VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

REPORTOVACÍ NÁSTROJ NAD GIT

GIT REPORTING TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH NEČAS**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Ing. TOMÁŠ VOJNAR, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Bakalářská práce se zabývá implementací softwarového nástroje pro vytváření a vizualizaci statistických informací o práci v souborech, které sleduje systém správy verzí - Git. Systém Git nemá funkce pro grafické znázornění statistik (grafem nebo tabulkou) a výstup informací o změnách v souborech je pouze v příkazovém řádku. Taková data, zejména u velkého počtu sledovaných souborů, není snadné analyzovat bez použití jiných existujících statistických nástrojů. Existuje několik nástrojů, které umí získat informace o změnách v Git, vyhodnotit statistiky a zobrazit výsledky. Hlavním problémem u těchto nástrojů je příliš dlouhá doba potřebná pro získání výsledků, podpora jen velmi stručných statistik nebo slabé uživatelské rozhraní. Nástroj implementovaný v této práci řeší efektivnější vyhodnocení výsledků pomocí databáze pro uložení informací o změnách z Git. Databáze oproti příkazovému řádku poskytuje rychlý přístup k uloženým informacím a časově náročné je pouze její založení. Snadné ovládaní nástroje zajišťuje grafické uživatelské rozhraní pomocí oken. Výsledkem jsou statistiky, které si může uživatel i přizpůsobit podle vlastních potřeb (která data budou zahrnuta a jejich rozsah). Navíc nástroj poskytuje přibližný odhad (matematicko-statistickou metodou nejmenších čtverců), jakým směrem se práce ve sledovaném projektu ubírá (zda je ve fázi vývoje, nebo dokončování a údržby).

Abstract

This bachelor thesis is about an implementation of the software tool dedicated for creation and visualization of statistical information files work. It also refers about Git version control system. System Git does not have any functions to display graphical statistics (graph or chart), therefore the output of the changes of the files is available in command line only. Such data, especially in a large amount of specific files, are not easy to analyze without using other existing statistic tools. There are couple of tools available which are capable of getting information of changes in Git, classify the statistics and display the results. The main disadvantage of such tools is a very long time needed to obtain the results. These tools support only very brief statistics and the user interface does not provide sufficient comfort either. The tool elaborated in this paper is offering more effective access to the changes information from Git using the saved data database. This database, compare to a single command line, is offering quick access to all information and timeline while there is only one time demanding operation – setting up the database. User-friendly interface is enabling an easy operation of the tool itself using window based interface. As a result there are various statistics (graphs and charts) which are user-editable. On the top of that the tool is equipped with an approximate estimation of the project work trends (mathematical-statistical method of the least squares).

Klíčová slova

systém, správa, verze, git, log, revize, vcs, report, nástroj, graf, statistika, databáze, schéma, vizualizace

Keywords

system, control, version, git, log, revision, vcs, report, tool, graph, plot, statistic, database, schema, visualization

Citace

NEČAS, Vojtěch. *Reportovací nástroj nad git.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Tomáš Vojnar, Ph.D.

Reportovací nástroj nad git

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Vojnara. Další informace mi poskytl Ing. Pavel Tišnovský. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Nečas
16. května 2018

Poděkování

Děkuji svému vedoucímu za vedení při tvorbě bakalářské práce. Dále děkuji Ing. Pavlu Tišnovskému za poskytnuté rady a podporu při tvorbě práce.

Obsah

Kapitola 1

Úvod

Současný vývoj a správa projektů v oblasti informačních technologií vyžaduje přehledné a jednoduché řízení. Nejčastěji jde např. o projekty nového software. Výběr vhodného způsobu dohledu nad prací a stavem dat, které jsou výsledkem činnosti mnohdy velkého počtu autorů, přináší úsporu času a především jasný přehled o provedené práci. K tomuto účelu se používají systémy správy verzí popsané v kap. 2, které umí zaznamenat změny nad sledovanými daty (soubory) a většinou nabízí i možnost zrušení konkrétních změn a návrat do původního stavu před změnami. Existujících systémů správy verzí je několik a každý má své výhody i nevýhody. Výběr záleží na charakteru dat, se kterými systém pracuje a na nárocích uživatelů. Jedním ze systémů správy verzí je i Git, jehož koncept, způsob ukládání revizí (verzí) a získávání informací o revizích popisuje kap. 2.4. Pokud chceme informace ze systému Git podrobněji analyzovat a prezentovat, musíme použít jiné nástroje, protože Git umí poskytnout jen soupis informací o změnách a pouze v příkazovém řádku. Existující statistické nástroje (popsané v kap. 2.4.6), které jsou schopné informace o změnách v repozitáři vyhodnotit, výsledky naformátovat a poskytnout je uživateli formou textu, tabulky nebo grafu. Nedostatkem je u těchto nástrojů omezení uživatele pouze na implementované statistiky, které uživatel nemůže měnit podle potřeby, nebo nevhodný formát výsledků. Většina nástrojů vyžaduje s každým dalším sestavením výstupu opětovnou analýzu všech informací z Git a tím se stává nástroj nepoužitelným u rozsáhlých projektů (dlouhé čekání na výstup).

Hlavním cílem práce je implementace nástroje schopného prezentovat statistické informace z Git a zobrazit je uživateli formou grafů. Nástroj používá k uložení dat z Git databázi s vhodně navrženým schématem. Databáze slouží jako pomocná strutura, aby data nemusela být získávána přímo z Git s každým sestavováním statistik a její návrh popisuje kap. 3. O implementaci nástroje pojednává kap. 4, kde popisuji implementaci uživatelského rozhraní, implementaci součásti nástroje, která spravuje databázi a způsob jakým se statistiky zobrazují. V kap. 5 popisuji jaké typy statistik nástroj generuje a v kap. 6 se věnuji základnímu otestování nástroje na skutečných repozitářích.

Zadání vypsala společnost Red Hat Czech, s.r.o., pod vedením odborného konzultanta Ing. Pavla Tišnovského.

Kapitola 2

Systém správy verzí

Systém správy verzí (VCS[1]) uchovává přístup ke skupině objektů a historii změn v nich. V případě softwarových projektů jsou objekty obvykle soubory[2] obsahující zdrojový kód, dokumentaci, externí knihovny atd. Oblast sledovaných dat nazýváme repozitář a sledování probíhá se zaměřením na změny v čase. Základem je uchování informace o každé změně a její přesný rozsah. Lze využít i možnost návratu souboru (nebo skupiny souborů) do původního stavu pomocí dříve zaznamenané verze [11]. Velké projekty mívají změny téměř neustále. Bez použití VCS může uživatel ručně provést kopii dat do jiného umístění (adresáře), které vhodně pojmenuje, ale takový přístup s sebou nese velké riziko ztráty orientace ve velkém množství záložních umístění a snadno dochází k chybám. Proto jsou VCS systémy tak oblíbené. Statistiky z VCS poskytují informace o autorech, kteří se podíleli na úpravách konkrétních souborů a navíc i užitečný přehled o změnách, díky kterému snadno lokalizujeme hledané změny. Výhodou VCS je snadná správa i v situaci, kdy autoři mění data repozitáře z geograficky vzdálených míst (v různém časovém pásmu), nebo když používají odlišných pracovních zvyků, protože rozhraní VCS prezentuje informace unifikovaně. Informace o změnách uchovávají VCS zpravidla v logovacích souborech. Pestrost typů VCS vychází z postupného vývoje a z různých požadavků uživatelů na funkce.

2.1 Lokální

Poskytují základní možnost zálohovat data před aplikováním plánované změny a zaznamenat změnu. Lokální VCS přináší automatizaci s využitím uložení všech změn dat (včetně informací o změnách) do lokální databáze. Příkladem lokálních VCS je RCS (Revision Control System) distribuovaný na operačních systémech UNIX nebo SCCS (Source Code Control System) vyvinutý společností IBM a později také implementován i pro systém UNIX. Výhoda místní lokalizace spočívá v jednoduchosti, naopak nevýhodou je problematický přístup týmově spolupracujících autorů z jiných míst a stabilita systému. Selhání úložiště lokální stanice s uloženou databází změn bez předchozí zálohy samotné databáze na jiné úložiště znamená ztrátu všech zaznamenaných dat [10].

[1] Z anglického Version Control System. Může dojít k záměně zkratky VCS a CVS (z anglického Concurrent Version System), což je jiný systém pro sledování verzí a který je součástí většiny distribucí Linuxu. CVS má charakter obecných VCS, ovšem při vývoji VCS hrál důležitou roli. Podporuje souběžnou práci více uživatelů nad stejnými soubory, což u dřívějších VCS nebylo možné a práce jednoho uživatele až do okamžiku uvolnění blokovala práci ostatním uživatelům [21].

[2] Adresář (složka) se považuje za typ souboru, jak je tomu například u operačního systému UNIX [23].

2.2 Centralizované

Snaha vyvinout VCS snadno přístupné většímu počtu spolupracujících autorů vedla k návrhu aplikovaném na architektuře klient/server. Takové systémy nazýváme centralizované VCS (CVCS). Princip spočívá v činnosti centrální stanice (serveru) uchovávající všechny verze souborů. Jednotliví autoři (klienti) mohou ze serveru stahovat potřebná data a informace, včetně původních verzí. Hlavním rysem je práce pouze na klientské pracovní stanici, kde až po dokončení úprav a potrzení dochází k aktualizaci kopie přeposláním na centrální stanici. Příkladem jsou systémy SVN (Apache Subversion), CVS nebo Perforce s kvalitním uživatelským grafickým rozhraním. Oproti lokálním VCS spočívá výhoda v nižší režii na správu databáze verzí. Citlivým místem je podobně jako u lokálních VCS selhání způsobené výpadkem úložiště s hlavní databází verzí. Opravitelné problémy s úložištěm způsobí nedostupnost centrální stanice ostatním klientům po dobu oprav a nelze uložit změny provedené na straně klientů. Neopravitelné poškození centrálního úložiště má za následek ztrátu všech dat s tím, že k obnově lze použít zálohu úložiště serveru nebo verze uložené na stanicích klientů s omezením na jejich aktuální verze [10].

2.3 Distribuované

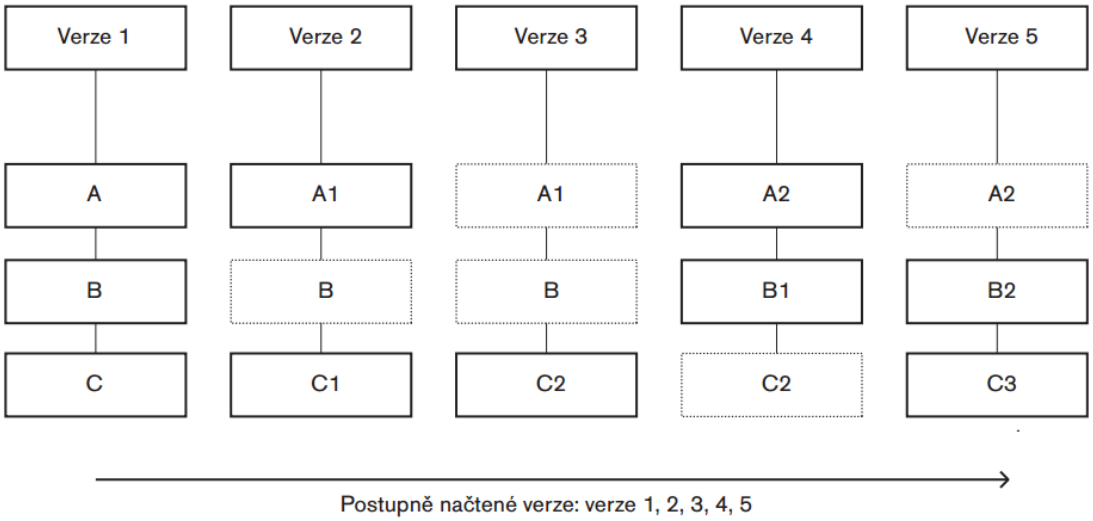
Vychází z konceptu centralizovaných systémů s tím, že distribuovaný VCS (DVCS) snižuje jejich nedostatky na minimum. Hlavní výhoda spočívá v možnosti obnovy při kolizi centrální stanice. Ztracená data lze snadno získat od některého z klientů, protože klienti uchovávají celkovou kopii repozitáře. Nově nabízí distribuované VCS i správu více repozitářů najednou s možností rozdílných rolí klientů pro konkrétní repozitář. Příkladem distribuovaného VCS jsou Git, Bazaar, Darcs nebo Mercurial [10].

2.4 Systém Git

Vzhledem k tomu, že tato práce se zaměřuje primárně na systém Git, popíšeme si nyní blíže principy fungování tohoto systému. Správné pochopení principů je důležité pro využití výhod, které Git přináší a pro pochopení odlišností od ostatních systémů.

Začneme stručnou historií systému Git. Historie Git je úsce spojena se systémem Linux. Operační systém Linux má jádro vyvíjené s pomocí mnoha autorů a systému správy verzí. Od roku 2002 využívali vývojáři Linuxu komerční systém správy verzí Bit-Keeper. Časem však nepříznivé vztahy mezi komunitou vývojářů Linuxu a společností poskytující systém Bit-Keeper vyústily v zastavení poskytování systému bezplatně. Situace nedovolovala vývojářům Linuxu hradit náklady za licenci a proto se jeden z hlavních tvůrců Linuxu, Linus Torvalds, rozhodl vytvořit nový systém správy verzí, který nazval Git[3]. Systém měl dle plánů vykazovat jednoduchost, rychlost, distribuovanost, schopnost spravovat na kvalitní úrovni projekty velkého rozsahu (např. Linuxové jádro) a podporu práce s daty na několika místech současně. Všechny tyto požadavky systém uspokojuje. Nyní kromě samotného Git existuje i řada platforem a služeb pro správu pomocí Git (např. Bitbucket, Codebase, GitHub, GitLab nebo Assembla).

[3]Vývoj do značné míry ovlivnily znalosti z dříve používaného systému Bit-Keeper. Torvalds využil objektový koncept a první funkční verzi nového systému se mu podařilo vytvořit za velmi krátký čas (jeden týden) [21].



Obrázek 2.1: Záznamy verzí se snímky [10]

2.4.1 Zpracování dat

Git vnímá oblast sledovaných dat jako vlastní souborový systém, nad kterým uchovává informace o každé změně. Nosičem informace o změně je snímek. Pokud se data v systému jakkoli změní, Git zaznamená snímek souborů. Přístup ke snímkům později probíhá pomocí referencí na snímek, což je výhodné v případě, kdy nedošlo ke změnám u jiných souborů, protože nezměněné soubory jsou reprezentovány pouze referencí na jejich předchozí verzi. Příklad je na obr. 2.1, ve kterém písmena A, B, C znamenají soubory, čísla u písmen verze souborů a „Verze“ označuje snímek s uvedením pořadového čísla. Takový přístup umožňuje zpracovat změny nebo poskytnout informace téměř okamžitě a šetří se i paměťové zdroje, na které systém zaznamenává data. Běžný VCS zapisuje informace o sledovaných datech ve formě seznamů změn, ke kterým došlo v jednotlivých souborech a případně uchovává každou verzi souborů v původní podobě. Takové řízení není oproti Git efektivní ani rychlé.

Konzistentního přehledu o stavu dat dosahuje Git použitím kontrolních součtů. Zaznamenání (uložení) každé změny vyžaduje sestavení kontrolního součtu, který je reprezentován unikátním SHA-1 otiskem. Jedná se o posloupnost čtyřiceti hexadecimálních znaků[4], tvořících řetězec. Výpočet otisku probíhá na základě obsahu souboru. Vzhledem k integraci podmíněného využívání otisků v nejnižších úrovních Git není možné provést změnu bez vědomí systému. Otisky navíc zajišťují i ověření, že nedošlo k poškození souboru např. při přenosu po síti a systému Git slouží také jako jednoznačné identifikátory v databázi změněných souborů [10]. Usnadnění organizace souborů dosahuje Git pomocí rozdělení hlavního spravovaného repozitáře do více oblastí. Soubor může být v jednom ze tří stavů (změněn na straně klienta, připraven k zápisu nebo zapsán do hlavního repozitáře) a podle toho rozdělen do příslušné oblasti. Primárně Git pracuje s lokálním úložištěm a není nutné se spojovat s centrální (nebo jinou) stanicí, protože všechny informace o změnách jsou uchovávány a aktualizovány na každé stanici průběžně. Klienti pracují na své lokální kopii hlavního repozitáře a změny se zapíší do hlavního repozitáře až v okamžiku potvrzení změn.

[4]Konkrétně jde o znaky (0–9) a (a–f).

2.4.2 Dostupnost systému

Git je možné nainstalovat ze zdrojových souborů[5] a není třeba grafického uživatelského rozhraní, protože veškeré informace mohou být získány a zobrazeny pomocí příkazového řádku[6]. Systém je dostupný ve verzích pro většinu operačních systémů (Linux, Windows a MacOS). Aktualizace a další užitečné soubory lze získat z veřejně přístupného repozitáře a to již pomocí samotného systému Git provedením kopie potřebných souborů z hlavního repozitáře na lokální stanici. Po instalaci se nastaví prostředí Git upřesněním konfiguračního umístění v systému, ze kterého budou získávány informace o uživatelích a také založením uživatelského jména a e-mailu (pokud zatím uživatel neexistuje). Uživatelské jméno bude identifikovat autora u všech operací a změn, které budou zaznamenány z lokálního umístění tohoto autora s použitím jeho uživatelského účtu. Vysvětlení nejasností a pomoc s Git přináší nápověda spustitelná i v offline režimu.

2.4.3 Repozitáře

Uživatel Git má možnost repozitář (oblast sledovaných dat) vytvořit dvěma způsoby. První možností je zahájení sledování oblasti dat (příkazem `git init` v adresáři s projektem) a druhá varianta spočívá ve vytvoření kopie již existujícího Git repozitáře[7] z jiného místa v síti do místního Git systému (příkazem `git clone`). Zmíněný druhý způsob sice není rychlý, protože se najednou kopíruje celý obsah vzdáleného repozitáře, ale je důležitý pro následnou možnost zálohy vzdáleného repozitáře pomocí místního. Od okamžiku zavedení repozitáře zaznamenává Git pouze změny (přírůstky), které uživatel připravil k zápisu a následně jejich zápis také potvrdil (příkazem `git commit`). Souhrn zaznamenávaných změn označujeme pojmem revize, nebo anglicky „commit“. Pokud došlo ke změně v souboru, obsahuje revize i informace o změnách ve znacích souboru a jejich počet. Rozlišují se přidané (add) a odstraněné (delete) znaky. Adresář s repozitářem Git obsahuje i složku vytvořenou automaticky při založení repozitáře a pojmenovanou `.git`, kam Git ukládá záznamy o revizích[8] a další důležité informace o repozitáři. Rozsah sledovaných dat v repozitáři může uživatel ovlivnit podle potřeby vyloučením souborů, které nemají být sledovány. Jejich seznam se ukládá do souboru `.gitignore`, který podporuje zápis i formou regulárních výrazů[9]. Nejčastěji se vyloučení aplikují na automaticky generované soubory nebo logy (např. vytvářené sestavovacím systémem překladače).

2.4.4 Koncept větvení

Vyšší přehlednost při práci na více místech repozitáře přináší větve. Kdykoli můžeme v Git vytvořit novou větev, která bude vnímána jako nová reference na některou z revizí repozitáře. Revize, na kterou větev aktuálně ukazuje, lze změnit (přemístěním ukazatele příkazem `git checkout`) tak, aby větev ukazovala na jinou požadovanou revizi. Nově založený repozitář má standardně jen jedinou (hlavní) větev s výchozím názvem `master`. Smyslem práce nad konkrétní vedlejší větví je, že neovlivní práci a obsah práce na hlavní větvi repozitáře.

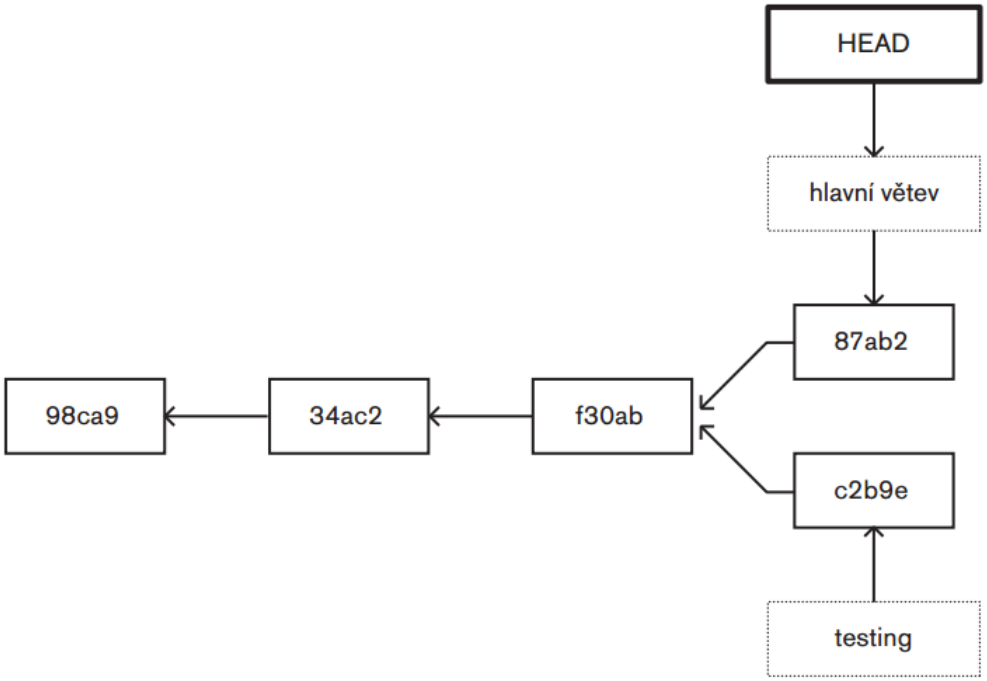
[5] Existují balíky určené pro konkrétní operační systém (dostupné na http://git-scm.com/download).

[6] Podmínkou je přítomnost knihoven `curl`, `zlib`, `omenssl`, `expat` a `libiconv`, které Git využívá a je zapotřebí knihovny nainstalovat před samotnou instalací Git.

[7] Kopírování tímto způsobem se nazývá klonování.

[8] Záznamy se skládají např. z informací o autorech změn, měněných souborech, rozsahu a konkrétních seznamech změn, data atd.

[9] Regulární výraz (používaný v unixovém rozhraní) je textový řetězec, který může obsahovat znaky speciálního významu [23].



Obrázek 2.2: Větvení práce [10]

Větvení popisuje obr. 2.2, kde uživatel v době poslední zaznamenané revize f30ab vytvořil novou větev s názvem testing (nová revize c2b9e) a práce v hlavní větvi (revize 87ab2) pokračuje samostatně. V okamžiku potřeby přidání práce k hlavnímu repozitáři se provádí tzv. sloučení větví (příkazem `gitmerge`). Informaci o tom, která větev je aktuální a nad kterou se aktuálně pracuje, Git zjišťuje pomocí ukazatele zvaného `HEAD`[10]. Nově vytvořený repozitář má ukazatel nastaven na jedinou větev `master`. Každá větev (kromě první `master` větve) obsahuje i informaci o svých předcích a orientace ve struktuře probíhá za použití odkazů na předky větví. Operace větvení se vyznačuje rychlým provedením, protože Git oproti jiným VCS nekopíruje žádné soubory, ale pouze vytvoří nový soubor reprezentující větev, obsahující kontrolní součet SHA-1 (snímek) revize. Git na úrovni příkazového řádku nabízí vizualizace větví v repozitáři fomou grafu (příkazem `git log --graph`) složeného ze znaků příkazového řádku symbolizujících propojení větví a okamžiky významných změn.

Začlenění změn z jedné větve do větve jiné nazýváme slučování větví. Situace může být složitější za předpokladu více než jednoho předchůdce u začleňované revize. V takovém případě Git určí nejvhodnějšího společného předka a použije ho jako základnu pro sloučení. Jiné systémy správy verzí vždy nepodporují automatické dohledání předka a proto uživatel u některých systémů musí vhodného předka (revizi) ručně uvést [10]. Další problémy mohou nastat u změn, které proběhnou na stejných místech souborů odlišně. Systém neví, které změny zaznamenat v nové revizi a které naopak ne. Konflikty řeší sám uživatel výběrem jen některých změn, nebo úpravou změn tak, aby zahrnovaly co nejvíce úprav a přitom nezpůsobovaly konflikty. Výpis konkrétních změn v souborech uživatel získá pomocí některého nástroje pro slučování (např. `kdiff3`, `tkdiff` nebo `opendiff`) a to podle toho, který má Git nastaven jako výchozí. Po skončení úprav ve slučovacím nástroji se Git dotáže uživatele, zda

[10] Ukazatel se posouvá automaticky s každou revizí.

konflikty vyřešil a pokud uživatel potvrdí vyřešení, připraví Git k zápisu i vyřešené soubory bez konfliktů. Před přidáním lokálních změn do vzdáleného repozitáře je nutné provést nejprve načtení změn ze vzdáleného repozitáře (příkaz `git fetch`) k lokální stanici a až poté připravit lokální změny k zápisu, protože mohlo na vzdáleném repozitáři dojít i k jiným změnám a neaktualizované části by způsobovaly konflikty. Git dokáže řešit konflikty také procesem zvaným přeskládání, které zahrnuje provedení změn zapsaných na jedné větvi a také na větvi jiné. Přeskládání není doporučená operace nezkušeným uživatelům, protože dojde k zahození existujících revizí a vytvoření nových, což může za určitých okolností způsobit problémy[11]. Existují i odkazy na stav větví pocházejících ze vzdálených repozitářů. Takové odkazy nazýváme vzdálené větve a plní funkci záložek umístění větví ve vzdálených repozitářích při posledním připojení. Ačkoli mají vzdálené větve pro uživatele charakter lokálních větví, nelze je přesouvat a k přesunům dochází automaticky po síti.

Některá místa v historii změn repozitáře mají pro vývojáře velký význam (např. dokončení verze vyvíjeného software). Git obsahuje funkci k jejich označení (příkaz `git tag`) prostou nebo anotovanou značkou [10]. Prosté značky mají podobu větve, která pouze ukazuje na konkrétní revizi a nemění se. Anotovaná značka obsahuje údaje o jejím autorovi a podporuje navíc ověření programem GNU Privacy Guard[12].

2.4.5 Získání informací o repozitáři

Informace o revizích nad repozitářem lze ze systému Git získat pomocí příkazu `git log`. Výstupem je seznam záznamů o revizích. Samostatný příkaz `git log` vypíše všechny záznamy od nejnovějších až po nejstarší, kde uvede u každého záznamu SHA-1 otisk zkrácený na sedm znaků, autora, datum a zprávu (poznámku) zapsanou k revizi. Když potřebujeme omezit počet výsledků, nebo se zaměřit na konkrétní revize nebo informace z nich (např. pouze konkrétní autory, soubory nebo data), příkaz `git log` podporuje velký počet přepínačů, které omezují nebo formátují výstup. Kromě `git log` disponuje Git i příkazy pro zjištění informací o nastavení repozitáře, sledovaných nebo ignorovaných souborech, uživatelích, všech vzdálených repozitářích atd. Výstupy příkazů `git log` a dalších jsou základními vstupními daty pro aplikaci, kterou v bakalářské práci navrhuji a implementuji. Použité příkazy komentuje kap. 4.2.

2.4.6 Existující statistické nástroje nad Git

Bylo vytvořeno mnoho aplikací, které jsou schopny generovat statistické informace o Git repozitářích. Největší odlišnost u existujících nástrojů spočívá v typech statistik, které dokážou vytvořit. Některé sledují pouze commity, jiné i soubory, uživatele, tagy a další. Odlišnosti jsou také v uživatelském rozhraní, požadavcích na běh, licencích a zda je poskytována aktuální podpora ze strany autorů nástroje. Jednodušší z pohledu použití jsou nástroje, které lze spustit přímo (např. nástroje implementované jako webová stránka) a bez nutnosti předchozí instalace nebo překladu. Samotný Git systém zahrnuje pouze jednoduchý výpis informací o změnách pomocí příkazového řádku, zmíněný v kap. 2.4.5. Některé z nástrojů jsou součástí větších služeb (nebo programů) pracujících nad systémem Git. Výhodou pak je, že uživatel s používáním takové služby získá kromě systému správy verzí i statistické funkce. Příkladem je služba *GitHub*, která umí formou grafu zobrazit statistiku commitů,

[11] Nedoporučuje se přeskládávat revize odeslané do veřejných repozitářů (vzniklo riziko změn na vzdálené straně) [10].

[12] GPG je program schopný data šifrovat a dešifrovat, elektronicky podepisovat, zpětně ověřovat podpis a realizovat správu šifrovacích klíčů.

intenzitu práce autorů nebo frekvenci změn. Grafy jsou doplněny schématy závislostí (vazba na jiný repozitář) a kdo si repozitář stáhnul [5]. Podobnou webovou službou je *Bitbucket*, která dokáže vykreslit graf statistiky commitů a součet dle dne v týdnu (rozděleného na hodinové intervaly) v období od prvního commitu do aktuálního okamžiku. Statistické funkce je nutné v Bitbucket nejprve nainstalovat [2]. Ostatní komentované nástroje byly vytvořeny výhradně ke statistickým účelům.

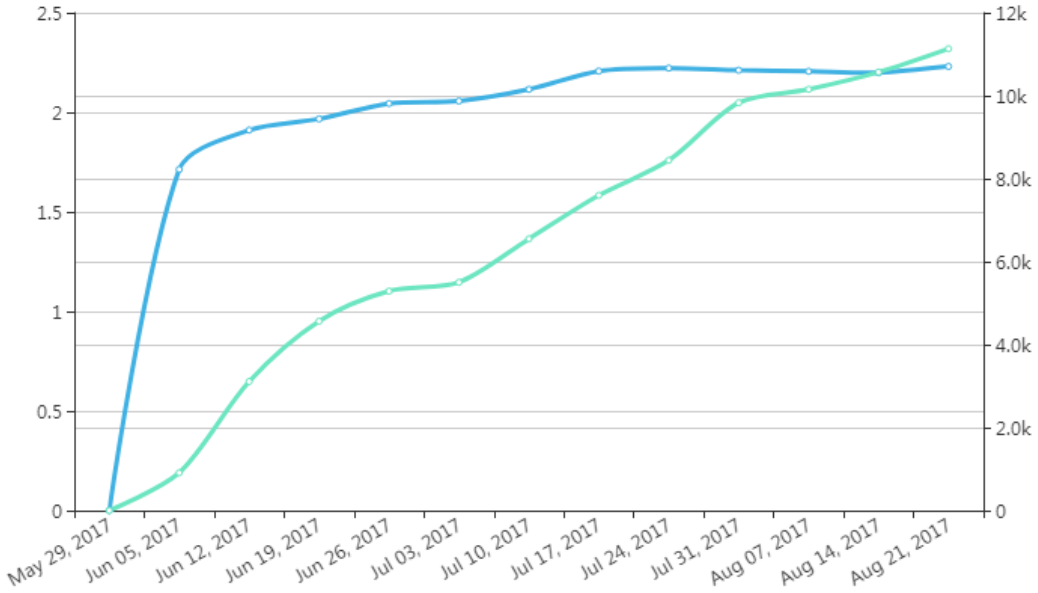
GitStats

Nástroj GitStats patří mezi starší statistický nástroj, který je třeba před použitím nainstalovat a poté spustit nad repozitářem. Výsledkem je HTML (nutno ručně otevřít vytvořený HTML soubor) stránka obsahující informace a grafy v oddělených kartách. Generování probíhá u větších repozitářů s tisíci a více commity pomaleji (v řádech minut) a výsledek zahrnuje informace o commitech, autorech, souborech (četnost a typy), počty add, delete a souhrn k repozitáři [3].

Gitential

Kvalitním analytickým nástrojem je Gitential. Dokáže importovat data z GitHub i Bitbucket (včetně SSH podpory) a dovoluje uživateli vybrat, které konkrétní repozitáře zahrnout do statistik. Výhodou Gitential je nápověda dostupná ke každé vyobrazené statistice (vysvětlivky k hodnotám a jednotkám ve statistikách) a tím minimalizace nesprávné interpretace uživatelem. Přehlednost grafů zvyšuje možnost zobrazit pouze vybrané podgrafy (žádný, všechny, nebo jen některé), takže k vykreslení více typů dat podobného charakteru a stejného rozsahu alespoň jedné osy postačí jedna plocha grafu. Statistiky obsahují data o commitech, autorech a souborech (jejich typ). Navíc nástroj poskytuje graf poměru produktivní a neproduktivní práce (kódu), který vypočítá podle délky doby mezi vytvořením souboru a prováděním změn v něm. Krátký časový interval znamená neproduktivní práci. Další statistikou je produktivita autorů vycházející z průměrného počtu hodin práce autora a průměru editovaných řádků, kde více řádků za kratší časový úsek znamená vyšší produktivitu. U autorů zobrazuje pracovní odchylky, vytížení a vzájemné propojení dle spolupráce. Gitential dále zobrazuje podíl řádků implementačních a testovacích souborů (obsahujících v cestě k souboru nebo názvu řetězec „test“). Uživatel může omezit výběr dat pro statistiku určením repozitáře, autorů, data (rozsah) a podřetězcem ze zprávy commitů. U výběru dat usnadňuje aplikace výběr nabízenými intervaly (den, hodina, měsíc, atd.) [9]. Nedostatek představuje volba typů grafů (rozsahu), která není u některých statistik optimální, protože dochází k překrývání zobrazených hodnot (především krabicové a bublinkové grafy) a grafy obsahující barevně odlišená data využívají opakovaně barvy stejného odstínu pro odlišné prvky (vhodnou volbu barev objasňuje kap. 4.3.2). Gitential vyhodnocuje oproti jiným nástrojům i stav repozitáře (projektu) a tím nám sděluje, jakým směrem se práce v repozitáři ubírá (rozvoj nebo údržba po dokončení hlavních částí). Tyto statistiky nelze jednoduše vyhodnocovat, protože zohledněná metrika (délka kódu, počet řádků, počet souborů nebo autorů atd.) nemusí jednoznačně vypovídat o stavu projektu. Kód píší vývojáři v různých jazycích a styl jejich práce se liší [16]. Statistika kódové složitosti u Gitential vychází z měření hloubky zdrojového kódu, kterou můžeme změřit počtem úrovní vnoření do řídicích struktur kódu[13]. Když v kódu dochází k implementaci nových funkcionalit, většinou se za-

[13] Analýzou obsahu souborů se hledají prázdná místa na začátku řádku reprezentovaná odsazením.



Obrázek 2.3: Graf složitosti kódu v Gitential [9]

nořuje více do hloubky. Oproti tomu následná refaktorování[14] kód zjednodušují a snižují tím i hloubku zanoření. Hloubky se mezi změnami souborů porovnají a vypočítá se jejich kumulativní průměr. Výsledný graf zobrazuje směr, kterým se práce v repozitáři ubírá [9]. Příkladem je složitost kódu na obr. 2.3, kde vodorovná osa vyjadřuje čas, levá svislá osa určuje průměrný počet odsazení v řádcích kódu, pravá svislá osa určuje objem kódu počtem řádků, modrá křivka vyjadřuje složitost kódu a zelená křivka jeho objem.

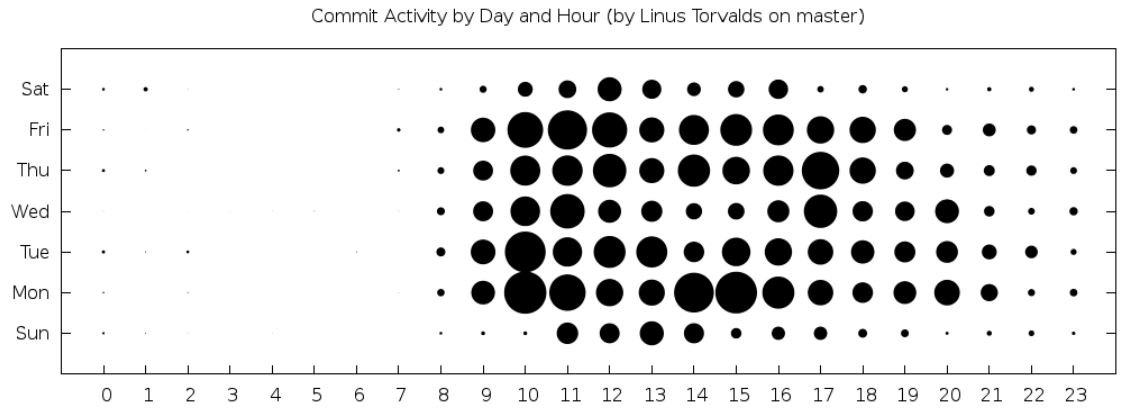
Pepper

Jednoduchým generátorem statistik je nástroj Pepper, který podporuje systémy Git, Mercurial a Subversion. Před použitím musí být nainstalován a pracuje v příkazovém řádku. Aplikace vygeneruje sestavu grafů z informací o aktivitě uživatelů, změnách v souborech, pracovní vytíženosti a ostatní důležité aktivitě [7]. Příklad grafu s aktivitou uživatele Linuse Torvaldse je na obr. 2.4, kde vodorovná osa určuje denní hodinu, svislá osa den v týdnu a černé body intenzitu zaznamenaných commitů za celé období práce v repozitáři. Vzhled grafů působí jednoduše, ale plní svou funkci velmi dobře. Rozsahy (hustota) grafů jsou optimalizovány a nedochází k nevhodnému překrývání popisků os. Pepper vykresluje grafy mnoha typů, což odpovídá charakteru zobrazovaných dat.

GitInspector

Nástroj GitInspector poskytuje (bez nutnosti nástroj před použitím nainstalovat) na výstupu volitelného formátu (HTML, stdout, JSON, atd.) statistiky pro autory, zahrnuje počty commitů, add, delete a jejich poměr. Parametrizace výstupů nástroje je zaměřená na výběr typů souborů k anaýze. Poskytnuté prstencové grafy autorů nástroj optimalizuje, takže části příliš malé k samostatnému zorazení sloučí do jedné části pod označe-

[14] Refaktorování znamená změnit vnitřní strukturu softwaru za účelem snazšího pochopení kódu a zjednodušení oprav [22].



Obrázek 2.4: Graf aktivity z nástroje Pepper

ním „ostatní“. Nevýhoda spočívá v rychlosti aplikace, protože generování výstupu probíhá velmi dlouho a uživatel je prostřednictvím příkazového řádku pouze omezeně informován o průběhů zpracování. Oproti ostatním komentovaným nástrojům docházelo při testování použití GitInspector nad repozitářem obsahujícím znaky odlišných znakových sad k problémům a neschopnosti nástroje statistiku sestavit, protože je závislý na nastavení příkazového řádku, ve kterém běží. Běh nástroje vyžaduje příkazový řádek podporující normu kódování Unicode (až 1 114 112 různých znaků).

Kapitola 3

Návrh nástroje Git reporting tool

Nástroj sestává z databáze pro uložení získaných informací o repozitáři z Git a aplikace s grafickým uživatelským rozhraním. Aplikace se z pohledu funkcí skládá z části zajišťující komunikaci s databází a správu databázových dat, další součástí aplikace jsou funkce generování a zobrazování statistik. Tato kap. popisuje návrh schématu databáze a implementace aplikace je popsána v následující kap. 4.

3.0.1 Návrh databázového schématu

Databáze slouží aplikaci jako centrální úložiště všech potřebných dat, ze kterých aplikace poskytuje po jejich zpracování výstup. Návrh vychází ze znalostí o relačních databázích, kde jsou data uživatelem vnímána ve formě tabulek a kde operace nad daty databáze pracují nad těmito tabulkami. Navržené schéma sestává z tabulek: Repozitář, Autor, Commit, Soubor, Větev, Změna a Commit_větev. Primárním klíčem tabulky Commit je identifikátor commitu (SHA-1 otisk, viz kap. 2.4.1) zkrácený na prvních 7 znaků[1], u ostatních tabulek vlastní číselný identifikátor a tabulka Změny využívá kombinace identifikátoru commitu a souboru (změna je commit nad určitým souborem). Tabulka Commit_větev vyjadřuje náležitost větve ke commitu, protože commit může být zaznamenán v několika větvích. Navržené schéma je na obr. 3.1, kde jsou vyznačeny primární (PK) a cizí klíče (FK).

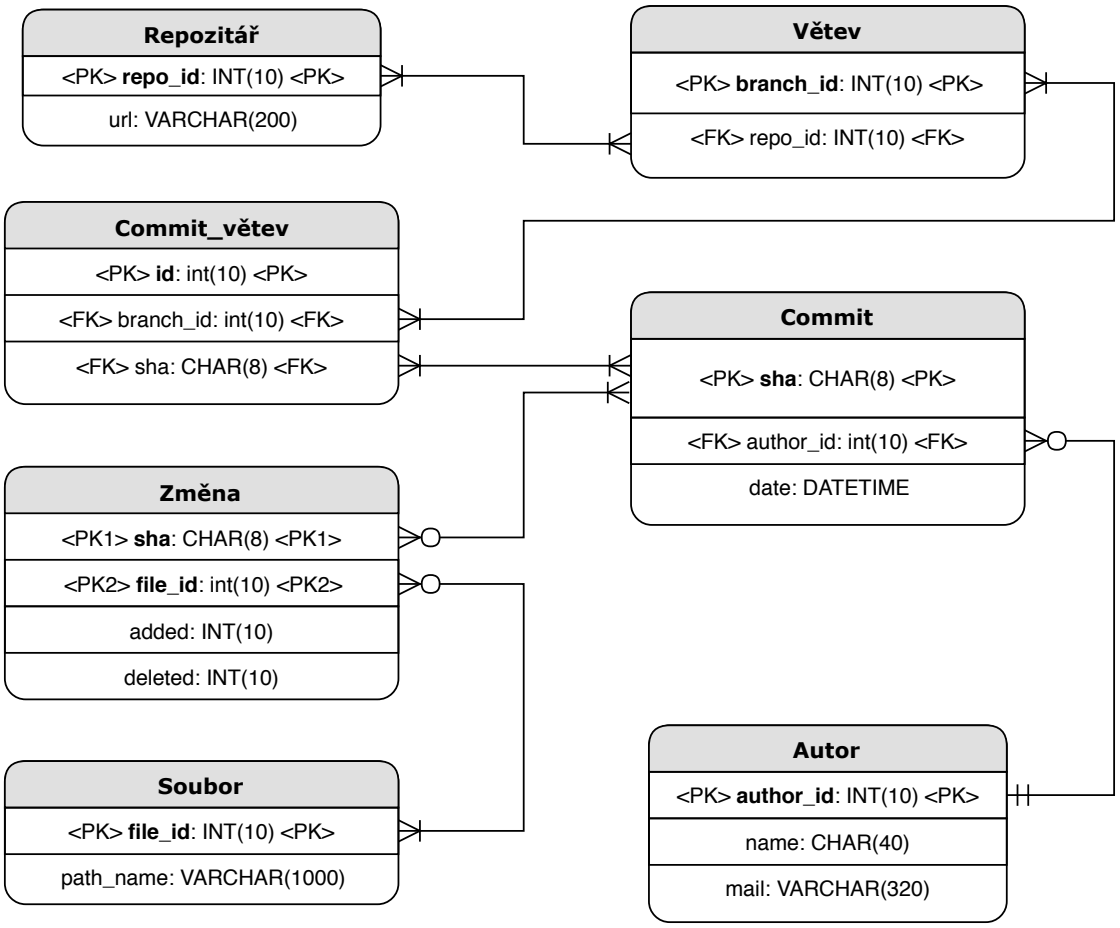
Tabulky navrhuji s ohledem na co nejmenší opakování stejných vstupních hodnot ve sloupcích tabulek. Opakující se data způsobují objemový nárůst databáze a pomalejší provádění operací nad databází. Každý záznam revize musí obsahovat vždy i odkaz na celou URL adresu repozitáře, která ho jednoznačně identifikuje. Podobná situace nastává u jmen souborů, autorů a větví.

Kardinalita v navrženém schématu není všude stejná a uvažované typy odpovídají charakteru dat v databázi z pohledu informací o změnách v repozitáři. Z tohoto důvodu například každý autor v databázi vytvořil alespoň jednu revizi, protože databáze uvažuje osoby přistupující do repozitáře až od okamžiku, kdy provedou nějakou změnu. Pouhé naklonování nečiní z uživatele autora. Ve vztahu větví a počtu revizí vyjadřuji i vztah větve bez revizí, kterými jsou nově vytvořené větve do okamžiku zapsání první revize na této větvi. Vytvoření větve nepovažuji za revizi, protože nejde o zásah do struktury sledovaných dat.[2].

Pro práci s databází využívám volně šířitelný systém řízení báze dat MySQL založený na jazyce SQL [19].

[1] Informace o unikátnosti při použití pouhých 7 znaků potvrzuje samotný autor Git, Linus Torvalds [4].

[2] Git rozlišuje u uživatelů mezi pojmem autor a tzv. „commiter“. Autor je ten, kdo změnu původně sestavil a napsal, oproti tomu „commiter“ je uživatel, který úpravu aplikoval jako poslední [1].



Obrázek 3.1: Databázové schéma

Kapitola 4

Implementace nástroje Git reporting tool

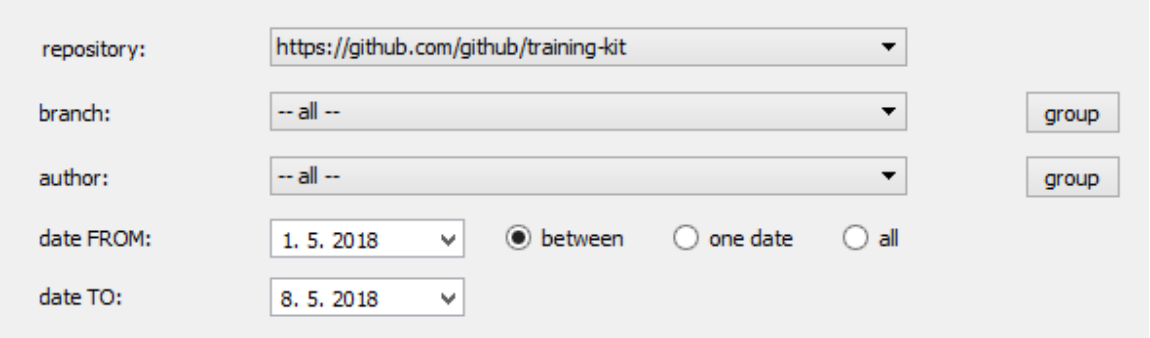
Hlavní část nástroje je naprogramovaná v jazyce Python (verze 3.6) s využitím objektově orientovaného konceptu. Programovací jazyk byl zvolen s ohledem na jednoduchou syntaxi, podporu správy využité paměti za běhu a silnou podporu pro zpracování struktur. Python disponuje i dynamickou běhovou kontrolou typů a mezí indexů (zejména v cyklech) a tím zajišťuje bezpečnější běh programu. Použití přídavných modulů v Pythonu, kterých existuje velmi mnoho, s cílem vytvořit flexibilní aplikaci, je velmi snadné a lze snadno ověřit, zda jde o oficiální modul, nebo modul vydaný jiným uživatelem[1]. Interpret Pythonu může být použitý interaktivně, bez nepřímého kompilování a linkování, což usnadňuje experimentování a vývoj programu. Podpora mnoha programovacích paradigmat (procedurální, objektové, atd.) dovoluje použití i při slabší praktické zkušensti s objektově orientovaným programováním [20]. Vzhledem k distribuovanému charakteru Git a různorodosti autorů v repozitáři dochází k záznamům (např. zpráva v commitu) v různých znakových sadách (jazycích) a správné kódování závisí na možnostech programovacího jazyka. Vhodným řešením kódování je použití normy Unicode. Python od verze 3.x[2] umožňuje řetězce implicitně ukládat jako typ Unicode a není třeba při zpracování řetězce striktně převádět.

4.1 Uživatelské rozhraní

Nástroj uživatel ovládá prostřednictvím grafického uživatelského rozhraní ve formě oken a program interaguje s uživatelem v případě potřeby pomocí výzev (např. zadání požadované vstupní hodnoty). Rozhraní je naprogramováno pomocí aplikačního rámce Qt s využitím modulu `PyQt4` jako doplňku programovacího jazyka Python. Zvolil jsem Qt verzi 4.8.7, protože jde o déle používanou, testovanou a dostačující verzi. Základními komponentami rozhraní jsou okna, tlačítka, informační lišta a přepínatelné karty obsahující grafy, tabulky a informace. Nástroj po spuštění vyzve uživatele pomocí dialogu k zadání přihlašovacích údajů k databázi (jméno, uživatel, heslo) a zda požaduje pouze přihlášení, nebo i vytvoření nové databáze. Vytvoření databáze vyžaduje znát cestu k metadatům repozitáře (složka .git) a její uvedení může uživatel provést pomocí okna průzkumníka souborů. Zakládání

[1] Neoficiální moduly nemusí být dostatečně otestovány a riziko obsažených chyb je vyšší než u oficiální distribuce.

[2] Oproti verzi 2.x, kde jsou řetězce implicitně typovány jako řetězce ASCII standardu (až 128 různých znaků).



Obrázek 4.1: Panel výběru dat

databáze trvá určitou dobu a nástroj uživatele informuje o stavu vytváření prostřednictvím stavové lišty. Při úspěšném přihlášení k databázi (nebo jejím založení) nahradí dialogové okno hlavní, které obsahuje ovládací tlačítka, panel parametrů pro výběr při požadavku na vlastní statistiku (viz obr. 4.1), pás karet obsahujících statistiky a informační lištu ve spodní části okna. Panel parametrů se skládá z volby data a výběrových seznamů pro větve a uživatele. K výběru několika položek najednou slouží tlačítka „group“ u výběrových seznamů pro větev a autora.

4.2 Správa databáze

Před sestavením prvních výstupů je zapotřebí založit zdrojovou databázi (pokud již není pro daný repozitář založena) a připojit se k ní. Obsluhu databáze zajišťuje třída `MyDatabase` (její metody). Bez sestavení platného připojení k databázi nelze získat žádné výstupy, přestože program lze spustit. Stav připojení je zobrazen na informační liště. Uživatel zvolí jméno databáze, ze které se sestaví statistiky nebo která bude vytvořena. Program rozlišuje požadavek na vytvoření databáze a připojení, což musí uživatel vybrat a program neprovádí rozhodnutí automaticky. Následně dojde k pokusu o vytvoření databáze a při úspěchu začne nástroj načítat zdrojová data z Git, která ukládá do tabulek databáze. Obslužné operace s databází provádí nástroj pomocí modulu MariaDB[3]. Existuje mnoho modulů podporujících MySQL a modul MariaDB byl vybrán s ohledem na jeho aktuální podporu. Většina dat o repozitáři obsahuje informaci o čase, přičemž MariaDB zahrnuje mnoho funkcí (viz kap. 4.3.2, odstavec věnovaný rozsahu) pro práci s časem a tím zpracování usnadňuje a zpřehledňuje. Zdrojem dat z Git je nejen příkaz `log`, ale pro zajištění efektivity i `ls-remote`, `rev-list` a `branch`. Požadovaného výstupu informací lze dosáhnout i více způsoby, ovšem některé nemusí být efektivní[4]. Program informuje uživatele o průběhu vytváření databáze prostřednictvím informační lišty s přibližným uvedením podílu zpracovaných dat. Po úspěšném vytvoření databáze, nebo připojení k existující, pracuje program výhradně s daty získanými z databáze a není třeba dalších operací přímo nad systémem Git (pochopitelně

[3]Modul vyvíjí společnost MariaDB Corporation Ab, MariaDB Foundation [8]. Předchůdcem MariaDB byl systém řízení báze dat MySQL u kterého po přesunu pod jinou vývojářskou společnost došlo k pochybnostem o dalším osudu systému a proto hlavní vývojář Michael Widenius navázal vývojem pod novým jménem MariaDB. Díky tomu došlo k udržení licence svobodného softwaru GNU GPL.

[4]Příkladem je použití příkazu `git log` s argumentem `--reverse` oproti `git rev-list`, protože obě vypíší seznam všech commitů v pořadí od nejstarších k nejnovějším, ale první trvá značně déle v závislosti na počtu commitů (nejprve získá všechny commity a teprve poté aplikuje omezující pravidla).

s výjímkou atualizací databáze). Dotazy aplikované programem na kurzor[5] databáze poskytují na svém výstupu pole s položkami představujícími řádky tabulek, které splňují kritéria v zadánému dotazu. Mnohařádkové výstupy lze poté efektivně zpracovat pomocí `for` cyklů.

4.3 Formátování statistik

Vizualizace dat vyžaduje zvolit vhodné struktury a jejich formát. Nevhodné zpracování způsobuje nepřehlednost, nepřesnost, nejednoznačnost a v horším případě i chybný výsledek. Základními prvky statistik nástroje jsou tabulka nebo graf, případně obojí.

4.3.1 Tabulky

Tabulka prezentuje data jednoduše a přímo s minimálními úpravami vzhledu. Tabulky byly zvoleny z důvodu potřeby prezentovat všechny přesné hodnoty s možností seřazení hodnot podle volby uživatele, což některé existující nástroje (popsané v kap. 2.4.6) s tabulkami neumožňují. Tabulky mohou obsahovat i výběrové volby (přítomné zaškrtávací políčko v každém řádku), které nástroj používá pro získání seznamu položek vybraných uživatelem k sestavení volitelné statistiky uvedené v kap. 5.0.4. Řazení probíhá s ohledem na velikost (číselné hodnoty) a řetězce (alfanumerické znaky). Možnost řazení (zestupně nebo vzestupně) poskytuje pouze tabulka se záhlavím, které může být u tabulek před jejich vykreslením skryto (např. tabulka souhrnné statistiky). Když tabulky obsahují mnoho řádků a jejich velikost převyšuje velikost pracovního okna, zobrazí nástroj tabulku s posuvnou lištou. Díky přítomnosti řazení nemusí statistiky obsahovat samostatné seznamy maximálních nebo minimálních hodnot (např. nejaktivnější autor). Řazení je navíc nutné i z toho důvodu, že většina grafů vychází z časové osy a položky nelze automaticky seřadit při generování grafu jinak než dle času.

4.3.2 Grafy

Graf dokáže sdělit hodnoty stejně dobře jako tabulka a navíc lépe vyjádřit i vztahy mezi nimi. Použití grafu vychází z potřeby nástroje vyjádřit se sdělením i průběh, s jakým dochází ke změnám hodnot. Nástroj používá většinu grafů sloupcového typu, které jsou před vykreslením optimalizovány. Sestavení grafu předchází kontrola počtu zdrojových položek (nutné především u volitelné statistiky popsané v kap. 5.0.4, jejíž rozsah volí uživatel) a informuje uživatele oknem se zprávou, pokud není co vykreslovat. Pokud zdrojová data zahrnují méně než tři zaznamenané položky (ležící na vodorovné ose), pak se graf nevykresluje a nástroj vygeneruje pouze tabulku. U malého počtu položek lze snadno rozpoznat rozdíly mezi nimi a grafy sestávající z jedné nebo dvou položek ztrácí smysl. Větší počet položek v grafu vyžaduje správné nastavení formátu sloupců a os. Nejvíce ovliňuje rozměry grafu rozsah vodorovné osy, která představuje u většiny statistik čas. Před generováním probíhá kontrola času zaznamenání prvního commitu a nástroj vygeneruje pouze grafy časových intervalů (rok, měsíc, týden nebo den), jejichž začátek je staršího data nebo roven času prvního commitu[6]. Nástroj používá několika způsobů jak ovlivnit správné formátování – omezuje rozsah, volí vhodnou indexaci os, nastavuje šířku sloupců, užívá vhodných barev a pomocné mřížky.

[5] Kurzor je pojmenovaná oblast kontextu (pracovní oblast) databáze, obsahující informace týkající se aktuálně prováděných databázových příkazů [18].

[6] Aktualizací databáze a znovusestavením statistik může uživatel rozšířit zobrazované časové úseky, pokud přibudou v databázi záznamy s dostatečným časovým rozmezím.

Rozsah

Každá časová statistika sestává z grafu posledního časového úseku (např. poslední měsíc) a grafu za všechny časové úseky (např. všechny měsíce od založení repozitáře). V prvním případě je vodorovná osa omezena počtem vhodně zvolených nejbližších nižších jednotek a ve druhém případě jsou počty seskupeny do stejných časových úseků (např. po měsících všech let od ledna až do prosince, tedy 12 hodnot). Nejnižší zobrazovaná přesnost v řádu hodin u tatistik dne je vzhledem k charakteru práce v Git dostačující, protože i u velkých repozitářů s mnoha autory dochází ke změnám v řádech hodin. Předpoklad nejkratší frekvence vychází z efektivního stylu práce vývojářů v praxi, kteří aktualizují až změny dostatečného rozsahu a po delších časových intervalech [12].

Ke zpracování času a posunu v datech využívá nástroj Python moduly `dateutil`, `calendar` a `datetime`. Objekt typu `datetime.datetime` nese časovou informaci s přesností na mikrosekundy. Takovou přesnost nástroj ve statistikách nepotřebuje a zaokrouhlení probíhá implicitně s použitím vestavěných metod (vynulováním ignorovaných atributů) nebo extrahování pouze potřebné části časového údaje. Posun času bez nutné explicitní kontroly rozsahu měněných atributů objektu `datetime.datetime` (např. den) realizuje nástroj pomocí objektu `datetime.timedelta` reprezentujícího požadovaný časový rozdíl a následného přičtení nebo odečtení od původního objektu `datetime.datetime`. Počet dnů ve statistice posledního měsíce je vždy stanoven na 30 dnů, ostatní jednotky dle kalendáře. Odlišnost zpracování statistiky posledního časového úseku a statistiky souhrnu za všechny časové úseky spočívá v tom, že nástroj u souhrnu využívá (kromě Python modulů) i vestavěných funkcí na úrovni SQL dotazů. Použité funkce SQL (`YEAR`, `MONTH`, `WEEK` a `DAYOFYEAR`) získávají počet požadovaných jednotek z časového údaje, které lze v dotazu `SELECT` seskupit aplikováním `GROUP BY`. Z připravených databázových záznamů sestaví nástroj graf a před jeho zobrazením upraví popis os.

Indexace

Modul `matplotlib` poskytuje podporu pro úpravu os s časovými hodnotami. Základem jsou metody (např. `matplotlib.dates.MonthLocator` pro indexy měsíců) lokalizující indexy zvolené jednotky. Všechny zaznamenané indexy jsou rozděleny na hlavní a vedlejší, protože časové úseky obsahující přechod vyššího řádu vyžadují doplňující informaci pro jednoznačné určení času (např. hodnota roku v přechodu měsíce prosince a ledna). Použití hlavních a vedlejších indexů dovoluje zlepšit formát popisků zobrazením dodatečné informace pouze v místě potřeby a ostatní indexy obsahují kratší popis. Samotný formát popisků volí nástroj převážně číselný s upřednostňováním zkratek pro označení názvu měsíců (Jan–Dec), nebo dnů v týdnu (Mon–Sun) získaných pomocí převodů. Časový údaj je v databázi reprezentován typem `date` a při potřebě převodu na požadovaný formát lze v `SELECT` dotazech použít funkci `DATE_FORMAT`, nebo na úrovni Python modulu `datetime` použít metodu `datetime.datetime.strftime`[7]. Statistiky bez časových údajů (např. typy souborů) nástroj formátuje přímo sestavením výčtu položek reprezentujících popisky os. Svislá osa vyjadřující četnost je formátována pouze vynucením zobrazení celočíselného formátu, protože v některých situacích modul `matplotlib` zobrazuje desetinná místa i u celých čísel. Výšku grafu přizpůsobuje modul automaticky s ohledem na největší zaznamenané hodnoty. Některé praktiky vizualizace dat využívají v použitelných případech možnosti změny měřítka

[7] Je třeba dávat pozor na odlišnosti formátovacích stejných specifikátorů, protože např. specifikátor minut má v MySQL podobu `'%i'`, ale v Python modulu `datetime` podobu `'%M'`.

osy (tzv. podseknutí) pro úsporu místa, tím však může docházet k nesprávnému vnímání výsledku a proto takové formátování nepoužívám.

Šířka sloupců

Šířka sloupců by měla být zvolena takovým způsobem, aby mezi jednotlivými sloupci byla mezera a sloupce nesplývaly. Mezery mezi sloupci by pro dobrou čitelnost měly být rozlišitelně menší nebo větší, než šířka sloupců. Stejná šířka mezer a sloupců může negativně ovlivnit čitelnost grafu v místech vyššího počtu sloupců a tím způsobit i optický klam vzhledem k indexům s nulovou hodnotou (absence sloupce). Optimální poměr šířky mezery je poloviční hodnota šířky sloupců [14].

Barva

Vnímání vykresleného grafu ovlivňují i použité barvy. Barvu tvoří dle Munsellova systému tři základní prvky: sytost, odstín (např. červená) a jas (vnímaná světlost) [15]. Menší sytost činí barvu více jemnou, v pastelových tónech a v nejmenší sytosti přechází barva v šedý odstín. Proto je lepší volit v grafech barvy menší sytosti, aby v případě potřeby bylo možné důležitá data zvýraznit užitím vyšší sytosti. Prvotní použití plné sytosti znemožňuje zvýraznit některá data a jedinná možná změna sytosti (snížení) způsobuje opačný efekt[8]. Odstín barev vnímá každý člověk jiným způsobem a více odstínů použitých v jedné skupině dat vytváří odlišnost mezi nimi. Z tohoto důvodu je nežádoucí aplikovat na stejnou skupinu dat různé odstíny a místo toho použít raději různou sytost jednoho odstínu. Pro snadnou čitelnost grafu se volí taková barva pozadí, která je konstantní a působí s barvami dat dostatečně kontrastně [13]. Nástroj používá ve všech grafech pro datové řady modrou barvu přibližně střední sytosti, pro pozadí bílou, popisky os černou a ostatní části grafu (mřížka, osy) sytější šedou. Výjimku tvoří souhrnná statistika s úsečkou červené barvy (viz obr. 5.2) a statistika souborů (kap. 5.0.3) sestávající z různých dat v jednom grafu, kde údaje „add“ reprezentuje modrá barva a počty „delete“ červená (viz obr. 5.3 reprezentující celkové počty add a delete v jednotlivých měsících). Grafy spojnicového charakteru používají stejných barev jako sloupcové grafy.

Snadnost formátování ovlivnila provedenou volbu typu grafů, které nástroj generuje. Byly uvažovány koláčové grafy, dobře zobrazující vzájemný poměr dat, ale problém nastává u popisu většího počtu malých odlišných položek (výsečí) a výběru barevného schématu. Koláčové grafy jsou názorným příkladem, kdy metody vestavěných modulů (např. `matplotlib`) nemusí dostačovat, protože barvy jsou metodami vybírány s málo nebo příliš odlišnými odstíny. Výsledek by vyžadoval oproti sloupcovým grafům hlubší optimalizace.

[8] Vedlejší výhodou změny sytosti a zachování odstínu je dobrá čitelnost při převodu do černobílé barvy, nebo odstínů šedi, což se často používá při tisku.

Kapitola 5

Výstupní statistiky implementovaného nástroje

Hlavní účel nástroje je poskytování statistik a doplňujících informací uživateli. Téměř každý graf doplňuje informační tabulka s hodnotami zahrnutými v grafu, aby uživatel mohl přehledně vyhledat přesné hodnoty bodů v grafu bez nutnosti podrobného zkoumání. Všechny statistiky nástroj zobrazuje na přepínatelných kartách v prostoru hlavního okna, které jsou pojmenovány tematicky podle typu statistiky: souhrnná, commity, statistika autorů, statistika souborů a volitelná statistika. Statistika commitů sleduje pouze jejich počet a proto není blíže komentována.

K sestavení statistik (kromě volitelné) dochází najednou při spuštění nástroje, po připojení k databázi, případně po vyžádání znovusestavení statistik ze strany uživatele. Všechny statistiky uvažují datum poslední aktivity jako datum posledního commitu, takže datum spuštění nástroje parametry statistik nijak neovlivňuje. Časové úseky použité ve statistikách jsou rok, měsíc, týden a den. Nástroj generuje ke každému intervalu statistiku posledního časového úseku (např. poslední měsíc) a také statistiku za všechny časové úseky od počátku repozitáře (např. součty aktivit dle měsíců všech let).

5.0.1 Souhrnná statistika

Statistika souhrnu sestává z tabulky zahrnující základní data o repozitáři (název, datum poslední aktivity, datum založení, atd.). Tabulka nese i informaci o vlastníkovi (autorovi), který repozitář vytvořil a datum vytvoření. V praxi lze uvažovat za okamžik vytvoření repozitáře zaznamenání prvního commitu, ale vždy tomu tak není. Vlastník repozitáře může provést vytvoření časově o mnoho dříve, než nastane první aktivita v repozitáři a především vlastník nemusí být totožný s autorem prvního commitu. Problém zjištění informace o vlastníkovi spočívá v absenci této informace v systému Git. Pouze služba GitHub [17] (viz kap. 2.4.6) zaznamenává informaci o vlastníkovi a tuto informaci lze získat pomocí rozhraní zpřístupňující odpovídající strukturu formátu JSON[1] (položka „owner“). Na obr. 5.1 můžeme vidět, že zakladatel („creator“) a autor prvního commitu („commiter“) se liší. Pokud je repozitář spravován prostřednictvím GitHub, nástroj vyžádá informace z metadat, jinak se za vlastníka považuje autor prvního commitu. Implementačně nástroj rozlišuje situaci, kdy se nelze připojit k síti (získat data ze služby GitHub) a za autora repozitáře pak i v případě GitHub repozitářů považuje prvního autora commitu. Tabulku doplňuje graf počtů

[1] Metadata jsou k dispozici přes `https://api.github.com/repos/autor/repozitar`, kde položky „autor“ i „repozitar“ lze získat z URL repozitáře.

name	training-kit
url	https://github.com/github/training-kit
created	2013-12-20 16:47:46
creator	github
first committed	2013-12-20 16:47:47
commiter	Matthew McCullough
merge	matthew@github.com
commits	3239
last commit	2018-05-14 18:49:41
files	3277
adds	717443
dels	629637
branches	18
average commit time	2 hours, 27 minutes, 11 seconds
authors	100

Obrázek 5.1: Základní tabulka s informacemi o repozitáři

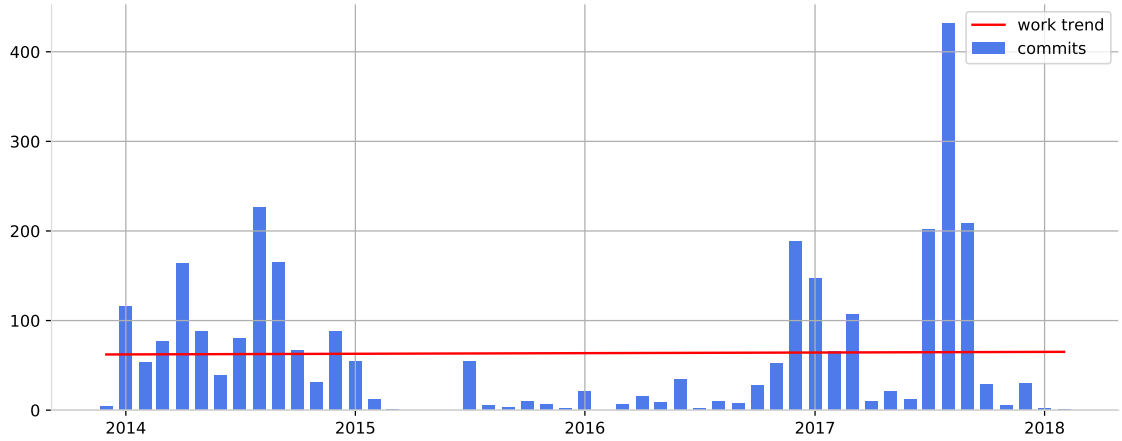
commitů zaznamenaných v celém období repozitáře s vykreslením úsečky trendu odhadující stav živosti repozitáře. Tuto statistiku je náročné sestavit (viz nástroj Gitential v kap. 2.4.6) a protože zdrojem dat je databáze, je použito proložení úsečkou pomocí metody nejmenších čtverců. Klesající úsečka znamená v repozitáři fázi udržování a stoupající pokračující vývoj. Příklad na obr. 5.2 značí vyváženost vzhledem k počtu commitů.

5.0.2 Statistika autorů

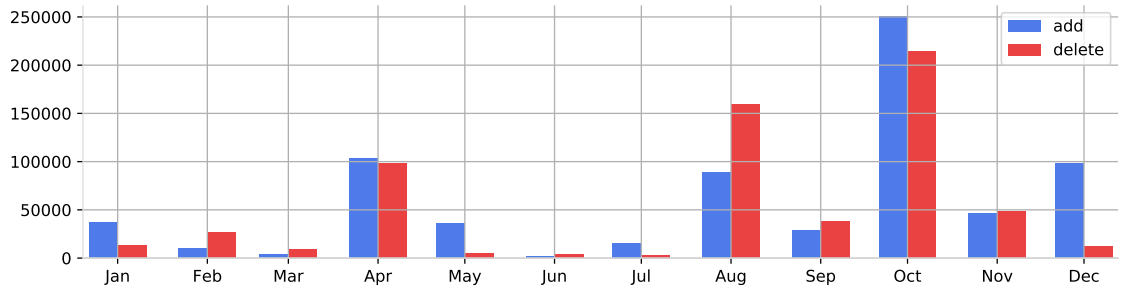
Statistika autorů poskytuje informaci o počtech nově příchozích autorech (sledování prvních commitů autorů). Navíc zobrazuje i tabulkový seznam všech autorů repozitáře s kontaktními a doplňujícími informacemi (počet změn v souborech nebo počet commitů). Znalost informací o příchozích autorech má významný vliv při sledování závislosti počtu autorů na intenzitě aktivit v repozitáři.

5.0.3 Statistika souborů

Statistika souborů zahrnuje statistiku přípon (včetně prázdných) souborů a tím přehled o typech souborů, nad kterými se v repozitáři pracuje. Uživatel nástroje musí brát v úvahu, že Git umoňuje některé soubory v repozitáři ze sledování vyloučit a tím je ignorovat (viz kap. 2.4.3), takže statistika typů souborů nemusí poskytovat absolutní přehled o složení repozitáře. Dále obsahuje podkategorii statistik změn, která představuje počty změn (add, delete) ve všech souborech. Změny jsou v grafech zobrazeny společně, takže uživatel získá přehled o jejich vzájemném poměru (viz obr. 5.3).



Obrázek 5.2: Souhrn commitů s trendem práce



Obrázek 5.3: Změny souborů za poslední rok

5.0.4 Volitelná statistika

Nástrojem automaticky vygenerované statistiky souhrnu, autorů a souborů zahrnují pevná časová období, všechny uživatele a všechny soubory v každé statistice. Praxe vyžaduje sledovat i individuální části repozitáře. Volitelná statistika umožňuje uživateli vybrat časové rozmezí, soubory a uživatele, kteří budou zahrnuti v této statistice, a to i v kombinovaném výběru, takže lze dosáhnout statistik nad skupinou zdrojových dat (např. skupina uživatelů odpovídající reálnému týmu autorů v repozitáři).

Kapitola 6

Testování

Použití a funkce nástroje jsem testoval na několika repozitářích GitHub. Vybral jsem jak repozitáře menšího rozsahu, tak větší se stovkami až tisíci commity a desítkami uživatelů. Nástroj jsem navíc pro zátěžový test spustil i nad rozsáhlým repozitářem Linuxového jádra [6], který zahrnuje stovky tisíc commitů. Výsledky mi poskytly přehled o schopnostech nástroje a jeho citlivých místech. Vybrané repozitáře s autory mnoha národností umožnily otestovat schopnost nástroje správně dekódovat různé znakové sady.

Založení databáze probíhá po potvrzení přihlašovacích údajů k databázi snadno a rychle. Další průběh už závisí na velikosti repozitáře. Jakmile začne nástroj voláním příkazu `git log` získávat informace o repozitáři, dochází u velkých repozitářů k velmi dlouhé časové prodlevě. Doba čekání je podobná jako u přímého volání příkazu v příkazovém řádku (např. pro Linuxový repozitář s odezvou okolo 10 minut). Dochází k seskupování dat před jejich formátováním a poté následuje samotné ukládání do databáze, které trvá nejdéle. Ze sedmi měřených sestavování databáze vychází průměrná doba zpracování přibližně na 720 commitů za minutu. Takovou dobu je nutné čekat pouze při prvním sestavení databáze a následně postačí rychlé průběžné aktualizace, které zahrnují jednotky commitů podle frekvence aktualizování a aktivity v repozitáři. Po úspěšném vytvoření databáze zobrazí nástroj hlavní okno a vykreslí základní sestavu statistik. Doba jejich generování a vykreslování závisí na hustotě dat v grafech, protože značnou dobu stráví modul `matplotlib` formátováním časových os. Pokud časový rozsah commitů dostačuje všem časovým intervalům, které nástroj vyhodnocuje, trvá sestavení a vykreslení nejvyššího možného počtu statistik řádově desítky sekund. Oproti jiným existujícím nástrojům komentovaným v kap. 2.4.6 spočívá výhoda mého řešení v databázi, která je rychlým zdrojem dat pro opakované sestavování statistik. Některé z jiných nástrojů (např. Gitstats nebo Gitinspector) potřebují s každým vytvořením statistiky znovu analyzovat metadata (pomocí `git log`) o repozitáři a tím se použití stává u větších repozitářů neefektivní.

Při implementaci jsem se zaměřil také na testování databázových dotazů. K databázi přistupuji pomocí externího programu HeidiSQL, ve kterém jsem ověřoval výstupy z databáze. Velmi důležitá je volba sloupců v dotazech s agregačními funkcemi `GROUP BY`, jinak dotaz neposkytuje správné výsledky.

Sledováním dat při testování jsem nalezl chybu, která se v některých repozitářích může objevit a nevzniká na straně nástroje. Jedná se o špatný čas, který je společně s informacemi o commitu ukládán do databáze. Příkladem je jeden commit z Linuxového repozitáře, který je staršího data než samotný repozitář[1]. Tuto odlišnost způsobuje špatné nastavení

[1] Jedná se o commit s datem 2001-09-17 v repozitáři, který byl vytvořen 2011-09-04.

data na počítači autora v okamžiku zaznamenávání commitu a může být opravena pouze ručním prepsáním časových informací v commitu. Takové commity, pokud mají značně odlišné datum oproti repozitáři, způsobují ve statistikách neobvykle řídké grafy a statistiky zkreslují.

Kapitola 7

Závěr

Cílem bakalářské práce byl návrh a implementace nástroje poskytujícího statistiky z informací systému Git. Před implementací byly v samostatné kap. popsány základní principy systémů správy verzí se zaměřením na získávání informací z Git a bylo navrženo databázové schéma pro vhodné uložení dat o revizích. Výsledné statistiky prezentuje implementovaný nástroj formou tabulek a grafů. Implementace byla provedena se zaměřením na schopnost aplikace vizualizovat nejdůležitější data a poskytnout uživateli možnosti výběru dat, ze kterých budou statistiky sestaveny.

7.1 Dosažené výsledky

Databázové schéma bylo navrženo tak, aby zahrnovalo všechny informace potřebné pro reprezentaci výstupů a nástroj nemusel opakovaně získávat informace přímo z Git. Nástroj umí vytvořit databázi a naplnit ji daty z Git, nebo se připojit k existující databázi a umožnit její aktualizaci. Prostřednictvím grafického uživatelského rozhraní poskytuje nástroj statistiku počtu commitů, počtu a typů souborů a seznam autorů s počty nových. Nástroj zobrazuje statistiky formou sloupcových nebo spojnicových grafů, které před zobrazením optimalizuje tak, aby byly dobře čitelné. Grafy doplňují tabulky s přesnými hodnotami dat z grafu, procentuálním poměrem hodnoty a umožňují uživateli data seřadit (dle sloupců). Kromě statistik obsahuje výstup i souhrnný přehled informací o repoziráři sestávající z tabulky základních údajů (jméno, autor, datum vytvoření, atd.) a grafu commitů za celé období repozitáře. Pro přibližné určení stavu práce v repozitáři (zda je projekt stále vyvíjen nebo přechází do fáze udržování) zahrnuje graf souhrnu i úsečku vytvořenou pomocí metody nejmenších čtverců. Nástroj umožňuje uživateli vybrat, která data z databáze budou zdrojem pro statistiky a uložit graf do souboru různých formátů.

7.2 Návrh dalších rozšíření

Z poznatků získaných při používání nástroje vyplývá několik oblastí, na které se dá zaměřit při dalším rozvoji. Vzhledem k databázi postrádá aplikace větší dynamičnost, protože do okamžiku další aktualizace informací o revizích z Git může mít uživatel aplikace k dispozici méně záznamů aktivit. Návrh a implementace automatických aktualizací databáze nástroj vylepší. Velký význam má statistika aktuálního stavu repozitáře, zda je projekt ve vývoji, nebo ve fázi dokončování a udržování. Hledání způsobů sestavování takové statistiky vyžaduje prostudovat metody analýzy obsahu repozitáře (především z pohledu re-

fatorizace kódu) a zvyklostí vývojářů, které mohou i bez znalosti skutečného stavu a účelu projektu poskytnout přehled o fázi vývoje. Implementovaná aplikace poskytuje jen základní odhad směru, jakým se práce v repozitáři ubírá (vývoj nebo dokončování), pomocí metody nejmenších čtverců nad počty commitů a proto se nabízí její zdokonalení.

Literatura

[1] Reference Manual [online; 10.5.2018]. https://git-scm.com/docs.

[2] Awesome Graphs for Bitbucket: visualized statistics for Git and Mercurial repositories. 1997, [online; 2.5.2018]. https://blog.bitbucket.org/2015/08/05/awesome-graphs-for-bitbucket-visualized-statistics-for-git-and-mercurial-repositories/

[3] GitStats. 1999, [online; 2.5.2018]. http://gitstats.sourceforge.net

[4] GitHub. 2007, [online; 2.5.2018]. https://github.com/git/git/commit/dce96489162b05ae3463741f7f0365ff56f0de36

[5] GitHub Help. 2007, [online; 3.5.2018]. https://help.github.com/articles/about-repository-graphs/

[6] GitHub torvalds/linux. 2007, [online; 9.5.2018]. https://github.com/torvalds/linux

[7] Pepper. 2007, [online; 7.5.2018]. https://jgehring.github.io/pepper/

[8] About MariaDB. 2009, [online; 3.5.2018]. https://mariadb.org/about/

[9] Measuring software development. 2016, [online; 28.4.2018]. http://blog.gitential.com/2017/11/24/measuring-software-development/

[10] Chacon, S.: *Pro Git*. Praha: CZ.NIC, c2009, ISBN 978-80-904248-1-4.

[11] Charlton, G. M.: Distributed Version Control and Library Metadata. *Code4Lib Journal*, , č. 3, 2008: str. 86, ISSN 1940-5758, [online; 11.3.2018]. http://journal.code4lib.org/articles/86

[12] Cheung, K. W.; Hodaňová, A.: *Vývojářův kód*. Brno: Computer Press, první vydání, 2013, ISBN 978-80-251-3786-4.

[13] Few, S.: Practical Rules for Using Color in Charts. *Visual Business Intelligence Newsletter*, February 2008, [online; 21.4.2018]. http://www.perceptualedge.com/articles/visual_business_intelligence/rules_for_using_color.pdf

[14] Few, S.: Bar Widths and the Spaces in Between. September 2016, [online; 20.4.2018]. http://www.perceptualedge.com/articles/visual_business_intelligence/bar_widths.pdf

[15] G., K. R.: The early development of the Munsell system. *Color Research & Application*, ročník 27, č. 1: s. 20–27, doi:10.1002/col.10002, [online; 21.4.2018]. https://onlinelibrary.wiley.com/doi/abs/10.1002/col.10002

[16] Hindle, A.; Godfrey, M.; Holt, R.: Reading Beside the Lines. In *2008 16th IEEE International Conference on Program Comprehension*, Amsterdam: IEEE, 2008, ISBN 978-0-7695-3176-2, ISSN 1063-6897, s. 133–142, doi:10.1109/ICPC.2008.13, [cit. 8.5.2018]. http://ieeexplore.ieee.org/document/4556125/

[17] Žužak Ivan: Re: GIt repo creator name [e-mailová komunikace]. 2018, [cit. 9.5.2018].

[18] Kroenke, D.; Auer, D. J.; Goner, J.: *Databáze*. Brno: Computer Press, první vydání, 2015, ISBN 9788025143520.

[19] Luboslav Lacko: *Mistrovství v SQL Server 2012*. Brno: Computer Press, první vydání, 2013, ISBN 9788025137734.

[20] Lutz, M.: *Learning Python*. Sebastopol: O'Reilly Media, Čtvrté vydání, 2009, ISBN 9780596158064.

[21] Masters, J.; Blum, R.: *Linux profesionálně*. Brno: Zoner Press, první vydání, 2008, ISBN 978-80-86815-71-8.

[22] McConnell, S.: *Dokonalý kód*. Brno: Computer Press, první vydání, 2005, ISBN 80-251-0849-X.

[23] Skočovský, L.: *Principy a problémy operačního systému UNIX*. Brno: SCIENCE, první vydání, 1993, ISBN 80-901475-0-X.

Příloha A

Zdroje informací k implementaci aplikace

- Python - https://docs.python.org/3.5/
- MySql - https://dev.mysql.com/doc/
- Qt - http://doc.qt.io/qt-4.8/
- PyQt - http://pyqt.sourceforge.net/Docs/PyQt4/
- Matplotlib - https://matplotlib.org/2.2.2/index.html
- HeidiSQL - https://www.heidisql.com
- MariaDB - https://mariadb.org