Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Charakteristiky OSS projektů pro selektivní těžbu a analýzu jejich dat

Plzeň 2023 Jan Černogurský

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan ČERNOGURSKÝ**
Osobní číslo:	**A21N0021P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační systémy**
Téma práce:	**Charakteristiky OSS projektů pro selektivní těžbu a analýzu jejich dat**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou open-source softwarových (OSS) projektů, využití nástrojů pro jejich řízení a možnostmi těžby jejich dat.
2. Prozkoumejte dosavadní zdroje popisující různé charakteristiky OSS projektů a jejich dat s přihlédnutím k detekci procesních anti-vzorů.
3. Navrhněte co nejkompletnější sadu charakteristik určujících vhodnost OSS projektu k těžbě a analýze jeho dat. Dále navrhněte aplikaci pro identifikaci projektů z daného zdroje na základě zadaných charakteristik.
4. Implementujte aplikaci z předchozího bodu pro jeden zdroj OSS projektů a vybranou podsadu charakteristik. Při implementaci dbejte především na její rozšiřitelnost o další zdroje dat a charakteristiky.
5. Aplikaci otestujte a ověřte vhodnost výstupu procesu identifikace projektů podpořeného aplikací pro výzkumné účely.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Pícha**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 18. května 2023

Jan Černogurský

Abstract

This thesis deals with the research and analysis of open-source software (OSS) and its potential for improving software development. The main goal is to identify the appropriate characteristics of OSS projects and their data and to perform their detailed analysis, taking into account the detection of process anti-patterns. The work focuses on the exploration of existing sources describing the characteristics of OSS projects and their data, with an emphasis on the detection of process anti-patterns. Based on this mapping, a set of characteristics is proposed, which serves as a basis for identifying suitable projects for further analysis. Part of the work is the implementation of an auxiliary application for locating projects based on a selected set of characteristics.

Abstrakt

Tato diplomová práce se zabývá výzkumem a analýzou open-source softwaru (OSS) a jeho potenciálem pro zlepšení vývoje softwaru. Hlavním cílem je identifikovat vhodné charakteristiky OSS projektů a jejich dat a provést podrobnou analýzu charakteristik projektů s přihlédnutím k detekci procesních anti-vzorů. Práce se zaměřuje na zmapování existujících zdrojů popisujících charakteristiky OSS projektů a jejich dat, s důrazem na detekci procesních anti-vzorů. Na základě tohoto mapování je navržena sada charakteristik, která slouží jako základ pro identifikaci vhodných projektů pro další analýzu. Součástí práce je implementace pomocné aplikace pro vyhledávání projektů na základě vybrané sady charakteristik.

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Petru Píchovi za odborné vedení, cenné rady a čas, který mi věnoval v průběhu celé práce.

Obsah

1 Úvod

V dnešní době je open-source software (OSS) nedílnou součástí vývoje softwaru a inovací. Stále více firem a organizací si uvědomuje jeho výhody a začíná se na OSS projekty více orientovat.[28] To zahrnuje nejen používání open-source softwaru, ale také aktivní účast v open-source komunitách, přispívání k existujícím OSS projektům, nebo vytváření vlastních OSS projektů. Přesto je systematický přístup k selektivní těžbě a analýze dat z těchto projektů stále na počátku. Analýza dat z OSS projektů má obrovský potenciál. Může napomoci ke zlepšení vývoje softwaru, například prostřednictvím identifikace osvědčených postupů (vzorů) a varování před špatnými postupy (anti-vzory).

Práce se zaměřuje na zmapování stávajících zdrojů, které popisují různé charakteristiky OSS projektů a jejich dat, s důrazem na detekci procesních anti-vzorů. Hlavním cílem je navrhnout co nejkompletnější sadu charakteristik určujících vhodnost OSS projektu k těžbě a analýze jeho dat. Tato sada charakteristik pak poslouží jako základ pro návrh aplikace určené k identifikaci projektů z daného zdroje.

Práce je rozdělena do několika částí. První část se zabývá popisem vývoje softwaru, jeho metodik, jeho druhům a následně projektovým řízením. V následující části dochází k popisu problematiky těžby OSS dat. Další část se zaměřuje na podrobnou analýzu vybraných charakteristik OSS projektů a jejich vhodnosti pro těžbu a analýzu jejich dat. Následující část je zaměřena na vybrání vhodné podmnožiny zmíněných charakteristik a návrhu aplikace k hledání projektů. Výstup této aplikace by měl sloužit jako vstup pro nástroj Software Process Anti-pattern Detector (SPADe) a měl by tak usnadnit práci s tímto nástrojem. Usnadnění spočívá v tom, že projekty, které budou sloužit jako vstup pro nástroj SPADe, již budou roztříděné podle vybraných charakteristik. Jako součást práce je realizována implementace aplikace pro jeden konkrétní zdroj OSS projektů a vybranou podsadu charakteristik. Klíčovým aspektem implementace je její budoucí rozšiřitelnost o další zdroje dat a charakteristiky. Po implementaci následuje testování a ověřování vhodnosti výstupu procesu identifikace projektů podpořeného aplikací pro výzkumné účely.

2 Vývoj softwaru

Vývoj softwaru je proces vytváření, navrhování a tvorby softwarových aplikací pro různá použití. Zahrnuje širokou škálu činností, mezi které patří shromažďování požadavků, návrh, kódování, testování, ladění a údržbu. Proces vývoje softwaru je proces s několika etapami, z nichž každá vyžaduje specializované dovednosti a znalosti.

Tato kapitola poskytne přehled o procesu vývoje softwaru a také nástrojů, metodik a technik používaných k vytváření softwarových aplikací. Dojde k přiblížení pojmů projekt, proces a druhů softwarových licencí. Následně budou rozebrány metodiky vývoje softwaru. Tyto pojmy jsou důležité pro ucelený přehled pro porozumění této práci.

2.1 Proces

Proces slouží jako prostředek k dosažení cíle. Člověk stanoví proces sestávající z několika po sobě jdoucích kroků nebo činností. Procesy jsou kromě osobních sfér, jako jsou domácnosti, všudypřítomné v různých oblastech, jako je obchod, správa věcí veřejných, pedagogika a vývoj softwaru. Prostřednictvím procesů v rámci obchodních podniků dosahují korporační subjekty vytváření produktů/služeb a současně řídí zdroje s ohledem na operace, které vedou k požadovaným výsledkům.

Existují dvě hlavní klasifikace procedur: procedury, které se provádějí ručně, a procedury, které se spoléhají na automatizaci. V manuálních procesech jednotlivci využívají své fyzické schopnosti a zdroje k provádění úkolů, zatímco automatizované procesy využívají technologie, jako je software nebo počítače, k dosažení stejných cílů bez lidského zásahu. V posledních letech se stále více preferuje automatizace operací, a to hlavně v důsledku potenciální schopnosti zlepšovat efektivitu, minimalizovat výdaje a snižovat pravděpodobnost chyb způsobených jednáním lidí.

Vyvrcholení projektu je obvykle dosaženo dokončením několika etap, přičemž jedna etapa musí být dokončena, než může začít další. Každý krok na této cestě je odlišný a může vyžadovat různé akce nebo povinnosti, aby bylo možné jej provést efektivně. Například ve výrobě každá etapa zahrnuje nákup nezbytných materiálů a jejich konečné dodání spotřebitelům po testování a výrobě. Podobně vývoj softwaru zahrnuje shromažďování požadavků s následným návrhem systémů s kódováním vedoucím k nasazení systému spolu s přesným hodnocením v průběhu těchto etap.

Pro jednotlivce odpovědného za zaručení úspěchu procesu je zásadní zaznamenávat a archivovat každou fázi. Dokumentace by se měla skládat z komplexních podrobností týkajících se každého úkolu, který je třeba dokončit, všech nezbytných zdrojů požadovaných k provedení a také předpovědí očekávaných výsledků po implementaci procesu. Tento postup dokumentování se osvědčuje také v případě, že dojde k řešení problémů.

Postupem času lze procesy zdokonalovat a optimalizovat pro lepší výsledky. Analýza procesu je klíčem k provedení nezbytných úprav ke zvýšení účinnosti. Pokud například dokončení úkolu trvá delší dobu, může vyžadovat další kroky nebo přerozdělení zdrojů pro rychlejší dokončení. Tato strategie se ukazuje jako užitečná při zvyšování celkové produktivity zlepšováním klíčových operací v průběhu času.

Software proces, často také označován jako vývojový životní cyklus softwaru, popisuje sérii logických kroků nebo fází, které jsou potřebné k vytvoření nebo úpravě softwarového produktu.[14] Tyto fáze mohou zahrnovat návrh, vývoj, testování, nasazení a údržbu softwaru. Důležitým cílem správy softwarových procesů je zlepšení efektivity a kvality softwarových produktů a také předvídatelnost a řiditelnost vývoje.

2.2 Projekt

Projekty jsou dočasné sady činností a procesů, které se používají k vytvoření jedinečného produktu, služby nebo výsledku. Obvykle se skládají ze tří klíčových prvků: příprava, realizace a závěr. Příprava zahrnuje pochopení cílů projektu, sestavení harmonogramu a finančního plánu spolu s určením povinností pro členy týmu. Provedení zahrnuje zahájení samotného úsilí a průběžné sledování jeho postupu; v této fázi mohou být nutné změny projektu. Závěr zahrnuje ověření, že všechny závazky byly správně splněny v rámci rozpočtových omezení ve vhodném časovém rámci a zároveň byly splněny všechny požadované výsledky předem stanovené projektem.

Snahy se často spojují prostřednictvím konglomerátu různých odvětví a rolí s cílem spolupracovat na dosažení konečného cíle. Tyto podniky mají také schopnost urychlit změny nebo vylepšení v rámci podniku. Tyto výkony mohou být kolosální co do rozsahu nebo nepatrné, ale vždy usilují o vytvoření výrazného aktiva, které se vyplatí pro instituci.

Existují různé způsoby řízení projektu. Metody používané v tradičním projektovém řízení zahrnují segmentaci projektů na menší úkoly, jejich přidělování členům týmu a zajištění pečlivosti prostřednictvím následných postupů. S růstem technologií došlo k odpovídajícímu vývoji v technikách pro-

jektového řízení, protože mnoho organizací nyní používá softwarové nástroje, které pomáhají správné organizaci a sledování pokroku. Dosažení úspěchu vyžaduje pečlivé plánování spojené s precizním realizačním úsilím od zkušených manažerů, kteří jsou schopni zdatně manipulovat se zdroji. Dohled nad projekty může být časově náročný a nákladný, a proto je nezbytné zajistit odpovídající administrativu v každé fázi, která je s nimi spojena. Je také nezbytné pochopení celkových cílů a zároveň zachování schopnosti přizpůsobení, pokud to projekty vyžadují.

I když nelze popřít hodnotu, kterou mohou projekty přinést, jejich řízení s sebou přináší nové výzvy. Správné plánování a řízení uvedeného projektu je zásadní pro zajištění úspěchu pro všechny zúčastněné strany – což by spolu s poskytnutím vhodných zdrojů a dovedností daného týmu mělo být prioritou. Je třeba věnovat pozornost pravidelnému sledování pokroku, aby byly veškeré nezbytné úpravy prováděny včas. Selhání tohoto kroku by mohlo mít velmi negativní následky na to, jak efektivně budou cíle plněny.

Každý projekt začíná cílem, který lze přesně definovat a kvantifikovat. Tento cíl ukotvuje celý projekt a musí být pečlivě vymezen pro všechny zúčastněné strany. Pro každého účastníka dané operace je zásadní, aby dovedl rozpoznat a pochopit, k čemu v rámci svého úsilí směřuje.

Kromě cílů je dalším důležitým aspektem projektu doba trvání. Toto časové období musí být stanoveno v rané fázi a mělo by zahrnovat zásadní body vedoucí k dokončení a také počáteční a koncová data, která musí být určena tak, aby byla proveditelná. Dále je důležité, aby tato časová osa ukazovala pokrok ve vztahu k definovaným cílům při zachování přesnosti po celou dobu její implementace.

Lidé s vlastním zájmem o projekt, známí jako „stakeholders“, mají na projektu nějaký zájem (stake). Může se jednat klienty, členy týmu uvedeného projektu a další subjekty, které budou sklízet odměny z jeho úspěchů. Efektivní a kvalitní komunikace je základem pro to, aby tito jednotlivci zůstali informováni o tom, co se děje, a aby byly proaktivně řešeny jejich potřeby.

Podniky i projekty procházejí riziky. Stanovení těchto rizik je zásadním aspektem, pokud jde o řízení úkolů, a musí být vždy zohledněno ve fázi plánování a realizace. Aby mohl být projekt či úkol úspěšný, je třeba nejprve rozpoznat, vyhodnotit a snížit rizika pomocí účinných strategií zmírňování.

Mezi nezbytné součásti každého projektu se řadí i rozpočet. Jedná se o finanční plán, který nastiňuje odhadované náklady a zdroje potřebné pro úspěšné dokončení projektu. Rozpočet na projekt zároveň slouží jako vodítko, které zajistí, že projekt bude dodán ve stanoveném časovém rámci a v očekávané kvalitě. Hraje zásadní roli v každém projektu bez ohledu na jeho velikost nebo složitost. Rozpočet je zásadní pro zajištění toho, aby

projekt zůstal ekonomicky životaschopný. Mezi náklady je vhodné zahrnout cokoli, co je k úspěšnému dokončení projektu potřeba, od materiálu potřebného k práci až po mzdy všech pracovníků, kteří se podílejí na realizaci projektu. Správně navržený rozpočet by měl zajistit, že projektu nedojdou finanční prostředky, což by mělo za následek zbytečná zpoždění nebo nedodání očekávaných výsledků. Dále může sloužit jako nástroj pro měření výkonnosti projektu, kdy porovnáním skutečných nákladů s plánovanými může projektový tým identifikovat případné odchylky a provést nezbytné úpravy plánu.

Softwarový projekt je organizovaná aktivita zaměřená na vytvoření nového softwarového produktu nebo vylepšení existujícího. Tyto projekty mohou zahrnovat vše od malých změn v existujícím programu až po vývoj velkých softwarových systémů od základu.[13]

Softwarový projekt obvykle zahrnuje několik fází, které se mohou překrývat nebo následovat po sobě. Tyto fáze mohou zahrnovat:

1. Definici a analýzu požadavků – určení, co software musí dělat.
2. Návrh – vytvoření podrobného plánu pro software.
3. Implementaci – skutečné psaní kódu.
4. Testování – ověření, že software funguje tak, jak by měl.
5. Nasazení – uvedení softwaru do provozu.
6. Údržbu – opravy chyb, které se objeví po nasazení, a aktualizace pro zlepšení funkčnosti nebo výkonu.

Každý softwarový projekt také vyžaduje řízení projektu, které zahrnuje plánování, organizaci, řízení zdrojů, sledování pokroku a řešení problémů, které vzniknou během vývoje. Softwarové projekty podléhají různým softwarovým licencím.

2.3 Softwarové licence

Softwarové licence jsou právní smlouvy mezi uživatelem a poskytovatelem softwaru, které definují, jak lze software používat, distribuovat, upravovat a sdílet. Chrání tak práva duševního vlastnictví jejich tvůrců a objasňují, jaké akce jsou povoleny při používání uvedené technologie. Zanedbání těchto právních dokumentů může mít důsledky pro všechny zúčastněné strany, proto je nezbytné porozumět různým typům softwarových licencí, aby bylo

možné přesně dodržovat jejich podmínky a zároveň se zcela vyhnout právním důsledkům.

Základní klasifikace je rozdělení licencí na proprietární a open-source. Rozdíl mezi nimi je takový, že proprietární software má přísné podmínky licenční smlouvy, které omezují použití softwaru na společnost nebo jednotlivce, kteří licenci vlastní, a obvykle neumožňují nikomu jinému upravovat nebo redistribuovat software. Naproti tomu open-source mívá méně omezení v licenčních smlouvách a svým uživatelům poskytuje větší svobodu použití, modifikace nebo dokonce redistribuce bez jakýchkoli sankcí. Proprietární typ je organizacemi často využíván z bezpečnostních důvodů, zatímco open-source slouží spíše jako platforma, kde si komunita může mezi sebou volně sdílet úryvky kódu za určitých podmínek uvedených v jejich příslušné licenční dokumentaci.

Dohody týkající se softwarových licencí mají velkou váhu při definování toho, jak lze software používat, distribuovat, upravovat a sdílet. Pro odpovědného uživatele softwaru i jeho poskytovatele je zásadní porozumět právním důsledkům licenčních smluv. Tyto smlouvy obvykle obsahují klauzule, které zakazují neschválené používání, pozměňování nebo šíření programu. Při dodržování uvedených podmínek v rámci těchto licenčních ujednání mají uživatelé možnost chránit svá oprávnění a zároveň minimalizovat veškerá možná rizika z právního hlediska. Jakékoli porušení těchto podmínek může vést k přísným sankcím, jako například peněžní pokuty a soudní příkazy. Kromě toho by některé licenční dokumenty mohly obsahovat prohlášení omezující odpovědnost poskytovatele vůči jakýmkoli sporům, které vzniknou v důsledku problémů s používáním nebo problémů souvisejících s chybným provozem systému. Pochopením licenčních podmínek a jejich dodržováním lze předcházet možným právním rizikům.

2.4 OSS a druhy licencí

Iniciativy pro spolupráci jsou projekty s otevřeným zdrojovým kódem, které udržují dobrovolníci, kteří na nich společně pracují. Obvykle jsou dostupné pod licencí open-source, která umožňuje komukoli volně upravovat, využívat a distribuovat základnu kódu. Tato vzájemná spolupráce uživatelů nabízí vývojářům možnost spolupracovat na rozmanité řadě schémat – od menších osobních až po kolosální podniky s celoodvětvovým vlivem.

Projekty s otevřeným zdrojovým kódem se často používají zejména ve vývoji webových aplikací. Renomované iniciativy jako *Joomla, Drupal* a *WordPress* se mohou pochlubit miliony uživatelů po celém světě. Takové spo-

lečnosti poskytují vývojářům robustní základ pro vytváření účinných webových aplikací. Tyto společnosti navíc nacházejí uplatnění v různých oblastech včetně zabezpečení sítí, datové vědy a strojového učení.

Primárním atributem open-source projektů je jejich veřejná přístupnost, což znamená, že každý může zkoumat a revidovat kód. Tato funkce usnadňuje rychlé úpravy vývojáři a bezproblémové sdílení s ostatními uživateli. Kromě toho přístup založený na spolupráci často řídí proces vývoje těchto projektů, což vede k urychlení pokroku a lepšímu pochopení fungování softwaru.

Je zcela běžné, že open-source iniciativy jsou distribuovány pod veřejnou licencí. To znamená, že kód je k dispozici zdarma a může být upravován nebo redistribuován kýmkoli, kdo ho chce; to nejen dává sílu vývojářům softwaru, ale také inspiruje jedinečné aplikace s univerzální přitažlivostí.

Iniciativy svobodného softwaru jsou navíc konstruovány s jasným konstrukčním procesem. Veřejnosti je vystaveno, jak mohou všichni jednotlivci zapojení do tohoto podniku sledovat a poskytovat rady nebo aktualizace programového kódu, čímž vytvářejí prostředí, které podporuje zrychlený růst a současně prohlubuje porozumění uvedené iniciativě.

Iniciativy s otevřeným zdrojovým kódem mají řadu jedinečných vlastností, které jsou nedílnou součástí specifikace a nastínění jejich podstaty. S rozsáhlým pochopením těchto nuancí a ve spojení s dopadem, který mohou mít na prosperitu zmíněných snah, mohou zúčastnění zaručit dlouhověkost obou subjektů na nadcházející roky.

Open-source projekty se zpravidla liší velikostí kódu. Čím rozsáhlejší je kódová základna projektu s otevřeným zdrojovým kódem, tím větší může být jeho prosperita. To vyplývá z přirozené vlastnosti větších kódů, které poskytují další funkce, které mohou snadno zaujmout potenciálního uživatele. Nevýhodou je, že správa a ladění rozsáhlých databází může vývojáře zahltit; proto je důležité najít rovnováhu mezi rozsáhlými kódy a udržovatelností.

Je běžné, že projekty OSS mají prosperující komunitu jednotlivců, kteří software používají. Tyto skupiny nabízí jak pomoc, tak motivaci těm, kteří pracují na rozvoji, což vede k rychlé akci, když se objeví problémy nebo je třeba provést změny. Kromě toho mohou tyto skupiny pomoci šířit povědomí o projektu samotném, takže noví uživatelé jej snáze najdou. Klíčovým rozlišovacím faktorem mezi projekty s otevřeným zdrojovým kódem je jejich základna přispěvatelů. Ty úspěšné mají tendenci mít různé zázemí a schopnosti na všech úrovních dovedností. Tato rozmanitost jim umožňuje nejen zůstat relevantní, ale také začlenit inovativní pohledy z různých zdrojů do zlepšování své práce.

Je zcela zřejmé, že open-source projekty nabízejí značné výhody. Podporují prostředí spolupráce a kreativity mezi vývojáři a zároveň poskytují dostatek příležitostí pro učení a spolupráci mezi kolegy. Další výhodou pro podniky je množství zdrojů dostupných v rámci těchto typů podniků, které lze použít k vytvoření efektivnějších řešení v různých odvětvích. Trend směřující k využívání softwaru s otevřeným zdrojovým kódem v dnešním počítačovém prostředí stále nabírá na síle; zdá se pravděpodobné, že to bude pokračovat jako hlavní pilíř ve světě vývoje softwaru, který se bude posouvat kupředu i v dalších letech. V následujících oddílech se podrobněji zaměříme na konkrétní příklady licencí, které se často využívají v oblasti vývoje open-source software.

2.5 Metodiky vývoje softwaru

Metodiky vývoje softwaru jsou (polo-)standardizované frameworky pro podporu vývoje softwaru ověřené mnohaletou praxí, které vývojáři softwaru používají k organizaci a řízení úkolů a činností vývoje softwaru. Tyto metodiky poskytují strukturovaný rámec pro proces vývoje a poskytují vývojářům návod, jak nejlépe přistupovat ke svým projektům.

Vývoj softwaru je složitý a nákladný proces. V dnešní době je k dispozici velké množství různých specializovaných nástrojů pro podporu procesu vývoje softwaru nebo jeho různých druhů. V posledních letech bylo možné zaznamenat významné změny ve schopnostech a vlastnostech těchto nástrojů - probíhal nepřetržitý vývoj směrem k integrovanějším sadám nástrojů a platformám. Nástroje se vzájemně lépe a těsněji integrují a umožňují tak vzájemně propojovat stále více dříve oddělených částí procesních řetězců.[18]

Mezi zástupce známých metodik vývoje softwaru patří vodopádový model, agilní metodika, extrémní programování, Scrum, Kanban a DevOps.

2.5.1 Vodopádový model

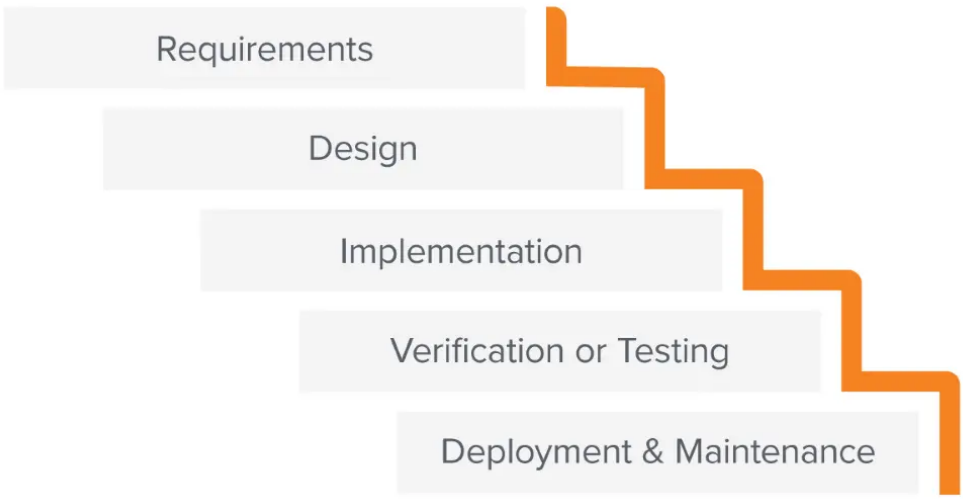
Vodopádový (Waterfall) model je tradiční přístup k řízení projektů, který se vyznačuje sekvenčním, lineárním tokem úkolů. Jedná se o vysoce strukturovaný přístup, který přechází z jedné fáze do další v logickém pořadí a vyžaduje, aby každá fáze byla dokončena dříve, než může začít další. Tento typ přístupu se běžně používá u projektů, které jsou dobře definovány s jasnými cíli a záměry.

Metodika vodopádu zahrnuje řadu fází:

- Požadavky a specifikace: V této fázi projektový tým a zúčastněné

strany identifikují požadavky projektu a vypracují soubor specifikací.

- Návrh: Jakmile jsou identifikovány požadavky a specifikace, projektový tým navrhne řešení. To zahrnuje návrh architektury, uživatelského rozhraní, databází a dalších komponent, které budou tvořit systém.
- Vývoj: V této fázi je aplikován návrh a implementován software.
- Testování a ladění: Jakmile je software vyvinut, projektový tým jej otestuje, aby se ujistil, že splňuje požadavky a specifikace. Všechny zjištěné problémy jsou opraveny.
- Nasazení: Software je nasazen do produkčního prostředí.
- Údržba: Po nasazení softwaru je monitorován a udržován, aby byl zajištěn optimální výkon.



Obrázek 2.1: Proces vodopádové metodiky [27]

Vodopádový model je oblíbeným přístupem pro řízení projektů, protože je jednoduchá a snadno pochopitelná. Navíc je užitečná pro projekty, které jsou dobře definované a mají jasný cíl. Může však být obtížné se v tomto modelu přizpůsobit měnícím se požadavkům a také není příliš vhodný pro projekty, které vyžadují mnoho iterací a experimentů.

2.5.2 Agilní metodika

Metodika agilního vývoje softwaru se v poslední době stala jednou z nejčastěji používaných technik vývoje softwaru. Spíše než dlouhé cykly uvolňování

v dříve populární metodice vodopádu, agilní technika navrhuje pravidelně krátké cykly uvolňování „sprintů“. To umožňuje zákazníkům a zainteresovaným stranám více se zapojit do procesu vývoje softwaru.[7]

Ústředním bodem je týmová práce, při které pracovní skupiny a klienti kombinují své úsilí k rychlému vytvoření funkčního produktu. Cílem je nejen efektivita, ale také spolupráce mezi různými frakcemi pro konečný cíl úspěchu. Zúčastněné strany jsou povzbuzovány spíše spoluprací než individuálním úsilím tak, aby bylo dosaženo optimálního výsledku, z něhož budou mít prospěch všichni. Tento přístup v konečném důsledku přináší efektivnější výsledky, které budou dobře přijaty i zúčastněnými stranami.

Výhoda agilní metodiky spočívá v její schopnosti přizpůsobit se a posouvat během vývojové fáze, což z ní dělá hlavního kandidáta pro projekty s měnícími se požadavky nebo nejasnými konečnými cíli. Iterativní přístup umožňuje nepřetržitý proud zpětné vazby, který vede ke kvalitnějším softwarovým výstupům. Práce v tomto rámci zároveň podporuje týmovou spolupráci mezi všemi zúčastněnými členy a nakonec zdokonaluje komunikační dovednosti a zároveň zlepšuje schopnosti řešit problémy.

Naopak implementace agilní metodiky se může ukázat jako skličující úkol kvůli její větší složitosti ve srovnání s vodopádovým modelem. Potřeba neustálé spolupráce a komunikace mezi členy týmu může být i nevýhodou, jelikož ve větších týmech se stává namáhavým úsilím. Kromě toho může přijetí iterativního procesu vést k rozšíření rozsahu, kdy projekty překonávají svá počáteční omezení.

2.5.3 Extrémní programování

Extrémní programování (Extreme Programming; XP) je agilní metodika vývoje softwaru, která klade důraz na týmovou práci, komunikaci a zpětnou vazbu. XP je známý svým přísným dodržováním osvědčených postupů, včetně testování řízeného vývoje (Test-driven development; TDD) a párového programování.[4]

Základní ideologie XP se soustředí na rychlé a efektivní dodávání špičkového softwaru. Aby toho bylo dosaženo, XP implementuje řadu metodik, které upřednostňují týmovou práci a efektivní výměnu mezi členy týmu. Mezi tyto klíčové postupy patří párové programování, které zahrnuje spolupráci dvou vývojářů na jednom projektu, přičemž sdílejí jak ovládání klávesnice, tak přístup k monitoru. Tento proces podporuje získávání znalostí a současně zmírňuje chyby, aby se zlepšily celkové standardy kvality.

2.5.4 Scrum

Procedura Scrum představuje dobře definované rozložení a systém pro progresi růstu. Zahrnuje časté setkávání a rituály, posilující korespondenci i týmovou práci. Cyklická metoda vytváří prostor pro neustálou zpětnou vazbu spolu s pokrokem, což má za následek vynikající dodávku softwaru. Design také vychází vstříc adaptabilním doménám, což jej činí vhodným pro podniky s neurčitými vyhlídkami.

Tento přístup se opírá o kompaktní skupiny, které používají krátké, cílené sprinty k dosažení opakujících se cílů. Tyto sprinty obvykle mají předem dohodnutou dobu trvání, často rozsáhlejší než jeden nebo dva týdny. Proces Scrumu má v zásadě v úmyslu rozdělit rozšířené podniky na praktické segmenty; nastiňuje odpovědnosti členů týmu a zjišťuje cíle pro každou jednotlivou fázi a zároveň určuje, jak je každý sprint hodnocen. Každý sprint má předem stanovený závěr.

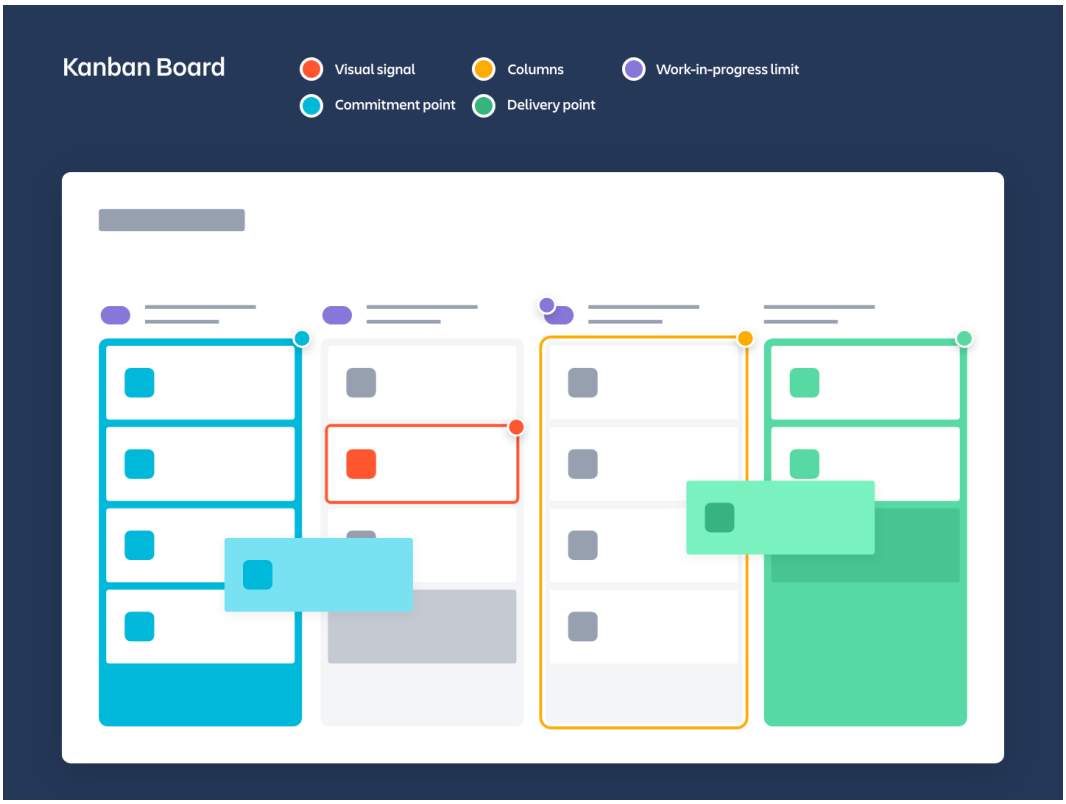
Zavedení metodiky Scrum větším týmům může představovat potíže. Spolupráce a komunikace jsou nanejvýš důležité, ale když velikost týmu roste, je těžší je udržet. Kromě toho je nezbytná odbornost v rámci týmu, proto nemusí být vhodné pro nováčky nebo amatéry takový proces implementovat.

Scrum je nejpopulárnější, volně použitelný agilní přístup. Školit Scrum může kdokoliv (při dodržení Attribution Share-Alike license od Creative Commons), akreditace trenérů a tréninkových organizací je dobrovolná a poskytuje ji několik subjektů. [23]

2.5.5 Kanban

Kanban jako strategie preferuje vizualizaci postupu práce a stanovení limitů pro počet rozpracovaných úkolů v daném okamžiku. Také se zaměřuje na dodání dokončených úkolů.[5] Kanban vznikl původně v oblasti výroby a později byl adaptován pro potřeby vývoje softwaru. Tato metodika zahrnuje principy a postupy, které slouží k úspěšnému řízení implementace během celého vývojového procesu.

Jádrem metodiky Kanban je vizualizace práce. Aby týmy zhmotnily a vyjasnily vývojové postupy, využívají jedinečný nástroj nazvaný „Kanban boards“. Tyto tabule vizuálně ztělesňují každý úkol. Obvykle jsou rozděleny do různých sloupců zarovnaných s různými fázemi vývoje produktu nebo služby. Mohou být digitální nebo fyzické s použitím karet/lepících poznámek jako reprezentace úkolů. Účelem je nabídnout explicitní pohled na všechny aktivity a zároveň pomoci najít oblasti, které vyžadují upřesnění identifikací omezení (úzká místa). Na obrázku 2.2 lze vidět příklad takového digitálního Kanban boardu.



Obrázek 2.2: Příklad Kanban boardu

2.5.6 DevOps

DevOps představuje soubor postupů, které mají za cíl zkrátit čas mezi implementací systémových změn a jejich nasazením do běžného provozu, přičemž stále zajišťují vysokou kvalitu.[19] DevOps je kombinací vývoje (development) a provozu (operations), odtud název.

DevOps je přístup k vývoji softwaru, který klade důraz na spolupráci a komunikaci mezi vývojáři softwaru a odborníky na provoz informačních technologií. Cílem DevOps je zlepšit efektivitu práce a efektivitu procesu vývoje softwaru odstraněním bariér mezi těmito tradičně oddělenými týmy. Podle definice navržené Bass, Weber a Zhu lze DevOps shrnout jako soubor postupů, které mají za cíl zkrátit časový interval mezi implementací změn v systému a nasazením těchto změn do produkce. Tento proces je prováděn s důrazem na zajištění vysoké kvality produktu.[3]

Jedním z klíčových principů DevOps je tzv. „continuous delivery“, které zahrnuje časté a automatické vydávání aktualizací softwaru. Automatizací procesu vydání mohou týmy DevOps zkrátit čas a úsilí potřebné k nasazení nového softwaru a zároveň zlepšit spolehlivost a kvalitu vydávaného softwaru.

DevOps také zdůrazňuje použití agilních vývojových metodik, které zahrnují časté iterace a nepřetržitou zpětnou vazbu. Tento přístup umožňuje týmům rychle reagovat na měnící se požadavky a potřeby zákazníků, což vede k efektivnějšímu vývoji softwaru.

Mezi hlavní benefity patří zvýšení důvěry mezi různými články společnosti, rychlejší dodávky software, schopnost lepšího řešení kritických problémů a snazší řízení nezbytných pracovních činností.[2]

3 Projektové řízení

Projektové řízení je důležitou součástí každé úspěšné organizace. Je to proces plánování, organizace a řízení zdrojů za účelem úspěšného dokončení projektu. Kromě toho zahrnuje také stanovení cílů a také sledování a kontrolu průběhu projektových aktivit.[21]

Projektové řízení zahrnuje v první řadě odhalení cílů projektu a vytvoření uspořádání zaměřeného na dosažení těchto cílů. Schéma by mělo obsahovat harmonogram, finanční odhad, personální předpoklady i postupy pro řízení projektů. Implementace těchto strategií a sledování pokroku jsou navíc nezbytné pro úspěšné řízení projektu. Garantovat včasné dokončení v rámci schválených financí nakonec leží na bedrech pověřeného manažera.

Být projektovým manažerem vyžaduje kvalitní komunikační a organizační schopnosti spojené se schopností zvládat složité projekty. Je také nezbytné, aby měli nadání probudit vášeň ve svých týmech, aby společně dosáhli úspěchu. Kromě toho by každý kompetentní projektový manažer měl dokázat identifikovat potenciální rizika a vytvářet strategie k jejich minimalizaci.[13]

Pro každou organizaci je klíčové mít efektivní a spolehlivé projektové řízení. Projektové řízení vede k systematizovanému přístupu, který zaručuje dokončení úkolu v rámci rozpočtových omezení, podle plánu a s optimálními výsledky. Navíc podporuje lepší komunikaci mezi členy týmu, což vede k vynikajícím výsledkům a vyšší úrovni kvality práce. Velký význam pro zajištění uspokojivé úrovně produktivity má efektivní administrace projektů od začátku do konce; to začíná vytvořením uspořádaných struktur, které jsou explicitně přizpůsobeny k dosažení předem definovaných cílů, přičemž je třeba věnovat velkou pozornost nejen finančním rozhodnutím, ale také časovým rámcům, které jsou ústředními aspekty kulminujícími v míře úspěšnosti – což je zásadní faktor. Usnadnění nabízené prostřednictvím pokynů při interakci mezi kolegy navíc zvyšuje přesnost.

Identifikace a vyhnutí se potenciálním problémům prostřednictvím efektivního projektového řízení vede ke snížení nákladů pro organizace. Projekty, které překračují rozpočet nebo u nich dochází ke zpožděním, mohou odčerpat značné množství peněžních zdrojů, a proto jsou tato proaktivní opatření nezbytná. Zlepšení procesů a postupů v identifikovaných oblastech navíc zvyšuje provozní efektivitu, což vede i k úsporám nákladů pro organizaci.

Koordinace a řízení projektu od koncepce až po dokončení je nezbytnou součástí každého prosperujícího podniku. Tento proces zahrnuje rozpoznání

cílů, vymýšlení strategií pro realizaci uvedených cílů a následně dohled nad pokrokem při současném sledování výsledků. Slouží jako ochrana proti neočekávaným výdajům tím, že předvídá možné překážky dříve, než nastanou. Kompetentní poradenství při řízení projektů je nejen klíčové, ale také prvořadé; přináší jak zvýšení produktivity, tak snížení výdajů v rámci organizací jako celku.

V této kapitole budou rozebrány praktiky projektového řízení a nástroje pro správu vývoje aplikací.

3.1 Praktiky projektového řízení

Praktiky projektového řízení představují techniky, metody nebo procesy, které jsou považovány za efektivní při použití v konkrétním stavu nebo za určitých okolností. [16] Tyto protokoly jsou zavedeny za jediným účelem: zajistit, aby se každý cíl projektu naplnil tím, že se zaručí využití špičkových zdrojů při zachování rozpočtových omezení a konkrétních časových plánů. Tyto postupy existují jako podpůrné vodítko ve všech fázích postupu týmu směrem k požadovaným cílům.

Uspořádání pokynů, jimiž se řídí řízení a provádění projektu, lze charakterizovat třemi základními klasifikacemi: plánování, realizace a kontrola. Plánování zahrnuje nastínění požadovaných úspěchů spolu s jejich doprovodnými cíli, stanovení zásad pro složení týmu a také vymezení přiřazení rolí v rámci týmu. Realizace je součástí efektivní správy zdrojů, takže pracovní závazky jsou dokončeny včas a zároveň jsou dodržovány vyšší stupně kvality. Konečně, kontrola se týká prověřování vývoje v průběhu času a navíc odhalování oblastí ohrožených rizikem nebo s již vznikajícími komplikacemi - nejlépe ještě před tím, než nabudou účinnosti a jakmile budou uznána vhodná opatření, mohou být přijata odpovídajícím způsobem.

Úspěšné postupy řízení projektů jsou postaveny na základních principech. Patří mezi ně komunikace, spolupráce a neochvějný závazek k neustálému zlepšování. Bez efektivní komunikace mezi členy týmu je téměř nemožné, aby projekty uspěly podle plánu. Zůstat informován a aktuální je v tomto ohledu zásadní. Spolupráce slouží jako další klíčový aspekt optimální realizace projektu – všichni členové spolupracují na dosažení stanovených cílů a zároveň směřují své úsilí produktivně se zaměřením na snížení redundance, aby se zajistilo, že nedojde k žádné odchylce od původního účelu. Tím je plnění termínů snazší, pokrok probíhá hladce ve srovnání s přístupem, kdy si každý dělá, co chce, bez referenčních bodů nebo jasných cílů. Aniž by to bylo jakkoli omezeno, týmy mohou využívat neustálé zlepšování,

které klade důraz na identifikaci oblastí, které vyžadují více pozornosti.

Existuje mnoho přístupů k implementaci postupů projektového řízení. Mnoho společností volí využití specializovaných softwarových systémů, které jim umožňují efektivně řídit své projekty. Tyto programy nabízejí řadu nástrojů, které usnadňují plánování, monitorování a podávání zpráv o pokroku dosaženém v každé fázi životního cyklu daného projektu, stejně jako přístup k nezbytným zdrojům a dokumentům určeným pro členy týmu zapojené do uvedených snah.

Kromě implementace technik projektového řízení mají podniky možnost využívat alternativní strategie, jako jsou workshopy a školení, aby dosáhly odbornosti s ohledem na řízení projektů. Workshopy jsou přínosné, protože mohou jednotlivce vybavit nezbytnými kompetencemi potřebnými k úspěšnému dohledu nad jejich příslušnými snahami. Podobně vzdělávací programy nabízejí komplexní vzdělávací instrukce, které pomáhají členům týmu pochopit různé aspekty procesu a pomáhají jim identifikovat klíčové odpovědnosti s ním spojené.

Implementace protokolů projektového řízení je nezbytná pro dosažení úspěchu v uskutečnění závazku. Dodržováním těchto opatření mohou společnosti zaručit, že jejich úsilí bude provedeno rychle, v rámci finančních omezení a s maximální dokonalostí. Efektivní správa jednotlivými firmami navíc umožňuje rozpoznat, kde by bylo možné provést vylepšení, a poskytnout odpovídající úpravy.

3.2 Application Lifecycle Management

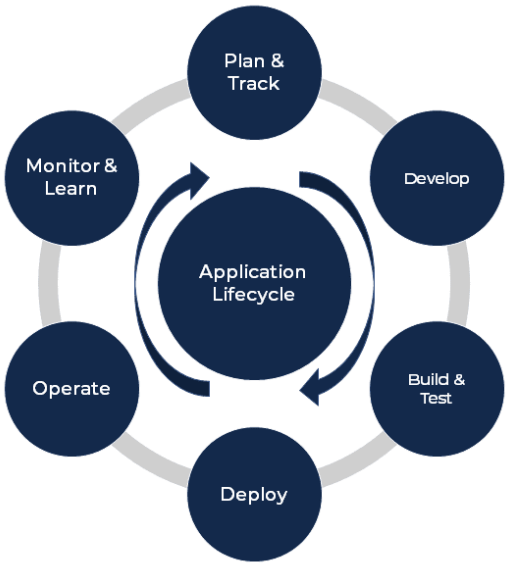
K dohledu nad celým životním cyklem vývoje softwaru se používá disciplína zvaná Application Lifecycle Management (ALM). Zahrnuje správu řízení změn, konfigurací, verzí, sestavení, nasazení, znalostní báze, komunikace, kooperace a další operace. Konečným cílem je efektivní proces, který přísně dodržuje standardy vysoké kvality při zachování správné organizace v každé fázi existence produktu. Existují ALM nástroje, které tuto disciplínu pomáhají realizovat.

Použití nástrojů ALM při vývoji softwaru přináší řadu výhod, které celý proces zjednodušují. Tyto zdroje umožňují týmům efektivně dohlížet a regulovat postup projektu, zaručují dodržení termínů a zároveň zajišťují správný průběh. Kromě toho pomáhají organizovat a zefektivňovat celkový přístup pro maximální účinnost; poskytuje zavedený systém pro dohled nad více úkoly a cíli, který minimalizuje dobu dokončení.

Nástroje ALM nabízejí řadu výhod, včetně zajištění vysoce kvalitních

procesů vývoje softwaru. Poskytují komplexní řadu nástrojů pro testování, ladění a nasazení pro okamžité zmírnění jakýchkoli potenciálních chyb nebo závad. Kromě toho umožňují sledování výkonu produktu, který zajišťuje spokojenost zákazníků s ním. ALM také výrazně snižuje náklady na vytváření softwaru automatizací postupů a poskytováním rámců pro sledování pokroku ve všech fázích vývoje. Kromě samotného snižování nákladů na vývoj mohou tato řešení zlepšit funkčnost aplikace a zároveň minimalizovat výskyt problémů s údržbou a podporou během jejího životního cyklu, což je kritická funkce, kterou je třeba vzít v úvahu při hledání optimální efektivity vybraného přístupu k řízení projektů.

Výhody softwarových nástrojů ALM jsou četné a přispívají ke zvýšení efektivity při vývoji softwaru a zároveň zůstávají ekonomicky proveditelné. Mají schopnost zaručit systémovou a efektivní metodu pro vývoj vysoce kvalitních softwarových řešení. Navíc snižují náklady související s návrhem a údržbou programu. Tyto nástroje jsou pro týmy, které se podílejí na programování projektů, velmi cenné, protože pomáhají efektivně podporovat pozitivní výsledky projektů a tím přispívají k úspěchu.



Obrázek 3.1: Životní cyklus aplikace [17]

3.3 Nástroje pro správu verzí

Jedním z kritických nástrojů pro řízení určitého typu projektu vývoje softwaru jsou systémy správy verzí (VCS). VCS se týká aplikace, která průběžně monitoruje úpravy provedené v úložišti zdrojového kódu, což umožňuje vývojářům spolupracovat a pohodlně získávat předchozí verze kódování.

Základním účelem VCS je pomáhat vývojářům mít přehled o změnách provedených ve zdrojích kódu v průběhu času, což usnadňuje týmovou spolupráci a sledování pokroku. Dokumentací každé změny, která byla aplikována na zdrojový kód, poskytuje VCS vývojářům šanci vrátit se k předchozím verzím podle svého uvážení. Prostřednictvím tohoto nástroje lze změny snadno kontrolovat a v případě potřeby lze případně obnovit předchozí konfigurace.

Kromě monitorování změn v kódu lze systémy správy verzí použít pro řízení uživatelských rolí a oprávnění. To dává programátorům možnost navrhovat odlišné větve vývoje a zároveň udělovat exkluzivní přístup ke každé větvi pouze určeným vývojářům. Takový systém zaručuje, že všichni spolupracují na identických verzích, čímž se snižuje jakákoli vyhlídka na konfliktní kódy vznikající při vývoji.

VCS umožňuje vývojářům pohodlně distribuovat svůj kód s kolegy. Prostřednictvím společné databáze jsou spolupracovníci schopni bezpečně spolupracovat na iniciativách a vytváří mezi sebou pomyslné tržiště s nápady a možnými řešeními. To umožňuje snadnou výměnu myšlenek, které posouvají projekty k úspěchu.

Pro jakýkoli projekt vývoje softwaru je implementace systému správy verzí cenným přínosem. Vývojáři mohou s VCS s jistotou dosahovat optimálních výsledků svých projektů při zachování aktuálních bezpečnostních opatření.

3.4 Nástroje pro správu nasazení

Sloučení vývoje softwaru a informačních technologických operací, jinak známé jako DevOps, nabízí unikátní soubor postupů. Tyto metodiky jsou určeny k urychlení životního cyklu systému (viz obrázek 3.1) a zároveň k pravidelné implementaci aktualizací, které jsou v souladu s obchodními cíli. Klíčová složka k dosažení úspěchu spočívá v efektivní komunikaci, spolupráci mezi zapojenými týmy, integračních technikách využívaných pro účely automatizace ve spojení s měřitelnými výsledky ze spolupráce mezi vývojáři a dalšími profesionály v informačních technologických rolích. Tímto přístupem se vytváří prostředí pro spolupráci obou sektorů, což vede k vzájemné odpovědnosti a pozitivním výsledkům v rámci agilní metodiky, která pod-

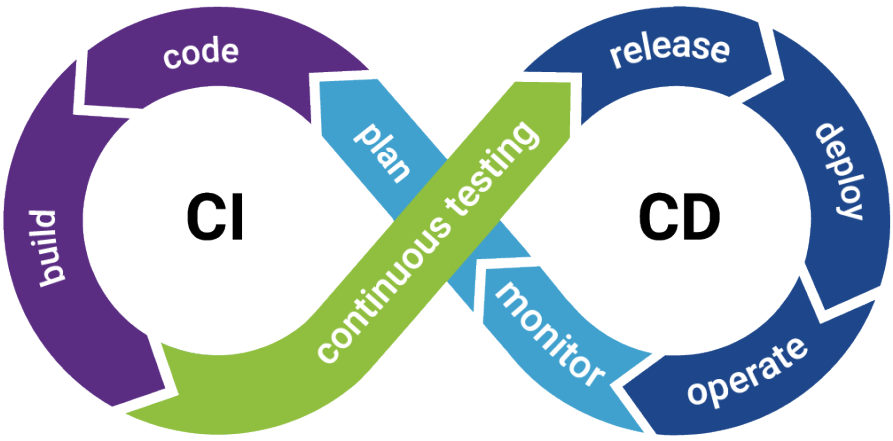
poruje efektivní procesy při změnách infrastruktury dodávek pro účinnou implementaci nebo vylepšení softwaru.

V oblasti DevOps jsou základní ideologie Continues Integration (CI) a Continues Delivery (CD), které společnosti používají k rychlé výrobě vysoce kvalitního softwaru a zároveň zajišťují jeho efektivní testování. [26] Tyto koncepty mají nesmírnou hodnotu pro jakoukoli organizaci, která se snaží urychlit svůj vývojový proces, přísně dodržovat harmonogramy projektů, udržovat globální standardy zajišťování kvality a také zvyšovat úroveň spokojenosti klientů.

Akt nepřetržité integrace (CI) zahrnuje sloučení kódu od různých vývojářů do samostatné softwarové verze. Bezpodmínečně každá instance, kde dojde k úpravě kódu, vede k integraci a důkladnému testování na případné chyby. To slouží jako záruka, že změny nebudou bránit zavedeným operacím ani přinášet nové závady. Jako takový pomáhá minimalizovat dobu opravy chyb a zabraňuje výskytu nových chyb.

Automatizace dodávání softwaru se označuje jako nepřetržité doručování (CD). CD zahrnuje nasazení produkčního kódu, testovací procedury a ověření funkčnosti systému. Použití tohoto procesu zkracuje jak dobu odezvy zákazníků na nové funkce/opravy chyb, tak i zkracuje dobu odstraňování problémů/ladění.

Aby podnik mohl efektivně vyrábět a distribuovat software, jsou klíčové CI i CD. Vývojáři mohou rychle identifikovat závady pomocí CI, zatímco automatizaci usnadňuje CD při dodávání uvedeného softwaru. Výsledkem je celkové snížení výrobních nákladů při vytváření a distribuci takových softwarových produktů. Na obrázku 3.2 je znázorněn proces CI/CD.



Obrázek 3.2: Proces CI/CD [6]

3.5 Nástroje pro řízení OSS projektů

3.5.1 Systémy správy zdrojového kódu

Systémy správy zdrojového kódu (SCM) jsou nástroje používané ke správě zdrojového kódu pro vývoj softwaru. Pomáhají vývojářům spolupracovat na kódu, sledovat změny a udržovat kontrolu verzí. Dva populární SCM pro projekty OSS jsou **Git** a **SVN**.

Git je distribuovaný systém správy verzí, který umožňuje vývojářům pracovat na kódu současně, bez konfliktů. Poskytuje kompletní historii změn provedených v kódové základně, což usnadňuje sledování a vracení změn. Git je oblíbený nástroj pro správu projektů OSS, protože podporuje spolupráci, správu verzí a větvení a slučování.

GitHub a **GitLab** jsou dvě oblíbené webové platformy pro hostování úložišť Git. Poskytují výkonnou platformu pro spolupráci a automatizaci, což vývojářům usnadňuje souběžnou práci na kódu, správu verzí i sledování změn. Obě platformy jsou široce používány pro projekty OSS, protože poskytují centrální místo pro informace související s projektem a usnadňují přispívání do projektu.

Online platforma **GitHub** se může pochlubit nepřeberným množstvím řídících zdrojů zaměřených na řízení softwarových projektů. Je nesmírně populární a je absolutním standardem při současné tvorbě softwaru.

GitHub nabízí centralizované centrum pro zpracování softwarových projektů, zahrnuje všechny důležité funkce, jako je správa verzí, sledování problémů a platformy pro spolupráci. Spoléhá na distribuovaný systém správy verzí Git, který umožňuje vývojářům pracovat společně v reálném čase a zároveň regulovat úpravy provedené v jejich kódové základně. Pomocí webového rozhraní GitHubu lze efektivně spravovat úložiště Git a tím i bezproblémovou správu a spolupráci mezi týmy pracujícími na zdrojových kódech.

Spolupráce je prominentním aspektem funkčnosti GitHubu. Simultánní práce s kódem, odesílání změn ke kontrole a správa problémů souvisejících s projektem jsou všechny služby, které platforma nabízí vývojářům. Tyto funkce usnadňují zpracování vstupů od mnoha přispěvatelů při zachování konzistence a stability v základně kódu.

GitHub nabízí působivé možnosti správy verzí, jako je větvení a slučování, což umožňuje kodérům pracovat na různých iteracích kódové základny současně a zároveň integrovat upgrady do primárního programu. GitHub navíc poskytuje prvotřídní nástroje pro správu verzí, které zjednodušují šíření softwaru a procesy údržby.

GitHub se může pochlubit významným atributem v podpoře automa-

tizace. Software umožňuje využití silných zdrojů pro nepřetržitou integraci a nasazení (CI/CD), čímž zjednodušuje konstrukci kódu, testování a odesílání, aby se minimalizovaly lidské zásahy, a zároveň umožňuje vývojářům více času soustředit se na kódování spíše než na činnosti projektového řízení.

V oblasti vývoje softwaru je **GitLab** nástrojem, který umožňuje přístup ke správě s otevřeným zdrojovým kódem pro sledování různých úložišť kódu. Může se pochlubit rozmanitými funkcemi pro správu projektů a lze jej přirovnat ke GitHubu, pokud jde o jeho schopnost umožnit produktivní partnerství mezi spolupracovníky na konkrétních projektech kódování prostřednictvím zjednodušené integrace, dostupnosti nástrojů pro kontrolu a také funkcí automatizace. Navíc umožňuje spolupráci více vývojářů v reálném čase s přidanými funkcemi pro správu verzí. Součástí jeho funkční sady jsou také nástroje pro efektivní koordinační úsilí, jako je usnadnění činností sledování problémů, správa žádostí o sloučení. Posílení průběžné integrace a průběžného nasazování (CI/CD) propojuje výskyty všech dotčených oblastí v různých fázích daných projektů, na nichž se podílí.

GitHub a GitLab mají jakožto online centra, která nabízejí rozsáhlé zdroje pro dohled nad softwarovými iniciativami, mnoho společného. Navzdory jejich paralelám však mezi nimi existuje řada rozdílů, které stojí za zmínku.

GitHub a GitLab se liší ve svém cílovém publiku. GitHub upřednostňuje hostování iniciativ open-source softwaru (OSS), zatímco GitLab se soustředí na vývoj softwarových projektů na podnikové úrovni. Tato preference se odráží ve funkcích a balíčcích jednotlivých platforem dostupných uživatelům; konkrétně GitHub poskytuje úroveň pro OSS, která je zdarma a zároveň nabízí přístup k placeným plánům pro podniky, které hledají další nástroje, na rozdíl od GitLabu, který nabízí jak bezplatnou komunitní edici, tak i exkluzivní podnikový model za různé ceny.

Způsob, jakým zvládají průběžnou integraci a nasazení (CI/CD), se výrazně liší. GitLab nabízí komplexní sadu nástrojů pro CI/CD, která zahrnuje vestavěný kanál. Naproti tomu GitHub závisí na integracích třetích stran, aby dosáhl stejné úrovně automatizace, pokud jde o CI/CD. V důsledku toho lze namítnout, že GitLab vyžaduje méně externích doplňků, protože je vybaven komplexnějším soběstačným softwarem ve srovnání s přístupem GitHubu, který se spoléhá na více externích zdrojů, což může vyžadovat další práci k dosažení podobných výsledků prostřednictvím automatizace.

GitLab nabízí širší škálu možností a příležitostí. Může být zřízen v rámci organizace nebo zřízen na internetu, což poskytuje větší kontrolu nad vývojovým prostředím. Alternativně je GitHub převážně cloudový s nedostatečnými příležitostmi pro instalaci na místě.

Podobnost mezi GitHub a GitLab, pokud jde o spolupráci a správu verzí, je poměrně významná. Oba nabízejí webově orientovanou platformu pro práci s repozitáři Git, které jsou vybaveny výkonnými zdroji určenými pro hodnocení kódu, sledování problémů a dynamiku partnerství. Tyto dva programy navíc uživatelům nabízejí robustní popisné vlastnosti, jako jsou možnosti dělení a slučování, spolu s dalšími praktickými nástroji.

Subversion (SVN) je centralizovaný systém správy verzí, který se často používá. Umožňuje monitorování a vracení změn v kódových základnách, čímž se v konečném důsledku vytvoří celý výčet těchto změn v průběhu času. Projekty OSS často využívají SVN pro práci s větvemi, možnosti slučování a schopnost přesně sledovat verze v průběhu vývojových fází. Všechny soubory související s kódováním jsou uloženy v jednom centrálním úložišti díky pevné architektuře SVN – to umožňuje vývojářům současnou práci a zároveň vyžaduje neustálé připojení k internetu pro jakékoli velké příspěvky jednotlivých členů, aniž by došlo ke ztrátě cenných datových bodů nashromážděných během spolupráce s ostatními členy týmu.

TortoiseSVN je oblíbený nástroj pro správu repozitářů SVN, protože poskytuje snadno použitelné rozhraní pro práci s kódem a repozitáři. SVN je široce používán pro projekty OSS, nicméně v posledních letech se stal méně populární kvůli vzestupu distribuovaných systémů pro správu verzí, jako již zmíněný Git. Zatímco SVN má stále své místo ve světě vývoje softwaru, mnoho vývojářů přešlo na novější a výkonnější nástroje.

Redmine je výkonný a oblíbený nástroj pro řízení OSS projektů. Jeho hlavní funkcionalita zahrnuje správu kódu, problémů, dokumentace a dalších aspektů projektu. Redmine je speciálně navržen pro správu OSS projektů a nabízí širokou škálu funkcí, které umožňují uživatelům lépe koordinovat a monitorovat procesy vývoje. Redmine umožňuje uživatelům snadno sledovat pokrok projektu díky výkonnému systému sledování a řešení problémů. Uživatelé mohou jednoduše vytvářet nové problémy, přidávat k nim komentáře a sledovat jejich stav a prioritu. Redmine také umožňuje uživatelům vytvářet a spravovat dokumentaci projektu, jako jsou například specifikace, návody a smlouvy. Jednou z klíčových funkcí Redmine je integrace s dalšími nástroji pro řízení projektů. Redmine umožňuje snadnou integraci s nástroji jako jsou například Git, SVN, Mercurial a CVS, což umožňuje správcům projektu sledovat a spravovat kódovou bázi projektu přímo z Redmine. Redmine také nabízí integraci s nástroji pro sledování času a výkonnosti, jako jsou například *TimeTracker* a *Harvest*. Další výhodou Redmine je jeho otevřený zdrojový kód a flexibilita. Uživatelé mohou snadno upravovat a přizpůsobovat Redmine svým potřebám díky široké škále pluginů a rozšíření. Redmine je také k dispozici ve více než 30 jazycích, což usnadňuje jeho použití pro

uživatele po celém světě.

3.5.2 Nástroje pro řízení projektů

Vývoj softwarových projektů je proces usnadněný různými nástroji pro správu. Tyto nástroje pomáhají při sledování úkolů, přidělování práce a koordinaci činností souvisejících s úspěchem projektu. Mezi preferovanými volbami několika špičkových projektů OSS jsou **Jira** a **Trello**, které odvádějí výjimečnou práci v oblasti správy vývoje softwaru s otevřeným zdrojovým kódem.

Všestranným nástrojem pro správu projektů je Jira, který slouží k monitorování závad, povinností a požadavků na funkce. Oblast centralizovaná se všemi informacemi relevantními pro projekt oplývá zčásti díky svým schopnostem, které umožňují vývojářům i manažerům, kteří se starají o projekt, sledovat pokrok a zároveň identifikovat problémy, které se během cesty objeví. Kromě toho poskytuje podporu směrem k agilním vývojovým metodikám, jako je Scrum nebo Kanban, a to prostřednictvím mnoha integrovaných možností.

Trello, snadno použitelný a instinktivní software pro řízení projektů, využívá nástěnky, seznamy a karty k efektivnímu sledování úkolů. Prostřednictvím vizuální prezentace vývojové fáze lze pohodlně sledovat pokrok a snadno identifikovat komplikace. Častěji se využívá ruku v ruce s dalšími technickými nástroji, jako je GitHub, který slouží komplexnímu účelu pro bezproblémové řízení projektů.

3.5.3 Nástroje průběžné integrace

Automatizace vytváření, testování a nasazování kódu je umožněna pomocí nástrojů Continuous Integration (CI). S jejich pomocí jakékoli úpravy základní struktury softwaru nepředstavují chyby ani jej žádným způsobem nepoškozují. Při zkoumání OS projektů, které k tomuto účelu hojně využívají nástroje CI, patří mezi dvě nejoblíbenější možnosti **Jenkins** a **Travis CI**.

Jenkins, open-source kontinuální integrační nástroj, je schopen pojmout různorodou škálu programovacích jazyků a technologií. Jeho funkčnost umožňuje vývojářům zavést testovací postupy, které lze provádět automaticky v reakci na jakékoli změny provedené v kódové základně. Jenkins, který je pozoruhodně všestranný, pokud jde o vývojové pracovní postupy, slouží jako robustní platforma pro účely automatizace, přičemž zůstává přizpůsobitelný podle preferencí nebo požadavků vývojářů.

Nástroj s názvem Travis CI je hostován v cloudu. Díky partnerství s GitHubem nabízí tato platforma přístupné prostředí pro vytváření a analýzu

kódu. Kodéři mají hned od začátku k dispozici několik technologií díky předinstalované infrastruktuře Travis CI, která vyhovuje požadavkům známých programovacích jazyků. Umožňuje tak okamžité procesy automatizovaného testování a nasazení.

3.5.4 Nástroje pro správu dokumentů

Organizace mohou využívat softwarová řešení známá jako nástroje pro správu dokumentů k ukládání, správě a sledování elektronických dokumentů. Tyto inovativní aplikace byly navrženy za účelem zlepšení způsobu provádění pracovních postupů a zároveň zvýšení opatření pro zajištění kvality snížením chyb spojených s ruční manipulací s papírováním.

Nástroje pro správu dokumentů lze rozdělit do dvou odlišných kategorií. První typ zahrnuje ty, které se zaměřují na ukládání a vyhledávání dokumentů, zatímco druhá kategorie zahrnuje ty, které jsou určeny pro účely spolupráce.

Společnosti se mohou rozhodnout využít on-premise řešení pro správu dokumentů, která jsou implementována na jejich interní servery a pod dohledem informačního technologického oddělení. Tyto konkrétní nástroje umožňují větší míru přizpůsobení než jiné, ale vyžadují, aby bylo na údržbu v této oblasti věnováno více zdrojů.

Druhou možností jsou nástroje pro správu dokumentů v cloudu. Tyto nástroje jsou přístupné prostřednictvím webového prohlížeče, protože jsou hostovány vzdáleně. Zatímco cloudové systémy pro správu dokumentů mají zjednodušený přístup a navigaci, v některých případech mohou postrádat rozsáhlé možnosti přizpůsobení.

Nástroje pro správu dokumentů mají pro podniky řadu výhod. Jako například:

- **Zvýšená efektivita**: Díky využití řešení pro správu dokumentů lze automatizovat podřadné úkoly, jako je třídění a načítání souborů, a tím poskytnout pracovníkům více času věnovat se naléhavým záležitostem.
- **Optimalizace produktivity** je podpořena implementací nástrojů pro správu dokumentů, které organizacím umožňují rychlý přístup k dokumentům a jejich obnovu, což vede ke zkrácení doby dokončení úkolu a také ke zvýšení celkové efektivity.
- **Zvýšená bezpečnost**: Nástroje pro správu dokumentů zpřístupňují robustní bezpečnostní funkce, například uživatelská oprávnění a řízení přístupu, aby bylo zaručeno, že oprávnění k přístupu k důvěrným dokumentům mohou mít pouze oprávnění pracovníci.

- **Sofistikovaná spolupráce**: S využitím softwaru pro správu dokumentů mohou různí jednotlivci přistupovat k jedinému dokumentu a upravovat jej najednou. To zlepšuje týmovou práci a spolupráci mezi uživateli.

3.5.5 Nástroje pro komunikaci

Úspěch projektů vývoje softwaru závisí na schopnosti efektivně komunikovat, zejména proto, že tyto projekty bývají komplikované a zahrnují velké množství jednotlivců s různými zájmy. V softwarových projektech se technický personál většinou zabývá skutečnými technickými artefakty projektu. Tyto technické artefakty zahrnují zdrojový kód softwaru, konfigurační soubory, soubory infrastruktury (např. skripty sestavení) a testovací případy. V mnoha případech jsou technické artefakty v softwarových projektech složité a obvykle obtížně vyjádřitelné verbálně, protože se obvykle skládají ze zkratek, klíčových slov a speciálních znaků, které nejsou součástí našich přirozených mluvených jazyků, nebo na ně odkazují.[22]

Pro usnadnění spolupráce a sdílení informací mezi týmy jsou komunikační nástroje nezbytné, minimalizují tak výskyt mylných představ nebo nepřesností. Tato kapitola se snaží prozkoumat typické komunikační metody často používané během úsilí o vývoj softwaru a zároveň pojednává o jejich výhodách a vlastnostech.

Použití komunikačních nástrojů má řadu výhod. Nejen, že zvyšuje celkovou produktivitu, ale také snižuje chyby a nedorozumění tím, že usnadňuje přehlednost v korespondenci. Tyto nástroje poskytují jedinečné místo pro projektovou dokumentaci, zvyšují její kvalitu a usnadňují sdílení informací a také jejich bezproblémové sledování. V konečném důsledku tyto aplikace umožňují větší transparentnost mezi zúčastněnými stranami, což umožňuje hladší aktualizace průběhu všech problémů souvisejících s projekty.

Při vývoji softwaru máme k dispozici řadu komunikačních nástrojů. Příklady zahrnují nástroje pro videokonference, platformy pro řízení projektů a systémy pro spolupráci na kódu. Programy pro rychlé zasílání zpráv, jako je *Slack* [1] nebo *Microsoft Teams*[2], se ukázaly jako efektivní pro rychlou a snadnou výměnu informací mezi členy týmu bez ohledu na jejich polohu nebo časová pásma, ve kterých se nacházejí. Mezitím virtuální schůzky umožňují spolupráci mezi vzdálenými týmy v reálném čase prostřednictvím aplikací,

[1] https://slack.com
[2] https://teams.microsoft.com

jako je *Zoom*[3], *Skype*[4] nebo *Google Meet*[5], které jsou vhodné při jednání s různými místy po celém světě. Kromě těchto existujících aplikací zmíněných výše je nabídka vysoce účinných organizátorů úkolů nazvaných *Trello* spolu s dalšími podobnými, včetně *Asana* a *Jira*, které pomáhají sledovat pokrok v projektech a zároveň podporují víceúrovňovou účast zainteresovaných stran, což je ideální pro řízení sofistikovaných snah.

3.6 Dolování dat

Jednou z metod využívaných pro získávání cenných informací z rozsáhlých souborů dat je metoda známá jako dolování dat (data mining). Využitím technik, jako je strojové učení, statistická analýza a vizualizace dat, je možné ve shromážděných informacích odhalit vzorce a trendy. Díky své schopnosti poskytovat vhled do shromážděných dat pro účely lepšího rozhodování podniků se stalo dolování dat nepostradatelným nástrojem v dnešním světě obchodu a řízení společností.

Praxe získávání smysluplných informací z velkých souborů dat, běžně známá jako dolování dat, nachází několik aplikací v různých odvětvích. Jeho potenciál lze využít k rozpoznání nákupních tendencí a zvyků, odhalení podvodů nebo nesrovnalostí v transakcích a zároveň předpovídání budoucích změn v postojích spotřebitelů. Stejně tak je užitečný pro analýzu trendů akciového trhu identifikací vzorců, které mohou ovlivnit jeho výkonnost pozitivně nebo negativně. Dolování dat může pomoci analyzovat potřeby a preference zákazníků, což usnadňuje vytváření projektů, které lépe vyhovují jejich očekáváním. Tímto způsobem lze získat lepší pochopení o tom, jaký druh funkcionality a uživatelského rozhraní bude nejatraktivnější pro cílovou skupinu. Účetní otázky týkající se financí mohou z tohoto procesu rovněž velmi těžit, přičemž anomálie budou rychle odhaleny a vyřešeny prostřednictvím kontroly analýzy finančních záznamů. Podobně zkoumání databází zdravotní péče pomocí metod dolování dat pomáhá odhalit relevantní poznatky o léčebných postupech a dokonce i epidemiologických opatřeních přijatých proti konkrétním nemocem přítomným v dané komunitě/skupině populace/atd. – to vše bez lidského zásahu.

Dolování dat není bez problémů. Jednou z největších výzev je množství dat, které je potřeba zpracovat. Soubory dat mohou být velmi velké a složité, takže je obtížné identifikovat vzory a trendy. Dolování často probíhá za po-

[3]https://zoom.us/
[4]https://www.skype.com/
[5]https://meet.google.com/

moci ETL (Extract-Transform-Load) procesu, tedy extrakce, transformace a načtení.[8]

Dalším problémem, který vzniká, je přesnost dat. Není neobvyklé, že datové sady obsahují chyby a nesrovnalosti, což z nich činí impozantní úkol z hlediska odhalování opakujících se vzorců nebo sklonů.

3.6.1 Dolování dat z OSS projektů

Získávání cenných poznatků a vzorů z datových sad souvisejících s vývojem softwaru, včetně úložišť kódu, systémů pro sledování chyb, e-mailových konferencí a fór, je podstatou dolování dat v projektech softwaru s otevřeným zdrojovým kódem. Prostřednictvím tohoto procesu analýzy mohou vývojáři činit informovaná rozhodnutí zkoumáním trendů v rámci těchto různých zdrojů informací.

Prostřednictvím dolování dat v open-source softwarových projektech lze využít řadu výhod. Jednou z nejvýznamnějších výhod je schopnost rozpoznat vzory a trendy v procesu vývoje softwaru. Skvělým příkladem by byla identifikace známých metod kódování nebo detekce chyb v rané fázi při jeho vytváření. Pozorování také umožňuje uznání těm vývojářům, kteří nabízejí vstupy do projektu určením jejich významu pro něj, čímž je odpovídajícím způsobem odměňuje na základě úrovně jejich příspěvku.

Jedním z výhodných aspektů využití dolování dat v rámci open-source softwarových projektů je potenciál detekovat a zkoumat chyby v programování. Prostřednictvím důkladného prozkoumání hlášení o chybách spolu s příslušnými informacemi má dolování dat schopnost pomoci vývojářům rozpoznat převládající vzorce mezi chybami, sledovat, jak často se tyto problémy během určitého období vyskytují, a také určit, odkud tyto problémy pocházejí.

Kromě toho může využití síly dolování dat vést k lepšímu řízení projektů v podnicích s otevřeným zdrojovým softwarem. Analýza dat zahrnující aspekty, jako jsou hlášení chyb, časové osy a organizace úkolů, umožňuje manažerům určit problémová místa, která vyžadují pozornost, a současně je zlepšovat. Využití této techniky navíc povede k určení, kdy by mohly být projekty dokončeny, pomocí předvídání časů dokončení, čímž se celkově zlepší plánování těchto iniciativ.

V oblasti vývoje softwaru existuje mnoho nástrojů s otevřeným zdrojovým kódem, ze kterých si lze vybrat, pokud jde o dolování dat. Některé příklady zahrnují *Python, R a Weka*. Tyto nástroje umožňují důkladnou analýzu jakékoli dané datové sady za účelem získání cenných informací, jako jsou trendy nebo vzorce, které nemusí být na první pohled patrné. S těmito funk-

cemi platformy mají uživatelé velký užitek z řady nabídek, včetně možností strojového učení, statistické analýzy a lepšího porozumění prostřednictvím vizuálních reprezentací.

Stručně řečeno, využití dolování dat v rámci open-source softwarových projektů představuje řadu výhod pro vývojáře, vedoucí projektů a společnosti. Odhalením vzorců a trendů v procesu vývoje softwaru je možné včas identifikovat chyby a zároveň zlepšit postupy řízení. Softwarové nástroje, které jsou přístupné prostřednictvím platformy s otevřeným zdrojovým kódem, mají různé možnosti, pokud jde o analýzu dat z různých fází vytváření produktu; takže jejich integrace do praxe je v moderních programovacích prostředích stále naléhavější. Vzhledem k tomu, že tyto typy kolaborativních programů stále rostou v oblibě mezi spotřebiteli i profesionály, bude potřeba spolehlivých informací získaných využitím technologie tohoto typu růst, zejména pokud jde o informované rozhodovací procesy zaměřené na zvýšení kvality na všech úrovních souvisejících s konkrétním programem nebo aplikace se vyvíjí.

3.6.2 Dolování dat z ALM

V dnešní době jsou data většiny OSS projektů rozprostřena v různých ALM nástrojích. Z těchto nástrojů je data nejprve nutno získat a následně je zpracovat. Dolování dat jednotlivých nástrojů není vždy jednotný a stejný přístup nelze použít pro všechny nástroje. Nejzákladnějším přístupem k dolování dat je připojení přímo k databázi, kde nástroj ukládá informace. S výjimkou serverů, které vlastníme, je přímý přístup k databázi jen zřídka možností. Téměř všechny repozitáře nástrojů ALM však poskytují nějaký druh API a určitě všechny ty, které jsme vybrali, poskytují. Ve všech případech je k dispozici minimálně REST API a ve většině případů i API pro Java4 a další programovací jazyky. Většina nástrojů také podporuje export dat v různých formátech, jako je JSON, XML nebo prostý text. Nevýhodou tohoto přístupu je, že kvůli problémům specifickým pro nástroj je často nutné exporty provádět ručně nebo vyžadují další kroky, a tedy větší úsilí s sotva lepšími výsledky ve srovnání s API.[25]

3.6.3 Vzory a anti-vzory

Při vývoji softwaru jsou vzory (patterns) a anti-vzory (anti-patterns) důležitými pojmy pro pochopení toho, jak vytvořit vysoce kvalitní software. Vzor je osvědčené řešení běžného problému, které lze použít v mnoha situacích. Anti-vzor je naproti tomu řešením běžného problému, které je neúčinné nebo

kontraproduktivní. [20]

Oblast softwarového designu běžně implementuje vzory, které pomáhají vývojářům řešit rozsáhlé problémy s přesností a produktivitou. Prostřednictvím těchto jedinečných konfigurací lze efektivně zpracovávat komplexní pracovní postupy, efektivně spravovat datové struktury a bezproblémově řešit chyby. Zdokumentovaný plán používání vzorů převládá v celé komunitě pro své optimální postupy umožňující různé strategie řešení problémů.

Návrhový vzor, který se běžně implementuje při vytváření softwaru, je vzor Model-View-Controller (MVC). Jeho účelem je rozdělit odpovědnosti v rámci aplikace, konkrétně pokud jde o správu dat, prezentaci uživatelského rozhraní a procesy zpracování řídicí logiky. Tím vytváří definovaný rámec pro uspořádání kódu, který usnadňuje vývojové komplikace.

Na druhou stranu mohou vznikat kontraproduktivní řešení známá jako anti-vzory. Ať už záměrně, nebo ne, vývojáři se mohou pokusit vyřešit problém, aniž by pochopili jeho hlavní problémy, a vybrat v procesu nevhodná řešení. Implementace anti-vzory často vytváří problematický kód, který je náročný na udržení a je náchylný k chybám a zároveň je přinejlepším neefektivní.

The God Object je notoricky známý anti-vzor ve světě vývoje softwaru. Tento vzor odkazuje na singulární objekt aplikace, který přebírá všechny funkce, což způsobuje vážné problémy při údržbě a testování optimalizace kódu. Zásada oddělení zájmů je tímto chybným přístupem porušena, což ztěžuje kodérům efektivní plnění jejich úkolů.

Je možné kombinovat jak vzory, tak anti-vzory, což pomáhá při identifikaci a řešení běžných problémů s vývojem softwaru. Návrhové vzory, pokud je vývojáři pochopí a implementují, mohou vytvářet kód, který je lépe strukturovaný, optimalizovaný a také udržitelnější. Na druhou stranu, vyhýbání se anti-vzorům povede ke snížení možností chyb při celkovém zjednodušení vývojových postupů. V důsledku toho je možné zvýšit celkovou kvalitu kódu.

Vždy je rozumné hledat v projektech zákonitosti a nesrovnalosti. Získá se tak ucelený pohled na situaci, kdy ji lze s rozvahou vyřešit a dbát na to, aby se případné komplikace řešily okamžitě, jakmile nastanou.

3.6.4 SPADe

Je důležité se zaměřit na Software Process Anti-pattern Detector (SPADe), jenž je inovativním nástrojem vyvinutým na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni pro detekci anti-vzorů softwarových procesů. Jeho ústředním konceptem je těžit data z úložišť softwarových projektů v různých nástrojích ALM a analyzo-

vat je na přítomnost anti-vzorů, opakujících se událostí, chování, konceptů a metod, se zvláštním důrazem na anti-vzory, tedy vzorce, jejichž výskyty mají negativní dopad na projekt a jeho produkt. Primární motivací je spojit potenciál obrovských objemů projektových dat uložených v nástrojích ALM s teoretickým návodem na procesy, postupy a (anti-)vzory automatizovaným způsobem, aby se zlepšila úspěšnost projektů a znalosti v oboru.

Struktura nástroje SPADe je rozdělena do čtyř částí:

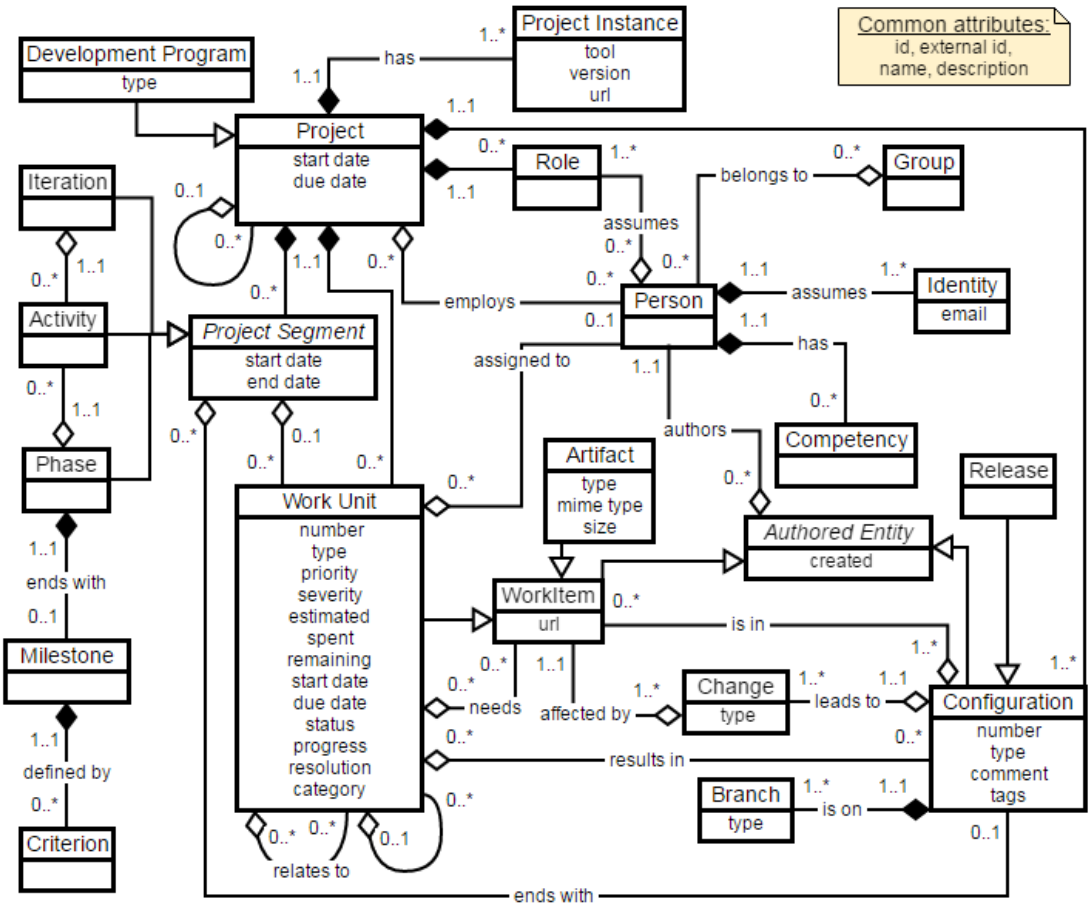
Dolování dat

Má za úkol získávat data z jednotlivých ALM nástrojů pomocí standartního ETL procesu. Z důvodu nejednotnosti rozhraní jednotlivých nástrojů jsou nadefinovány všem nástrojům vlastní ETL datové pumpy. ETL je proces, při kterém jsou získávána data z jednoho nebo více systémů a transformována do jednotné formy, ve které jsou následně uložena. [15]

Datový sklad

Databázový systém jedinečného druhu, známý jako datový sklad, ukládá a vyhodnocuje velké množství informací pocházejících z různých kanálů.[1] Primární funkcí je zásobit detaily z operačních systémů – například záznamy o zákaznických transakcích – a později je sezónně prozkoumat se záměrem získat znalosti o provádění obchodu. Sloučení mezi relačními databázemi vedle OLAP kostek často vytváří datové sklady společně s jejich menšími edicemi označovanými jako „datové tržiště“.

Datový sklad nástroje SPADe má jasně definovanou strukturu. Datový sklad nástroje SPADe obsahuje celkem 47 entit, které obsahují souhrnná data o jednotlivých projektech získaná z nástroje ALM. Datový sklad v aktuální verzi nástroje SPADe je více podobný standardní relační databázi. Základní struktura datového skladu je znázorněna pomocí jeho zjednodušeného doménového metamodelu na obrázku 3.3. Úplný fyzický datový model je příliš komplexní a není relevantní pro tuto práci, proto v ní není uveden.



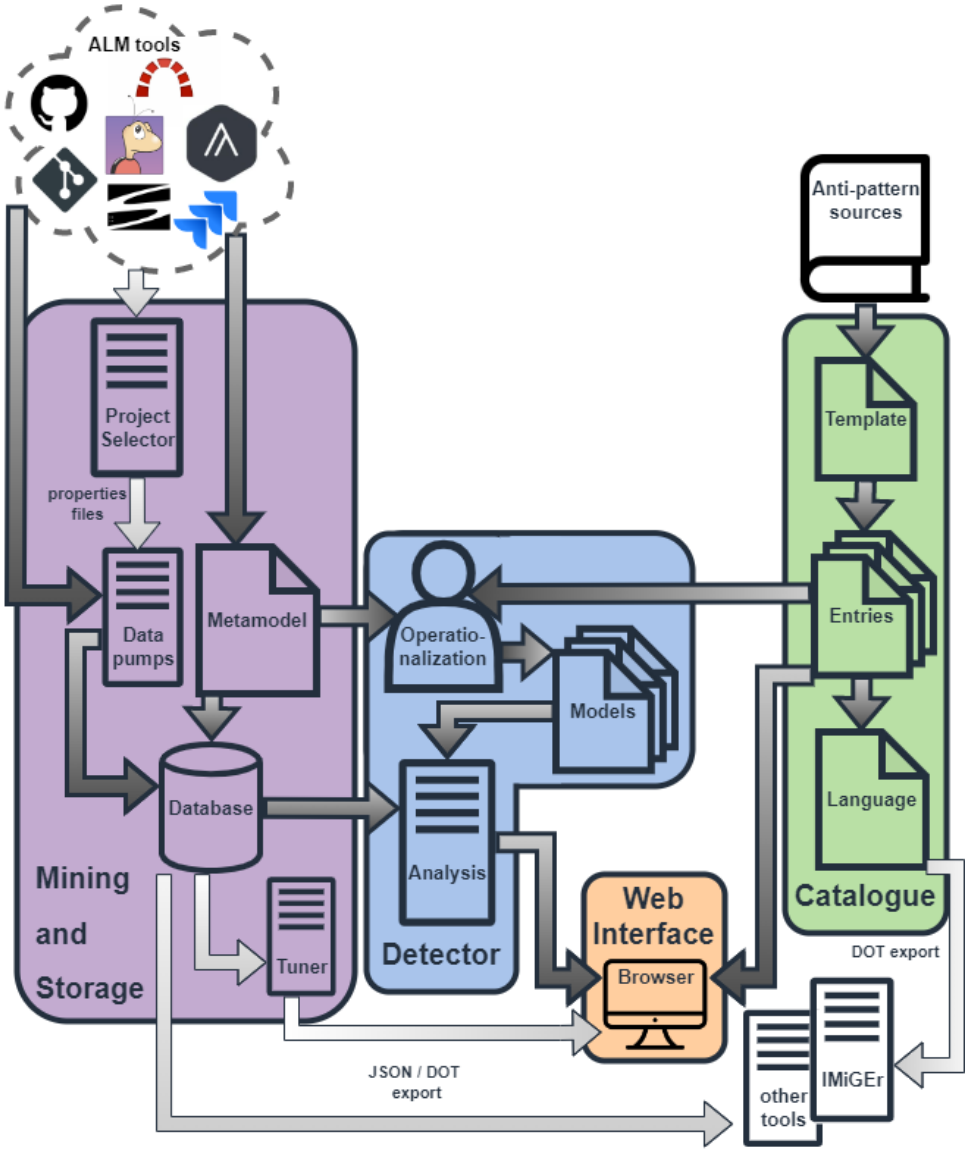
Obrázek 3.3: Doménový model datového skladu SPADe [25]

Aplikační vrstva

Mezivrstva, ve které jsou zpracovávány výsledky dotazů z datového skladu a ty jsou následně předávány prezentační vrstvě.

Prezentační vrstva

Uživatelská část umožňující uživateli procházet uložená data a zobrazování různých metrik a statistik nad daty.



Obrázek 3.4: Architektura nástroje SPADe [24]

Na obrázku 3.4 lze vidět schéma systému SPADe s již implementovanou pomocnou aplikací `Projekt Selector`, jejíž implementace je součástí této diplomové práce. Tato aplikace je podrobně popsána v kapitolách 5 a 6.

4 Analýza charakteristik OSS projektů

Tato sekce se věnuje různým charakteristikám OSS projektů. Charakteristik existuje mnoho a každá z nich je jinak důležitá pro finální výsledek projektu. Abychom mohli nástroj SPADe zásobovat vhodně vybranými projekty, je nutné nejdříve provést analýzu těchto charakteristik a určit, jaké jsou více a jaké jsou méně důležité. Pro analýzu byly zvažovány a dále prozkoumány nástroje *GitHub*, *GitLab*, *Jira*, *Redmine*, *Assembla*. Pro průzkum sloužily povětšinou volně přístupné dokumentace a možnosti API jednotlivých nástrojů. Po důkladném průzkumu byl z těchto nástrojů vybrán ten, který se jevil jako nejvhodnější. K analýze charakteristik bylo tedy využito výhradně možností nástroje *GitHub*, jelikož se jedná o jednu z nejrozšířenějších variant internetových hostingových služeb a nabízí tedy obrovské množství volně přístupných projektů. Také obsahuje velké množství různých charakteristik, které jsou vhodné pro využití k dolování dat OSS projektů. V případě analýzy GitHubu sloužily jeho funkce (funkce nabízené správou repozitářů a možnosti API) a jeho poměrně dopodrobna zpracovaná on-line dokumentace.

Platforma	**Všechny projekty**	**OSS projekty**	**Počet uživatelů**	**API**
GitHub	> 413 milionů	> 50 milionů	> 100 milionů	Ano
GitLab	> 200 000	> 30 000	> 30 milionů	Ano
Jira	Nezveřejněno	Nezveřejněno	> 180 000	Ano
Assembla	Nezveřejněno	Nezveřejněno	> 1 milion	Ano
Redmine	Nezveřejněno	Nezveřejněno	Nezveřejněno	Ano

Tabulka 4.1: Orientační porovnání platforem

Jednotlivé atributy tabulky 4.1 až na GitHub není snadné dohledat. Společnosti tyto informace buď nezveřejňují a nebo tyto informace vůbec nevedou. Hodnoty v tabulce jsou proto z několika zdrojů v časovém rozmezí 2021-2023. Hodnoty jsou pouze informační, nicméně nepoměr mezi platformami je stále velmi výrazný. Ze srovnání GitHub jasně vyplývá jako největší

hostitel zdrojového kódu a i OSS projektů na světě.

Následují jednotlivé charakteristiky projektů a jejich bližší analýza.

4.1 Visibility

Způsob, jakým uživatelé mohou projekt najít, používat a interagovat s ním, se nazývá jeho viditelnost (visibility). Nástroje pro správu projektů nabízí různé funkce a nástroje pro vylepšení úrovně viditelnosti, kterou má projekt.

Zlepšení objevitelnosti projektu na volně dostupném úložišti lze dosáhnout zajištěním toho, že bude mít expresivní a výstižný název. Název by měl ztělesňovat jak svůj cíl, tak vlastnosti, aby uživatelé rychle pochopili, co projekt obnáší. Srozumitelný název nejen zvyšuje snadnou dostupnost, ale také usnadňuje uživatelům vyhledávání a porovnávání podobných projektů na platformě.

Jeden zásadní prvek, který má dopad na viditelnost projektu, se soustředí na způsob, jakým je prezentován jeho popis (description). Popisný text hraje zásadní roli, protože umožňuje uživatelům zúžit vyhledávání pomocí specifických klíčových slov, která jsou pro ně relevantní, což umožňuje cílenější výsledky z mnoha dostupných projektů. Výstižného, ale komplexního shrnutí toho, co projekt obnáší a nabízí, lze dosáhnout pomocí dobře zpracovaných popisů. Měl by jasně předávat potřebné informace a zároveň zahrnovat použitelné fráze nebo značky spojené výslovně s daným projektem, to vše v omezeném prostoru, což maximalizuje účinnost.

GitHub nabízí řadu nástrojů pro zlepšení viditelnosti projektu, jako jsou témata (topics) a štítky (tags). Tyto štítky poskytují účinný prostředek pro kategorizaci projektů a zefektivnění interakce uživatelů s nimi prostřednictvím organizace. Pro další ilustraci, přiřazení stavu „open-source“ danému projektu vyjasňuje jeho povahu již z definice.

Soubor README projektu GitHub má významnou váhu. Slouží jako počáteční bod interakce mezi uživatelem a projektem a umožňuje vlastníkům poskytnout úvod nebo nabídnout pokyny k instalaci spolu s dalšími souvisejícími detaily.

Je důležité, aby osoba, která má na starosti projekt, zvážila, zda bude jejich úložiště přístupné komukoli, nebo zda bude omezeno pouze na osoby s povolením. Toto je jedna ze zásadních voleb, která musí být učiněna během procesu tvorby. Majitel by měl zvážit výhody i nevýhody před výběrem jedné z možností, protože s každou možností jsou spojeny významné výhody a nevýhody, které by mohly mít dopad nejen na něj samotné, ale i na ostatní, kteří mohou využívat nebo přispívat k jejich práci prostřednictvím této plat-

formy.

Repozitáře na GitHubu, které jsou veřejné, může snadno prohlédnout a popřípadě stáhnout kdokoliv. V zásadě to znamená, že každý má svobodu procházet a analyzovat kód, který je v něm přítomen, větvit se z něj nebo vytvářet požadavky na stažení pro tato úložiště. Veřejně dostupné zdroje pomáhají podporovat komunitní vstupy do projektů s otevřeným zdrojovým kódem, což z nich činí vhodnou volbu s ohledem na spolupráci. Kromě tohoto přínosu, který spočívá v usnadnění příspěvků komunity a stimulaci synergického úsilí, sdílení vlastního úložiště na takových platformách dále přispívá k rozšiřování portfolií vývojářů, což jim v konečném důsledku pomáhá získat lepší pracovní příležitosti, které mohou vyhovovat jejich dovednostem.

Soukromé projekty vyžadující omezený přístup se nejlépe hodí pro soukromá úložiště GitHub, která zviditelní kód pouze uživatelům autorizovaným vlastníky projektů. Osobní experimenty, stejně jako vlastní nebo důvěrné závazky, které vyžadují kontrolu nad sledováním kódu, mohou těžit z používání těchto exkluzivních úložišť. Veřejné šíření v takových případech nepřichází v úvahu, a proto se soukromé úložiště, jako je toto, hodí se svými jedinečnými bezpečnostními opatřeními, která za všech okolností zajišťují diskrétnost.

Pozoruhodný kontrast mezi veřejnými[1] a soukromými repozitáři spočívá v zabezpečení, které nabízejí. Soukromá nastavení jsou obecně bezpečnější než jejich veřejné protějšky, protože neomezený přístup může jednotlivcům umožnit identifikovat mezery v systému nebo se do nich snadno nabourat. Při práci s open-source softwarem je tedy nutné přísně dodržovat doporučená bezpečnostní opatření; omezovat vystavení citlivým údajům tam, kde je to možné, a zároveň pravidelně kontrolovat potenciální zranitelnosti prostřednictvím pravidelných auditů dostupného kódu. Z toho tedy vyplývá, že tato výše uvedená rizika by mohla být jedním primárních impulsů o pokusy o identifikaci vzorů nebo anti-vzorů v metodikách analýzy projektu.

4.2 Živost projektu

Pouhá přítomnost projektu na úložišti automaticky neznamená, že je „živý“ nebo aktivní. Živost projektu závisí na několika faktorech, včetně četnosti commitů, sledování problémů, zapojení komunity a činností údržby.

Projekt na GitHubu, který je „živý“, je ten, který je pravidelně aktualizován o nové změny v kódové základně. To představuje důkaz, že správci projektu aktivně usilují o vylepšení a přidání nových funkcí do svého softwa-

[1] K analýze charakteristik se využívají veřejné repozitáře, protože jsou přístupné bez výslovného povolení od příslušných správců.

rového systému. I když by se mohlo zdát, že frekvence aktualizací odpovídá rozsahu a složitosti provádění projektů, delší období bez jakékoli aktivity může znamenat, že již neprobíhají postupy vývoje nebo údržby.

Přítomnost otevřených úkolů a jejich sledování může také naznačovat živost projektu. Sledování úkolů (issues) je systém, který uživatelům umožňuje hlásit chyby, navrhovat nové funkce a žádat o podporu. Projekt, který je naživu, tak bude mít aktivní sledovač problémů s nedávnou aktivitou, jako jsou nahlášené, vyřešené nebo aktualizované nové úkoly. Vlastníci nebo správci projektů by navíc měli aktivně reagovat na úkoly, buď je uznat, nebo poskytnout řešení, aby uživatelé věděli, že jejich zpětná vazba je brána vážně.

Zapojení komunity může hrát klíčovou roli při udržení projektu. Prosperující úsilí vyžaduje zapojení a účast uživatelů, kteří nejen zasílají pull-requesty, ale také poskytují podporu a aktivně ji propagují prostřednictvím platforem sociálních médií nebo jiných kanálů, které mají k dispozici.

Akty aktualizace závislostí, zlepšení bezpečnostních zranitelností a optimalizace výkonu jsou akce, které demonstrují vitalitu projektu. Tyto operace sdělují přihlížejícím, že ti, kdo dohlížejí na projekt nebo jej řídí, vynakládají úsilí na jeho vylepšení a zároveň drží krok s avantgardními technologiemi a technikami. Důslednou prací na uvedeném snažení zaručuje jeho účelnost a relevanci pro publikum, které jej často využívá. Uživatelé mohou navíc přispět k živosti projektu poskytováním zpětné vazby, hlášením problémů a propagací projektu ostatním.

Aktivní přístup vlastníků projektu můžeme sledovat pomocí některých GitHub charakteristik, jako například datum posledního pushnutí, poslední aktualizace, posledního commitu.

Úložiště na GitHubu mohou být archivovány. Když je úložiště archivováno, jeho issues, žádosti o stažení, kód, štítky, milníky, projekty, wiki, vydání, odevzdání, značky, větve, reakce, upozornění na skenování kódu, komentáře a oprávnění se stanou read-only. Chce-li kdokoliv provést změny v archivovaném úložišti, musí se nejdříve archivace zrušit.[10]

Když je projekt archivován, znamená to, že již není aktivně udržován nebo vyvíjen vlastníkem nebo správcem projektu. Archivovaný stav je v podstatě způsob, jak mohou majitelé projektu signalizovat, že se na projektu již nepracuje a uživatelé by neměli očekávat žádné další aktualizace nebo činnosti údržby (lze jej tedy považovat za „mrtvý“ či pozastavený).

Charakteristiky živosti celého repozitáře jsou důležité pro vyhledávání mezi nimi a jejich následné další zpracování.

4.3 Velikost projektu

Velikost projektu se primárně týká velikosti jeho kódové základny nebo počtu souborů a řádků kódu v úložišti. Větší kódová základna může být náročná na správu a údržbu a může být obtížnější identifikovat a opravit chyby a problémy. Naproti tomu menší kódová základna je lépe zpracovatelná, snáze se udržuje a je méně náchylná k chybám a problémům s výkonem.

Velká kódová základna může ovlivnit výkon projektu, takže je pomalejší a hůře reaguje. Když například uživatel požaduje stránku nebo funkci z projektu na GitHubu s velkou kódovou základnou, server musí načíst a zpracovat všechny potřebné soubory, což může trvat značnou dobu. To může vést k pomalejšímu načítání, sníženému uživatelskému dojmu a nižšímu zapojení.

Navíc velká kódová základna může také ovlivnit použitelnost projektu. Pro nové přispěvatele může být obtížnější porozumět architektuře projektu a tomu, jak funguje. To může mít za následek nižší počet přispěvatelů a menší zapojení komunity. Naproti tomu menší kódová základna bývá snáze pochopitelná a noví přispěvatelé se mohou rychleji zorientovat, což vede k aktivnější komunitě.

Dále ovlivňuje velikost projektu jeho údržbu. S tím, jak se kódová základna rozrůstá, se údržba projektu stává náročnější a chyby může být obtížnější identifikovat a opravit. Také je obtížnější přidávat nové funkce nebo zlepšovat výkon projektu. V důsledku toho vyžadují větší projekty více času a zdrojů na údržbu a je pravděpodobnější, že časem zastarají. Výhodou větších projektů mohou být bohatší funkce, což může přilákat větší uživatelskou základnu. Je na majitelích a správcích projektu, aby našli správnou rovnováhu mezi funkčností a ovladatelností, aby byl zajištěn úspěch a udržitelnost jejich projektu. Velikost projektu může ovlivnit následné těžení dat různými způsoby. Větší projekty mohou obsahovat více dat a informací, které lze analyzovat a vytěžit. To může znamenat, že těžení dat může trvat déle a vyžadovat více výpočetního výkonu. Na druhou stranu, větší projekty mohou také poskytnout více užitečných informací a náhledů.

GitHub se snaží poskytovat dostatek úložného prostoru pro všechna úložiště Git, i když existují přísné limity velikost souborů a úložišť. Aby uživatelům zajistili výkon a spolehlivost, aktivně monitorují signály celkového stavu úložiště. Stav úložiště je funkcí různých vzájemně se ovlivňujících faktorů, včetně velikosti, frekvence potvrzení, obsahu a struktury. GitHub omezuje velikost souborů povolenou v úložištích. Pokud se uživatel pokusí přidat nebo aktualizovat soubor, který je větší než 50 MB, obdrží od Gitu varování. Změny se stále úspěšně promítnou do jeho úložiště, ale může zvážit odstranění potvrzení, aby minimalizoval dopad na výkon.[12]

4.4 Commits

Jednou z nejdůležitějších funkcí GitHubu je schopnost zadávat změny do úložiště. Potvrzení (Commits) jsou stavebními kameny správy verzí, které umožňují vývojářům sledovat změny provedené v jejich kódové základně v průběhu času.

V Gitu je commit sada změn provedených v úložišti. Každé potvrzení má jedinečný identifikátor, zprávu o odevzdání a časové razítko. Potvrzení se používají ke sledování změn provedených v kódové základně a jsou nezbytné pro spolupráci mezi více vývojáři.

Když vývojář provede změny v souboru v úložišti, musí nejprve vybrat jím vytvořené změny, které chce potvrdit. Zavedení změny znamená, že vývojář přidal změny do seznamu změn, které budou zahrnuty do příštího potvrzení. Jakmile jsou změny připraveny, může vývojář vytvořit potvrzení poskytnutím zprávy o potvrzení, která popisuje provedené změny.

V GitHubu se potvrzení vytváří pomocí rozhraní příkazového řádku Git nebo grafického uživatelského rozhraní (GUI), které poskytuje GitHub, popřípadě ve vývojovém prostředí daného jazyka. Vývojáři mohou použít webové rozhraní GitHub k zobrazení historie změn repozitáře a porovnat je v průběhu času.

Pravidelným prováděním změn mohou vývojáři vytvořit historii projektu, kterou lze použít k pochopení toho, jak se kódová základna vyvíjela. Tato historie je také užitečná pro odstraňování problémů a identifikaci, kdy byla zavedena chyba.

Kromě sledování změn jsou commity také důležité pro spolupráci mezi více vývojáři. Když více vývojářů pracuje na stejné kódové základně, revize jim poskytují způsob, jak sledovat změny provedené ostatními a zajistit, aby všichni pracovali ze stejné kódové základny. Vytvořením popisných zpráv potvrzení mohou vývojáři komunikovat změny provedené v kódové základně a poskytnout kontext ostatním vývojářům.

Počet commitů v úložišti GitHub může poskytnout určitý přehled o úrovni aktivity a historii úložiště. Zde je několik stanovisek o projektu, která může naznačovat počet commitů v úložišti:

1. **Aktivní vývoj**
   Pokud má úložiště vysoký počet potvrzení, znamená to, že se projekt aktivně vyvíjí. Čím více potvrzení má úložiště, tím je pravděpodobnější, že dochází k častým aktualizacím a vylepšením kódové základny.

2. **Kvalita kódu**
   Úložiště s velkým počtem revizí by také mohlo naznačovat, že vývojáři

aktivně provádějí změny, aby zlepšili kódovou základnu. Častá potvrzení mohou naznačovat, že tým pravidelně kontroluje a aktualizuje kód, aby řešil chyby, optimalizoval výkon a implementoval nové funkce.

3. **Spolupráce**
   Vysoký počet commitů by také mohl naznačovat, že projekt má zdravou a aktivní komunitu přispěvatelů. To může být pro projekt pozitivní znamení, protože to ukazuje, že komunita je zapojena a investuje do rozvoje projektu.

4. **Složitost**
   V některých případech může být velký počet potvrzení varovným signálem, že kódová základna je příliš složitá nebo obtížně udržovatelná. To může nastat v případě, že potvrzení nejsou dobře zorganizovaná a nedodržují osvědčené postupy pro potvrzení změn.

Zatímco počet commitů v úložišti může poskytnout určitý pohled na úroveň aktivity a historii projektu, je důležité si uvědomit, že tato samotná charakteristika nemusí nutně indikovat kvalitu nebo úspěch projektu.

Commity v OSS projektech mohou být využity pro těžení dat k odhalování vzorů a anti-vzorů v softwarovém vývoji. Analýza commitů umožňuje identifikovat osvědčené postupy (vzory) a špatné řešení problémů (anti-vzory) ve zdrojovém kódu, což může pomoci zlepšit kvalitu kódu a celkovou efektivitu projektu. Existuje velké množství použití commitů v dolování dat, například:

- Nástroje pro textovou analýzu mohou zpracovat popisky commitů a další související texty, aby identifikovaly klíčová slova, témata a koncepty, které se objevují v průběhu vývoje projektu. Tyto informace mohou pomoci odhalit oblasti zájmu, problémy a řešení používaná vývojáři.

- Lze využít analýzu časových řad. Tato metoda sleduje časovou návaznost commitů, aby zjistila trendy a vzory v aktivitě projektu. Analyzováním časových řad můžeme zjistit, jak se mění frekvence commitů, jaké části projektu jsou nejvíce aktivní a jak se v průběhu času mění kvalita kódu.

- Nástroje pro statickou analýzu kódu mohou procházet zdrojový kód a identifikovat časté problémy, chyby a nekonzistence. Tyto nástroje mohou pomoci odhalit anti-vzory v kódu a upozornit na oblasti, které vyžadují zlepšení.

- Vytváření vizuálních reprezentací dat z commitů může usnadnit identifikaci vzorů a anti-vzorů. Vizualizace mohou zahrnovat grafy, heatmapy nebo jiné vizuální reprezentace, které zobrazují informace o změnách, přispěvatelích a kvalitě kódu v čase.

- Sémantická analýza může pomoci odhalit, jak jsou jednotlivé části kódu propojeny a jak se navzájem ovlivňují. Tato metoda spočívá v prozkoumání významu kódu a jeho souvislostí s ostatními částmi projektu. Identifikace takových vztahů může poskytnout užitečný vhled do vzorů a anti-vzorů v projektu.

- Porovnávání commitů a metrik z různých OSS projektů může poskytnout užitečný kontext pro identifikaci vzorů a anti-vzorů. Tímto způsobem lze zjistit, jak se konkrétní projekt vyvíjí ve srovnání s ostatními a jak se vývojáři vypořádávají s podobnými problémy a výzvami.

4.5 Hvězdy

Hvězdy (Stars) jsou důležitou metrikou pro měření popularity projektu. Představují způsob, jak mohou uživatelé označit projekt, který považují za zajímavý, užitečný nebo působivý. Jedná se o jednoduchý a lehký způsob, jak mohou uživatelé ukázat své uznání projektu, aniž by potřebovali zdlouhavou recenzi nebo zpětnou vazbu. Hvězdy jsou navíc veřejnou podporou projektu, což může pomoci přilákat více uživatelů a přispěvatelů do projektu.

Samotný počet hvězd však k hodnocení kvality nebo užitečnosti projektu Github nestačí. Vysoký počet nutně neznamená, že je projekt dobře navržený, dobře zdokumentovaný nebo bez chyb. Stejně tak nízký počet nemusí nutně znamenat, že projekt není užitečný nebo působivý. Proto je důležité při hodnocení jeho kvality nebo užitečnosti vzít v úvahu kontext, ve kterém byl projekt označen. Například projekt, který byl označen hvězdou velkým počtem uživatelů, zejména těch, kteří jsou odborníky v příslušné oblasti, může naznačovat, že projekt je dobře navržený, dobře zdokumentovaný a užitečný. Na druhou stranu velký počet hvězd od uživatelů, kteří nejsou odborníky v příslušné oblasti, nemusí být dobrým ukazatelem kvality nebo užitečnosti projektu.

Hvězdy tedy slouží zejména jako ukazatel popularity a rozšířenosti projektu. Jedná se tedy o jednu z nejdůležitějších charakteristik. Počet hvězd na GitHubu projektu může ovlivnit dolování dat open-source software (OSS) projektů uložených na GitHubu tím, že indikuje atraktivitu a důvěryhodnost

projektu. Projekty s vyšším počtem hvězd mohou být považovány za populárnější a atraktivnější pro dolování dat, protože větší zájem ze strany komunity může naznačovat vyšší kvalitu kódu nebo širší uplatnění. Navíc, počet hvězd může být použit jako metrika při výběru projektů pro analýzu, jelikož vyšší počet hvězd často značí lepší podporu ze strany komunity a pravděpodobnější pokračování vývoje. Tímto způsobem mohou být projekty s vyšším počtem hvězd považovány za důležitější a relevantnější pro dolování dat.

Pokud je dolování dat zaměřeno na identifikaci trendů a vzorců v open-source software (OSS) projektech, projekty s vyšším počtem hvězd mohou sloužit jako reprezentativnější příklady úspěšných projektů, které se vyznačují svými jedinečnými vlastnostmi a přístupy. Analýza těchto úspěšných projektů může pomoci identifikovat klíčové faktory, které přispívají k jejich úspěchu, jako jsou efektivní vývojové postupy, inovace, kvalitní dokumentace nebo aktivní komunita přispěvatelů. Díky zjištění těchto faktorů mohou být analýzy dolování dat použity k identifikaci trendů a osvědčených postupů, které by mohly být aplikovány na jiné projekty, aby se zvýšila jejich šance na úspěch. V důsledku toho by mohla být pozornost věnována projektům, které se potýkají s různými problémy nebo se nacházejí v různých fázích vývoje, a poskytnout jim cenné informace o tom, jak zlepšit své postupy a dosáhnout lepších výsledků.

Je důležité zdůraznit, že analýza projektů s vyšším počtem hvězd může poskytnout užitečné informace o úspěšných projektech, ale neměla by zastínit analýzu menších projektů či těch s nižším počtem hvězd. Tyto projekty mohou také poskytovat cenné informace a ukázat alternativní přístupy k řešení problémů. Proto je vhodné zahrnout projekty různých velikostí a stupňů úspěchu do analýz dolování dat, aby byl zajištěn co největší přehled o různorodosti OSS projektů.

Velký počet hvězd mají zejména velké a známé projekty vyvíjené buď velkou společností a nebo s velkou komunitou. Mezi takové projekty patří například *Tensorflow*[2], *React*[3] *a FreeCodeCamp*[4]. Tyto tři projekty mají k roku 2023 výrazně přes sto padesát tisíc hvězd.

[2]https://github.com/tensorflow/tensorflow
[3]https://github.com/facebook/react
[4]https://github.com/freeCodeCamp/freeCodeCamp

4.6 Forks a branches

Rozdělení (forking) projektu je běžnou praxí při vývoji softwaru. Když vývojář rozdělí projekt, vytvoří kopii kódové základny a začne provádět změny na své vlastní verzi. To může být užitečné z různých důvodů, jako je provádění přizpůsobení, přispění k původnímu projektu nebo použití jako výchozí bod pro nový projekt.

Forking zahrnuje vytvoření repliky existujícího úložiště, ve kterém lze provádět úpravy, aniž by to ovlivnilo jeho zdroj. Jednotlivci se obvykle rozhodnou pro rozvětvení repozitářů, aby přispěli změnami k projektu jiného nebo aby využili takové projekty jako základ svých vlastních výtvorů.

Naproti tomu větev (branch) odkazuje na nezávislou úpravu zdrojového kódu v úložišti. To umožňuje upravovat různé funkce nebo experimentovat s různými řešeními bez zásahu do primární verze. Používání větvení je běžné při zavádění nových atributů, aplikaci oprav problémů a testování neprozkoumaných cest při zachování funkčnosti v základní kódové základně.

Jednou z výhod rozvětvení projektu je, že umožňuje vývojářům experimentovat se změnami, aniž by to ovlivnilo původní kódovou základnu. Mohou vytvořit novou větev na své rozvětvené verzi a provádět změny, jak uznají za vhodné. To jim umožňuje testovat nové funkce, opravovat chyby nebo přidávat funkce bez obav z porušení původního projektu.

Forking projektu také umožňuje vývojářům přispívat do původního projektu kontrolovaným způsobem. Mohou provádět změny ve své vlastní rozvětvené verzi a poté vytvořit požadavek na stažení pro sloučení těchto změn zpět do původní kódové základny. To umožňuje původnímu správci projektu zkontrolovat změny a rozhodnout, zda je sloučit či nikoli.

Kromě toho může být rozvětvení užitečné pro vytváření nových projektů, které jsou založeny na existující kódové základně. Vývojář může vytvořit větev projektu a poté ji upravit tak, aby vyhovovala jejich potřebám, a vytvořit tak nový projekt, který je podobný, ale odlišný od původního. To může ušetřit značné množství času a úsilí ve srovnání s tím, když se začíná bez předlohy.

Forking může být užitečný pro spolupráci s dalšími vývojáři. Více vývojářů může rozdělit projekt a pracovat na svých vlastních verzích, sdílet kód a změny podle potřeby. To může usnadnit spolupráci a umožnit více lidem pracovat na stejném projektu současně.

Zde je několik postřehů o tom, co může počet forků naznačovat o projektu na GitHubu.

4.6.1 Popularita a povědomí

Počet forků může ukázat, kolik vývojářů má o projekt zájem. Tato data mohou pomoci změřit popularitu projektu v komunitě vývojářů. Vysoký počet forků naznačuje, že projekt přitahuje významnou pozornost a že vývojáři vidí v projektu hodnotu.

4.6.2 Kvalita kódu

Počet větví může být ukazatelem kvality kódu. Když je projekt rozvětvený, často se používá jako základ pro vytvoření nového projektu. Pokud je kód projektu špatně napsaný, je méně pravděpodobné, že bude rozvětvený. Na druhou stranu, pokud je kód projektu dobře strukturovaný a snadno srozumitelný, je pravděpodobnější, že bude rozvětvený.

4.6.3 Spolupráce a přínos

Počet rozvětvení může také indikovat úroveň spolupráce a příspěvku k projektu. Vyšší počet větví znamená, že do projektu přispívá více lidí. To může vést k lepší kvalitě kódu, novým funkcím a vylepšeným funkcím.

4.6.4 Údržba projektu

Na údržbě projektu se může podepsat i počet rozvětvení. Pokud projekt nebyl aktualizován nebo udržován, je nepravděpodobné, že bude rozvětven. Vysoký počet forků může naznačovat, že projekt je aktivně udržován a aktualizován, což poskytuje novým vývojářům důvěru, že projekt je podporován.

4.6.5 Podpora komunity

Počet forků může také naznačovat míru podpory projektu komunitou. Vysoký počet forků ukazuje, že projekt má vyhrazenou uživatelskou základnu, která jej podporuje. Tato podpora může mít mnoho forem, jako jsou hlášení chyb, požadavky na funkce a příspěvky do kódu.

Využití počtu forků a branchů pro dolování dat

Počet forků a branchů OSS projektu na může ovlivnit následné dolování dat tím, že poskytuje informace o aktivitě a zapojení komunity do projektu. Projekty s vysokým počtem forks a branches mohou naznačovat, že projekt je aktivně vyvíjen a má silnou komunitu přispěvatelů. Při provádění dolování

dat na OSS projektech na GitHubu je důležité zohlednit počet forků a branchů, aby bylo možné získat ucelený pohled na zdraví, dynamiku a potenciál projektu.

4.7 Issues

Jednou ze základních funkcí GitHubu je nástroj pro sledování úkolů (Issues), který umožňuje vývojářům sledovat a spravovat chyby, požadavky na funkce a další úkoly související s jejich projektem. Issues jsou zásadní součástí projektu, protože pomáhají udržovat kvalitu a zajišťují, že jsou splněny potřeby uživatelů.

Jedná se o jednu ze základních charakteristik OSS projektů na GitHubu. Projekty se pomocí této charakteristiky dají třídit podle toho, jestli vůbec issues nemají a nebo zda je mají a popřípadě jaká a kolik. Tento fakt může o celém repozitáři mnohé vypovídat (více v sekci 4.8).

Issues na GitHubu umožňují sledovat vývoj práce. Pokud se zmíníte o problému v jiném problému nebo žádosti o stažení, časová osa problému ukáže křížový odkaz, což umožní sledovat související práci. Pro označení probíhající práce můžete propojit problém s požadavkem na stažení. [11]

Issues při vývoji OSS projektu (a ne jen u projektů s otevřeným kódem) na GitHubu může být několik, mezi ty nejčastější patří:

4.7.1 Hlášení o chybách

Hlášení chyb (bug report) je jedna z nejběžnějších forem problémů v projektu na GitHubu. Bug je chyba v softwaru, která způsobuje, že se chová neočekávaným způsobem. Uživatelé nebo vývojáři mohou hlásit chyby v projektu a správci projektu je mohou sledovat a opravovat. Hlášení chyb je nezbytné pro zajištění vysoké kvality softwaru a splnění požadavků uživatelů.

4.7.2 Požadavky na funkce

Požadavky na funkce (feature request) jsou dalším běžným typem problémů v projektu na GitHubu. Uživatelé nebo vývojáři mohou mít nápady na nové funkce nebo vylepšení softwaru a mohou si tyto funkce vyžádat v nástroji pro sledování problémů. Požadavky na funkce jsou důležité pro vylepšení softwaru a zvýšení jeho užitečnosti pro uživatele.

4.7.3 Problémy s dokumentací

Dokumentace je nezbytnou součástí každého softwarového projektu. Často je však opomíjena nebo jí není věnována dostatečná pozornost. Problémy s dokumentací v projektu na GitHubu mohou zahrnovat chybějící nebo neúplnou dokumentaci, zastaralé informace nebo matoucí pokyny. Dobrá dokumentace je důležitá pro zpřístupnění softwaru uživatelům a vývojářům. Většina velkých otevřených projektů by tedy měla mít kvalitní dokumentaci.

4.7.4 Problémy s kvalitou kódu

Problémy s kvalitou kódu mohou nastat v projektu z různých důvodů, například špatné postupy kódování, nedostatek testování a špatná rozhodnutí o návrhu. Tyto problémy mohou ztížit údržbu softwaru a mohou vést k chybám nebo jiným problémům.

4.7.5 Problémy s údržbou

Údržba projektu GitHub může být náročný úkol, zejména pro open-source projekty s mnoha přispěvateli. Správci musí zajistit, aby byl projekt aktuální, bez chyb a kompatibilní s jiným softwarem. Problémy s údržbou mohou nastat kvůli nedostatku zdrojů nebo odborných znalostí, takže je náročné udržovat projekt aktuální a bezpečný.

4.7.6 Komunitní záležitosti

Projekty GitHub jsou často řízeny komunitou a v projektu mohou nastat problémy s komunitou. Tyto problémy mohou zahrnovat toxické chování, neproduktivní diskuse nebo konflikty mezi přispěvateli. Problémy komunity mohou poškodit reputaci projektu, odradit od příspěvků a způsobit, že bude méně vstřícný k novým uživatelům.

4.8 Počet Issues

Počet issues OSS projektu na GitHubu může ovlivnit následné dolování dat tím, že poskytuje informace o problémech a výzvách, kterým projekt čelí. Issues mohou obsahovat diskuse a návrhy na řešení problémů, což může poskytnout cenné informace pro dolování dat. Počet úkolů v projektu může hodně říci o kvalitě, udržovatelnosti a popularitě projektu. Problémy se týkají úkolů, chyb nebo návrhů hlášených uživateli nebo vývojáři projektu.

Počet úkolů může naznačovat, jak aktivně se na projektu pracuje, jak reagují vývojáři na zpětnou vazbu od uživatelů a úroveň pozornosti k detailům v projektu.

Jedním ze zásadních aspektů projektu nejen na GitHubu je jeho kvalita, která se často odráží v množství problémů, které má. Vyšší počet úkolů nemusí vždy znamenat, že projekt je nekvalitní. Ve skutečnosti to může znamenat opak. Projekt s mnoha úkoly často znamená, že vývojáři na projektu aktivně pracují a reagují na zpětnou vazbu od uživatelů. Je tomu tak proto, že když se více dívá na projekt, šance na identifikaci úkolů a jejich nahlášení je vyšší.

Na druhou stranu projekt s nízkým počtem úkolů může znamenat, že projekt není příliš populární nebo že není mnoho uživatelů, kteří by mohli hlásit úkoly. Může to však také znamenat, že vývojáři vynaložili značné úsilí na to, aby byl projekt vysoce kvalitní a bez chyb. Stojí za zmínku, že počet úkolů není jediným měřítkem kvality projektu. Na kvalitu projektu mají vliv i další faktory, jako je složitost projektu a počet přispěvatelů.

Počet úkolů může také naznačovat udržitelnost projektu. Projekt s mnoha nevyřešenými úkoly může naznačovat, že projekt je špatně udržovaný nebo opuštěný. To by mohlo být varovným signálem pro uživatele, kteří na projekt spoléhají, a může to znamenat, že projekt není vhodný pro dlouhodobé používání.

V neposlední řadě může počet problémů vypovídat o popularitě repozitáře. Projekt s mnoha úkoly a aktivními přispěvateli může naznačovat, že je široce využíván a oblíbený. To se jeví být dobrým ukazatelem potenciálu projektu pro růst a rozvoj.

Issues se dělí na otevřené (open) a uzavřené (closed).

Otevřené úkoly v úložišti Github jsou úkoly, které momentálně nejsou vyřešeny. To znamená, že problém byl nahlášen, ale řešení ještě nebylo implementováno. Otevřené úkoly jsou obvykle přiřazeny vývojáři nebo členovi týmu, který je odpovědný za vyřešení úkolu. Problémy mohou zůstat otevřené z různých důvodů, jako je čekání na další informace od reportéra, čekání na schválení od projektového manažera nebo jednoduše proto, že úkol je složitý a vyžaduje více času na vyřešení.

Uzavřené úkoly v úložišti Github jsou úkoly, které již byly vyřešeny. To znamená, že řešení bylo implementováno a problém již neovlivňuje projekt. Když je úkol uzavřen, je důležité poskytnout podrobné vysvětlení, jak byl úkol vyřešen, aby ostatní členové týmu nebo veřejnost, pokud je tedy repozitář veřejný, mohli řešení pochopit a poučit se z něj.

Rozdíly mezi otevřenými a uzavřenými issues

Jedním z klíčových rozdílů mezi otevřenými a uzavřenými issues je úroveň aktivity. Otevřená issues jsou obvykle aktivnější, s průběžnými diskusemi a aktualizacemi od členů projektu, kteří na problému pracují. Uzavřené problémy jsou na druhé straně méně aktivní a obvykle se aktualizují pouze tehdy, když je hlášen nový problém, který souvisí s uzavřeným problémem.

Dalším rozdílem mezi otevřenými a uzavřenými úkoly je úroveň priority. Otevřené úkoly mají obvykle vysokou prioritu, protože představují chyby nebo funkce, které ovlivňují projekt. Na druhé straně uzavřené záležitosti mají nižší prioritu, protože již projekt neovlivňují. Uzavřené úkoly však mohou projektu stále poskytovat hodnotu tím, že slouží jako reference pro budoucí úkoly nebo poskytují náhled do historie projektu.

Význam počtu issues

Vysoký počet otevřených úkolů v úložišti Github může být známkou zdravého projektu. To může znamenat, že projekt má velkou uživatelskou základnu a je aktivně využíván a testován. Kromě toho může velký počet otevřených problémů naznačovat, že projekt je aktivně udržován a že vývojáři řeší úkoly a provádějí změny v kódové základně v reakci na zpětnou vazbu od uživatelů. Na druhou stranu, velké množství otevřených problémů může také naznačovat, že projekt se snaží udržet krok s poptávkou po nových funkcích nebo opravách chyb. V tomto případě to může být známkou toho, že projekt potřebuje další zdroje nebo že současný tým potřebuje lépe upřednostňovat problémy, aby efektivně řídil pracovní zátěž.

Stejně tak malý počet otevřených úkolů může být známkou vyspělého a stabilního projektu. Může to znamenat, že se již vyřešilo mnoho chyb a problémů, které byly hlášeny během jeho vývojové fáze, a že je projekt nyní ve stabilním stavu s několika zbývajícími úkoly, které je třeba vyřešit. Malý počet otevřených issues však také může naznačovat, že se projektu nedostává příliš pozornosti nebo zpětné vazby od uživatelů. To může být problematické, protože chyby a úkoly mohou nastat i poté, co projekt dosáhl stabilního stavu. Nedostatek otevřených issues může naznačovat, že uživatelé projekt aktivně nepoužívají nebo že nehlásí problémy, když se s nimi setkají.

4.9 Pull requests

Pull requesty (požadavky na stažení) jsou klíčovou funkcí Githubu, která umožňuje vývojářům přispívat do projektu a spolupracovat s dalšími při-

spěvateli. Jsou výkonným nástrojem pro správu změn v projektu, kontrolu a diskusi o navrhovaných změnách a integraci těchto změn do hlavní kódové základny projektu.

Požadavek na stažení je požadavek přispěvatele na sloučení změn provedených ve větvi projektu s hlavní kódovou základnou. To umožňuje ostatním přispěvatelům a správcům projektů přezkoumat navrhované změny, poskytnout zpětnou vazbu a nakonec se rozhodnout, zda změny přijmout nebo odmítnout. Jedná se o důležitý krok v kolaborativním vývoji projektu a umožňuje uživatelům přispět svými odbornými znalostmi a zkušenostmi ke zlepšení projektu. Počet žádostí o stažení představuje celkový počet žádostí o sloučení, které byly odeslány přispěvateli do projektu.

Požadavky na stažení lze použít pro různé účely, jako je oprava chyb, přidávání nových funkcí nebo zlepšování dokumentace. Jsou základním nástrojem pro správu příspěvků do projektu, protože poskytují strukturovaný způsob, jak mohou přispěvatelé navrhovat změny a správci projektu mohou tyto změny kontrolovat.

Kromě toho, že poskytují strukturovaný způsob správy příspěvků, nabízí žádosti o stažení také spolupráci a diskusi mezi přispěvateli. Umožňují přispěvatelům diskutovat o navrhovaných změnách, navrhovat alternativní řešení a identifikovat potenciální problémy nebo konflikty. To může pomoci zlepšit kvalitu kódu a dokumentace a zajistit, aby projekt vyhovoval potřebám svých uživatelů.

Další důležitou výhodou pull requestů je to, že zprostředkovávají kontinuální integraci a kontinuální doručování (CI/CD). Pomocí nástrojů pro automatizované testování a nasazení lze automaticky vytvářet a testovat požadavky na stahování a nasazovat je do pracovního prostředí pro další testování. To umožňuje přispěvatelům a správcům projektu zajistit, aby navrhované změny fungovaly tak, jak bylo zamýšleno, a nezaváděly nové chyby nebo problémy.

Počet žádostí o stažení je užitečná metrika pro vyhodnocení aktivity a úrovně zapojení projektu. Žádosti o stažení představují primární způsob, jakým mohou uživatelé přispívat do projektu, a vysoký počet žádostí o stažení může znamenat, že projekt je aktivní a má živou komunitu.

Vysoký počet žádostí o stažení může znamenat, že projekt je aktivní a má živou komunitu. To může být dobré znamení pro uživatele, kteří mají zájem přispět do projektu, protože to naznačuje, že existuje zdravá komunita vývojářů, kteří na projektu aktivně pracují. Kromě toho může vysoký počet žádostí o stažení naznačovat, že projekt je oblíbený a má velkou uživatelskou základnu, což může být důležitým faktorem při hodnocení kvality a užitečnosti projektu.

Na druhou stranu nízký počet žádostí o stažení nemusí nutně znamenat, že je projekt neaktivní nebo nekvalitní. Některé projekty mohou mít malou uživatelskou základnu nebo mohou být zaměřeny na konkrétní část, což může omezit počet potenciálních přispěvatelů. Některé projekty mohou mít navíc vysokou úroveň složitosti nebo mohou vyžadovat specializované znalosti, což může omezit počet přispěvatelů, kteří jsou schopni smysluplně přispět.

Samotný počet pull requestů však nestačí k hodnocení kvality nebo užitečnosti projektu Github. Je důležité zvážit kvalitu požadavků na stažení, jako je relevance změn, úroveň dokumentace a dodržování standardů kódování.

4.10 Licence

Licence je právní smlouva, která upravuje používání softwaru nebo jiných kreativních děl. Specifikuje podmínky, za kterých lze software nebo dílo používat, upravovat a distribuovat. V kontextu GitHubu je licence sada podmínek, které definují, jak může uživatel používat kód v úložišti.

Filtrováním podle druhu licence můžeme jednoduše roztřídit velké množství OSS projektů do menších podmnožin. Jedná se tedy o celkem důležitou charakteristiku, kterou je vhodné v následném zpracování dat projektů brát v potaz.

4.10.1 Důvody licencování na GitHubu

Licencování je důležitým aspektem správy úložiště na GitHubu. Zde je několik zásadních důvodů, proč je licencování důležité:

1. **Ochrana kódu**
   Licence pomáhá chránit vlastníkův kód a zabránit neoprávněnému použití nebo distribuci. Bez licence mohou ostatní používat nebo distribuovat cizí kód bez svolení autora, což by mohlo poškodit vývoj projektu nebo vést k právním sporům.

2. **Podpora spolupráce**
   Jasná licence může pomoci podpořit spolupráci a usnadnit ostatním přispívat do projektu. Poskytnutím jasných podmínek, jak lze kód používat, upravovat a distribuovat, lze zajistit, aby potenciální přispěvatelé rozuměli pravidlům a byli si jisti svou schopností přispívat.

3. **Nastavení očekávání**
   Licence může pomoci nastavit očekávání, jak kód používat a jaké pří-

spěvky jsou vítány. Jasné uvedení podmínek, za kterých lze kód používat, může pomoci zajistit, aby potenciální uživatelé a přispěvatelé chápali, co se od nich očekává a co s kódem mohou a nemohou dělat.

4.10.2 Typy licencí na GitHubu

Na GitHubu je k dispozici mnoho různých typů licencí, od tolerantních licencí, které umožňují široké použití a úpravy, až po restriktivní licence, které kladou přísná omezení na to, jak lze kód použít. Mezi nejběžnější typy licencí patří:

1. **Licence MIT**
   Licence MIT je otevřená licence, která dává uživatelům široká práva k použití, modifikaci, kopírování, distribuci a sublicencování díla, stejně jako právo na prodej odvozených děl. Je jednou z nejpopulárnějších softwarových licencí používaných ve vývoji softwaru.

2. **Licence BSD**
   Licence BSD (Berkeley Software Distribution) je považována za velmi liberální a umožňuje širokou škálu použití a modifikací softwaru. Existuje několik verzí BSD licence. BSD licence umožňuje bezplatné šíření zdrojového kódu, které může být sdíleno a použito jakýmkoli způsobem. Uživatelé mohou upravovat zdrojový kód a vytvářet odvozená díla, která mohou být distribuována pod stejnými podmínkami jako původní software. BSD licence vyžaduje, aby uživatelé udržovali uvedení původních autorů a copyrightu ve zdrojovém kódu a v dokumentaci. Autoři nejsou odpovědní za žádné škody vzniklé v důsledku použití softwaru.

3. **GNU General Public License (GPL)**
   GPL je omezující licence, která vyžaduje, aby jakýkoli kód odvozený z původního kódu byl také licencován pod GPL. To zajišťuje, že veškeré změny provedené v kódu zůstanou open-source.

4. **Licence Apache**
   Licence Apache je oblíbená open-source licence, která se často používá v softwarovém vývoji. Umožňuje použití, úpravy a distribuci kódu, pokud je zachováno původní upozornění na autorská práva a jsou jasně označeny jakékoli úpravy.

Typ licence ovlivňuje atraktivitu projektu pro potenciální přispěvatele. Některé licence, jako jsou permisivní licence (např. MIT, BSD), umožňují širší možnosti použití a modifikace, což může přitahovat více přispěvatelů a zvyšovat šance na úspěch projektu. Na druhou stranu, projekty s více restriktivními licencemi, jako je například GNU General Public License (GPL), mohou omezovat komerční využití a mít menší počet přispěvatelů. Licence také mohou signalizovat úroveň profesionalismu a závazku projektu vůči komunitě. Projekty s jasně definovanými licenčními podmínkami ukazují, že se jejich tvůrci zaměřují na dlouhodobou udržitelnost a transparentnost. To může naznačovat, že projekt má silnější základnu a je pravděpodobněji životaschopný v dlouhodobém horizontu.

4.10.3 Význam licencí při dolování dat

V kontextu OSS projektů uložených na GitHubu je důležité respektovat licence jednotlivých projektů, které ovlivňují, jak mohou být data použita, sdílena a upravována. Několik způsobů, jak licence ovlivňují dolování dat OSS projektů:

Přístup k datům

Licence určuje, zda může být kód a související data použita pro dolování dat. Některé licence umožňují volný přístup a použití dat, zatímco jiné mohou mít omezení.

Redistribuce

Pokud chceme výsledky dolování dat sdílet s ostatními, licence určuje, zda je to povoleno a za jakých podmínek. Například některé licence mohou vyžadovat, abychom uvedli původní autory a licence.

Modifikace

Pokud chceme upravit nebo rozšířit kód OSS projektu jako součást procesu dolování dat, licence určuje, zda je to povoleno a za jakých podmínek.

4.11 Přispěvatelé

Přispěvatelé (Contributors) – jednotlivci, kteří přispívají k projektu odesláním kódu, dokumentace, hlášení o chybách a dalších materiálů.

Role přispěvatelů v úložišti GitHub je důležitá pro vývoj projektu. Hrají zásadní roli při vytváření, údržbě a vylepšování kódové základny, zajišťují, že projekt splňuje potřeby svých uživatelů, a řeší chyby a problémy, které se časem objeví.

Jedním ze způsobů, jak analyzovat přispěvatele do úložiště na GitHubu, je podívat se na úroveň jejich aktivity. To lze měřit počtem příspěvků každého přispěvatele do projektu. Přispěvatelé, kteří přispěli velkým počtem příspěvků, mohou být považováni za aktivnější nebo zapojenější do projektu než ti, kteří přispěli méně.

Dalším způsobem, jak analyzovat přispěvatele, je podívat se na oblasti jejich odborných znalostí. To lze měřit analýzou příspěvků každého přispěvatele do kódu a identifikací konkrétních oblastí kódové základny, do které přispěli. Přispěvatelé, kteří přispěli do mnoha různých částí kódové základny, mohou být považováni za všestrannější nebo znalejší než ti, kteří se zaměřili na jedinou oblast.

Třetím způsobem, jak analyzovat přispěvatele, je podívat se na jejich komunikační dovednosti a schopnosti spolupráce. To lze měřit analýzou rozsahu, v jakém se přispěvatelé zapojují s ostatními členy projektového týmu, prostřednictvím diskusí, žádostí a dalších prostředků komunikace. Přispěvatelé, kteří jsou aktivní v těchto oblastech, mohou být považováni za více spolupracující nebo efektivnější členy týmu než ti, kteří jsou méně angažovaní.

Zde je několik aspektů, které je třeba zvážit při analýze přispěvatelů:

- **Počet přispěvatelů:** Počet přispěvatelů do úložiště může být ukazatelem popularity a úrovně zapojení komunity do projektu. Více přispěvatelů obecně znamená aktivnější a široce využívaný projekt.
- **Četnost příspěvků:** Četnost příspěvků může naznačovat, jak aktivní je vývoj projektu. Projekty, které dostávají pravidelné příspěvky, jsou obecně stabilnější a aktuálnější.
- **Rozmanitost přispěvatelů:** Rozmanitost přispěvatelů může naznačit, jak inkluzivní a přístupný projekt je. Různorodá základna přispěvatelů může do projektu přinést různé pohledy, nápady a dovednosti.
- **Pověst přispěvatelů:** Pověst přispěvatelů může naznačovat kvalitu kódu a úroveň odbornosti zapojené do projektu. Pokud má úložiště mnoho přispěvatelů s dobrou pověstí a historií vysoce kvalitních příspěvků, je to dobré znamení, že je projekt dobře udržovaný.

- **Rozdělení příspěvků**: Rozdělení příspěvků mezi přispěvatele může naznačit, do jaké míry je proces vývoje založen na spolupráci. Pokud je za většinu příspěvků odpovědných několik málo přispěvatelů, může to naznačovat méně spolupracující prostředí.

Nejlépe měřitelným aspektem je počet přispěvatelů. Tuto hodnotu samotný GitHub i GitHub API nabízejí, avšak každý z nich vede informace o jiných přispěvatelích. API obsahuje pouze přispěvatele, kteří něčím do projektu přispěli a nejsou pouze vedeni jako "přispěvatel". Celkově může počet přispěvatelů do úložiště na GitHubu poskytnout cenné informace o vývoji a údržbě projektu. Jedná se tak o důležitou charakteristiku.

4.12 Wiki

Wiki je sbírka stránek, které mohou uživatelé společně upravovat, a může poskytnout cenné informace o kódové základně úložiště, historii projektů a postupech vývoje.

Přítomnost wiki v úložišti GitHub může poskytnout několik náznaků o vývoji projektu. Za prvé, wiki může naznačovat, že se vývojáři úložiště zavázali dokumentovat projekt a poskytovat uživatelům užitečné informace. Poskytnutím podrobné dokumentace a pokynů mohou vývojáři usnadnit ostatním pochopení projektu a přispět k jeho vývoji.

Za druhé, wiki může indikovat vyspělost projektu. Pokud má úložiště dobře vyvinutou wiki, naznačuje to, že projekt byl nějakou dobu ve vývoji a vytvořil komunitu přispěvatelů. U vyspělého projektu se silnou komunitou je pravděpodobnější, že bude mít stabilní kódovou základnu, aktivní vývoj a vyšší úroveň uživatelské podpory.

Za třetí, wiki může poskytnout pohled na postupy vývoje přispěvatelů projektu. Wiki může například obsahovat informace o standardech kódování projektu, testovacích procedurách a postupech správy verzí. Tyto informace mohou být užitečné pro nové přispěvatele, kteří se učí, jak pracovat s kódovou základnou projektu.

Za čtvrté, wiki může poskytnout prostor pro spolupráci a diskusi mezi přispěvateli projektu. Tím, že umožňuje přispěvatelům sdílet nápady a diskutovat o problémech rozvoje, může wiki pomoci podpořit pocit komunity a týmové práce v rámci projektu.

Celkově může přítomnost wiki v úložišti GitHub poskytnout cenné informace o vývoji, vyspělosti a postupech spolupráce projektu. Tím, že wiki poskytuje prostor pro dokumentaci, diskusi a spolupráci, může pomoci podpořit růst a úspěch OSS projektu.

4.13 Jazyk

Jazyk používaný v úložišti na GitHub odkazuje na programovací jazyk používaný k zápisu kódu v něm. Analýza jazyka použitého v úložišti může poskytnout důležité poznatky o použité technologii, úrovni odborných znalostí potřebných k přispění a potenciálních aplikacích projektu.

Jedním z aspektů analýzy jazyka používaného v úložišti je posouzení jeho popularity. Různé programovací jazyky se liší v popularitě, přičemž některé jsou více používané než jiné. Repozitáře, které používají populárnější programovací jazyky, mohou být dostupnější pro širší okruh vývojářů, kteří s větší pravděpodobností znají používaný jazyk.

Při analýze jazyka použitého v projektu je dalším faktorem, který je třeba vzít v úvahu, ekosystém nástrojů a knihoven dostupných pro daný jazyk. Populární programovací jazyky mají často prosperující ekosystém nástrojů a knihoven, které lze použít ke zlepšení efektivity vývoje a údržby kódu. Repozitáře, které používají jazyk s velkým ekosystémem, mohou těžit ze zvýšené efektivity vývoje a kvality kódu.

Při analýze použitého jazyka je důležité zvážit úroveň odborných znalostí potřebnou pro příspěvek do úložiště. Některé programovací jazyky jsou složitější než jiné, což vyžaduje vyšší úroveň odborných znalostí, aby s nimi mohla efektivně pracovat. V důsledku toho může být přispívání do repozitářů, které používají složité jazyky, obtížnější, protože vyžadují vyšší úroveň odbornosti.

Jazyk projektu na webhostingových službách (nejen na GitHubu) může ovlivnit dolování dat open-source software projektů z několika důvodů:

1. **Filtr:** Při provádění dolování dat mohou být OSS projekty filtrovány podle jazyka, aby byly získány relevantní data. Pokud se zaměřujete na konkrétní programovací jazyk, může to ovlivnit výběr projektů pro analýzu.

2. **Popularita jazyka:** Některé programovací jazyky jsou populárnější než jiné, což může vést k většímu množství dostupných OSS projektů v těchto jazycích. Tento fakt může mít vliv na zastoupení daných jazyků ve výsledcích dolování dat a na možnost nalezení zajímavých projektů.

3. **Specifika jazyka:** Každý programovací jazyk má své vlastní vlastnosti, funkce a ekosystém knihoven, které mohou ovlivnit, jak se OSS projekty v daném jazyce vyvíjejí a jakou strukturu mají. To může mít dopad na analýzu dat z těchto projektů, například při identifikaci vzorů nebo trendů.

4. **Komunita:** Každý programovací jazyk má svou vlastní komunitu vývojářů, která může ovlivnit kulturu a způsoby spolupráce na OSS projektech. To může ovlivnit dolování dat těchto projektů, například při zkoumání interakce mezi účastníky projektu, zpětné vazby nebo diskuzí.

5. **Nástroje pro analýzu:** Některé nástroje pro dolování dat a analýzu mohou být lépe přizpůsobeny určitým programovacím jazykům. To může ovlivnit schopnost zpracovat a analyzovat data z OSS projektů v daném jazyce.

4.14 Releases

Vydání (Release) v úložišti GitHub se týká konkrétní verze softwaru, která byla považována za připravenou k nasazení do produkčního prostředí. Vydání může poskytnout cenné informace o vyspělosti softwaru, frekvenci aktualizací a úrovni pozornosti, kterou vývojový tým věnuje úložišti.

Jedním ze způsobů analýzy vydání v úložišti GitHub, je podívat se na frekvenci vydání. Častá vydání mohou naznačovat aktivní a agilní vývojový proces, kdy vývojáři pracují na neustálém vylepšování kódu a včasném řešení chyb a problémů. Méně časté verze však mohou naznačovat stabilnější projekt nebo pomalejší vývojový proces.

Dalším způsobem, jak analyzovat vydání, je podívat se na kvalitu změn kódu provedených mezi vydáními. Vysoce kvalitní změny kódu mohou zahrnovat nové funkce, vylepšený výkon a opravy chyb. Změny kódu nižší kvality mohou přinést nové chyby nebo narušit stávající funkce. Analýza kvality změn kódu může vývojářům pomoci identifikovat oblasti kódové základny, které vyžadují více pozornosti, a stanovit priority oblastí pro zlepšení.

Třetím způsobem je podívat se na zapojení uživatelů do jednotlivých vydání. To lze měřit analýzou počtu stažení nebo instalací každého vydání, počtu hlášení o chybách nebo odeslaných požadavků na funkce a celkové zpětné vazby od uživatelů. Vysoké zapojení uživatelů může znamenat oblíbený a užitečný projekt, zatímco nízké zapojení uživatelů může znamenat projekt, který nesplňuje potřeby svých uživatelů.

Celkově může analýza vydání v úložišti poskytnout cenné informace o procesu vývoje, kvalitě kódu a zapojení uživatelů. Analýzou frekvence, kvality a zapojení uživatelů do verzí mohou vývojáři zlepšit proces vývoje, vytvářet vysoce kvalitní kód a vytvářet úspěšné softwarové projekty, které splňují potřeby uživatelů i vývojářů.

Počet vydání v úložišti poskytuje důležité informace o vyspělosti a úrovni aktivity softwarového projektu.

Úložiště, které má vysoký počet vydání během významného časového období, naznačuje, že software je vyspělý, stabilní a aktivně udržovaný. To znamená, že se vývojový tým zavázal pravidelně aktualizovat software a řešit problémy, které mohou nastat. Kromě toho může vysoký počet verzí naznačovat, že software používá velká uživatelská základna, protože aktualizace jsou nezbytné pro splnění vyvíjejících se potřeb uživatelů.

Naopak úložiště s malým počtem nebo žádnými verzemi může naznačovat, že software je stále ve vývoji nebo že není aktivně udržován. To může být varovným signálem pro potenciální uživatele, protože občasné aktualizace mohou znamenat, že software je nestabilní nebo nespolehlivý. Nedostatek vydání může navíc naznačovat, že vývojový tým je méně angažovaný nebo méně oddaný projektu.

Počet vydání tedy jako jediný nemá sám o sobě velkou vypovídající hodnotu, avšak je velmi vhodné na tuto charakteristiku brát zřetel a popřípadě ji u některých projektů využít k analýze projektu. Jedná se tedy o důležitou charakteristiku, nicméně je třeba ji doplnit o další.

Vliv Releases na GitHubu na dolování dat OSS projektů

1. **Sledování vývoje projektu**
   Releases umožňují nástrojům dolování dat sledovat vývoj a pokrok projektu v čase. Tímto způsobem mohou analytici získat přehled o vývoji projektu, jeho stabilitě a rychlosti vydávání nových verzí.

2. **Identifikace důležitých změn**
   Díky informacím o Releases mohou nástroje dolování dat identifikovat důležité změny, které byly provedeny mezi verzemi. To zahrnuje nové funkce, opravy chyb nebo aktualizace bezpečnosti, což může být užitečné pro analýzu kvality kódu a udržitelnosti projektu.

3. **Srovnání projektů**
   Releases mohou sloužit jako srovnávací metrika mezi různými OSS projekty uloženými na GitHubu. Analytici mohou porovnat frekvenci a obsah Releases, aby získali představu o tom, který projekt je aktivnější, stabilnější a lépe udržovaný.

4. **Identifikace závislostí**
   Nástroje na dolování dat mohou prostřednictvím Releases identifikovat závislosti mezi různými projekty. Například, pokud je projekt A závislý

na projektu B, změny v Releases projektu B mohou ovlivnit funkčnost a bezpečnost projektu A.

5. **Nástroje pro analýzu**
   Některé nástroje pro dolování dat a analýzu mohou být lépe přizpůsobeny určitým programovacím jazykům. To může ovlivnit schopnost zpracovat a analyzovat data z OSS projektů v daném jazyce.

Za účelem efektivního dolování dat OSS projektů uložených na GitHubu je důležité sledovat a analyzovat Releases, protože poskytují klíčové informace o vývoji, stabilitě a udržitelnosti projektů.

5 Výběr charakteristik a návrh aplikace

Nejdříve bylo nutné vybrat ze zkoumaných nástrojů jeden jako zdroj projektů a pro tento zdroj následně navrhnout aplikaci. K analýze sloužily všechny již zmíněné nástroje, avšak dominoval mezi nimi GitHub, proto při vybírání zdrojů byl hlavním kandidátem právě tento nástroj. Jako zdroj projektů byl tedy vybrán GitHub, jelikož je skvělým zdrojem pro vyhledávání projektů s otevřeným zdrojovým kódem díky své rozsáhlé sbírce úložišť a snadno použitelné vyhledávací funkci. Zde je několik důvodů, proč byl pro hledání projektů s otevřeným zdrojovým kódem vybrán právě tento nástroj:

Velká sbírka úložišť: GitHub má obrovskou sbírku úložišť s otevřeným zdrojovým kódem, která pokrývá širokou škálu programovacích jazyků, rámců a technologií. Podle oficiálního blogu GitHubu [9] bylo jen v roce 2022 vytvořeno přes 52 milionů OSS projektů.

Funkce snadného vyhledávání: GitHub poskytuje výkonnou funkci vyhledávání, která umožňuje filtrovat úložiště podle programovacího jazyka, tématu, hvězdiček, větví a dalších. Díky tomu lze snadno najít ten správný projekt.

Aktivní komunita: GitHub má velkou a aktivní komunitu vývojářů, kteří přispívají k open-source projektům. Podle výše zmíněného zdroje [9] to bylo k roku 2022 až 413 milionů přispěvatelů a to pouze k OSS projektům. To znamená, že s větší pravděpodobností lze na tomto nástroji najít dobře udržované OSS projekty, které jsou aktivně rozvíjeny a podporovány.
Jednou z klíčových výhod GitHubu je jeho výkonné API, které umožňuje vývojářům integrovat funkčnost GitHubu do vlastních aplikací a nástrojů a zároveň umožňuje poměrně snadné vyhledávání projektů. K vyhledávání projektů bylo tedy vybráno API. Nejdříve je tedy nutno toto API zanalyzovat a zjistit jeho funkčnosti a možnosti pro vyhledávání OSS projektů. A následně z již zmíněné analýzy charakteristik vybrat vhodné charakteristiky, které byly vhodné pro selektivní těžbu OSS projektů.

5.1 Analýza API

Pro vypracování byla využita GitHub API [1] verze *2022-11-28*, v době vypracování se jednalo o aktuální verzi. Tato verze byla použita z důvodu snahy o co nejdelší životnost finální aplikace pro vyhledávání projektů.

Pro přístup k GitHub API se musí nejprve ověřit requesty. GitHub podporuje několik mechanismů ověřování, včetně OAuth 2.0, osobních přístupových tokenů a základního ověřování. OAuth 2.0 je nejbezpečnější a nejdoporučovanější možnost, protože umožňuje uživatelům udělovat aplikacím třetích stran přístup ke svému účtu GitHub, aniž by sdíleli své přihlašovací údaje.

Osobní přístupové tokeny jsou také oblíbenou možností, protože poskytují způsob, jak ověřit požadavky API pomocí tokenu, který lze kdykoli odvolat nebo znovu vytvořit. Pro využití v aplikaci se jeví vhodnější použití osobních přístupových tokenů, které si uživatel vygeneruje na svém GitHub účtu a následně jej zadá do aplikace, která jej využije při připojení k API. Avšak vystává zde problém, jelikož **počet požadavků na jednoho ověřeného uživatele je omezen na 5000 za hodinu**[2], pro neověřené uživatele činí limit 60 požadavků za hodinu. Z tohoto nepoměru je jasně patrné, že se musí využít ověřovacího tokenu a k API přistupovat pouze ověřených requestů. Toto musí být v implementaci ošetřeno.

GitHub API implementuje JSON (JavaScript Object Notation) jako prostředek pro prezentaci datových struktur. Jedná se o ideální možnost webového rozhraní API díky své odlehčené funkci a snadnému čtení a zápisu. Výběr podporuje několik různých klasifikací, mezi něž patří úložiště, požadavky na stahování, problémy uživatelů spolu s organizacemi, které zobrazují jedinečné funkce a metody přístupné prostřednictvím volání koncových bodů prováděných na aplikačním programovacím rozhraní (API).

GitHub API nabízí široký výběr přístupových bodů, které umožňují programátorům spravovat a získávat data z platformy. Tyto přístupové body jsou seskupeny do sekcí v závislosti na jejich relevantních informacích, včetně úložišť, problémů, požadavků na stažení nebo uživatelů. Každý koncový bod je svázán s konkrétním HTTP slovesem (GET, POST, PUT nebo DELETE) a lze k němu přistupovat prostřednictvím jeho jedinečné adresy URL. Jeden takový bod umožňuje přístup k podrobnostem o jakémkoli daném úložišti na webu.

Pro přístup k informacím v úložišti na GitHubu slouží volání API *„GET*

[1] https://api.github.com/

[2] Všechny omezení API jsou platné k roku 2023 a je pravděpodobné, že se budou postupem času měnit.

/repos/:owner/:repo“. Tento konkrétní koncový bod umožňuje získat různé podrobnosti o úložišti konkrétního vlastníka. Nahrazením *„:owner“* jménem vlastníka úložiště a *„:repo“* názvem jejich příslušného úložiště získáme přístup k informacím, jako jsou otevřené žádosti o stažení nebo problémy spojené konkrétně s danou instancí. Využití této funkce v kódu umožňuje snadnou integraci mezi repozitáři a dalšími aplikacemi.

Chceme-li získat informace, můžeme zadat požadavek GET na koncový bod přístupný na *https://api.github.com/repos/:owner/:repo*. Parametr „*:owner*“ odkazuje na jednotlivce nebo skupinu, která vlastní úložiště, zatímco „*:repo*“ se vztahuje k jeho názvu.

5.1.1 Vyhledávání projektů

Pro hledání mezi repozitáři slouží klíčové slovo „*search*“, celé volání se tedy provádí na *https://api.github.com/search/*. Po něm následuje dotaz (query). Při správném sestavení URL se lze dostat na koncový bod API, který obsahuje informace o repozitářích. Součástí query může být několik charakteristik, podle kterých lze projekty vyhledávat. Pomocí těchto charakteristik tedy lze jedním requestem vyfiltrovat určité množství projektů. Z důvodu omezeného množství dotazů (5000 za hodinu), kdy by uživatel pro osažení efektivnosti neměl dotazy plýtvat, je tedy lze považovat za "levné dotazy". Charakteristiky, které lze přímo psát do query se tedy jeví jako velmi vhodné pro využití v aplikaci pro hledání projektů. Samotné query má také omezení, jelikož do něj lze umístit pouze 5 charakteristik. Jelikož je záměrem vyhledávat OSS projekty, je zásadní v dotazu nevynechat vlastnost otevřeného projektu (v query se jedná o frázi *„is:public“*). Tímto použitím pro tuto aplikaci povinného argumentu, který bude použit při každém dotazu, zbývá prostor pro čtyři další charakteristiky.

5.1.2 Sestavení dotazovacích URL

Pro správné fungování vyhledávání je tedy nutné správné formulování URL, na které se nachází endpoint se seznamem projektů. API vrací omezený seznam projektů na jedné URL a to maximálně 100 a využívá takzvaného stránkování (pagination), kdy se musí pro větší množství repozitářů přepínat mezi jednotlivými stránkami se seznamem hledaných repozitářů. Při hledání více projektů je tedy nutné do url přiřadit i atribut představující číslo stránky (*"page=[číslo stránky]"*). Do URL lze přiřadit i další zobrazovací atributy. Mezi vhodné pro použití se jeví počet repozitářů na jedné stránce (*"per_page=[počet repozitářů]"*), seřazení dle počtu hvězd (*"sort=stars"*) a se-

stupné seřazení (*"order=desc"*). Výsledný sestavený dotaz tedy může vypadat například takto:

```
https://api.github.com/search/repositories?
q=is:public&sort=stars&order=desc&per_page=50&page=1
```

Tento dotaz vrátí pouze 50 otevřených repozitářů seřazených podle počtu hvězd v sestupném pořadí. Tento dotaz lze doplnit ještě o další charakteristiky.

5.1.3 Limitace počtu projektů

GitHub REST API má limitaci pro vyhledávání, poskytuje maximálně 1000 výsledků pro každé vyhledávání. Existuje tedy maximální počet projektů, které lze jedním dotazem i za použití stránkování získat. Stránkování umožňuje rozdělit výsledky do menších částí a získat je postupně, například po 50 položkách na jednu stránku. Tím se usnadňuje práce s velkým množstvím dat. Přesto je třeba si uvědomit, že i při použití stránkování se nedostanete k více než tisíci projektům.

5.1.4 Obsah repozitářů na API

API obsahuje všechny volně dostupné informace o repozitářích. Avšak jen u některých chrakteristik se nachází přímo vypovídající hodnota. Do velké míry jsou tyto hodnoty nahrazeny URL odkazem na vlastní stránku, kde se nacházejí podrobnější informace. Vezmeme-li tento fakt v potaz, můžeme charakteristiky dále třídit na dotazově "levné" a "drahé". Jelikož pokud dostaneme informaci přímo jedním dotazem (například jméno, popis, velikost, počet open issues, ...), je to při hledání velkého množství projektů zásadně méně náročné na dotazování, než pokud budeme muset procházet jednotlivé úrovně s dalšími URL odkazy, než se konečně dostaneme na koncový bod a zjistíme hodnotu (například počet přispěvatelů nebo počet closed issues). Při výběru charakteristik na to musí být brán zřetel.

5.2 Výběr charakteristik

Jak je patrné z kapitoly 4, charakteristik je několik a je nutné mezi nimi rozlišovat. K rozdělení a hodnocení charakteristik sloužily následující vlastnosti:

5.2.1 Dotazovací náročnost

Jak již bylo zmíněno, API je limitováno počtem dotazů. Je tedy vhodné při výběru charakteristik vybírat ty, jejichž hodnota se dá zjistit za pomocí relativně nízkého počtu dotazů. Jinými slovy by bylo neefektivní plýtvat limitovaným počtem dotazů na charakteristiky, které nejsou tak důležité jako jiné. Nejvhodnějšími (nejméně dotazově náročné) jsou charakteristiky, které je možné doplnit do vyhledávací URL. Jedná se o následující charakteristiky:

- **Archivace projektu** – Jedná se o hodnotu, která určuje, zda je projekt archivován či nikoliv. V dotazovací query se jedná o výraz *"archived:true"*, pokud je projekt archivovaný a *"archived:false"*, pokud není.

- **Poslední push** – Datová informace o posledním pushnutí do repozitáře. V dotazovací query se jedná o výraz *"pushed:>[datum]"*.

- **Počet hvězd** – Informace o minimálním počtu hvězd projektů, které API vrátí jako odpověď na request. V query se jedná o výraz *"stars:>[individuálně zvolená počet hvězd]"*. Tato charakteristika může zásadně ovlivnit počet vrácených projektů. Pokud chceme, aby API vrátilo větší množství projektů, je třeba zvolit nižší číslo, avšak nesmí se jednat o úplně malou hodnotu, jelikož bychom mohli získat nekvalitní projekty. Jako výchozí hodnota bylo zvoleno *1000* hvězd, nicméně tuto hodnotu si může uživatel libovolně (pouze tedy kladná celá čísla) nastavit.

- **Počet forků** – Hodnota reprezentující minimální počet forků projektů, které API odpoví na dotaz. V query se pro nalezení počtu forků využívá výrazu *"forks:>[počet forků]"*. Jako výchozí minimální hodnota při zvolení této charakteristiky, byla zvolena hodnota 1. Tato hodnota je však volitelná a uživatel ji tedy může libovolně změnit. Stejně jako u počtu hvězd může zadat pouze celá kladná čísla větší rovna jedné.

- **Issues** – Informace o tom, zda-li má repozitář nějaké úkoly. Tato hodnota je binární a je vhodné jí nechat jako volitelnou pro uživatele.

V query se jedná o výraz *"is:issue"*, pokud repozitář má obsahovat nějaký issue.

Správnou kombinací těchto charakteristik můžeme dosáhnout maximálního počtu query atributů (5) a dosáhnout tak co nejkomplexnějšího vyhledávání za co nejmenší ztrátu přidělených requestů. Při maximálním výběru všech možných pěti query charakteristik můžeme z limitace API získat jedním dotazem až 100 projektů. Tyto projekty však, jak již bylo zmíněné, obsahují určité informace (jméno, popis, počet open issues,...), ale u většiny charakteristik obsahují další url, která odkazuje na endopoint, až na kterém je možné výslednou hodnotu vyhledat. Je proto nutné vyhodnotit, které charakteristiky jsou vhodné pro další použití přiděleného množství dotazů.

5.2.2 Informačně-přínosová hodnota charakteristiky

Dalším faktorem pro výběr charakteristik je jejich informační přínos. Z analýzy charakteristik vyplynulo několik, které se jeví jako důležitější pro dolování dat než jiné. Když z tohoto výběru ještě odebereme charakteristiky použitelné v dotazovací query (viz 5.2.1), získáme malou množinu charakteristik, které mohou o projektu mnohé vypovídat. Získání těchto charakteristik z API je však dotazově náročnější a je tak vhodné vybrat jen ty nejdůležitější. Mezi ty byl vybrán **počet přispěvatelů** (viz 4.11) a **počet otevřených a zavřených issues** (viz 4.8). Obě tyto hodnoty je více než vhodné uschovávat pro další případné zpracovávání projektů. Hodnoty počtu přispěvatelů a počtu všech issues se nacházejí na vlastních koncových bodech API a jejich vyhledávání tedy navyšuje počet potřebných dotazů. Hodnota všech issues je tvořena součtem otevřených a uzavřených issues. Počet uzavřených lze tedy získat jako rozdíl všech a otevřených issues, přičemž obě tyto hodnoty se nacházejí na jiných koncových bodech API.

5.3 Návrh aplikace

5.3.1 Rozšiřitelnost

Jeden z požadavků na aplikaci je její možná budoucí rozšiřitelnost. Rozšiřitelnost je možná zejména přidáním další zdrojů OSS projektů jiných, než je GitHub. Následně je vhodné uvažovat nad rozšiřitelností charakteristik, pokud by byla v budoucnu potřeba nějakých, které nebyly pro tuto verzi vybrány, popřípadě pokud novější verze GitHubu budou obohaceny o další charakteristiky.

5.3.2 Požadovaný výstup

Dalším požadavkem bylo vytvoření *properties* souboru, který bude přesně strukturovaný a bude obsahovat url adresu projektu a název nástroje, ze kterého data pocházejí (GitHub). Soubor se také musí jmenovat stejně jako projekt. Tento properties soubor slouží jako vstup pro nástroj SPADe.

Jako další výstupy jsou přidány textový soubor (*txt*) a soubor oddělený čárkami (*csv*). Tyto výstupy už však mohou obsahovat i další charakteristiky a parametry projektů, které byly vybrány analýzou a následně získány během vyhledávání projektů. Po vyhledání projektů se vytvoří nové soubory obsahující nalezené projekty. Na každém řádku souboru se nachází informace o jednom projektu. Informace jsou za sebou na každém řádku v následujícím pořadí:

1. Jméno
2. URL projektu
3. Velikost projektu
4. Popis
5. Licence
6. Počet open issues
7. Počet closed issues
8. Počet hvězd
9. Datum poslední aktualizace projektu (poslední push)
10. Počet přispěvatelů

5.3.3 Proces získávání projektů

Získávání projektů se liší pro každý použitý zdroj dat. Nicméně u zdrojů poskytujících API se logika získávání projektů zásadně moc neliší. Chcemeli najít projekty za pomocí GitHub API, musíme provést následující kroky:

1. **Definice kritéria vyhledávání:** Prvním krokem při hledání projektu na GitHub API je definování kritérií vyhledávání. Funkce vyhledávání GitHub umožňuje uživatelům filtrovat projekty na základě těchto kritérií. Kromě toho lze konkrétní projekty vyhledávat pomocí klíčových slov.

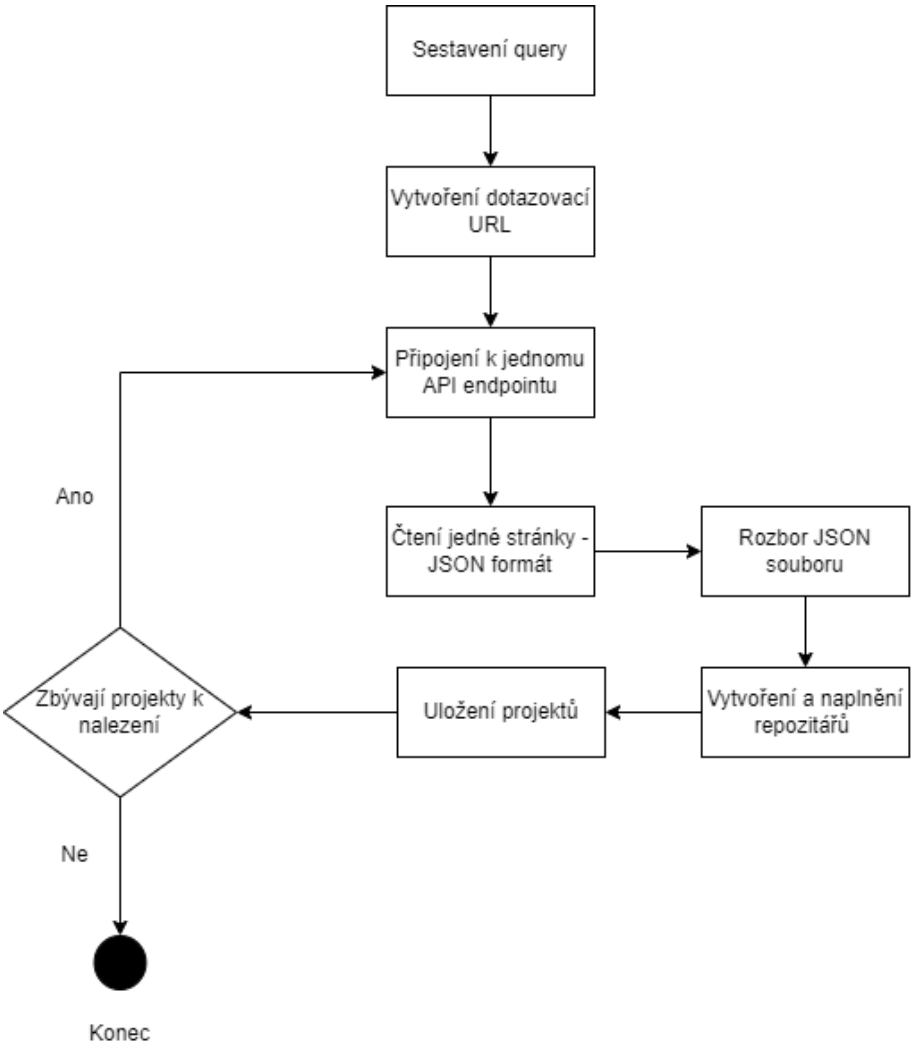
2. **Přístup k GitHub API:** Jakmile jsou kritéria vyhledávání definována, lze k rozhraní GitHub API přistupovat pomocí zvoleného programovacího jazyka. Rozhraní API obsahuje sadu koncových bodů, které umožňují interakci s úložištěm GitHub. Pro požadavky rozhraní API je vyžadováno ověření uživatele a měl by být použit klíč nebo token rozhraní API.

3. **Poslání žádosti:** Po přístupu k API lze odeslat požadavek na příslušný koncový bod. Koncový bod vrátí objekt JSON obsahující projekty, které odpovídají definovaným kritériím vyhledávání. Vrácené výsledky lze dále filtrovat pomocí parametrů požadavku.

4. **Analýza odpovědi:** Objekt JSON vrácený rozhraním API obsahuje informace o každém projektu, jako je název, popis, programovací jazyk a adresa URL úložiště. Analýza těchto informací umožňuje uživateli určit, které projekty odpovídají jejich kritériím vyhledávání.

5. **Uložení projektů:** Jedná se o vybrání vhodných informací o projektech a jejich společnému uložení. Uložené soubory poté představují výstup aplikace.

V následujícím obrázku 5.1 je nastíněn diagram s procesem získávání projektů z GitHub API. Diagram zobrazuje celý proces od sestavení dotazovací query až po uložení projektů.



Obrázek 5.1: Proces úspěšného uložení projektů

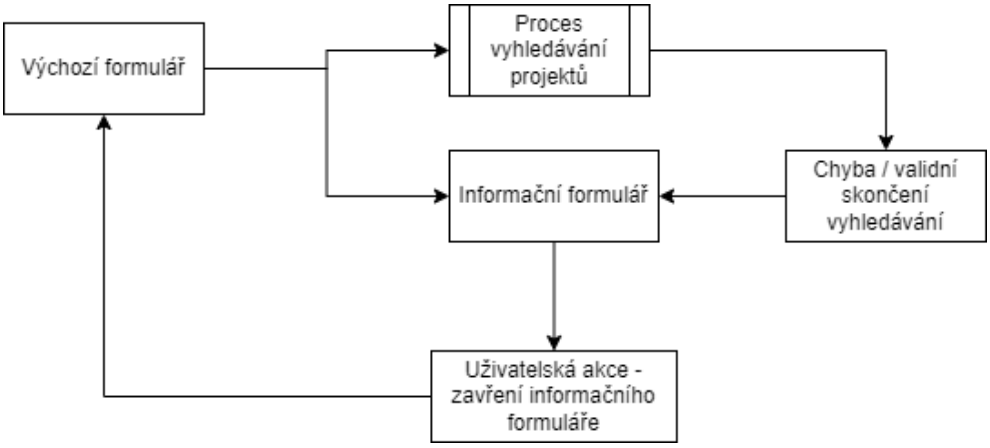
5.3.4 Struktura aplikace

Problematika vybízí k vytvoření jednoduché a přehledné aplikace. Struktura by tedy neměla být nijak složitá. Aplikace je tvořena dvěma formulářovými okny. První formulář se objeví po inicializování aplikace. Slouží k vyplnění potřebných údajů, nejdříve si uživatel zvolí zdroj dat a podle toho se mu zobrazí možnosti daného zdroje, které je nutné vyplnit (API token, přihlašovací jméno, zdroj dat a charakteristiky). Po spuštění tlačítkem k tomu určeném by mělo dojít k automatickému vyhledávání vhodných projektů, k prvnímu formuláři by tedy uživatel již neměl mít přístup. Po spuštění vyhledávání se otevře druhý informační formulář, který informuje o tom, zda aplikace stále běží a vyhledává projekty. Po skončení běhu, ať už úspěšném či neúspěšném,

se na informačním formuláři vypíše zpráva. Po validním skončení běhu programu jej bude možné z prvního formuláře pustit znovu. Za validní skončení běhu programu lze brát tyto scénáře:

- Projekty byly nalezeny a následně uloženy bez chyby.
- Nebyly nalezeny žádné projekty odpovídající vybraným charakteristikám. Uživatel například zvolil nesmyslné hodnoty charakteristik (např. u GitHubu milion hvězd – k roku 2023 neexistuje žádný takový projekt).
- Chyba: chybí token – Uživatel bude vyzván k doplnění tokenu.
- Chyba: nevalidní token – Uživatel bude vyzván k doplnění validního tokenu.
- Chyba: Překročen limit požadavků uživatele – Uživatel bude informován o chybě a bude vyzván k vyčkání a obnovení uživatelského limitu dotazů nebo ke změně autentizačních údajů.

Při nekontrolovatelné chybě v podobě nevalidní odpovědi od API nebo nepředvídatelnému chování programu, bude program ukončen a dojde tak k zamezení dalších chyb a problémů například s náročností na výkon.



Obrázek 5.2: Proces zobrazování aplikačních oken

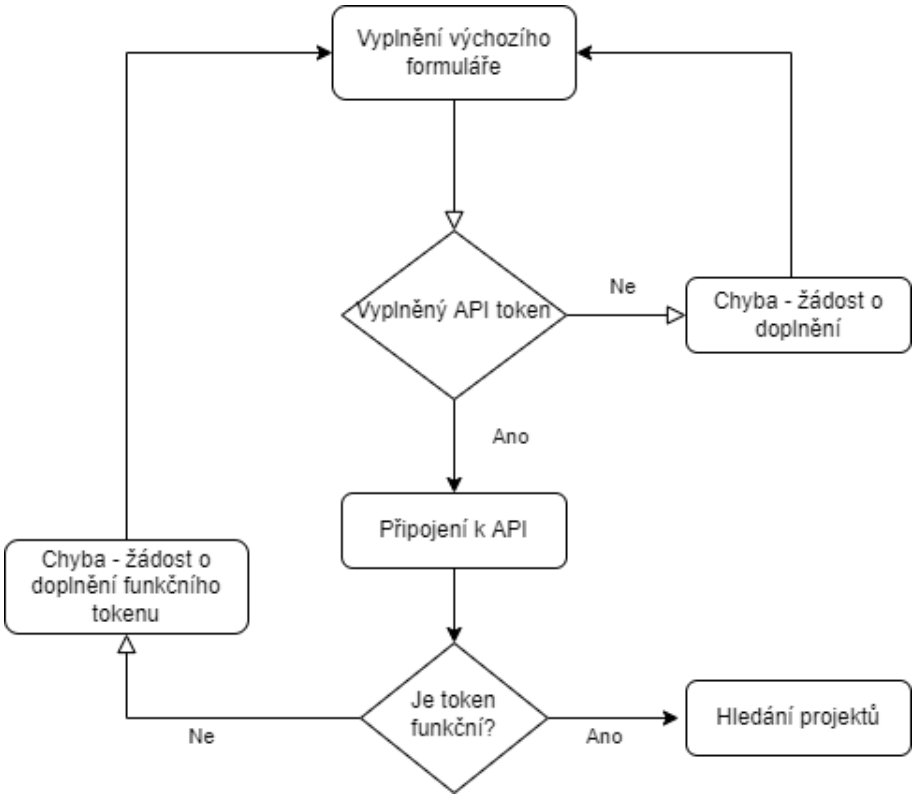
Na obrázku 5.2 se nachází diagram grafické logiky aplikace. V diagramu je nastíněn proces zobrazování oken aplikace.

Kontrola tokenů

Tokeny jsou v podstatě řetězce znaků, které slouží jako heslo a umožňují nositeli se autentizovat při odesílání požadavků na API. Většina API (GitHub zejména) dbá na autentizaci dotazů. Při připojování je tedy nutné zkontrolovat, zda uživatel zadal autentizační token. Pokud se tak nestalo, je vyzván, aby tak učinil. Zamezí se tak zbytečnému vyhledávání, které by skončilo chybovou odpovědí (chyba "401 Unauthorized") u vyhledávání charakteristik, ke kterým je ověření potřeba (například počet přispěvatelů). Tokeny na GitHubu lze vytvářet a spravovat v nastavení účtu uživatele v části „Nastavení vývojáře“. Uživatelé mohou také tokeny kdykoli odvolat a zakázat tak přístup ke svým zdrojům.

Různé koncové body API vyžadují autentizaci, aby se k nim mohlo přistupovat. Pokud uživatel nezadá platné ověřovací informace, nebo pokud jsou jeho ověřovací informace již neplatné, bude mu odepřen přístup. Aby tomuto uživatel zamezil, měl by se ujistit, že poskytuje správná ověřovací pověření nebo v případě potřeby získat nová pověření. Principu se zmíněnou chybovou odpovědí lze v aplikaci využít pro validaci tokenu. Pokud tedy uživatel zadá token, začne vyhledávání projektů a API vrátí zmíněnou chybovou odpověď, aplikace průběh vyhledávání ukončí a informuje uživatele, aby zadal funkční token. Poté může vyhledávání znovu začít.

Na obrázku 5.3 se nachází proces kontroly GitHub API tokenu aplikací pro hledání získávání projektů z tohoto API. V diagramu je nastíněno ošetření nezadání žádného tokenu a následně ošetření funkčnosti tokenu.



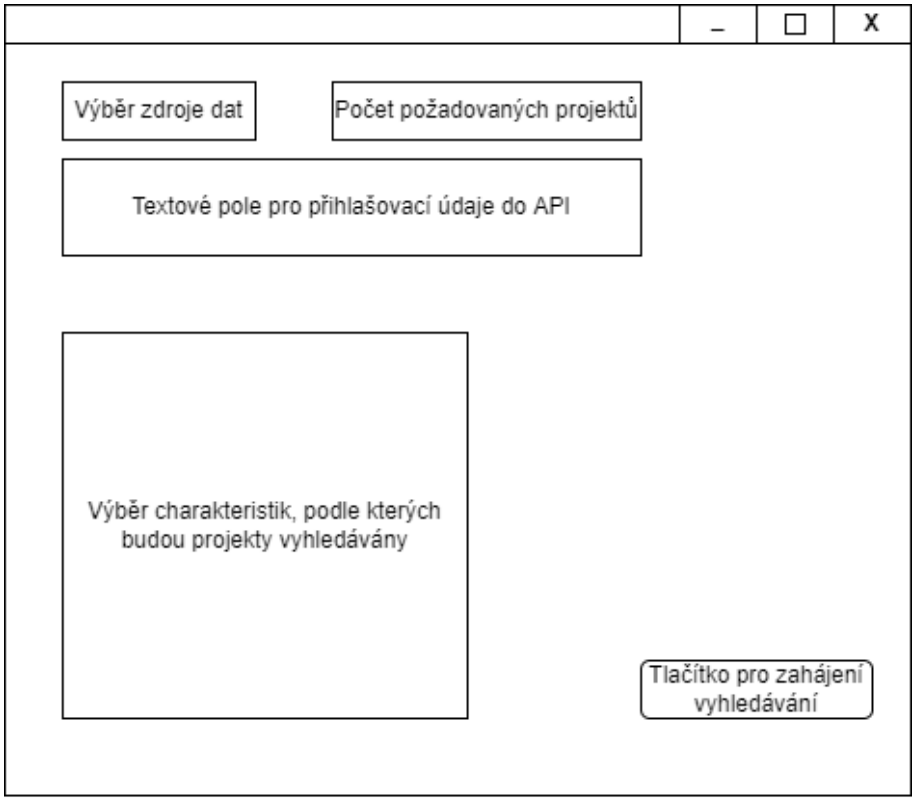
Obrázek 5.3: Proces kontroly API tokenu

5.3.5 Návrh formulářů aplikace

Jak již bylo zmíněno, pro přehlednost a jednoduchost aplikace lze její funkcionalitu navrhnout do dvou formulářů. Při inicializaci aplikace se uživateli zobrazí hlavní formulář a při spuštění procesu vyhledávání projektů se zobrazí informační formulář, který uživatele informuje o průběhu vyhledávání a případně o různých chybách.

Hlavní formulář

Jedná se o formulář, do kterého uživatel zadá potřebné údaje pro připojení k API a vybere charakteristiky, podle kterých se budou projekty vyhledávat. Následně tlačítkem spustí proces vyhledávání. Při procesu vyhledávání se formulář zamkne, aby do něj uživatel nemohl zasahovat. Formulář zůstane zamknutý, pokud je zobrazený informační formulář. Zamknutím se zamezí nevalidní manipulaci s aplikací a tím i případným chybám, jako je například další spuštění vyhledávání projektů. Na obrázku 5.4 je původní a následně i zrealizovaný návrh hlavního formuláře aplikace.



Obrázek 5.4: Návrh hlavního formuláře

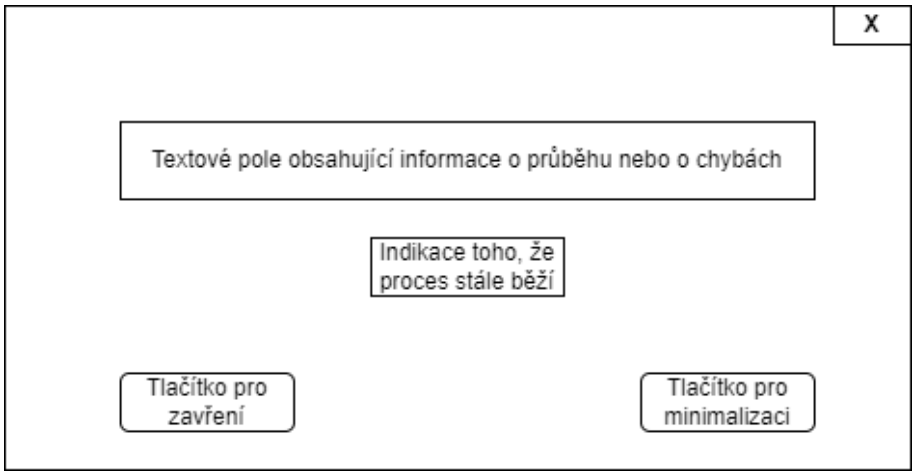
Informační formulář

Zobrazí se při vyhledávání procesu. Jelikož může být vyhledávání časově náročné, pomocí textového pole a indikátoru postupu informuje uživatele o tom, zda stále dochází k vyhledávání projektů. Tímto by mělo dojít k větší uživatelské přívětivosti aplikace (uživatel nemusí být v nejistotě, zda program běží a má program nechat běžet dál nebo zda-li došlo k chybě a měl by program ukončit).

Případné chyby nebo úspěšné dokončení vyhledávání se také zobrazí v textovém poli. Formulář také obsahuje tlačítko pro zavření a minimalizování okna. K minimalizování okna může dojít i během procesu vyhledávání projektů.

Tlačítko pro zavření se však objeví až po dokončení procesu vyhledávaní. Chce-li uživatel proces vyhledávaní ukončit, stačí, když ukončí celou aplikaci výchozím tlačítkem pro zavření okna.

Na obrázku 5.5 je původní a následně i zrealizovaný návrh informačního formuláře aplikace.



Obrázek 5.5: Návrh informačního formuláře

6 Implementace

V této kapitole bude popsána implementace, struktura a funkcionalita aplikace pro hledání OSS projektů na základě předem vybraných charakteristik. Aplikace byla vytvořena v programovacím jazyce *Java* za využití několika volně dostupných knihoven. Jazyk Java byl zvolen na základě požadavku zadavatele. Tento jazyk se i vybízí jako vhodný, zejména z důvodu, že i nástroj SPADe je implementovaný v tomto jazyce. Jednotnost jazyků nabízí několik výhod. Pokud se vývojáři dobře orientují v programovacím jazyce použitém k vývoji obou aplikací, mohou mezi nimi plynule přecházet, aniž by museli podstupovat rozsáhlou rekvalifikaci. Použití společného vývojového jazyka pro obě aplikace usnadňuje udržování jednotných standardů kódování, návrhových vzorů a architektonických preferencí napříč těmito dvěma projekty.

6.1 GUI

`JavaFX` byl vybrán jako preferovaný framework pro vytváření grafického uživatelského rozhraní (Graphical User Interface; GUI) této aplikace a to z několika důvodů.

1. JavaFX poskytuje bohatou sadu ovládacích prvků a rozvržení uživatelského rozhraní, což umožňuje vytvářet moderní a vizuálně přitažlivá rozhraní. Tyto ovládací prvky jsou vysoce přizpůsobitelné, takže je lze snadno přizpůsobit konkrétním potřebám aplikace.

2. JavaFX má vynikající podporu pro multimédia a grafiku, což umožňuje vytváření vysoce kvalitních animací a speciálních efektů. To je užitečné zejména v aplikacích, které vyžadují vysokou úroveň interaktivity a vizuální zpětné vazby.

3. JavaFX je dobře integrován se zbytkem ekosystému Java, včetně populárních IDE (Integrated development environment), jako je `Eclipse` a `IntelliJ IDEA`. To usnadňuje vývoj a ladění aplikací JavaFX pomocí známých nástrojů a pracovních postupů.

4. JavaFX poskytuje podporu napříč platformami, což umožňuje nasazení stejného kódu na více platformách, včetně Windows, macOS a Linux. To usnadňuje oslovení širšího publika s aplikací a zachování konzistentního uživatelského dojmu napříč různými zařízeními.

Všechny z výše uvedených vlastností frameworku byly použity při vytváření aplikace.

6.1.1 Hlavní formulář

Pro ovládání hlavního formuláře slouží třída `Controller.java`. Jedná se o třídu řadiče `JavaFX`, která implementuje rozhraní `Initializable`. Zodpovídá za zpracování uživatelských interakcí s uživatelským rozhraním `MainForm.fxml`, které obsahuje několik textových polí, polí se seznamem a zaškrtávacích políček.

Třída definuje několik proměnných instance, které reprezentují prvky uživatelského rozhraní, jako je `tbName, tbKey, cmbSource, cmbNumber, cbStars, cbIssues a cbArchived`. Definuje také několik polí výchozích hodnot pro pole se seznamem: `Zdroje, Projekty a Roky`.

Metoda `initialize` se zavolá při prvním načtení ovladače a nastaví výchozí hodnoty pro pole se seznamem a ve výchozím nastavení skryje určité prvky uživatelského rozhraní.

Třída také definuje několik metod, které zpracovávají uživatelské interakce s uživatelským rozhraním, jako jsou `SetPush, SetYear, SetIssues, SetStars, SetArchived a SetForks`. Tyto metody jsou volány, když uživatel interaguje s odpovídajícími prvky uživatelského rozhraní, jako jsou zaškrtávací políčka a pole se seznamem. Podle toho aktualizují hodnoty proměnných instance.

Po kliknutí na tlačítko „Find projects“ se spustí metoda `FindProjects`. Metoda slouží ke spuštění získávání projektů z API a tedy k se spuštění hlavní výpočetní fázi aplikace. Tato metoda používá JavaFX framework pro vytvoření uživatelského rozhraní a zobrazení upozornění a spuštěných stavů. Také vytváří nové vlákno pro běh objektu API na pozadí, aby se zabránilo blokování UI vlákna. Metoda používá různé třídy a metody jazyka Java k plnění svých úkolů, včetně *Alert, FXMLLoader, Task a Stage*. Zde je stručný přehled toho, co tato metoda dělá:

1. Získá vybraný zdroj dat a počet projektů, které se mají najít, z odpovídajících rozevíracích seznamů.

2. Získá název a klíč API, které uživatel zadal. Pokud chybí klíč API, uživatel bude upozorněn na to, aby zadal fungující klíč API.

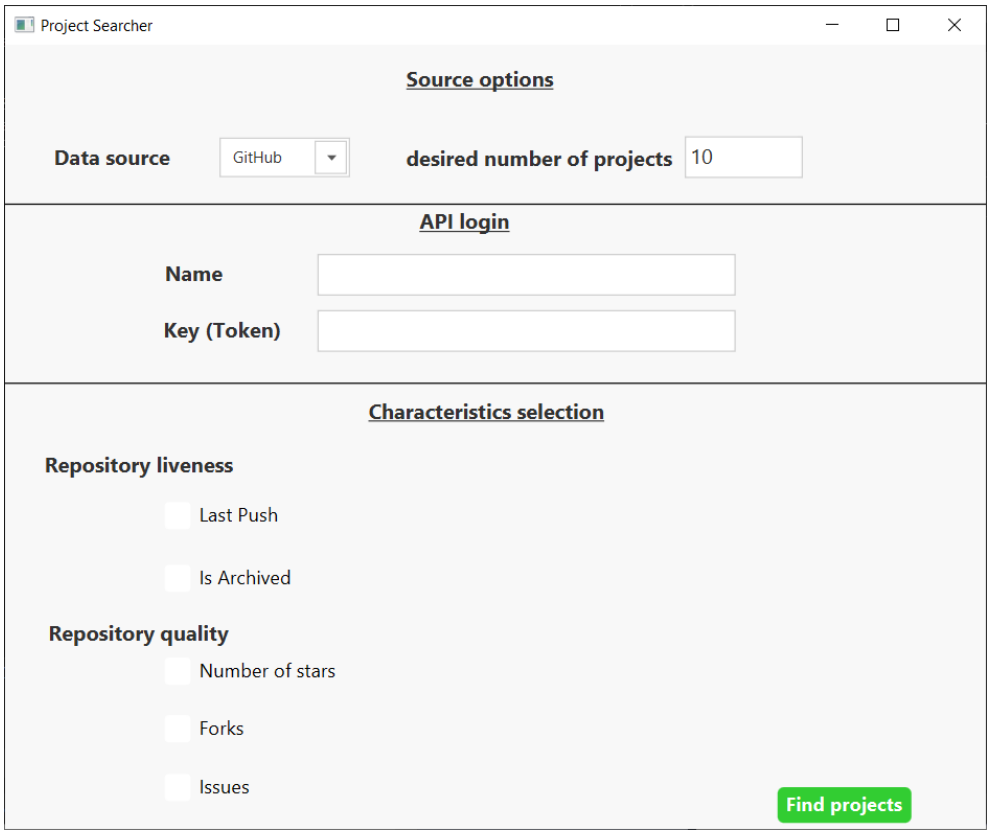
3. Pokud je klíč API přítomen, zobrazí se potvrzovací upozornění, zda uživatel chce začít hledat projekty.

4. Pokud uživatel potvrdí, vytvoří se objekt API s potřebnými parametry a spuštěný stav bude zobrazen k ukázání průběhu hledání projektu.

5. Vytvoří se úloha (Task) k běhu objektu API na pozadí, aby se zabránilo blokování UI vlákna.

6. Po uzavření spuštěného stavu se zkontroluje, zda je objekt API stále v provozu. Pokud ano, program bude ukončen. Pokud ne, pouze se uzavře spuštěný stav a zobrazí se hlavní stav.

Celkově tato třída slouží jako prostředník mezi uživatelským rozhraním a backendovou logikou aplikace. Zachycuje vstupy uživatelů a převádí je na smysluplná data, která aplikace používá k provádění určitých úkolů.



Obrázek 6.1: Vzhled hlavního formuláře

Na obrázku 6.1 je okno hlavního formuláře aplikace pro zvolený zdroj dat "GitHub".

6.1.2 Informační formulář

Pro ovládání funkcí informačního formuláře, který se spustí při zahájení vyhledávání projektů, slouží třída `RunningController.java`. Tato třída obsahuje sadu metod a atributů, které umožňují spravovat prvek GUI, který zobrazuje průběh dlouhotrvajícího procesu. Tento ovladač se používá ve spojení se souborem `RunningForm.fxml` a poskytuje funkce, jako je aktualizace indikátoru průběhu, nastavení labelů s informativním textem a zobrazení/skrytí tlačítka `Zavřít`, které zavře formulář. Vzhled formuláře je patrný na obrázku 6.2.

Třída `controller` má několik veřejných metod, ke kterým lze přistupovat z jiných částí aplikace. Může se například použít metoda `setInfoLabel()` k aktualizaci informačního štítku novou zprávou nebo metoda `setDoneLabel()` k nastavení textu hotového štítku. Kromě toho lze použít metodu `setProgressIndicator()` k zobrazení, že proces běží, nebo metodu `showButton()` k zobrazení nebo skrytí tlačítka `Zavřít`.

Třída také obsahuje metodu `initialize()`, která inicializuje prvky GUI a spustí vlákno, které aktualizuje indikátor průběhu a tlačítko Zavřít. Kromě toho existují dvě metody, `setIsRunning()` a `getIsRunning()`, které umožňují nastavit a získat hodnotu booleovského příznaku, který označuje, zda proces stále běží nebo nikoliv.

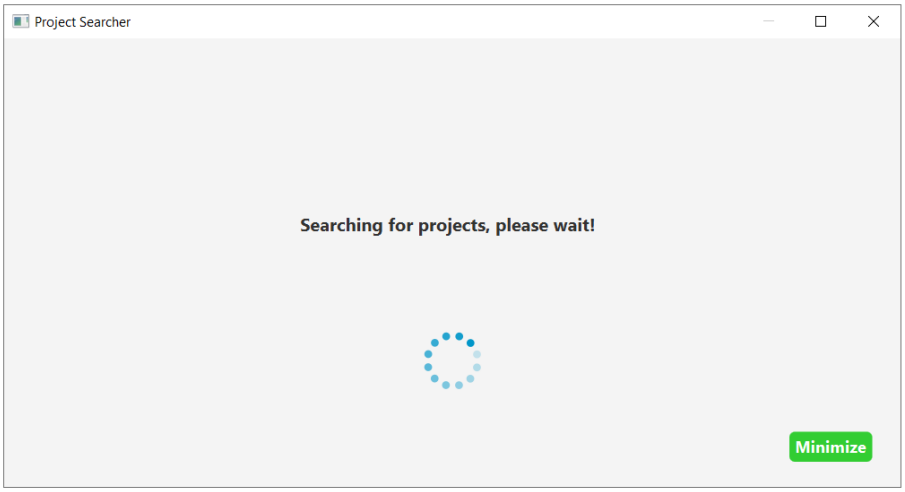
Nakonec existují dvě metody obsluhy události,`Minimize()` a `Close()`, které se spouštějí, když uživatel klikne na tlačítko minimalizace nebo zavřít. Tyto metody získají zdrojový uzel události, získají objekt `Stage`, který obsahuje uzel, a provedou odpovídající akci (minimalizace nebo uzavření fáze).

Celkově třída `RunningController` poskytuje jednoduchý, ale efektivní způsob, jak řídit průběh dlouhotrvajících procesů v JavaFX aplikaci.

6.2 Komunikace s API

Třída `GitHubAPI` poskytuje funkce pro vyhledávání veřejných úložišť na GitHubu pomocí GitHub API. Třída obsahuje proměnné instance, které obsahují klíč API, počet úložišť na stránku, počet nalezených úložišť, počet uskutečněných požadavků a odkaz na objekt `Controller` a `RunningController`. Objekt `Controller` je zodpovědný za správu uživatelského rozhraní aplikace, zatímco objekt `RunningController` je zodpovědný za správu stavu aplikace.

Konstruktor třídy `GitHubAPI` přebírá několik parametrů, včetně objektu `Controller`, objektu `RunningController`, názvu API, klíče API, roku, příznaku issues, příznaku hvězdiček, archivace, forks a počet projektů k vyhle-



Obrázek 6.2: Vzhled informačního formuláře

dávání. Klíč API je pro konstruktor povinným parametrem, zatímco ostatní parametry jsou volitelné. Konstruktor inicializuje proměnné instance a vytvoří nové vlákno pro vytváření požadavků API na hledání zadaného počtu projektů.

Metoda `createUrl()` třídy `GitHubAPI` je zodpovědná za vytvoření řetězce URL pro prohledávání veřejných úložišť na GitHubu na základě vyhledávacích kritérií zadaných v parametrech konstruktoru. Metoda přebírá parametry, jako je rok, příznak vydání, příznak hvězd, archivovaný příznak, příznak rozvětvení a číslo stránky. Metoda vrací řetězec URL, který lze použít k hledání veřejných úložišť na GitHubu.

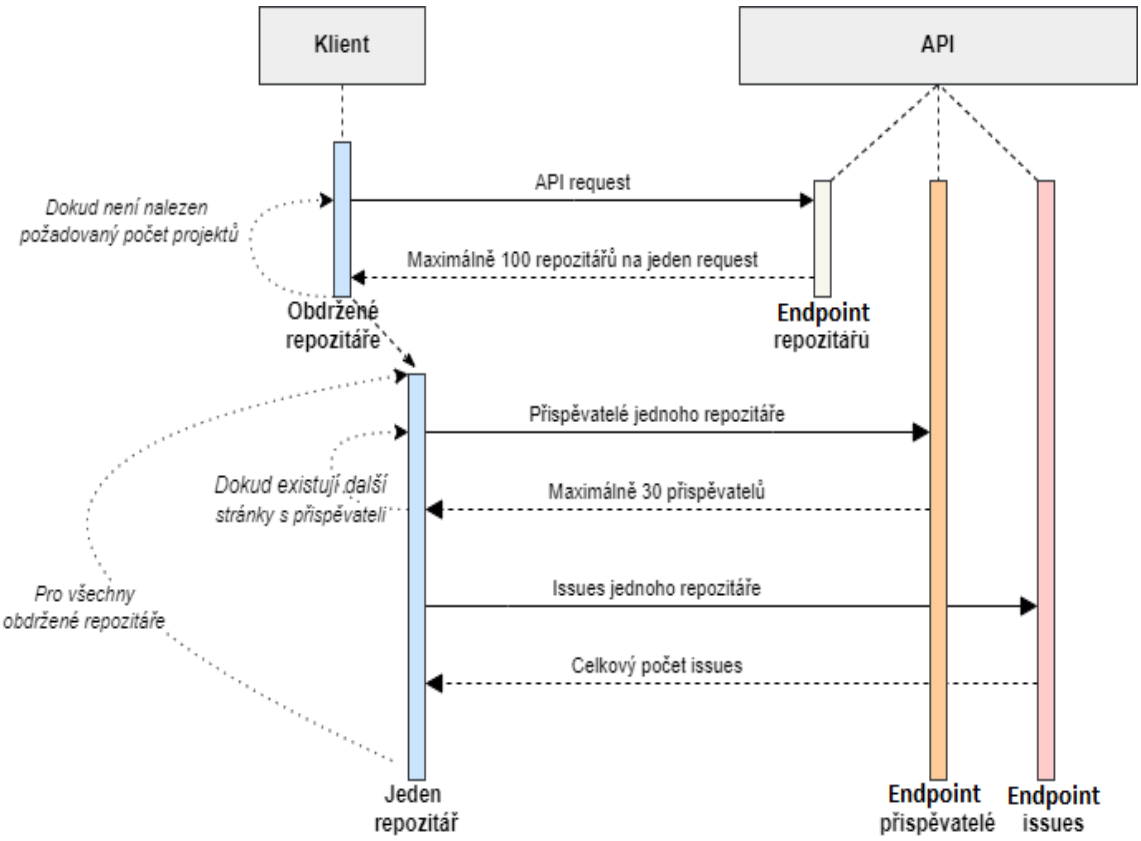
Metoda `connectToApi()` třídy `GitHubAPI` je zodpovědná za vytvoření připojení k GitHub API pomocí zadané adresy URL a klíče API. Metoda bere URL a klíč API jako parametry a vrací objekt `HttpURLConnection`, který lze použít ke čtení dat z API.

Metoda `getAllProjects()` získá po připojení k API celkový počet projektů, které odpovídají dotazu. Tato hodnota se následně využívá pro randomizaci projektů.

Metoda `fillProjects()` třídy `GitHubAPI` je zodpovědná za čtení dat z API a naplnění projektů získanými daty. Metoda bere jako parametry objekt `HttpURLConnection` a klíč API a čte data z API pomocí objektu `HttpURLConnection`. Metoda pak naplní projekty získanými daty.

Třída `GitHubAPI` také obsahuje několik instančních proměnných, které se používají ke sledování stavu aplikace. Například proměnná `NumberOfReposFound` sleduje počet úložišť nalezených při hledání, zatímco proměnná `NumberOfRequests` sleduje počet požadavků na GitHub API.

Celkově třída `GitHubAPI` poskytuje užitečný a pohodlný způsob, jak hledat veřejná úložiště na GitHubu pomocí GitHub API. Třída je dobře organizovaná, s jasnými a stručnými metodami, které jsou snadno pochopitelné a použitelné. Třída má také dobré zpracování chyb, s příslušnými zprávami zobrazenými uživateli, pokud jsou v odpovědích API nějaké chyby.



Obrázek 6.3: Proces komunikace s API

Obrázek 6.3 popisuje komunikaci mezi klientem (aplikací pro hledání projektů) a GitHub API.

Randomizace projektů

GitHub API vrací stejnou odpověď na stejně složený dotaz. Maximálně může jednomu uživateli vrátit tisíc nejlépe hodnocených repozitářů. Tyto repozitáře jsou rovnoměrně distribuovány ve vrácených stránkách (maximálně 100 repozitářů na jedné stránce). Pokud chce uživatel vyhledat méně projektů, musí být ošetřeno alespoň částečné vrácení různých projektů. Pokud tedy uživatel zvolí charakteristiky, kterým odpovídají tisíce projektů, API dovolí uživateli přistoupit pouze k prvnímu tisíci z nich. Z této podmnožiny

je vhodné při každém spuštění vybrat různé projekty. Toto je v aplikaci ošetřeno tak, že při každém připojení k API se zjistí celkový počet vrácených projektů (ten se může pohybovat ve statisících). Pokud je tento počet větší než maximální počet přístupných projektů (1000), dojde k náhodnému výběru stránky, ze které se projekty pomocí JSON objektů následně budou získávat. Tímto ošetřením by se mělo omezit duplikacím vyhledaných a zpracovaných projektů při několikanásobném spuštění aplikace. Jedná se však pouze o částečné (jediné možné) řešení a k duplikaci může dále docházet. Toto omezení není totiž s možnostmi nabízenými GitHub API nijak řešitelné.

Dotazová náročnost

Počet případů, kdy se musí požádat o informace prostřednictvím rozhraní API, závisí na tom, který koncový bod a parametry jsou použity. Vezměme si například to, že chceme zavolat repozitáře přes vyhledávací URL. Odeslání pouze jednoho požadavku na koncový bod vyhledávání/repozitář se specifickými parametry dotazu je často dostačující. Pokud však hledáme podrobnější údaje o každém úložišti (např. záznamy přispěvatelích), je třeba odeslat více požadavků směrem k příslušným koncovým bodům, jako jsou `repos/:owner/:repo/contributors` – jeden na vrácenou položku z původního dotazu.

Akce	Metoda	Počet requestů
Připojení k endpointu s repozitáři	`connectToApi()`	1
Zjištění počtu všech projektů	`getAllProjects()`	1
Zjištění počtu všech přispěvatelů	`getApiContributors()`	≥ 1
Zjištění počtu všech přispěvatelů	`getApiIssues()`	1

Tabulka 6.1: Přehled potřebných requestů pro jeden projekt

Tabulka 6.1 představuje souhrn nezbytných akcí, metod a počtu požadavků požadovaných k provedení různých úkolů pro jeden projekt pomocí

rozhraní API. Tabulka uvádí čtyři akce, z nichž každá odpovídá specifické metodě a počtu požadavků požadovaných k provedení úlohy.

Zjištění celkového počtu přispěvatelů může být nejnáročnější charakteristikou, co se počtu dotazů týče. Pokud má repozitář více jak 30 přispěvatelů, nestačí pouze jeden dotaz. Musí se pomocí stránkování prohledat všechny stránky na koncovém bodu `.../contributors` a na každé stránce iterativně spočítat jednotliví přispěvatelé. Na každou další stránku poté připadá další použitý dotaz.

Rychlost dotazování a odezvy

Existuje několik faktorů, které mohou ovlivnit rychlost GitHub API. Mezi ně patří mimo jiné: specifika koncových bodů, množství požadovaných dat a podmínky sítě mezi koncovými body klienta a serveru. API je obvykle známé pro svou schopnost poskytovat rychlou odezvu, kdy většina odpovědí proběhne během několika stovek milisekund až do několika sekund poté, co byl na server odeslán požadavek.

Stojí za zmínku, že různé komponenty mají vliv na rychlost, s jakou GitHub API reaguje. Mezi takové faktory patří mimo jiné: složitost požadovaných dat, množství vydaných požadavků a stav autorizace žadatele. Kromě toho může být doba odezvy omezena rychlostními limity uloženými ve vrstvě API. Situace, kdy uživatelé překročí tato omezení, by mohla mít za následek omezení nebo dočasné pozastavení přístupu k uvedenému rozhraní.

Aby byly rychlost vyřízení požadavků GitHub API efektivnější, musí se dodržovat doporučené osvědčené postupy. Patří mezi ně implementace vhodných metod ukládání do mezipaměti a snížení objemu požadavků při současné optimalizaci datových dotazů – výsledkem je celkově méně odpovědí vrácených rozhraním API. Kromě ukládání do mezipaměti, které by v aplikaci nemělo využití, jsou implementovány všechny ostatní postupy.

6.3 Ukládání projektů

Pro ukládání projektů slouží třída `Writer.java`. Tato třída je zodpovědná za vytvoření výstupního souboru a souboru vlastností pro každé úložiště v daném poli. Třída obsahuje dvě metody, `writeFile` a `writeToProperties`.

Metoda `writeFile` má čtyři parametry: objekt `RunningController`, pole objektů `Repository`, `String` představující typ souboru, který se má vytvořit, a `int` představující počet zbývajících projektů, které mají být prohledány. Tato metoda vytvoří složku s názvem „ProjectsFound“ (pokud již

neexistuje) s aktuálním časovým razítkem jako názvem. Poté vytvoří soubor s názvem „Projects_“, za kterým následuje aktuální časové razítko a zadaná přípona souboru. Pokud soubor ještě neexistuje, vytvoří se nový soubor. Pokud soubor již existuje, přidá do stávajícího souboru nové projekty. Metoda pak zapíše informace o každém projektu do souboru, přičemž každý řádek představuje celý projekt (název, velikost, popis atd.). Pokud nezůstávají žádné projekty, metoda zobrazí zprávu o úspěchu a zobrazí tlačítko "Close".

Metoda `writeToProperties` bere jako parametr pole objektů `Repository`. Tato metoda vytvoří adresář s názvem "Properties" (pokud již neexistuje) a podadresář s aktuálním časovým razítkem jako názvem. Poté iteruje každé úložiště v daném poli a pro každé vytvoří soubor vlastností. Soubor vlastností obsahuje čtyři řádky: `privateKey=`, `url=<url_úložiště>.git`, `tool=GITHUB` a `repo=<url_úložiště>.git`. Soubor se vytvoří s názvem projektu jako názvem souboru a uloží se do dříve vytvořeného podadresáře. Projekty jsou průběžně ukládány po větším množství a ne pouze po jednom. U GitHub projektů bylo zvoleno ukládání po počtu projektů na endpointu s repozitáři. Jinými slovy, pokud se na endpointu nachází 10 projektů, budou uloženy až po jejich zpracování. Vytvoří se tedy soubor s 10 projekty a poté může aplikace dále pokračovat na další endpoint, ze kterého získá dalších 10 projektů a pro ty následně vytvoří nový soubor, do kterého projekty uloží a tak dále, dokud nebudou nalezeny všechny projekty. Během jednoho běhu aplikace se tak může vytvořit několik adresářů s různými časovými značkami pro `properties` soubory a několik `csv/txt` souborů, obsahujícími přímo projekty, taktéž s různými časovými značkami. Tento princip byl zvolen jako kompromis žádného průběžného ukládání a ukládání projektů po jednom. Jelikož vyhledávání většího počtu projektů může být časově náročné a bez průběžného ukládání by například při překročení limitů API mohlo dojít ke ztrátě již získaných projektů, nevyužití průběžného ukládání není vhodné. Nicméně na druhou stranu ukládání projektů po jednom může být při velkém množství projektů paměťově a výpočetně náročnější. Tento kompromis se tedy jeví jako zlatá střední cesta mezi užitečností a výkonností. Třída `Write` je nicméně připravena na jakoukoliv možnost.

6.4 Rozšířitelnost

V zadání práce je zmíněno, že by aplikace měla být rozšířitelná. Na to byl brán zřetel. Metody jsou jednoduše a přehledně pojmenovány a celý kód je

důkladně okomentován, program je rozdělen do menších modulů nebo komponent, z nichž každý odpovídá za určitou funkcionalitu. Program má jasně oddělené činnosti, přičemž každý modul by měl odpovídat za konkrétní úkol nebo funkci. To usnadňuje úpravu nebo výměnu modulu bez ovlivnění ostatních částí kódu. Usnadňuje to přidávání nových funkcí bez ovlivnění stávajícího kódu. Toto vše bylo provedeno ze snahy o co největší zjednodušení následného rozšíření aplikace. Nicméně modularita aplikace byla ovlivněna využitím frameworku JavaFX a jeho funkcionalitou. Omezení je zřejmé například tím, že každý formulář má jeden kontrolér, který odpovídá za dění v daném formulářovém okně. Jelikož byla snaha nepředimenzovat aplikaci, ale ponechat ji co nejjednodušší pro uživatele, je využito dvou formulářů a tedy i dvou kontrolérů. Možné negativum tohoto lze vnímat v přehlednosti v třídě hlavního kontroléru (`Controller`), jelikož ten musí obsahovat všechny proměnné a FXML objekty (např. textová pole, tlačítka, boxy, ...) pro každý zdroj. Grafická modulárnost různých zdrojů dat je zde implementována za pomocí tzv. kotevních panelů (Anchor pane), ty se používají pro ukotvení k odsazení od okrajů hlavního panelu a mohou obsahovat všechny FXML objekty daného zdroje. Logika výběru mezi zdroji pak spočívá v tom, že pro každý zdroj dat je vytvořen vlastní Anchor pane, který obsahuje specifické atributy daného zdroje. Přepínáním pomocí comboboxu pro výběr zdrojů pak dochází ke střídání právě zobrazeného panelu (pomocí funkce „.setVisible()“). Pokud by tedy mělo dojít k rozšíření aplikace o další zdroj dat, stačí vytvořit nový panel s vybranými grafickými prvky a přidat jej do hlavního formuláře `MainForm.fxml`. V kontroléru stačí následně přidat nový zdroj do výběrového comboboxu a ošetřit jednotlivě vstupy a funkce nově přidaných prvků stejně, jako je tomu u již implementovaného GitHubu (lze využít již připravených metod například pro nastavení jména/klíče nebo metody spravující validitu hodnot v textových polích). Následně musí přidat do projektu třídu pro získávání dat z nového zdroje, která musí obsahovat logiku připojování a získávání projektů ze zdroje a musí v ní být ošetřeny všechny případné výjimky – tento krok je zásadním a vyžaduje důslednou analýzu například možností daného API. Poté již z kontroléru stačí zavolat metodu třídy implementující logiku získávání projektů z daného zdroje. Kód je důkladně okomentován a i místa pro zavedení metod, proměnných a dalších funkcionalit pro přidání nového zdroje jsou v kódu zvýrazněna a kód je celkově připraven pro možné rozšíření.

Jako nejvíce zjevné rozšíření se nabízí doplnění aplikace o další zdroje OSS projektů (GitLab, Jira, ...). Při analýze možností API různých zdrojů bylo zjištěno, že u GitLabu by se logika připojování a získávání dat z API neměla nijak výrazně lišit od GitHubu. Například autentizace za pomocí

tokenu funguje obdobně. Nicméně je zde několik odchylek, které by se musely ošetřit (koncové body GitLab API například neobsahují stejné informace o projektech, jako je tomu u GitHubu).

Další rozšíření může spočívat v přidání dalších charakteristik projektů. K tomu může dojít při přidání nového zdroje dat, nicméně pokud by byl z nějakého konkrétního důvodu zájem o charakteristiku, která v rámci získávání projektů z GitHubu, není implementována, kvůli modulárnosti aplikace by přidání nové charakteristiky nemělo být nijak omezeno.

7 Testování

První funkční verze aplikace byla uživatelsky testována, aby se zavčasu zjistily nedostatky a možná vylepšení a došlo k jejich ošetření.

Během vývoje byl důraz kladen na průběžné manuální a jednotkové testování vybraných metod. Složitější a důležité metody programu byly testovány pomocí jednotkových testů, aby se zajistila jejich správná funkcionalita, výkonnost a odolnost vůči chybám. Tyto metody představují klíčové části systému, jejichž správná funkčnost je nezbytná pro celkovou spolehlivost a účinnost aplikace.

Během vývoje aplikace bylo kromě průběžného jednotkového testování také prováděno manuální testování několika klíčových metod systému. Tento přístup byl zvolen zejména z důvodu různých omezení a odpovědí API, které bylo třeba ošetřit a validovat v praxi. Manuální testování umožnilo lépe pochopit chování API, identifikovat potenciální problémy a řešit je efektivněji.

Při manuálním testování klíčových metod byla prováděna kontrola, zda aplikace správně komunikuje s API, zda řádně zpracovává různé typy odpovědí a zda správně reaguje na různá omezení, jako jsou například limity počtu požadavků nebo formáty dat. Díky tomuto přístupu bylo možné odhalit a opravit chyby, které by mohly být přehlédnuty při použití pouze jednotkových testů, a zároveň zvýšit důvěru v kvalitu a spolehlivost aplikace. Jelikož má aplikace i GUI, bylo důležité manuálně otestovat a ošetřit různé uživatelské vstupy a celkově zajistit co nejlepší uživatelský zážitek.

7.0.1 Uživatelské testování

Vybírání uživatelů pro testování

Pro testování aplikace byli vybráni čtyři uživatelé, kteří splňovali určitá kritéria. Uživatelé byli vybíráni na základě různých kritérií, jako je věk, úroveň zkušeností s podobnými aplikacemi a různé požadavky na použití aplikace. Tyto kritéria byla zohledněna při výběru uživatelů, aby testování bylo co nejobjektivnější a účelné.

Představení aplikace a poskytnutí pokynů k testování

Před testováním byla aplikace představena uživatelům a byly poskytnuty pokyny k testování. Uživatelům bylo zadáno, aby používali aplikaci po určitou dobu a zapisovali si poznámky o aplikaci. Zpětná vazba byla poskytnuta

písemně, aby bylo možné získat co nejvíce podrobností o zkušenostech uživatelů s aplikací.

Zpětná vazba od uživatelů

Po dokončení testování byla poskytnuta ústní zpětná vazba od všech čtyř uživatelů. Zpětná vazba obsahovala informace o tom, jak uživatelé aplikaci používali, co se jim líbilo a nelíbilo, a také případné chyby nebo problémy, na které narazili. Zpětná vazba byla podrobná, což umožnilo získat cenné informace o uživatelské přívětivosti aplikace, na kterou byl brán největší zřetel.

Analýza zpětné vazby a úpravy aplikace

Na základě zpětné vazby byly provedeny úpravy aplikace. Snažilo se brát v úvahu všechny poznatky získané z testování, aby bylo možné vylepšit aplikaci. Byly provedeny úpravy, které měly zajistit, aby byla aplikace snadněji použitelná a aby byla co nejvíce v souladu s požadavky uživatelů. Například došlo k nahrazení dříve používaných comboboxů s počty hvězd a forků za textová pole, do kterých může uživatel zapsat hodnoty sám. Došlo tedy spíše k otevření uživatelských možností. Nicméně aplikace je pro uživatele, který nemá o problematice přehled, bez patřičných pokynů v zásadě jednoduchá na použití, ale složitá na pochopení využití aplikace.

7.0.2 Testování aplikace

Kromě testování aplikace uživateli byla aplikace také testována pomocí jednotkových (JUnit) testů. JUnit testy jsou automatizované testy, které slouží k ověření správné funkcionality různých částí aplikace. Tyto testy jsou důležité, protože umožňují rychle a opakovaně ověřit správnost chování aplikace po provedení změn.

Při vytváření JUnit testů byly testovány všechny hlavní funkce aplikace a to zejména funkce ve třídě implementující připojování ke GitHub API a v třídě sloužící pro zápis projektů do výstupních souborů. Testy byly provedeny na různých úrovních.

Testování aplikace pomocí JUnit testů bylo úspěšné a ukázalo, že klíčové části aplikace fungují správně a bez chyb. Pokud se v budoucnu provedou změny v aplikaci, bude možné znovu spustit JUnit testy, aby se ověřilo, zda nové změny nepřinesly žádné chyby do aplikace.

Během testování aplikace byla testována také odolnost aplikace proti nevalidním vstupům. Bylo ověřeno, zda aplikace správně zvládá neočekávané

situace a zda umí vyhodnotit a obsloužit chybné vstupy uživatele.

Výsledky testování ukázaly, že aplikace dokáže správně reagovat na nevalidní vstupy a zobrazovat uživateli srozumitelné zprávy o chybách. Nicméně se nepředpokládá, že by uživatelé záměrně zadávali nevalidní informace nebo se ze zásady pokoušeli běh aplikace nějak narušovat, jelikož okruh případných uživatelů zahrnuje akademickou sféru a zejména členy projektu SPADe, kterým by aplikace měla práci usnadnit.

8 Závěr

Náplní této diplomové práce bylo seznámit se problematikou open-source softwarových (OSS) projektů, využití nástrojů pro jejich řízení a možnostmi těžby jejich dat. Následně prozkoumat dosavadní zdroje popisující různé charakteristiky OSS projektů a jejich dat s přihlédnutím k detekci procesních anti-vzorů. Poté navrhnout co nejkompletnější sadu charakteristik určujících vhodnost OSS projektu k těžbě a analýze jeho dat. Dále navrhnout aplikaci pro identifikaci projektů z daného zdroje na základě zadaných charakteristik. A poté implementovat tuto aplikaci pro jeden zdroj OSS projektů a vybranou podsadu charakteristik. Nakonec tuto aplikaci otestovat a ověřit vhodnost výstupu procesu identifikace projektů.

Došlo k identifikaci jednotlivých charakteristik OSS projektů. Z nich bylo vybráno několik nejvhodnějších a u nich byla provedena podrobná analýza s přihlédnutím na těžbu dat zaměřenou na hledání procesních anti-vzorů. Těchto charakteristik bylo identifikováno 14. Z nich byla poté zvolena nejlépe se jevící podmnožina. Výběru této podmnožiny předcházela analýza nástrojů pro tvorbu OSS projektů a vybrání nejvhodnějšího z nich (GitHub). U tohoto nástroje došlo k analýze možnosti získávání informací o OSS projektech a byla vybrána možnost jeho REST API. Toto API bylo také nejdříve důkladně prozkoumáno a zanalyzováno, aby se zjistily jeho možnosti a omezení. Poté byla vybrána podmnožina charakteristik, které v implementované aplikaci slouží k filtrování vyhledávaných OSS projektů.

Výstup této aplikace může sloužit jako vstup pro nástroj SPADe, který slouží pro identifikaci procesních anti-vzorů a zároveň je přidán i výstup, který obsahuje podrobné informace o projektech. Toto může ulehčit následné vybrání projektů, jejichž data se mohou následně těžit a analyzovat. Při vývoji aplikace bylo dbáno na možnost jejího následného rozšíření. Aplikace i její výstupy byly důkladně analyzovány a byla testována jejich vhodnost pro další použití. V současném stavu může aplikace sloužit pro selekci vhodných projektů, jejichž následné využití může sloužit k dosažení významných výzkumných výsledků při analýzách open-source a průmyslových projektů.

Výstupy této práce jsou analýza charakteristik OSS projektů, aplikace pro identifikaci projektů podle zvolených charakteristik a properties soubory s jednotlivými projekty (tyto soubory slouží jako vstup pro nástroj SPADe) a soubory, které obsahují podrobné informace o jednotlivých projektech.

Literatura

[1] Amazon. *What is a data warehouse?* [online]. Amazon, 2019. [cit. 2023/03/28]. Dostupné z: https://aws.amazon.com/data-warehouse/.

[2] Atlassian. *DevOps: Breaking the Development-Operations barrier* [online]. Atlassian.com, 2019. [cit. 2022/04/21]. Dostupné z: https://www.atlassian.com/devops.

[3] Bass, L. – Weber, I. – Zhu, L. *DevOps: A Software Architect's Perspective*. SEI Series in Software Engineering. Pearson Education, 2015. Dostupné z: https://books.google.cz/books?id=fcwkCQAAQBAJ. ISBN 9780134049878.

[4] Beck, K. – Gamma., E. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000. ISBN 978-0321278654.

[5] Bocasay. *The Kanban method in IT development projects* [online]. https://www.bocasay.com/. [cit. 2023/02/20]. Dostupné z: https://www.bocasay.com/kanban-method-it-development-projects/.

[6] AB Tasty, F. *Continuous Integration and Delivery (CI/CD) Explained* [online]. AB Tasty, 2023. [cit. 2023/03/22]. Dostupné z: https://www.flagship.io/ci-cd/.

[7] Edeki, C. Agile software development methodology. *European Journal of Mathematics and Computer Science*. 2015, 2, 1.

[8] Geeks, G. *ETL Process in Data Warehouse* [online]. Geeks for Geeks, 2019. [cit. 2023/02/12]. Dostupné z: https://www.geeksforgeeks.org/etl-process-in-data-warehouse/.

[9] GitHub. *GitHub Blog* [online]. github.blog, 2022. [cit. 2023/04/20]. Dostupné z: https://github.blog/2022-11-17-octoverse-2022-10-years-of-tracking-open-source/.

[10] GitHub, I. *Archiving repositories* [online]. GitHub, 2023. [cit. 2023/02/14]. Dostupné z: https://docs.github.com/en/repositories/archiving-a-github-repository/archiving-repositories.

[11] GitHub, I. *About issues* [online]. GitHub, 2023. [cit. 2023/02/15]. Dostupné z: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.

[12] GitHub, I. *About size limits on GitHub* [online]. GitHub, 2023. [cit. 2023/02/14]. Dostupné z: https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-large-files-on-github.

[13] Greene, J. – Stellman, A. *Applied Software Project Management*. O'Reilly, first edition, 2005. ISBN 0596009488.

[14] Humphrey, W. S. *Managing the Software Process*. Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201180952.

[15] IBM. *What is ETL?* [online]. IBM Cloud Education, 2020. [cit. 2023/02/12]. Dostupné z: https://www.ibm.com/cloud/learn/etl.

[16] Ilieş, E. – Crişan, E. – Mureşan, I. N. Best Practices in Project Management. *Review of International Comparative Management*. March 2010, 11, s. 9. Dostupné z: http://www.rmci.ase.ro/no11vol1/Vol11_No1_Article4.pdf.

[17] IncWorx. *Power Platform ALM: Application Lifecycle Management* [online]. IncWorx. [cit. 2023/03/20]. Dostupné z: https://www.incworx.com/blog/power-platform-alm.

[18] KUMAR, B. A. *Evaluation of Fiji National University Campus Information Systems.* [online]. International Journal of Advanced Research in Computer Science, 2011. [cit. 2023/02/10].

[19] L. Bass, I. W. – Zhu, L. *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015. ISBN 978-0134049847.

[20] Laplante, P. – Neill, C. *Antipatterns: Identification, Refactoring, and Management*. Auerbach Publications, 2006. ISBN 0849329949.

[21] ManagementMania. *Projekt* [online]. ManagementMania.com, 2015. [cit. 2023/03/20]. Dostupné z: https://managementmania.com/cs/projekt.

[22] Niinimäki, T. et al. Reflecting the choice and usage of communication tools in global software development projects with media synchronicity theory. *Journal of Software: Evolution and Process*. 2012, 24, 6, s. 677–692. doi: https://doi.org/10.1002/smr.566. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.566.

[23] Potifob. *Co je a co není PRINCE2, PRINCE2 Agile a Scrum?* [online]. POTIFOB. [cit. 2023/02/20]. Dostupné z: https://potifob.cz/Co-je-a-co-neni-PRINCE2-PRINCE2-Agile-a-Scrum.

[24] Pícha, P. *Software Process Anti-pattern Detector (SPADe)* [online]. Ing. Petr Pícha. [cit. 2023/04/20]. Dostupné z: https://github.com/ReliSA/SPADe.

[25] Pícha, P. *Detecting software development process patterns in project data. Technická zpráva DCSE/TR-2019-04*. Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2019. Dostupné z: https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2019/Rigo_P%c3%adcha_2019_4.pdf.

[26] Sahoo, J. *What is a CI/CD Pipeline ? Continuous Integration and Continuous Delivery Explained.* [online]. OpsMx, 2021. [cit. 2023/04/21]. Dostupné z: https://www.opsmx.com/blog/what-is-a-ci-cd-pipeline/.

[27] Team, A. C. *Waterfall Methodology: A Complete Guide* [online]. Adobe, 2022. [cit. 2023/03/15]. Dostupné z: https://business.adobe.com/blog/basics/waterfall.

[28] Hauge – Ayala, C. – Conradi, R. Adoption of open source software in software-intensive organizations – A systematic literature review. *Information and Software Technology*. 2010, 52, 11, s. 1133–1154. ISSN 0950-5849. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0950584910000972.

Přehled zkratek

- **OSS** – Open-source software
- **GPL** – General Public License
- **XP** – Extreme Programming
- **ALM** – Application Lifecycle Management
- **OLAP** – Online Analytical Processing
- **VCS** – Version Control System
- **CD** – Continuous Deployment
- **CI** – Continuous Integration
- **SCM** – Supply Chain Management
- **SVN** – Subversion
- **CSV** – Comma-separated Values
- **TXT** – Text File
- **ETL** – Extract-Transform-Load
- **GUI** – Graphical User Interface
- **API** – Application Programming Interface
- **REST** – Representational State Transfer
- **XML** – Extensible Markup Language
- **JSON** – JavaScript Object Notation
- **SPADe** – Software Process Anti-pattern Detector
- **URL** – Uniform Resource Locator

Seznam obrázků

Seznam tabulek

A Obsah ZIP souboru

Adresářová struktura a obsah ZIP souboru je popsán v následujícím seznamu:

- **Text_prace** – Složka obsahující text diplomové práce, obrázky a zdrojové kódy textu v LaTeXu.
- **Poster** – Adresář obsahující výsledný poster ve formátu pdf a pub.
- **Readme.txt** – Textový soubor popisující obsah zip souboru a adresářovou strukturu.
- **Vysledky** – Složka s ukázkou výstupů z aplikace.

V souboru se nachází složka **Aplikace_a_knihovny** a v ní podsložka **ProjectSearcher**, která obsahuje:

- **src** - Složka obsahující zdrojové kódy aplikace a to inicializační třídu AppInit.java a složku `controller`, která obsahuje zbylé třídy. Podsložka `lib` obsahuje knihovny.
- **out** - Složka obsahující dvě podsložky, v jedné z nich (artifacts) se nachází výstupní BATCH a JAR soubor.
- **ProjectsFound** - Tato složka slouží pro ukládání souborů s podrobným popisem projektů.
- **Properties** - Tato složka slouží pro ukládání properties souborů.
- **javadoc** - Tato složka slouží pro Java dokumentaci.
- **Token** - Textový soubor s autentizačním tokenem pro GitHub.
- **Knihovny** – Adresář obsahující knihovnu `json-20220924.jar` a soubor s JavaFX SDK 19.0.2.1.

B Příručka nasazení

V této příručce je popsán postup nasazení aplikace pro vyhledávání souborů.

1. Aplikace je implementovaná v Javě. Je nutné mít na svém počítači příslušnou verzi Javy a to `"19.0.1"`

2. Java Runtime Environment (JRE) nebo Java Development Kit (JDK): JRE umožňuje spouštět Java aplikace, zatímco JDK navíc obsahuje nástroje pro vývoj Java aplikací. JDK zahrnuje JRE, takže pokud máte JDK, JRE už nepotřebujete. JDK 19 je dostupné v přiloženém adresáři `Knihovny` a nebo na (`https://www.oracle.com/java/technologies/javase/jdk19-archive-downloads.html`.).

3. Nastavte staženou verzi javy jako výchozí. Například vytvořením nové systémové proměnné `JAVA_HOME` s příslušnou cestou k uloženému souboru z bodu 2. Poté upravte systémovou proměnnou `Path` tak, že do ní přidáte nový záznam s cestou z bodu 2.

4. Stáhněte si JavaFX SDK např. ze stránky `https://gluonhq.com/products/javafx/`/. Vyberte verzi JavaFX, která odpovídá Vaší verzi Javy (19). Rozbalte stažený archiv a umístěte JavaFX SDK do vhodné složky na Vašem počítači

B.1 JAR soubor

JAR (Java ARchive) je formát souboru používaný pro distribuci a spouštění Java aplikací nebo knihoven. Spuštění aplikace pomocí JAR souboru by mělo být za splnění podmínek správných verzí a správných cest funkční. Použité knihovny jsou již součástí JAR souboru.

B.1.1 Jak spustit JAR soubor:

BATCH soubor "launcher.bat"

V přiloženém ZIP souboru se nachází BATCH soubor, který stačí upravit, spustit a JAR soubor a aplikace se spustí.

Cesta k souboru:

`ProjectSearcher\out\_artifacts_ProjectSearcher_jar\launcher.bat`

Obsah souboru `launcher.bat`:

```
"C:\Program Files\Java\jdk-19\bin\java.exe"
--module-path "C:/Program Files/Java/javafx-sdk-19.0.2.1/lib"
--add-modules javafx.fxml,javafx.controls,javafx.graphics -jar
C:\<cesta_ke_složce>\ProjectSearcher\ProjectSearcher
\out\artifacts\ProjectSearcher_jar\ProjectSearcher.jar
```

V souboru upravte cestu ke složce `ProjectSearcher` a Vaší cestu k JavaFX SDK, pokud se liší od výše uvedeného obsahu. Poté soubor uložte a spusťte (ikonou souboru nebo příkazem v terminálu: "launcher.bat").

POZNÁMKA: *"–module-path"* lze nastavit i jako systémovou proměnnou:

1. Otevřete příkazový řádek (Command Prompt) nebo terminál (Terminal) v systému Windows.
2. Přejděte do složky, která obsahuje zdrojový kód vaší JavaFX aplikace, pomocí příkazu cd:

   ```
   cd <cesta_ke_složce>
   ```

3. Nastavte proměnnou prostředí PATH_TO_FX na cestu k JavaFX SDK lib složce. Například:

   ```
   set PATH_TO_FX="C:\<cesta_ke_složce>\javafx-sdk\lib"
   ```

4. V BATCH souboru pak stačí *"–module-path"* nahradit za

   ```
   --module-path %PATH_TO_FX%
   ```

B.2 Kompilace programu v terminálu

1. Otevřete příkazový řádek (Command Prompt) nebo terminál (Terminal) v systému Windows[1].

2. Přejděte do složky, která obsahuje extrahovanou složku s projektem `cd`:

```
cd <cesta ke složce>/ProjectSearcher/src
```

3. Spusťte kompilační příkaz:

```
javac --module-path "C:/Program Files/Java/javafx-sdk-
19.0.2.1/lib" --add-modules javafx.fxml,javafx.controls,
javafx.graphics -cp".;
<cesta_ke_složce>\ProjectSearcher\src\lib\json-20220924.jar"
controller/*.java AppInit.java
```

4. Spusťtě aplikaci:

```
java --module-path "C:/Program Files/Java/javafx-sdk-
19.0.2.1/lib" --add-modules javafx.fxml,javafx.controls,
javafx.graphics -cp".;
<cesta_ke_složce>\ProjectSearcher\src\lib\json-20220924.jar"
AppInit.java
```

Pokud se objevily problémy, ujistěte se správností verzí Javy a správností cest. Popřípadě využijte BATCH soubor nebo vývojové prostředí Intelij[2], Eclipse[3] a podobné.

[1] "-module-path "C:/Program Files/Java/javafx-sdk -19.0.2.1/lib"- tato cesta se může lišit na základě místa uložení stažené JavaFX SDK.

[2] https://www.jetbrains.com/idea/

[3] https://www.eclipse.org/

C Uživatelská příručka

Aplikace je určena pro přehledné a rychlé vyhledávání projektů z webhostingových nástrojů.

C.1 Výběr zdroje dat

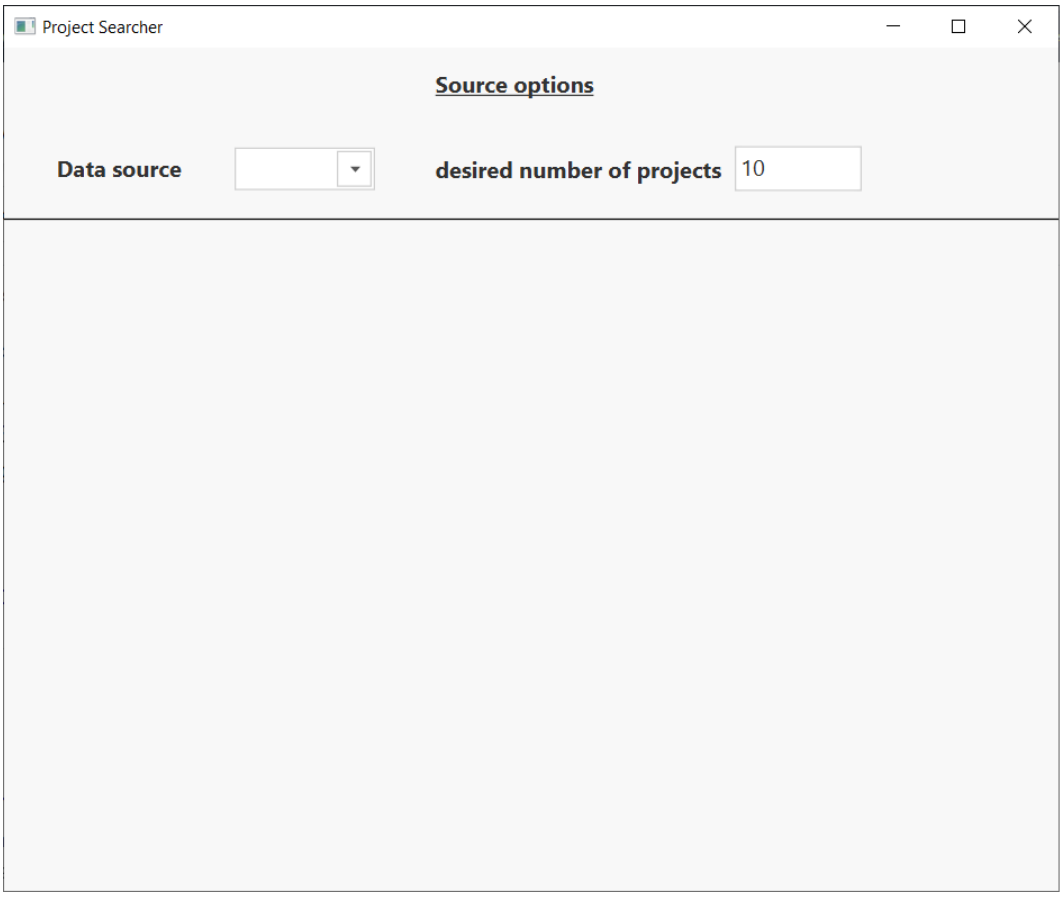
Po spuštění aplikace se otevře hlavní formulář s možnostmi nastavení zdroje. Uživatel si vybere zdroj dat (implementován GitHub) a podle vybraného zdroje se zobrazí výběr charakteristik daného zdroje. V textovém poli uživatel může zadat požadované množství hledaných projektů. Toto pole má například pro GitHub omezení maximálně 1000 projektů a je ošetřeno tak, aby uživatel nemohl zadat větší množství. Uživatel by měl brát v úvahu časovou náročnost vyhledávání, jelikož vyhledávání většího množství projektů může zabrat i několik desítek minut. Není od věci tedy vyhledávání spouštět na menším množství projektů.

Token

Pro autentizaci GitHub API je potřeba si vygenerovat na svém GitHub účtu token. Je tedy potřeba použít stávající účet nebo si zřídit účet nový (`https://github.com/join`). Následně vygenerovat token: `setting -> Developer Settings -> Personal Acces Token -> generate new token`. Popřípadě lze samozřejmě využít i již vytvořený jiný token a negenerovat nový.

Pro účely kontroly této práce je v ZIP souboru vložen **textový soubor `Token`, ve kterém se nachází funkční token**[1] autora této práce, který lze využít. Není tak tedy nutné používat vlastní token.

[1] Tento ukázkový token přestane z důvodu bezpečnosti po kontrole DP platit a nebude nadále funkční.

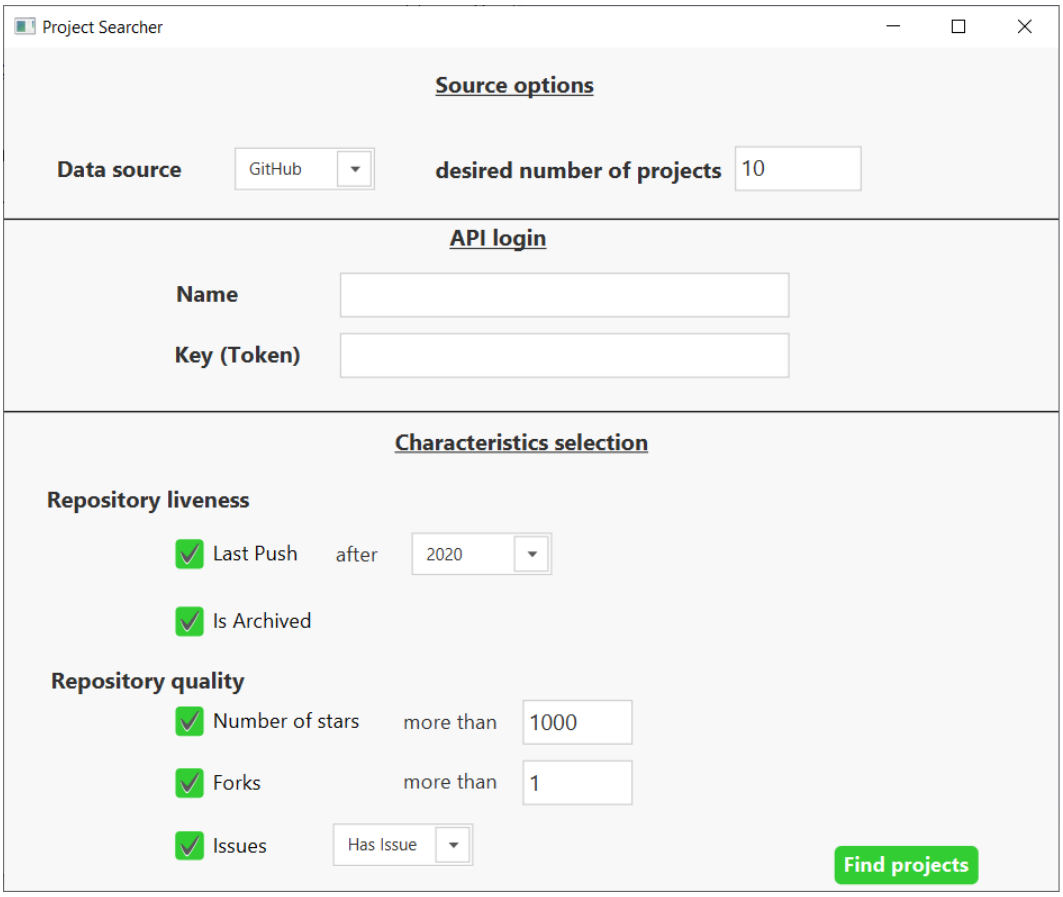


Obrázek C.1: Výchozí obrazovka aplikace

C.2 Nastavení charakteristik

Po zvolení zdroje se zobrazí vybrané charakteristiky projektů nacházejících se na tomto zdroji dat. Uživatel pomocí check-boxů a k nim přiděleným textovým polím vybere charakteristiky, podle kterých se budou projekty vyhledávat.

Obsahuje-li zdroj dat možnosti pro autentizaci připojení, rovněž zde bude zobrazena. V případě GitHubu se jedná o uživatelské jméno a API token, přičemž vyplnění API tokenu je podmíněno pro možnost spuštění vyhledávání.

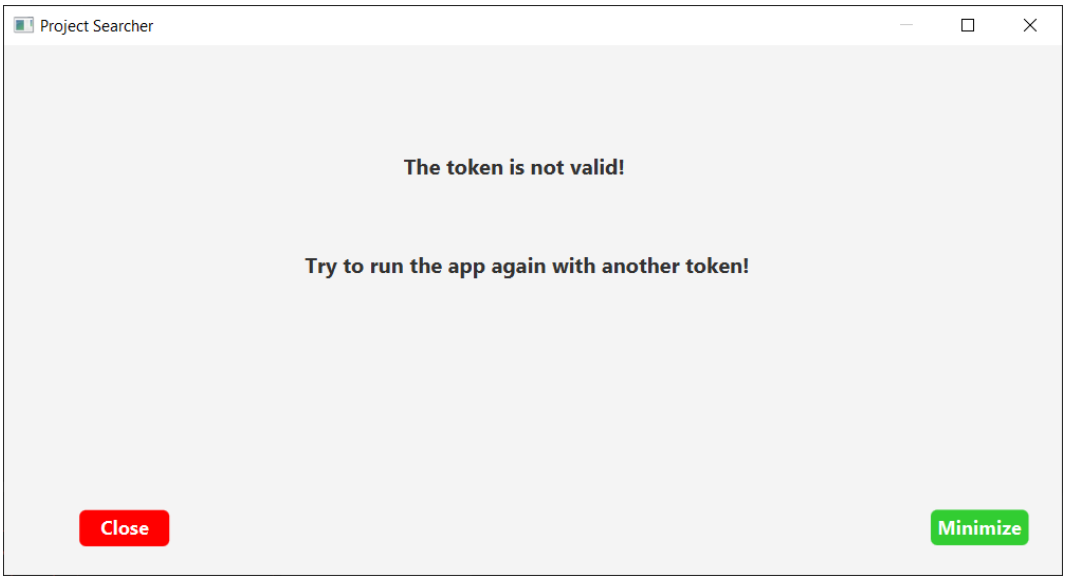


Obrázek C.2: Obrazovka se zvoleným zdrojem dat "GitHub"

C.3 Průběh hledání

Má-li uživatel vybrané charakteristiky (není podmínkou), může začít vyhledávání pomocí tlačítka "Find Projects". Po spuštění se zobrazí informační okno, pokud uživatel nevyplnil povinné údaje (API token), bude vyzván, aby tak učinil. Pokud token vyplnil, zobrazí se informační okno se shrnutím jména a tokenu, kliknutím na tlačítko "Ok" uživatel spustí vyhledávání.

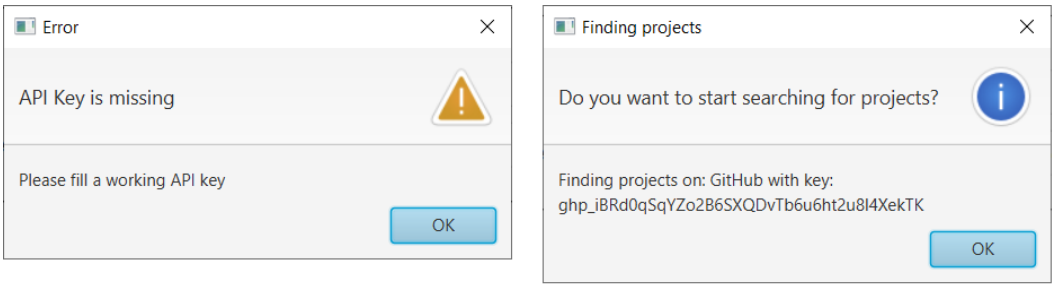
Při vyhledávání se minimalizuje hlavní formulář a otevře se informační formulář, který uživatele informuje o průběhu vyhledávání.



Obrázek C.3: Informace o nevalidním tokenu

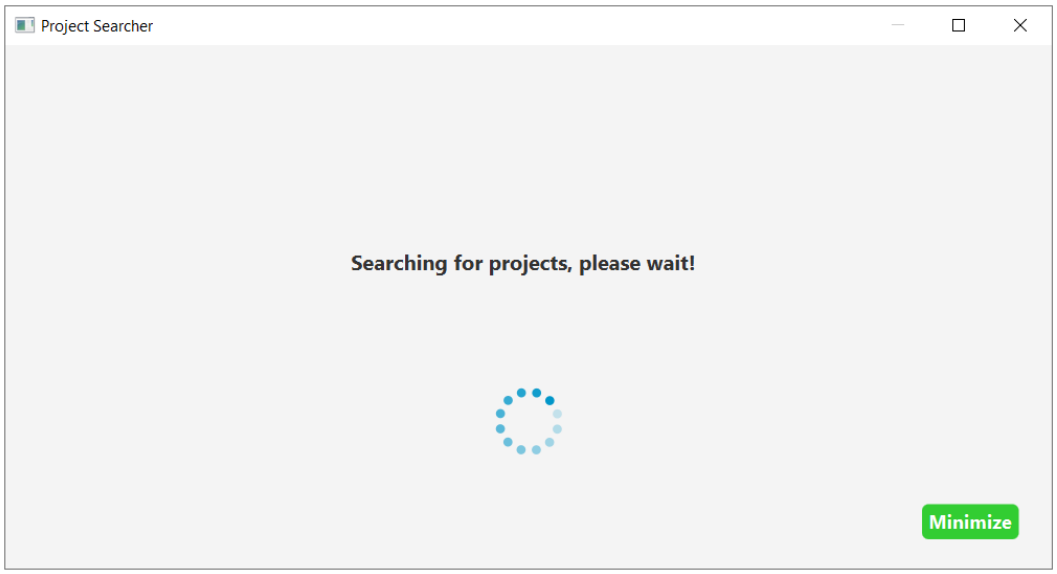
C.4 Výsledky

Výsledků průběhu vyhledávání je několik. V případě chyby, která nastala uživatelským zásahem (nevalidní API token nebo kombinace charakteristik, pro které neexistuje projekt) bude uživatel informačním oknem o chybě informován a při uzavření tohoto okna může vyhledávání spustit znovu.



Obrázek C.4: Oba možné scénáře informačních oken po spuštění vyhledávání

V případě úspěšného vyhledání projektů bude informován o počtu nalezených projektů (ten se na základě výběru charakteristik může lišit od požadovaného počtu projektů, nicméně projektů odpovídajícím zvoleným charakteristikám nemusí existovat požadované množství) a o nově vytvořených souborech. Struktura vytvořených souborů se nachází v příloze D.



Obrázek C.5: Informační formulář s indikátorem běhu programu

D Výstupní soubory

Při úspěšném vyhledání souborů budou vytvořeny dva druhy souborů. Jedním z nich je properties soubor, který následně může sloužit jako zdroj pro nástroj SPADe. Druhými jsou cvs/txt soubory. Ty obsahují podrobné informace o projektech. Na snímku obrazovky aplikace D.1 lze vidět okno s výsledkem hledání.

Obrázek D.1: Informační formulář s výsledkem hledání

V následující tabulce lze vidět strukturu vytvořených složek a souborů. V aplikaci je ošetřena situace, že pokud složky `ProjectsFound` a `Properties` neexistují, budou vytvořeny. Do nich se následně ukládají jednotlivé soubory s projekty.

Složka/Soubor	*Obsah*
`*/ProjectsFound/Projects_xxx`	podrobné informace o několika projektech
`*/Properties/2023-xxx`	properties soubour s jedním projektem

Tabulka D.1: Výchozí struktura vytvořených souborů

D.1 Properties soubor

Properties soubor obsahuje informace o jednom projektu. Název souboru je tvořen pouze názvem projektu.

Zde je obsah souboru `node.properties` [1]:

```
privateKey=
url=https://github.com/nodejs/node.git
tool=GITHUB
repo=https://github.com/nodejs/node.git
```

[1] Jedná se o OS projekt nodejs/node dostupný na https://github.com/nodejs/node.git

D.2 Textový a CSV soubor

TXT a CSV soubory obsahují podrobné informace o několika vyhledaných projektech (vždy maximálně o 100 projektech). Funkce a struktura je podrobně popsána v podsekci 5.3.2.

Zde je ukázka obsahu souboru `Projects_2023-05-06_16-19-46.txt` [2], který obsahuje dva projekty:

```
rcrowley/go-tigertonic,https://github.com/rcrowley/go-tigertonic,
267,A Go framework for building JSON web services inspired by
Dropwizard,Other,28,25,998,2018-07-24T09:26:32Z,19

RisingStack/node-style-guide,https://github.com/RisingStack/
node-style-guide,58,A mostly reasonable approach to JavaScript -
how we write Node.js at RisingStack,MIT License,11,-5,993,
2019-10-17T20:19:43Z,5
```

Jak je z názvu souboru patrné, obsahuje časovou značku vytvoření souboru. Obsah CSV souboru se od textového neliší.

[2] V ukázce obsahu informace o jednotlivých projektech zabírají tři a čtyři řádky vzhledem k rozměru stránky, nicméně v souborech jsou vždy na jednom řádku