Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Jazyk PERL

Tomáš STREJČEK

Vedoucí práce: Doc. Ing. Vojtěch Merunka, Ph.D.

! ! !

Místo této strany vložíte zadání bakalářské práce.
(Do jedné vazby originál a do druhé kopii)

! ! !

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Jazyk PERL“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30. 3. 2010 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Vojtěchu Merunkovi, Ph. D., za vedení při vypracovávání této práce, Josefu Štěpánkovi za pomoc s korekturami a úpravy typografie a Zdeňku Hříchovi za neustálé popohánění.

Jazyk PERL

PERL Language

Souhrn

Tento text popisuje programovací jazyk Perl, stavbu jazyka, jeho nejdůležitější prvky i s příklady, jeho přednosti i nedostatky a uvádí a popisuje nejčastější příklady jeho využití. Dále analyzuje popularitu jazyka vyhodnocenou pomocí několika různých technik. Součástí textu je také srovnání Perlu se v současné době nejvíce používanými programovacími jazyky dle SLOC/FPA metrik. Práce zahrnuje také několik plně komentovaných zdrojových kódů, jež plně ukazují a vysvětlují použití tohoto programovacího jazyka.

Klíčová slova: perl, programování, vývoj, popularita jazyků, srovnání jazyků

Summary

This paper describes programming language Perl, structure of language, its most importatnt parts with examples of usage, its advatages and disadvantages and states and describes the most common examples of its usage. It also anlyse popularity of the language evaluated using several different methods. This thesis also compares Perl with the top languages in programming field using SLOC/FPA metrics. Text also includes a few fully commented source codes, presenting and explaining udage of this language.

Keywords: perl, programming, development, language popularity, language comparison

Obsah

Seznam obrázků a tabulek

Obrázky:

Tabulky:

1. Úvod

Perl je jeden z nejmodernějších programovacích jazyků. Kombinuje v sobě mnohé přístupy a technologie vyvinuté v posledních 20 letech, které uplynuly od jeho vzniku. V současné době je možné ho potkat takřka v jakémkoliv zařízení, operačním systému i programovém vybavení. Od doby svého vzniku si získal mnoho příznivců, zastánců, ale také odpůrců jak v řadách akademické obce, tak i v řadách vývojářů z komerční a open source sféry.

Perl je v dnešní době možno použít k řešení takřka jakékoliv problematiky. V nedávné době počet modulů (rozšíření) pro tento jazyk dosáhl 18 000, čímž se zařadil mezi funkčně nejbohatší a nejvyspělejší jazyky. Tento jazyk také urazil dlouhou cestu v otázce bezpečnosti a zabezpečovacích prvků, což potvrzuje řada bezpečnostních certifikátů od renomovaných společností. Je ironické, že se také stal jedním z nejčastějších nástrojů hackerů a kybernetických kriminálních živlů.

2. Cíl práce

Cílem této práce je obecné seznámení s programovacím jazykem Perl, možnostmi, které nabízí, a nejběžnějšími příklady použití spolu s ukázkami zdrojových kódů. Budou probrány základní vlastnosti tohoto jazyka, jeho syntaxe a nejzákladnější prvky a konstrukce, které tento jazyk používá.

Cílem je také porovnat Perl s ostatními oblíbenými jazyky a srovnat oblíbenost v dnešní době nejpoužívanějších jazyků. Jazyky také budou srovnány na bázi vědeckých matematických metod SLOC a FPA. Zároveň bude provedeno srovnání s nejběžněji používanými (mainstreamovými) jazyky – C, PHP, Java a Borland Pascal (Delphi) – i z pohledu programátora. K tomu budou použity ukázky z jednotlivých jazyků na jednoduché úloze. Součástí práce je také předvedení a ukázka nejběžnějších použití jazyka s plně komentovaným kódem.

3. Metodika

Při psaní této práce byly využity základní postupy odborné práce. Je to například:

- sběr podkladů,
- praktické znalosti,
- tvorba v programovacím jazyku,
- analýza,
- dedukce,
- syntéza,
- komparace.

Práce je založena na vlastních zkušenostech s programovacím jazykem Perl a na analýze literatury a zdrojů uvedených v seznamu literatury a zdrojů.

3.1. Struktura práce

První kapitola obsahuje úvod do této práce.
Druhá kapitola popisuje cíl práce.
Třetí kapitola se zabývá metodikou a strukturou tohoto dokumentu.
Čtvrtá kapitola se zabývá popsáním vlastností, syntaxe a prvků programovacího jazyka Perl.
Pátá kapitola srovnává jazyk Perl s ostatními známými jazyky.
Šestá kapitola popisuje odborné metodiky hodnocení softwaru a programovacích jazyků.
Sedmá kapitola vyhodnocuje jazyk Perl a některé ostatní jazyky pomocí metodik vysvětlených v kapitole 6.
Osmá kapitola provádí porovnání zdrojových kódů několika vybraných jazyků.
Devátá kapitola popisuje výhody a nevýhody jazyka perl a uvádí závěrečné zhodnocení.
Desátá kapitola obsahuje seznam použité literatury a zdrojů.
Příloha A obsahuje seznam operačních systémů, na kterých je možno provozovat základní distribuci Perlu.
Příloha B prezentuje jednoduchou tabulku operátorů jazyka Perl.

I. Literární rešerše

4. Jazyk Perl

4.1. Důvod vzniku Perlu

Jak se píše v dokumentaci Perlu [PERL], původním účelem vzniku Perlu byla manipulace s textem a textovými soubory, která v sobě sdružovala schopnosti jazyka AWK, unixové příkazové řádky (tzv. *shellu*), a programu pro editování (transformování) textu *sed*.

Jazyk byl vyvíjen s myšlenkou, že všechno se dá řešit více způsoby. Odtud plyne hlavní motto jazyka *"there's more than one way to do it"* (existuje víc jak jeden způsob jak to udělat), které je uvedené i na obálce knihy *Programming Perl* od autora jazyka Larryho Walla.

Tímto přístupem ovšem jazyk ve svých počátcích získal špatnou pověst, protože mnoho počítačových programátorů bylo schopno vytvořit v zásadě umělecká díla v podobě krátkých „zašmodrchaných“ skriptů různorodé délky řešící triviální, nebo naopak nesmírně komplikované úlohy. Často se velmi krátký skript používal na velice složitý problém.

V současné době se ovšem jazyk nejvíce používá na psaní systémových skriptů v operačních systémech *UNIX/Linux/BSD* a programování webových aplikací pro rozsáhlé webové portály.

4.2. Historie Perlu

V internetové publikaci *Beginner's Introduction to Perl* [Sheppard, 2000] se uvádí, že za vznikem Perlu stojí programátor a lingvista Larry Wall, kterému vadila nedokonalost jazyka AWK a vůbec nedostatek nástrojů pro práci s textem. Jako dalším důvodem se uvádí i potřeba řídit počítače na dálku, což jazyk rovněž umožňuje.

Z programu se nakonec vyvinul nový programovací jazyk, i když poměrně dlouho trvalo, než dostal své jméno – *Perl*. Uvádí se dva možné významy této zkratky – *Practical Extraction and Report Language* – praktický extrakční a reportovací jazyk, popřípadě *Pathologically Eclectic Rubbish Lister* – což by se velmi volně dalo přeložit jako „patologicky eklektický vypisovač blbostí“.

František Dařena ve své publikaci [Dařena, 2005] uvádí, že uvolnění interpretu jazyka pro veřejnost v roce 1987 se setkalo s nečekaným ohlasem. Tato odezva vedla Larryho a jeho spolupracovníky k dalšímu obohacování jazyka – z nástroje na zpracování textů se vyvinul „skutečný programovací jazyk“ s ladicími nástroji (debuggery), interpretery,

rozsáhlou sítí knihoven, dokumentace, podpůrných programů atd. V současnosti se vývoji a rozšiřování jazyka věnuje kolektiv lidí, kteří si říkají Perl Porters, a Larry Wall stojí v čele.

Jazyk byl ve svých počátcích zaměřen hlavně na platformu *Unix/Linux*, ovšem v současné době běží na více než 100 různých operačních systémech. Jejich úplný soupis je uveden v *příloze A*.

4.3. Úvod do jazyka Perl

Kniha *Programming Perl* [Wall, a další, 2000] uvádí, že Perl je všeobecný programovací jazyk původně vyvinutý jako nástroj pro manipulaci s textem (prohledávání, formátování, parsování a další zpracování), ale v současné době je používaný jako nástroj pro širokou škálu úloh. Patří mezi ně např. správa systémů (*UNIX/Linux*), ale je možné provádět správu serverů na platformě Windows, vývoj webových stránek a aplikací, programování síťových a vícevláknových aplikací, ale také vývoj umělé inteligence, popřípadě skriptování ve hrách a mnoho dalších.

Jak uvádí dokumentace [PERL], hlavním záměrem při vývoji tohoto jazyka bylo, aby byl *robustní a efektivní* (spíše než krásný a elegantní), což zároveň částečně zkomplikovalo snahu o snadnou použitelnost.

Perl podporuje několik různých programovacích *paradigmat* (tedy způsobů programování a vývoje):

a) **Procedurální (strukturální) programování.**
   Způsob vývoje aplikací, jenž krok za krokem popisuje, co přesně má program udělat za využití *řídích struktur*, *proměnných*, *procedur* (funkcí) a *bloků*.
   např.: *Visual Basic*, *Pascal*, *C*, ze starších *Algol*, *Cobol*

b) **Objektově orientované programování (OOP).**
   Toto je přístup, který modeluje aplikací pomocí datových struktur nazývaných objekty, obsahujících *vlastnosti* (atributy) a *metody* (jejich funkce).
   např.: *Java*, *C#*, částečně *C++*

c) **Funkcionální programování.**
   Programovací vzor sestávající se ze sledu propojených funkcí. Z největší části se využívá pro matematické aplikace, ale je možno se s ním setkat i v umělé inteligenci nebo průmyslové grafice (využití LISPu v AutoCADu apod.).
   např.: *LISP*, *Scheme*, *Haskell*

4.4. Vlastnosti Perlu

Larry Wall ve své knize [Wall, a další, 2000] uvádí, že při tvorbě Perlu bylo z mnoha různých jazyků i nástrojů vytaženo to nejlepší, co dané prostředky poskytovaly. V různých částech programu psaného v Perlu je proto možno zahlédnout mnoho různých prvků připomínající jiné jazyky. Jelikož je Perl původně linuxový nástroj, nejvíce si toho odnesl z linuxové implementace příkazového řádku a jeho standardních nástrojů pro programování a práci s textem.

Například asociativní pole, neboli tzv. hashe, převzal z programovacího jazyka AWK, který je pod Linuxem suplován nástrojem *awk*. Způsob definování seznamů (jiný druh pole) zase převzal z *Lispu*. Ovšem největší práce byla odvedena na jedné z nejsilnějších schopností Perlu – *regulárních výrazech*, které jsou převzaty z nástroje *sed* (stream editor).

Všechny tyto části byly implementovány za účelem usnadnění úloh pro práci s textem a na zpracování dat.

V páté verzi Perlu, kdy došlo ke kompletnímu přepsání celého jazyka tak, aby vyhovoval potřebám moderních programátorů i moderní doby, bylo přidáno mnoho nových schopností a vlastností:

a) Podpora komplexních datových struktur – různé druhy polí, seznamů, stromů a hashů.
b) Podpora OOP.
c) Modulární systém.

V rozhovoru z roku 1999 [Wall, 1999] Larry Wall tvrdí, že celý smysl *Perlu 5* a jeho modulovacího systému je podpořit růst komunity kolem Perlu a její kultury spíše než růst jádra programovacího jazyka.

Všechny verze jazyka Perl provádějí automatické zavádění datových typů na proměnné (takže není nutné se zabývat typováním proměnných) i správu paměti, kdy *interpret* (překladač) sám určuje, kdy je kterou část paměti možno uvolnit.

4.4.1. Design (návrh) jazyka

Larry Wall dále v rozhovoru [Richardson, 1999] říká, že základní myšlenka návrhu vychází z klesajících cen počítačového vybavení a z rostoucí ceny práce profesionálních programátorů. Starší programovací jazyky byly vytvářeny za účelem co nejefektivnějšího využití velmi drahého hardwaru. Perl se snaží co nejefektivněji využít placenou pracovní sílu.

Perl obsahuje mnoho vlastností, které mají za účel usnadnit vykonávání standardních úkonů programátorů, což bohužel může způsobovat větší nároky programu na paměť počítače i jeho výkon. Mezi tyto vlastnosti patří například automatická správa paměti, dynamické typování proměnných, řetězce, seznamy a asociativní pole (hashe), regulární výrazy a další.

Jak je uvedeno v knize *Programming Perl* [Wall, a další, 2000], velká část návrhu Perlu vychází z lingvistických principů:

a) Běžné konstrukce by měly být krátké.
b) Nejdůležitější informace/data by měla být na začátku.
c) Program by měl být dobře čitelný, i pokud komplikuje práci interpreteru.

Syntaxe Perlu odráží pravidlo, že různé věci by neměly vypadat stejně. Z tohoto důvodu mají různé druhy polí a proměnných různé počáteční znaky, různé druhy polí využívají jiné typy závorek. Takto se tedy jazyk naprosto liší například od *Lispu*, kde je využíván jeden typ závorek úplně pro všechno a zdrojové kódy se tak stávají velmi matoucími.

Návrh Perlu je uzpůsoben tak, že nevynucuje žádné programovací paradigma a je možno v průběhu psaní programu přecházet z jednoho do druhého a různě je kombinovat. Tím se bohužel částečně snižuje čitelnost, ale může se tím značně ulehčit práce.

Úvod knihy *Programming Perl* [Wall, a další, 2000] začíná větou *„Perl is a language for getting your job done,“* což v překladu zhruba znamená *„Perl je jazyk, s kterým zvládnete dokončit práci.“* Důsledkem této filozofie ovšem je, že zdrojové kódy Perlu nebývají příliš úhledné a uspořádané. Zahrnuje totiž velkou spoustu nejrůznějších vlastností, toleruje výjimky ze svých pravidel a vyžaduje heuristiku pro vyřešení syntaktických víceznačností.

Kvůli „promíjející“ povaze kompileru může být občas velmi těžké dohledat ve zdrojovém kódu chyby. Při řešení různého chování některých vestavěných funkcí v seznamovém nebo skalárním kontextu manuál pouze udává, že *„v zásadě to udělá to, co chcete, pokud ovšem nevyžadujete důslednost“*.

K Perlu se váže několik dalších sloganů [Sheppard, 2000], které poměrně přesně popisují design jazyka i to, jakým způsobem tvůrci uvažovali při jeho vytváření, případně jakým způsobem je možno tento jazyk využívat.

> *„Perl: the Swiss Army Chainsaw of Programming Languages.“*
> *„Perl: švýcarská motorová pila prog. jazyků.“* (v narážce na švýcarský nůž)

> *„No unnecessary limits."*

„Žádná nepodstatná omezení.“

Perl je také nazýván jako "lepící páska internetu" [Smith], v narážce na to, že mnoho základních systémových aplikací, na kterých běží nejdůležitější systémy a páteře v internetu, využívá nějakým způsobem i Perlu jako jazyka pro jejich vývoj a údržbu.

Do verze jazyka *Perl 5* neexistovaly žádné souhrnné specifikace nebo standardy a ani není znám žádný záměr něco takového vytvořit pro současné verze Perlu. Po celou dobu existence jazyka pro něj existuje pouze jediná implementace překladače (*interpreteru*) a jazyk se vyvíjel zároveň s jeho vylepšováním. Interpreter se tím pádem stává jakousi specifikací jazyka.

Vývoj jazyka *Perl 6* naproti tomu začal vytvořením specifikací a několika projekty, jež měly za cíl implementovat některé části specifikací.

4.4.2. Možnosti využití jazyka

Kniha *Programming Perl* [Wall, a další, 2000] popisuje všestranné možnosti využití jazyka Perl, jelikož Perl je možno uplatnit mnoha různými způsoby v mnoha různých oborech, což je z velké části zapříčiněno mnoha standardními moduly i moduly třetích stran. Všechny tyto moduly jsou umístěny na serverech organizace CPAN.

Zpočátku byl Perl využíván hlavně pro tvorbu webových stránek formou CGI skriptů a jako jeden z prvních skriptovacích jazyků také pro tvorbu webových aplikací (následují ho jazyky *Python* a *PHP*, které z Perlu značně vychází). V dnešní době se ovšem do popředí protlačují moderní jazyky, popřípadě moderní implementace starších jazyků, jako je Ruby a jeho webová implementace *Ruby on Rails*, popřípadě Server-Side *JavaScript*, čili JavaScript provozovaný na serveru a ne v klientském prohlížeči, a některé další. Webový Perl je z větší části využíván hlavně v rozsáhlých webových službách s mnoha tisíci návštěv za minutu.

Perl je také velmi často využíván jako jazyk spojující v sobě komponenty z jiných programovacích jazyků, popřípadě jako jazyk zprostředkovávající jejich komunikaci, a to hlavně v momentě, kdy k tomu tyto jazyky nebyly přímo navrženy.

Příkladem by mohla být například administrace webových serverů na bázi Linux + Apache + MySQL + PHP (tzv. *LAMP server*), kdy pro rozhraní administrace je využito PHP, které uvnitř volá připravené procedury napsané v Perlu, který teprve provádí zásahy do nastavení systému, konfigurace služeb, nebo přímo komunikuje se službami. Výsledkem tohoto řešení je vyšší efektivita použitého softwaru, nezávislost na komunikačních vláknech s uživatelským prohlížečem a samozřejmě vyšší bezpečnost administrace jako takové.

Další úlohou, na kterou je Perl využíván (hlavně v prostředí Unix/Linux), je generování různých rozsáhlých reportů, kombinujících v sobě data z mnoha různých zdrojů, popřípadě i jejich sběr. Tyto možnosti tvoří z Perlu jazyk všestranně využívaný systémovými správci. Tomu napomáhá i to, že kratší programy a skripty napsané v Perlu je možno spouštět přímo z příkazové řádky.

Programy v Perlu, pokud jsou pečlivě napsány, mohou být také bezproblémově přenositelné mezi Unixem a Windows.

Protože Perl umí používat mnoho grafických knihoven, není problém do aplikace naprogramovat uživatelské rozhraní, například pomocí modulu *Perl/Tk*.

4.4.3. Implementace jazyka

Perl je implementován jako *nízkoúrovňový interpret*, toto popisuje například dokumentace Perl API [PERL]. Samotný interpret a převážná část modulů jsou napsány v jazyku C. Interpret se skládá ze 150 tisíc řádků kódu, což se zkompiluje do 1 MB velkého spouštěcího souboru (pouze na běžných architekturách), případně může být přeložen do dynamické knihovny a vkládán do ostatních jazyků.

V základní distribuci jazyka se nachází téměř 500 standardních modulů, zahrnujících zhruba 200 tisíc řádku kódu jazyku Perl a dalších 350 tisíc řádků jazyku C.

Interpret má *objektově orientovanou architekturu,* což se dá nejlépe potvrdit na způsobu uložení složitých proměnných (ať už polí, hashů a dalších) – jsou uloženy jako struktury jazyka C. Operace nad těmito strukturami jsou založeny na bázi C maker, typedefů a funkcí, které jsou podstatou *Perl C API* (aplikačního rozhraní Perlu) sloužícího pro komunikaci a práci s moduly a dalšími rozšířeními. Podoba API může být pro nezasvěcené zarážející, ovšem jeho vstupní body jsou tvořeny podle konzistentního schématu, což poskytuje dobrý návod pro ty, kdo ho pravidelně využívají.

4.4.4. Dostupnost jazyka

Dle zveřejněné licence [Wall] je Perl volný software licencovaný pod Artistic License a GNU General Public License. Jsou dostupné distribuce pro většinu operačních systémů, avšak převládá hlavně na Unixu a na něm založených systémech (Linux, BSD, ...). Byl však portován pro většinu moderních (a mnoho zastaralých) operačních systémů. Perl může být zkompilován přímo ze zdrojového kódu na všech na Unixu založených POSIX vyhovujících operačních systémech [PERL], nebo jinak s Unixem kompatibilních

systémech. Zřídka je to však nezbytné, jelikož Perl je součástí základní instalace většiny populárních operačních systémů na bázi Unixu.

Pod Microsoft Windows se typicky využívá některé zkompilované distribuce Perlu, nejčastěji *Strawberry Perl* (open source distribuce), nebo *ActivePerl* od společnosti ActiveState, což je částečně komerční distribuce, ovšem plusem je možnost placené podpory. Pod Windows je sice možné zkompilovat Perl ze zdrojového kódu, ovšem je k tomu potřeba množství dodatečných nástrojů, což značně komplikuje kompilaci a využití modulů z CPAN (hlavně těch napsaných v C).

Programátoři využívající ActivePerl jsou tedy odkázáni na využití překompilovaných modulů od společnosti ActiveState. Bohužel rozsáhlost repozitáře a množství vytvářených modulů (plus nedostatek prostředků pro udržení aktuálnosti repozitáře) dlouhodobě způsobuje velké problémy při vývoji v této distribuci jazyka.

Strawberry Perl, open source distribuce pro Windows, má pravidelně vydávané aktualizace (nyní zpravidla čtvrtletně), spolu s předěláváním modulů podle požadavků vývojářů. Prvotním impulzem pro vznik této distribuce byla snaha o vyřešení dlouhodobých problémů distribuce ActivePerl, možnost instalovat moduly na různé platformy a také o dlouhodobé vyřešení situace a problémů na platformě Windows.

Na Windows je možné spustit Perl i pomocí emulátoru *Cygwin*, který ve Windows vytváří Unixu podobné prostředí, a umožňuje tak provozovat Unixové distribuce Perlu i jejich moduly.

4.4.5. CPAN

CPAN, neboli *Comprehensive Perl Archive Network* [CPAN], je název archivu, ve kterém jsou uloženy veškeré moduly dostupné pro Perl. Zároveň je to také název modulu, který pracuje jako interface pro přístup do archivu a umožňuje stahování a instalaci jednotlivých modulů. O CPAN se také opírá celá komunita Perlu.

4.5. Jazyk Perl

Celá následující kapitola je parafrázována z vyčerpávajícího popisu Františka Dařeny [Dařena, 2005] s občasným doplněním z knihy Pavla Satrapy [Satrapa, 2001].

Ve většině učebnic programování bývá jako první program „Nazdar světe“. Následujícími způsoby je možno vytvořit skript, který na monitor počítače vytiskne tuto frázi.

```
say "Nazdar světe.";
```

Jako alternativu je možno použít starší:

```
print "Nazdar světe.\n";
```

což je naprosto ekvivalentní zápis se stejným výstupem.

4.5.1. Prvky jazyka

4.5.1.1. Výrazy

Velkou část programu tvoří většinou výrazy. Výraz se skládá z *operandů* a *operátorů*, a po vyhodnocení vždy udává nějakou hodnotu (nebo hodnotu `undefined`, tj. nedefinovaná hodnota). Podle toho, jaké operátory se ve výrazu nacházejí, jsou s operandy prováděny operace, které ovlivňují konečný výsledek výrazu. Je-li za výrazem uveden středník, stává se z něj *příkaz*.

```
$x = 5; vs. if(x==5){ .. }
```

Tedy ne každý výraz je příkaz, avšak každý příkaz je výraz – pokaždé má nějakou hodnotu bez ohledu na to, zda s touto hodnotou dále pracuji či ne. Program v Perlu je tedy posloupností příkazů a výrazů, které jsou odděleny středníkem.

4.5.1.2. Příkazy

Každý příkaz v sobě může obsahovat modifikátory, které například uvádějí, zda se vůbec provede, případně kolikrát apod.

```
$x = 1;
print "$x = ", $x;
print "$x = ", $x if $x != 0;
print "$x = ", $x while ++$x >= 3;
```

Pod příkazy spadají i tzv. řídící struktury, ty ovšem v textu nebudou podrobněji popsány, jelikož jsou vesměs ve všech programovacích jazycích stejné. Patří mezi ně větvení (příkazy *if* a *switch*) a příkazy cyklů (příkazy *for* a *while*) a mnohé další.

4.5.1.3. Bloky

Všechny příkazy je možno spojit do jednoho *bloku*. Blok je určen počáteční a konečnou složenou závorkou. Blok se může sestávat i z jednoho příkazu. Bloky se nejčastěji užívají v příkazech větvení nebo u příkazů cyklů. Bloku je možno přidělit i název, tzv. *návěští* – to se píše velkými písmeny a ukončuje se dvojtečkou, za níž následuje samotný blok.

```
if($x)
{
   #blok;
}

PRG:
{
   #blok s návěštím;
}
```

Jelikož se použití složených závorek vyskytuje i u jiných prvků než u bloků, je nutné vždy dávat pozor na kontext, v jakém jsou závorky použity.

4.5.1.4. Operátory

Operátory jsou součástí výrazů a pracují s operandy. Zápis operátorů je možno provádět třemi způsoby:

a) prefixový tvar – operátor se nachází před operandem:
```
$a = ++$b;
```
b) postfixový tvar – operátor se uvádí až za operandem

```
$a = $b++;
```

c) infixový tvar – operátor se uvádí mezi operandy

```
$a = $b + $c;
```

Každý operátor má přesně určený počet operandů (tzv. *arita*) a operandy se zpracovávají v pořadí udávaném prioritou jednotlivých operátorů.

Operátorů je v Perlu poměrně velké množství. Jsou uvedeny v *příloze B*.

4.5.1.5. Bílé znaky (whitespaces)

Bílé znaky jsou znaky ze začátku ASCII tabulky – tedy například mezera, tabulátor, zalomení řádku a další.

Přestože nejsou přímou součástí programu (kromě řetězců a řetězcových literálů), jsou bílé znaky nedílnou součástí zdrojového kódu. Zvyšují jeho přehlednost a jejich chybné použití může mít vliv na správný běh programu.

4.5.1.6. Komentáře

Nedílnou součástí všech programovacích jazyků jsou *komentáře*, které umožňují programátorovi do zdrojového kódu zapsat poznámky k daným částem kódu, vysvětlení důvodu zvoleného řešení a mnohé další užitečné (i neužitečné) věci. V poslední době se také pomocí speciálním způsobem zapsaných komentářů generuje kompletní dokumentace software. Jako příklady mohou být uvedeny JavaDoc, PHPDoc a další.

Existuje několik druhů komentářů:

a) *Jednořádkový* – začíná znakem `#` a končí koncem daného řádku.
b) *Víceřádkový* – začíná znakem `=` následovaným alespoň jedním znakem a končícím na jiném řádku `= cut` (tento typ zápisu se nejčastěji používá v dokumentaci samotného Perlu a také všech jeho standardních modulů).

4.5.1.7. Podprogramy

Tyto části kódu, v jiných jazycích či situacích nazývané funkce, procedury, metody, i jinak, jsou podstatou všech programů. V Perlu jsou všechny typy podprogramů nazývány jako funkce. Podprogramy mohou nebo nemusí vracet nějakou hodnotu.

4.5.1.8. Identifikátory

Identifikátory jsou jména přidělená jednotlivým prvkům programu, aby bylo možné je od sebe odlišit. Protože je jazyk Perl *case-sensitive*, záleží vždy na velikosti písmen ve jménech identifikátorů (ale i v řetězcích nebo řetězcových literálech).

Identifikátory začínající písmenem nebo podtržítkem mohou mít libovolnou délku, ovšem je doporučeno tvořit maximálně 255 znaků dlouhé názvy. Dále mohou obsahovat písmena, číslice a podtržítka. Identifikátory, které začínají číslem, mohou obsahovat pouze čísla. Identifikátory začínající jiným znakem jsou zpravidla považovány za speciální a omezují se na tento jediný znak a jsou také považovány za globálně platné. Název identifikátoru může obsahovat i symbol `::`, kterým se oddělují jména balíků a jména balíků od jmen proměnných či podprogramů.

4.5.1.9. Literálové symboly

Toto je speciální druh identifikátorů, začínající a končící dvěma podtržítky. Používají se pro speciální účely, např. `__LINE__` pro zjištění čísla řádku právě běžícího skriptu, `__FILE__` pro zjištění jména otevřeného souboru a další.

4.5.1.10. Proměnné

Proměnné jsou nositeli různých vlastností a hodnot a tvoří podstatu programu. Pomocí názvu proměnné vyvoláváme hodnoty z paměti počítače. Proměnné se dělí na *skalární* (obsahují jednoduchou hodnotu) a *seznamové* (obsahují složené, vícerozměrné hodnoty).

Skalární proměnné obsahují číslo nebo řetězec, nebo mohou obsahovat pointer (odkaz na adresu v paměti). Proměnná je však vždy chápana podle toho, v jakém kontextu je použita.

4.5.1.11. Ovladače

Tyto prvky umožňují pracovat se vstupními a výstupními zdroji dat. Perl obsahuje několik různých ovladačů, např. STDIN pro čtení ze standardního vstupu příkazové řádky, popřípadě STDOUT pro výstup do konzole.

4.5.1.12. Formáty a šablony

Pomocí formátů se definuje formátování výstupu programu. Používají se hlavně v případě, kdy je nutné formátovat velmi rozsáhlý textový výstup (například v případě generování rozsáhlých reportů).

4.5.1.13. Moduly a balíky

Modul je soustavou datových struktur, které souvisejí s určitým problémem. Takový modul můžeme nazvat knihovnou. Modul je kód v separátním souboru, který lze použít ve více programech. Tímto způsobem lze zdrojový kód programu rozdělit na několik logických částí.

4.5.1.14. Regulární výrazy

Jedním z nejsilnějších nástrojů Perlu jsou regulární výrazy. Ty slouží k analýze a práci s textovými daty. Mezi nejběžnější činnosti s regulárními výrazy patří porovnávání textu oproti výrazu. Například jestli je daný text adresa URL, e-mailová adresa, nebo jestli se text skládá pouze z povolených znaků a podobně.

4.5.2. Datové typy

Na rozdíl od některých programovacích jazyků má Perl relativně malý počet datových typů. Neumožňuje také vytvářet nové pojmenované abstraktní datové typy jako například v Pascalu nebo v C.

Tab. 1 Symboly u datových typů a ostatních klíčových struktur jazyka PERL

Znak	Příklad	Popis
$	$a	Skalární proměnná
@	@a	Pole
%	%a	Hash
	A	Ovladače a manipulátory
&	&a	Procedury a funkce
*	*a	Typegloby

V této kapitole jsou popsány nejzákladnější datové typy.

4.5.2.1. Skalární proměnné

Skalární proměnné jsou obyčejné proměnné známé z ostatních programovacích jazyků, ačkoli proměnné v perlu jsou dynamicky typované, čili se předem neurčuje, zda bude v proměnné číslo nebo řetězec nebo něco jiného.

```
$a = 5;
```

4.5.2.2. Pole a seznamy

Pole slouží pro uložení množiny hodnot, ať už přímo souvisejících nebo ne. K hodnotám se přistupuje pomocí takzvaných indexů.

```
@pole = (1,2,3);
```

```
@pole = qw(1 2 3);
```

4.5.2.3. Asociativní pole – hash

Asociativní pole je nadále v textu nazýváno anglickým termínem *hash*. Je to speciální typ pole, v němž jako index slouží řetězec znaků. Je tedy velmi snadné vyhledávat v poli potřebné hodnoty.

```
%hash = ('ovoce'=>'banán','zelenina'=>'mrkev');
```

```
%hash = ('ovoce','banán','zelenina','mrkev');
```

4.5.3. Kontext

Kontext je v Perlu jednou z nejpodstatnějších věcí. Stejně jako běžném jazyce mají některá slova různý význam podle toho, v jaké souvislosti je použijete, tak i v Perlu se proměnné chovají podle toho, jak se s nimi zachází a kde se používají. Pokud si programátor špatně uvědomí nebo zvolí kontext, může to v programu způsobovat těžko odhalitelné chyby.

Dvěma základními kontexty jsou *skalární* a *seznamový* kontext.

4.5.3.1. Skalární kontext

```
$a = 5;
```

Na levé straně je uvedena jednoduchá proměnná (začíná symbolem `$`), vyhodnocuje se tedy ve skalárním kontextu.

```
$a = @pole;
```

Toto se opět vyhodnocuje ve skalárním kontextu. Jelikož však na levé straně není jednoduchá proměnná, ale nachází se tam pole, je do hodnoty `$a` přiřazen počet prvků pole, protože je pole vyhodnoceno ve skalárním kontextu.

4.5.3.2. Seznamový kontext

Změna nastává, pokud by se na pravé straně nacházel seznam. Ten místo počtu prvků seznamu vrátí poslední prvek v seznamu.

```
@pole = ('jedna','dve','tri');
```

V poli je uložen seznam se třemi hodnotami.

```
$a = ('jedna','dve','tri');
```

V proměnné `$a` je uložena hodnota `'tri'`, protože byl seznam vyhodnocen skalárně.

```
$b = @pole;
```

Opět vyhodnocení `@pole` skalárně, v proměnné `$b` bude uložena velikost pole, tedy 3.

4.5.3.3. Speciální kontexty

Pavel Satrapa ve své knize [Satrapa, 2001] uvádí další speciální typy kontextů, jako je *logický kontext*, který vzniká tam, kde se vyhodnocuje podmínka, a jeho cílem je udat hodnotu *pravda* nebo *nepravda*. Celý výraz je nejprve vyhodnocen ve skalárním kontextu a výsledná hodnota se poté posoudí podle pravidel logických operátorů.

Častým řešením podmínky pro ověřování neprázdného pole se používá následující řešení:

```
if(@pole)
{
  ..
```

```
}
```

`@pole` se nejprve vyhodnotí ve skalárním kontextu, což buď vrátí 0 (čili *nepravda*), nebo číslo větší než nula (čili *pravda*).

Autor [Satrapa, 2001] popisuje i další typ kontextu – tzv. *prázdný kontext*. Nastává tehdy, pokud se použije výraz tam, kde není vyžadována žádná hodnota.

Posledním speciálním typem kontextu je *vkládací kontext*. Nastává uvnitř dvojitých uvozovek a v několika dalších případech. Při jeho vyhodnocování dochází k překladu speciálních posloupností znaků na speciální symboly.

```
print "ahoj\n";
```

Napíše ahoj a pomocí `\n`, které se přeloží jako zalomení řádku, řádek zalomí.

5. Postavení jazyka Perl

5.1. Popularita programovacích jazyků celosvětově

Popularita jednotlivých programovacích jazyků se číselně vyjadřuje velmi těžko. Často se také liší jak v rámci kontinentů, tak i v rámci jednotlivých (i sousedních) zemích. Je nutné proto přistoupit ke zkombinování statistik z různých zdrojů, aby odhady byly co nejpřesnější.

Dle serveru *LangPop.com* [LangPop] se nejčastěji používají následující aspekty pro hodnocení popularity:

a) Na prvních místech se nachází statistiky vyhledávačů, kde se většinou ověřuje počet dotazů v měsíci, nebo počet stránek obsahující určité sousloví. V 90 % případů se praktikuje vyhledávání fráze „název_jazyka *programming*“, případně pouze „název_jazyka“, což ovšem může být dosti scestné, jelikož hodně programovacích jazyků pouze přebrala jméno nějakého reálného objektu, tudíž se výsledky mohou mísit a statistiky jsou pak velmi nepřesné.
b) Jako další důležité měřítko slouží statistiky z online obchodů a knihkupectví, které jsou schopny poskytnout poměrně slušný obraz ohledně komerčního uplatnění jednotlivých jazyků.
c) Nejméně přesnou metodou je analyzování nejpoužívanějších programátorských diskuzních fór a sledování klíčových slov v hlavičkách jednotlivých vláken.
d) Co se týče komerční uplatnitelnosti jednotlivých jazyků, naprosto nejpřesnější informace je možné získat z pracovních portálů (samozřejmě při uplatnění patřičných filtrů na zúžení výběru pouze na nabídky míst pro programátory daného jazyka).

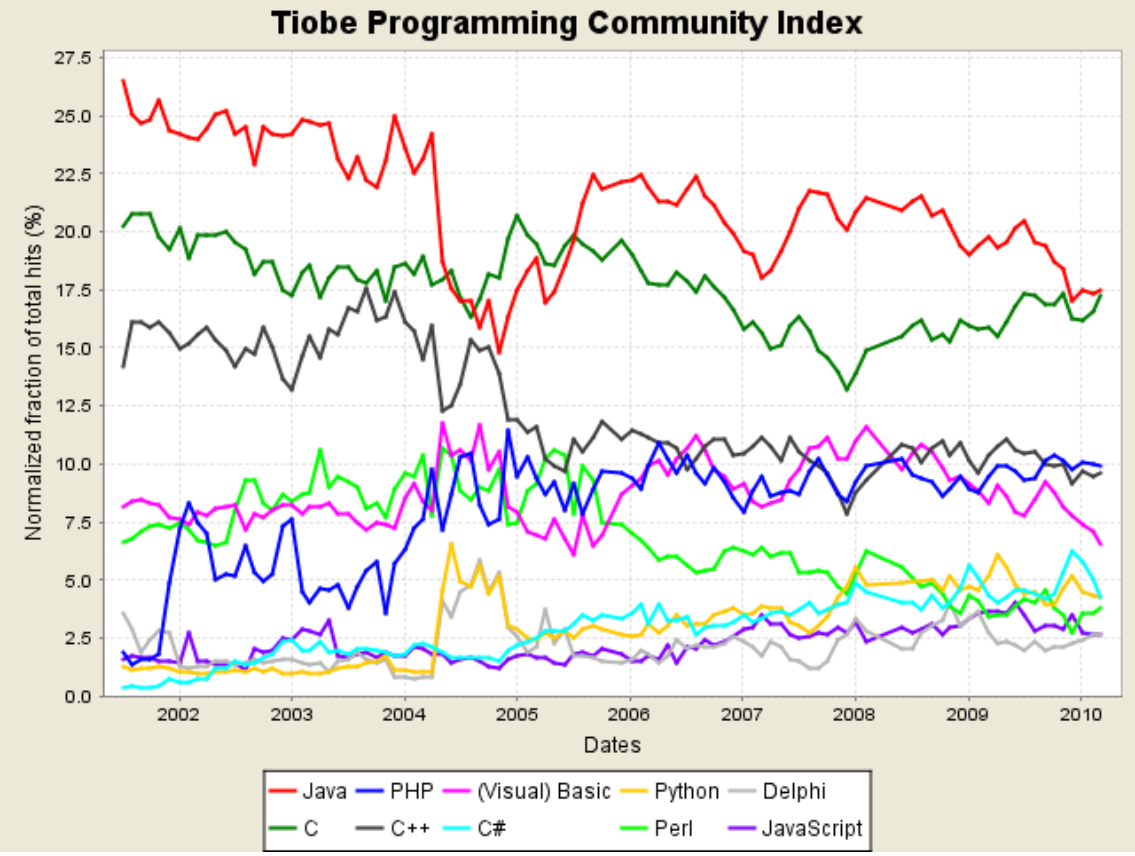
5.1.1. Dlouhodobý trend vývoje popularity TOP10 programovacích jazyků podle TIOBE

TIOBE Programming Community index [TIOBE] indikuje popularitu jednotlivých programovacích jazyků. Index je aktualizován jednou měsíčně a jeho historie sahá až do roku 2003.

Rating jednotlivých jazyků je založen na počtu schopných programátor daného jazyka celosvětově, dostupnosti odborných kurzů a dodavatelů třetích stran. Dále jsou do výpočtů

zahrnuty statistiky vyhledávačů *Google*, MSN (*Bing*) a *Yahoo*, případně statistiky a záznamy z webové encyklopedie *Wikipedia* a mediálního serveru *YouTube*.

Plná definice výpočtu může být nalezena na adrese:
http://www.tiobe.com/content/paperinfo/tpci/tpci_definition.htm



Obr. 1 Vývoj zastoupení jednotlivých jazyků podle [TIOBE]

5.1.2. Grafy z vyhledávače Google a jeho nástroje Google Trends – Perl celosvětově

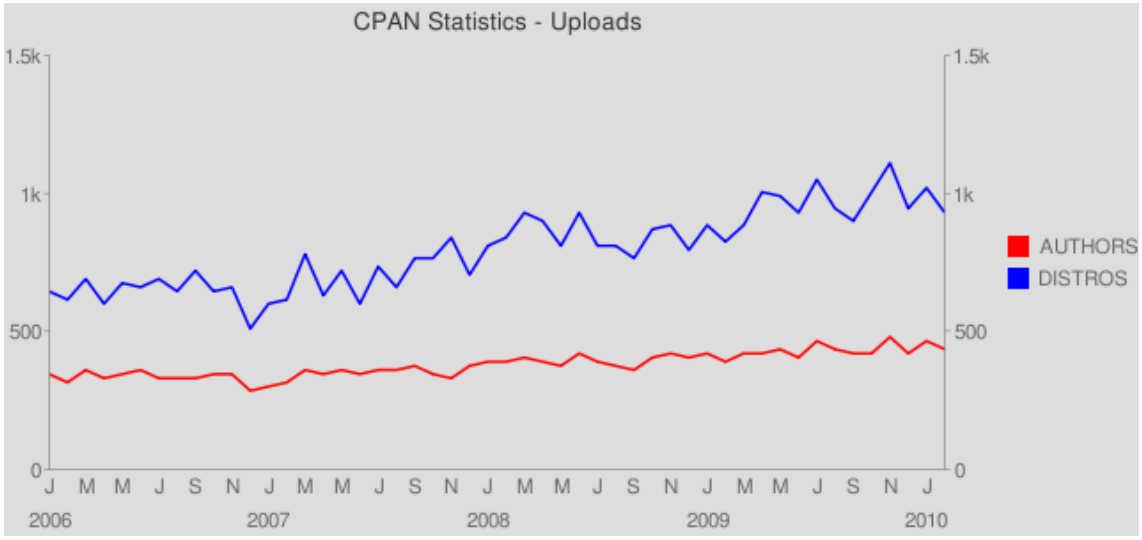
Obr. 2 Google Trends – Perl celosvětově podle [Google]

Z grafu velmi výrazně vyplývá klesající hledanost klíčového slova Perl.

5.1.3. Statistiky CPAN

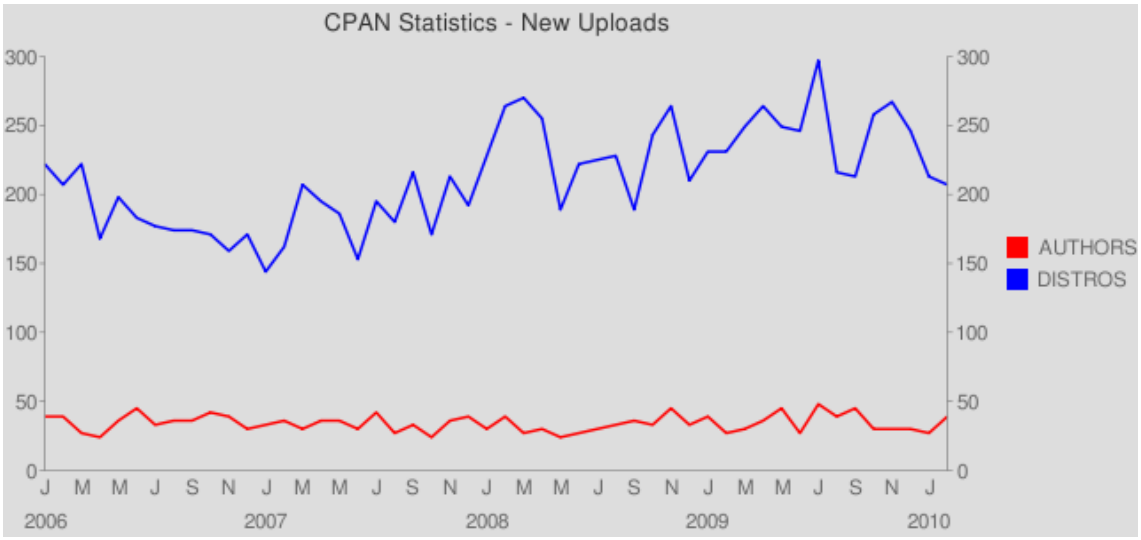
Statistiky CPAN přehledně ukazují aktivitu celé komunity okolo jazyka Perl.

5.1.3.1. Upload všech projektů



Obr. 3 CPAN Statistiky – Upload všech projektů podle [CPAN]

5.1.3.2. Upload nových projektů



Obr. 4 CPAN Statistiky – Upload nových projektů podle [CPAN]

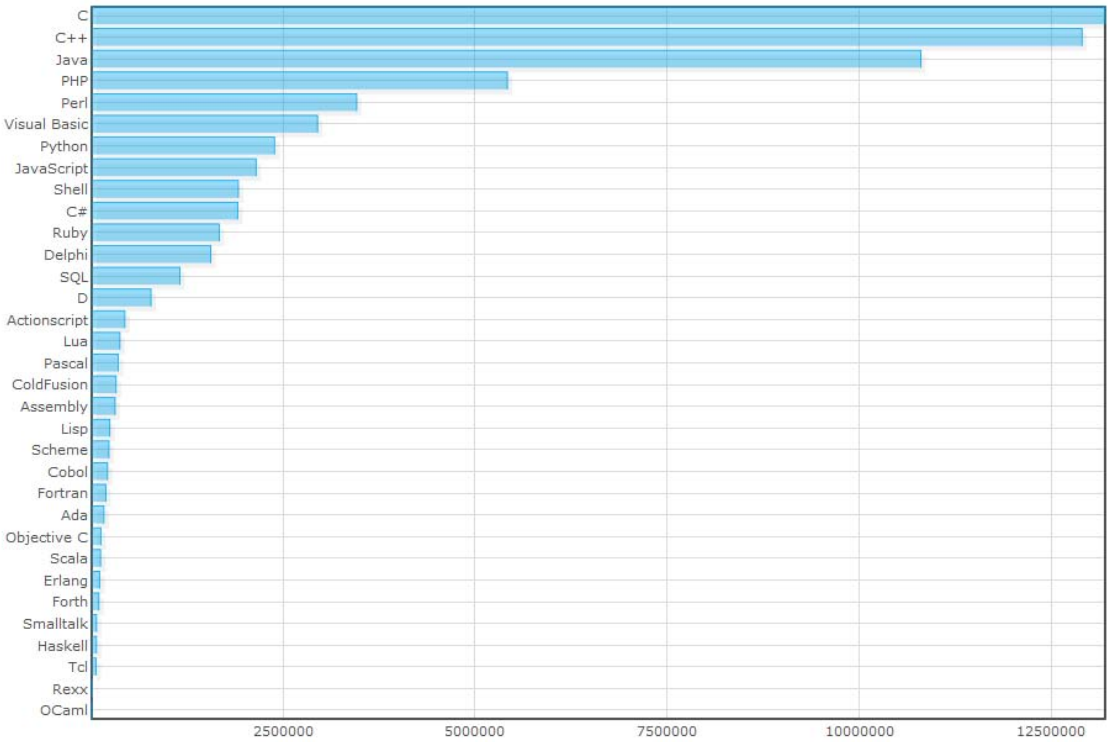
Jak je z grafů vidět, aktivita komunity neustále vzrůstá, stejně jako počet nových modulů i množství aktualizací modulů stávajících.

5.1.4. LangPop.com Pogramming language popularity list

5.1.4.1. Výsledky vyhledávání v Yahoo

Toto je velmi hrubý odhad popularity podle počtu existujících stránek o jazyku. Perl se zde umístil na pátém místě, v čemž určitě hraje roli velmi rozsáhlá komunita uživatelů.

Perl zde zaujímá 5. místo v popularitě ve vyhledávání na serveru Yahoo.

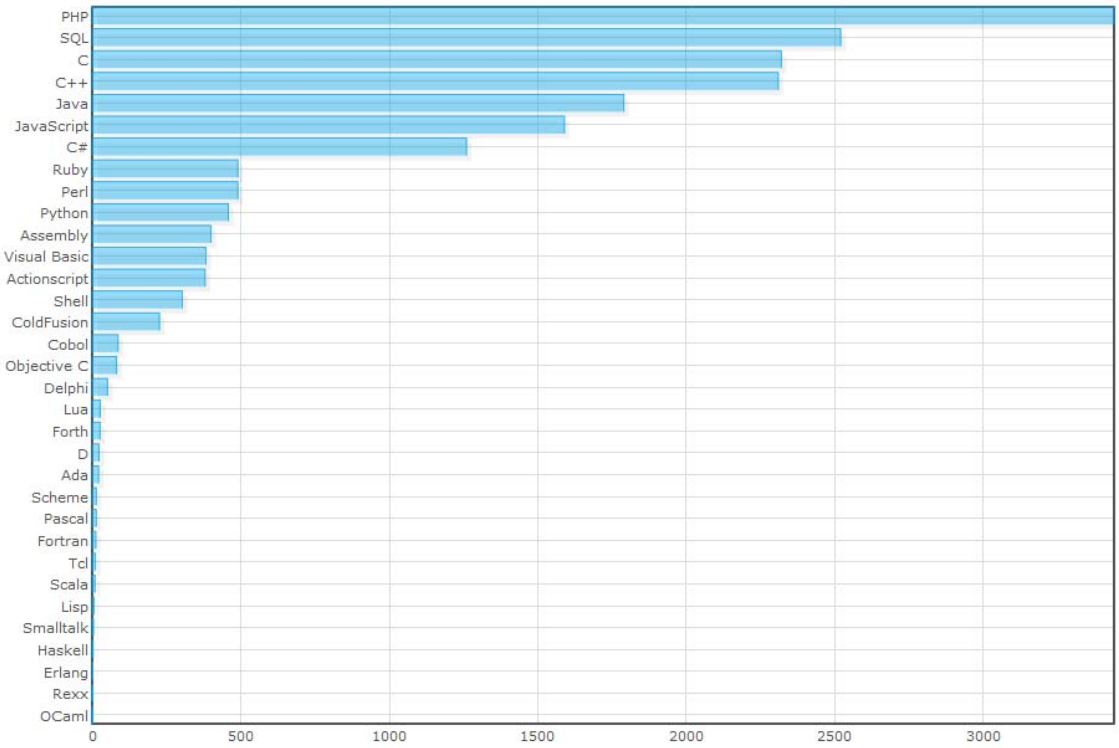


Obr. 5 Výsledky hledání v Yahoo podle [LangPop]

5.1.4.2. Craigslist

Craigslist je americký pracovní portál a vyhledávanost programátorů daného jazyka velmi dobře odráží popularitu jazyka, protože velká část programátorů se nejvíce zaměřuje na lukrativní jazyky.

Perl zde zaujímá 9. místo v popularitě v komerčním prostředí (posuzováno podle nabídky pracovních míst).



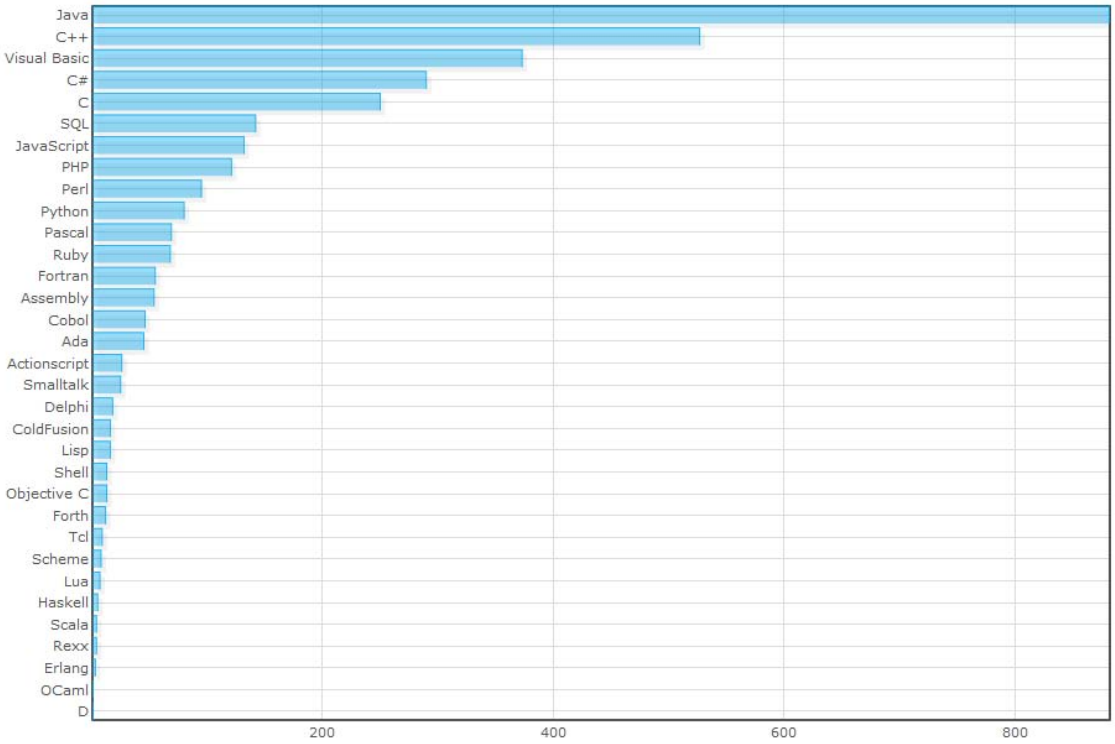
Obr. 6 Popularita v komerčním sektoru podle [LangPop]

5.1.4.3. Powell's Books

V měření se většinou používají data z Amazonu, největšího světového online knihkupectví, ovšem v době sbírání dat nebyla data dostupná. Server LangPop proto využil data z běžného knihkupectví se standardní nabídkou knih. Autoři měření sami tvrdí, že bude delší dobu trvat, než data vyladí, a proto tento graf nemá vysokou vypovídací hodnotu.

Autoři dále uvádějí, že knihy velmi snadno vyloučí nezavedené jazyky, kterých je několik set, ovšem pouze vážné jazyky jsou zastoupeny více než jednou vydanou knihou.

Perl zde zaujímá 9. místo v popularitě v počtu vydaných knih s touto tématikou.

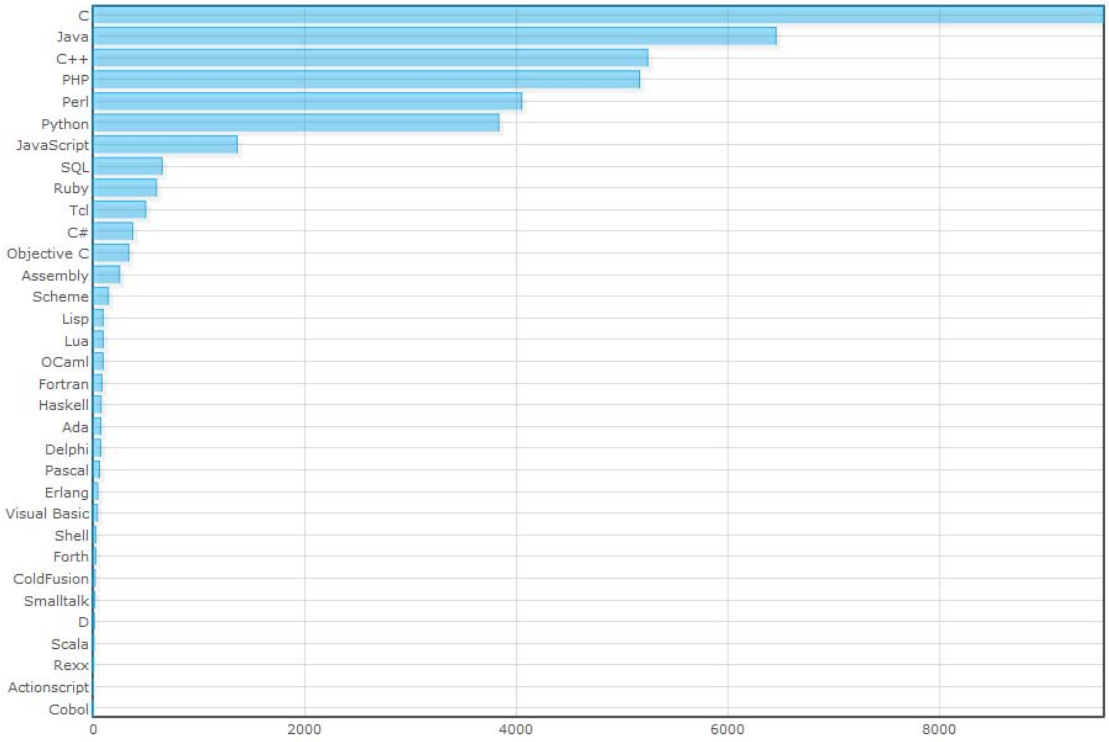


Obr. 7 Popularity v knihkupectví podle [LangPop]

5.1.4.4. Freshmeat

Freshmeat je server, na kterém se publikují open source projekty, které přešly do nějaké seriózní fáze. Statistiky z tohoto serveru velmi dobře poskytují přehled o tom, jaké jazyky se nejvíc uplatní v neplacené sféře.

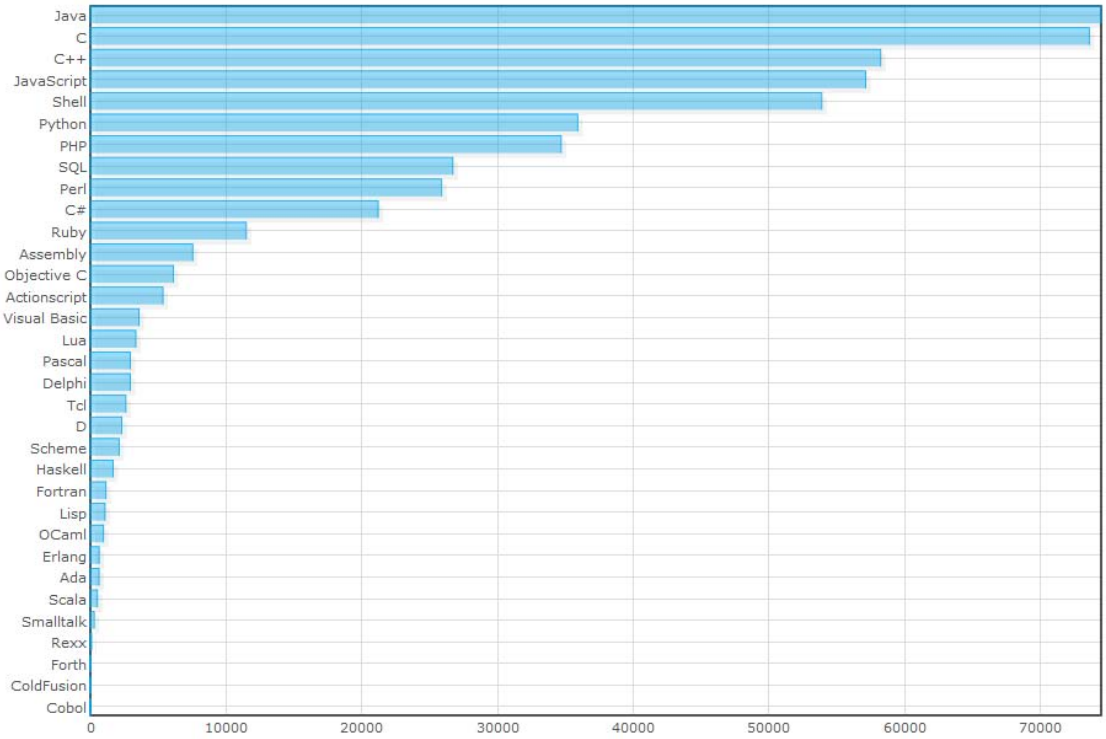
Perl zde zaujímá 5. místo v popularitě v open source komunitě serveru Freshmeat.



Obr. 8 Popularity v Open-Source podle [LangPop] – server Freshmeat

5.1.4.5. Ohloh

Čísla jsou založena na počtu lidí přispívajících do jednotlivých projektů v daném jazyku. Perl zde zaujímá 9. místo v popularitě open source trackeru Ohloh.

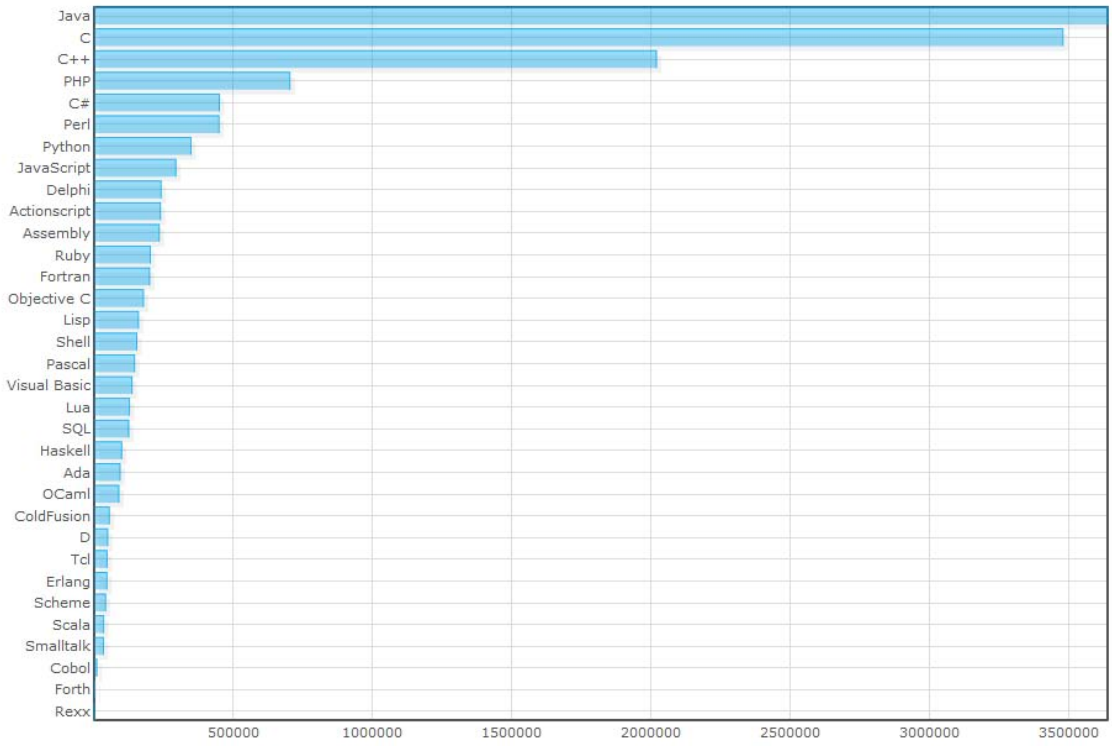


Obr. 9 Popularita v Open-Source podle [LangPop] – server Ohloh

5.1.4.6. Google Code

Google Code je všeobecné úložiště pro zdrojové kódy a programy ať komunitní nebo soukromé, díky jednoduššímu založení účtu i jednodušším pravidlům pro sdílení zdrojových kódů.

Perl zde zaujímá 6. místo v popularitě a množství projektů napsaných v jazyce Perl.

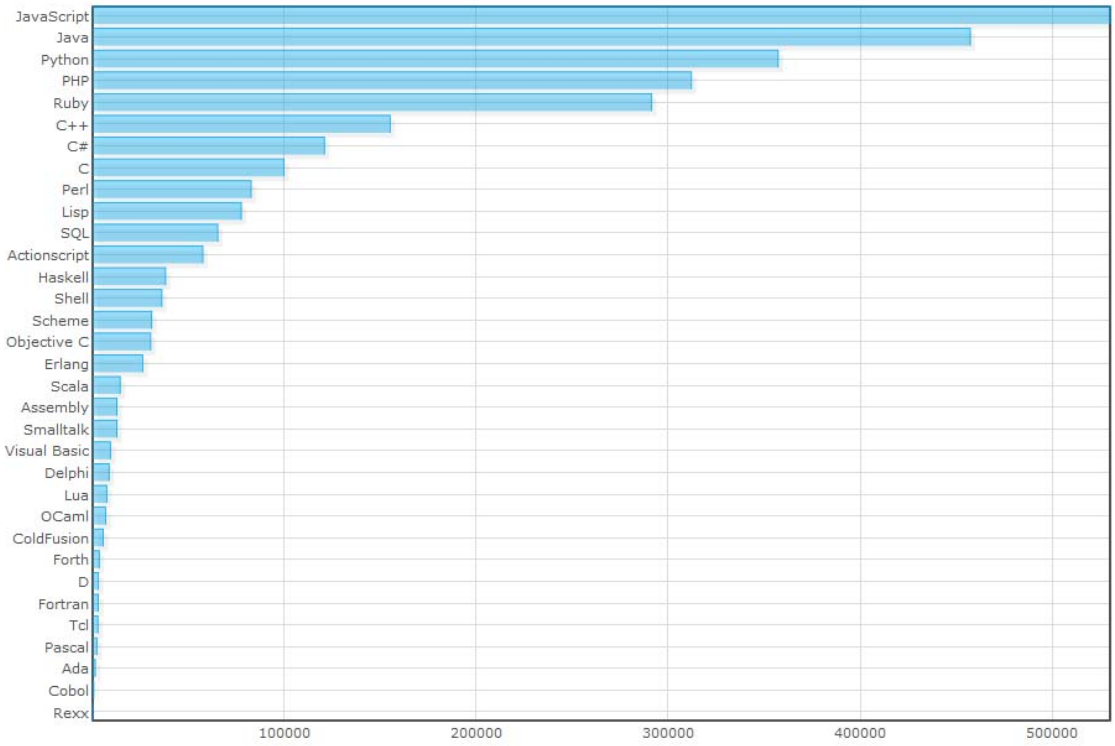


Obr. 10 Popularity v Google Code podle [LangPop]

5.1.4.7. Del.icio.us

Del.icio.us je americká služba pro sdílení zajímavých odkazů. Veškerá data jsou založena na tom, co sami lidé považují za zajímavé (přidají si odkaz k oblíbeným, případně mu přidělí klíčová slova a důležitost). Také to samozřejmě nejvíce ovlivňují módní vlny.

Perl zde zaujímá 9. místo v popularitě a množství odkazování ve službě Del.icio.us.

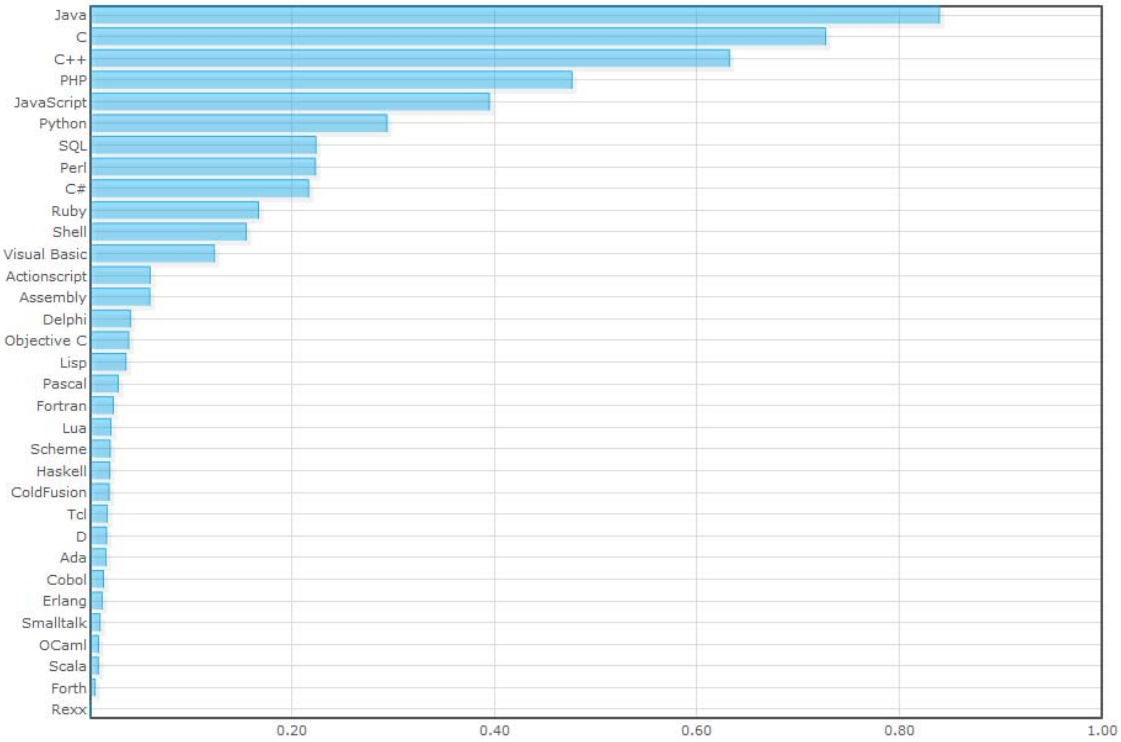


Obr. 11 Popularita v odkazové službě podle [LangPop]

5.1.4.8. Normalizované srovnání

Graf zobrazující kombinaci všech výsledků měření, pokud všechna měření mají stejnou váhu a důležitost.

Perl zde zaujímá 8. místo po zkombinování měření popularit měřených serverem LangPop, pokud je všem měřením přiřazena stejná důležitost.



Obr. 12 Celková popularita podle [LangPop]

5.2. Perl v České republice

5.2.1. Zastoupení jazyků v serverech hostujících CZ domény

Porovnání zastoupení jednotlivých jazyků ve webových serverech v rámci České republiky, které provádí Jakub Vrána ze serveru *php.vrana.cz*, známý český publicista, lektor a vývojář a také jeden z autorů české dokumentace jazyka PHP. Jsou sledovány pouze *.cz* domény, ovšem spousta serverů tají použité technologie – je velmi pravděpodobné, že zvlášť u utajených linuxových serverů bude na velké části provozuschopný Perl.

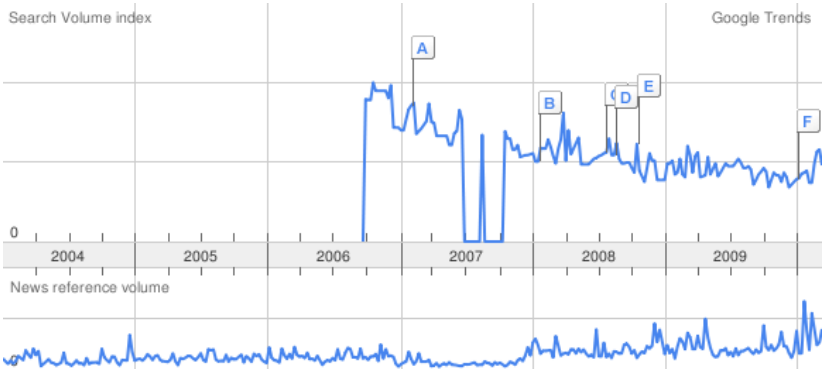
Tab. 2 Zastoupení Perlu na .cz doménách v únoru 2010 podle [Vrána, 2010]

	IP		domén		změna
PHP	15581	48 %	240917	38 %	−3 %
.NET	5153	16 %	90390	14 %	+1 %
Perl	1291	4 %	14227	2 %	−1 %
Python	1029	3 %	12208	2 %	0 %
Ruby	148	0 %	4627	1 %	0 %
Java	263	1 %	1314	0 %	0 %

Kromě PHP a .NET mají ostatní programovací jazyky používané na webech zanedbatelný podíl.

V rámci republiky je dlouhodobě nejrozšířenějším jazykem PHP.

5.2.2. Google Trends – Perl v české republice



Obr. 13 Vyhledávání Perlu v české republice podle [Google]

Přes značné kolísání grafu je znát pokles vyhledávání slova Perl v češtině.

6. Metodologie porovnání programovacích jazyků

6.1. Metoda SLOC

Metoda SLOC je značně nepřesná, zvláště u jazyků jako je Perl, kde je minimalismus na prvním místě a příkazy se (ačkoli v přehledné formě) často píší za sebe na jeden řádek. Proto v dnešním světě nemá porovnávání dle počtu řádků zdrojového kódu velký smysl.

6.2. Metoda analýzy funkčních bodů

IFPUG na svém webu [IFPUG] uvádí, že metoda analýzy funkčních bodů je založena na součtu všech funkcí obsažených v softwaru tak, aby byl srozumitelný pro uživatele. Její měřítko se přímo vztahuje k obchodním požadavkům, pro které má být daný software určený. FPA může být proto snadno aplikována napříč širokým spektrem vývojových prostředí, a to po celou dobu vývoje projektu, od prvotního určení požadavků až po plné nasazení v provozu. Současně je z ní možné pro podpoření softwaru odvodit další obchodní měřítka, jako například produktivitu vývojového procesu a jednotkovou cenu.

Samotná metoda *analýzy funkčních bodů* je určována v několika etapách. Když použijeme standardizovanou sadu základních kritérií, každou podnikovou funkci lze podle jejího typu a složitosti vyjádřit jako číselný index. Tyto indexy jsou vytvořeny, za účelem získání prvotního měřítka rozsahu, které je následně normalizováno začleněním řady faktorů vztahujících se k softwaru jako celku. Konečným výsledkem je bodový index funkce, který měří rozsah a složitost softwarového produktu.

Ve zkratce, metoda analýzy funkčních bodů přináší objektivní porovnatelné měřítko, které pomáhá při ohodnocování, plánování, řízeni a kontrole vývoje software.

Při hodnocení programovacích jazyků se používá měřítko nutného počtu výrazů v daném jazyce pro dosažení 1 funkčního bodu.

II. Vlastní projekt

7. Porovnání s mainstream. jazyky

7.1. Objektivní způsoby porovnání prog. jazyků

Porovnání je možné provést několika různými matematickými metodami, ať už metodou SLOC (*Source lines of code*), tedy metodou, založenou na počítání řádků kódu, nebo pomocí takzvané analýzy funkčních bodů (*Function point analysis* – FPA).

7.1.1. Srovnání jazyků využitím metod SLOC a FPA

Tab. 3 Funkční body programovacích jazyků podle [QSM, 2009]

číslo	Jazyk	QSM SLOC/FP Data	
		Průměr	Medián
1	ABAP (SAP)	18	18
2	Powerbuilder	28	22
3	Smalltalk	28	19
4	VB.Net	28	-
5	SQL	31	30
6	VBScript	38	37
7	HTML	43	42
8	PL/SQL	47	39
9	SAS	50	35
10	Visual Basic	50	52
11	JavaScript	54	55
12	Java	55	53
13	ASP	56	50
14	J2EE	57	50
15	Perl	57	57
16	C#	58	59
17	PL/1	58	57
18	C++	59	53
19	ColdFusion	68	56
20	COBOL	80	78
21	FORTRAN	90	118
22	C	148	107
23	Ada	154	-
24	Assembler	209	203

7.2. Porovnání zdrojových kódů jazyků

7.2.1. Základní program

7.2.1.1. Popis programu

Program otevře soubor, načte ho do pole a jednotlivé řádky rozparsuje podle oddělovače (středník). Poté data v poli sloučí středníkem a vypíše je do konzole po řádcích.

Úloha by měla být realizována co nejjednodušeji za použití jednoduchých základních operací s řetězci. Ve vyšších programovacích jazycích by samozřejmě nebyl problém využít vestavěné třídy, ale to by ovšem již bylo nad rámec jednoduchého srovnání zdrojových kódů.

7.2.1.2. Program napsaný v Perlu

Kód je napsaný čitelně, přestože by pomocí *inline* operátorů bylo možné ho zkrátit na mnohem menší počet řádků. Takto je velmi dobře čitelný i pro neprogramátory.

```
#!/usr/bin/perl
use strict;

$soubor = 'soubor.csv';
open(SOUBOR, $soubor) or die "Nemohu otevřít $soubor\n";

@data = ();

while($radek = <SOUBOR>)
{
  chomp($radek);
  @radek = split /;/, $radek;
  push(@data, \@radek);
}

foreach(@data)
{
  print join(';',$_),"\n";
}

close SOUBOR;
```

7.2.2. Porovnání s C

Operace s řetězci jsou v jazyce C dost náročné, následující kód tedy neplní zadaný úkol, je realizována pouze první část – načtení jednotlivých datových polí do samostatných proměnných. Jejich opětovné sloučení by vyžadovalo minimálně další stránku až dvě zdrojového kódu.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 25
#define MAX_SIZE 32

int main(int argc, char *argv[])
{
    char tmp[1024] = {0x0};

    char data[MAX][MAX_SIZE] = {0x0};

    char delimiter = ';';

    char nazev_souboru[] = "soubor.csv";

    int pocet_zaznamu = 0;
    int pocet_poli = 0;

    FILE *soubor = fopen(nazev_souboru,"r");

    if(in==NULL)
    {
        printf("Nemuzu otevrit soubor\n");
        exit(EXIT_FAILURE);
    }

    int cislo_radku = 0;

    while(fgets(tmp,sizeof(tmp),soubor)!=0)
    {
        pocet_zaznamu ++;

        char *radek = strtok(tmp,delimiter);

        while(*radek)
        {
            strcpy(data[cislo_radku],radek);
            cislo_radku++;
```

```
            radek = strtok('\0',delimiter);
        }
    }

    int i = 0;

    for(i=0;i<cislo_radku ;i++)
    {
        printf();
    }

    fclose(soubor);

    return 0;
}
```

7.2.3. Porovnání s Javou

Java, jakožto jazyk silně využívaný v komerčním prostředí díky své výborné přenositelnosti, si dokáže velmi schopně poradit i s takovouto jednoduchou úlohou.

```
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.StringTokenizer;

public class csv

{

public static void main(String args[]) throws IOException
{

    String nazev_souboru = "test1.csv";

    String str_radek;

    String delimiter = ";";

    ArrayList data = new ArrayList();

    FileInputStream soubor = new FileInputStream(nazev_souboru);

    DataInputStream vstup = new DataInputStream(soubor);
```

```
    while ((str_radek = vstup.readLine()) != null)
    {
        ArrayList radek = new ArrayList();
        StringTokenizer st = new StringTokenizer(radek,";");
        while(st.hasMoreElements())
        {
            String pole = st.nextToken();
            radek.add(pole);
        }
        data.add(radek);
    }


    for(String s : data)
    {
        StringBuilder sb = new StringBuilder();
        for(String x : s)
            sb.append(x + delimiter);
        sb.delete(sb.length()-delimiter.length(), sb.length());

        System.out.println(sb.toString());
    }
}
}
```

7.2.4. Porovnání s PHP

V zásadě nejkratší možný zápis v PHP; zároveň ukazuje jednoduchost a sílu tohoto skriptovacího jazyka.

```
<?php
$soubor = 'soubor.csv';
$delimiter = ';';
$data = file($soubor);
foreach($data as $radek)
    $radek = explode($delimiter,$radek);
foreach($data as $d)
    echo implode($delimiter,$d);
?>
```

7.2.5. Porovnání s Delphi (Boland Pascal)

```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
TForm1 = class(TForm)
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var soubor: TextFile;
    text: String;
    fileName: String;
    strings: TStringList;
    i: Smallint;
    join: String;
begin
fileName := ExtractFilePath(Application.ExeName) + 'test.txt';
if FileExists(fileName) then begin
  AssignFile(soubor, fileName);
  Reset(soubor);
  while not Eof(soubor) do begin
    ReadLn(soubor, text);
    strings := TStringList.Create;
    Assert(Assigned(strings));
    strings.Clear;
    strings.Add(text);
    strings.Delimiter := ';';
    strings.DelimitedText := text;
```

```
    join := '';
    for i := 0 to strings.Count - 1 do begin
      join := join + ';' + strings[i];
    end;
    ShowMessage(copy(join, 2, Length(join)));
    strings.Free;
  end;
  CloseFile(soubor);
end else begin
  ShowMessage('Soubor nenalezen!');
end;
end;

end.
```

7.2.6. Shrnutí

V takovém to porovnání musel jazyk C jednoznačně prohrát, jelikož práce s řetězci v jazyce C je poměrně značně komplikovaná, určitou možností by bylo využití tříd pro práci s řetězci, ale to by program pouze zjednodušilo, než výrazně zkrátilo.

PHP je jednoznačně vítěz v jednoduchosti programu se zachováním dobré čitelnosti, ovšem nebyl by problém program napsat více ve stylu Perlu a pak by vypadal víceméně velmi podobně, jelikož PHP značně vychází přímo z Perlu.

Přestože program v Javě je velmi robustní a nutná kompilace do *bytecode* značně zpomaluje čas od napsání do prvního spuštění programu, ukazuje velmi dobře robustnost jazyka i jeho připravenost na všemožné úlohy.

8. Aplikace

8.1. Skript na scanování portů vzdáleného počítače

Toto je velmi jednoduchá ukázka, proč je jazyk také oblíbený mezi bezpečnostními odborníky – pomocí pár příkazů je možno provést audit všech počítačů v síti.

```
#!/usr/bin/perl

#program na scanování portů

use IO::Socket;
#použije balík pro práci se sockety

my ( $peer, $port );
#definice lokálních proměnných peer a port

( $peer = $ARGV[0] ) || &usage;
#pokud skript nemá zadaný argument, vypíše př. použití (procedura usage) a
ukončí se

my %porty = split /:/,
"ftp:21:ssh:22:telnet:23:smtp:25:name:42:nameserver:42:finger:79:http:80:www:80
:www-http:80:hosts2-ns:81:imaps:993:pop3s:995:";
#do hashe porty se uloží data vzniklá po rozdělení řetězce podle dvojteček

%porty = reverse %porty;
#převrátí hash a vymění klíče za hodnoty

my @porty2 = sort by_number keys %porty;
#třídící procedurou sort seřídí hash s porty a uloží do lokální proměnné porty2

&list if $ARGV[0] eq "list";
#pokud je první argument skriptu 'list', program pomocí procedury list vypíše
všechny kombinace portů v hashi

foreach $port (@porty2) {
    $sock = IO::Socket::INET->new("$peer:$port");
    #skript se pokusí otevřít daný socket na vzdáleném počítači

    print "Port $port $porty{$port} je otevřen\n" if ($sock);
    #pokud je spojení úspěšné, vypíše zprávu
}
print "Hotovo\n";
exit;
```

```
sub usage {
  print "
  PortScan
  Usage: portscan <ip address>
         portscan list\n\n";
  exit;
  #ukončí program
}

#procedura pro výpis portů
sub list {
    foreach $port (@ports) {
        print "Port $port $connected{$port}\n";
    }
    exit;
    #ukončí program
}

#procedura pro řazení hashe - od nejmenšího po největší
sub by_number {
    if ( $a < $b ) { -1 }
    elsif ( $a > $b ) { 1 }
    else { 0 }
}
```

8.2. Okomentovaný skript Logrotate

Logrotate je jedním z nejpoužívanějších skriptů v systému UNIX/Linux sloužící k administraci. Jeho úlohou je pravidelně pomocí CRONu vzít všechny logovací soubory ze systémové složky, ve které se ukládají logy, přečíslovat je a zkomprimovat, tím je vždycky aktuální log poměrně malé velikosti, což urychluje čtení i zapisování do logu.

Tento skript předvádí celou škálu operací v Perlu. Každý řádek je doplněn komentářem, co konkrétně skript v daném místě provádí.

```
#! /usr/bin/perl -w
#skript bude v případě chyby i menších nedostatků ve skriptu vyhazovat
chyby (parametr -w)
use strict;
#použije se přísná kontrola skriptu

# $Id: logrotate.perl,v 1.6 2002/01/28 14:43:53 khera Exp $

use constant USAGE => "usage:logrotate [-z] [-p PID] [-s SIG] -r N
file...\n";
#vytvoření konstanty USAGE, vypisující jednoduchý návod na použití
skriptu
```

```
use IO::File;
#použije se balík pro vstupní a výstupní oberace se soubory

use Getopt::Std;
#použije se balík s funkcemi pro získávání parametrů příkazové řádky
spuštěného skriptu

my %opt = ();
#vytvoří se hash opt

getopts('zp:s:r:',\%opt);
#podle zadaného schématu se do hashe rozparsují jednotlivé parametry
skriptu

scalar(@ARGV) or die USAGE;
#užití pole ARGV ve skalárním kontextu vrátí velikost pole, pokud 0,
skript se ukončí a vypíše konstantu USAGE

die USAGE unless $opt{'r'};
#pokud není nastaven parametr skriptu r, skript se ukončí

sub rotate ($$) {
  my($file,$howmany) = @_;
  my($cur);

  return if ($howmany < 0);

  unlink ("$file.$howmany","$file.$howmany.gz");
  #odstraní první s daným jménem

  for ($cur = $howmany; $cur > 0; $cur--) {
    my $prev = $cur - 1;
    rename("$file.$prev","$file.$cur") if (-f "$file.$prev");
    #jestliže existuje předchozí verze, přejmenuj soubor
    rename("$file.$prev.gz","$file.$cur.gz") if (-f "$file.$prev.gz");
    #jestliže existuje předchozí verze v komprimovaném stavu, přejmenuj
    soubor
  }

  rename("$file","$file.0");
  #přejmenuje původní soubor

  my $fh = new IO::File $file, O_WRONLY|O_CREAT, 0644;
  $fh->close();
  #vytvoří nový soubor logu
}

#nejdříve se prjedou všechny soubory
foreach my $file (@ARGV) {
```

```
  rotate ($file,$opt{'r'});
}

if ($opt{'p'}) {
  my $sig = $opt{'s'} || 'HUP';
  unless (kill $sig, $opt{'p'}) {
    warn "Failed to send $sig to $opt{'p'}\n";
    warn " -- disabling compression of file\n" if (delete $opt{'z'});
  }
}

#nakonec se provede zkomprimování souoru, pokud potřeba
if ($opt{'z'}) {
  foreach my $file (@ARGV) {
    system "gzip $file.0";
  }
}

exit(0);

__END__
```

9. Závěr

9.1. Výhody

- Perl je schopný běžet na všech platformách.
- Perl má všechny vlastnosti potřebné pro vývoj velkých projektů
  - Je modulární.
  - Obsahuje objektově orientované programování.
- Perl obsahuje jednu z nejschopnějších implementací regulárních výrazů ze všech jazyků.
- Perl má perfektní management paměti.
- Množství efektivně implementovaných datových struktur.
- Velmi silná komunita CPAN neustále produkující nové moduly.

9.2. Nevýhody

- Není snadné vytvořit spustitelný soubor (exe).
- Perl není běžně dostupný na všech operačních systémech, pouze na nejnovějších Linuxových, Unixových a Solarisových distribucích.
- Instalace obzvlášť na Windows není příliš jednoduchá.
- Pokud program používá nějaké moduly z CPAN, je nutné je doinstalovat i na ostatních počítačích, kde má program běžet.
- Velmi strmá učící křivka.

9.3. Celkové zhodnocení jazyka

Perl je velmi silný programovací jazyk, který najde uplatnění téměř ve všech oblastech lidské činnosti, ovšem jeho největší síla tkví v nízkoúrovňové oblasti správy počítačů a operačních systémů, případně při tvorbě dynamických webových aplikací. V oblasti tvorby desktopového softwaru většinou společnosti i programátoři sáhnou po jednodušších nástrojích pro tvorbu takovýchto aplikací, ať již to je C#, Java nebo Delphi.

Přestože popularita Perlu dlouhodobě klesá nebo stagnuje, jak začíná být vytlačován novějšími, více specializovanými jazyky, či jazyky, které alespoň částečně potlačují jeho

nevýhody, nachází se v žebříčcích popularity stále na horních pozicích. Díky podpůrné komunitě pro něho také vznikají stále novější a modernější nástroje a knihovny, a přestože za ním nestojí žádná ohromná komerční organizace jako Microsoft za C# nebo Sun Microsystems za Javou, si Perl rozhodně vybojoval svoje místo v knihovnách programátorů i v komerčním prostředí. Jelikož už autoři jazyka několik let připravují novou verzi jazyka Perl, je jisté, že Perl má stále ještě co říct.

10. Seznam použité literatury

1. *Programming Language Popularity.* [Online] DedaSys LLC. [Citace: 24. 3 2010.] http://langpop.com/.
2. CPAN**.** CPAN FAQ. *CPAN.* [Online] [Citace: 24. 3 2010.] http://www.cpan.org/misc/cpan-faq.html.
3. CPAN Trends. *CPAN Testers Statistics.* [Online] [Citace: 24. 3 2010.] http://stats.cpantesters.org/trends.html.
4. DAŘENA, František. 2005. *Myslíme v jazyku PERL.* 1. vydání. Praha : Grada Publishing, a.s., 2005. str. 700. ISBN 80-247-1147-8.
5. GOOGLE. Perl. *Google Trends.* [Online] Google. [Citace: 24. 3 2010.] http://www.google.com/trends?q=perl.
6. IFPUG. *ABOUT FUNCTION POINT ANALYSIS.* IFPUG. [Online] [Citace: 24. 3 2010.] http://www.ifpug.org/about/about.htm.
7. PERL**.** *Perl Documentation.* perldoc. [Online] [Citace: 24. 3 2010.] http://perldoc.perl.org.
8. QSM. 2009. *Function Point Languages Table*. QSM. [Online] 11 2009. [Citace: 24. 3 2010.] http://www.qsm.com/?q=resources/function-point-languages-table/index.html.
9. RICHARDSON, Marjorie. 1999. *Larry Wall, the Guru of Perl.* LINUX Journal. [Online] 1. 5 1999. [Citace: 24. 3 2010.] http://www.linuxjournal.com/article/3394.
10. SATRAPA, Pavel. 2001. *PERL pro zelenáče.* 2. vydání. Praha : Neocortex, 2001. str. 224. ISBN 80-86330-02-8.
11. SHEPPARD, Doug. 2000. *Beginners introduction to Perl.* [Online] 16. 10 2000. [Citace: 24. 3 2010.] http://www.perl.com/pub/a/2000/10/begperl1.html.
12. SMITH, Ben. *The Importance of Perl.* [Online] [Citace: 24. 3 2010.] http://www.oreillynet.com/pub/a/oreilly/perl/news/importance_0498.html.
13. TIOBE. *TIOBE Programming Community Index for March 2010.* TIOBE. [Online] [Citace: 24. 3 2010.] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
14. VRÁNA, Jakub. 2010. *Verze PHP v ČR – únor 2010.* PHP triky. [Online] 5. 2 2010. [Citace: 24. 3 2010.] http://php.vrana.cz/verze-php-v-cr-unor-2010.php.
15. WALL, Larry. *Perl Licensing.* Perl. [Online] [Citace: 24. 3 2010.] http://dev.perl.org/licenses/.
16. —. 1999. *Perl, the first postmodern computer language.* perl.com. [Online] 9. Březen 1999. [Citace: 24. 3 2010.] http://www.perl.com/pub/a/1999/03/pm.html.

17. Wall, Larry, Christiansen, Tom a Orwant, Jon. 2000. *Programming Perl.* 3. vydání. Beijing : O'Reilly Media, 2000. ISBN 0-596-00027-8.

III. Přílohy

Příloha A

Seznam operačních systému, na kterých může běžet Perl podle [PERL]

Aktuální soupis i s odkazy na zdrojový kód perlu je vždy k nalezení na adrese *http://www.cpan.org/ports/*.

Zde je starší soupis přímo z dokumentace Perlu:

1. AIX
2. BeOS
3. BSD/OS (BSDi)
4. Cygwin
5. DG/UX
6. DOS DJGPP 1)
7. DYNIX/ptx
8. EPOC R5
9. FreeBSD
10. HI-UXMPP (Hitachi)
11. HP-UX
12. IRIX
13. Linux
14. Mac OS Classic
15. Mac OS X (Darwin)
16. MPE/iX
17. NetBSD
18. NetWare
19. NonStop-UX
20. ReliantUNIX (formerly SINIX)
21. OpenBSD
22. OpenVMS (formerly VMS)
23. Open UNIX (Unixware) (since Perl 5.8.1/5.9.0)
24. OS/2
25. OS/400 (using the PASE) (since Perl 5.8.1/5.9.0)
26. PowerUX
27. POSIX-BC (formerly BS2000)
28. QNX
29. Solaris
30. SunOS 4

31. SUPER-UX (NEC)
32. Tru64 UNIX (formerly DEC OSF/1, Digital UNIX)
33. UNICOS
34. UNICOS/mk
35. UTS
36. VOS
37. Win95/98/ME/2K/XP 2)
38. WinCE
39. z/OS (formerly OS/390)
40. VM/ESA

Příloha B

Přehled operátorů v Perlu.

`**`		Exponentiation
`+ -`	`* /`	Addition, subtraction, multiplication, division
`%`		Modulo division
`& |`	`^`	Bitwise AND, bitwise OR, bitwise exclusive OR
`>>`	`<<`	Bitwise shift right, bitwise shift left
`||`	`&&`	Logical OR, logical AND
`.`		Concatenation of two strings
`x`		Returns a string or array consisting of the left operand (an array or a string) repeated the number of times specified by the right operand

All of the above operators also have an assignment operator, e.g., .=

`->`		Dereference operator
`\`		Reference (unary)
`!`	`~`	Negation (unary), bitwise complement (unary)
`++`	`--`	Auto-increment (magical on strings), auto-decrement
`==`	`!=`	Numeric equality, inequality
`eq`	`ne`	String equality, inequality
`<`	`>`	Numeric less than, greater than
`lt`	`gt`	String less than, greater than
`<=`	`>=`	Numeric less (greater) than or equal to
`le`	`ge`	String less (greater) than or equal to
`<=>`	`cmp`	Numeric (string) compare. Returns -1, 0, 1.
`=~`	`!~`	Search pattern, substitution, or translation (negated)
`..`		Range (scalar context) or enumeration (array context)
`?:`		Alternation (if-then-else) operator
`,`		Comma operator, also list element separator. You can also use =>.
`not`		Low-precedence negation
`and`		Low-precedence AND
`or`	`xor`	Low-precedence OR, exclusive OR