EKONOMICKÁ UNIVERZITA V BRATISLAVE

Fakulta hospodárskej informatiky

Evidenčné číslo: 103004/B/2017/36086129773340932

PRODUKTIVITA PRÁCE V PROGRAMOVACÍCH JAZYKOCH

Bakalárska práca

2017 Pavol Hriňa

EKONOMICKÁ UNIVERZITA V BRATISLAVE

Fakulta hospodárskej informatiky

PRODUKTIVITA PRÁCE V PROGRAMOVACÍCH JAZYKOCH

Bakalárska práca

Študijný program: Hospodárska informatika

Študijny odbor: Hospodárska informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: Ing. Igor Bandurič, PhD.

Bratislava 2017 Pavol Hriňa

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: .....

Podpis študenta

Poďakovanie

Chcel by som poďakovať mojmu školiteľovi Ing. Igorovi Banduričovi, PhD., za jeho ochotu, odborné názory a spätnú väzbu na bakalársku prácu. Taktiež sa chcem poďakovať ostatným ľuďom, s ktorými som túto prácu konzultoval.

ABSTRAKT

HRIŇA, Pavol: Produktivita práce v programovacích jazykoch. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky, katedra aplikovanej informatiky. – Vedúci záverečnej práce: Ing. Igor Bandurič, PhD. – Bratislava: FHI EU, 2017, počet strán 43.

Cieľom záverečnej bakalárskej práce je vytvoriť webovú aplikáciu pre pridávanie a pripájanie sa k podujatiam v dvoch rôznych programovacích jazykoch, a porovnať produktivitu práce v oboch. V práci sme použili webový framework Django, ktorý je napísaný v programovacom jazyku Python, a webový framework Ruby on Rails, ktorý je založený na jazyku Ruby. Práca je rozdelená do troch kapitol. V prvej časti záverečnej práce je obsiahnutá problematika produktivity práce v programovacích jazykoch. Definovali sme si rôzne druhy merania produktivity, a taktiež faktory ovplyvňujúce produktivitu v programovaní. V závere kapitoly sme si zadefinovali použité programovacie jazyky a ich frameworky. V druhej kapitole je definovaný cieľ práce, metódy skúmania použité v práci a spôsob sčítavania. V tretej kapitole sme si zhodnotili vytvorenú webovú aplikáciu, namerané hodnoty a vyriešili výsledok práce. Výsledkom riešenia danej problematiky je určenie programovacieho jazyka s vyššou produktivitou práce

Kľúčové slová: produktivita práce v programovacích jazykoch, Ruby on Rails, Django

ABSTRACT

HRIŇA, Pavol: Productivity in Programming Languages. – The University of Economics in Bratislava. Faculty of Business Informatics, Department of Applied Informatics. – Supervisor of the bachelor thesis: Ing. Igor Bandurič, PhD. – Bratislava: FHI EU, 2017, number of pages 43.

The aim of the final bachelor thesis is to create a web application for adding and connecting to events in two different programming languages, and to compare the productivity of work in both. We used the Django web framework written in Pytho programming language and the web framework Ruby on Rails, based on Rubz language. The thesis is divided into three chapters. In the first part of the final paper the issue of productivity in programming languages is included. We've defined different types of productivity measurement, as well as factors affecting productivity in programming. At the end of the chapter we have defined both programming languages and their frameworks. In the second chapter, we defined the goal of the work, the methods of study used in the work and the method of evaluating. In the third chapter, we evaluated the created web application, the measured values and solved the result of the work. The solution to this problem is the determination of programming language with higher productivity

Keywords: Productvity in Programming Languages, Ruby on Rails, Django

Zoznam obrázkov

Zoznam tabuliek

Zoznam vzorcov

Obsah

Úvod

„Softvérová účinnosť sa znižuje na polovicu každých 18 mesiacov a kompenzuje tak Moorov zákon.“. Autorom tohto výroku bol Niklaus Wirth, tvorca programovacieho jazyka Pascal. Uvedený výrok z časti vystihuje aktuálnu situáciu v informačných technológiách a upozorňuje na rýchly priam až exponenciálny vývoj tejto oblasti.

JavaScript? PHP? Aké sú možnosti, keď sa niekto rozhodne pre programovanie webovej aplikácie? Prečo by niektoré jazyky mali byť uprednostnené pred inými? Od zrodu internetu sa vytváranie webových stránok stalo veľmi lukratívnou a vyhľadávanou zručnosťou. Na základe spoločnosti, ktorá sa bez webu dnes už nevie zaobísť, trend stále stúpa. Vzdelávacie platformy, akou je napr. Udemy (poskytovateľ online kurzov), robia učenie programovacích zručnosti jednoduchšie a prístupnejšie. Dôležité je najmä vybrať si ten správny programovací jazyk na učenie. Aké sú teda kritéria pre výber toho správneho programovacieho jazyka?

Produktivita práce tradične odkazuje na pomer medzi množstvom vyrobeného tovaru a nákladmi vynaloženými na jeho vytvorenie. Pri vývoji softvéru sú však veci zložitejšie ako pri výrobe tovaru. Vývoj softvéru sa chápe ako inžiniersky proces. Existuje veľké množstvo faktorov, ktoré ovplyvňujú produktivitu jednotlivcov a tímov. Použitý proces vývoja softvéru, štýl písania kódu alebo spôsob zálohovania, môže vo veľkej miere ovplyvniť efektivitu jednotlivca a taktiež skupiny.

Prvá časť našej bakalárskej práce je venovaná vymedzeniu produktivity práce v programovacích jazykoch Obsahuje taktiež rozbor hlavných časti programovacích jazykoch Python a Ruby, ich históriu a vznik, definíciu syntaxu a použitie. V rámci prvej časti práce si taktiež bližšie zadefinujeme použité webové frameworky Django a Rails. Druhá časť práce sa venuje definovaniu cieľa práce a použitej metodiky. V tretej kapitole sú znázornené výsledky práce.

1 Súčasný stav riešenej problematiky

Vplyvom informačných technológii môžeme rozvoj sveta popísať Moorovim zákonom, ktorý hovorí o exponenciálnom raste technológii. Žijeme v storočí, ktoré je popisované veľkými zmenami, hlavne vďaka informačným technológiám. Život sa výrazne zmenil po nástupe počítačov a internetu. Na súčasných programátorov je vystavený enormný tlak, kvôli nepretržitému vývoju vo svete informačných technológii, a preto sa musia prispôsobiť novým trendom a neustále aplikovať potrebné inovácie. Aj kvôli tomu sa kladie veľký dôraz na správny výber programovacieho jazyka, aby sa stíhalo držať krok s trendom.

1.1 Produktivita práce v programovacích jazykoch

Predpokladom pre zvládnutie a zvýšenie produktivity je schopnosť merania a porovnávania s normami a internými referenčnými kritériami. Všetky procesy, ktoré majú vplyv na produktivitu, je potrebné posúdiť s cieľom identifikovania potenciálnych možností zlepšenia, čo môže viesť k optimálnemu výkonu.

1.1.1 Definovanie pojmov

Na základe noriem IEEE z roku 1993 pre metriky softvérovej produktivity je "produktivita definovaná ako pomer výstupného produktu v závislosti od vstupného materiálu spotrebovaného v produkcii." Aj keď je zložité tento pomer patrične kvantifikovať k schopnosti riešenia problémov v programovaní, bol rozsiahle študovaný v kontexte softvérového inžinierstva.

V šesťdesiatych rokoch uskutočnil Edward Nelson jednu z prvých štúdií zameranú na identifikáciu faktorov produktivity práce programátorov[1]. Nelson zistil, že produktivita programátorov sa vzťahuje na najmenej 15 faktorov. V roku 2000 Capers Jones identifikoval

[1] EDWARD NELSON, V 1966. Management handbook for the estimation of computer programming costs. Systems Develioment Corporation, 1993

približne 250 faktorov, o ktorých tvrdil, že ovplyvňujú výkon programátora[2]. Zhrnutím týchto dvoch výskumov Endres a Rombach uvádzajú, že zníženie na 10 až 15 parametrov by mohlo znamenať výrazné zjednodušenie v meraní produktivity v programovaní. [3]S takým množstvom faktorov ovplyvňujúcim merania nie je vôbec prekvapením, že sa v literatúrach uvádza nespočetné kvantum ukazovateľov produktivity.

Všeobecný cieľ programátorov je zvýšenie rýchlosti tvorby softvéru s rovnakou pracovnou silou, bez degradácie a s prípadným zlepšením kvality softvéru. Vo všeobecnosti existujú dva spôsoby na dosiahnutie tohto cieľa. Po prvé, môžeme zvýšiť efektivitu vývojárov tým, že im poskytneme programovacie jazyky a nástroje, ktoré zlepšia programovú produktivitu. Po druhé, môžeme rozšíriť komunitu vývojárov tým, že sprístupníme programovanie širšej verejnosti. Používanie higherlevel programovacích jazykoch a programovacích rozhraní podporuje obidve navrhnuté stratégie. Začlenením vyššej úrovne abstrakcie programovacích jazykoch má za následok zrýchlenie a zjednodušenie vývoja aplikácii. Musíme však zabezpečiť, aby tieto výhody neprišli na úkor výkonu vytvoreného produktu. Aplikácie napísané v high-level programovacích jazykoch, ktoré sú určené na riešenie veľkých problémov na paralelných strojoch, nesmú byť výrazne menej účinné, ako tie isté aplikácie napísané v lower-level programovacích jazykoch. Ak sú, potom je pravdepodobné, že programovací jazyk nebude na prácu akceptovaný. To je jeden z kameňov úrazu higher-level programovacích jazykoch, a preto musí analýza produktivity zahŕňať metriky úsilia a výkonnosti v oblastí programovania. Okrem toho musia byť tieto dve metriky prepojené, aby bolo možné správne vyhodnotiť kompromis medzi silou jazyka a efektívnosťou programu.

Podobne, ak majú byť prijaté high-level programovacie jazyky, programy napísané v nich nemôžu vykazovať viac chýb v programe, spotrebovať viac pamäte alebo byť menej prenosné, ako keby boli napísané prostredníctvom low-level konkurentom. To znamená, že pre každé vývojové zadanie musí byť programovací jazyk ohodnotený s ohľadom na aspoň dve kritéria: čas a úsilie potrebné na písanie alebo / a výkonnosť kódu, ktorú prinesie výsledok.[4]

[2] CAPERS JONES, V 1966. Management handbook for the estimation of computer programming costs. Systems Develioment Corporation, 1993
[3] ALBERT ENDRES A DIETER ROMBACH, V 2003. Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. [online] [190 - 192]. Pearson Education Limited, Harlow, England. 2003
[4] KEN KENNEDY, CHARLES KOELBEL,V 2014. ROBERT SCHREIBER, Defining and measuring the

1.1.2 Základné požiadavky metrík produktivity

Požadovanou vlastnosťou metriky produktivity softvéru je schopnosť porovnávať veľké množstvo rôznych systémov. Navyše výpočet metriky musí spôsobovať len nízke náklady a to vedie k štyrom kľúčovým požiadavkám:

- Metrika by mala platiť pre mnoho rôznych programovacích jazykoch.
- Úsilie (náklady a čas) na vykonanie by mali byť nízke.
- Metrika musí byť objektívna a opakovateľná. To znamená, že zhodnotenie toho istého systému vykonaného rôznymi jednotlivcami musí priniesť rovnaké výsledky.
- Chyby pri meraní by sa mali vylúčiť.

Značí to použitie metrík, ktoré môžu byť aspoň z časti automatizované. Ďalej si ukážeme, aké ukazovatele výkonnosti sa dajú zahrnúť do celkového vyhodnotenia produktivity programovacích jazykoch.

1.1.3 Druhy merania produktivity

1.1.3.1 Metriky času a nákladov

Zvyšovaním produktivity sa rovnako snažíme znížiť celkový čas na riešenie problému *P*, ktorý označujeme *T(P).* To znamená, že chceme minimalizovať

$$T(P) = I(P) + rE(P)$$

Vzorec 1: Celkový čas problému

productivity of programming languages.[online].
Dostupné na internete: < https://www.cs.indiana.edu/~achauhan/Teaching/B629/2006-Fall/CourseMaterial/2004-ijhpca-kennedy-productivity.pdf >

kde *P* je problém záujmu, *I(P)* je čas potrebný na vyriešenie problému *P*. Veličina *E(P)* je priemerná doba realizácie za chod programu a *r* je váhový faktor špecifický pre daný problém, ktorý odzrkadľuje relatívnu význam minimalizácie vykonávacieho času v porovnaní s časom implementácie. Preto *r* bude väčšie pre programy, ktoré majú bežať mnohokrát bez akejkoľvek zmeny.

Ak je vhodnejšie zamerať sa na náklady za vygenerované riešenie než na čas za riešenie, dostaneme sa v podstate na rovnaký model.

$$C(P) = D(P) + rM(P)$$

Vzorec 2: Celkové náklady problému

kde *D(P)* je cena vývoja kódu aplikácie a *M(P)* je priemerná cena strojového času pre jeden cyklus aplikácie. Z dôvodu izomorfizmu[i] medzi časovými a nákladovými modelmi, väčšina navrhnutých stratégii môžu byť použité na oba problémy.

Formulácia v rovnici 1 predpokladá, že implementácia a vykonanie sú činnosti, ktoré sa nebudú prekrývať. Je to jednoznačne zjednodušenie riešenia, ktoré nám vyhovuje. Avšak vzorec nám ponecháva obťažný problém pri výbere vhodnej hodnoty pre *r,* ktorá odhadne pre konkrétnu aplikáciu, koľko krát bude daná aplikácia spustená. Ak ide o aplikáciu, ktorá má byť spustená iba raz, potom *r* môže byť 1, ale väčšina aplikácii bude spustená mnohokrát bez zmien medzi sériami, s výnimkou zmien údajov a drobných zdrojových vylepšení. Preto je dôležité zabezpečiť, aby *r* nebolo podceňované. Ako viete koľko krát bude program používaný v priebehu jeho života? Pre získanie odpovede sa musíme spoliehať na skúsenosti svoje a iných. Všetci výkonnostní programátori musia implicitne robiť a používať odhad tohto parametra. Pri každej snahe o ladenie programu sa ďalšia práca zastaví v bode znižovania výnosov, kde dodatočný čas vývoja nie je kompenzovaný predpokladaným znížením času vykonania ako vážene odhadom. Zatiaľ čo *r* nie je známe (okrem výnimočných prípadov, kde už program je vyradený), je možné ho užitočne odhadnúť. Všimnite si, že formulácia 1 neobsahuje pojem času kompilácie. V bežných jazykoch je čas kompilácie vo všeobecnosti malý vzhľadom na čas chodu kódu. Pokročilejšie jazyky však môžu vyžadovať celo-programovú kompiláciu, čo by mohlo výrazne zvýšiť čas kompilácie na oplátku za zvýšenie dĺžky štartovania programu. Na druhom konci tohto spektra

interpretované jazyky rušia časovú penaltu kompilácie, hoci na úkor času spustenia; aplikácie, ktoré vykazujú nízke *r* alebo nízke *E(P),* výsledný zisk používateľskej produktivite určite pomáha vysvetliť ich popularitu (Odhaduje sa, že Matlab má približne jeden milión používateľov). To vyvoláva otázku, či by mal byť čas kompilácie zahrnutý do modelu. Ak áno, mal by to byť nezávislý termín alebo mal by byť nejakým spôsobom spojený s časom realizácie alebo časom implementácie? Tento termín zahrnieme ako tretí, nezávislý termín. Kvôli tomu tu máme ďalší, podrobnejší model časového riešenia

$$T(P) = I(P) + r_1 C(P) + r_2 E(P)$$

Vzorec 3: Celkový čas problému v závislosti od doby kompilácie

Kde *C(P)* je priemerná doba kompilácie, r_1 je počet kompilácii a r_2 je počet cyklov. Tento model sa hodí na riešenia, kde je čas kompilácie prvoradým problémom.

1.1.3.2 Metriky počtu riadkov (Lines of Code - LoC)

"Measuring software productivity by lines of code is like measuring

progress on an airplane by how much it weighs."- Bill Gates

Počítanie riadkov kódu je jednou z najstarších a najpoužívanejších softvérových metrík na odhad veľkosti softvérového systému[5]. Opakovane sa tvrdilo, že táto metrika nedostatočne zachycuje zložitosť softvérových systémov. Z toho dôvodu sa považuje za nesprávne spoliehať sa na riadky kódu, keď ide o posúdenie produktivity vývojárov alebo komplexnosť projektu. Vylúčením nadbytočných časti kódu z tejto metriky, kombinovaním výsledku s celkovým potrebným úsilím a mierou defektu vo výsledku vieme vyskladať vysoko objektívny a efektívny, merateľný ukazovateľ produktivity.

Pre čo najrelevantnejšie vyhodnotenie pomocou danej metódy by sa v celkovom počte riadkov zdrojového kódu nemali nachádzať nasledujúce triedy kódov:

[5] C. E. WALSTON AND C. P. FELIX, V 1977. A method of programming measurement and estimation [54-73]. IBM Systems Journal, 1977

- Opakovane použitý kód (napr. externé knižnice)
- Testovací kód (nie je súčasťou konečného produktu)
- Redundantný kód
- Mŕtvy kód (nedosiahnuteľný kód)
- Generovaný kód

Štandardné formátovanie kódu sa musí použiť na zdrojový kód pred uskutočnením počítania.

1.1.3.3 Metrika počítania defektov

Všeobecne sa uznáva, že vytváranie kódu je oveľa drahšie ako implementácia "rýchleho a špinavého" riešenia, ktoré môže obsahovať značné množstvo chýb. Na zabezpečenie spravodlivosti metriky je možné navzájom porovnávať iba projekty, ktoré vykazujú podobnú chybovosť.

1.1.3.4 Metrika Man-days

Počet man-day-ov zahŕňa všetko úsilie, ktoré bolo vložené na vypracovanie projektu. Zahŕňajú sa tu také činnosti ako sú administratívne úlohy, analýza požiadaviek, implementácia alebo testovanie. Zvažujú sa tu taktiež nadčasy a neplatená práca.

1.1.3.5 Metrika funkčných bodov (Function points)

Analýza funkčných bodov (AFB) nie je nový koncept. Koncepcia funkčných bodov predstavil Alan J. Albrecht v roku 1979. V roku 1984 skupina medzinárodných funkčných bodov (IFPUG) bola zriadená s cieľom objasniť pravidlá, stanoviť normy a podporiť ich používanie a vývoj.[6] V súčasnosti však väčšina ostáva neznáma alebo ignorovaná.

[6] A. ABRAN AND P. N. ROBILLARD, V 1994. Function points: A study of their measurement processes and scale transformations [171-184]. Journal of Systems Software, vol. 25, 1994

Technológia analýzy funkčných bodov kvantifikuje používateľské funkcie obsiahnuté v softvérovej aplikácii v zmysle, ktorý je pre používateľa softvéru zmysluplný. Body sú časovo rozlíšené pre každú funkčnosť implementovanú v softvéri, pričom viac bodov je priradených k zložitejším funkciám (užívateľ si zvolí podmienky). Analýza funkčných bodov sa priamo týka business požiadaviek, ktoré ma softvér riešiť. Preto sa dá ľahko aplikovať v rôznych vývojových prostrediach a v rôznej fáze životného cyklu vývoja softvéru – od fázy predčasných požiadaviek až po prijatie zákazníkom. AFB zvyšuje svoju popularitu ako štandardná technika pre dimenzovanie softvéru.[7]

1.1.4 Faktory ovplyvňujúce produktivitu

V tejto časti si zadefinujeme faktory, ktoré výraznejšie ovplyvňujú produktivitu práce jednotlivca alebo skupiny. Patria sem:

- Vlastnosti produktu. Proces a výroba ovplyvňujú pracovnú produktivitu programátorov jednotlivcov, ako aj vývojových tímov.
- Účasť používateľa, skúsenosti programátora s programovacím jazykom.
- Faktory ako je tímová kultúra, identita, súdržnosť, podpora inovácií a tímová komunikácia ovplyvňuje tímovú produktivitu.[8]
- Veľkosť tímu, používaný programovací jazyk (3GL alebo 4GL), vývojová platforma a vývojové techniky. Veľkosť tímu ma vplyv na produktivitu v tíme a na náklady na projekt
- Firemné procesy, organizačná štruktúra, fyzické zariadenie, podniková vízia, náročnosť zadania. [9]

[7] H. K. RAJU AND Y. T. KRISHNEGOWDA, V 2013. Software Sizing and Productivity with Function Points [1-5]. Lecture Notes on Software Engineering, Vol. 1, No. 2. May 2013
[8] WAGNER, S., & RUHE, M., V 2008. A Systematic Review of Productivity Factors in Software Development. State Key Laboratory of Computer Science in 2008
[9] GOPARAJU PURNA SUDHAKARA, AYESHA FAROOQBAND SANGHAMITRA PATNAIKC, V 2012. Measuring productivity of software development teams [65 – 75]. Serbian Journal of Management 7, 2012

1.2 Meranie výkonu a efektívnosti aplikácie

Čas vývoja programu a čas vykonávania sú ovplyvňované výberom programovacieho jazyka. Pritom je dôležité zvážiť plnú silu nového jazyka. Kód je možné napísať vo formáte Fortran v napr. jazyku Java, ale to by nám povedalo veľmi málo o sile a účinnosti Java programovacieho jazyka. To znamená, že pri porovnávaní dvoch programovacích modelov na rovnakom aplikačnom meradle je nevyhnutné, aby aplikácia bola kódovaná v prirodzenom štýle pre každý programovací model. Cieľom nie je ukázať, že môžete napísať program s porovnateľným výkonom v high-level programovacom jazyku, ale skôr merať výkonové náklady zaplatené pri písaní aplikácie v najprirodzenejšom štýle. Preto je najlepšie, aby rôzne verzie aplikácii napísali vývojári bez znalosti o tom, ako optimalizačné kompilátory pracujú pre daný model.[10]

Je tiež dôležité, aby bol súbor referenčných hodnôt reprezentatívny pre skutočné aplikácie bežiace v systéme, aby sa vývojári systému neopravené neuplatňovali.

Pri presne meraných hodnotách realizácie je jednoduché vypočítať ε z definície:

$$\varepsilon = \frac{E(P)_0}{E(P)_L}$$

4 Výpočet relatívnej efektivnosti

kde ε označuje relatívnu efektívnosť, E označuje čas realizácie, P_0 je verzia daného programu napísaná odborníkom v štandardnom programovacom jazyku, P_L v high-level programovacom jazyku.

[10] KEN KENNEDY, CHARLES KOELBEL,V 2014. ROBERT SCHREIBER, Defining and measuring the productivity of programming languages. Dostupné na internete: < https://www.cs.indiana.edu/~achauhan/Teaching/B629/2006-Fall/CourseMaterial/2004-ijhpca-kennedy-productivity.pdf >

1.3 Programovací jazyk Python

Po zostrojení počítačov v skorých rokoch, museli byť následne naprogramované. Vedci predstavili binárne programovacie jazyky (1 a 0) ako riešenia, ale bolo to veľmi zložité a časovo náročné. Tento problém zapríčinil vývoj v programovacích jazykoch a viedol k vytváraniu high-level jazykov.

1.3.1 História programovacieho jazyka Python

Python bol koncipovaný koncom osemdesiatych rokov a jeho implementácia začala v decembri 1989 prostredníctvom pána Guildo van Rossum z Holandska ako nástupca jazyka ABC, schopného spracovania výnimiek a prepojenia s operačným systémom Amoeba. Van Rossum je hlavným autorom Python a jeho pokračujúca ústredná úloha pri rozhodovaní o smerovaní Pythonu sa odzrkadľuje v názve, ktorý mu priniesla komunita Python – benevolentný diktátor života.[11]

1.3.2 Definícia

Medzi modernými high-level jazykmi sa Python ako všeobecný programovací jazyk dostal medzi výber programátorov. Jeho filozofia kladie dôraz na jednoduchú čitateľnosť kódu a syntax umožňuje programátorom vyjadrovať koncepty v nízkom počte riadkov kódu, na rozdiel od možnosti napr. v jazyku C. Jazyk poskytuje konštrukcie určené na písanie programov malého i veľkého rozsahu. Python podporuje viacero programovacích paradigiem vrátane objektovo-orientovaných a funkčných programov alebo procedurálnych štýlov. Obsahuje systém dynamického typu a automatickú správu pamäte a má rozsiahlu a komplexnú štandardnú knižnicu. Python sa väčšinou používa ako skriptovací jazyk, ale

[11] MASOUD NOSRATI, V 2011. Python: An appropriate language for real world programming. World Applied Programming, Vol (1), No (2) [110-117], June 2011. Dostupne na internete: < http://waprogramming.com/download.php?download=50ae4a1125d607.12866725.pdf>

používa sa aj v širokej škále kontextu bez skriptovania. Základná filozofia jazyka je zosumarizovaná do piatich bodov:

- Krásny vzhľad je lepší ako škaredý
- Explicitne volanie je lepšie ako implicitné
- Jednoduchosť je lepšia vlastnosť ako zložitosť
- Komplexne riešenie je lepšie ako komplikované
- Čitateľnosť sa cení

1.3.3 Syntax a sémantika

Syntax programovacieho jazyka Python je súbor pravidiel, ktorý definuje ako bude program Python napísaný a interpretovaný (systémom aj ľuďmi). Python bol navrhnutý tak, aby bol vysoko čitateľným jazykom. Má pomerne prehľadné vizuálne usporiadanie a používa anglické kľúčové slová.

Jednoduchosť programovania v Python je zobrazená na známom "Hello world" programe:

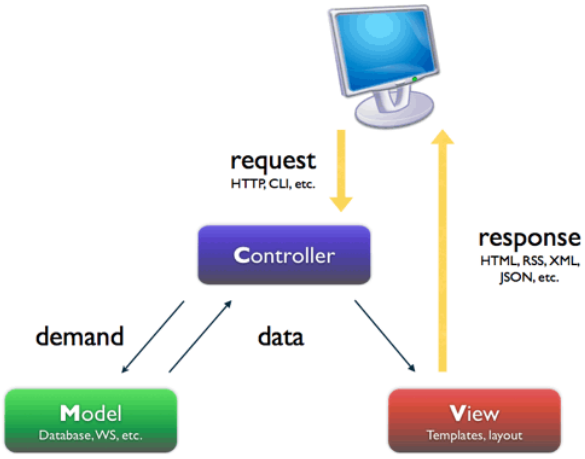
print("Hello World")

Jednou z veľkých výhod a rozdielov oproti iných programovacích jazykoch je, že používa medzery, skôr ako zátvorky alebo kľúčové slová na vymedzenie blokov.

1.3.4 Webový framework Django

Je bezplatný a open source webový rámec (web framework), ktorý je napísaný v jazyku Python a ktorý následuje architektronický vzorec Model-View-Template (MVT). Hlavným cieľom Djanga je uľahčiť vytváranie komplexných databázových webových stránok. Django zdôrazňuje opätovne využitie kódu, rýchly vývoj a princíp neopakovať sa pri písaní. Oproti Ruby on Rails však poskytuje zaujímavú možnosť automatickej tvorby administrácie projektu, ktorá sa vygeneruje dynamicky na základe dátového modelu. Jadro

frameworku Django obsahuje objektovo-relačný mapper, ktorý je sprostredkovateľom medzi dátovým modelom (definovaným ako triedu v Python) a relačnou databázou. [12]



1 Vysvetlenie Model-View-Controller (MVC)

Zdroj: https://cms-assets.tutsplus.com/uploads/users/263/posts/21627/image/mvc.png

[12] JEFF FORCIER, PAUL BISSEX, WESLEY CHUN, V 2009. Python Web Development with Django®. Addison-Wesley, USA, 2009

1.4 Programovací jazyk Ruby

Ruby je populárny dynamický skriptovací jazyk, ktorého cieľom je poskytnúť používateľom slobodu voľby medzi mnohými rôznymi spôsobmi, ako robiť to isté inak a dáva programátorom pocit prirodzenosti pri písaní kódu. Zatiaľ čo vyššie uvedené veci uľahčujú programovanie v Ruby, je ťažké vytvoriť nástroje na analýzu a transformáciu, ktoré by fungovali na zdrojovom kóde Ruby

1.4.1 História programovacieho jazyka Ruby

Programovací jazyk Ruby bol vyvinutý Yukihirom Matsumotom na univerzite v Japonsku. Dôvod bol ten, že autor chcel skriptovací jazyk, ktorý je výkonnejší ako Perl, ale objektívnejší ako Python. Vývoj začal vo februári 1993 a prvá alfa verzia Ruby bola vydaná v decembri 1994. Bola vyvinutá ako alternatíva k skriptovacím jazykom ako sú Perl a Python. Ruby si požičiava z Perl a knižnica tried je v podstate objektovo orientovaná reorganizácia Perlovej funkčnosti. Ruby si tiež požičiava z jazyka Lisp a Smalltalk. [13]

1.4.2 Definícia

Ruby je open-source objektovo orientovaný skriptovací jazyk. Vďaka jednoduchému syntaxu je pomerne ľahký na naučenie. Je však dostatočne výkonný a dokáže konkurovať známejším a starším skriptovacím jazykom, ako sú napríklad Python a Perl. Ruby sa radi ako prenosný, hoci bol vyvinutý na Linux, funguje tiež na iných OS ako sú UNIX, Windows, DOS a ďalších. Ruby sa nekompiluje (podobne ako Java alebo alebo C#), ale je kompilovaný počas behu. Hodí sa takmer všade, kde je potrebná efektívnosť, prehľadnosť ale nie rýchlosť.

[13] MICHAEL FURR, JONG-HOON (DAVID) AN, JEFFREY S. FOSTER, MICHAEL HICKS, V 2009. The Ruby Intermediate Language [online]. University of Maryland, College Park. Dostupný odkaz: < https://www.cs.umd.edu/projects/PL/druby/papers/druby-dls09.pdf>

1.4.3 Syntax a sémantika

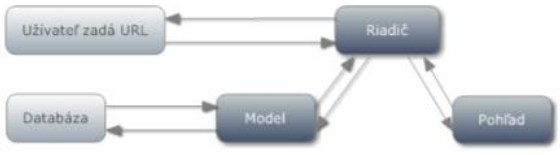
Syntax Ruby je v podstate podobný tomu v Perl alebo v Python. Definície triedy a metódy sú signalizované kľúčovými slovami, zatiaľ čo kódové bloky môžu byť definované kľúčovými slovami alebo zátvorkami. Jedným veľkým rozdielom Ruby v porovnaní s Python a Perl je to, že udržiava svoje premenné inštancie úplne súkromné pre triedu a dovoľuje k ním pristupovať iba cez prístupové metódy (GET / SET).

Ukážeme si taktiež vykonanie základného príkazu "Hello World" v Ruby:

puts 'Hello World'

1.3.4 Webový framework Rails

Rails je open source webová štruktúra postavená na programovacom jazyku Ruby (spolu názov Ruby on Rails). Umožňuje programátorovi ľahko vytvárať pokročilé databázové webové stránky s podpory a generovania kódu. To znamená, že ak programátor dodržuje určitú následnosť súborov, tak veľa funkcii bude pracovať hneď s malým množstvom napísaného kódu. Používa rovnako ako Python Model-View-Controller (MVC) koncept.[14] MVC je spôsob ako transparentne oddeliť rôzne časti aplikácie. Modelová časť sa stará o dáta a manipuláciu s nimi (napr. aj komunikácia s databázou). Controler v doslovnom preklade znamená riadič, ktorý je zodpovedný za riadenie aplikácie (interakcia medzi používateľom a serverom). View alebo doslovný preklad pohľad, je grafické rozhranie aplikácie.



2 MVC s vysvetlením

Zdroj: http://blog.thedigitalgroup.com/chetanv/wp-content/uploads/sites/23/2015/06/image_thumb.png

[14] RICK BORUP, V 2010. An Introduction to Ruby and Rails [online]. Dostupný odkaz: < http://www.ita-software.com/papers/borup_ruby_published.pdf>

2. Cieľ práce, metodika práce a metódy skúmania

Cieľ práce:

Cieľom práce je vytvoriť jednoduchý program v aspoň dvoch programovacích jazykoch (jedným z nich by mal byť jazyk Ruby) a porovnať produktivitu práce v oboch. (Pozn: Autor práce by nemal mať veľké skúsenosti ani s jedným z nich)

Pre potreby naplnenia daného cieľa boli stanovené nasledovné čiastkové ciele:

- Naštudovať problematiku:
  - programovacích prostredí Django (Python) a Rails (Ruby)
  - produktivity práce v programovacích jazykoch
- Vytvorenie webovej aplikácii v oboch programovacích jazykoch
- Definovanie metrík použitých na meranie produktivity v programovacích jazykoch
- Vyhodnotenie výsledkov meraní

Metodiky práce a metódy skúmania:

Pre zistenie výsledkov práce sme zostavili tabuľku s nasledovnými parametrami (pre oba programovacie jazyky). Počas celej doby programovania aplikácie sme hodnoty merali a simultánne zaznamenávali do tabuľky. Veľká časť meraných parametrov pochádza z teórie obsiahnutej v prvej kapitole. Ostatné sme si určili po dohode s pánom školiteľom.

Ide o nasledovné parametre:

- Kategória:
  - *Inštalácia prostredia*
  - *Učenie sa programovacieho jazyka*
  - *Písanie kódu*
  - *Analýza programu*
  - *Celková analýza programu*
  - *Prevádzka aplikácie*

- Metrika:
    - *Čas potrebný pre inštaláciu prostredia*
    - *Náročnosť inštalácie prostredia*
    - *Čas vynaložený vypracovaním tutoriálu*
    - *Náročnosť pochopenia tutoriálu*
    - *Náročnosť pochopenia jednotlivých príkazov*
    - *Čas vynaložený vypracovaním danej časti*
    - *Náročnosť vypracovania danej časti*
    - *Početnosť riadkov jednotlivých časti programu*
    - *Pridávanie dodatočných knižníc*
    - *Celkový počet chýb pri písaní kódu*
    - *Záujem na programovacom fóre www.stackoverflow.com*
    - *Dĺžka načítania stránky*
    - *Dĺžka zapnutia servera*
    - *Veľkosť projektovej zložky*
- Merná jednotka:
    - *Hodina*
    - *Škála od 1 do 5 (1 je najvyššia)*
    - *Počet (riadkov)*
    - *Počet otázok (približne)*
    - *Sekunda*
    - *Mb (Mega byte)*
- Časť:
    - *Príprava*
    - *Vypracovanie tutoriálu*
    - *Vytváranie, editácia a vymazávanie udalosti*
    - *Správa užívateľov - registrácia, prihlásenie, odhlásenie*
    - *Správa návštevnosti podujatí užívateľov*
    - *Grafická úprava*
    - *Práca s databázou SQLlite3*
    - *Analýza*
    - *Prevádzka*

Spôsob sčítavania:

Sčítavanie (pripočítavanie hodnôt) je neoddeliteľnou súčasťou analýzy údajov. Následne sme si pre každú kategóriu som si vyhodnotil favorita, ktorý získal bod v danej kategórii. Ak bol stav v kategórii vyrovnaný, tak nedostal bod žiaden programovací jazyk. Výsledkom produktivity práce je programovací jazyk s vyšším počtom bodov.

3 Výsledky práce a diskusia

Na dosiahnutie výsledkov, ktoré budú obsiahnuté v tejto kapitole, sme postupovali tak, že sme si najskôr naštudovali problematiku produktivity v programovacích jazykoch a určili metriky sledujúce pri programovaní. Neskôr sme si simultánne naprogramovali obe aplikácie, kde sme zároveň zaznamenávali určené metriky. V poslednom kroku sme vyhodnotili programovací jazyk , ktorý vykazoval vyššiu produktivitu práce.

3.1 Predstavenie webovej aplikácie

Webová aplikácia pre pridávanie a pripájanie sa k podujatiam rôzneho druhu.

3.1.1 Funkcionalita

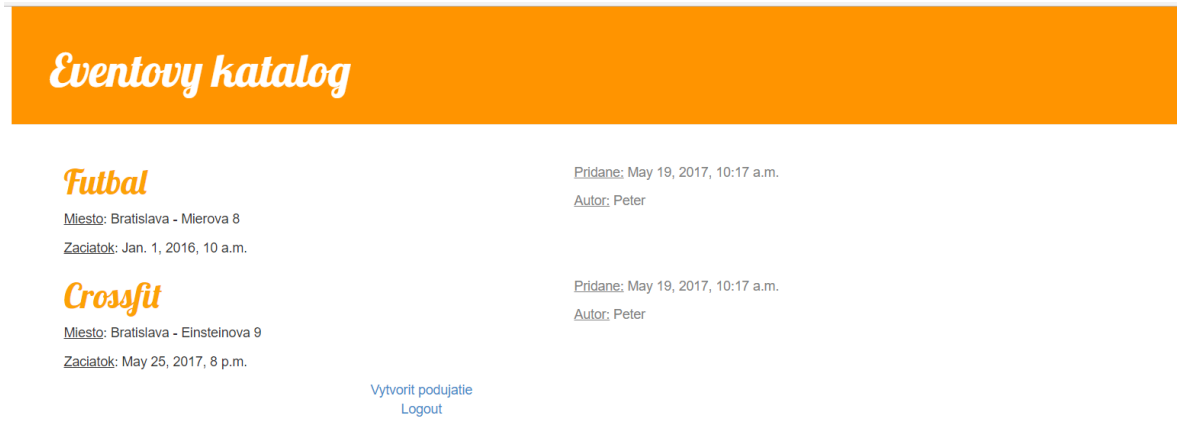
- *Správa užívateľov* - registrácia, prihlásenie, odhlásenie.
- *Správa podujatí* - vytváranie, editácia a vymazávanie udalosti.
- Možnosť pripojenia sa už k existujúcemu podujatiu a odpojenia sa

3.1.2 Predstavenie webovej aplikácie

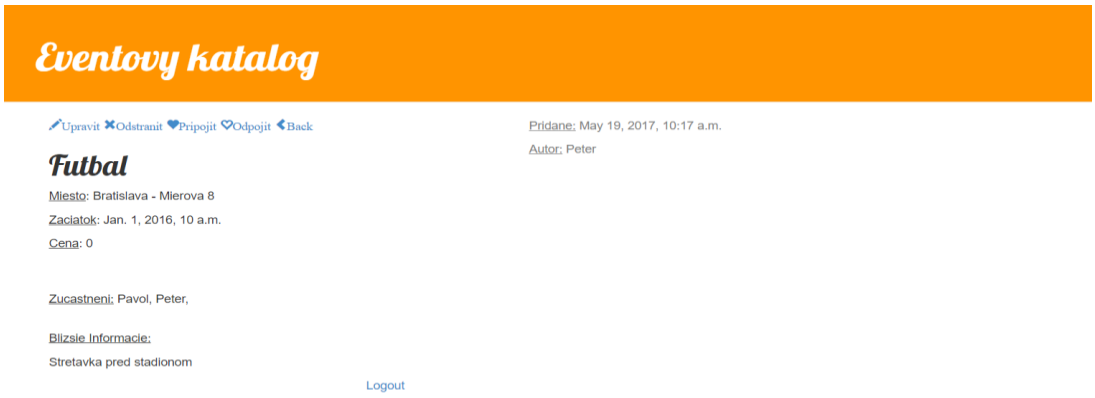
Zdroj: Vlastné spracovanie *3 Webová aplikácia - Registrácia užívateľov*

Zdroj: Vlastné spracovanie *4 Webová aplikácia – Prihlasovanie*

Zdroj: Vlastné spracovanie *5 Webová aplikácia - Vytvorenie podujatia*



Zdroj: Vlastné spracovanie *6 Webová aplikácia - Výpis podujatí*



Zdroj: Vlastné spracovanie *7 Webová aplikácia - Detail podujatí*

Užívateľ sa pre prístup do podujatí musí prihlásiť. Inakšie sú mu podujatia skryté a nie je mu ani umožnené ich pridávať.

3.2 Predstavenie použitej databázy

V obidvoch prípadoch bola použitá databáza SQLlite3, ktorá je štandardne zabudovaná v Rails a Django. V tejto kapitole si predstavíme ako vyzerá komunikácia s databázou v oboch programovacích jazykoch.

3.2.1 Komunikácia s databázou v Django:

Oproti Rails je robenie zmien v databáze jednoduchšie. Prvým krokom je zmena v tabuľke Models.py podľa ktorých sa neskoršie vytvoria zmeny v databáze. Ďalším krokom je zadanie nasledujúcich príkazov. Po zadaní príkazu *migrate* sa vytvorí taktiež informačný súbor s vykonanými zmenami.

```
(myvenv) C:\Python36-32\bc_python>python manage.py makemigrations
Migrations for 'events':
  0007_mask.py:
    - Create model Mask
```

Zdroj: Vlastné spracovanie *8 Django – makemigrations*

```
(myvenv) C:\Python36-32\bc_python>python manage.py migrate
Operations to perform:
  Synchronize unmigrated apps: messages, staticfiles
  Apply all migrations: events, contenttypes, admin, auth, sessions
Synchronizing apps without migrations:
  Creating tables...
    Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying events.0007_mask... OK
```

Zdroj: Vlastné spracovanie *9 Django – migrate*

3.2.2 Komunikácia s databázou v Rails:

V Rails je trošku zložitejšie robenie zmien medzi databázou a kódom. V prvom kroku je potrebné spustiť príkaz *generate migration,* kde sa vytvorí migračný súbor. Do neho sa vkladajú všetky zmeny, ktoré chceme v databáze uskutočniť. Vytvorí sa taktiež

informačný súbor s vykonanými zmenami. Nakoniec pre uplatnenie zmien je potrebné spustiť príkaz *migrate.*

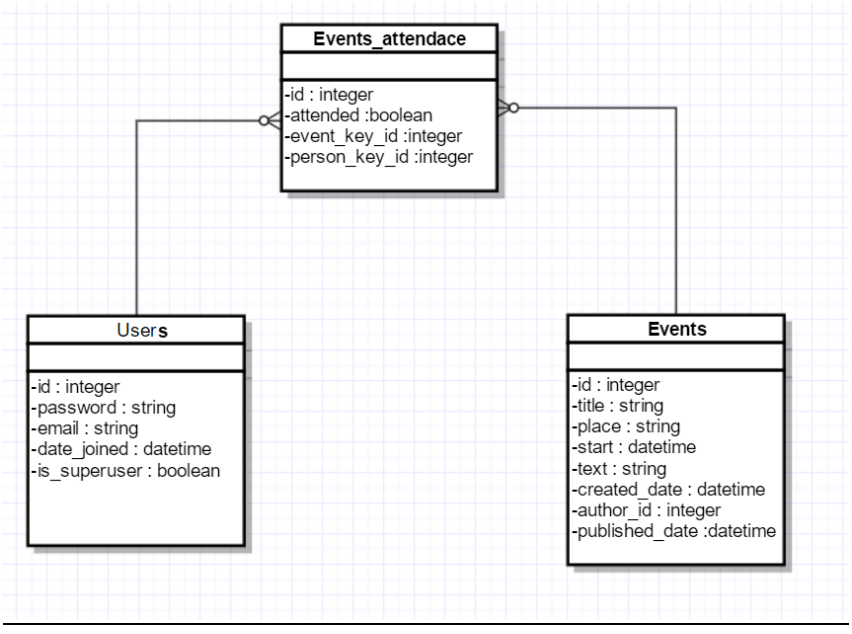
```
C:\RailsInstaller\Projects\events>rails generate migration new_migration
C:/RailsInstaller/Ruby2.3.0/lib/ruby/gems/2.3.0/gems/activesupport-4.0.0/lib/active_support/values/time_zone.rb:282: warning: circular ar
gument reference - now
Expected string default value for '--helper'; got true (boolean)
Expected string default value for '--jbuilder'; got true (boolean)
      invoke  active_record
      create    db/migrate/20170519115702_new_migration.rb
```

Zdroj: Vlastné spracovanie *10 Rails - generate migration*

```
C:\RailsInstaller\Projects\events>rake db:migrate
C:/RailsInstaller/Ruby2.3.0/lib/ruby/gems/2.3.0/gems/activesupport-4.0.0/lib/active_support/values/time_zone.rb:282: warning: circular ar
gument reference - now
==  NewMigration: migrating ===================================================
-- create_table(:new_table)
   -> 0.0596s
==  NewMigration: migrated (0.0606s) ==========================================
```

Zdroj: Vlastné spracovanie *11 Rails – Migrate*

3.2.3 Použitý model



Zdroj: Vlastné spracovanie *12 Diagram webovej aplikácie*

3.3 Znázornenie použitého kódu v oboch jazykoch

V tejto kapitole znázornime a porovnáme kód, ktorý bol použitý pre vytvorenie podujatia (v databáze vytvorenie inštancie v objekte Events). Komentáre k časti sú za znakom #.

3.3.1 Django (Python)

```
# Definicia objektu / tabulky Event v models.py
class Event(models.Model):
    author = models.ForeignKey('auth.User')
    title = models.CharField(max_length=200)
    place = models.CharField(max_length=100)
    start = models.DateTimeField()
    price = models.IntegerField()
    text = models.TextField()
    created_date = models.DateTimeField(
            default=timezone.now)
    published_date = models.DateTimeField(
            blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

    def __str__(self):
        return self.title

# Definicia vytvorenia formy pre objekt Event vo forms.py
class EventForm(forms.ModelForm):

    class Meta:
        model = Event
        fields = ('title', 'text','place','start','price',)
```

```
# Definicia cesty pre vytvorenie novej instancie v objekte Event v
urls.py
url(r'^post/new/$', views.event_new, name='event_new'),
# HTML subor pre vytvorenie instancie v objekte Event v post_edit.html
{% block content %}
    <h1>Pridaj event</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Create
Event</button>
    </form>
{% endblock %}

# Definicia funkcii pre vytvorenie instancie podujatia v subore view.py
def event_new(request):
    if request.method == "POST":
        form = EventForm(request.POST)
        if form.is_valid():
            event = form.save(commit=False)
            event.author = request.user
            event.published_date = timezone.now()
            event.save()
            return redirect('event_detail', pk=event.pk)
    else:
        form = EventForm()
    return render(request, 'events/post_edit.html', {'form': form})


# Volanie funkcie event_new
{% if request.user.is_authenticated %}
<a href="{% url 'event_new' %}">Vytvorit podujatie  </a>
{% endif%}
```

3.3.2 Rails (Ruby):

```
# Definicia funkcii pre pridavanie instancie do objektu Events v
events_controller.rb

  def new
    @event = Event.new

  end


  def create
    @event = Event.new(event_params)
    @event.username = current_user.username
    @event.save

    redirect_to event_path(@event)
  end


# Volanie funkcie new_event. Kontroluje sa, ci zakaznik je prihlaseny.
index.html.erb
<% if logged_in? %>
<h6><%= link_to "Vytvorit podujatie", new_event_path, class: "new_event"
%></h6>
<% end %>
```

```
# Definicia vytvorenia formy pre objekt Event vo new.html.erb

<h1>Pridaj event</h1>

<%= form_for(@event) do |f| %>
  <ul>
  <p>
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </p>
```

```
  <p>
    <%= f.label :body %><br />
    <%= f.text_area :body %>
  </p>
  <p>
    <%= f.label :place %><br />
    <%= f.text_field :place %><br />
  </p>
  <p>
    <%= f.label :start %><br />
    <%= f.datetime_local_field :start %><br />
  </p>
  <p>
    <%= f.label :price %><br />
    <%= f.number_field :price %><br />
  </p>

  <p>
    <%= f.submit %>
  </p>
<% end %>
<%= link_to 'Back', events_path %>

# Vykonanie vytvorenia eventu v new_event.html.erb
<%= render "all_look" %>
<h1>Podujatia</h1>
+
<ul id="event">
  <% @events.each do |event| %>
    <li>
      <%= link_to event.title, event_path(event) %>
    </li>
  <% end %>
</ul>

<%= link_to "Vytvorit nove podujatie", new_event_path, class:
"new_event" %>
```

3.4 Meranie programovacej produktivity a vyhodnotenie výsledkov

V tejto kapitole si znázorníme zaznamenávanie programovacej produktivity do tabuľky podľa definovaných kategórii.

3.4.1 Tabuľka programovacej produktivity webovej aplikácie

KATEGÓRIA	METRIKA	MERNÁ JEDNOTKA	ČASŤ	Django (Python)	Rails (Ruba)	VYSLEDOK
Inštalácia prostredia	Čas potrebný pre inštaláciu prostredia	Hodina	Príprava	2	1	**Django (Python)**
Inštalácia prostredia	Náročnosť inštalácie prostredia	Škála od 1 do 5	Príprava	2,5	2	**Rails (Ruby)**
Učenie sa programovacieho jazyka	Čas vynaložený vypracovaním tútoriálu	Hodina	Vypracovanie tutoriálu	30	25	**Rails (Ruby)**
Učenie sa programovacieho jazyka	Náročnosť pochopenia tútoriálu	Škála od 1 do 5	Vypracovanie tutoriálu	3,5	3	**Rails (Ruby)**
Učenie sa programovacieho jazyka	Náročnosť pochopenia jednotlivých príkazov	Škála od 1 do 5	Vypracovanie tutoriálu	2	3	**Django (Python)**
Písanie kódu	Čas vynaložený vypracovaním danej časti	Hodina	Vytváranie, editácia a vymazávanie udalosti	8	10	**Django (Python)**
Písanie kódu	Náročnosť vypracovania danej časti	Škála od 1 do 5	Vytváranie, editácia a vymazávanie udalosti	2	3	**Django (Python)**
Písanie kódu	Početnosť riadkov jednotlivých časti programu	Počet	Vytváranie, editácia a vymazávanie udalosti	172	212	**Django (Python)**
Písanie kódu	Čas vynaložený vypracovaním danej časti	Hodina	Správa uživateľov - registrácia, prihlásenie, odhlásenie	7	4	**Rails (Ruby)**
Písanie kódu	Náročnosť vypracovania danej časti	Škála od 1 do 5	Správa uživateľov - registrácia, prihlásenie, odhlásenie	3,5	2	**Rails (Ruby)**

KATEGÓRIA	METRIKA	MERNÁ JEDNOTKA	ČASŤ	Django (Python)	Rails (Ruba)	VYSLEDOK
Písanie kódu	Početnosť riadkov jednotlivých časti programu	Počet	Správa uživateľov - registrácia, prihlásenie, odhlásenie	130	208	**Django (Python)**
Písanie kódu	Čas vynaložený vypracovaním danej časti	Hodina	Správa návštevnosti podujatí uživateľov	10	12	**Django (Python)**
Písanie kódu	Náročnosť vypracovania danej časti	Škála od 1 do 5	Správa návštevnosti podujatí uživateľov	2	3,5	**Django (Python)**
Analýza programu	Početnosť riadkov jednotlivých časti programu	Počet	Správa návštevnosti podujatí uživateľov	50	42	**Rails (Ruby)**
Písanie kódu	Čas vynaložený vypracovaním danej časti	Hodina	Grafická úprava	5	5	-
Písanie kódu	Náročnosť vypracovania danej časti	Škála od 1 do 5	Grafická úprava	3	3	-
Analýza programu	Početnosť riadkov jednotlivých časti programu	Počet	Grafická úprava	65	65	-
Písanie kódu	Čas vynaložený vypracovaním danej časti	Hodina	Práca s databázou SQLlite3	5	6	**Django (Python)**
Písanie kódu	Náročnosť vypracovania danej časti	Škála od 1 do 5	Práca s databázou SQLlite3	2	3,5	**Django (Python)**
Celková analýza programu	Pridávanie dodatočných knižníc	Škála od 1 do 5	Analýza	3	1,5	**Rails (Ruby)**
Celková analýza programu	Celkový počet chýb pri písani kódu	Počet	Analýza	95	72	**Rails (Ruby)**
Celková analýza programu	Záujem na programovacom fóre www.stackoverflow.com	Počet otázok (približne)	Analýza	150000	270000	**Rails (Ruby)**
Prevádzka aplikácie	Dĺžka načítania stránky	Sekunda	Prevádzka	1	3	**Django (Python)**
Prevádzka aplikácie	Dĺžka zapnutia servera	Sekunda	Prevádzka	7	10	**Django (Python)**
Prevádzka aplikácie	Veľkosť projektovej zložky	Mb	Prevádzka	48	4,5	**Rails (Ruby)**

Zdroj: Vlastné spracovanie *1 Porovnávanie programovacej produktivity*

3.4.2 Vyhodnotenie meraných parametrov

V tejto kapitole si vyhodnotíme výsledky znázornené v tabuľke [Zdroj: Vlastné spracovanie 1 Porovnávanie programovacej produktivity a vyberieme programovací jazyk, ktorého pracovná produktivita je vyššia.

KATEGÓRIA	**VÝSLEDOK**
Inštalácia prostredia	-
Učenie sa programovacieho jazyka	Rails (Ruba)
Písanie kódu: Vytváranie, editácia a vymazávanie udalosti	Rails (Ruba)
Písanie kódu: Správa uživateľov - registrácia, prihlásenie, odhlásenie	Rails (Ruba)
Písanie kódu: Správa návštevnosti podujatí uživateľov	Django (Python)
Písanie kódu: Grafická časť	-
Písanie kódu: Práca s databázou SQLlite3	Django (Python)
Celková analýza programu	Rails (Ruba)
Prevádzka programu	Django (Python)
SPOLU	**Django (Python)**

Zdroj: Vlastné spracovanie *2 Vyhodnotenie nameraných hodnôt*

Ako vidíme z predchádzajúcej tabuľky, vyššiu produktivitu práce pre tento projekt naprogramovania webovej aplikácie dosiahol programovací jazyk Django, ktorý je napísaný v jazyku Ruby.

Záver

Cieľom našej bakalárskej práce bolo vytvoriť jednoduchý program v aspoň dvoch programovacích jazykoch (jedným z nich by mal byť jazyk Ruby) a porovnať produktivitu práce v oboch. Súčasne sme sa v prvej kapitole zamerali na aktuálnu situáciu v problematike produktivity práce v programovacích jazykoch. Definovali sme si rôzne druhy merania produktivity, a taktiež faktory ovplyvňujúce produktivitu v programovaní. V závere kapitoly sme si zadefinovali použité programovacie jazyky Python s webovým frameworkom Django a Ruby s webovým frameworkom Rails. V druhej kapitole sme si zadefinovali cieľ práce, ktorý bolo potrebné dosiahnuť. Následne sme si určili parametre, sledované kategórie v práci a spôsob sčítavania a vyhodnotenia výsledkov.

Poznatky, ktoré sme zistili z teoretickej časti sme následne aplikovali do poslednej časti práce. Na základe týchto poznatkov sme si zhodnotili výsledky meraní a zisteniami sme určili, že produktívnejší programovací jazyk v danom zadaní je Django, napísaný v programovacom jazyku Python. Rozdiel vo výsledku nebol až taký výrazný.

Za najväčší prínos našej práce považujeme práve skúsenosti nadobudnuté prostredníctvom programovania webovej aplikácie. Podľa nášho subjektívneho názoru sme osobne viac priklonení k používaniu webového frameworku Django. Práca s ním bola výrazne jednoduchšia na pochopenie a taktiež vykazovala zrozumiteľnejšie súvislosti.

Na základe nadobudnutých skúsenosti z našej bakalárskej práce sme si prehĺbili naše programátorské vedomosti z oblasti HTML, CSS, skriptovacích jazykoch Ruby a Python a taktiež z vytvárania webových prostredí.

Zoznam použitej literatúry

Knižné zdroje:

[1] EDWARD NELSON, V 1966. Management handbook for the estimation of computer programming costs. Systems Develioment Corporation, 1993

[2] CAPERS JONES, V 1966. Management handbook for the estimation of computer programming costs. Systems Develioment Corporation, 1993

[5] C. E. WALSTON AND C. P. FELIX, V 1977. A method of programming measurement and estimation [54-73]. IBM Systems Journal, 1977

[6] A. ABRAN AND P. N. ROBILLARD, V 1994. Function points: A study of their measurement processes and scale transformations [171-184]. Journal of Systems Software, vol. 25, 1994

[7] H. K. RAJU AND Y. T. KRISHNEGOWDA, V 2013. Software Sizing and Productivity with Function Points [1-5]. Lecture Notes on Software Engineering, Vol. 1, No. 2. May 2013

[8] WAGNER, S., & RUHE, M., V 2008. A Systematic Review of Productivity Factors in Software Development. State Key Laboratory of Computer Science in 2008

[9] GOPARAJU PURNA SUDHAKARA, AYESHA FAROOQBAND SANGHAMITRA PATNAIKC, V 2012. Measuring productivity of software development teams [65 – 75]. Serbian Journal of Management 7, 2012

[12] JEFF FORCIER, PAUL BISSEX, WESLEY CHUN, V 2009. Python Web Development with Django®. Addison-Wesley, USA, 2009

Internetové zdroje:

[3] ALBERT ENDRES A DIETER ROMBACH, V 2003. Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. [online] [190 - 192]. Pearson Education Limited, Harlow, England. 2003

[4] KEN KENNEDY, CHARLES KOELBEL,V 2014. ROBERT SCHREIBER, Defining and measuring the productivity of programming languages.[online]. Dostupné na internete: < https://www.cs.indiana.edu/~achauhan/Teaching/B629/2006-Fall/CourseMaterial/2004-ijhpca-kennedy-productivity.pdf >

[10] KEN KENNEDY, CHARLES KOELBEL,V 2014. ROBERT SCHREIBER, Defining and measuring theproductivity of programming languages [online]. Dostupné na internete: < https://www.cs.indiana.edu/~achauhan/Teaching/B629/2006-Fall/CourseMaterial/2004-ijhpca-kennedy-productivity.pdf >

[11] MASOUD NOSRATI, V 2011. Python: An appropriate language for real world programming. World Applied Programming, Vol (1), No (2) [online] [110-117], June 2011. Dostupne na internete: < http://waprogramming.com/download.php?download=50ae4a1125d607.12866725.pdf>

[13] MICHAEL FURR, JONG-HOON (DAVID) AN, JEFFREY S. FOSTER, MICHAEL HICKS, V 2009. The Ruby Intermediate Language [online]. University of Maryland, College Park. Dostupný odkaz: <https://www.cs.umd.edu/projects/PL/druby/papers/druby-dls09.pdf>

[14] RICK BORUP, V 2010. An Introduction to Ruby and Rails [online]. Dostupný odkaz: < http://www.ita-software.com/papers/borup_ruby_published.pdf>

[i] *Izomorfizmus* - vzťah medzi dvoma objektmi (systémami) poukazujúci na úplnú zhodu al. podobnosť ich štruktúry, funkčná zhoda