VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÝ NÁSTROJ PRE ANALÝZU CHYBOVÝCH SPRÁV

WEB-BASED CRASH DATA QUERYING TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MICHAL CYPRIAN**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Mgr. ADAM ROGALEWICZ, Ph.D.**
SUPERVISOR

BRNO 2017

Bachelor's Thesis Specification/19186/2016/xcypri01

Brno University of Technology - Faculty of Information Technology

Department of Intelligent Systems | Academic year 2016/2017

Bachelor's Thesis Specification

For: **Cyprian Michal**
Branch of study: Information Technology
Title: **Web-Based Crash Data Querying Tool**
Category: Software analysis and testing

Instructions for project work:

1. Study the existing querying tools for crash reports.
2. Analyse user needs when dealing with a huge number (hundreds of thousands) of crash reports.
3. Design a tool dedicated to querying crash reports with an attention to sort, filter and join the reports according to various criteria.
4. Develop the designed querying tool as a web-based application on the top of Fedora Analysis Framework.
5. Assess the results and discuss further enhancements.

Basic references:

- Fedora Analysis Framework (home page): http://research.redhat.com/fedora-analysis-framework-faf/
- Socorro, the Mozilla Crash Stats project (home page): https://wiki.mozilla.org/Socorro
- Ubuntu ErrorTracker (home page): https://errors.ubuntu.com/
- Linux Kernel Oops (home page): http://oops.kernel.org/

Requirements for the first semester:
First three items.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Bachelor's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Rogalewicz Adam, Mgr., Ph.D.**, DITS FIT BUT
Consultant: Filák Jakub, Ing., RedHatCZ
Beginning of work: November 1, 2016
Date of delivery: May 17, 2017

L.S.

Petr Hanáček
Associate Professor and Head of Department

Abstrakt

Stovky tisíc chybových hlásení denne je zozbieraných od používateľov operačných systémov Fedora a CentOS prostredníctvom nástroja Automatic Bug Reporting Tool. Po spracovaní sú tieto dáta odosielané na Fedora Analysis Framework server, ktorý má slúžiť vývojárom a správcom jednotlivých komponent operačného systému na analýzu nedostatkov. V súčasnej dobe poskytuje Fedora Analysis Framework štatistické dáta s možnosťou základného filtrovania. Cieľom tejto práce je navrhnúť a implementovať nástroj na triedenie, filtrovanie a spájanie dát, ktorý uľahčí určovanie priority práce vývojárov. Návrh sady nástrojov je založený na analýze existujúcich riešení, nedostatkoch nahlásených v repozitári projektu Fedora Analysis Framework a informácií získaných priamo od používateľov tohoto nástroja.

Abstract

Hundred of thousands crash reports are collected from Fedora and CentOS users every day by Automatic Bug Reporting Tool. Processed data are sent to Fedora Analysis Framework server and used by developers and operating system component maintainers to find common problems. Fedora Analysis Framework provides statistic data and a basic way to filter them. The aim of this thesis is to design and implement tool to sort, filter and join the reports, which will help developers to prioritize their work. The design of this toolset is based on existing solutions, issues reported in Fedora Analysis Framework project repository and information provided by its users.

Kľúčové slová

Fedora Analysis Framework, ABRT, hlásenia o pádoch, kľúčové problémy, analýza dát

Keywords

Fedora Analysis Framework, ABRT, crash reports, core problems, data analysis

Citácia

CYPRIAN, Michal. *Webový nástroj pre analýzu chybových správ*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rogalewicz Adam.

Webový nástroj pre analýzu chybových správ

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána doc. Mgr. Adama Rogalewicza, Ph.D. a Ing. Jakuba Filáka. Všetky literárne pramene a publikácie, z ktorých som čerpal sú uvedené v zozname literatúry.

.....
Michal Cyprian
17. mája 2017

Poďakovanie

V prvom rade by som sa rád poďakoval vedúcemu mojej práce doc. Mgr. Adamovi Rogalewiczovi, Ph.D., ktorý dohliadal na moju prácu z pedagogického hľadiska. Moja veľká vďaka patrí tiež členom tímu ABRT, ktorí boli ochotní konzultovať so mnou riešené problémy po technickej stránke, hlavne Ing. Jakubovi Filákovi a Mgr. Miroslavovi Suchému, za to, že sa so mnou delili o svoje bohaté znalosti v oblasti webových technológií a informačných systémov.

Obsah

Kapitola 1

Úvod

Vždy prítomnou súčasťou sveta softvéru a počítačových programov sú chyby, neočakávané správanie a pády. Vyskytujú sa pri vývoji softvéru, testovaní ale tiež v produkcií. Chyby sa objavujú na všetkých úrovniach systému. Aj v programoch, ktoré boli vyvíjané a ladené veľkými komunitami ľudí po desaťročia, ako napríklad Linux kernel sa ich každý deň vyskytne veľké množstvo.

Dáta o okolnostiach pádu sú veľmi hodnotnou informáciu pre vývojárov, ktorí pracujú na odstraňovaní nedostatkov softvéru. Okrem testovania v rámci vývojového tímu je možné analyzovať chyby, ktoré sa objavili u koncových používateľov. Tento postup môže výrazne pomôcť pri hľadaní príčin nesprávneho správania programu a uľahčiť odstránenie nedostatkov.

Mnoho organizácií, zaoberajúcich sa vývojom a distribúciou softvéru začalo využívať nástroje na automatizované zbieranie chybových hlásení od používateľov. V rámci projektu Fedora, bola vytvorená množina nástrojov na tento účel pod názvom Automatic Bug Reporting Tool (ABRT). Na konci procesu detekcie chýb, odosielania a spracovávania chybových dát je Fedora Analysis Framework (FAF), webová aplikácia, na filtrovanie a analýzu veľkého množstva zozbieraných hlásení. Tento nástroj ponúka štatistický prehľad hlásení a základné možnosti filtrovania, ktoré často nie sú dostatočné keď vývojár potrebuje určiť dôležitosť jednotlivých hlásení a jednoduchým spôsobom sa dopracovať k dátam, ktoré sú relevantné pre jeho komponenty.

Cieľom tejto práce je vyhodnotiť potreby konkrétnych používateľov FAF pri prezeraní veľkého množstva hlásení, porovnať existujúce riešenia a na základe získaných informácií navrhnúť a implementovať nástroje, ktoré vyriešia niekoľko z nedostatkov FAF a urýchlia prácu vývojárov pri výbere problémov, ktorými je potrebné sa zaoberať s najvyššou prioritou.

Kapitola 2

Analýza pádov softvéru

Problematiku analýzy pádov softvéru a odhaľovania ich príčin sa zaoberá Robert B. Grandy vo svojom článku [9]. Nasledujúce podkapitoly 2.1, 2.2 a 2.3 sú voľným prekladom úvodnej časti tohoto článku. V podkapitole 2.4 je popísaných niekoľko existujúcich nástrojov, pomocou ktorých sú analyzované chyby softvéru v rámci určitých komunít a organizácií.

2.1 Úrovne analýzy

Známe príslovie hovorí, že múdry muž urobí chybu len raz, kým hlupák opakuje rovnakú chybu znovu a znovu. Pri aplikovaní príslovia na softvérové zlyhania, sa vývojári softvéru individuálne poučia z chýb, ktoré urobili, často však chýba aplikovanie zmien založených na softvérových chybách na úrovni organizácií.

Jedným z účinných spôsobov ako vyhodnocovať softvérové zlyhania je rozšíriť analýzu nedostatkov z individuálnych prípadov na celé organizácie. Toto riešenie prináša okrem analýzy softvérových pádov aj možnosť vyhodnocovať hlavné príčiny týchto pádov, ktoré je možné využiť na zmeny zaužívaných procesov tak, aby chyby nenastávali opakovanie. Celú podstatu tohoto procesu je možné zhrnúť to nasledujúcich piatich krokov:

- Rozšírenie kolekcie chybových dát tak, aby zahŕňali informáciu o hlavnej príčine. Postupné postupovanie od reakcií na pády k preventívnym zmenám.
- Analýza dát o chybách na úrovni celej organizácie. Nachádzanie opakujúcich sa vzorov v pádoch a odhaľovanie slabín produktov.
- Analýza hlavných príčin na podporu rozhodovania, aké zmeny musia byť uskutočnené. Hlavné príčiny je možné získavať zoskupovaním informácií o pádoch na základe podobnosti. Takto štrukturované dáta umožnia porozumieť príčinám a jednotlivým typom pádov.
- Aplikácia získaných poznatkov na školenie ľudí a vykonanie potrebných zmien v procesoch vývoja a údržby.
- Rozvinutie analýzy pádov a hlavných príčin na dlhodobý a efektívny proces vylepšovania softvérových produktov.

Jednou z populárnych metód analýzy činnosti organizácie je hodnotenie procesov. Väčšina hodnotení dokumentuje odpovede respondentov na subjektívne otázky, vytvorené na základe určitého modelu ideálnych postupov vývoja softvéru. Ak je tento model validný

a odpovede ľudí zodpovedajú realite, model poskytuje dobrý obraz stavu organizácie. Teda výsledky môžu a nemusia byť aktuálne a motivačné.

Kombinácia analýzy zlyhaní a ich hlavných príčin je potenciálne cennejšia ako subjektívne hodnotenie, pretože vyjadruje závažnosť zlyhaní pre špecifickú organizáciu. Kľúčovým bodom je fakt, že chybové dáta sú najdôležitejším z dostupných zdrojov informácií na rozhodovanie o forme procesu zlepšovania softvéru. Navyše tieto dáta poskytujú možnosť objektívne posúdiť výsledný efekt po aplikovaní zmien.

2.2 Reakcia na chybové dáta

Po prvotnej analýze je možné na správy o pádoch reagovať rýchlou opravou alebo môžu byť ignorované. Nespokojnosť používateľa je minimalizovaná ak je problém, ktorý znemožňuje jeho prácu alebo činnosť rýchlo opravený. Alternatívnym postupom je ignorovanie dát o vyskytnutej chybe alebo oveľa pomalšia reakcia. Ignorovanie chybových dát môže priniesť významné následky a negatívne ovplyvniť činnosť organizácie dodávajúcej softvérové produkty.

Odpoveď na objavené chyby softvéru by nemala spočívať iba v bezprostrednej reakcií pomocou rýchlej opravy. Okrem ohrozenia spokojnosti zákazníkov a zvyšovania nákladov, je tu niekoľko dôsledkov, ktoré môžu nastať ak bezprostredné reakcie nie sú nasledované krokmi na eliminovanie zdrojov odhalených chýb:

- Ľudia si môžu zvyknúť na reaktívny spôsob uvažovania, čo prináša domnenie, že manažment považuje distribuovanie produktov s chybami za prípustné.
- Manažment primárne oceňuje reaktívne správanie, avšak opravy nutné na odstránenie chýb v neskorších fázach vývoja alebo po vydaní softvéru sú nákladné.
- V reaktívnom prostredí vyjadrujú ľudia rýchlo svoju nespokojnosť s prácou jednotlivcov, čo pôsobí demoralizujúco najmä preto, že príčina väčšiny chýb je v nedostatku školenia, dokumentácie a zaužívaných procesoch, nie v individuálnej nekompetentnosti.

Efektívne reagovanie na chyby, ich okamžitou opravou je dôležitou súčasťou procesu tvorby softvérových produktov. Z dôvodu, že okolnosti činnosti organizácií sa rapídne menia, množstvo z nich nemá priestor na zmenu starých návykov, používať chybové dáta na okamžitú reakciu bez vyhodnotenia spôsobov ako eliminovať podobné problémy v budúcnosti. Eliminácia hlavných príčin potenciálnych budúcich chýb musí byť zahrnutá v dlhodobej stratégií úspešnej organizácie.

2.3 Analýza zlyhaní

Využitie chybových dát na elimináciu hlavných príčin vzniku chýb začína zmenou princípov práce s dátami. Reaktívny prístup sa vo všeobecnosti zameriava na jednotlivé chyby a pýta sa aká je ich závažnosť. Okrem toho porovnáva prioritu opravy konkrétnej chyby v porovnaní s ostatnými a sústredí sa na časový rámec v ktorom musia byť chyby opravené. Prístup komplexnej analýzy sa v prvom rade zaoberá otázkou čo spôsobilo vniknuté chyby, ktoré zo zlyhaní spôsobuje najväčšie plytvanie zdrojov a ako je možné podobným chybám v budúcnosti predísť.

Diskutovanie hlavných príčin je cesta ako začať so zlepšovaním prístupu v rámci organizácie. Chybové dáta môžu poskytovať merateľný základ pre rozhodnutia, ktoré je potrebné vykonať. Pokračovaním v sledovaní chybových dát, môže organizácia vyhodnocovať mieru úspešnosti aplikovaných riešení.

2.4 Existujúce riešenia

Na účel zberu informácií o chybách softvéru a vyhodnocovanie ich príčin slúži množstvo webových nástrojov. Konkrétna množina softvérových produktov a typ produktov, ku ktorým sa viažu zbierané

dáta sú rôzne, ale v základných princípoch zbierania a analýzy dát sú výrazné podobnosti. Najvýznamnejšími zástupcami z kategórie open source softvéru, ktorý je voľne dostupný vrátane zdrojových kódov sú okrem FAF nástroje *Socorro* [1], *Error tracker* [2] a *Linux Kernel Oops* [3].

2.4.1 FAF

V roku 2009 sa v rámci komunitného projektu Fedora rozbehla iniciatíva vybudovať sadu nástrojov, ktorá pomôže používateľom operačných systémov Fedora a CentOS objaviť a nahlásiť chyby softvérových produktov. Na tento účel vnikol ABRT, ktorého štruktúra je priblížená v kapitole 3. ABRT bol postupne rozširovaný o ďalšie pomocné nástroje a komponenty. Jedným z týchto nástrojov je FAF, ktorý hrá v súčasnej dobe dôležitú úlohu pri poskytovaní spätnej väzby vývojárom. Použité technológie a štruktúra FAF sú podrobnejšie rozpísané v kapitole 4.

2.4.2 Socorro

Aplikácie spoločnosti Mozilla využívajú od verzií Firefox 3, Thunderbird 3 a SeaMonkey 2 open source systém na hlásenie pádov Mozilla Crash Reporter, ako je zdokumentované v článku [12]. V prípade pádu niektorého z produktov tejto spoločnosti je používateľ upozornený na tento pád a je mu umožnené odoslať dáta o páde na webový server.

Úlohu spracovávania a poskytovania dát vývojárom v tomto prípade plní webový server Socorro. Toto webové rozhranie ponúka celkové štatistiky počtu pádov pre jednotlivé aplikácie a ich verzie a detailné grafy, ktoré umožňujú zobraziť počet pádov rôznych verzií produktu v rámci zvoleného časového intervalu. Z hľadiska analýzy konkrétnych pádov je užitočný prehľad chýb, ktoré sa vyskytli v najväčšom počte. Tieto sú prezentované formou tabuľky, kde je možné dáta filtrovať na základe aplikácie a jej verzie, rozmedzia dátumov, operačného systému a typu pádu.

2.4.3 Error tracker

Error tracker Linuxovej distribúcie Ubuntu sa z pomedzi open source nástrojov na tento účel naviac podobá na FAF z hľadiska účelu, na ktorý bol navrhnutý, typu a množstva hlásení, ktoré spracováva.

Celková spoľahlivosť jednotlivých verzií operačného systému sa vyjadruje pomocou dennej intenzity výskytu chýb. Tento spôsob merania je podľa dokumentácie projektu [17]

[1] https://crash-stats.mozilla.com/
[2] https://errors.ubuntu.com/
[3] http://www.kerneloops.org/

vypočítavaný ako podiel celkového počtu chýb za daný deň a celkového počtu počítačov, ktoré v daný deň mohli nahlásiť chybu v prípade že by sa vyskytla. Množstvo počítačov, na ktorých je systém Ubuntu nainštalovaný nie je v tejto štatistike zohľadnených z dôvodu, že nemajú odosielanie hlásení aktivované.

Denná intenzita výskytu chýb je vizualizovaná v grafe pre posledných niekoľko verzií systému. Z grafu je možné vyčítať, že staršie verzie systému sú výrazne stabilnejšie v porovnaní s poslednými verziami, ktoré ešte neboli dostatočne vyladené.

Výskyty samotných pádov sú prezentované v tabuľke, ktorá zahŕňa počet pádov, binárny softvérový balík, ktorého sa pád týka, verziu systému, kde sa pád objavil prvý a posledný krát, funkciu a priradený bug. Zobrazované dáta je možné filtrovať podľa viacerých kritérií a zoradiť na základe zvoleného stĺpca tabuľky.

Na prezeranie konkrétnych dát, ktoré sa viažu k určitému pádu sú potrebné práva člena skupiny, ktorá má oprávnenie s dátami pracovať. Dokumentácia na wiki stránkach [17] hovorí, že hlásenia pádov sú zoskupované na základe spoločnej príčiny. Prehľad určitého hlásenia obsahuje vizualizáciu počtu výskytov v čase, tabuľku verzií balíka a stacktrace. Zaujímavosťou je rozhranie, pomocou ktorého môže niektorý zo správcov navrhnúť dodatočný zber dát pre konkrétnu skupinu hlásení, toto rozhranie však podľa dostupných informácií ešte nebolo implementované.

2.4.4 Linux Kernel Oops

Webové rozhranie Linux Kernel Oops sa špecializuje na zber dát o odchýlkach od korektného správania Linuxového jadra a získava dáta z viacerých zdrojov. Užívateľské rozhranie je veľmi jednoduché, na úvodnej stránke je štatistika operačných systémov, modulov jadra a funkcií, v ktorých sa objavuje najviac problémov. Jednotlivé hlásenia sa dajú opäť prezerať v tabuľke s možnosťou filtrovania na základe modulu, funkcie, súboru a niekoľkých ďalších kritérií.

Kapitola 3

Automatic Bug Reporting Tool (ABRT)

ABRT [1] je nástroj, ktorého úlohou je pomôcť používateľom nájsť, a následne nahlásiť pády aplikácií vrátane získania dodatočných informácií ako verzie komponentov a backtrace. Jeho hlavným cieľom je uľahčiť používateľom hlásenie pádov a urýchliť proces hľadania ich príčiny. ABRT je súčasťou linuxovej distribúcie Fedora od verzie 11.

Autori ABRT chránia metódu detekcie a hlásenia pádov, využívanú týmto nástrojom patentom [2]. Nasledujúca kapitola približuje základnú štruktúru ABRT, ktorá je predmetom tohoto patentu a popisuje konkrétne implementácie nástrojov, ktoré sú súčasťou projektu ABRT.

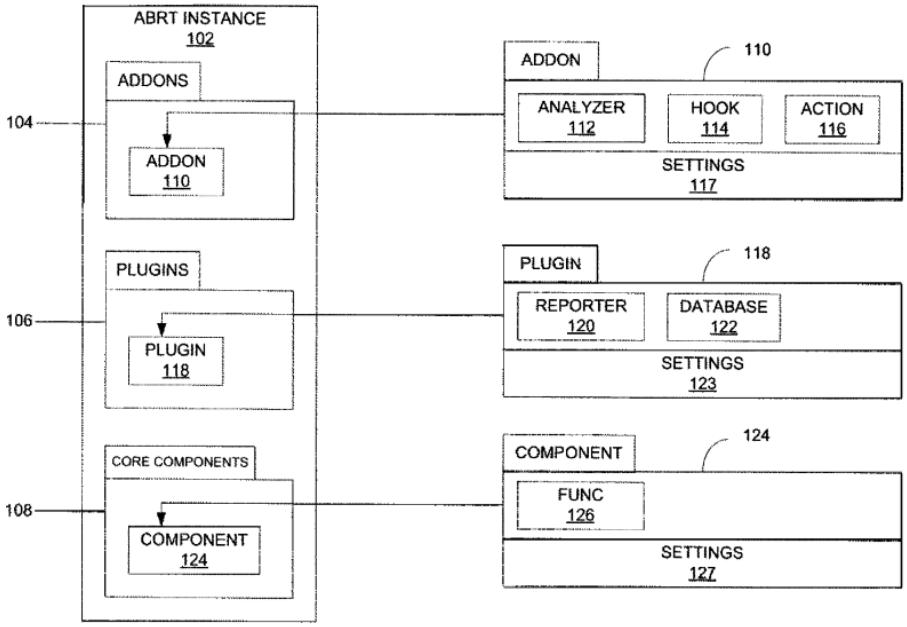
3.1 Štruktúra nástroja

Na pomoc vývojárom pri hľadaní potenciálnych chýb v softvéri, zberom informácií po zlyhaní programu bolo vyvinutých množstvo nástrojov. Jeden z príkladov tohoto typu softvéru vytvára snímok časti pamäte počítača v čase pádu. Na výskyt zlyhania procesu je používateľ zvyčajne upozornený správou o chybe. Následne mu býva poskytnutá možnosť odoslať záznam o chybe tvorcovi daného softvéru a dodať mu tak užitočnú informáciu k hľadaniu problému v programe.

Softvér navrhnutý spomenutým spôsobom je obmedzujúci z hľadiska detekcie pádu a odosielania dát jednému špecifickému cieľu. Určiť akým spôsobom má byť informácia získaná, aký typ dát je potrebné zozbierať a kam sa majú dáta odoslať môže byť náročná úloha zvlášť pre používateľa, ktorý nemá znalosti o postupoch pri ladení softvéru.

ABRT je zložený z viacerých spolupracujúcich celkov. Prvou časťou je nástroj na detekciu pádov v systéme, na ktorý nadväzuje modul zbiehajúci dodatočné dáta o páde. O hlásenie chybovej správy na jeden alebo viacero vzdialených serverov sa stará tretí komponent spôsobom špecifikovaným v knižnici hlásenia pádov príslušného typu softvéru.

Obrázok 3.1 znázorňuje schému jednej inštancie softvéru nazývaného ABRT. Inštancia ABRT obsahuje modul rozšírení, zásuvný modul a hlavný komponent. Modul rozšírení obsahuje jeden alebo viac rozširujúcich komponentov, ktoré špecifikujú spôsob detekcie pádov. Každý z rozširujúcich komponentov dokáže detektovať aspoň jeden typ softvérových pádov. Príkladmi rozširujúcich komponentov je zachytávanie pádov aplikácií napísaných v C/C++, Linuxového jadra alebo neodchytených výnimiek Python aplikácií. Po detekovaní pádu je spustená obslužná akcia príslušného komponentu.



Obr. 3.1: Schéma ABRT, publikovaná v rámci patentu [2]

Každý zásuvný modul poskytuje jeden zo spôsobov hlásenia chyby. Chyba môže byť odoslaná na vzdialený server alebo uchovaná v lokálnej databáze. Hlavný modul sa skladá z jedného alebo viacerých komponentov. Komponentami hlavného modulu môžu byť GUI, Applet, TUI alebo Démon.

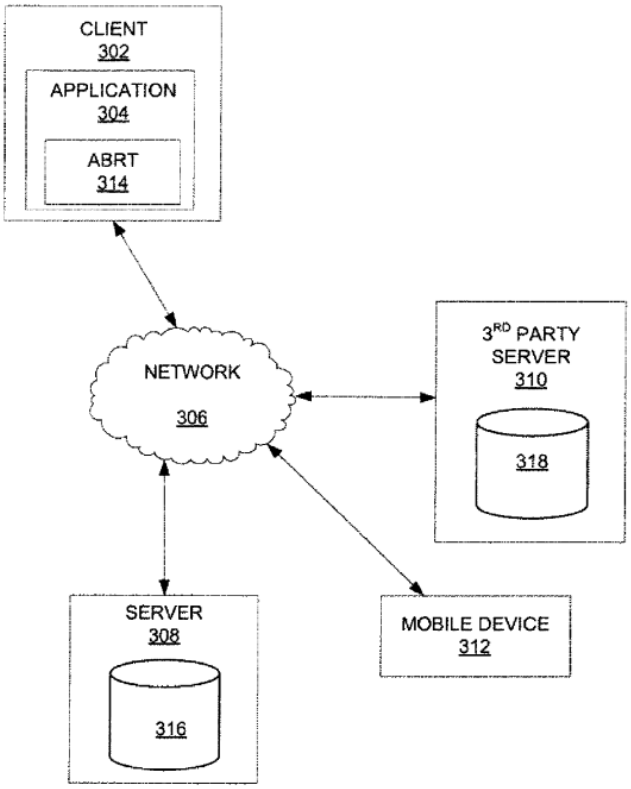
Keď dôjde k pádu aplikácie, táto udalosť je spracovaná rozširujúcim modulom ABRT špecifickým pre danú platformu alebo jazyk a informácia o tejto chybe je predaná ABRT démonovi. Démon uloží dáta do adresára /var/tmp/abrt a následne vykoná niekoľko akcií. Prvým krokom je zber dodatočných dát o systéme, následne vytvorí backtrace z core dump informácií a upozorní používateľa na udalosť ku ktorej došlo.

3.2 Zber dodatočných informácií pre ABRT

Komponenty na zber dodatočných informácií a získanie backtrace sú implementované v rámci nástroja satyr [16]. Ide o kolekciu nízko úrovňových algoritmov pre spracovanie zlyhaní programu. Po analýze zlyhania je vytvorené a následne odoslané hlásenie o vzniknutej chybe. Nástroj satyr dokáže získať potrebné dáta z viacerých zdrojov ako stacktraces vytvorené GDB, Informácie o neodchytených výnimkách jazyka Python a kernel oops správy. Informácie môžu byť tiež vyextrahované z core dump neočakávane ukončených procesov používateľa. Pokiaľ ide o viacvláknové procesy, sú odhalené vlákna, ktoré chybu spôsobili. Typický postup pri hlásení pádu pozostáva z generovania mikro hlásenia, ktoré je navrhnuté tak, aby bolo úplne anonymné a mohlo byť spracovávané plne automaticky.

Obrázok 3.2 zobrazuje schému procesu hlásenia softvérového pádu po sieti. Na strane klienta beží aplikácia ABRT klienta nakonfigurovaného na komunikáciu s ABRT serverom. Komunikácia je možná s primárnym serverom, serverom tretej strany alebo prenosným zariadením. Každý server poskytuje pamäťový priestor na ukladanie chybových hlásení. ABRT server môže dáta zdielať s inými servermi a prenosnými zariadeniami. Po priatí

Obr. 3.2: Sieťová komunikácia ABRT, publikovaná v rámci patentu [2]



informácie o zlyhaní určitého softvéru je táto chyba klasifikovaná logikou na strane servera. Chybové dáta sú uchované aby mohli byť neskôr analyzované automatizovaným spôsobom alebo človekom.

Konkrétnym príkladom knižnice na odosielanie hlásení po sieti je projekt libreport [11]. Ide o knižnicu, ktorá slúži na hlásenie chybových dát rôznym cieľom ako mailing listy, lokálne súbory a vzdialené servery. Knižnica pracuje s dátami uloženými formou súborov v lokálnom adresári. Libreport poskytuje nízkoúrovňové rozhranie na tvorbu a úpravy adresárov obsahujúcich chybové dáta a zbierku nástrojov na odosielanie lokálne uložených dát. Podporovaných je niekoľko spôsobov odoslania dát:

- Uloženie formou lokálneho súboru
- Odoslanie prostredníctvom e-mailu na predom určenú adresu
- Odoslanie po sieti s využitím rôznych protokolov (HTTP, XML-RPC, FTP)

Kapitola 4

Fedora Analysis Framework (FAF)

V podkapitole 3.2 boli popísané možnosti odosielania hlásení rôznym cieľom, ktoré ABRT podporuje. V prvej fáze po nasadení nástroja boli upozornenia vývojárom a správcom komponentov Linuxovej distribúcie Fedora zobrazované prostredníctvom systému Bugzilla [8]. Primárnym účelom Bugzilly je ale zber hlásení vytvorených manuálne niektorým z používateľov, nie automaticky generovaných hlásení. Toto riešenie neplnilo svoj účel veľmi efektívne, Bugzilla bola hláseniami z ABRT celkom zaplavená. Vývojári si väčšinou nemohli dovoliť podrobne sa zaoberať všetkými hláseniami, tak ich často filtrovali a nemohli efektívne plniť svoj účel.

Táto situácia viedla k myšlienke na vytvorenie štatistiky, ktorá by vývojárom pomohla identifikovať, ktoré pády zasahujú zásadný počet používateľov a preto je potrebné sa nimi zaoberať s väčšou prioritou a zároveň takto odlíšiť okrajové prípady, ktorých priorita je nízka, alebo sa nimi vôbec nie je potrebné zaoberať.

Na vyriešenie vzniknutého problému bol vytvorený FAF [5], ktorý v tomto prípade plní úlohu prvej úrovne spracovávania chybových dát. Zbiera veľké množstvo podobných hlásení a v prípade že ide o známy problém rozšíri štatistiku daného typu zlyhaní. Na Bugzille je potom vytvorený jeden záznam pre celú množinu súvisiacich zlyhaní. Okrem toho poskytuje FAF štatistický prehľad dát o nahlásených pádoch. ABRT sa vďaka tomuto mohol stať plnohodnotným nástrojom na uľahčenie práce vývojárov.

4.1 Použité technológie

FAF je dynamickou webovou aplikáciou postavenou nad ABRT stack, ktorý je zložený z nástrojov satyr, libreport a ABRT. Hlavným implementačným jazykom FAF je Python.

Python sa v priebehu posledných rokov zaradil medzi najobľúbenejšie programovacie jazyky hlavne vďaka svojej jednoduchosti, čistej syntaxy a vyššej úrovni abstrakcie. Kód v jazyku Python je dobre čitateľný, vďaka čomu sa ľahko udržuje a veľkým benefitom je možnosť využitia v širokom portfóliu rozličných oblastí informačných technológií. Python je možné použiť na rozličné projekty s výnimkou oblastí, kde je kritický výpočtový výkon a rýchlosť behu výsledného programu. Okrem efektívneho spracovania textu, analýzy veľkých dát, vedeckých, numerických výpočtov a tvorby grafických rozhraní poskytuje Python množstvo frameworkov na vývoj webových aplikácií. FAF využíva populárny mikroframework Flask [15].

4.1.1 Webový framework Flask

Miguel Grinberg vyzdvihuje v úvode svojej knihy [10] výhody frameworku Flask. Voľný preklad jeho myšlienok je nasledovný:

Flask sa líši od iných frameworkov tým, že necháva vývojárom voľné ruky a možnosť plne kontrolovať dizajn ich aplikácie. V prípade, že sa vývojár rozhodne vyriešiť určitý problém neštandardným spôsobom vo väčšine frameworkov naráža na množstvo prekážok. Ide o prípady využitia iného databázového systému alebo metódy na autentifikáciu používateľov.

Flask bol navrhnutý s ohľadom na podobné prípady. Podporuje množstvo relačných databázových systémov, dokáže pracovať s NoSQL databázou alebo nemusí byť databázový systém použitý vôbec. Základom je robustné jadro, ktoré zahŕňa základnú funkcionalitu, potrebnú pre každú webovú aplikáciu a očakáva, že zvyšok bude pokrytý niektorým z množstva rozšírení tretích strán. Pri použití tohoto frameworku si môže vývojár zvoliť komponenty a rozšírenia vhodné pre jeho aplikáciu, alebo napísať vlastné, ktoré budú presne zodpovedať potrebám jeho projektu.

Systém definovania ciest aplikácie je postavený na prehľadných dekorátoroch. *Jinja2* [1], knižnica na prácu so šablónami je s frameworkom dobre integrovaná, jednoducho sa používa a obsahuje silné nástroje v podobe makier, filtrov a hierarchie dedičnosti šablón. Efektívne spracovávanie je zaručené just in time kompiláciou vytvorených šablón do Python byte kódu.

Okrem nástrojov na tvorbu základu pre flexibilnú webovú aplikáciu ponúka Flask dobre prepracované nástroje na vývoj a ladenie programu. S napísaním minimálneho množstva kódu je možné vytvoriť lokálny server na účely vývoja a spustiť aplikáciu priamo zo zdrojových kódov. Všetky potrebné informácie na ladenie ako HTTP komunikácia a vykonávané databázové transakcie je možné sledovať priamo v termináli po aktivovaní módu ladenia programu. Nástroje na ladenie a manažment ciest aplikácie a Web Server Gateway Interface (WSGI) sú súčasťou knižnice *Werkzeug* [2] na ktorej Flask závisí.

4.1.2 Komunikácia s databázovým systémom

FAF pracuje s desiatkami gigabajtov dát. Ako dátové úložisko je využívaný databázový systém *PostgreSQL* [3]. Databázový model aplikácie a komunikácia s databázou je implementovaná s využitím knižnice SQLAlchemy. Táto flexibilná knižnica, ktorá tvorí akýsi most medzi relačnými databázami a tradičným programovaním v jazyku Python je priblížená v knihe [13] autorov Jasona Myersa a Ricka Copelanda.

SQLAlchemy je knižnica využívaná na interakciu so širokou škálou databázových systémov. Umožňuje vytvorenie databázového modelu a dotazov spôsobom, ktorý pripomína štandardné triedy a konštrukcie jazyka Python. Táto knižnica bola napísaná v roku 2005, využíva ju množstvo spoločností a je považovaná za tradičný spôsob práce s relačnými databázami v rámci aplikácií implementovaných v jazyku Python. Prostredníctvom SQLAlchemy je možné komunikovať s najpoužívanejšími databázovými systémami ako PostgreSQL, MySQL, SQLite, Oracle a mnoho ďalších.

Knižnica SQLAlchemy poskytuje množstvo flexibility vďaka podpore dvoch základných režimov. Prvým je SQL Expression Language (často označovaný ako jadro) a druhým

[1] http://jinja.pocoo.org/
[2] http://werkzeug.pocoo.org/
[3] https://www.postgresql.org/about/

Object-relational mapping (ORM). Tieto režimy môžu byť použité nezávisle alebo spoločne podľa potrieb konkrétnej aplikácie.

SQL Expression Language je spôsob reprezentovania bežných konštrukcií a výrazov jazyka SQL spôsobom ľahko integrovateľným do Python kódu, ide o jemnú abstrakciu nad SQL. Abstrakcia je zameraná na databázovú schému a štandardizovaná tak, aby umožňovala použitie konzistentnej syntaxe pre väčšie množstvo databázových systémov.

SQLAlchemy ORM je podobný implementáciám tohoto princípu v iných jazykoch. Poskytuje vysokú úroveň abstrakcie nad SQL Expression Language. Definícia schémy je pri použití SQLAlchemy ORM trochu odlišná z dôvodu zamerania na používateľom definované objekty namiesto databázovej schémy, ktorá sa nachádza pod touto abstrakciou. Používateľ definuje triedu, ktorá dedí z bázovej triedy nazývanej *declarative_base*. Táto trieda obsahuje kontajner metadát a mapovanie atribútov na tabuľku v databáze. Po definovaní modelu vo forme tried, pracuje vývojár vo svojom kóde s inštanciami namapovanej triedy, každý objekt reprezentuje jeden riadok databázovej tabuľky. Pri pristupovaní k atribútom objektu sú v pozadí vykonávané dotazy nad databázou potrebné na získanie dát. Triedam je možné pridať metódy, ktoré zapúzdrujú logiku pracujúcu nad dátami z databázového systému. Knižnica prináša syntax na vykonávania dotazov nad databázou prostredníctvom namapovaných tried. Dotazovanie, filtrovanie a spracovávanie dát je vo FAF implementované výlučne s využitím ORM.

Rozsiahlejšie projekty sa väčšinou nezaobídu bez úprav databázového modelu. Na účel vykonávania databázových migrácií, bez nutnosti odstránenia a znovu vytvorenia databázy definovanej pomocou SQLAlchemy slúži nástroj Alembic [3]. Alembic poskytuje spôsob ako programovo vytvoriť a uskutočniť migrácie. S využitím tejto knižnice môže vývojový tím kedykoľvek prispôsobiť databázový model zmenám, ktoré aplikácia vyžaduje. Je možné takto pridať alebo odobrať z tabuliek stĺpce, vytvoriť nové tabuľky a ďalšie úpravy modelu. Vďaka naviazanosti na SQLAlchemy je toto možné aplikovať na široké spektrum databázových systémov. Alembic tak isto umožňuje všetky zmeny databázového modelu vykonávať pomocou skriptov napísaných čisto v jazyku Python.

4.1.3 Vizualizácia

Neoddeliteľnou súčasťou vývoja webových aplikácií je prezentácia dát používateľovi, vizualizácia pomocou diagramov a grafov a tiež voľba vzhľadu aplikácie. V posledných rokoch sa v tejto oblasti objavilo veľké množstvo nových knižníc a frameworkov.

FAF sa spolieha na framework Patternfly [14]. Ide o komunitný projekt, ktorý presadzuje jednotnosť základných rysov dizajnu aplikácií. Tím Patternfly navrhol a vytvoril sadu šablón na základe porozumenia častým prípadom použitia a požiadavkom projektov. Každý zo vzorov zahŕňa dokumentáciu, vzhľad a ukážkový prípad použitia. Niektoré vzory sú rozšírené o ukážku kódu s priamou demonštráciou jeho fungovania. Tieto demonštrácie sú postavené na frameworkoch *Bootsrap 3* [4] a *AngularJS* [5].

V knižnici Patternfly sa nachádzajú vzory pre rozličné prvky webového užívateľského rozhrania od horizontálneho a vertikálneho menu, upozornení, cez rôzne druhy grafov až po tabuľky a formy na zadávanie vstupných dát. Každý z týchto komponentov užívateľského rozhrania má viacero variant a poskytuje tak možnosť výberu najvhodnejšieho riešenia pre konkrétnu aplikáciu. Za projektom stojí aktívna komunita, ktorá pomáha s podporou všetkých nástrojov.

[4] http://bootstrapdocs.com/v3.3.6/docs/
[5] https://angularjs.org/

4.2 Hlásenia a Problémy

Základnou entitou analýzy z pohľadu FAF sú hlásenia (reports). Každé hlásenie je skupinou zlyhaní softvéru, ktoré boli zozbierané od používateľov. Hlásenie pádu obsahuje informácie o systéme, kde neočakávané ukončenie procesu nastalo a príslušný backtrace. V prípade, že majú prichádzajúce správy o pádoch identický backtrace ako niektoré zo spracovaných hlásení, považujú sa za hlásenie rovnakého zlyhania. Každému hláseniu je priradený jeden komponent operačného systému. Komponentom je RPM Package Manager (RPM) balíček, ktorého súčasťou je súbor s popisom programu, ktorý prestal pracovať, tento súbor a komponent sa nachádza na konci backtrace.

Detailný prehľad jedného z pádov poskytuje informácie o prvom a poslednom výskyte daného hlásenia, prehľad počtu výskytov na rozličných operačných systémoch a architektúrach a počet výskytov v jednotlivých verziách komponentu. Okrem toho je tu aj niekoľko grafov znázorňujúcich počet výskytov daného hlásenia v čase, graf zobrazujúci zastúpenie verzií operačného systému a tabuľka obsahujúca backtrace daného pádu.

Hlásenia, ktorých backtrace je výrazne podobný sú zoskupované do problémov. Problém môže tvoriť niekoľko desiatok hlásení, ich podobnosť je na toľko veľká že sú pravdepodobne spôsobené rovnakou hlavnou príčinou. Hlásenia sú rozdelené podla typu pádu a následne zhlukované na základe podobnosti backtrace. Z takto vzniknutých zhlukov sa vytvárajú problémy. Skript na zhlukovanie hlásení je nad produkčnou databázou spúšťaný približne raz za desať minút, preto môžu v zložení problémov nastávať malé zmeny. Do problému sú často zoskupované hlásenia viazané na viacero rozličných komponentov.

Dáta zobrazované v tabuľke problémov je možné filtrovať podľa množstva kritérií. Základné filtrovanie na základe verzie operačného systému, názvu komponentu, správcu alebo skupiny správcov, typu a dátumu je rozšírené o sadu pokročilých filtrov. V tejto kategórií sa nachádza vyhľadávanie v určitom rozsahu verzií komponentov, stavu priradených bugov, architektúry, výskytu funkcie alebo súboru v backtrace a niekoľko ďalších. Prehľad konkrétneho problému poskytuje podobne ako je to u hlásenia záznam o prvom a poslednom výskyte, tabuľky počtu výskytov na jednotlivých operačných systémoch, architektúrach a počet výskytov u balíkov operačného systému. Na každý problém sa môže viazať niekoľko bugov otvorených rozličných systémov na sledovanie bugov, najčastejšie ide o systém *Bugzilla* [6].

4.3 Akcie

Súčasťou FAF je CLI ktoré slúži na vykonávanie rôznych pomocných akcií na strane servera. Väčšina z nich pracuje s databázovým systémom. Nachádzajú sa tu moduly, ktoré zabezpečujú prepojenie s ďalšími službami ako napríklad bug tracker systémy a skripty na rýchle získanie určitých informácií z databázy. Stav databázy sa rýchlo mení, neustále pribúdajú nové a nové hlásenia. Z tohoto dôvodu je potrebné vykonávanie niektorých akcií periodicky opakovať. Ide o funkcie, ktoré po vykonaní určitých výpočtov doplňujú a aktualizujú záznamy v databáze, príkladom je spomínaná tvorba problémov zoskupovaním hlásení na základe ich podobnosti.

[6] https://bugzilla.redhat.com/

Kapitola 5

Konkrétne nedostatky FAF

V úvode tejto práce bolo poukázané na fakt, že nástroje, ktoré FAF ponúka na prezeranie a filtrovanie stoviek tisíc zozbieraných hlásení a problémov, do ktorých sú hlásenia zoskupované, často nepostačujú na naplnenie potrieb vývojárov pracujúcich s týmito dátami. V nasledujúcej kapitole je popísaných niekoľko konkrétnych problémov, ktoré vyplývajú z *nedostatkov nahlásených v repozitári projektu FAF* [1] alebo z informácií získaných priamo od vybraných používateľov aplikácie.

5.1 Neprehľadný zoznam problémov

V kapitole 4.2 bol vysvetlený princíp zoskupovania hlásení s podobným backtrace do problémov. V niektorých prípadoch problém obsahuje niekoľko desiatok hlásení. Väčšina z týchto hlásení sa viaže k rôznym komponentom operačného systému z čoho vyplýva, že daný problém sa týka veľkého množstva komponentov.

V tabuľke obsahujúcej prehľad problémov sa v bunke pre komponent zobrazuje zoznam všetkých komponentov, ktoré sa k danému problému viažu. Pri veľkom počte problémov sa takto stáva tabuľka veľmi neprehľadnou ako je možné vidieť na obrázku 5.1. V prípade, že nastaví používateľ filter na základe komponentu, ktorý spravuje nie je možné na prvý pohľad tento komponent v dlhom zozname vidieť, čo robí orientáciu v tabuľke zbytočne náročnou.

V prípade prehľadávania problémov bez nastavenia filtra na špecifický komponent, typ alebo časové rozmedzie sú výsledkom dotazu často tisíce záznamov tabuľky. Aktuálna implementácia zoradí výsledky podľa počtu výskytov daného problému a obmedzí dotaz na prvých štyridsať výsledkov. V prípade vyžiadania ďalších výsledkov sa dotaz opakuje s rovnakým limitom a posunutím na nasledujúcich štyridsať výsledkov.

Pri prezeraní tabuľky je vždy k dispozícii iba štyridsať záznamov s určitým posunutím a nie je možné v dátach efektívne vyhľadávať, zoraďovať ich podľa zvoleného kľúča alebo sa presunúť na koniec zoznamu. Tento spôsob prezentácie dát robí ich prehľadávanie v mnohých prípadoch pomerne ťažkopádnym.

[1] https://github.com/abrt/faf/issues/

Obr. 5.1: Tabuľka problémov, screenshot z *produkčného servera FAF*[2]

922665	python-subliminal, hammertime, python-jenkinsapi, gcAdmin, googleplaydownloader, python-sphinx-autobuild, bugwarrior, Mayavi, lets-encrypt, dnsyo, thefuck, httpie, python-nikola, ransack, everpad, elasticsearch-curator, osbs-client, paws, livestreamer, ReviewBoard, glances, RBTools, sflvault-client-qt, khal, bpython, docker-compose, pulseaudio-dlna, scudcloud, packagedb-cli, ccutil, copr-cli, python-coverage, python-virtualenv, boom, fig, python-jira, python-spec, python-setuptools, python-bucky, musicbox, letsencrypt, youtube-dl, supernova, devassistant, pylint, python-neovim-gui, python-frappe-bench, openshot, atomicapp, yubikey-neo-manager, pagure-importer, python-autopep8, python-watchdog, ahkab, rdopkg, fontdump, python-sphinx, python-alembic, python-xhtml2pdf, deluge, beets, python-doit, ntfy, rst2pdf, ptpython, fedmsg-notify, mullvad, vdirsyncer, sflvault-client, python-scrapy, babel, python-pip, git-buildpackage, ipython, python-rcm-nexup, gaphor	resolve	FIXED	~~BZ#1298083~~ ~~BZ#1297516~~ ~~BZ#1264426~~
2786438	abrt, dnf, python3	__memcpy_sse2_unaligned	NEW	
2754077	telepathy-mission-control, nautilus, firefox, nemo, ibus, totem, gnome-shell, geany, gnome-contacts, pluma, caja, emacs, tracker, Thunar, libreoffice, gvfs, file-roller, synapse, lollypop, gedit, python3, evolution, eog, evolution-data-server, PackageKit, transmission, gnome-settings-daemon, cinnamon, xfdesktop, rygel, kupfer	g_slice_alloc	FIXED	~~BZ#1036184~~ ~~BZ#1213444~~ ~~BZ#1170419~~ ~~BZ#975327~~ ~~BZ#1177387~~

5.2 Priraďovanie komponentov problémom

Ku každému z hlásení je priradený komponent nadchádzajúci sa na konci jeho backtrace ako bolo priblížené v kapitole 4.2. Pri zoskupovaní hlásení do problémov sa k problému viažu všetky unikátne komponenty hlásení, ktoré sú do problému zahrnuté. Pri podrobnejšej analýze backtrace jednotlivých hlásení, ktoré boli na základe podobnosti spojené do problému je možné vidieť, že backtrace je často rozdielny iba v poslednom alebo posledných niekoľko rámcoch. Napríklad ide o pád knižnice, ktorá je volaná z kódu rôznych spustiteľných súborov. V takomto prípade sa prichádzajúce hlásenia viažu ku komponentom týchto binárnych súborov, pričom jadro problému, riadok kódu ktorý pád spôsobil, sa nachádza na začiatku backtrace. Komponent súboru, v ktorom je jadro problému sa v zozname komponentov problému vôbec nenachádza a teda ho nie je možné vyhľadať pri filtrovaní tohoto komponentu.

Podobné prípady sú väčšinou riešené spôsobom, že správcovia komponentov problému nahlásia bug prostredníctvom systému Bugzilla a priradia ho komponentu, ktorý je za problém zodpovedný. Samotný FAF však žiadne riešenie neponúka.

[2] https://retrace.fedoraproject.org/faf/problems/

5.3 Upozornenia

ABRT upozorňuje správcov komponentov na vzniknuté udalosti prostredníctvom e-mailových správ. Takáto správa je zaslaná pri vzniku nového hlásenia alebo problému a v prípade, že počet výskytov niektorého problému alebo hlásenia dosiahne určitú hranicu. Niekoľko používateľov vyjadrilo svoju nespokojnosť s veľkým množstvom upozornení, ktoré pre nich neposkytujú žiadnu užitočnú informáciu.

Systém upozornení je implementovaný pomocou fedmsg (FEDerated MeSsaGe bus) [6]. Fedora Infrastructure pozostáva z množstva služieb, ktoré medzi sebou potrebujú komunikovať. Tím Fedora Infrastructure si stanovil za cieľ zjednotiť spôsob zasielania správ medzi jednotlivými aplikáciami. Na tento účel bola vytvorená knižnica fedmsg implementovaná v jazyku Python, ktorá definuje rozhranie na bezprostredné zasielanie a prijímanie správ.

5.4 Chýbajúca interakcia so systémom

FAF poskytuje používateľom zozbierané dáta o hláseniach a umožňuje ich filtrovať. V niektorých prípadoch sa stáva, že určité hlásenie nie je pre správcu komponentu, ktorého sa týka užitočné z dôvodu nedostatku informácií, prípadne používateľ vie, že je problém už vyriešený v upstreame projektu alebo sa ním nechce zaoberať z iného dôvodu. V podobných situáciách nie je možná žiadna priama interakcia so systémom, jedinou možnosťou je uzatvoriť príslušný bug v niektorom z podporovaných Bug tracker systémov v prípade že má hlásenie bug priradený. Uzatvorenie bugu sa spätne prejaví vo FAF zmenou stavu problému na základe stavu priradeného bugu. Priama interakcie na úrovni FAF, môže pri vhodnom návrhu výrazne zlepšiť možnosti organizácie práce vývojárov.

5.5 Vizualizácia

Dôležitou súčasťou analýzy dát je vizualizácia. FAF obsahuje niekoľko názorných zobrazení dát formou grafov. Okrem grafu počtu pádov na jednotlivých verziách operačných systémov v základnom sumári aplikácie je to hlavne koláčový graf, ktorý zobrazuje pomer počtu výskytov hlásení u operačných systémov na stránke problému a zásobníkový stĺpcový graf počtu výskytov hlásenia v prehľade každého z hlásení.

Množstvo dát a súvislostí, ktoré FAF obsahuje má veľký potenciál na zlepšenie názornosti podania informácií pomocou vizualizácie. Príkladom môže byť graf výskytu určitého hlásenia v závislosti na verzií komponentu. Takéto zobrazenie, by mohlo pri vhodnom prevedení poskytnúť prehľad o tom, či zmeny v novej verzií komponentu vyriešili problém, ktorý daný typ zlyhaní spôsoboval.

5.6 Zhodnotenie nedostatkov

Existujúce open source nástroje na analýzu chybových hlásení priblížené v kapitole 2.4 neprinášajú veľké množstvo prostriedkov na riešenie nájdených nedostatkov a zlepšenie možností organizovania práce pre vývojárov. Základné entity a spôsob ich prezentácie sú vo všetkých prípadoch podobné. Niektoré zo spôsobov zoraďovania dát, a prevedenia ich prezentácie môžu byť inšpiráciou pri návrhu nadstavby pre FAF, ale žiadne komplexné riešenia, ktoré by mali potenciál zásadne rozšíriť možnosti FAF tieto nástroje neposkytujú

Z nedostatkov priblížených v predchádzajúcich podkapitolách bolo potrebné vybrať podmnožinu, ktorou sa budeme v rámci tejto práce zaoberať. Kľúčom k výberu bola hlavne dôležitosť, implementačná a časová náročnosť a potenciál na zvýšenie efektivity práce a spokojnosti používateľov.

V prvom rade sa pokúsime navrhnúť zlepšenie práce s hlavnou tabuľkou problémov 5.1. Ide o vstupný bod aplikácie na vyhľadávanie dát, ktorými sa bude používateľ zaoberať. Práve na tomto mieste dochádza k veľkému množstvu dotazovania pri rôznom nastavení filtrov a prostredníctvom výsledkov v tabuľke pristupujú používatelia ku všetkým podrobnejším informáciám. Z tohoto dôvodu je prehľadnosť a flexibilita tabuľky veľmi dôležitá.

Ďalším bodom 5.4 bude pridanie dodatočných možností interakcie používateľov so systémom. Možnosť priamo ovplyvniť a zorganizovať zobrazované dáta má potenciál výrazne zjednodušiť a spríjemniť prácu pri analyzovaní veľkého množstva dát.

V rámci tejto práce by sme sa chceli zamerať aj na integráciu ovládania zasielaných upozornení 5.3 do používateľského rozhrania FAF a zmenšenie počtu situácií, keď sa správcom určitých komponent zobrazujú hlásenia a problémy, ktorých jadro s ich komponentami vôbec nesúvisí.

Kapitola 6

Návrh a Implementácia

V nasledujúcej kapitole sú podrobne rozobraté riešenia nedostatkov a vylepšení FAF, ktoré sú predmetom tejto práca na základe záverov z kapitoly 5.6. Každá podkapitola je venovaná jednému z riešených problémov a obsahuje návrh riešenia, podstatné časti implementácie a vysvetlenie benefitov, ktoré používateľovi prináša. Detaily implementácie je možné si prehliadnuť v *repozitári projektu* [1].

6.1 Tabuľka problémov

Pri prehľadávaní enormného množstva riadkov tabuľky je užitočné mať možnosť zoraďovať riadky podľa zvoleného stĺpca a vyhľadávať v nich kľúčové hodnoty. Toto FAF aktuálne neumožňuje, čo robí prehľadávanie neefektívnym a závislým na nastavení dostatočne špecifického filtrovania. Prehľadávanie tabuľky s možnosťou radenia podľa zvoleného stĺpca sa nachádza aj v aplikácií ErrorTracker [17] spomínanej medzi existujúcimi riešeniami v kapitole 2.4. Jazyk JavaScript ponúka niekoľko knižníc a riešení na tvorbu takejto tabuľky.

Framework Patternfly [14] využívaný vo FAF na prezentáciu dát, ponúka vzor na vytvorenie potrebného typu tabuľky, založený na *jQuery* [2] module DataTables [4]. Tento modul rozširuje prístupnosť a možnosti práce s dátami HTML tabuliek a umožňuje používateľom rýchlo získať užitočné dáta. Okrem toho zahŕňa funkcie na zoraďovanie, vyhľadávanie kľúčových reťazcov a stránkovanie. Modul DataTables poskytuje vývojárom možnosti konfigurácie a spracovania dát na strane servera aj klienta. Pre použitie FAF je vhodnejšia a ľahšie integrovateľná možnosť spracovávania vopred vygenerovanej HTML tabuľky na strane klienta. Použitie knižnice týmto spôsobom vyžaduje iba pridanie triedy **datatable** zvolenej tabuľke a nakonfigurovanie objektu tabuľky zadaním hodnôt potrebných parametrov.

Pre efektívne využitie potenciálu tabuľky je potrebné získať všetky záznamy zodpovedajúce aktuálnemu filtrovaniu. Ako bolo priblížené v kapitole 4.1.2, FAF pri dotazovaní využíva SQLAlchemy ORM. Pri dotazovaní týmto spôsobom je pri prvom dotaze získaný zoznam objektov mapovaných na riadky databázovej tabuľky problémov. Dodatočné dáta sú získavané v čase generovania HTML dokumentu prostredníctvom atribútov mapovaného objektu, čoho výsledkom je vykonávanie množstva ďalších SQL dotazov pre každý z objektov. Výhodou tohoto postupu je jednoduchšia syntax ale v prípade veľkých dát je dopad na rýchlosť významný.

[1] https://github.com/mcyprian/faf/commits/user_experience_improvement
[2] https://jquery.com/

Po odstránení limitu pre dotaz na štyridsať výsledkov sa komplexné dotazovanie s využitím ORM ukázalo ako nevhodné v prípadoch, keď sú výsledkom tisícky záznamov, keďže získanie dát trvá rádovo stovky sekúnd. Ako vhodné riešenie vzniknutého problému sme zvolili využitie druhého z režimov SQLAlchemy a prepísanie kľúčového dotazu na získanie dát potrebných do základnej tabuľku problémov priamo do jazyka SQL.

Komplikáciou bol fakt, že hlavný dotaz na získanie dát tabuľky problémov sa dynamicky mení na základe špecifikovaných filtrov vyhľadávania. Pri návrhu bolo potrebné so zmenami počítať a vytvárať dotaz tak, aby umožňoval úpravy častí kódu za príkazom **FROM** a tiež filtra na výsledné riadky (príkaz **WHERE**). Ďalšou podmienkou bolo vloženie premenlivého počtu SQL parametrov a uloženie príslušných hodnôt spôsobom, ktorý zabezpečí vykonávanie dotazu proti prípadným útokom typu SQL injection.

Na docielenie všetkých vymenovaných vlastností sme navrhli tvorbu dotazu na princípe postupného skladania dvojice premenlivých častí dotazu a ukladanie hodnôt použitých parametrov do dátovej štruktúry typu slovník. Konkrétna implementácia tvorby SQL dotazu na základe hodnoty jedného z filtrov je zobrazená v ukážke 6.1.

Kód 6.1: Implementácia jedného z filtrov v rámci dynamickej tvorby SQL dotazu

```
    if binary_names or source_file_names:
        names_subquery = """
JOIN (SELECT DISTINCT reports.problem_id AS problem_id
      FROM reports
      JOIN reportbacktraces
          ON reports.id = reportbacktraces.report_id
      JOIN reportbtthreads
          ON reportbacktraces.id = reportbtthreads.backtrace_id
      JOIN reportbtframes
          ON reportbtthreads.id = reportbtframes.thread_id
      JOIN symbolsources
          ON symbolsources.id = reportbtframes.symbolsource_id
      WHERE reportbtthreads.crashthread = True
            AND ({0})) AS binary_search
ON binary_search.problem_id = func.id
"""
    if binary_names:
        binary_name_dict = {"binary_name_" + str(index): item
                            for index, item in enumerate(binary_names)}

        table_list += names_subquery.format(" OR ".join(
            ["symbolsources.path LIKE :{0} ".format(name)
             for name in binary_name_dict.keys()]))

        params_dict.update(binary_name_dict)
```

Po vyhodnotení všetkých podmienok filtrovania a zložení častí dotazu do výslednej podoby sú tieto časti SQL kódu konkatenované do výsledného reťazca a dotaz je zavolaný pomocou metódy **execute** s predaním slovníka parametrov a ich hodnôt.

Kód 6.2: Konkatenácia častí dotazu a jeho zavolanie

```
search_condition = "WHERE " + " AND ".join(search_condition)
search_condition += " GROUP BY func.id ORDER BY count DESC"
statement = text(select_list + table_list + search_condition)

return db.engine.execute(statement, params_dict)
```

Výsledkom dotazu je zoznam n-tíc jazyka Python obsahujúcich získané dáta, ktoré je možné integrovať do zvyšku kódu bez veľkého množstva zásadných zmien. Pri implementácií tohoto riešenia bol využitý vývoj riadený testami. Výsledky dotazu získané pri nastavení každého z filtrov a rôznych kombinácií filtrov bol porovnávaný s výsledkom pôvodnej implementácie a upravovaný do momentu úplnej zhody, prípadne minimalizovania rozdielov vo výsledných dátach.

Zmena implementácie dotazu priniesla znateľné zvýšenie rýchlosti. Tabuľka 6.1 obsahuje porovnanie doby, za ktorú je k dispozícií HTML tabuľka požadovaných dát. V prípade, že sú výsledkom dotazu tisíce riadkov, čo je v prípade FAF veľmi časté, pracuje riešenie pomocou čistého SQL mnohonásobne rýchlejšie. Vďaka tomuto zlepšeniu rýchlosti bolo možné odstrániť z dotazu akýkoľvek limit a stránkovanie na úrovni dotazu mohlo byť plne nahradené vstavaným stránkovaním knižnice DataTables.

Počet riadkov	**Filtre**	**Čisté SQL (s)**	**ORM (s)**
16 915	filtrovanie podľa dátumu	2.6	451.6
4 586	filtrovanie podľa dátumu	1.7	116.4
158	kombinácia množstva filtrov	1.57	8.17

Tabuľka 6.1: Porovnanie rýchlosti dotazu s použitím ORM a dotazu v čistom SQL.

Zvolená formulácia SQL dotazu prináša okrem rýchlosti, možnosť oddelenia hľadaných komponentov. V prípade, že je nastavený priamy filter komponentov alebo filter komponentov priradených určitému používateľovi prípadne skupine, sú v druhom stĺpci tabuľky problémov zvýraznené iba tieto komponenty a pridaná informácia o počte ďalších komponentov nevyhovujúcich filtru je uvedená v zátvorke. Ukážka tabuľky je na obrázku 6.1. Týmto spôsobom sa nám podarilo vyriešiť problém spomínaný v kapitole 5.1, znázornený na obrázku 5.1.

Nová implementácia vyžaduje v niektorých prípadoch odosielanie naozaj veľkého množstva dát klientovi, oveľa viac ako je vhodné nahrávať do pamäte počítača. Framework Flask prináša pre podobné prípady riešenie vo forme *streamovania obsahu* [3]. Pri vytváraní hlavnej tabuľky problémov bola štandardná implementácia generovania HTML dokumentu prostredníctvom šablóny nahradená streamovaním obsahu, čo v prípade potreby zaručí odosielanie dát klientovi po častiach.

6.2 Tabuľka hlásení

FAF okrem základnej tabuľky problémov, ktorej vylepšeniu bola venovaná predchádzajúca podkapitola pôvodne obsahoval ďalšiu globálnu tabuľku obsahujúcu jednotlivé hlásenia. Podobne ako u problémov bolo možné hlásenia filtrovať podľa rôznych kritérií.

[3] http://flask.pocoo.org/docs/0.12/patterns/streaming/

Obr. 6.1: Upravená tabuľka problémov, screenshot z testovacej inštancie FAF

Search:

ID	Component	Count	Crash function	State	Bugs
275228	**python**	78	meth	CLOSED	2
2024912	**pyp2rpm, python-virtualenv** (39 more)	50	resolve	NEW	0
285530	**pyp2rpm** (71 more)	31	resolve	CLOSED	15
922665	**python-virtualenv** (74 more)	25	resolve	CLOSED	3
346795	**python** (3 more)	22	remove_capslock_feedback	CLOSED	5
130123	**python** (7 more)	20	RC4	NEW	0
275229	**python** (1 more)	20	_print	NEW	0
186882	**python** (1 more)	17	new_threadstate	NEW	0
275206	**python**	14	test	CLOSED	1
1950571	**python-virtualenv** (1 more)	13	copyfile	NEW	0

Showing 1 to 10 of 40 entries

Počet výskytov určitého hlásenia v časovom rozmedzí, na určitej architektúre alebo verzií operačného systému sa prejaví ako vlastnosť problému, ktorého súčasťou sa hlásenie stalo. Z tohoto dôvodu išlo o miernu duplicitu v spôsobe prehľadávania a filtrovania dát. Pomerne častým prípadom sú aj problémy, ktoré obsahujú iba jedno hlásenie, alebo malý počet veľmi podobných hlásení. Takéto problémy sú z hľadiska obsahu takmer kópiou príslušného hlásenia.

Na stránky s prehľadom a detailnými informáciami problému sme pridali tabuľku hlásení, z ktorých sa problém skladá. Opäť tu bola použitá tabuľka obohatená o funkcie modulu DataTables a okrem možností vyhľadávania a zoraďovania dát v tabuľke nie je nevyhnutné ďalšie filtrovanie z dôvodu relatívne malého počtu záznamov. Pôvodná centrálna tabuľka hlásení bola odstránená.

Táto zmena upevnila logiku štruktúry aplikácie. Pri práci s dátami je potrebné postupovať od najvyššej úrovne abstrakcie vo forme problémov k hláseniam a ich prostredníctvom k súborom, funkciám až k jednotlivým riadkom kódu.

6.3 Archív hlásení

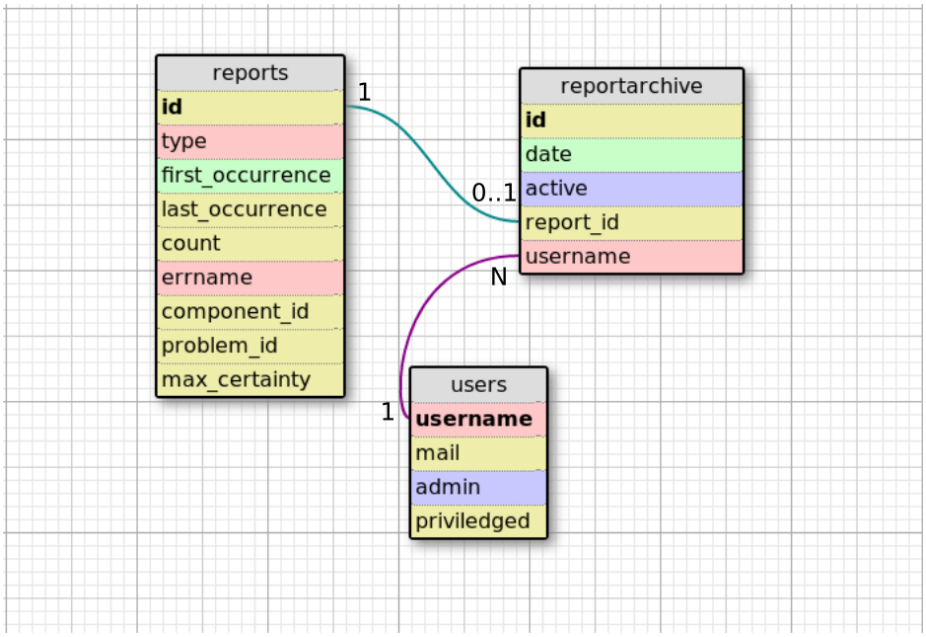
Ďalším z nedostatkov FAF, ktorý sme sa rozhodli riešiť v rámci tejto práce je nedostatok priamej interakcie používateľov so systémom. Zlepšenie v tomto smere sme navrhli v prvom rade formou archívu hlásení.

Prihlásený používateľ s právami správcu pre komponent, môže hlásenia, ktoré sa k tomuto komponentu viažu vložiť do archívu. Účelom tohoto mechanizmu je docieliť, aby sa používateľ nemusel opakovane zaoberať hláseniami, ktoré považuje z určitého dôvodu za neužitočné. Archivované hlásenia sa prestanú zobrazovať v tabuľke hlásení. Hlásenia, ktoré boli vložené do archívu je možné si prezerať po aktivovaní prepínača na zobrazenie archivovaných hlásení. Správca komponentu môže archivované hlásenia kedykoľvek obnoviť a budú opäť zaradené medzi ostatné hlásenia.

Pre umožnenie archivovania hlásení bola potrebná mierna zmena databázového modelu. Návrh databázovej tabuľky pre ukladanie informácie o archivácií hlásení a jej previazanie s tabuľkami databázového modelu sú znázornené na obrázku 6.2. Okrem úpravy definície databázového modelu pomocou SQLAlchemy bol pridaný skript na upgrade, respektíve

downgrade databázy s využitím nástroja Alembic. Vďaka tomuto skriptu je možné novú tabuľku pridať do existujúcej databázy tak, aby zodpovedala databázovému modelu po vykonanej úprave.

Obr. 6.2: Návrh časti databázového modelu pre archív reportov, vytvorený v prostredí *WWW SQL Designer* [4]



Keď je určité hlásenie vložené do archívu, vytvorí sa väzba záznamu tabuľky **reports** na novovytvorený riadok v tabuľke **reportarchive**. V zázname **reportarchive** je uložený dátum archivovania, položka **active** typu Boolean je nastavená na hodnotu true a cudzí kľúč **username** určuje väzbu na užívateľa, ktorý hlásenie do archívu vložil. Pri obnovení archivovaného hlásenia je hodnota položky **active** zmenená na false, v prípade opätovného vloženia do archívu sú dáta v tabuľke **archivreport** prepísané novými hodnotami.

Odosielanie dát o archivovaní hlásenia na server bolo implementované s využitím technológie Ajax aby nebolo potrebné opakované načítavanie celej stránky hlásenia, ktoré vyžaduje vykonanie množstva komplexných dotazov na získanie informácií o počte pádov jednotlivých verzií komponentu, zloženie backtrace a vykreslenie grafov.

Archív hlásení bol navrhnutý jednoduchým spôsobom, ale tento koncept má potenciál ďalšieho rozširovania. V prípade pozitívnej spätnej väzby od používateľov po nasadení, bude možné napríklad rozšíriť archív aj na problémy. Toto rozšírenie by mohlo spočívať vo vložení problému do archívu v prípade, že sú archivované všetky hlásenia, z ktorých problém pozostáva alebo naopak pri manuálnom archivovaní problému vložiť do archívu všetky jeho hlásenia.

[4] http://ondras.zarovi.cz/sql/demo/

6.4 Upozornenia

Problémom upozornení bol fakt, že niektorí z používateľov dostávali priveľké množstvo e-mailov upozorňujúcich na veľký počet výskytov určitého hlásenia alebo problému a neboli schopní tieto upozornenia obmedziť.

Nastavenia upozornení pre koncových používateľov všetkých služieb využívajúcich fedmsg [6] je možné spravovať pomocou Fedora Notifications [7]. Ide o systém vyvinutý tak, aby umožnil používateľom rôznych služieb komunikujúcich pomocou fedmsg nastaviť preferovaný spôsob zasielania upozorňovaní a ich filtrovanie. Okrem toho má existencia Fedora Notifications predísť nutnosti opakovanej implementácie kódu potrebného na odosielanie e-mailov v každej aplikácií v rámci projektu Fedora.

Aplikácia Fedora Notifications ponúka tvorbu filtrov zložených z viacerých pravidiel. Okrem všeobecných pravidiel na filtrovanie určitého balíka, používateľa alebo reťazca, ktorý zodpovedá regulárnemu výrazu, sa tu nachádza sada pravidiel špecifických pre FAF. Tieto pravidlá umožňujú nastaviť filter na základe počtu výskytov hlásenia alebo problému, vypnúť zasielanie upozornení na výskyt nového hlásenia a niekoľko ďalších filtrov.

Vývoj akejkoľvek logiky na správu upozornení v rámci FAF by bol duplicitou aplikácie Fedora Notifications a bol by v rozpore s ideou tejto aplikácie. Ako vhodné riešenie sme zvolili pridanie odkazu na Fedora Notifications na hlavnú lištu základnej stránky FAF.

6.5 Priraďovanie komponentov problémom

Pri vhodnej analýze množiny backtrace v probléme by bolo možné vytvoriť logiku na určovanie komponentu, v ktorom sa s vysokou pravdepodobnosťou nachádza problematická funkcia, respektíve riadok kódu. Priradenie tohoto komponentu skutočne zodpovednému za problém by zásadne zlepšilo prehľadnosť a fungovanie celého systému.

Komplikáciou v tomto smere je fakt, že pri tvorbe backtrace ku konkrétnemu hláseniu je zaznamenávaný súbor, funkcia a riadok každého rámca, chýba tu však informácia o komponentoch operačného systému, ktorých súčasťou sú dané súbory. Po určení súboru, v ktorom sa pravdepodobne nachádza jadro problému by nebolo možné určiť komponent, ktorý má byť problému priradený.

Pre pridanie chýbajúcich dát do backtrace by boli potrebné rozsiahle zmeny nástroja satyr [16] v nízko-úrovňovom kóde, ktorý beží priamo na zariadení, kde k pádu došlo. Keďže sa tento nástroj skladá z modulov pre jednotlivé typy pádov a neodchytených výnimiek, bolo by nevyhnutné pridať zber dodatočných informácií do každého z modulov. Implementácia automatizovaného riešenia tohoto nedostatku by bola enormne časovo náročná.

V rámci tejto práce sme sa rozhodli urobiť aspoň prvý krok v tomto smere, formou umožnenia manuálnej zmeny komponentov problému. V prípade, že správca určitého komponentu rozozná medzi problémami prípad, ktorý je na základe backtrace prislúchajúcich hlásení spôsobený súborom patriacim inému komponentu, môže tento komponent, prípadne množinu komponentov problému priradiť a prepísať tak pôvodne priradené komponenty. Po vykonaní tejto akcie sa problém viac nebude zobrazovať pri filtrovaní pôvodne priradeného komponentu. Správcom komponentov, ktorým je problém manuálne priradený sa v prehľade problémov problém objaví s informáciu, ktorý používateľ komponenty problému zmenil.

Na umožnenie tejto funkcionality bola opäť potrebná úprava databázového modelu a tvorba Alembic skriptu podobne ako v prípade archívu reportov. Nová tabuľka **problem-reassign** obsahuje dátum priradenia komponentov, väzbu na problém, ktorého sa zmena týka a na používateľa, ktorý zmenu vykonal.

Cieľom tohoto vylepšenia je urýchliť spoluprácu a komunikáciu medzi vývojármi pracujúcimi s FAF, umožniť odstránenie problémov, ktorých príčina sa nachádza v iných komponentoch z prehľadu problémov každého z používateľov a zamedziť situáciám, keď je otvorený bug na základe dát z FAF pre komponent, ktorý iba využíva knižnicu v ktorej sa jadro problému nachádza.

6.6 Nasledujúce kroky a testovanie

Táto kapitola bola venovaná návrhu konkrétnych riešení nedostatkov a vylepšení, ktoré boli v rámci práce implementované. Jedným z bodov, ktoré boli motiváciou pri tvorbe zadania práce bolo, aby sa výsledný kód stal príspevkom do open source projektu FAF. Prvým krokom bude teda odoslanie vytvorených patchov na posúdenie administrátorom v upstreame tohoto projektu. Po prípadných úpravách, ktoré budú po posúdení vyžiadané a ukončení diskusie v rámci komunity vývojárov prispievajúcich do FAF by sa mala finálna verzia patchov stať súčasťou kódu v upstreame. Po akceptovaní sa zmeny dostanú do novej verzie FAF a budú nasadené na produkčnom serveri. Tento proces môže trvať niekoľko mesiacov.

Jednotlivé vylepšenia a nové funkcie boli po implementácií testované na vzorke reálnych dát z databázy FAF a konzultované s niekoľkými členmi vývojového tímu ABRT. FAF má však veľké množstvo používateľov, ktorí sa zaoberajú rôznym typom softvéru od kernelu, cez dynamické jazyky po grafické knižnice a množstvo iných oblastí. Každý z používateľov má pri práci s FAF iné postupy a teda aj rozdielne požiadavky. Z tohoto dôvodu bude dôležité vyhodnotiť spätnú väzbu od členov komunity po nasadení zmien a na základe záverov, ktoré bude možné urobiť pokračovať vo vývoji vytvorených nástrojov.

Kapitola 7

Záver

Hlavným cieľom tejto bakalárskej práce bolo vyhodnotiť potreby používateľov FAF pri analýze veľkého množstva chybových hlásení, následne navrhnúť a implementovať sadu nástrojov na odstránenie najvýraznejších nedostatkov aplikácie a celkové zlepšenie používateľskej skúsenosti.

Po zozbieraní podnetov od členov komunity Fedora pracujúcich s FAF a porovnaní čiastkových riešení použitých v existujúcich aplikáciách s podobným účelom bolo navrhnutých niekoľko nástrojov na zlepšenie a rozšírenie funkcionality aplikácie. Tieto vylepšenia boli úspešne implementované a vyskúšané na reálnych dátach. Niektoré z pridaných funkcií poskytujú možnosť ďalšieho rozširovania v prípade pozitívnej spätnej väzby od používateľov po nasadení na produkčnej inštancií FAF. Okrem riešení, ktoré boli v rámci práce implementované boli počas analýzy nedostatkov načrtnuté ďalšie vylepšenia, ktoré môžu byť základom návrhu pri budúcom vývoji aplikácie.

Pridané zlepšenia prinášajú používateľovi viacero benefitov. V prvom rade je to skrátenie doby získavania základnej tabuľky problémov. Čas trvania dotazu sa pri veľkých objemoch dát podarilo skrátiť zo stoviek na štyri až päť sekúnd, vďaka čomu bolo možné odstrániť stránkovanie na úrovni dotazu, poskytnúť okamžite všetky dáta a pridať možnosti flexibilného vyhľadávania a zoraďovania týchto položiek bez nutnosti ďalšieho prenosu dát zo servera.

Významným zlepším možností práce s FAF je aj pridanie dvojice funkcií na interakciu používateľa so systémom, vo forme archívu neužitočných hlásení a možnosť zmeny priradených komponentov problému. Okrem týchto zmien boli vykonané malé zmeny v logike a štruktúre prezentovania dát a pridanie odkazu na systém správy upozornení v rámci Fedora Infrastructure.

Literatúra

[1] *ABRT project documentation*. 2014, [Online; navštívené 05.03.2017].
URL http://abrt.readthedocs.io/en/latest/

[2] Arapov, A.; Moskovcak, J.; Prikryl, Z.: *Automatic bug reporting tool*. 05 2009.
URL https://www.google.com/patents/US8694831

[3] Bayer, M.: *Alembic's documentation*. 2010.
URL http://alembic.zzzcomputing.com/en/latest/

[4] *DataTables*.
URL https://datatables.net/

[5] *Fedora Analysis Framework*.
URL https://retrace.fedoraproject.org/faf/summary/

[6] *Federated Message Bus*. [Online; navštívené 1.04.2017].
URL http://www.fedmsg.com/en/latest/

[7] *Fedora Notifications*. [Online; navštívené 2.04.2017].
URL https://apps.fedoraproject.org/notifications/about

[8] *Bugzilla*. 2008.
URL https://fedoraproject.org/wiki/Bugzilla

[9] Grady, R. B.: *Software Failure Analysis for High-Return Process Improvement Decisions*. Hewlett-Packard Journal, Srpen 1996, [Online; citované 05.03.2017].
URL http://www.hpl.hp.com/hpjournal/96aug/aug96a2.pdf

[10] Grinberg, M.: *Flask Web Development*. O'Reilly Media, 2014, ISBN 978-1-449-37262-0.

[11] *libreport*.
URL https://github.com/abrt/libreport

[12] *Mozilla Crash Reporter*. [Online; navštívené 5.03.2017].
URL http://kb.mozillazine.org/Breakpad

[13] Myers, J.; Copeland, R.: *Essential SQLAlchemy*. O'Reilly Media, 2016, ISBN 978-1-491-91646-9.

[14] *Patternfly*.
URL https://www.fullstackpython.com/object-relational-mappers-orms.html

[15] Ronacher, A.: *Flask's documentation*. 2010.
URL http://flask.pocoo.org/docs/

[16] *satyr*.
URL https://github.com/abrt/satyr

[17] *ErrorTracker*. Ubuntu wiki, [Online; navštívené 5.03.2017].
URL https://wiki.ubuntu.com/ErrorTracker

Prílohy

Príloha A

Obsah priloženého pamäťového média

A.1 Partícia thesis

Na tejto partícií sa nachádza súborový systém FAT32.

A.1.1 Thesis

Adresár obsahujúci zdrojové súbory technickej správy a výsledný dokument vo formáte PDF.

A.1.2 README.pdf

Súbor obsahujúci manuál k lokálnemu spusteniu aplikácie a obsahu disku.

A.2 Partícia demo

Na tejto partícií sa z dôvodu mapovania databázového systému nachádza súborový systém ext4.

A.2.1 FAF_demo

Adresár obsahujúci dátové úložisko a docker-compose skript na lokálne spustenie rozšírenej aplikácie FAF a databázového systému.

Príloha B

Manuál

Táto príloha obsahuje návod na lokálne spustenie aplikácie FAF rozšírenej o množinu nástrojov implementovanú v rámci tejto práce.

B.1 Inštalácia Docker CE

Pre inštaláciu Docker CE prosím nasledujte inštrukcie na oficiálnych webových stránkach projektu https://www.docker.com/community-edition pre vašu platformu. Následne je potrebné nainštalovať nástroj Docker Compose, inštrukcie je možné nájsť na stránke https://docs.docker.com/compose/install/.

B.2 Spustenie FAF

Pre spustenie kontajnera s inštanciou FAF, kontajnera s databázovým systémov a ich prepojenia je potrebné:

1. nastaviť prihlasovacie heslo pre databázový server prostredníctvom premennej prostredia FAF_DB_PASSWORD, napríklad `export FAF_DB_PASSWORD=secretpassword`
2. spustiť príkaz `docker-compose up` v adresári FAF_demo na priloženom pamäťovom médiu s právami roota.

Na inicializáciu aplikácie je potrebné pripojenie k internetu, pre stiahnutie potrebných Docker images. Po úspešnom rozbehnutí oboch kontajnerov je aplikácia dostupná v na adrese localhost a porte 5000. Na účely otestovania funkcií pre prihlásených používateľov je možné využiť login **faftest** a heslo **3K9M7B9XVC1vqDPJ**.