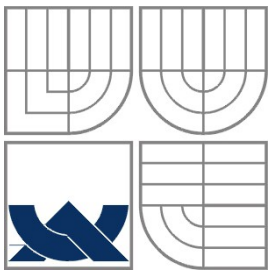
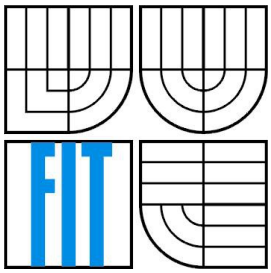


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁSTROJ PRO VYHODNOCENÍ KVALITY PŘÍSPĚVKŮ PROJEKTŮ OPEN-SOURCE

A TOOL FOR EVAULATION OF QUALITY OF CONTRIBUTION TO OPEN-SOURCE PROJECTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RADIM ŠPIGEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2014

Abstrakt

Tato práce se zabývá zjištěním kvality příspěvků při vývoji open-source projektů, napsaných v jazyce Python. Zjištění této kvality je prováděno pomocí nástrojů pro statickou analýzu a hypotetickým algoritmem založeným na době, jakou daný příspěvek vydržel nezměněn. Taktéž se tato práce věnuje verzovacím systémům, zvláště pak distribuovanému systému Git. Dále jsou zde rozebrány základní softwarové metriky a vizualizační knihovna D3.

Abstract

The aim of this work is an examination of contributions quality in development of open-source projects written in programming language Python. Detection of this quality is being probed by static analysis tools and hypotethical algorithm which is based on duration of current contribution changelessness. In this work there is a basic description of revision control systems and distributed system Git. There is also a mention about basic software metrics and D3 library.

Klíčová slova

Git, D3, svobodný systém, Python, JavaScript, metriky kvality softwaru, verzovací systémy, Pylint

Keywords

Git, D3, open-source, Python, JavaScript, quality software metrics, version control systems, Pylint

Citace

Radim Špigel: Nástroj pro vyhodnocení kvality příspěvků projektů open-source, bakalářská práce, Brno, FIT VUT v Brně, 2014

Nástroj pro vyhodnocení kvality příspěvků projektů open-source

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Další informace mi poskytl pan Chris Ward. V práci jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení (Radim Špigel)
Datum (19. května 2014)

Poděkování

Chtěl bych poděkovat panu Ing. Aleši Smrčkovi, Ph.D. za odborné vedení mé práce. Dále pak panu Chrisi Wardovi z firmy Red Hat za zadání, konzultace a poskytnuté informace.

Obsah

1. Úvod

Tato bakalářská práce se zabývá vyhodnocováním kvality příspěvků pro open-source[1] projekty napsané v jazyce Python. Hlavním cílem této práce je navrhnout algoritmus, který zjistí, jak dlouho vydržely dané příspěvky v projektu a zda toto kritérium má vliv na kvalitu autora z hlediska vývoje.

Vzhledem k tomu, že dnes je čím dál více vývojářů a lidí zajímajících se o svobodný software, je tato práce zaměřená hlavně pro vedoucí těchto projektů. Dále je také tato práce vhodná pro vývojáře, které zajímá hypotetická kvalita jejich příspěvků založená na ohodnocení každého příspěvku pomocí statického analyzátoru Python kódu Pylint a nástroje pro softwarové metriky Radon.

Jednou z dalších částí této práce je vizualizace dat, jelikož je pro porovnání výsledků mnohem přínosnější než-li surová data získaná během měření. Díky vizuálnímu zobrazení je vidět zda kód, v našem případě příspěvek, byl kvalitní, či ne. Podle toho je možné odhadnout kvalitu daného autora.

Výsledky této práce by měly zvýraznit jak se kvalita projektů měnila během vývoje. V této práci se zaměřím na dva projekty. Prvním projektem pro klasifikaci kvality je tento nástroj, abych odhalil poklesy kvality při jeho vývoji, a také chyby, které mohly vzniknout během vývoje. Druhým projektem pro ohodnocení je projekt zvaný Metrique, který je také psaný v Pythonu a slouží pro extrakci dat, jejich analýzu a zobrazení výsledků [13].

V následujících kapitolách budete seznámení se základními metrikami kvality softwaru, a testováním softwaru. Oba tyto pojmy mají velikou spojitost s metrikami kvality softwaru. Poté se statickou analýzou kódu, která je stěžejním analytickým kritériem v tomto nástroji. Hlavním nástrojem statické analýzy použitým v tomto projektu, je nástroj pro analýzu kódu psaného v jazyce Python, nástroj Pylint.

Dále tato práce obsahuje krátké seznámení s verzovacími systémy, hlavně se systémem Git. V neposlední řadě je zde zmínka o vizualizační knihovně D3, která v dnešní době zaujímá důležité místo mezi vizualizačními nástroji pro zobrazení ve webovém rozhraní.

Nejvíce bych zdůraznil kapitolu, která se věnuje samotnému návrhu a implementaci algoritmů pro ohodnocení příspěvků, vzhledem k době jejich trvání, v rámci vývoje open-source projektů psaných v programovacím jazyce Python. Na závěr jsou zde uvedeny výsledky z testů mého nástroje, které byly provedeny nad mým projektem a projektem Metrique a ohodnocení, zda tyto projekty byly úspěšné či ne. Jsou zde také popsány návrhy na rozšíření a optimalizování tohoto nástroje pro ohodnocení kvality příspěvků.

1 Open-source (Otevřený software) je počítačový software s volně dostupným zdrojovým kódem.

2. Metriky kvality softwaru

V této kapitole čtenáře seznámím se základními podmínkami pro kvalitu softwaru. Kvalita softwaru [14] je definována normou ISO 9126[2]. Kvalitní software musí fungovat v souladu s požadavky zákazníka a jeho vývoj musí:

- být prováděn dokumentovanými metodami
- splňovat platné zásady pro tvorbu softwaru

Kvalita by měla být sledována již od počátku zahájení práce na projektu. V okamžiku testování je většinou pozdě. Je odvozena z měření, které je dvojího typu - přímé a nepřímé.

Dále jsou různé interpretace naměřených hodnot. Kvalita softwaru je podmnožinou Softwarových metrik, které zahrnují různé metriky pro daný projekt/výrobek. Pro ohodnocení kvality kódu se používá **statická** a **dynamická analýza** neboli **statické a dynamické testování softwaru.**

Hlavní důvody pro používání metrik softwaru jsou zjištění kvality projektu, kvality vývojářů a odhad doby vývoje dalších projektů.

Jednou z hlavních vlastností systému pro měření kvality jsou:

- jasně definovány cíle měření
- data musí být užitečná

Další vlastností je otevřenost a modifikovatelnost. Vzhledem k tomu, že potřeby vedoucích projektů či managementu se mění, musí se měnit i metriky. Samotné měření je odděleno od vyhodnocení. Toto měření by mělo být jako podpora pro práci a nebýt hrozbou nebo demotivací pro vývojáře.

Přenosnost samotných softwarových metrik je malá, protože má velký rozptyl a tyto metriky závisí na typu projektů, dosažitelné technologii a kvalitě programátorů. Jelikož programátoři jsou schopni hodně ovlivnit určité metriky.

Softwarové metriky jsou interní, tyto metriky jsou známy jen během vývoje na základě sledovaných procesů. Externí metriky se dají zjistit z hotového produktu, další metriky jsou implicitní či explicitní.

Pro samotné ohodnocení kvality softwarových produktů jsou důležitá data (kód, výstupy z programu atd.). Je několik kategorií kvality dat. Například vnitřní kvalita, dostupnost, kontextuálnost a reprezentační kvalita. Kvalitu dat je nutno měřit či odhadovat. Můžeme ji zlepšovat například tím, že odstraníme okrajová a opakovaná data. Další způsob jak zlepšit kvalitu dat je doplnit chybějící data a parametry.

Datové typy metrik mají příslušnost ke třídě. Hlavní operací je rovnost, což je výskyt elementu v množině. Hodnocení je obvykle v číselné formě, prvek uspořádané množiny pro níž je použita operace porovnání. Pro interval je použita lineární transformace.

Posledním typem, který zmíním ve spojitosti s daty, je číselná metrika. To je např. délka programu, doba řešení atd. Číselné metriky jsou v podstatě všechny smysluplné operace pro reálná čísla. V normě ISO 9126 jsou definovány stovky metrik. Tyto metriky používají většinou velké firmy.

Samotná data před průchodem přes hodnotící systém, který je založen na hodnoticích metrikách, musí být upravena do správného tvaru. Jde o postup vylučování **okrajových dat,** která jsou nesprávná, to znamená úmyslně změněná, chybně zanesená nebo data v nesprávném formátu. Také je potřeba vyloučit **duplicitní záznamy**. Je důležité sjednotit formát dat a doplnit chybějící

2 ISO 9126 je mezinárodní standard pro ohodnocení softwarové kvality. Nově je tento standard označen ISO/IEC 25010:2011. Je možné ho nalézt zde[23].

údaje, aby bylo možné soubor rozumně zobrazovat. Například vylučování duplicitních údajů je při nesjednoceném formátu velice komplikované. Nad daty se samozřejmě provádí další nejrůznější operace např. **Parciální replikace, sjednocení metrik obecně** atd. [16].
Nejčastěji používané atributy pro kvalitu dat jsou **Relevantnost, Přesnost, Včasnost, Dostupnost, Porovnatelnost, Koherence a Úplnost.**

- **Relevance** je míra, do jaké data splňují svůj účel, pro který jsou používaná. Týkají se daného problému. Relevantnost dat také závisí na frekvenci zjišťování.
- **Přesnost (Accurancy)** je míra přesnosti dat. Spadá sem například směrodatná odchylka.
- **Včasnost (Timeless)** je doba, za kterou lze data aktualizovat tzn. jak jsou data aktuální.
- **Dostupnost (Accessibility) -** jakým způsobem jsou existující data dostupná.
- **Porovnatelnost (Comparability)** je metrikou hodnotící možnost samotného porovnávání a spojování dat z různých zdrojů.
- **Koherence (Coherence)** vyjadřuje do jaké míry byla data vytvořena podle kompatibilních pravidel.
- **Úplnost (Completeness)** udává, jaká část potencionálních dat je zachycena v databázi nebo zda výběr těchto dat pokrývá rovnoměrně celý výběrový prostor.

Metriky kvality je téměř nezbytné zahrnout do metadat[3].

2.1. Sofwarové metriky

V této podkapitole seznámím čtenáře hlavně se základními sofwarovými metrikami[4], které jsou pro vývoje a ověření kvality softwaru důležité. Cílem je získat objektivní, reprodukovatelné a kvantifikovatelné měření. Toto měření může určit odhad nákladů, testování zabezpečování jakosti softwaru, ladění, optimalizaci výkonu softwaru a optimální přiřazení programátorů nebo testerů pro dané úkoly [14].

- **LLOC** – počet logických řádků kódu. Každý řádek obsahuje přesně jeden výraz.
- **LOC –** počet všech řádků v daném souboru. Tato metrika zahrnuje i prázdné řádky kódu, komentáře.

  Příklad:

  ```
  for (i = 0; i < 100; i += 1) printf("Hello World"); /* Komentář */
  ```

  Tento kód obsahuje: LOC = 1, LLOC = 2 (for cyklus a printf funkce),

  počet řádků komentářů = 1

3 Metadata jsou strukturovaná data o datech. Mohou sloužit k snadnějšímu vyhledávání či k zpřesnění výsledků.

4 Metrika = měřitelný údaj

- **Halsteadova metrika -** tato metrika je založena na předpokladu, že všechny programy se skládají z konečného počtu programových jednotek. Tyto jednotky jsou nazývány tokeny, které jsou rozeznatelné v syntaktické fázi překladače.
  n1 = počet unikátních nebo rozdílných operátoru v implementaci
  n2 = počet unikátních nebo rozdílných operandů v implementaci
  N1 = celkový počet všech unikátních nebo rozdílných operátorů použitých v implementaci
  N2 = celkový počet všech unikátních nebo rozdílných operandů použitých v implementaci

  Příklad **operátorů**: „+“, „*“ , if, for, atd. Příklad **operandů**: i, 2, „string“, atd.

- **McCabe -** počet podmíněných příkazů (if), počet cyklů (for, while) a počet přepínačů (case) v programu. Metrika **McCabe** je dobrým indikátorem složitosti programu. Jiný název pro tuto metriku je cyklomatická složitost programu.
  McCabe = Počet hran – Počet uzlů + 1

- **Prod** je metrika produktivit, počet jednotek vytvořených za člověko-měsíc

- **Team** je velikost týmu v čase. Tato velikost je měřena od začátku práce. Umožňuje zpřesnit rozsah pracnosti a doby řešení vývoje.

- **Team_mean** je průměrná velikost týmu.

- **Defect** je metrika udávající počet míst v programech, které bylo nutno opravit. Je možné udávat defekt na 1000 řádků nebo udávat defekt po etapách.

- **Change** udává počet změněných míst v souboru v určitém časovém intervalu.

- **MTBF** (Mean Time Between Failures) je střední doba mezi poruchami v určitém časovém intervalu.

- A další...

Pro ohodnocení některých z těchto zmíněných softwarových metrik jsem použil nástroj **Radon.**

2.1.1 Nástroj pro hodnocení softwarových metrik Radon

Radon je nástroj psaný v jazyce Python. Tento nástroj počítá různé metriky kódu jako jsou McCabe metrika, Halstead metrika, Index udržitelnosti, počet řádků kódu, počet řádků komentářů a prázdné řádky (viz. Sofwarové metriky).

Index udržitelnosti je metrika, která ohodnotí jak je zdrojový kód udržitelný. Pro tento výpočet se používají jak McCabe metrika, tak počet řádků kódu tak i Halsteadova metrika.
V tomto nástroji se využívá vzorec vypočítaný z derivátu SEI (Software Engineering Institute)[5] a derivátu užívaném ve Visual Studiu [3].

5 http://www.sei.cmu.edu/

V = Halstead metrika
G = McCabe metrika
LOC = počet řádků kódu
CM = počet řádků komentářů

Vzorec používaný v Radonu:
MI = MAX (0, (171 - 5,2 * log2(V) - 0,23 * G - 16,2 * log2(LOC) + 50 * sin($\sqrt{2,4 * CM}$))

2.2. Stručný přehled o testování softwaru

Testování softwaru slouží k empirickému výzkumu kvality [24]. Samotné testování často nahrazuje zajišťování kvality. Musí se určit cíle testování a také co vše se má testovat. Vybírají se testy a připravují nástroje, které jsou pro zadaný úkol nutné. Proces testování se provádí ve více iteracích.

Výsledky testů by měly být v ideálním případě: prošel nebo neprošel. Testováním softwaru není možné zajistit naprostou bezchybnost softwaru, protože nelze nasimulovat nekonečné množství vstupních hodnot a zkontrolovat nekonečné množství výstupních hodnot. Testováním lze velice efektivně poukázat na to, zda jsou přítomny chyby, ale nelze prokázat, že je software bezchybný.

- **Statické testování** nevyžaduje, aby program běžel. Toto testování je možné začít před vytvořením prototypu.

- **Dynamické testování,** pro toto testování je nutná existence spustitelné verze softwaru a probíhá hlavně poskytováním různých vstupů a posuzování výstupů daného programu, který má být testován.

- **Černá skříňka -** při tomto testování se zaměřujeme na vstupy a výstupy, aniž bychom znaly jak je tento systém implementován. Smyslem toho je analyzovat chování softwaru k očekávaným vlastnostem tak, jak ho vidí uživatel.

- **Bílá skříňka -** při tomto testování má tester k dispozici zdrojové kódy daného softwaru. To znamená, že je možné sledovat jak výstupy, tak i to co se děje v programu. Díky tomu umožňuje testujícímu lépe odhadnout, kde hledat chybu.

2.2.1 Dynamická analýza

Dynamická analýza je prováděna při spuštění a běhu programu na virtuálním či skutečném počítači. Pro správný a efektivní průběh dynamické analýzy, je nutné, aby program byl spuštěn s dostatečným počtem vstupů, abychom detekovali nesprávné či zajímavé chování.
Nástroje pro dynamickou analýzu: Valgrind, DynInst, Dmalloc atd.

2.2.2 Statická analýza

Statická analýza [2] je v podstatě strukturní analýza kódu bez spuštění daného programu, který je analyzován. Tato analýza je prováděna nad určitou verzí kódu. Často je spojována s automatizovaným nástrojem. Jejím hlavním účelem je zvýšení kvality a bezpečnosti softwaru.

Velkou výhodou oproti dynamické analýze je to, že tuto analýzu je možné provádět nad projektem, kdy ještě není spustitelný. Díky tomu je možné dříve odhalit chyby v kódu. Při statické analýze je kód procházen několikrát a jsou analyzovány potenciální problémy. Aplikování analýzy ve fázi, kdy kód ještě není spustitelný, může ušetřit čas i uspořit náklady, než kdyby byl kód testován, až když je spustitelný. Jelikož náklady na opravu chyb rostou exponenciálně od doby kdy vznikly chyby, k jejím objevením.

Statická analýza je také vhodná, oproti dynamické analýze, na rozsáhlé projekty, protože dynamická analýza je drahá a časově náročná. Existuje spousta programů pro statickou analýzu kódu. Např. pro programovací jazyk C/C++: Cpplint, Cppcheck, Lint, GrammaTech CodeSonar, pro Javu: GrammaTech CodeSonar, Squale, FindBugs, pro Python: Pylint atd. Více nástrojů je možné naleznout zde [21].

Jelikož tato práce se zabývá ověřováním kvality příspěvku k open-source projektů, které jsou napsány v Pythonu, bude nás zajímat hlavně nástroj Pylint.

2.2.3 Nástroj pro statickou analýzu kódu - Pylint

Pylint je nástroj pro ohodnocení kvality kódu napsaného v jazyce Python . Zaměřuje se také na odhalení chyb. Snaží se prosadit standardizaci kódu a hledá takzvaný *bad code smells*[6]. Základním stylem pro kódování je PEP 008[7]. Pylint zobrazuje statistiku o počtu chyb/varování, které byly v kódu nalezeny. Zobrazuje se také kompletní zpráva, která je rozdělena do různých kategorií, jako jsou varování, chyby a chybné psaní „standardního“ kódu. Pylint je výborný nástroj pro statickou analýzu, ale není „vševědoucí“. Spoustu zpráv, pokud to nejsou chybové zprávy, je možné ignorovat.

Pylint je možné spouštět po nainstalování příkazem *pylint mujprogram.py.* Toto spuštění vrací statistiky o chybách a varování se základním nastavením pylintu.

Pylint či jeho deriváty jsou často integrovány do vývojových prostředí, např. pro VIM[8] se tento modul jmenuje python-mode. Tento zásuvný modul spojuje pylint, pyflakes, mccabe a další nástroje pro statickou analýzu. Podrobnější informace o nástroji pylint naleznete v dokumentaci pro tento produkt [7].

6 Bad code smells - může být příznakem, že zdrojový kód indikuje hlubší problém. Nemusí to ovšem znamenat chybu.

7 PEP 008 – konvence psaní zdrojového kódu v jazyce Python

8 VIM – konzolový editor v operačních systémech Linux

3. Verzovací systémy

V této kapitole budete seznámeni s verzovacími systémy, což je velice známý pojem v softwarovém inženýrství [6][8]. Jelikož čím dál více vývojářů a firem využívá těchto systémů, shrnu počátek vývoje a základní pojmy používané s verzovacími systémy. V následujících podkapitolách popíši základní vlastnosti verzovacích systémů Git (viz. Git - distribuovaný systém), Mercurial (viz. Stručný popis Mercurial verzovacího systému) a Subversion (viz. Souhrnný popis verzovacího systému Subversion).

3.1. Historie verzovacích systémů

Jeden z prvních systémů pro správu verzí byl vyvinut v Bellových laboratořích na přelomu 70. a 80. let. Tento systém se jmenoval SCCS a následně na něj navázal GNU RCS (autor Walter F. Tichy). Později si vývojáři oblíbili systém CVS, který byl vyvinut na základě skriptů od Dicka Gruneho, samotný systém navrhl a popsal Brian Berliner v roce 1989. Později se k němu připojil Jeff Polk.

Začátkem 21. století se rozšířilo SVN tzn. Systém Subversion. Byl vyvinut firmou CollabNet Enterprise, která se rozhodla nahradit používaný systém CVS systémem SVN. CollabNet Enterprise vyvíjela tento systém ve spolupráci s Karlem Fogela, Jimem Blandym, Benem Collins-Sussmanem, Brianem Behlendrofem, Jasonem Robbinsonem a Gregem Steinem. První verze SVN byla dokončena 21. srpna 2001. V roce 2001 také vzniká decentralizovaný systém GNU arch. V roce 2005 se objevují Git, Mercurial a Bazaar.

Verzovací systém je vlastně nástroj, který je v dnešní době nepostradatelným pomocníkem snad pro každého vývojáře. Dá se říci, že tento systém slouží jako úložiště zdrojových kódů a dalších důležitých souborů. Dále slouží k uchování historie změn či verzí. Díky uchování historie je možné detekovat chyby způsobené ve změnách verzí. Je možné se vracet k předchozí funkční verzi programu a to vše díky pouhému porovnání se starší fungující verzí. Tento systém umožňuje také spolupráci vývojářů na projektech. To znamená, že odpadá vzájemné přepisování kódů a souborů na sdíleném disku nebo „otravné“ kopírování přes poštovního klienta.

Verzovací systémy také podporují práci ve „větvích“. Tato vlastnost se hodí zejména při vývoji softwaru. Staré verze systému se často nemohou zavrhnout a nepodporovat, protože jsou nasazeny a je třeba vydávat různé opravy. Díky práci ve „větvích“ můžeme v jedné větvi mít systém funkční, třeba verze 1.1. a v další větvi pracovat na nové verzi systému, verzi 1.2. atd. Dají se také slučovat větve, tzn. sloučit funkcionalitu jedné větve do druhé. Pro rozsáhlejší projekty, na kterých se podílí více lidí, jsou tyto systémy pro správu verzí prakticky nezbytností.

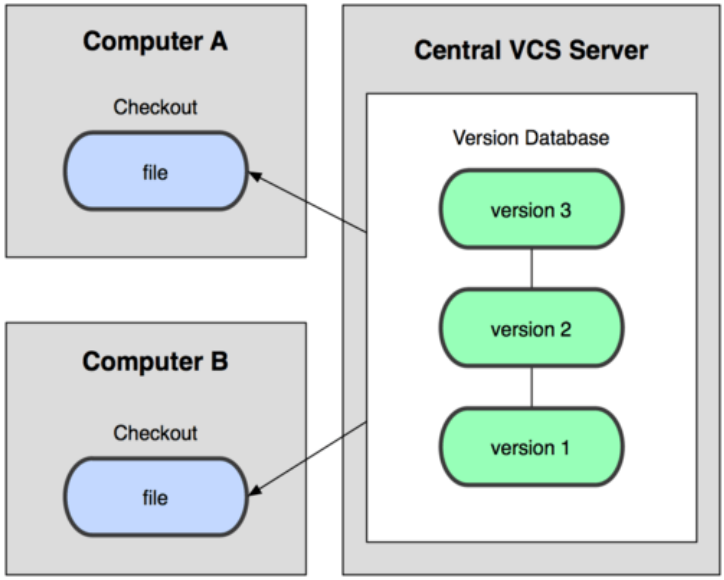
3.2. Centralizované verzovací systémy

Tyto systémy jsou používány typicky jako klient/server aplikace. To umožňuje, aby na stejném projektu pracovalo více lidí současně. Zároveň zajišťuje, aby změny provedené jakýmkoliv vývojářem byli konzistentní a mohli být distribuovány ostatním.

U **centralizovaných systémů** (viz. Ilustrace 1) máme kompletní historii změn na serveru, a na klientech pouze aktuální verzi a rozpracované soubory. Jsou zde také rozděleny role. Server je společné centrum a klienti jsou počítače jednotlivých vývojářů.

Tímto se liší od decentralizovaných systémů, kde jsou role rovnocenné. Ve všech uzlech

máme kompletní historii a je možné se vrátit k libovolné verzi.



Ilustrace 1: Centrální verzovací systém[6]

3.3. Decentralizované verzovací systém

Decentralizované neboli distribuované systémy (viz. Ilustrace 2) se liší od centralizovaných systému tím, že klienti mají svou historii uloženou, není zde centrální repozitář. Z toho plynou výhody jako je práce offline[9]. Klient/vývojář nemusí být připojen na síť, aby mohl pracovat a to vše díky tomu, že má kompletní správu na své lokální kopii, tzn. může bezpracně zálohovat a nemusí se bát, že když se vypne server ztratí data nebo nebude moci pokračovat v práci.

Další výhodou jsou vlastní větve, kde můžeme zlepšovat projekt, aniž bychom museli přepisovat starší funkční verzi. Stačí si projekt „naklonovat“ a začít vyvíjet. Tím odpadá starost o práva vývojářů, které se vztahují k centrálnímu úložišti.

Díky distribuovaným systémům je možné použít více scénářů práce. U centralizovaných systémů máme prakticky jen jeden jediný scénář práce.

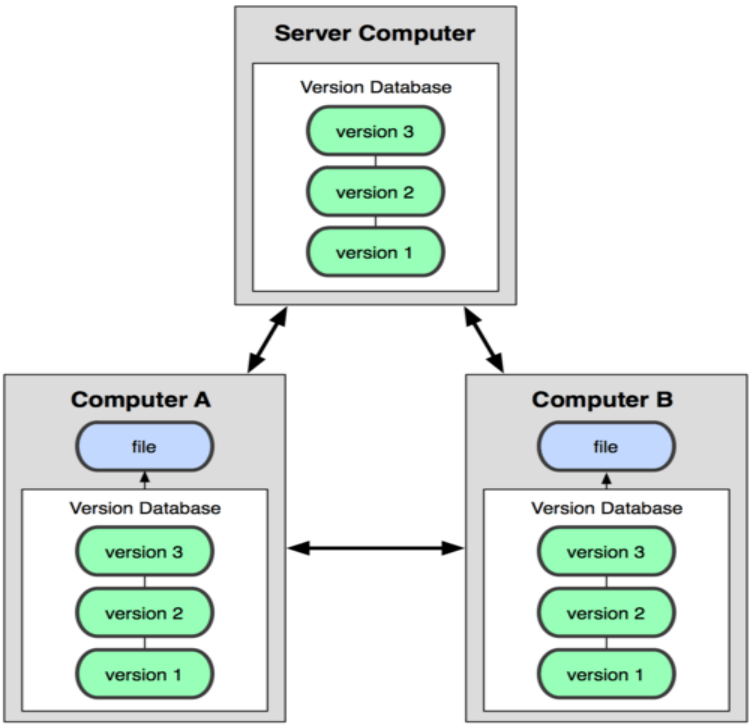
Distribuované systémy mají také pár nevýhod. Jednou z těchto nevýhod je vyšší složitost. Ten kdo byl zvyklý na centralizovaný systém, může mít problémy s tím, že se změny neukládají ihned na server po uložení změn (commit), ale až po uložení a odesláním na server (commit a následný push).

Oproti centralizovaným systémům je zde také jiné číslování verzí. To není tak přehledné, protože každý vývojář vyvíjí na svém klientu jiným způsobem, tzn. nejde zaručit stejná verze, protože každý vývojář programuje specifickým způsobem.

Další nevýhodou distribuovaných systémů je to, že nejdou soubory zamykat, jak tomu bylo u centralizovaných systémů, kdy někdo zamkl soubory a tím dal najevo, že s těmito soubory pracuje. U distribuovaných souborů to nejde - je zde systém slučování.

Tyto nevýhody nejsou nijak zásadní, ale je dobré o nich vědět.

9 Offline = bez připojení či nutné komunikace se serverem



Ilustrace 2: Distribuovaný verzovací systém [6]

Jak jsem se již zmínil, pro distribuované systémy je **více scénářů práce**. Nastíním zde pár základních scénářů. U distribuovaných systémů se nemusíme omezovat na klasické uspořádání klient-server, není to podmínkou. Vzdálené úložiště je společný server, každý vývojář k němu má přístup. Pracovní kopii má každý vývojář u sebe. Postupně každou změnu (commit) ukládá u své kopie. Tím vzniká verze, takzvaná sada změn (changeset). Tyhle změny se dějí pouze u vývojáře. Až vývojář bude chtít data nahrát na server, musí použít odeslání změn na server (push) a vše, co vývojář uložil v daném projektu, se nahraje na server.

Další možností je z daného serveru posílat (push) data na zálohovací server anebo tyto data stahovat ze záložního serveru (pull). Dále může být server veřejný. Tato výhoda spočívá v tom, že veřejnost nebude zatěžovat server, který využívají vývojáři.

Některý z vývojářů také může zpřístupnit svou práci na nějaký veřejný server a ostatní si můžou stahovat změny (pull), anebo mu pomoci s vývojem a přispívat (push) bez nutnosti přístupu k centrálnímu úložišti. Uspořádání je více druhů a je jen na vývojářích, jaké uspořádání si zvolí či si vymyslí svoje vlastní.

Jedny z nejpoužívanějších distribuovaných verzovacích systémů jsou **Mercurial, Git** a **Bazaar**.

3.4. Základní pojmy pro verzovací systémy

- **Repozitoř (Repository)** neboli centrální úložiště nám umožňuje organizovat a spravovat verze projektu. Je uložena fyzicky v souborovém systému nebo na serveru. Správa je prováděna klientskými nástroji.

- **Větev (Branch)** je určena k organizaci repozitoři. Pokud je z repozitoře „naklonována“ či vyzvednuta „větev“, vytvoří se na klientu přesně stejná adresářová struktura, jaká je v repozitoři.

- **Revize (Revision)** je pořadové číslo změny. Je určena ke sledování změn ve větvích v určitém čase. Každá změna v nějaké větvi vytvoří novou revizi. Tato revize obsahuje informace: co bylo změněno, kdo provedl změny, čas a poznámku.

- **Pracovní kopie** je kopie dat ze serveru na klientském stroji. Jsou to data, na kterých se pracuje a provádí se změny. Tyto změny se pomocí commitu posílají zpět na server do repozitoře.

- **Příspěvek (Commit)** slouží k odeslání provedených změn od posledního příspěvku do repozitoře. Příspěvek je atomický, když se provádí odeslání celé pracovní kopie. Pokud dojde k chybě během přenosu, nevytvoří se nová revize a tento commit není pro ostatní viditelný.

- **Konflikt** je stav signalizující, že stejný objekt byl právě commitován. Označuje, že tento objekt byl nahrán někým jiným, nachází se v repozitoři a je již změněn, tzn. liší se od pracovní kopie. Pokud je jeden nebo více souborů v konfliktu, nelze provézt atomický commit (commit celého pracovní kopie).

- **Changeset**, jak už název napovídá, je sada změn, které jsou poslány do repozitoře z pracovní kopie. Verzovací systém ukládá informace o provedených změnách. To znamená jen rozdíl mezi revizemi.

- **Merge** je sloučení větví. Sloučí větvě z pracovní kopie do repozitoře. Musí se určit interval pro sloučení, který je dán intervalem revizí.

- **Proces správy verzí -** při tomto popisu předpokládejme, že požadavky jdou sériově (za sebou), nikoliv paralelně. Zaprvé je nutné stáhnout projekt pomocí příkazu **checkout** do lokálního adresáře, tzn. vytvořit pracovní kopii. Dále se provádí modifikování, neboli editace souborů (přidávání, mazání). Poté se změny odešlou do repozitoře pomocí příkazu **commit**. Změny jsou viditelné pro všechny uživatele dané repozitoře. Spolu se změnami se zapisuje čas poslání, autor a komentář. Další vývojář může, pokud má staženou nejaktuálnější verzi (tu získá příkazem **update/pull**), pokračovat ve vývoji.

3.5. Souhrnný popis verzovacího systému Subversion

SVN je zkratkou pro subversion. Tento systém je pro správu a verzování zdrojových kódů a dalších souborů [25]. Tento systém je založen na principu **centrální** repozitoře. Byl vyvinut firmou CollabNet Inc. Je to vlastně náhrada za CVS, snaží se zachytit podobný styl práce a hlavně odstranit některé nedostatky CVS (kopírovaní či přesun adresářů, prostorová náročnost atd.). Samotné SVN má velice dobře popsanou anglickou dokumentaci, což je jistě výhoda. Další výhodou je, že tento systém je open-source. SVN je dostupný na OS Windows, Linux i Mac OS. Skládá se, jak už všechny centrální systémy, ze serveru a z klienta. Klientská část poskytuje nástroje pro práci s verzemi přímo

v pracovním adresáři a komunikaci se serverem. Server se stará o ukládání verzí. K repozitoři je možné přistupovat různými způsoby, tj. lokálně, přes nativní protokol *svn://* atd. Pro klientskou část existuje několik nástrojů, buď příkazová řádka nebo webové rozhraní či GUI, které má integrované funkce pro klienta. Vice informací naleznete na oficiálních stránkách [6].

3.6. Stručný popis Mercurial verzovacího systému

Mercurial je **distribuovaný** systém pro správu verzí [26]. Převážná část tohoto systému je psaná v programovacím jazyce Python. Obsahuje však i část „diff" (rozdíl mezi verzemi) napsanou v jazyce C. Je multiplatformní, což znamená, že má podporu jak ve Windows tak v UNIX systémech (Linux, Mac OS). Mercurial je primárně určen pro příkazovou řádku, má však i grafická uživatelská prostředí a je dostupný i přes web rozhraní.

Pro označení revizí používá SHA-1 hash[10] a HTTP protokol[11]. Umí také použít SSH připojení[12]. Hlavní cíle Mercurialu jsou výkonnost a škálovatelnost. Distribuovaný/decentralizovaný systém, odolná správa textových i binárních souborů, možnost pokročilého větvení a slévání.

Autorem Mercurialu je Mackall, který tento systém představil 19. dubna 2005 jako nástupce systému BitKeeper. Tento systém byl ukončen pro bezplatné používání. Avšak místo BitKeepru byl jako nástupce použit systém Git. Další informace o Mercurialu jsou na oficiálních stránkách [15].

3.7. Git - distribuovaný systém

Tato podkapitola obsahuje jen krátké představení systému **Git**. Ten je stejně jako Mercurial **distribuovaný** verzovací systém. Byl vytvořen Linusem Torvaldsem pro vývoj jádra Linuxu. Jelikož BitKeeper, který byl pro tuto úlohu používán, skončil s bezplatným používáním. V současné době je Git spravován Junio Hamanem. Je šířen pod licencí svobodného softwaru. Systém byl psán hlavně pro Linux, ale dnes je již tento systém kompatibilní i s Windows.

Na tento systém se zaměřím podrobněji. Pro svou práci využiji Git jako verzovací systém, který podrobím analýze příspěvků. Vybral jsem tento nástroj, protože dnes čím dál více vývojářů začíná využívat tento systém pro své projekty. Více informací o tomto systému naleznete v dokumentaci [8] či v knize [5].

Git byl navržen tak, aby podporoval nelineární vývoj. Podporuje rychlé vytváření a slučování „větví". Má specifické nástroje pro vizualizaci nelineární historie. Umožňuje každému vývojáři mít lokální kopii celé historie vývoje. Změny se importují do úložiště jako nové vývojové větve, které jsou buď začleněny nebo jsou vyvíjeny dál jako lokální větve. Úložiště mohou být zveřejněna různými protokoly jako jsou HTTP, FTP[13], rsync[14] nebo samotným protokolem Git, který je realizovaný pomocí soketu nebo pomocí SSH protokolu.

10 SHA-1 hash je kryptografická funkce, která produkuje 20 bajtovou hodnotu.

11 HTTP protokol je internetový protokol. Tento protokol slouží pro výměnu hypertextových dokumentů ve formátu HTML.

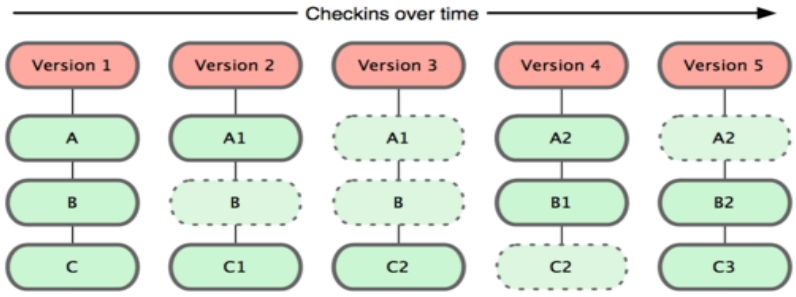
12 SSH – zabezpečený protokol v počítačových sítí, jenž používají TCP/IP protokol.

13 FTP – protokol pro přenos souborů mezi počítači využívající počítačové sítě.

14 rsync – nástroj, který slouží k synchronizaci souborů a adresářů mezi dvěma různými umístěními na unixových systémech.

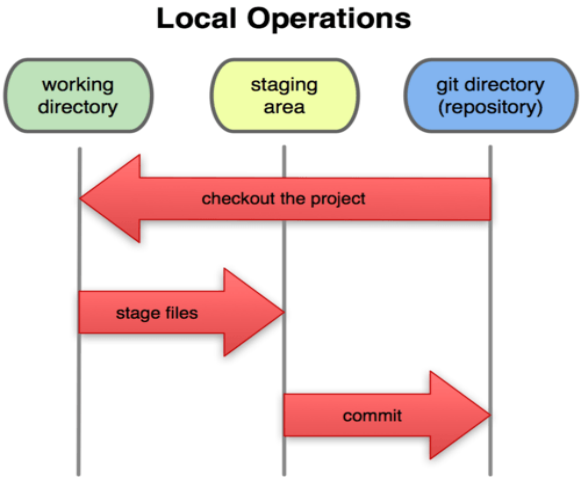
Git je jedním z nejrychlejších systémů pro správu verzí. Má také kryptografickou autentizaci historie. Git explicitně neuchovává vztahy mezi soubory, takzvané revize. To znamená, že pokud chceme zjistit historii změn jednoho souboru je nutné, aby Git prošel globální historii a zjistil zda nějaká změna mohla vést k modifikaci určeného souboru. Díky tomu systém Git umožňuje stejně efektivně zjistit historii pro libovolnou sadu souborů.

Git se liší od ostatních systému ukládáním informací (viz. Ilustrace 3). Používá systém „snapshots“. Tento systém je pro zachycení stavu např. detekci změny jména souboru nebo změny jeho umístění v repozitoři. Pro systém „snapshots“ je nutné využívat různých heuristik.



Ilustrace 3: Ověřování v průběhu vývoje [6]

Git využívá balíčkovací (pack) systém. Tyto balíčky ukládají více objektů. Balíček se komprimuje za pomoci heuristik. Pokud jsou v balíčku nové objekty, které nejsou v repozitoři, tak jsou tyto objekty ukládány odděleně. Pro prostorovou efektivnost je nutné provézt znovu-zabalení (repacking) ten provádí Git automaticky nebo manuálně po zadání příkazu **git gc.**



Ilustrace 4: Tři stavy pro systém Git [6]

Git má tři stavy: zapsáno (commited), změněno (modified) a připraveno k zapsání (staged) (viz. Ilustrace 4).
Stav „zapsáno“ označuje, že data byly úspěšně uloženy v lokální databázi. „Změněno“ je stav, kdy byly provedeny změny, ale soubor ještě nebyl zapsán do databáze. Poslední stav - „připraveno k zapsání“ znamená, že změněný je soubor v jeho aktuální verzi jsme určili k tomu, aby byl zapsán v

další revizizi (commit).

Git je vlastně rozdělen do tří hlavních částí. Tyto částí jsou adresář systému (Git directory), dále pracovní adresář (working directory) a poslední částí je oblast připravených změn (staging area).

Git implementuje několik slučovacích strategii.

- **Resolve,** tato strategie využívá tradiční tří-cestný slučovací algoritmus (3-way merge).
- **Recursive** je také variantou tří-cestného slučovacího algoritmu. Recursive je používán v případě sloučení (merge) či přetažení (pulling) jedné větve.
- **Octopus** je strategie pro slučování více než dvou hlavních větví.

Git samotný má dvě datové struktury. První je **proměnlivý index (stage nebo cache).** Tato struktura slouží pro uložení informace o pracovním adresáři a poslední nahrané revizi. Druhá struktura je neměnný objekt databáze, tak zvaná „append-only".

Objektová databáze obsahuje tzv. **blob objekt[15].** Tento objekt slouží pouze pro obsah souboru.

Další je **Stromový objekt,** který je ekvivalentem adresáře. Stromový objekt popisuje „snapshot" zdrojového stromu. Obsahuje název souboru, který popisuje zda je tento soubor symbolicky odkaz nebo adresář.

Commit (příspěvek) objekt je odkazem na stromy objektů v historii. Příspěvek obsahuje časové razítko, jména žádného nebo více mateřských commit objektů. Obsahuje taky odkaz na nejvyšší úrovně zdrojového adresáře v stromu objektů.

Tag objekt - značkovací objekt, je objektem, který obsahuje odkaz na jiné objekty. Může také obsahovat meta-data ve spojení s jiným objektem. Značkovací objekt se používá pro ukládání digitálního podpisu příspěvku.

Samotný index slouží jako spojovací bod mezi objektovou databází a pracovním adresářem. Každý objekt má kontrolní SHA-1 hash jeho obsahu. První dva bajty tohoto hashe odpovídají umístění v adresáři, zbytek hashe je název souboru pro daný objekt. Každou novou revizi Git ukládá jako nový „blob" objekt a tyto objekty jsou uloženy pomocí komprese zlib[16].

15 BLOB je označení pro datový typ blíže nespecifikovaných binárních dat.

16 ZLIB je svobodná softwarová knihovna užívaná pro kompresi dat.

4. Návrh a Implementace nástroje pro analýzu příspěvků verzovacího systému Git

V této kapitole popíši návrh řešení a mou implementaci tohoto projektu. Pro řešení tohoto zadání jsem zvolil programovací jazyk Python a verzovací systém Git. Tento návrh je pro operační systém Linux a jeho deriváty. V následujících kapitolách je popsán cíl práce, vizualizační nástroj D3 (viz. Představení vizualizační knihovny D3), návrh a implementace (viz. Návrh a implementace nástroje) a získané výsledky z testování tohoto nástroje (viz. Výsledky hodnocení).

4.1. Souhrnný popis cíle práce

Můj program je zaměřený na statickou analýzu, protože statickou analýzu je snadnější odhadnout pro tento daný úkol. Dynamická analýza by byla v rámci tohoto projektu složitá a i časově náročná, jelikož se prochází repozitoř od prvního po poslední příspěvek tzn., že bychom museli zaprvé stanovit určitou sadu testů pro danou repozitoř a dále tuto sadu aplikovat na každý commit. Problém by byl určit takovou sadu a také by nastal problém u větších projektů, kde je spousta příspěvků a kde by tato analýza zabrala obrovské množství času.

Moje řešení shromažďuje výstup ze statického analyzátoru kódu Pylint, výstup z nástroje Radon a moje teoretické hodnocení kvality. Všechny tyto výstupy jsou pro každý příspěvek. Moje teoretická klasifikace je založena na předpokladu, že když určitý příspěvek zůstane nezměněn v budoucnosti, byl tento příspěvek kvalitní. Pro toto ohodnocení využívám dva algoritmy.

První algoritmus – hodnotí pokud byly přidané řádky odebrány v některém z následujícím příspěvku, spočítá podle vzdálenosti mezi těmito příspěvky hodnocení.

Druhý algoritmus odečítá od maximální hodnoty (100) při každé iteraci, kde byly odebrány řádky ze seznamu přidaných řádků, danou hodnotu (viz. Hodnocení příspěvku verze druhá). Výstupy z každého nástroje jsou na konci průměrovány pro výstupní hodnotu pro uživatele.

Nevýhodou mého řešení je to, že je zaměřen pouze na zdrojové kódy psané v jazyce Python. Další nevýhodou je, že tento nástroj počítá jen teoretickou hodnotu kvality.

Podobné nástroje, které ověřují kvalitu kódu jsou například:

- **Coverity -** tento nástroj je určený pro statickou analýzu kódu psaného v C, C++ , Java a C#. Byl vyvinut na Standfordské univerzitě v Palo Alto. Používá se pro nalezení defektů a bezpečnostních slabin v zdrojovém kódu. Tento nástroj našel při zkoumání 53 open-source projektů 6000 chyb [17]. Nevýhodou tohoto nástroje je to, že je placený. Více informací je možné najít na oficiální stránce [18].

- **Sonorgraph** je také nástroj určený pro statickou analýzu kódu. Je pro kód, který je psaný v Javě. Tento program ověřuje závislosti ve zkompilovaných třídách a zdrojovém kódu. Počítá také softwarové metriky[19]. Výsledky pak zobrazuje. Opět jde, ale o placený nástroj [20].

4.2. Základní informace o systémech Linux

Linux je označení pro operační systém, který je založen na unixových systémech. Využívá unixové jádro. Toto jádro vychází ze standardů POSIX a Single UNIX Specification [9]. Jeho tvůrcem je Linus Torvalds a začal ho vyvíjet v roce 1991. Linux je víceúlohový a víceuživatelský systém. V současné době je pojmem Linux myšleno nejen jádro operačního systému, ale také veškeré programové vybavení. Linux je šířen pomocí distribucí a je open-source. Nejznámější distribuce jsou např. Ubuntu, Fedora, Debian, openSUSE a další.

Řešení i návrh tohoto nástroje je pro tento operační systém, jelikož na tento systém je vyvíjeno obrovské množství open-source aplikací psaných v Pythonu, ale i v mnoha dalších jazycích.

4.3. Stručný popis jazyka Python

Jazyk Python je dynamický interpretovaný, objektově orientovaný skriptovací jazyk [10]. Byl navržen Guido van Rossumem v roce 1991. Tento jazyk je vyvíjen jako open source a je dostupný pro Unix, Windows i Mac OS platformy. Pro Linux distribuce je většinou součástí základní instalace. Pro psaní programu v Pythonu je možné využít jak objektově orientovaného paradigma tak funkcionálního i procedurálního podle toho co je vhodné pro danou aplikaci. Jeho významnou vlastností je snadné a rychlé programovaní a to díky jednoduché syntaxy.

V mém řešení jsem využil několik open source knihoven jako je **Pandas [1], Gittle [22] a matplotlib [11]**. Vyžil jsem také knihovnu **mpld3 [12]**, která umožňuje jednoduché převedení grafických výstupů z matplotlib do grafů zobrazených pomocí knihovny D3 ve webovém prohlížeči.

4.4. Představení vizualizační knihovny D3

V této podkapitole velice stručně představím vizualizační modul D3, který je napsaný v javascriptu. Popíši základní pojmy používané s touto knihovnou.

Tato knihovna pomáhá lépe vizualizovat a manipulovat s DOM daty . Využívá technologii HTML, CSS a SVG. Více informací ohledně této knihovny naleznete v dokumentaci tothoto projektu [4].

- **DOM** neboli Document Object Model. Je to objektový model reprezentace pro HTML nebo XML dokumenty. Je to programovatelné rozhraní, které umožňuje modifikaci nebo přístup k obsahu, struktuře anebo stylu dokumentu. Při původním návrhu měl každý prohlížeč své rozhraní. Později W3C standardizovali W3C DOM, který je platformě a jazykově nezávislé. DOM umožňuje vlastně přístup k souboru jako ke stromu (datová struktura), jenž využívá XML parsery a XSL procesory.

- **HTML** je značkovací jazyk pro hypertext. Tímto jazykem se vytváří stránky pro systém World Wide Web (WWW), který umožňuje publikaci dokumentů na Internetu.

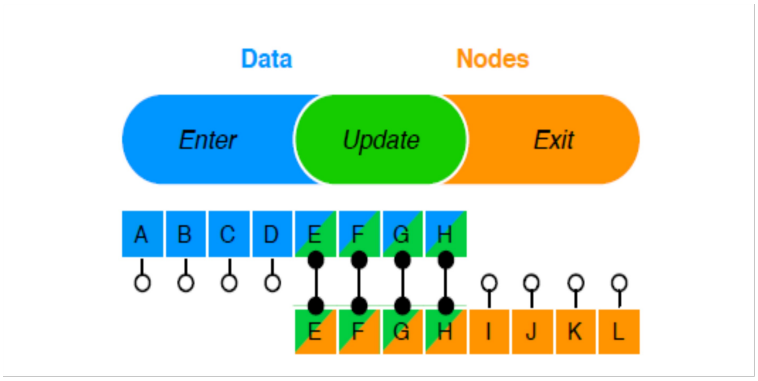
- **CSS,** v překladu kaskádové styly, slouží k popisu způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML. Tento jazyk byl navržen organizací W3C.

- **SVG** je značkovací jazyk a formát souborů, který popisuje dvourozměrnou vektorovou grafiku pomocí XML. Tento formát obsahuje součást komponent a ne obrazová data pixel po pixelu. Je ideální grafy, binární stromy, chronologie, finálové pavouky. Prostě pro jednoduchou grafiku.

D3 umožňuje svázat data s DOM a poté umožňuje následnou transformaci dokumentu. Umí například generovat HTML tabulky jen z pole čísel nebo vykreslí SVG graf pro stejné pole. D3 je velice rychlý, podporuje dynamické chování pro interakci a animace. Je také velice vhodný pro projekty s velkým obsahem dat.

- **Selektory,** díky D3 nemusíme například pro jednoduchou změnu barvy textu procházet všechny elementy v daného tagu. Stačí použít *d3.selectAll()* metodu, která tento proces udělá sama. Existuje více možností, jež jsou díky selektorům možné. Například nastavení atributů nebo stylů, zaregistrování posluchačů na událost. Přidání, odebrání nebo setřídění uzlů a i změna HTML nebo textového kontextu.

- **Dynamické vlastnosti** D3 nám umožňují jednoduše, ale účinně měnit vlastnosti daných elementů. Data jsou reprezentována jako pole hodnot a každá hodnota je možná projít jako první argument (d) v selekční funkci.



Ilustrace 5: Dynamické vlastnosti (Enter, Update, Exit) [5]

- **Enter a Exit** (viz. Ilustrace 5). Tyto selektory umí vytvořit a zrušit nový uzel, který už není nutný. Pokud bychom pouze aktualizovali uzly a nepoužili bysme Enter a Exit, automaticky se vyberou pouze ty prvky, pro které existují vybrané údaje. Díky Enter a Exit je možné, pokud máme více dat než je uzlů, přidávat nové uzly. Tak tedy základní vzorec pro postup se skládá z části pro aktualizaci uzlu a jeho modifikací. Pro použití *d3.enter()*, pro přidaní a použitím *d3.append()* a *d3.exit()* pro odstranění.

- **Transformace.** D3 využívá značek přímo z HTML, CSS a SVG. To znamená, že pokud bude vydán nová funkce je možné ji okamžitě používat, není potřeba nějakých nových nástrojů.

- **Transitions** tato funkce D3 se zaměřuje na transformaci, která přirozeně rozšiřuje animované

přechody. Tyto přechody je možné jednoduše interpolovat pomocí stylů a atributů v čase. Máme různé funkce např. linearní, kubickou a elastickou. Díky D3 je možné interpolovat jak zadaná čísla, tak čísla vložená do řetězce jako je třeba velikost fontů, datové cesty atd. a i složené hodnoty. Díky těmto vlastnostem je možné měnit atributy dynamicky. D3 nenahrazuje žádné komponenty prohlížečů, pouze je rozšiřuje o další možnosti a jednoduché použití.

4.5. Návrh a implementace nástroje

Návrh nástroje jsem koncipoval tak, aby byl spustitelný z příkazové řádky. Také aby byla možnost importovat ho jako knihovnu. To znamená, že první část by byla zaměřená na zpracování parametrů z příkazové řádky a následné předání samotné aplikaci. Samotná inicializace v jazyce Python by měla vypadat takto:

QMetric(url_repo, branch, specific_sha=None, threshold_argument=False, correction=False)

- url_repo - Odkaz na repozitoř, která se nachází na webovém serveru (např. Github.com).
- branch - „Větev" (viz. Základní pojmy pro verzovací systémy), tohoto projektu, kterou je zamýšleno ohodnotit.
- specific_sha - Sha klíč neboli příspěvek, od kterého chceme začínat ohodnocování. Tento příspěvěk by měl být pro nás zajímavý např. „Bug-fix". Tento parametr jsem zvolil jako volitelný, což znamená, že kdyby uživatel nezadal sha klíč, tak bychom prošli celou „branch" dané repozitoře.
- threshold_argument – Pokud je tento argument nastaven na hodnotu True, bude cyklus pro aktuální příspěvek ukončena po dosažení poloviční hodnoty vzdálenosti mezi příspěvky (viz. níže).
- correction – Pokud je tento argument nastaven na hodnotu „True", bude se upravovat výsledné hodnocení pro první algoritmus (viz. níže) po dokončení iterace. Tato korekce bude záviset na počtu zbylých řádků ze seznamu přidaných řádků pro daný příspěvek.

V další části bych se zaměřil na samotnou git repozitoř. První fáze je „klonování"/kopírování celé repozitoře pro danou „branch"/větev do adresáře */tmp*.

Důležitým aspektem pro další využití jsou samotná data, které je nutné získat z klonované repozitoře. Nejdůležitějšími daty jsou sha klíč, autor příspěvku, čas příspěvku, dále soubor, do kterého byla provedena změna a v poslední řadě samotná změna, tzv. „diff". Tyto údaje lze získat, pro verzovací systém Git, pomocí těchto příkazů:

- git log <= Tímto příkazem získáme sha klíče, jméno/přezdívku uživatele, který provedl změnu a čas kdy byla změna provedena.
- git diff sha-klič <= Tento příkaz nám umožní získat zbylé informace jako jsou řádky jež byly přidány a odebrány. Soubor, kterého se změna týká. Tento údaj se nachází v hlavičce výstupu.

Ve svém řešení jsem využil knihovny Gittle, která umožňuje jednoduché získání dat z git repozotoří. Metodou *commit_info()* z Gittle je možné získat daná data jako jsou sha klíč, přisppěvatele, čas kdy byl příspěvek proveden a další. Pomocí metody *diff()* lze získat přidané řádky, odebrané řádky a soubor do kterého byly změny provedeny. Po získání těchto údajů je nutné ohodnotit jak dlouho zůstaly tyto příspěvky nezměněny. Pro toto ohodnocení jsem navrhl více verzí postupů v následujících podkapitolách jsou finální verze a také zamítnuté verze algoritmů.

4.5.1 Finální návrh průběhu algoritmu

Směr ohodnocování je od prvního příspěvku po poslední. Hlavním cílem algoritmu je získat počet řádků, které byly přidány a zjistit jak dlouho tyto řádky zůstali v souboru nezměněny. Důležitý aspekt je vybrání prvního neboli startovacího příspěvku. Tento příspěvek může být, buďto první anebo některý vybraný příspěvek (viz. argument specific_sha výše).

Pro tento první nebo vybraný příspěvek budu dále používat název **vybraný příspěvek**. Navrhl jsem dva postupy pro ohodnocení daných příspěvků.

4.5.2 Hodnocení příspěvku první verze.

Iterace by byla prováděna přes všechny příspěvky dokud by jsme nenarazili na nejčerstvější/poslední příspěvek nebo by byly odstraněny všechny přidané řádky z výchozího stavu.

Jestliže by cyklus skončil tím, že by se dosáhlo posledního příspěvku a stále by nám zůstaly nějaké řádky v seznamu přidaných řádků, ohodnotili bychom tento příspěvek jako kvalitní a pokračovali bychom v ohodnocování s dalším následujícím příspěvkem.

Pokud by cyklus skončil tím, že by byly odstraněny všechny přidané řádky ze seznamu určíme teoretickou hodnotu kvality daného příspěvku podle vzdálenosti, která je mezi **vybraným příspěvkem** a příspěvkem, který je zrovna aktuální.

V případě, že se tento aktuální příspěvek nachází bezprostředně za **vybraným příspěvkem** (tato vzdálenost je 10% z celkového počtu příspěvků**),** je jeho hodnota nejnižší. Mírné zlepšení hodnoty je mezi 10% - 20% **z celkového počtu příspěvků** atd. Vzdálenost přesahující 40% je již dostatečná pro to, že je možné říct, že tento **vybraný příspěvek** je kvalitní**.**

Hodnocení příspěvku v implementaci:

- Vzdálenost < 10% ze všech příspěvků == hodnocení 0
- Vzdálenost > 10% &&[17] vzdálenost < 20% ze všech příspěvků == hodnocení 1
- Vzdálenost > 20% && vzdálenost < 30% ze všech příspěvků == hodnocení 2
- Vzdálenost > 30% && vzdálenost < 40% ze všech příspěvků == hodnocení 3
- Vzdálenost > 40% ze všech příspěvků == hodnocení 4

Vybraný příspěvek se po ohodnocování posune na následující příspěvek a iterace se opakuje od nového **vybraného příspěvku**.

Výsledná hodnota pro každý příspěvek je přepočítána na hodnotu v rozsahu 0 – 100. Vzorec použitý v implementaci je následující:

$$Výsledná\ hodnota = \left(\frac{hodnocení}{4} \right) * 100$$

Pro tento algoritmus jsou navrhnuty dva typy zpřesňujícího rozhodování podle zadaných parametrů k tomuto algoritmu.

Prvním z možností zpřesněním je rozhodování týkající se toho, **kolik přidaných řádků zůstalo v seznamu z výchozího stavu.** Pokud by zůstalo méně řádků nežli jedna třetina, bylo by od výsledného hodnocení odečtena určitá hodnota.

Zpřesňující argument v implementaci:

- Počet zbylých řádků < $\frac{1}{3}$ == výsledné hodnocení – 2
- Počet zbylých řádků > $\frac{1}{3}$ && Počet zbylých řádků < $\frac{1}{2}$ == výsledné hodnocení - 1

17 Tato značka je pro logickou operaci **and** neboli **a zároveň**.

Druhým zpřesněním se týká jak algoritmu verze jedna tak i algoritmu verze dva. V tomto zpřesnění se uvažuje o zavedení pomyslného „**prahu“.** Tím by se zmenšil čas iterování přes všechny příspěvky. Tento práh by mohla být **polovina z příspěvků, přes které má tento algoritmus iterovat.**

Pokud by první algoritmus dosáhl tohoto prahu tak by to znamenalo, že nebyly odstraněny všechny přidané řádky z výchozího stavu a je možné prohlásit tento příspěvek pro hodnocení prvním algoritmem za kvalitní (tedy hodnota 4 v implementaci).

Vliv na druhý algoritmus to má takový, že iterace pro daný příspěvek skončí dříve a výsledná hodnota tímto algoritmem bude vyšší. Tyto dvě zpřesnění hodnoty teoretické kvality byly otestovány a výsledky jsou viz. Výsledky hodnocení.

4.5.3 Hodnocení příspěvku verze druhá

Algoritmus je dá se říci stejný jako u první verze tzn. zajímají nás řádky, které byly odebrány z množiny přidaných řádků. Jen zde je v prvním kroku stanoveno **„výsledné“ hodnocení** pro každý příspěvek na hodnotu 100. A v průběhu ohodnocování, pokud jsou odebrány nějaké řádky ze seznamu přidaných řádků (z výchozího stavu) je odečtena od výsledného hodnocení tohoto příspěvku hodnota, která je vypočítána tímto vztahem:

- NegVal = Hodnota k odečtení
- CountAddedLines = počet přidaných řádků ve výchozím stavu
- RemovedLines = počet odebraných řádků

$$NegVal = \left(\frac{100}{CountAddedLines} \right) * RemovedLines$$

4.5.4 Uložení dat a ohodnocení dalšími nástroji pro analýzu

V řešení jsem využil pro uložení dat získaných při ohodnocování strukturu DataFrame[18] z knihovny Pandas. Tuto knihovnu jsem zvolil, protože je pro zpracování dat velice rychlá. Uložená data ve struktuře DataFrame se dají jednoduše a rychle agregovat a slučovat atd. Více o této knihovně a o spoustě užitečných tipů ohledně práce s daty je možné se dočíst zde [1].
Struktura dat jak jsem je ukládal do DataFrame je následující:

- **Autor příspěvků**
- **Seznam přidaných řádků**
- **Seznam odebraných řádků**
- **Sha klíč**
- **Index (požadovaného) příspěvku**
- **Počet přidaných řádků v tomto příspěvku**
- **Počet odebraných řádků v tomto příspěvku**
- **Čas vytvoření příspěvku**
- **Název souboru (do kterého byl proveden příspěvek)**

18 DataFrame je dvou-dimenzní heterogenní tabulková struktura, tzn má řádky i sloupce. Je primární strukturou pro knihovnu Pandas.

Třetí část je zaměřena na ohodnocení každého příspěvku pomocí nástroje pro statickou analýzu kódu. V implementaci je použit nástroj Pylint (další nástroje jsou např. Coverity, Lint viz. Statická analýza). Dalším nástrojem, který je vhodné použít pro ohodnocení je nástroj, jenž hodnotí softwarové metriky. Pro implementaci jsem použil Radon.

V implementaci je celý zadaný projekt (repozitoř) opět postupně ohodnocován/a od prvního po poslední příspěvek pomocí nástrojem pylint a radon. Abychom mohli zadaný projekt mohli ohodnotit z hlediska historie je nutné vrátit do předchozích stavů. Pro tuto nutnost je možné použit příkaz git rebase -i sha-klíč, který umožňuje návrat k předchozímu stavu projektu. V řešení jsem použil metodu *rollback()* z knihovny Gittle.

Výstupy ze statického analyzátoru a z nástroje pro softwarové metriky jsou dále ukládány do souboru, jelikož tyto hodnoty se nemění a ohodnocování těmito nástroji zabírá pro větší projekty velké množství času. V implementaci jsem využil knihovny pickle[19], která umožňuje „serializovat“ (převést) Python objekty na bajty a uložit do souboru i následně načíst zpět do programu.

Dalšími daty co by měli být uloženy jsou statistiky jako: průměrné hodnoty hodnocení získaných z nástroje Pylint, Radon a mých algoritmů. Tyto výstupy jsou v implementaci uloženy do souborů *result_projekt.txt* a *reuslt_projekt.json* (viz. Příloha A.). Tyto soubory by také obsahují statistiky jako je počet všech přidaných řádků, počet všech odebraných řádků, nejčastěji modifikovaný soubor, počet všech příspěvků a další. Pro analýzu jsem ukládal také výstup z metody *diff()* (viz. Příloha A.).

4.5.5 Návrh pro zobrazení získaných dat

Poslední částí je zobrazení výsledků získaných během klasifikace všemi nástroji. Tuto vizualizaci je možné provést pomocí nativních knihoven daného jazyka. Anebo pomocí nástroje D3.js či dalších nástrojů pro vizualizaci podle vlastní volby.

Pro tyto účely je vhodné, aby data byly uloženy ve formátu JSON[20], CSV[21] nebo TSV[22] či jiných formátech do souboru, aby mohly být zobrazeny pomocí grafů. Pro řešení jsem zvolil formát JSON (viz. Příloha A.).

Hlavní výsledný graf, by měl být graf hodnot hodnocení kvality všech příspěvků pro daného autora ze všech čtyř hodnotících systémů. V tomto grafu by mělo být zobrazeno porovnání hodnocení statického analyzátoru a teoretického hodnocení příspěvků z hlediska historie tzn. od prvního příspěvku po poslední. Toto porovnání by vykazovalo jaký vztah jednotlivé nástroje mezi sebou mají a zda detekují příspěvky s nízkou kvalitou. Takové příspěvky, kde byla nalezena chyba či špatný přístup k vývoji.

Zobrazení, v mém řešení, jsem implementoval pomocí knihovny **matplotlib[11]** a knihovny **mpld3[12]** napsané v jazyce Python a JavaScriptu. Grafy výsledků hodnocení pro každý příspěvek k danému přispěvateli je možné vytisknout jednotlivě, pro každý nástroj zvlášť do grafu. Anebo všechny výsledky mít v jednom grafu, kde je možné pozorovat souběh mezi nástroji.

Vizualizace pomocí **samotné knihovny D3** jsme nakonec **vyloučili**, jelikož **dostačující** vizualizaci jsou výstupní **grafy** z **matplotlib** a knihovny **mpld3.**

---

19 https://docs.python.org/2/library/pickle.html

20 JSON je datový formát nezávislí na počítačové platformě. Data mohou být organizována v polích či objektech.

21 CSV je formát uložení dat, kde jsou data oddělena čárkou.

22 TSV je datový formát, kde jsou uložená data oddělena tabulátorem.

4.5.6 Zamítnutá řešení

Předchozí zamítnuté řešení zahrnovalo cyklus, který počítá přidané a odebrané řádky od posledního příspěvku po první. To znamenalo, že fungoval opačně (zajímali nás odebrané řádky z počátečního stavu).

Další věcí kterou by tento algoritmus počítal, je kolikrát daný autor změnil jeden řádek ve stejném souboru. Pokud by tato hodnota přesahovala určitý práh, bylo by to bráno jako známka nižší kvality tohoto autora a snížilo by se mu hodnocení.

Tyto dva návrhy - kriterium pro kvalitu i algoritmus s opačným systémem procházení historie příspěvků, byly po konzultacích s panem Wardem, **zamítnuty**. Důvodem bylo to, že bychom museli aplikovat nějaký chytrý algoritmus, který by určil zda byl zadaný řádek změněn nebo byl pouze poupraven či posunut v souboru. Tento algoritmus by byl nejspíše založený na strojovém učení.

Další návrh, který byl po konzultacích **zrušen,** je již zmíněné zobrazení pouze pomocí knihovny D3 (viz. výše) a výsledná hodnota kvality autora. Místo této hodnoty je pro každého autora vypočítán průměr pro všechny hodnocení (pylint, radon, mé dvě hodnocení příspěvků). Tyto čtyři hodnoty poté slouží jako finální hodnocení daného autora.

4.6. Výsledky hodnocení

Výsledné hodnocení jsem prováděl nad mým projektem a projektem Metrique.
Můj projekt jsem vybral, abych porovnal vývoj mého programu z hlediska historie, ale také proto, že svůj projekt znám a bylo pro mě snazší orientovat se v příspěvcích. Také je tento projekt z hlediska rozsahu příspěvků malý. Metrique jsem vybral, protože tato práce je ve spolupráci s firmou Red Hat a také se dá považovat velikost tohoto projektu z hlediska počtu příspěvků za střední, až velký.

Ohodnocování algoritmy probíhalo několikrát. První ohodnocení proběhlo bez jakýkoliv zpřesňujících argumentů. To znamená, že každá iterace běžela od **vybraného příspěvku**[23] po poslední atd.

Druhé ohodnocení bylo spuštěno s upřesňujícím pravidlem týkajícím se prahové hodnoty. Předposlední bylo spuštěno s pravidlem týkajícím se hodnoty **neodebraných** řádků ze seznamu **přidaných** řádků po dokončení cyklu programu pro daný příspěvek.

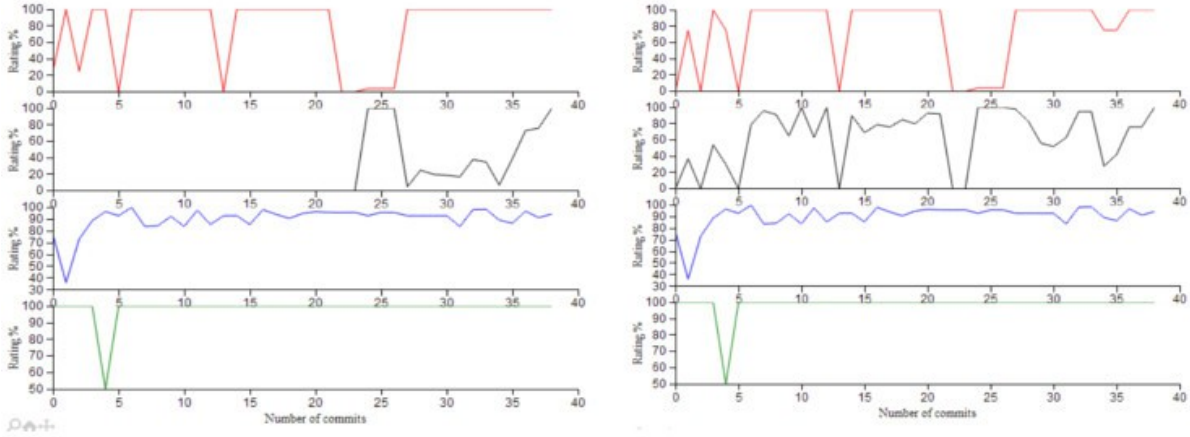
Poslední byla kombinace prahového zpřesnění a rozhodování na základě hodnoty neodebraných řádků ze seznamu přidaných řádků z výchozího stavu programu pro daný příspěvek.

Po pokusech jsem se rozhodl, že zde zdůrazním nejzajímavější výstupy. Tyto výstupy byli při zapnutých obou upřesňujících argumentů a u vypnutých upřesňujících argumentech.

4.6.1 Hodnocení projektu QMetric (tohoto nástroje)

- Červená křivka značí výstup mého **prvního algoritmu**, který hodnotí pouze pokud dosáhneme toho, že jsou všechny přidané řádky právě hodnotícího příspěvku, odebrány.
- Černá křivka je křivkou pro výstup **druhého algoritmu**, který odečítá od výsledné hodnoty hodnocení (100) hodnotu, která je vysvětlena v podkapitole Hodnocení příspěvku verze druhá.
- Modrá křivka je výstup z nástroje **pylint** a zelená je výstup z nástroje **radon**.

23 V tomto případě algoritmy začínaly od příspěvku číslo 1 tzn., že následujícím **vybraným příspěvkem** po ukončení cyklu byl příspěvek s číslem 2. atd.



Ilustrace 6: Grafy výsledných hodnocení ze všech klasifikačních nástrojů. Nalevo je graf s vypnutými upřesňujícími argumenty. Napravo je graf se zapnutými upřesňujícími argumenty

Z těchto grafů (viz. Ilustrace 6) vyplývá, že by nejkvalitnější příspěvky měly být příspěvky s indexem 25 – 29. Tyto příspěvky se týkají odstranění souborů a změny přístupu k vývoji (viz. podkapitola Možnost rozšíření).

Příspěvek číslo 4. získal horší hodnocení od nástroje radon. Důvodem tohoto ohodnocení bylo to, že tento příspěvek obsahoval větší počet řídících struktur[24] na počet řádků v metodách.

Podle výsledků mých hodnotících algoritmů verze 1. a 2. s vypnutými oběma upřesňujícími argumenty lze říci, že prvotní zpracování mého programu bylo nekvalitní. Jelikož žádný z mých prvotních příspěvků nevydržel déle než-li je polovina vzdálenosti mezi danými příspěvky a posledním commitem (viz. 25). Toto tvrzení je svým způsobem pravdivé, protože daný algoritmus i uzpůsobení celého programu se během vývoje, hlavně zpočátku, velice měnilo. Důvodem toho bylo, že v mé první verzi jsem se snažil tento algoritmus navrhnout tak, aby jeho průběh byl od posledního commitu po první. Tento postup byl špatný (viz podkapitola Zamítnutá řešení).

Výsledné křivky s vypnutými upřesňujícími argumenty nám sice ukáží, že z hlediska celého vývoje projektu jsem dostal nízké ohodnoceni. Ovšem neukáží nám přesnější průběžné hodnocení.

Proto je zde graf se zapnutými upřesňujícími argumenty. Ten nám ukazuje přesnější výsledky klasifikace teoretické kvality příspěvků. Nicméně většina nízce ohodnocených příspěvků získala mnohem lepší ohodnocení než-li v předchozím případě.

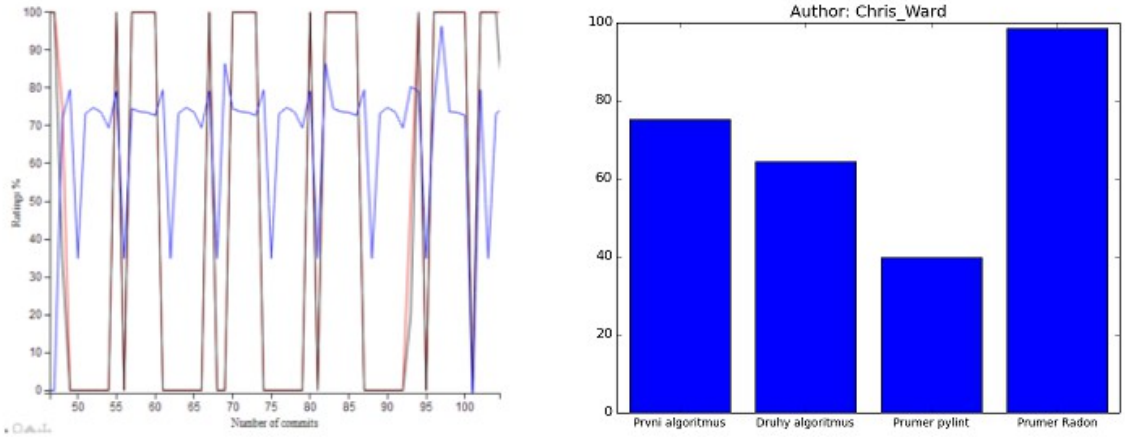
Jak vyplývá z grafů nejhorší hodnocení získávají příspěvky hned zpočátku vývoje, což potvrzuje mé tvrzení o změnách v počátcích.

Je vidět, že upřesňující pravidla mají u tak malého projektu (takový, který má malý počet příspěvků) velký vliv. Proto doporučuji tento nástroj spouštět, nad malými projekty s oběma upřesňujícími argumenty zapnutými. Průměrné ohodnocení kvality autora je v příloze C (Ilustrace 11).

24 Řídící struktura je konstrukce pro zápis programu. Řídící struktury rozhodují o dalším průběhu programu tzn. vytváří cykly, větvení či jinak mění program (if, for, while...).

4.6.2 Hodnocení projektu Metrique

Další grafy se týkají projektu **metrique** (viz. Ilustrace 7)**.** Tento projekt má velký rozsah příspěvků, a proto jsem vybral jen určité části grafů. V těchto částech mají moje teoretické hodnotící algoritmy podobnou charakteristiku jako výstup z nástroje **pylint**.



Ilustrace 7: Nalevo je graf hodnocení projektu v počáteční fázi. Napravo je výsledný graf průměrných výsledků ze všech nástrojů. Výsledné grafy pro další přispěvatele (viz. 34)

Tento stav nastal při zapnutí obou omezujících pravidel. Jak je vidět, je zde téměř periodické opakování jak nástroje **pylint**, tak mých hodnotícího algoritmů verze 2. i verze 1.

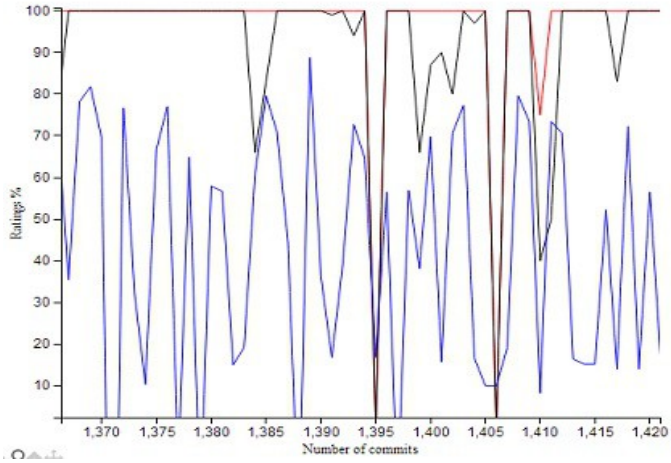
Například příspěvky s čísly 56, 68, 75, 81 mají stejnou charakteristiku pro **pylint** i můj program, protože byly tyto příspěvky provedeny do souboru */metrique/metrique/result.py*. Samotné příspěvky byly „pokusné“ kusy kódu, které byly téměř ihned při následujících commitech[25] vymazány.

Z grafu tohoto výsledného hodnocení pro všechny příspěvky daného autora, je vidět, že tento autor periodicky přispíval do stejných souborů. Je znát, že autor často měnil kód, a to je důsledek nižší kvality přístupu k vývoji.

V tomto případě jde o počáteční vývoj na projektu **metrique.** V této fázi se testovaly různé stavy. Toto tvrzení je potvrzeno po diskusi s vedoucím a hlavním vývojářem tohoto projektu panem Wardem. Podle této diskuse jsem se dozvěděl, že by mělo jít vidět postupné zlepšení/zvýšení kvality příspěvků a celého projektu. Z důvodu změny přístupu a postupu při vývoji. Je možné říci, že jsem toto tvrzení testoval.

Jak je vidět na následujícím grafu (viz. Ilustrace 8), v němž jsou příspěvky přibližně z konce aktuální vývojové linie, kvalita příspěvků mým hodnotícím systémem povětšinou stoupala. Tyto výsledky potvrzují zmíněnou domněnku pana Warda, že kvalita příspěvků do tohoto projektu byla mnohem vyšší, než-li na počátku.

25 commit = příspěvek viz. 11



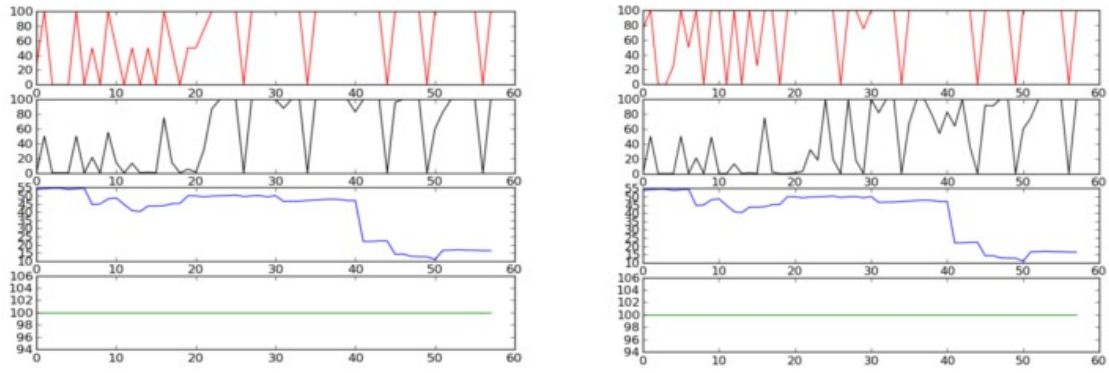
Ilustrace 8: Graf ohodnocení příspěvků projektu Metrique přibližně u konce aktuální vývojové linie.

Jako nejméně kvalitní byly označeny příspěvky s číslem 1395 a 1406. V obou těchto příspěvcích byl přidán pouze jeden řádek. Tento řádek byl poté odebrán v následujícím příspěvku.

V souvislosti s příspěvkem číslo 1395 proběhly změny i v dalších souborech, nejen v souboru *src/metriquec/metriquec/cubes/sqldata/generic.py* a to z důvodu opravy. Výsledkem bylo, aby se návratová hodnota vracela jen pokud budou splněny určené podmínky.

Commit s číslem 1406 byl odebrán ze souboru *src/metrique/metrique/core_api.py.* Změny opět proběhly ve více souborech a to z důvodu toho, aby se obešlo výjimce v běhu programu. Jak je vidět z hodnocení kvality mými algoritmy zbylých příspěvků, kvalita se držela na vysoké úrovni. To znamená, že průběh vývoje byl ke konci mnohem lepší. Ovšem ohodnocení získaná nástrojem pylint vyhodnotila, že by daný autor měl přehodnotit/přepsat a otestovat kód v souborech:
src/metriqued/metriqued/tornadohttp.py,
src/metriquec/metriquec/cubes/sqldata/teiid.py,
src/metriquec/metriquec/cubes/github/issue.py atd.
Jelikož pylint hlásí, že by v těchto souborech mohla nastat chyba.

Grafy pro ohodnocení kvality získaná při zadání příspěvku s označením „Bug-fix“.



Ilustrace 9: Nalevo graf se zapnutými omezujícími argumenty. Napravo je graf s vypnutými omezujícími argumenty

Poslední grafy (viz. Ilustrace 9) zobrazují, kdy byl zadán „určitý příspěvek“. Tento příspěvek byl „Bug-fix“ pro projekt **metrique**[26]**.** Tato oprava se týkala špatné návratové hodnoty při získávání dat a opravy podmínek k ověření, zda daná proměnná je instancí určitého typu.

Jak je vidět na grafech, řádky z prvního příspěvku byly pozměněny po dost dlouhé době, jelikož hodnota získaná mým prvním algoritmem je 80% (druhá červená křivka). Avšak tato doba musela být pod hranicí poloviny všech příspěvků do tohoto souboru. Vyplývá to z výstupního hodnocení druhým algoritmem, které bylo 0%.

Toto hodnocení bylo provedeno s omezeními. To znamená, že tyhle změny nevydrželi dlouhou dobu z hlediska vzdálenosti mezi příspěvky. Jak je vidět klesá i křivka pro hodnocení z nástroje **pylint**. Takže tuto opravu můžeme označit za méně kvalitní z hlediska hodnocení mým hypotetickým algoritmem i nástrojem pylint.

4.7. Odhalené chyby v projektu Gittle

Během testování jsem narazil na dva problémy u knihovny Gittle, které jsem nahlásil na „issue tracker“ na stránce *http://www.github.com*.
První problém se týkal klonování projektů. Hlásil chybu při označení verze daného projektu např. dulwich.errors.RefFormatError: refs/tags/dulwich-0.9.3^{}. Tento problém jsem obešel tím, že jsem systémově volal git clone[27].
Druhý problém byl takový, že pokud se zvolila branch (větev) jiná nežli „master“, metoda diff() nahlásila chybu klíče. Tento problém jsem obešel nastavením DEFAULT_BRANCH na „větev“, která je zadána při spuštění mého programu[28].

4.8. Možnost rozšíření

První možností pro rozšíření je filtrování příspěvků, které v podstatě jen zaznamenaly zrušení souboru. Jelikož tyto příspěvky jsou pak ve výsledku ohodnoceny jako kvalitní. Také filtrování pouze zdrojového kódu, tímto je myšleno vynechávat komentáře.

Další možností pro rozšíření je implementovat algoritmus založený na strojovém učení, pro výběr specifického příspěvku. Tento specifický příspěvek by měl být zajímavý nebo například označený jako „Bug-fix“ či oprava nějaké chyby v kódu. Toto strojové učení by zřejmě mělo být učení s učitelem, abychom tento algoritmus naučili hledat pouze ty příspěvky, které chceme.

Předposlední možností rozšíření je predikce hodnocení kvality příspěvků od autorů založených na získaných datech. Tato predikce by mohla být pouze průměrná hodnota získaná z dat anebo hodnota získaná z nástroje, který je implementován na principu strojového učení a určí předvídanou hodnotu ze získaných dat.

Posledním navrhovaným rozšířením by mohlo být to, že by se tento projekt přestal zabývat pouze kódem psaným v jazyce Python, ale rozšířila by se jeho působnost na další jazyky, jako jsou například C/C++, Java, C#, Perl a další. Tohle rozšíření by znamenalo přidat k tomuto projektu další nástroje pro statickou analýzu, pro každý jednotlivý jazyk, abychom mohli posoudit zda tato hodnocení odpovídají skutečnosti.

---

26 https://github.com/kejbaly2/metrique/commit/106a25e4f6270b73bb50f21062d2b1aa0f8879e1

27https://github.com/FriendCode/gittle/issues/33
28https://github.com/FriendCode/gittle/issues/31

5. Závěr

Cílem tohoto projektu bylo navrhnout algoritmus, který ohodnotí teoretickou kvalitu pro každý příspěvek a daného autora. Tento cíl jsem splnil tím, že jsem každý příspěvek ohodnotil výstupem z nástroje pylint, nástroje radon a hlavně mými dvěma klasifikačními algoritmy. Další možné rozšíření jsem uvedl v předchozí kapitole (viz.Možnost rozšíření).

Na základě vyhodnocení mých algoritmů jsem dospěl k závěru, že pokud příspěvatel změní ve svém příspěvku menší počet řádků, je velice pravděpodobné, že tento příspěvek bude označen jako nekvalitní. Jelikož je větší pravděpodobnost, že tyto řádky budou změněny/zrušeny v několika následujících příspěvcích.

Vzhledem k tomu, že navrhované algoritmy jsou experimentální a jsou založeny na jednoduchém principu přidaných a odebraných řádků, je možné říci, že z hlediska věrohodnosti určení hodnoty kvality, nejsou úplně přesné. Avšak tyto algoritmy splnili svůj účel a to určit „dobu“, po kterou daný příspěvek zůstal nezměněn a určit jeho teoretickou kvalitu na základě daných podmínek (viz. Hodnocení příspěvku první verze. a Hodnocení příspěvku verze druhá).

Mé algoritmy také objevili předpokládané chování v projektech QMetric a Metrique. Tímto chováním je myšleno počáteční nižší kvalita a postupné zlepšení. Toto zlepšení vzniklo v mém projektu na základě změny přístupu k vývoji.

Výstupy získané z mého nástroje, je možné použít jako další statistiky/metriky pro projekty, které chceme ohodnotit teoretickou kvalitou i statickou analýzou.

V první fázi jsem tento projekt pojal jiným způsobem, než bylo zamýšleno. Hodnotil jsem kolikrát se daný příspěvek opakoval, pouze na základě čísla řádků v souboru. Jelikož tato idea nebyla přesná, byla zamítnuta.

Místo toho postupu byl zvolen návrh, který určuje hodnotu kvality na základě toho, jak dlouho každý příspěvek zůstane nezměněn. Tento výsledek by měl odpovídat teoretické kvalitě autora tohoto příspěvku.

Další myšlenka byla, že by tento nástroj hodnotil jakýkoliv kód, ne jen kód psaný v jazyce Python. Ovšem to bylo zamítnuto, protože by toto řešení přesahovalo rámec této práce. Avšak tato myšlenka je v podkapitole Možnost rozšíření. Tento postup je teoretický a musel být nejdříve ověřen.

Vzhledem k různým možnostem rozšíření a potenciálu tohoto nástroje jsem se rozhodl, že budu pokračovat ve vývoji.

Díky tomuto projektu jsem si rozšířil vědomosti ohledně verzovacích systému, zvláště pak systému Git. Další získané vědomosti se týkaly jazyka Python a jeho knihoven Pandas, matplotlib, Gittle a mpld3.

Literatura

[1] MCKINNEY, Wes. *Python for data analysis*. 1st ed. Beijing: O'Reilly Media, 2012, xiii, 452 s. ISBN 978-1449319793.

[2] Static program analysis. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 2 April 2014 at 11:21. [cit. 2014-05-20]. Dostupné z: http://en.wikipedia.org/wiki/Static_program_analysis

[3] LACCHIA, Michele. Radon documentation. *Radon.* [online]. 16.5.2014 [cit. 2014-05-16]. Dostupné z: *https://radon.readthedocs.org/en/latest/index.html*

[4] BOSTOCK, Mike. D3 Wiki. *D3.* [online]. 16.5.2014 [cit. 2014-05-16]. Dostupné z: *https://github.com/mbostock/d3/wiki*

[5] CHACON, Scott. *Pro Git* [online]. [cit. 2014-05-16]. Dostupné z: https://github.s3.amazonaws.com/media/progit.en.pdf

[6] COLLINS-SUSSMAN, Ben, Brian W. FITZPATRICK a C. Michael PILATO. Version Control with Subversion. PILATO, C. *Version control with subversion* [online]. 2nd ed. Beijing: O'Reilly, 2008 [cit. 2014-05-16]. Dostupné z: http://svnbook.red-bean.com/en/1.7/svn.intro.whatis.html

[7] Pylint User Manual. *Pylint 1.2.0 documentation* [online]. 2014 [cit. 2014-05-17]. Dostupné z: http://docs.pylint.org/

[8] Git. *Git - Reference* [online]. 2013-2014 [cit. 2014-05-17]. Dostupné z: http://git-scm.com/docs

[9] Linux. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 16 May 2014 at 11:01 [cit. 2014-05-17]. Dostupné z: http://en.wikipedia.org/wiki/Linux

[10] Python (programming language). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 15 May 2014 at 23:07 [cit. 2014-05-17]. Dostupné z: http://en.wikipedia.org/wiki/Python_%28programming_language%29

[11] HUNTER, John, Daren DALE, Eric FIRING a Michael DROETTBOOM. Matplotlib. *Overview - Matplotlib 1.3.1 documentation* [online]. 2002 - 2012 [cit. 2014-05-17]. Dostupné z: http://matplotlib.org/contents.html

[12] Mpld3. *MPLD3 - Bringing Matplotlib to the Browser* [online]. 2014 [cit. 2014-05-17]. Dostupné z: http://mpld3.github.io/

[13] WARD, Chris. *Metrique* [online]. 2013-2014 [cit. 2014-05-17]. Dostupné z: https://github.com/kejbaly2/metrique

[14] KRÁL, J. *Měření softwaru Problémy používání SW metrik*. 2012. Dostupné z: https://is.muni.cz/el/1433/jaro2012/PA105/10MereniSW.pdf

[15] O'SULLIVAN, Bryan. Mercurial: The Definitive Guide. O'SULLIVAN, Bryan. *Chapter 1. How did we get here?* [online]. 2009 [cit. 2014-05-17]. Dostupné z: http://hgbook.red-bean.com/read/how-did-we-get-here.html

[16] SOLEY, Richard Mark. *How to Deliver Resilient, Secure, Efficient, and Easily Changed IT Systems in Line with CISQ Recommendations*. 109 Highland Ave, Needham, MA 02494, U.S.A., 2012. Dostupné z: http://www.omg.org/CISQ_compliant_IT_Systemsv.4-3.pdf

[17] Coverity. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 15 April 2014 [cit. 2014-05-18].
Dostupné z: http://en.wikipedia.org/wiki/Coverity#Technologies

[18] COVERITY, Inc. *Coverity* [online]. 2014 [cit. 2014-05-18]. Dostupné z: http://www.coverity.com/

[19] Sonargraph. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 14 May 2014 at 18:59 [cit. 2014-05-18].
Dostupné z: http://en.wikipedia.org/wiki/SonarJ

[20] *Sonargraph* [online]. 2014 [cit. 2014-05-18].
Dostupné z: http://www.hello2morrow.com/products/sonargraph

[21] List of tools for static code analysis. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 13 May 2014 at 13:28. [cit. 2014-05-18]. Dostupné z: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

[22] AARON, O'Mullan. *Pythonic Git for Humans* [online]. 2014 [cit. 2014-05-18]. Dostupné z: https://github.com/FriendCode/gittle

[23] ISO/IEC 9126. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 6 January 2014 at 18:55. [cit. 2014-05-20]. Dostupné z: http://en.wikipedia.org/wiki/ISO/IEC_9126

[24] BOROVCOVÁ, Anna. *Testování webových aplikací*. Praha, 5.8.2008. Dostupné z: https://docs.google.com/file/d/0B6-XMUrK_hQLYzUwYzI1ZmMtMTY5ZC00MDNlLWJhNTEtZDFiM2IxN2NkMjA3
Informatika, obor softwarové systémy. Univerzita Karlova v Praze. Vedoucí práce Mgr. Vladan Majerech, Dr.

[25] Apache Subversion. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 5 April 2014 at 10:39. [cit. 2014-05-19]. Dostupné z: http://en.wikipedia.org/wiki/Apache_Subversion

[26] Mercurial. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014, 15 May 2014 at 14:23 [cit. 2014-05-19].
Dostupné z: http://en.wikipedia.org/wiki/Mercurial

Seznam příloh

Příloha A. Reprezentace dat pro výstupní soubory

Pro reprezentaci dat jsem zvolil tři soubory. Do prvního souboru bude zapsána výstupní struktura ve formátu JSON, aby tyto data mohly být zpracovány pomocí D3 či jiným vizualizačním nástrojem. Další výstupní soubor je textový report udávající všechny získaná data. Poslední je soubor, který obsahuje sha klíč, čas příspěvku a diff tohoto příspěvku pro dohledání co bylo přidáno/smazáno za řádky.

Data ve formátu JSON

```
[
    {
        “Name”:string,
        “Added_lines”: number,
        “Removed_lines”: number,
        “All_commits”: number,
        “Average_Commits_Ratings_v_1”: number,
        “Average_Commits_Ratings_v_2”: number,
        “Average_Software_Metrics”: number,
        “Average_Pylint”: number,
        “Data”: [
                {
                    “Date”: string,
                    “Commit_Rating_v_1”: number,
                    “Commit_Rating_v_2”: number,
                    “Pylint_Rating”: number,
                    “Software_Metrics_Rating”: number
                }
            ],
        “Commits_to_most_modified_file”: number,
        “Most_modified_file”: string,
        “Count_of_Pylint_negative”: number,
        “Count_of_Pylint_positive”: number
    }
]
```

Data pro výstupní soubor

Data jsou u obou souborů zapsána od prvního příspěvku po poslední.

report_jmeno_projektu.txt

Project: Projekt

##

==

Authors: Autor

Current average hyphotetical rating quality version 1. is : %.2f

Current average hyphotetical rating quality version 2. is : %.2f

Current average pylint quality of project is : %.2f

Current average radon quality of project is : %.2f

==

Quality of contributor/s:

Index in graph: %d

Index to diff file: %d

File: soubor/i/s/cestou

Date: datum

Commit rating version. 1: %.02f

Commit rating version. 2: %.02f

Pylint rating: %.02f

Radon rating: %.02f

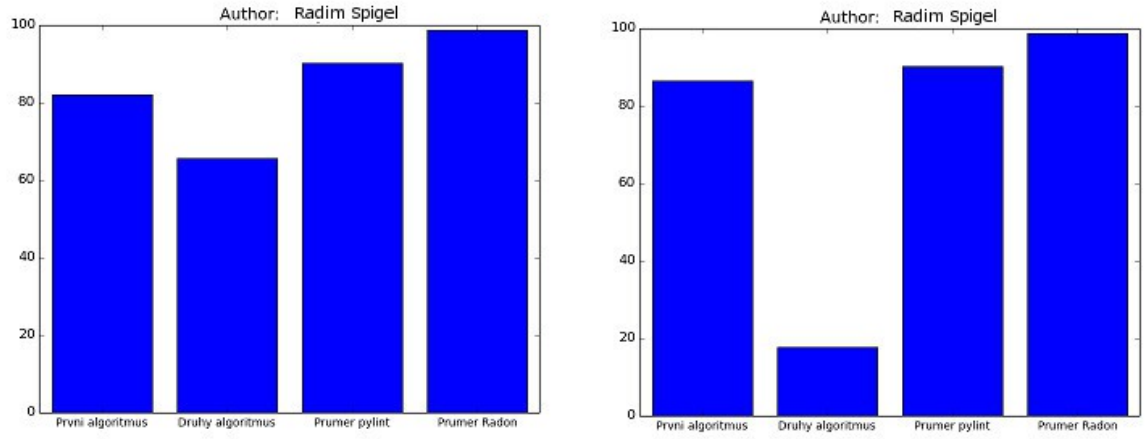
...atd.

diff_jmeno_projektu.txt

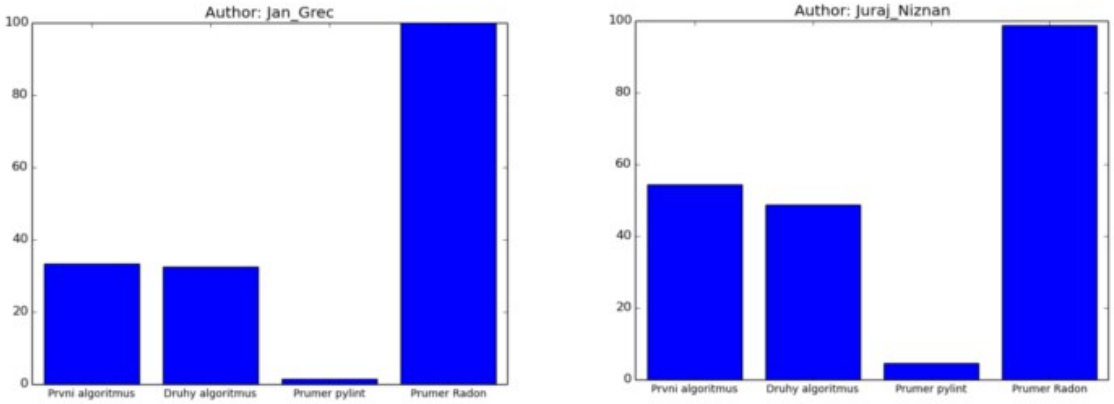
index: %d sha: sha klíč date: datum kdy byl příspěvek přidáním

NL	RL	DIFF
1	1	print „Hello world“
2	2	for i in list_:
2	3	- if i > 5:
2	4	- print i
3	4	+ if i < 5:
4	4	+ print i +1

Příloha B. Výsledné grafy



Ilustrace 10: Grafy průměrných výsledků. Nalevo je graf ,kde jsou všechny upřesňující argumenty zapnuty. Napravo je graf, kde jsou všechny upřesňující argumenty vypnuty.



Ilustrace 11: Grafy dalších přispěvatelů do projektu Metrique.

Příloha C. Obsah přiloženého CD

/QMetric/
Adresář obsahující celý program.

/QMetric/INSTALL.txt
Obsahuje popis nástrojů a knihoven, které je nutné stáhnout a nainstalovat pro běh programu.

/QMetric/setup.py
Tento soubor je možné spustit pro instalaci na operační systém Linux, až poté co jsou nainstalovány nástroje a knihovny uvedené v INSTALL.txt.
Instalace :
python setup.py build
python setup.py install

/QMetric/QMetric.py
Obsahuje celý tento projekt. Základní spuštění z příkazové řádky v operačním systému Linux je:
python QMetric.py https://github.com/Radymus/QMetric.git