostředí IntelliJ IDEA je popsán na webových stránkách https://plugins.jetbrains.com/docs/intellij/welcome.html#contribute [8].

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

O konfiguraci 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 nalezneme 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, JavaScript 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.[[1]](#footnote-1)

* 1. PlantUML

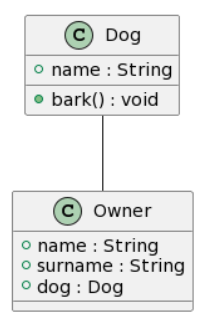
PlantUML je nástroj, který umožňuje vygenerovat z určitého textu diagram typu UML, či jiný diagram.[[2]](#footnote-2) Zdrojový kód musí pro správné generování splňovat podmínky kladené na zápis, tj. syntaxi jazyka.

Mezi základní diagramy, které umožňuje vytvořit, jsou:

* Diagram tříd
* Sekvenční diagram
* Diagram případu užití
* Diagram aktivit
* Stavový diagram
  + 1. Diagram tříd

Jedná se o grafické znázornění struktury systému vyvíjeného objektově orientovaným přístupem, kde základní entitou je třída. Entity (třídy) představují předpis pro „stejné“ objekty reálného světa, tzn. ty, které mají společné vlastnosti.

Diagram tříd může obsahovat kromě tříd také jejich vlastnosti, funkce (metody) a vazby mezi jednotlivými třídami. Příklad diagramu tříd můžeme vidět na Obrázku č. 1.



Obrázek 1 - diagram tříd

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 programovacího jazyka Java, tj. třídy a jejich komponenty.

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

Z první části (analýzy existujících řešení) jsme zjistili, 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. Tato práce tento plugin nezahrnuje z důvodu ztráty přehlednosti. Tzn. 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 je specifikováno pár základních parametrů, které budeme porovnávat.

|  |  |  |
| --- | --- | --- |
| 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 1 - srovnání PluntUML Parseru s vlastním řešením

Z tabulky č. 1 vidíme, že náš plugin bude na rozdíl od existujícího řešení konfigurovatelný. Do tabulky nejsou zaneseny 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[[3]](#footnote-3) | 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 2 - 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 potřeby vývoje 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) [11, 12, 13, 14].

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.[[4]](#footnote-4) Pro naše řešení postačí tato varianta pro svůj minimalismus.

Z důvodu malého rozsahu této práce je práce koncipována podle metodiky UP (iterativní vývoj, kdy cykly jsou řízeny riziky a případy užití).

Nejčastějšími problémy při vývoji softwaru jsou nedostatečná komunikace se zákazníkem, a tedy špatně specifikované požadavky, špatná komunikace mezi jednotlivými účastníky vývojářského týmu a mezi týmy samotnými, pozdní odhalení neřešitelného problému a mnoho dalších problémů. Díky tomu, že zastupuji roli několika členů vývojářského týmu, který by se za běžných okolností v praxi skládal z analytiků, architektů, vývojářů, manažerů a dalších účastníků, odpadlo riziko týkající se komunikace ve vývojářském týmu a s ním spojená rizika. Ne každé riziko se dá úplně odstranit jako tohle, avšak je nutné si ho uvědomit a udělat vše pro snížení jeho dopadu na výsledek. Na začátku kapitoly jsme si popsali UP, který by nám mohl pomoci dojít k úspěšnému dokončení vývoje. V následujících podkapitolách si popíšeme jednotlivé kroky vývoje softwaru nabízené UP, tj. vybrané principy a techniky.

Ještě než se vrhneme na vývoj, navrhneme, jak budeme dokumentovat proces vývoje, jak bude probíhat komunikace při vývoji a jaké technologie pro to zvolíme. Tohoto návrhu se budeme snažit při vývoji držet. Jednotlivé body tohoto návrhu mohou být v procesu vývoje shledány jako nevyhovující a mohou se změnit.

Pro dokumentaci využíváme GitHub [16], jež nám umožňuje vyvíjet, ukládat a udržovat software. Produkt je dostupný online, je zdarma a je používán mnoha vývojáři a firmami vyvíjejícími software. Pro časové plánování práce, která se provedla, provede a je prováděna, používáme Trello [17]. I tento produkt je dostupný online a je zdarma. Tyto dva produkty usnadňují samotnou práci při vývoji softwaru a umožňují paralelní práci vývojářského týmu.

Postup vývoje je zaznamenán pomocí road mapy. Road mapa je dokument, ve kterém si stanovíme časové milníky, kdy má být splněna určitá část vývoje od počátku vývoje až do jeho dokončení.

* 1. Vybrané techniky fáze zahájení

Již v minulých kapitolách jsme definovali problém orientace v komplexních objektově-orientovaných projektech. Zjistili jsme, zda existuje řešení problému a zdali nám vyhovuje. Po zjištění těchto informací jsme si stanovali vizi. Vize je představa budoucího stavu, která nám přináší určitý přínos a snažíme se ji naplnit. Vize vzniká při nalezení problému a snaze jej vyřešit či eliminovat.

Po ujasnění vize následují cíle. Pozor, není to totéž. Zatímco vize je abstraktní, často citově zabarvený pojem pro představu budoucího stavu, cíle již přechází do racionální roviny. Specifikují řešený problém, navrhují způsob jeho řešení a dají se rozložit na více cílů řešících určitou část celé problematiky. Cíle musí být stručné a musí z nich vyplývat záměr.

Abychom však při procesu vývoje řešili pouze náš problém a nezabíhali do jiné problematiky, je určen rozsah projektu. Mnohdy se stává, že při specifikaci požadavků, či v jiném kroku vývoje, zákazník požaduje něco navíc, co není primárním záměrem. Je nutno si určit, zda si rozšíření můžeme dovolit, či nikoli. Je zapotřebí si upřesnit hranice projektu, se kterými musí být také zákazník obeznámen a souhlasit s nimi. Ale jak? K tomu se využívají nejen odhady (odhady, kolik času zabere vývoj, odhad celkových nákladů), ale také různé analýzy, konzultace se specialisty či posouzení realizovatelnosti, tzn. zda lze za všech podmínek dosáhnout výsledku.

Mezi další kroky, které je třeba provést, než začneme vyvíjet software patří analýza uživatelů. Pokud neexistuje zákazník, který by měl zájem o vyvíjený produkt a produkt nebude mít žádný přínos (není řešením problému), je otázkou, zda se jím dále zabývat.

Následuje specifikace požadavků. Požadavky si klade zákazník a musí být schváleny všemi účastníky projektu. Specifikace požadavků by měla zahrnovat:

* Funkční požadavky
* Nefunkční požadavky
* Odvozené požadavky

Aby se předešlo neúspěchu a nespokojenosti (např., že se nestihl udělat důležitější požadavek, protože implementace požadavku s nižší váhou důležitosti zabral déle, než bylo plánováno) je vhodné dát jednotlivým požadavkům prioritu a uspořádat si je podle ní. Nezaručí nám to stoprocentní úspěšnost, avšak v časové nouzi máme jistotu, že ty nejdůležitější požadavky byly splněny.

* 1. Vybrané techniky fáze rozpracování

Nyní už se dostáváme hlouběji k návrhu řešení. V předchozí kapitole jsme si uváděli pouze hrubé odhady, analýzy a rozhodnutí, zda je projekt za daných podmínek realizovatelný. Nyní se přesouváme k technické stránce problému, čímž je specifikace vstupů a požadovaných výstupu a hrubý návrh architektury. Ten dále propracováváme k detailnějšímu, až máme přesně definováno, co se bude implementovat a jak se to bude implementovat.

Po návrhu architektury se přesuneme k nalezení rizik, definování jejich dopadů a složitostí, jejich řešení a také jejich uspořádání podle priorit. Jak jsme si popsali v úvodu kapitoly, UP mohou být řízeny mimo jiné také riziky. Vytvoříme prototyp, který bude mít za úkol všechna rizika odstranit. Toto řešení bude pouze verzí, kterou budeme rozšiřovat až do finální verze.

* 1. Vybrané techniky fáze konstrukce

Výstupem minulého kroku byl mimo jiné také prototyp aplikace. Ten byl vyvíjen na základě risk listu. V této části na onen výstup navážeme a rozšíříme si ho o požadované funkcionality podle požadavků. Při vývoji se soustředíme, aby architektura finálního implementovaného řešení byla stejná jako navrhovaná architektura v kapitole č. 3.1 - Vybrané techniky fáze rozpracování. Můžeme si ale dovolit mírné odchylky, např. pokud technologie, kterou jsme plánovali použít přestala být podporovaná, začala být zpoplatněná apod. Po celou dobu implementace klademe důraz na čitelnost kódu, tzn. dodržování pravidel jazyka.

Po splnění výše zmíněných kroků bychom měli dosáhnout finální verze softwaru.

* 1. Vybrané techniky fáze zavedení

Typů softwarů vyvíjených vývojáři, či firmami existuje celá řada. Buďto jsou to samostatně fungující softwary, které pro svou práci nepotřebují nic doinstalovávat, protože vše potřebné je zahrnuto v základním vybavením dnes již téměř každého počítače. Nebo to jsou samostatně fungující softwary, které ale pro svou práci vyžadují jiné softwary (např. databáze). Jak již bylo vysvětleno v kapitole č. 1.1 - Co je to plugin, pluginy nemohou existovat bez základního softwaru. Proto pro nasazení našeho softwaru musíme brát v úvahu veškeré možné komplikace svázané s touto informací.

Vysvětlení způsobu nasazení a použití předáváme zákazníkovi společně se softwarem. Jako dokument s popisem použití slouží tato práce, kde se představení vytvořeného řešení zabývá celá kapitola (kapitola č. 5 - Představení vytvořeného řešení). Po představení softwaru následuje rozhodnutí všech zúčastněných (vývojářský tým, zákazník), zda software splňuje požadavky a zda bylo dosaženo všeho, co bylo popsáno ve vizi.

Pokud nemáme jednoho zákazníka, který si software objednal na zakázku, ale plánujeme software nabídnout široké veřejnosti, musíme nalézt jiný způsob, jak rozhodnout, zda má software přínos a využití. V takovém případě je zapotřebí naplánovat způsob monitorování provozu a využití. Musíme však dbát na to, abychom nenarušili soukromí zákazníků a jednali stále v souladu se zákonem a podmínkami potřebného základního softwaru.

Software by měl být připraven na přidání dalších funkcionalit, úpravy (např. pokud zákazník za rok přijde s tím, že tuto funkcionalitu nevyužívá a rád by ji odstranil). Tím začne nový proces vývoje zase od začátku.

1. Realizace řešení práce

Tato kapitola popisuje proces vývoje. Proces vývoje je prováděn podle postupu uvedeném v předchozí kapitole.

* 1. Vize

Vize nám říká, jakého stavu chceme dosáhnout. Po sepsání vize není jisté, že docílíme požadovaného stavu, či nikoli. Vize je pouze prvotní impuls pro to, abychom jej rozvíjeli a pracovali na dosažení cíle.

|  |
| --- |
| Problém |
| Vývojáři pracující v jazyce Java v prostředí IntelliJ IDEA se nedokáží zorientovat v komplexním projektu, na kterém mají dále pracovat. Pokud si projekt jednou převedou do diagramu tříd, aby jej mohli vygenerovat podruhé, musí opět specifikovat stejné parametry, co má diagram zahrnovat. Tento problém vede k neefektivitě práce, jelikož musí projekt celý projít či si nastudovat dokumentaci k projektu, aby jej pochopili. |
| Řešení |
| Plugin, jež umožní vygenerovat a uložit PlantUML kód pro diagram tříd jazyka UML z otevřeného projektu psaného v jazyce Java v prostředí IntelliJ IDEA, aby si jej mohli programátoři kdykoliv zobrazit pomocí dalších nástrojů ve vizuální formě. Plugin nabízí opětovné generování se stejnými parametry bez nutnosti opětovné specifikace těchto parametrů. Tyto parametry si může uložit jako konfigurační soubor a spravovat je (zobrazit, odstranit, upravit). |
| Zainteresované strany |
| * Zákazníci (vývojáři) * Konzultant (vedoucí práce) * Vývojář softwaru |
| Požadované funkce |
| * Řešení problému přímo ve vývojovém prostředí IntelliJ IDEA * Generování PlantUML kódu pro diagram tříd jazyka UML z otevřeného projektu psaném v jazyce Java * Uložení vygenerovaného PlantUML kódu pro diagram tříd * Volba, co má generování zahrnovat * Jednoduché UI * Uložení parametrů generování (konfigurační soubor) * Opětovné generování bez nutnosti opětovného vyplňování parametrů * Správa konfigurací (zobrazení všech konfigurací daného projektu, odstranění konfigurace, úprava konfigurace) |

Tabulka 3 - vize

* 1. Analýza uživatelů

Potencionálními uživateli (zákazníky) jsou lidé, kteří pracují v prostředí IntelliJ IDEA a potřebují převést svůj projekt psaný v jazyce Java do textové podoby PlantUML pro diagram 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.

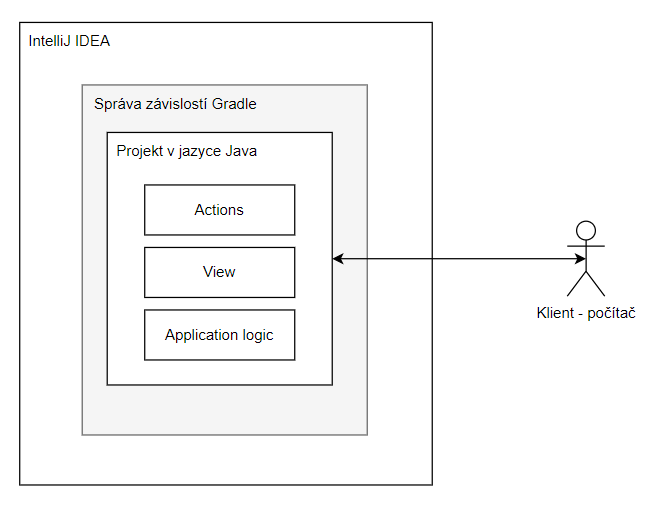
Požadavky lze specifikovat různými způsoby, např. use case diagramem, scénáři, pomocí user stories. User stories popisují požadavek z pohledu zákazníka způsobem, jako by si jej psal on sám. Požadavek však musí být minimalistický.

|  |
| --- |
| 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 později opět pracovat. |
| 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 zvolit, co má PlantUML kód pro diagram tříd jazyka UML zahrnovat, abych mohl přizpůsobit obsah vygenerovaného diagramu různým situacím. |
| 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 znovu zobrazit parametry generování podle uložené konfigurace ve vývojovém prostředí, abych mohl provést opětovné generování bez nutnosti opakovaného vyplňování. |
| 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 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. |

Tabulka 4 - požadavky

* 1. Architektura

Plugin je psán pro základní software, jímž je IntelliJ IDEA. Primárním jazykem tohoto prostředí je Java. V tomto prostředí, konkrétně v edici Ultimate, probíhá také samotný vývoj pluginu, stejně tak je vyvíjen jazykem Java. Jelikož plugin bude obsahovat mnoho závislostí, zahrnuje software Gradle (software pro automatizaci sestavování projektů) [18].



Obrázek 2 - diagram architektury

* 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. Na počátku vývoje byla definována rizika, která jsou zobrazena a popsána v tzn. risk-listu. Ten obsahuje mimo jiné také úroveň složitosti, dopad, který by riziko mohlo způsobit při jeho nevyřešení, a také způsob řešení. Abychom zjistili komplikace co nejdříve a mohli jim předejít, je třeba začít řešit rizika s velkým dopadem na úspěšnost softwaru jako první. Pokud se sejde několik problémů se stejným dopadem, seřadíme je dále podle úrovně složitosti, kdy nejsložitější řešíme opět jako první. Tímto způsobem si seřadíme celý seznam rizik a při vývoji postupujeme podle tohoto seznamu od rizika, které může nejvíc ovlivnit úspěšnost, až po riziko, které nejméně ohrožuje úspěch vývoje.

|  |  |  |  |
| --- | --- | --- | --- |
| Popis | Úroveň složitosti | Dopad | Řešení |
| 1. Neznalost tvorby pluginů, nedostatečné zkušenosti. | Vysoká | Velký | 1. Nalezení metody pro vytvoření pluginu. 2. Vytvoření prototypu podle zvolené metodiky. 3. Vytvoření finálního pluginu a následná práce na něm. |
| 1. Časově náročné 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 | 1. Navrhnout algoritmus pro procházení adresářové struktury a vybírání .java souborů z dané adresářové struktury. 2. Vytvořit prototyp tohoto algoritmu. 3. Rozvinout prototyp do finální podoby. |
| 1. Komplikované vytvoření algoritmu pro výběr požadovaných (ve formuláři zvolených) parametrů ze souborů – 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. Uvědomit si možné komplikace – správa přístupu, zda-li soubor existuje, zda pracujeme se souborem, či složkou. 2. Najít řešení těchto komplikací (řízení přístupu k souboru, při existenci stejného souboru se optat, zda jej chceme přepsat. 3. Otestovat – simulace komplikací a implementace prototypu pro jejich vyřešení, který dále dovedeme do finální podoby. |
| 1. Složitá orientace při hledání chyb a snaze navázat na řešení. | Střední | Vysoký | 1. Definovat si pravidla kódu. 2. Průběžně kontrolovat dodržování pravidel čistého kódu z 1. bodu. 3. Refaktorovat kód. |
| 1. Zákazník s pluginem neumí pracovat. | Nízký | Vysoký | 1. Při tvorbě UI se řídit pravidly UX (zkušeností zákazníků) 2. Proškolit zákazníky při předání softwaru. 3. Dát zákazníkovi dokument, ve kterém je popsáno, jak s pluginem pracovat. |
| 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 a nároky na paměť, optimalizovat. |
| 1. Dlouhá doba odezvy. |  |  | Otestovat časovou náročnost.  Optimalizovat. |

Tabulka 5 - risk list

* 1. Implementace řešení

Při implementaci řešení byl kladen důraz na dodržování pravidel pro psaní čistého a přehledného kódu. V kapitole č. 1.3 - Prostředí IntelliJ IDEA jsme si řekli, že prostředí nabízí různé nástroje pro rychlý a snadný vývoj. Za zmínku stojí jedna funkcionalita, jež napomáhá psaní čistého kódu, a to je reformátování. Reformátovat lze vybraný kód, či celý soubor. Další nápomocnou funkcionalitou je možnost refaktorování kódu. Obě tyto možnosti zpřehlední kód během pár kliknutí.

Při implementaci řešení jsme vycházeli z požadavků zákazníka a také z risk-listu, který je znázorněn v kapitole č. 4.5 - Definice rizik. Před samotnou vizí byla vytvořena road mapa (časová osa s milníky, které nám říkají, kdy má být co hotovo). Tato road mapa se podle potřeb v průběhu času měnila (podrobnější specifikace řešení, přesná data). Nyní však máme specifikovány veškeré potřebné informace pro implementaci a road mapa by se dále již neměla měnit.

Implementované funkční části pravidelně dokumentujeme a zálohujeme na GitHub a pravidelně kontrolujeme stav vývoje pomocí Trella.

* + 1. Prototypy

Po definování rizik jsme začali jednotlivá rizika minimalizovat v pořadí, v jakém jsou uvedená v seznamu (seřazena podle dopadu a složitosti) a následně dále rozvíjet řešení do finální podoby.

Verze, jíž v tomto kroku dosáhneme se nazývá prototyp. Je to tedy vzorový neúplný software, jež obsahuje funkcionality, jež shledáváme rizikovými, avšak neobsahuje veškeré funkce sepsané v požadavcích.

Nyní si detailněji popíšeme každou část, jež je součástí prototypu a jež jsme vyvíjeli na základě risk listu a jak byla později rozvinuta do finální podoby.

* + 1. Tvorba pluginu

Při tvorbě pluginu jsme vycházeli z návodu popsaném na webových stránkách   
IntelliJ Platform Plugin SDK [8]. Návod popisuje 3 různé způsoby vytvoření pluginu.

* Pomocí šablony uložené na GitHubu
* Pomocí softwaru Gradle (software pro automatizaci sestavování projektů) [18]
* Pomocí pluginu DevKit

První možnost ve své dokumentaci popisuje, že jejím hlavním cílem je urychlit postup vytváření projektu, tzn. nastavení, propojit projekt se správnými stránkami dokumentace a udržovat projekt. Jelikož s tvorbou pluginů teprve začínáme, byla by to ideální volba pro nás. Avšak nebyla využita, abychom si mohli celým postupem nastavení projektu projít sami a stejně tak si zvolit vlastní způsob dokumentace.

Již v návodu je napsáno, že je vysoce doporučeno využít druhé možnosti (vývoj za pomocí softwaru Gradle) namísto poslední (vývoj za pomoci pluginu DevKit) při vzniku nového projektu. Třetí možnost je zde popsána jako zastaralým způsobem vytváření pluginů, avšak již v návodu vytváření pomocí softwaru Gradle je uvedeno, že lze již pomocí pluginu DevKit vytvořit projekt založen na softwaru Gradle.

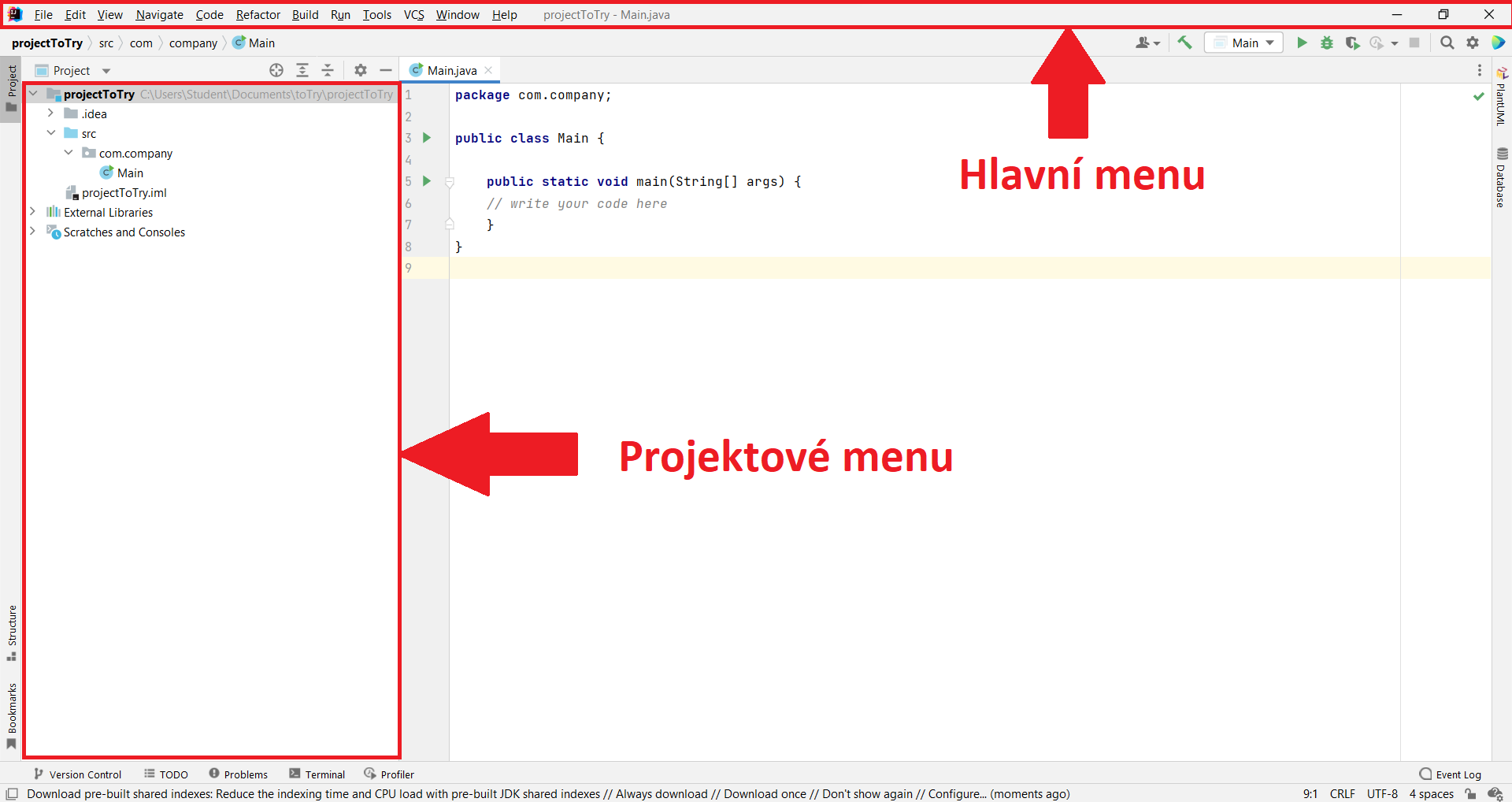
Po získání těchto informací a zhodnocení jsme se vydali cestou vývoje pomocí softwaru Gradle, přičemž jsme postupovali podle návodu. Na konci jsme získali výstup v podobě projektu, na němž jsme dále pracovali.

Řešení obsahuje dvě entity (třídy), které dědí (přebírají určité vlastnosti)[[5]](#footnote-5) ze třídy AnAction, která umožňuje pluginům přidávat své položky a služby do nabídek a panelů nástrojů. Tyto třídy jsou zaregistrovány v souboru plugin.xml a díky tomu mohou přijímat volání z platformy podle vstupů uživatele a provádět další akce.

Každá z těchto tříd se stará o svou část zpracování problematiky, kterou jsem si rozdělila na:

* generování kódu PlantUML pro diagram tříd jazyka UML
* správu konfigurací (nastavení parametrů generování)
  + 1. Tvorba formulářů

Již jsme si vytvořili plugin, jež obsahuje dvě entity, které budou provádět jisté operace. Před tím, než se tyto operace provedou však potřebujeme znát vstupy. Abychom tyto vstupy získali, musíme provést jisté kroky, abychom dosáhli interakce s uživatelem. Uživatel používá prostředí IntelliJ IDEA, která má své uživatelské rozhraní. Toto rozhraní se skládá mimo jiné z části projektového menu a z části hlavního menu (viz. Obrázek č. 3), kde se bude nacházet volba pro interakci s naším pluginem.



Obrázek 3 - panel nabídek a nástrojů prostředí IntelliJ IDEA

Interakce s uživatelem bude přes formulář. Jelikož se jedná pouze o dva formuláře, které bude plugin zobrazovat, postačí nám klihovna javax.swing. Existují samozřejmě i jiné, novodobější techniky pro tvorbu uživatelského rozhraní, ty však bývají mnohdy příliš robustní, což v našem případě není potřebné. Knihovna Swing [20], uzpůsobená pro práci na různých platformách, nabízí prvky pro tvorbu grafického uživatelského rozhraní.

Při tvorbě jsme vycházeli z detailnějších požadavků zákazníka a vše bylo zkonzultováno. Jednotlivé komponenty formuláře jsou ukázány v kapitole č. 5 - Představení vytvořeného řešení.

Tyto komponenty jsou uloženy v entitě, takže pro přidání stačí pouze tuto entitu rozšířit o další komponenty, či lze různé komponenty odstranit. To však pro to, aby se nám zobrazily ve formuláři nestačí. Další entitou jsou panely, z nichž je složeno celé okno formuláře. Prvky formuláře, jež uživatel uvidí jsou slučovány do panelů podle toho, čeho se týkají. Např. JRadioButton obsahující tlačítka pro výběr buď výchozího umístění souboru .puml, nebo určení vlastního umístění tohoto souboru jsou vedle sebe, týkají se jedné problematiky a jsou tedy uskupeny do jednoho panelu. Dalším příkladem může být panel, který obsahuje tlačítka, jež znamenají uzavření okna, potvrzení parametrů formuláře a říkají, co se má dále stát.

Panely jsou dále uskupeny v jeden velký panel, který je možné „scrollovat“ pro šetrnost velikosti okna.

Tímto způsobem je vytvořen první formulář, jež se týká volby parametrů pro generování. Po potvrzení formuláře se provedou tyto možné akce podle toho, jak jsme formulář potvrdili:

* Vygenerování a uložení pouze PlantUML kódu pro diagram tříd jazyka UML
* Vygenerování pouze konfiguračního soubor (tzn. uložení zvolených parametrů)
* Obě výše zmíněné varianty dohromady
* Zavření formuláře, aniž by se provedla jakákoliv akce

Druhý formulář je o něco jednodušší. Při volbě jeho zobrazení se projde adresářová struktura právě otevřeného projektu, vyhledají se všechny konfigurační soubory (soubory .myuml) a po otevření formuláře se nám vypíší. Každý z nich je vytvořen jako checkbox (zaškrtávací políčko), abychom mohli provádět akci nad více naráz. Tento formulář má pouze možnosti:

* Odstranit zvolené konfigurační soubory
* Vygenerovat a uložit PlantUML kód pro diagram tříd jazyka UML podle zvolených konfiguračních souborů
* Zavřít formulář, aniž by se provedla jakákoliv akce

Tímto jsme získali vstupy uživatele. O provedení akcí po potvrzení formulářů se starají již zmíněné entity popsané v kapitole č. 4.6.2 - Tvorba pluginu. Tyto entity zpracují vstupy a provádí další činnosti, které si popíšeme dále v textu, abychom získali výstup.

* + 1. Generování PlantUML kódu pro diagram tříd jazyka UML

Nyní už máme vstupy, co vše má generování zahrnovat. Podle zvoleného potvrzení formuláře také víme, jaký výstup budeme chtít ze vstupů získat. Pokud výstup zahrnuje PlantUML kód pro diagram tříd jazyka UML, provedou se následující kroky.

Projde se celá adresářová struktura patřičné složky, na kterou jsme klikli pro zobrazení formuláře. Vyberou se z ní veškeré .java soubory, které se dále budou zpracovávat.

Každý jednotlivý soubor se následně přečte a vyberou se z něj veškeré informace, jako je jeho název, v jakém balíčku se nachází, jaké má atributy apod. Tyto informace však musíme nějak uchovávat.

Pro uchování těchto informací byly vytvořeny entity:

* PackageX
* ClassX
* AttributeX
* MethodX
* InputParameterX

Této metodě se říká mapování, kdy jednotlivé výskyty uchováváme v obecném předpisu. Často se tento pojem používá v souvislosti s uchováváním dat do databáze, kdy převádíme věc uloženou v jednom formátu na jednom místě na tutéž věc v jiném formátu na jiném místě.

Každý tento soubor tedy namapujeme na výše zmíněné entity, čímž vzniknou objekty, jež uchováváme v seznamu. A za jakým účelem to vlastně převádíme? Soubor ve formátu .java může obsahovat mnoho pro nás nyní nepotřebných informací. Diagram tříd neslouží k tomu, abychom komplexní projekt zobrazili ve stejně komplexním diagramu. Diagram pouze ukazuje základní stavební prvky entit a jejich vazby. Proto je například tělo metody v diagramu tříd vynecháno. Stejně tak je tělo metody vynecháno při mapování .java souboru do výše vyjmenovaných entit. Entity zahrnují pouze ty nejdůležitější informace, jež budeme chtít v diagramu tříd zobrazit.

Seznam všech požadovaných .java souborů napamovaných na objekty vyjmenovaných entit je uchován a dále zpracován podle zvolených parametrů ve formuláři. I tyto parametry ve formě tlačítek a grafických prvků jsou špatně zpracovatelné. Proto si je namapujeme do entity jednodušší a vhodnější na zpracování, entity Config (zkratka pro termín „konfigurace“).

Po těchto krocích máme vše potřebné pro to, abychom mohli jednotlivé objekty převedené z .java souborů procházet a vybírat z nich pouze potřebné informace podle konkrétního objektu Config. Výsledné informace převádíme do formy PlantUML kódu podle pravidel jazyka a výsledný kód ukládáme. Pokud již soubor existuje, dotazujeme se uživatele, zda jej chce přepsat či nikoli.

Výsledkem těchto několik kroků je soubor obsahující PlantUML kód pro diagram tříd jazyka UML.

* + 1. Generování konfiguračního souboru

Generování konfiguračního souboru je o něco jednodušší. Již jsme jej zmínili v předchozí kapitole jako entitu Config, do kterého mapujeme parametry zvolené ve formuláři. Konkrétní výskyt této entity se pak nazývá objekt. Avšak všechny objekty, pokud bychom se je snažili uložit, uloží se nám pouze odkaz na místo v paměti, kde jsou uloženy. Proč tomu tak je nalezneme vysvětleno v knize H. Schildta. [19]

Abychom objekt převedli do čitelnější formy, používáme k tomu rozhraní JSONObject z knihovny javax.json. [21] Výsledkem je náš objekt ve formátu JSON, který ukládáme na požadované místo v paměti uživatelova zařízení. Opět se zde ptáme uživatele, zda chce soubor přepsat či nikoli, pokud již soubor existuje.

* + 1. Nasazení

Po implementaci a otestování výsledného softwaru, případném odladění chyb a opětovném otestování máme výsledný produkt. Při docílení tohoto stavu se celý vývojářský tým musí shodnout na tom, že je software připraven pro předání zákazníkovi. Zákazníka jsme si definovali již v počátcích vývoje. Naším zákazníkem není pouze jedna osoba, která si objednala software na zakázku. Našimi zákazníky budou programátoři, jež vyvíjejí své projekty v prostředí IntelliJ IDEA v jazyce Java. Jelikož tedy neznáme konkrétní uživatele, musíme se rozhodnout, jak produkt nabídnout a předávat.

Jednou z výhod vyvíjení projektu pomocí Gradle je, že výsledný produkt můžeme nasadit na JetBrains Marketplace. [10] Pokud jej chceme nasadit a nabízet, musíme jej publikovat na úložišti pluginů. Postup publikování je podrobně popsán v návodu na stránce https://plugins.jetbrains.com/docs/intellij/deployment.html#specifying-a-release-channel. [22]

1. Představení vytvořeného řešení

//dělá to tohle takhle

RESUMÉ

//metodika funguje, aplikace funguje

SUMMARY

//zda byly splněny 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

[14] BALDUINO, Ricardo. *Introduction to OpenUP (Open Unified Process)* [online]. srpen 2007, 9 [cit. 2022-05-14]. Dostupné z: https://www.eclipse.org/epf/general/OpenUP.pdf

[15] PlantUML in a nutshell. *Plantuml.com* [online]. 6023 Innovation Way, Carlsbad, California, United States: Ezoic [cit. 2022-05-14]. Dostupné z: https://plantuml.com/

[16] *GitHub* [online]. GitHub, 2022 [cit. 2022-05-15]. Dostupné z: https://github.com/

[17] *Trello* [online]. Atlassian, 2022 [cit. 2022-05-15]. Dostupné z: https://trello.com/home

[18] *Gradle* [online]. 2261 Market Street #4081, San Francisco, CA 94114, United States: Gradle, 2022 [cit. 2022-05-15]. Dostupné z: https://gradle.org/

[19] SCHILDT, Herbert. *Java: The Complete Reference, Eleventh Edition*. 11th edition. Osborne: McGraw-Hill Education, 2018. ISBN 978-1260440232.

[20] Package javax.swing. *Oracle: Java Platform Standard Edition 7 Documentation* [online]. 1993, 2020 [cit. 2022-05-16]. Dostupné z: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

[21] Interface JsonObject. *Oracle: Java Platform Standard Edition 7 Documentation* [online]. 2015 [cit. 2022-05-16]. Dostupné z: <https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html>

[22] Publishing Plugins with Gradle. *IntelliJ Platform Plugin SDK* [online]. JetBrains, 2000–2022, 22. března 2022 [cit. 2022-05-16]. Dostupné z: https://plugins.jetbrains.com/docs/intellij/deployment.html

SEZNAM POUŽITÝCH SYMBOLŮ

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

SEZNAM OBRÁZKŮ

[Obrázek 1 - diagram tříd 14](#_Toc103543219)

[Obrázek 2 - diagram architektury 26](#_Toc103543220)

SEZNAM TABULEK

[Tabulka 1 - srovnání PluntUML Parseru s vlastním řešením 17](#_Toc103543229)

[Tabulka 2 - srovnání PluntUML Parseru s vlastním řešením (parametry generování) 18](#_Toc103543230)

[Tabulka 3 - vize 24](#_Toc103543231)

[Tabulka 4 - požadavky 25](#_Toc103543232)

[Tabulka 5 - risk list 28](#_Toc103543233)

SEZNAM PŘÍLOH

1. Více informací je dostupných na webových stránkách https://www.jetbrains.com/idea/ [9]. [↑](#footnote-ref-1)
2. Seznam nabízených diagramů najdeme na webové stránce https://plantuml.com/ [15]. [↑](#footnote-ref-2)
3. Nerozlišuje třídy a rozhraní a neumožňuje jejich oddělený výběr. [↑](#footnote-ref-3)
4. Více o UP na https://www.itnetwork.cz/navrh/metodiky [11], v práci RNDr. Jaroslava Žáčka, Ph.D. [12] a na webové stránce https://www.sciencedirect.com/topics/computer-science/unified-process [13]. [↑](#footnote-ref-4)
5. Více o dědičnosti v knize Herberta Schildta. [19] [↑](#footnote-ref-5)