

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-64685

**UNIVERZÁNE, PLATFORMOVO NEZÁVSLÉ  
KOZOLOVÉ ROZHANIE  
DIPLOMOVÁ PRÁCA**

**2018**

**Bc. Juraj Vraniak**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-64685

**UNIVERZÁNE, PLATFORMOVO NEZÁVSLÉ**  
**KOZOLOVÉ ROZHANIE**  
**DIPLOMOVÁ PRÁCA**

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	RnDr. Igor Kossaczský, CSc.
Konzultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó

**Bratislava 2018**

**Bc. Juraj Vraniak**

Základné údaje

Typ práce:	Diplomová práca
Názov témy:	Akupunktúrne body – hľadanie, meranie a zobrazovanie
Stav prihlásenia:	schválené
Stav témy:	schválené (prof. Dr. Ing. Miloš Oravec - Garant študijného programu)
Vedúci práce:	doc. Ing. Marek Kukučka, PhD.
Fakulta:	Fakulta elektrotechniky a informatiky
Garantujúce pracovisko:	Ústav informatiky a matematiky - FEI
Max. počet študentov:	1
Akademický rok:	2016/2017
Navrhoľ:	doc. Ing. Marek Kukučka, PhD.
Abstrakt:	Akupunktúra patrí medzi najstaršie liečebné praktiky sveta a je to jedna z kľúčových častí tradičnej čínskej medicíny. Keďže použitie liečebných metód odvodených od akupunktúry je stále viac rozšírené, z medicínskeho pohľadu je nanajvýš aktuálne venovať sa základnému výskumu v tejto oblasti a pokúsiť sa objasniť základné fyziologické a biofyzikálne mechanizmy stojace za preukázanými klinickými efektami. Z prehľadu publikovaných elektrických vlastností akupunktúrnych bodov a dráh vyplýva potreba dôsledného overenia hypotézy elektrickej rozoznateľnosti akupunktúrnych štruktúr. Očakáva sa, že môžu mať nižšiu impedanciu a vyššiu kapacitu oproti okolitým kontrolným bodom na pokožke. Výstupom mapovania pokožky budú 2D a 3D napäťové/impedančné mapy z povrchu tela. S týmto prístupom bude možné nielen lokalizovať prípadný akupunktúrny bod, ale aj študovať jeho ohraničenie, povrchovú elektrickú štruktúru či jeho veľkosť. Súčasťou výskumu je aj realizovanie meraní závislosti impedancie od frekvencie v akustickom pásme frekvencií 100 Hz – 20 kHz a vplyvu rôznych parametrov na rozoznávanie pozície, tvaru a štruktúry akupunktúrnych bodov.

Obmedzenie k téme

Na prihlásenie riešiteľa na tému je potrebné splnenie jedného z nasledujúcich obmedzení

Obmedzenie na študijný program

Tabuľka zobrazuje obmedzenie na študijný program, odbor, špecializáciu, ktorý musí mať študent zapísaný, aby sa mohol na danú tému prihlásiť.

Program	Zameranie	Špecializácia
I-API aplikovaná informatika	I-API-MSUS Modelovanie a simulácia udalostných systémov	-- nezadané --
I-API aplikovaná informatika	I-API-ITVR IT v riadení a rozhodovaní	-- nezadané --

Obmedzenie na predmety

Tabuľka zobrazuje obmedzenia na predmet, ktorý musí mať študent odštudovaný, aby sa mohol na danú tému prihlásiť.

# SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Juraj Vraniak
Diplomová práca:	Univerzáne, platfor- movo nezávislé kozolové rozhanie
Vedúci záverečnej práce:	RnDr. Igor Kossaczky, CSc.
Konzultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó
Miesto a rok predloženia práce:	Bratislava 2018

Práca sa venuje problematike programovacích jazykov, ich syntaxea a dačo snád vymyslím.

Kľúčové slová: programovacie jazyk, analýza prekladu, prekladač, plugin, architektúra,

návrhové vzory

# ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Juraj Vraniak
Master's thesis:	Universal, platform independent console interface
Supervisor:	RnDr. Igor Kossaczký, CSc.
Consultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó
Place and year of submission:	Bratislava 2018

The aim of the thesis is dedicated to programming languages, their syntax and will figure something out later.

Keywords: programming language, translation analysis, translator, plugin, architecture, design patterns

## Vyhlásenie autora

Podpísaný Bc. Juraj Vraniak čestne vyhlasujem, že som diplomovú prácu Univerzít, platformovo nezávislé kozolové rozhodnutie vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol RNDr. Igor Kossaczský, CSc.

Bratislava, dňa 9.2.2018

.....  
podpis autora

# Pod'akovanie

Touto cestou by som sa chcel podakovať vedúcim práce RnDr. Igorovi Kossackému, CSc, Rndr. Peterovi Praženicovi, Ing. Gabrielovi Szabóovi za cenné rady, odbornú pomoc, trpezlivosť a konzultácie pri vytvorení diplomovej práce.

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Analýza stavu</b>	<b>2</b>
<b>2 Operačné systémy</b>	<b>3</b>
2.1 Windows . . . . .	3
2.2 MacOS . . . . .	3
2.3 Unix . . . . .	3
2.4 Linux . . . . .	4
<b>3 Programovacie jazyky</b>	<b>5</b>
3.1 Shell . . . . .	5
3.1.1 Výhody . . . . .	5
3.1.2 Nevýhody . . . . .	5
3.1.3 Popis a zhodnotenie jazyka . . . . .	6
3.2 Powershell/Classic command line . . . . .	7
3.2.1 Výhody . . . . .	7
3.2.2 Nevýhody . . . . .	7
3.2.3 Popis a zhodnotenie jazyka . . . . .	7
3.3 Python . . . . .	9
3.3.1 Výhody . . . . .	9
3.3.2 Nevýhody . . . . .	9
3.3.3 Popis a zhodnotenie jazyka . . . . .	9
<b>4 Analýza existujucich riešení</b>	<b>11</b>
4.1 ConEmu . . . . .	11
4.1.1 Skúsenosti . . . . .	11
4.2 cmdr . . . . .	12
4.2.1 Skúsenosti . . . . .	12
4.3 Babun . . . . .	12
4.4 MobaXterm . . . . .	13
4.4.1 Neplatená verzia . . . . .	13
4.4.2 Platená verzia . . . . .	14
<b>5 Zhodnotenie analyzovaných technológií</b>	<b>15</b>



<b>6</b>	<b>Programovacie jazyky a kompilátory</b>	<b>16</b>
6.1	Kompilátor proces prekladu . . . . .	16
6.1.1	Lexikálna analýza . . . . .	16
6.1.2	Syntaktická analýza . . . . .	17
6.1.3	Limitácia syntaktickej analýzy . . . . .	18
6.1.4	Semantická analýza . . . . .	18
6.1.5	Generovanie cieľového jazyka . . . . .	18
6.2	Iterpreter . . . . .	18
6.3	Abeceda a vyhradené slová jazyka . . . . .	19
6.4	Procedúry a algoritmy . . . . .	19
<b>7</b>	<b>Návrh</b>	<b>20</b>
7.1	Use case . . . . .	20
7.1.1	Popis use casov . . . . .	21
7.2	Prvotný nástrel . . . . .	21
7.3	Oddelenie štruktúry . . . . .	21
<b>8</b>	<b>Architektúra aplikácie</b>	<b>22</b>
8.1	Java . . . . .	22
8.2	Použite navrhové vzory . . . . .	22
8.2.1	Command - príkaz . . . . .	22
8.2.2	Factory - továreň . . . . .	24
8.2.3	Interpreter . . . . .	24
8.3	Komponenty aplikácie . . . . .	25
<b>9</b>	<b>Zhodnotenie výsledkov</b>	<b>26</b>
	<b>Záver</b>	<b>27</b>
	<b>Zoznam použitej literatúry</b>	<b>I</b>
	<b>Prílohy</b>	<b>I</b>
<b>A</b>	<b>CD s aplikáciou</b>	<b>II</b>
<b>B</b>	<b>Návod na spustenie a používanie aplikácie</b>	<b>III</b>

# Zoznam obrázkov a tabuliek

Obrázok 1	Ukážka ConEmu emulátora . . . . .	11
Obrázok 2	Ukážka cmder emulátora . . . . .	12
Obrázok 3	Ukážka babun emulátora . . . . .	13
Obrázok 4	Ukážka práce lexikálneho analyzátora . . . . .	17
Obrázok 5	Ukážka práce syntaktického analyzátora . . . . .	17
Obrázok 6	Use case pre navrhovanú aplikáciu . . . . .	20
Obrázok 7	Class diagram Command návrhového vzoru . . . . .	23
Obrázok 8	Sekvenčný diagram Command návrhového vzoru . . . . .	24
Obrázok 9	Class diagram Factory návrhového vzoru . . . . .	24
Obrázok 10	Prvé funkčné riešenie . . . . .	25
Obrázok 11	Class diagram pluginu . . . . .	26
Tabuľka 1	Ukážka prepínačov v podmienkovom výraze if . . . . .	6

# Zoznam algoritmov

1	Ukážka použitia pipe v powershell. . . . .	8
---	--	---

# Úvod

V dnešnej dobe rôznorodosť operačných systémov a absencia jednotnej platformy na vytváranie skrípt vo väčšine prípadov vyžadujú ich duplikovanie alebo viacnásobnú implementáciu. Čiastočným riešením tohto problému je použitie skriptovacieho jazyka s podporou cieľových platform. Zásadným problémom skriptovacích jazykov pri riešení tohto problému je absencia syntaktických a funkčných konštrukcií, ktoré sú už overené a široko používané, ako napríklad pajpa, izolovanie príkazov alebo presmerovanie štandardného a chybového vstupu a výstupu. Cieľom práce je analyzovať populárne konzolové rozhrania (napr. Bourne Shell, Power Shell, C-Shell) a skriptovacie jazyky (napr. Python, Groovy, Lua), porovnať ich syntax, funkcionality a limity. Následne navrhnúť nové konzolové rozhranie, ktoré bude spájať funkcionality identifikované ako výhody počas analýzy so zameraním na administrátorské úlohy. Pri implementácii je tiež kľúčovým faktorom identifikácia nových funkcionalít, ktoré by mohli uľahčiť vývoj robustných skrípt. Rozhranie musí umožňovať interaktívny aj skriptovaný módus. Očakáva sa možnosť integrácie rozhrania do iných systémov rôznej veľkosti a komplexity, od malých utilít a rutín až po enterprise aplikácie a ľahká rozšíriteľnosť rozhrania o nové príkazy a funkcionality.

# 1 Analýza stavu

## 2 Operačné systémy

Informatika a informačné technológie je pomerne mladá vedná disciplína. Jej začiatky je možné datovať od druhej polky dvadsiateho storočia, čo momentálne predstavuje necelých sedemdesiat rokov. Za tento čas informatika zaznamenala enormný rast či vo vývoji hardvéru alebo softvéru. Kus softvéru, ktorý umožňuje počítačom fungovať je operačný systém. Poznáme ich hneď niekoľko v nasledujúcich sekciách si ich stručne popíšeme.

### 2.1 Windows

Microsoft Windows uviedol svoje prvé operačné systémy v roku 1985 ako nadstavbu MS DOS. Jeho popularita rýchlo rástla až vyvrcholila dominantným postavením na trhu v osobných počítačoch. V roku 1993 začal vydávať špecializované operačné systémy pre servery, ktoré prinášali novú funkcionálnosť pre počítače používané ako servery. Pre účely automatizácie sa na Windows serveroch používajú hlavne powershell scripty, písané v rovnomenom jazyku powershell.

### 2.2 MacOS

Mac tiež ponúka serverovú verziu svojho operačného systému pod názvom OS X Server, ktorý začal písať svoju históriu v roku 2001, avšak neteší sa takej obľube ako Windows, unix alebo linux server. Skriptovacím jazykom pre OS X server nieje špecifický jazyk, je možné vybrať si z Pythonu, JavaScriptu, Perl, AppleScriptu, Swiftu alebo napríklad Ruby. Každý jazyk prináša určité plusy, ale zároveň mínusy čo je však najpodstatnejšie nie je tam štandardizovaný skriptovací jazyk.

### 2.3 Unix

Patrí medzi prvé operačné systémy pre servery, ktorých vývoj začal v roku 1970 v priebehu rokov vzniklo nespočetné množstvo nových verzií Unixu. Unixové servery sa tešili veľkej obľube hlavne v minulosti momentálne sú na ústupe hlavne kvôli vyšším nákladom na ich zaobstaranie a prevádzku. Pre účely unixu sa vytvoril shell script, obľúbený skriptovací jazyk, ktorý sa v rôznych obmenách teší veľkej obľube medzi administrátormi a automatizačnými programátormi.

## 2.4 Linux

Prvé vydanie Linux bolo 17. septembra 1991, bol rozšírený na najviac platforiem a momentálne sa pýši tým, že je jediný používaný operačný systém na TOP 500 superpočítačoch(mainframoch) Skriptovací jazyk shell script.

## 3 Programovacie jazyky

S príchodom osobných počítačov no najmä serverov, sa programátori zaujímali o automatizáciu procesov, ktoré na danom stoji bolo spočiatku potrebné spúšťať manuálne. Nakoľko tieto úlohy neboli na toľko komplexné ako samotné programy, ktoré spúšťali bolo vhodné na tieto úlohy využiť/vytvoriť skriptovacie jazyky. V nasledujúcej časti si priblížime zopár programovacích jazykov, ktoré sa v dnešnej dobe bežne používajú na tvorbu automatizovaných skriptov.

### 3.1 Shell

Je skriptovacím jazykom pre unixové distribúcie. Počas rokov prešiel roznoými zmenami a rozšíreniami. Verzie shellu sú: sh, csh, ksh, tcsh, bash. Bash sa momentálne teší najväčšej obľube no zsh je verzia shellu, ktorá má najviac rôznych rozšírení funkcionality ako aj veľa priaznivcov medzi developermi. V nasledujúcich častiach všeobecne zhodnotíme jednotlivé výhody resp. nevýhody tohoto skriptovacieho jazyka.

#### 3.1.1 Výhody

- automatizácia často opakujúcich sa úloh
- dokáže zbíhať zložitú zloženú príkazy ako jednoriadkový príkaz - tzv. reťazenie príkazov
- ľahký na používanie
- výborné manuálové stránky
- ak hovoríme o shell scripte je portabilný naprieč platformami linuxu-unixu
- jednoduché plánovanie automatických úloh

#### 3.1.2 Nevýhody

- asi najväčšou nevýhodou je že natívne nefunguje pod windowsom, existujú iba rozne emulátory a 3rd tooly, ktoré sprostredkujú jeho funkcionality.
- pomalé vykonávanie príkazov pri porovnaní s inými programovacími jazykmi
- nový proces pre skoro každý spustený príkaz
- zložitejšie na pamätanie si rôznych prepínačov, ktoré dané príkazy podporujú
- nejednotnosť prepínačov(hoc to by asi ani nešlo)



- neprenosný medzi platformami

### 3.1.3 Popis a zhodnotenie jazyka

Shell script je obľúbeným scriptovacím jazykom, vhodným na automatizovanie každodenných operácií. Je jedným z najpoužívanějších skriptovacích jazykov vôbec, nakoľko všetky linuxové, unixové servery využívajú práve tento jazyk ako svoj primárny. Medzi jeho silné stránky patrí jednoduchá manipulácia s crontable, pomocou ktorej vie admin jednoducho plánovať beh procesov. Avšak syntax jazyka sa učí ťažšie nakoľko používa rôzne prepínače, ktoré novému používateľovi nemusia byť sprvu jasné. V tabuľke uvádzame príklad prepínačov pre `if`, tiež je vhodné poznamenať, že shell script používa hranaté zátvorky namiesto okrúhlych na aké sme zvyknutý z väčšiny programovacích jazykov. `if` ponúka aj ďalšie prepínače no zhodnotili sme, že pre ilustráciu budú postačovať aj ukázané. Najväčšia nevýhoda je, že ani shell script nie je jazyk, ktorý by bol multiplatformový a teda ak by sme mali prostredie, kde servery bežia na rôznych operačných systémoch, potrebujeme poznať ďalší jazyk, ktorým docielime rovnaké alebo aspoň podobné výsledky.

String Comparison	Description
<code>Str1 = Str2</code>	Returns true if the strings are equal
<code>Str1 != Str2</code>	Returns true if the strings are not equal
<code>-n Str1</code>	Returns true if the string is not null
<code>-z Str1</code>	Returns true if the string is null
Numeric Comparison	Description
<code>expr1 -eq expr2</code>	Returns true if the expressions are equal
<code>expr1 -ne expr2</code>	Returns true if the expressions are not equal
<code>expr1 -gt expr2</code>	Returns true if expr1 is greater than expr2
<code>expr1 -ge expr2</code>	Returns true if expr1 is greater than or equal to expr2
<code>expr1 -lt expr2</code>	Returns true if expr1 is less than expr2
<code>expr1 -le expr2</code>	Returns true if expr1 is less than or equal to expr2
<code>! expr1</code>	Negates the result of the expression

Tabuľka 1: Ukážka prepínačov v podmienkovom výraze `if`

## 3.2 Powershell/Classic command line

Command line je základným skriptovacím jazykom pre windows distribúcie, ktorý poskytuje malé API pre svojich používateľov. Aj kôli tomu Microsoft prišiel s novým jazykom Powershell. Powershell je kombináciou príkazového riadku, funkcionálneho programovania a objektovo orientovaného programovania. Je založený na .NET, ktorý mu dáva istú mieru flexibility, ktorá v

TODO: dopísať dake sprostosti

Jeho výhody a nevýhody si popíšeme v nasledujúcich častiach.

### 3.2.1 Výhody

- bohaté api
- výborne riešený run-time
- flexibilný
- veľmi jednoduché prepnúť z .NET

### 3.2.2 Nevýhody

- bohaté api - nejednoznačné, kedy čo použiť
- niektoré výhody jazyka sú až nevhodne skryté pred používateľmi
- staršie verzie serverov nie sú Powershell-om podporované ako novšie
- dokumentácia je horšia ako v prípade Shell scriptu

### 3.2.3 Popis a zhodnotenie jazyka

Powershell je obľúbený medzi programátormi a administrátormi, ktorý pracujú pod operačným systémom Windowsu. Do nedávna kým Powershell bežal na .NET frameworku ho nebolo možné používať mimo operačných systémov Windows, avšak s príchodom frameworku .NET Core sa situácia zmenila. Tento framework je momentálne open source, jeho zdrojové kódy boli zverejnené a je doň možné prispievať. Okrem iného podporuje rovnaké alebo aspoň podobné štruktúry ako Shell script a poskytuje v niektorých prípadoch rovnaké príkazy ako napríklad : mv, cp, rm, ls. Jedným zo zásadných rozdielov medzi Shellom a Powershellom je ten, že kým v Shelli sú pre vstup aj výstup používané textové reťazce, ktoré je potrebné rozparsovať a interpretovať v Powershelli je všetko presúvané

ako objekt. Po preštudovaní si materiálov jazyka z Mastering Windows PowerShell Scripting - Second Edition, mi to príde ako najzásadnejší rozdiel, nakoľko ostatné veci boli zrejme navrhované v spolupráci s používateľmi Shell scriptu. Aj keď Powershell môže na prvý pohľad vypadáť výborne je to stále nástroj Windowsu, ktorý sa môže zo dňa na deň rozhodnúť, že toto už nebude v ďalších releasoch podporované a tým pádom by som sa pozrel po iných možnostiach.

Pre demonštráciu rozdielov pri odovzdávaní si parametrov medzi príkazmi prikladám nasledovný príklad, ktorý som našiel komunitných stránkach superusera.

---

**Algoritmus 1** Ukážka použitia pipe v powershell.

---

```
function changeName(\$myObject)
{
    if (\$myObject.GetType() -eq [MyType])
    {
        //vypíš obsah premennej
        \$myObject.Name
        //zmeň reťazec pre atribút name
        \$myObject.Name = "NewName"
    }
    return \$myObject
}

// Vytvorenie objektu s argumentom OriginalName a následné použitie funkcie
//PS> \$myObject = New-Object MyType -arg "OriginalName"
//PS> \$myObject = changeName \$myNewObject
//OriginalName
//PS> \$myObject.Name
//NewName
// Ukážka s využitím pipe
//PS> \$myObject = New-Object MyType -arg "OriginalName" | changeName
//OriginalName
//PS> \$myObject.Name
//NewName
```

---

## 3.3 Python

Do analýzy sme sa rozhodli pripojiť aj Python, nakoľko je jedným z najpopulárnejších programovacích jazykov súčasnosti. Je viacúčelový, patrí medzi vyššie programovacie jazyky, objektovo orientovaný, interaktívny, interpretovaný a extrémne používateľsky prijateľný.

### 3.3.1 Výhody

- Je ľahko čitateľný, tým pádom ľahšie pochopiteľný
- Syntax orientovaná na produktivitu
- Multiplatformový
- Množstvo rôznych knižníc
- Open source

### 3.3.2 Nevýhody

- Rýchlosť
- Chybovosť za behu
- Slabšia dokumentácia
- Nevhodný pre úlohy pracujúce s vyšším množstvom pamäte
- Nevhodný pre viac-procesorovú prácu

### 3.3.3 Popis a zhodnotenie jazyka

Nakoľko nemam dostatočné skúsenosti aby som mohol plnohodnotne zhodnotiť Python ako jazyk, pozitíva a negatíva som radšej pohladal na komunitných stránkach, kde ľudia píšu o svojich skúsenostiach s jazykom. Jazyk poskytuje štruktúry ako pipa, dokážeme v ňom jednoducho pracovať s procesmi, vytvárať triedy, inštancie, jednoducho prototypovať a odsymulovať rôzne problémy. Aj napriek veľkým výhodám, ktorými tento jazyk oplýva som si však nie istý stránkou bezpečnosti. Z vlastnej skúsenosti viem, že ak tento jazyk nieje používaný správne, je možné, že výsledok našej práce vyjde nazmar. Za príklad si zoberme package subprocesses, subprocesses.call() má optional shell parameter. V momente, že je nastavený na hodnotu true vystavujeme program zraniteľnosti tzv. Shell injection. Python je veľmi zaujímavý jazyk ale na prácu na serveroch kde môže byť

ohrozená bezpečnosť je jeho nasadenie zatiaľ veľmi otázne, preto ho nevidím ako vhodnú voľbu.

## 4 Analýza existujúcich riešení

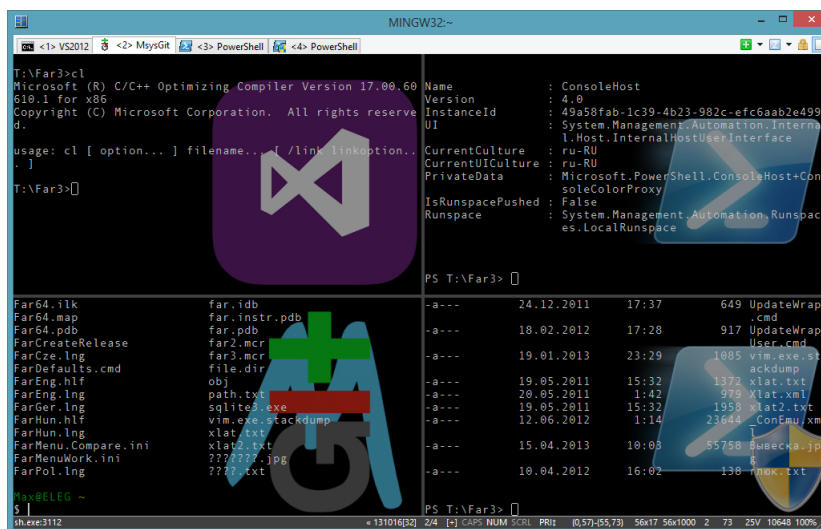
Existuje množstvo emulátorov a 3rd toolov, ktoré sprostredkujú funkcionality shell skriptu do Windowsu, niektoré z nich si predstavíme.

### 4.1 ConEmu

ConEmu je konzolový emulátor, ktorý poskytuje jednoduché GUI do ktorého je možné vložiť viacero konzol. Dokáže spúšťať jednoduché GUI aplikácie ako napríklad Putty, Cygwin. Poskytuje množstvo nastavení ako nastavenie kurzora, priehľadnosti, písma a pod. Podporuje Windows 2000 a neskoršie verzie. Neposkytuje verziu pre ine operačné systémy.

#### 4.1.1 Skúsenosti

ConEmu je podarený emulátor, ktorý je schopný vykonávať akýkoľvek skript. Používaním sme neprišli na závažné nedostatky, ktoré by neboli popísané v ich issues logu na githubu. Ale ako každý software je aj ConEmu náchylný na chyby. Podľa ich issues logu sa do ich oficiálnych releasov dostávajú rôzne problémy, ktoré neboli problémom v predchádzajúcich verziách. Na základe ich issue logu usudzujem, že ich testy niesú dostatočne vypracované na zaručenie kvality produktu, čo by sa pri použití v produkčných systémoch nestretlo s veľkým pochopením zákazníkov. Preto by som tento emulátor nepovažoval za dostatočnú náhradu za shell.



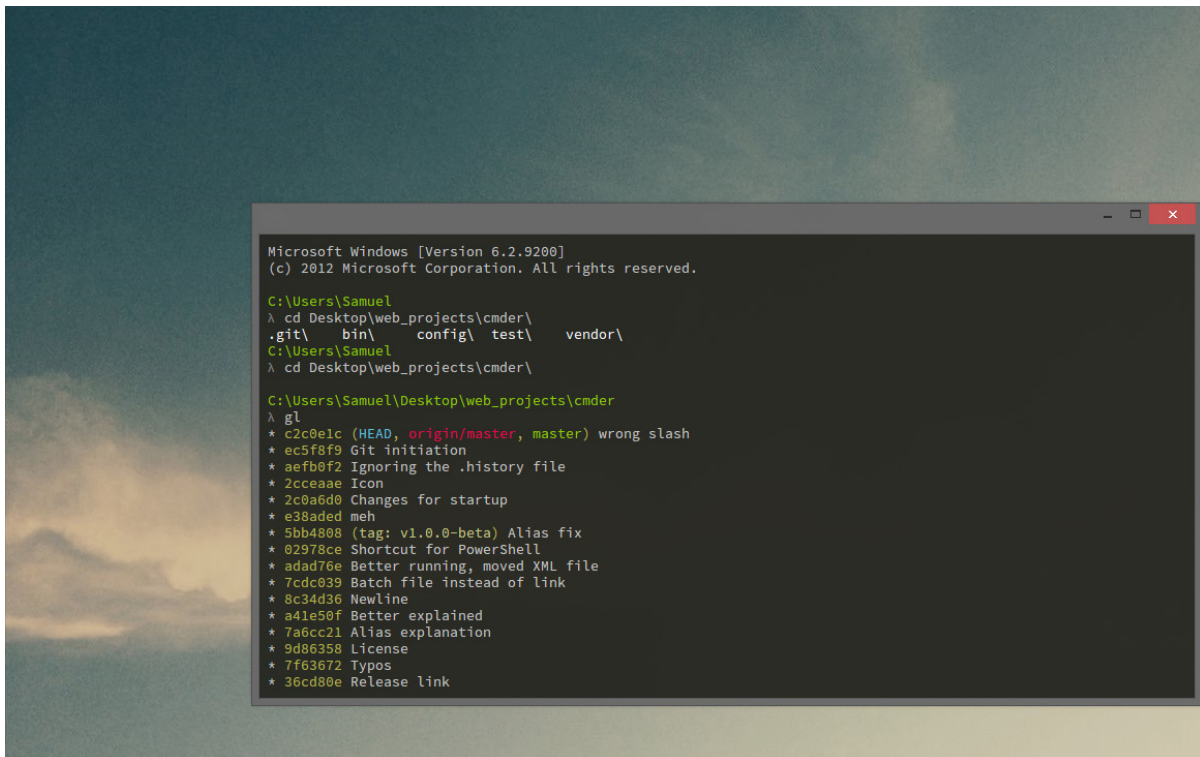
Obrázok 1: Ukážka ConEmu emulátora

## 4.2 cmdr

Cmder je ďalším príkladom emulátora shell terminálu. Vychádza z troch projektov ConEmu, Clink a Git pre Windows - voliteľná súčasť. ConEmu sme si predstavili v predchádzajúcej časti s jeho kladnými a zápornými. Clink, konkrétne clink-completions je v projekte využívaný na zvýšenie komfortu pri písaní skriptov, nepridáva ďalšiu shellovú funkcionálnosť.

### 4.2.1 Skúsenosti

ConEmu je príjemný nástroj, dokáže zjednodušiť človeku prácu obzvlášť ak je zvyknutý na programovanie v shell scripte. Nakoľko cmder používa ConEmu ako emulátor shell terminálu a je úzko určený pre Windows platformu nemožno hovoriť o multiplatformovom riešení.

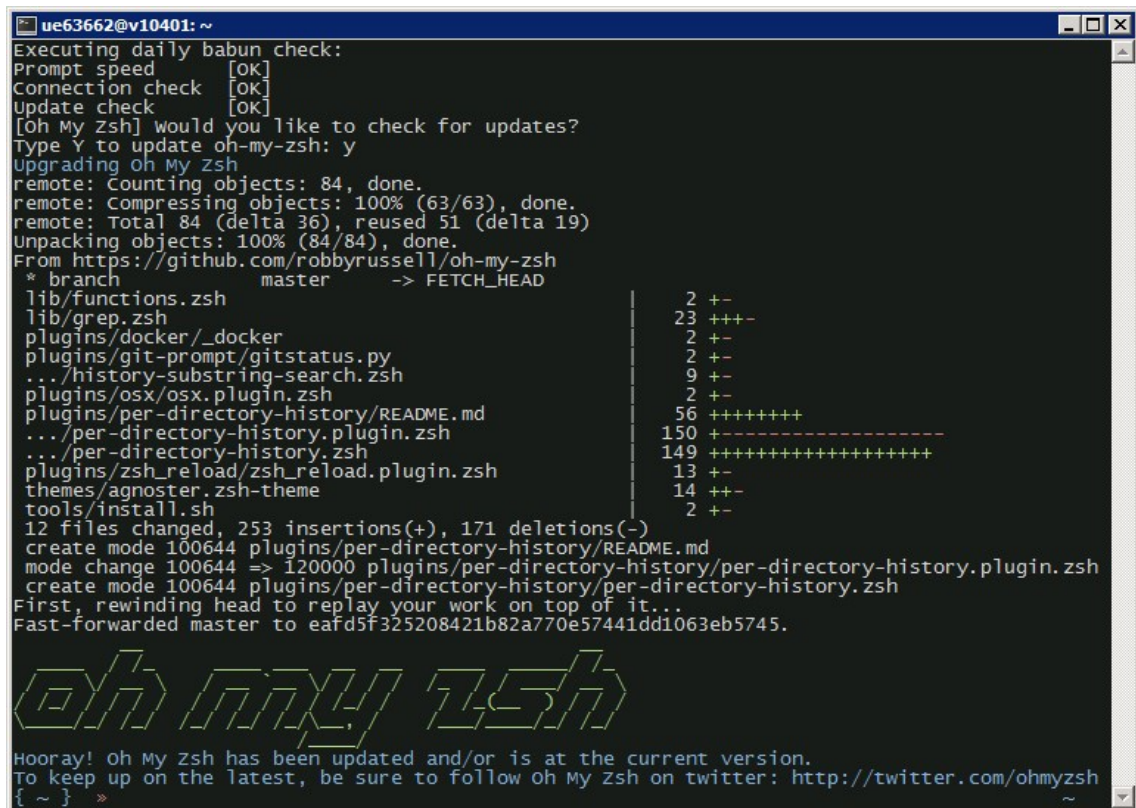


Obrázok 2: Ukážka cmder emulátora

## 4.3 Babun

Babun je ďalším z množstva emulátorov pre Windows. Je nadstavba cygwinu. Používa zshell a bash. Zafarbovanie textu podľa zdrojového jazyka. Je tam git, svn, puython, perl. Integrované sťahovanie nových packagov ktoré ponúka cygwin pomocou kľúčového slova pact. Nakoľko to používa zsh a bash nevidím problém s kompatibilitou skriptov, avšak

zas a znova je to terminál iba pre Windows, teda značne oklieštené možnosti.



```
ue63662@v10401: ~  
Executing daily babun check:  
Prompt speed [OK]  
Connection check [OK]  
Update check [OK]  
[Oh My Zsh] would you like to check for updates?  
Type Y to update oh-my-zsh: y  
Upgrading Oh My Zsh  
remote: Counting objects: 84, done.  
remote: Compressing objects: 100% (63/63), done.  
remote: Total 84 (delta 36), reused 51 (delta 19)  
Unpacking objects: 100% (84/84), done.  
From https://github.com/robbyrussell/oh-my-zsh  
* branch master -> FETCH_HEAD  
lib/functions.zsh 2 +-  
lib/grep.zsh 23 +++-  
plugins/docker/_docker 2 +-  
plugins/git-prompt/gitstatus.py 2 +-  
.../history-substring-search.zsh 9 +-  
plugins/osx/osx.plugin.zsh 2 +-  
plugins/per-directory-history/README.md 56 ++++++  
.../per-directory-history.plugin.zsh 150 +-----  
.../per-directory-history.zsh 149 ++++++  
plugins/zsh_reload/zsh_reload.plugin.zsh 13 +-  
themes/agnoster.zsh-theme 14 ++-  
tools/install.sh 2 +-  
12 files changed, 253 insertions(+), 171 deletions(-)  
create mode 100644 plugins/per-directory-history/README.md  
mode change 100644 => 120000 plugins/per-directory-history/per-directory-history.plugin.zsh  
create mode 100644 plugins/per-directory-history/per-directory-history.zsh  
First, rewinding head to replay your work on top of it...  
Fast-forwarded master to eafd5f325208421b82a770e57441dd1063eb5745.  
  
oh my zsh  
Hooray! Oh My Zsh has been updated and/or is at the current version.  
To keep up on the latest, be sure to follow Oh My Zsh on twitter: http://twitter.com/ohmyzsh  
{ ~ } »
```

Obrázok 3: Ukážka babun emulátora

## 4.4 MobaXterm

Poskytuje množstvo funkcionality avšak je zatažený licenciou v hodnote 50 eur.

### 4.4.1 Neplatená verzia

- Full X server and SSH support
- Remote desktop (RDP, VNC, Xdmcp)
- Remote terminal (SSH, telnet, rlogin, Mosh)
- X11-Forwarding
- Automatic SFTP browser
- Plugins support
- Portable and installer versions
- Full documentation



- Max. 12 sessions
- Max. 2 SSH tunnels
- Max. 4 macros
- Max. 360 seconds for Tftp, Nfs and Cron

#### **4.4.2 Platená verzia**

- Every feature from Home Edition +
- Customize your startup message and logo
- Modify your profile script
- Remove unwanted games, screensaver or tools
- Unlimited number of sessions
- Unlimited number of tunnels and macros
- Unlimited run time for network daemons
- Master password support
- Professional support
- Lifetime right to use

## 5 Zhodnotenie analyzovaných technológií

## 6 Programovacie jazyky a kompilátory

Pri programovacích jazykoch nás zaujímajú ich vyjadrovacie schopnosti ako aj vlastnosti z hľadiska ich rozpoznania. Tieto vlastnosti sa týkajú programovania a prekladu, pričom obe je potrebné zohľadniť pri tvorbe jazyka. V dnešnej dobe sa používajú na programovanie hlavne takzvané vyššie programovacie jazyky, môžeme ich označiť ako zdrojové jazyky. Na to aby vykonávali čo používateľ naprogramoval je potrebné aby boli pretransformované do jazyka daného stroja. Spomínanú transformáciu zabezpečuje prekladač, prekladačom máme na mysli program, ktorý číta zdrojový jazyk a transformuje ho do cieľového jazyka, ktorému rozumie stroj.[1]

### 6.1 Kompilátor proces prekladu

Aby bol preklad možný, musí byť zdrojový kód programu napísaný podľa určitých pravidiel, ktoré vyplývajú z jazyka. Proces prekladu je možné rodeliť na 4 hlavné časti.

- lexikálna analýza
- syntaktická analýza
- spracovanie sémantiky
- generovanie cieľového jazyka

Podrobnejšie si stručne popíšeme všetky štyri časti, ktoré majú pre nás z hľadiska prekladu najväčší zmysel.

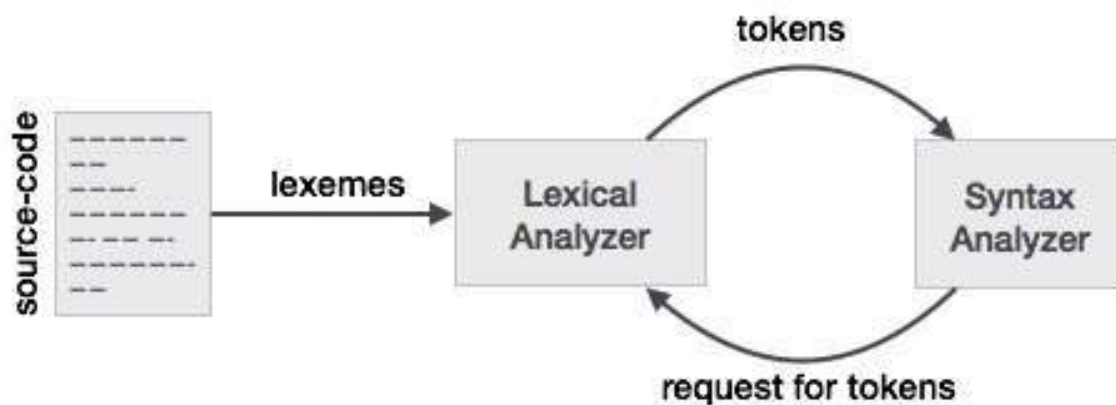
#### 6.1.1 Lexikálna analýza

Lexikálna analýza je prvou fázou kompilátora. Dopredu napísaný zdrojový kód je postupne spracovávaný preprocesorom, ktorý vytvára takzvané lexémy.

Lexémou nazývame postupnosť alfanumerických znakov. Tieto postupnosti znakov sú následne vkladané do lexikálneho analyzátora, ktorý ma za úlohu vytvoriť zo vstupných lexém tokeny slúžiace ako vstup pre syntaktický analyzátor.

Tokeny sa vytvárajú na základe preddefinovaných pravidiel, ktoré sa v programovacích jazykoch definujú ako pattern. V prípade, že lexikálny analyzátor nieje schopný nájsť pattern pred danú lexému musí vyhlásiť chybu počas tokenizácie.

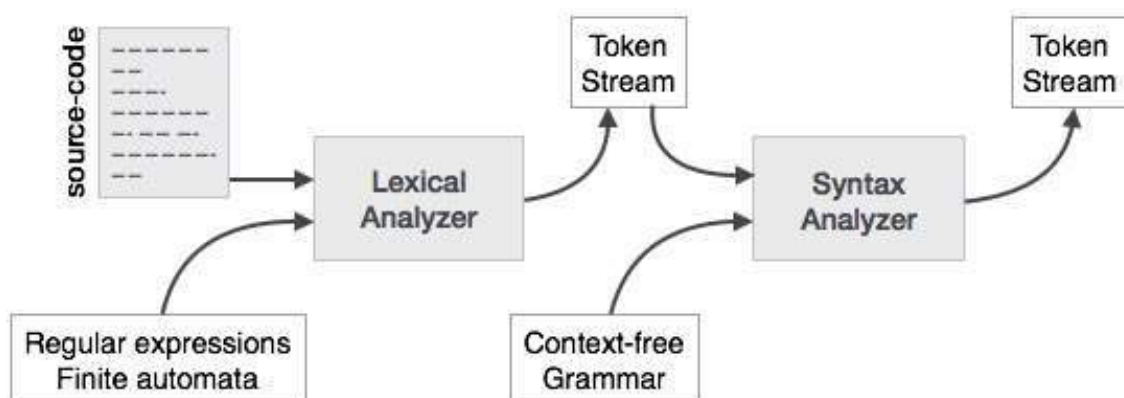
Výstupom z lexikálnej analýzy sú takzvané tokeny, ktoré tvoria vyššie jednotky jazyka ako kľúčové slová jazyka, konštanty, identifikátory, operátory a iné.



Obrázok 4: Ukážka práce lexikálneho analyzátora

### 6.1.2 Syntaktická analýza

Ďalšou fázou je syntaktická analýza. Úlohou Syntaktického analyzátora je kontrola správnosti vytvorených tokenov s uchovaním niektorých získaných informácií o štruktúre skúmanej syntactickej jednotky. Syntaktická analýza sa radí medzi bezkontextové gramatiky. Po skončení syntactickej analýzy prichádza na rad sémantická analýza.



Obrázok 5: Ukážka práce syntactickeho analyzátora

### 6.1.3 Limitácia syntaktickej analýzy

Syntaktický analyzátor získa vstup z tokenu, ktorý vytvorí lexikálny analyzátor. Lexikálne analyzátory sú zodpovedné za validitu tokenu. Syntaktické analyzátory majú nasledovné limitácie.

- nedokážu zistiť validitu tokenu
- nedokážu zistiť či je token používaný pred tým ako je deklarovaný
- nedokážu zistiť či je token používaný pred tým ako je inicializovaný
- nedokážu zistiť validitu operácie, ktorú token vykonáva

### 6.1.4 Semantická analýza

Sémantická analýza má za úlohu interpretovať symboly, typy, ich vzťahy. Sémantická analýza rozhoduje či má syntax programu význam alebo nie. Ako príklad zisťovania významu môžeme uviesť jednoduchú inicializáciu premennej.

```
int integerVariable = 6  
  
int secondIntegerVariable = "six"
```

Oba príklady by mali prejsť cez lexikálnu a syntaktickú analýzu. Je až na sémantickej analýze aby rozhodla o správnosti zápisu programu a v prípade nesprávneho zápisu informovala o chybe. Hlavné úlohy sémantickej analýzy sú:

- zisťovanie dosahu definovaných tokenov takzvaný scoping
- kontrola typov
- deklarácia premenných
- definícia premenných
- viacnásobná deklarácia premenných v jedno scope

### 6.1.5 Generovanie cieľového jazyka

Generovanie cieľového jazyka môžeme považovať za poslednú fázu kompilátora. V tejto fáze sa preklápa jazyk z vyššieho jazyka do strojového jazyka, ktorý úspešne prešiel cez analyzačné časti.

## 6.2 Interpreter

popis interpretera, hľadám cosi schopne.

### **6.3 Abeceda a vyhradené slová jazyka**

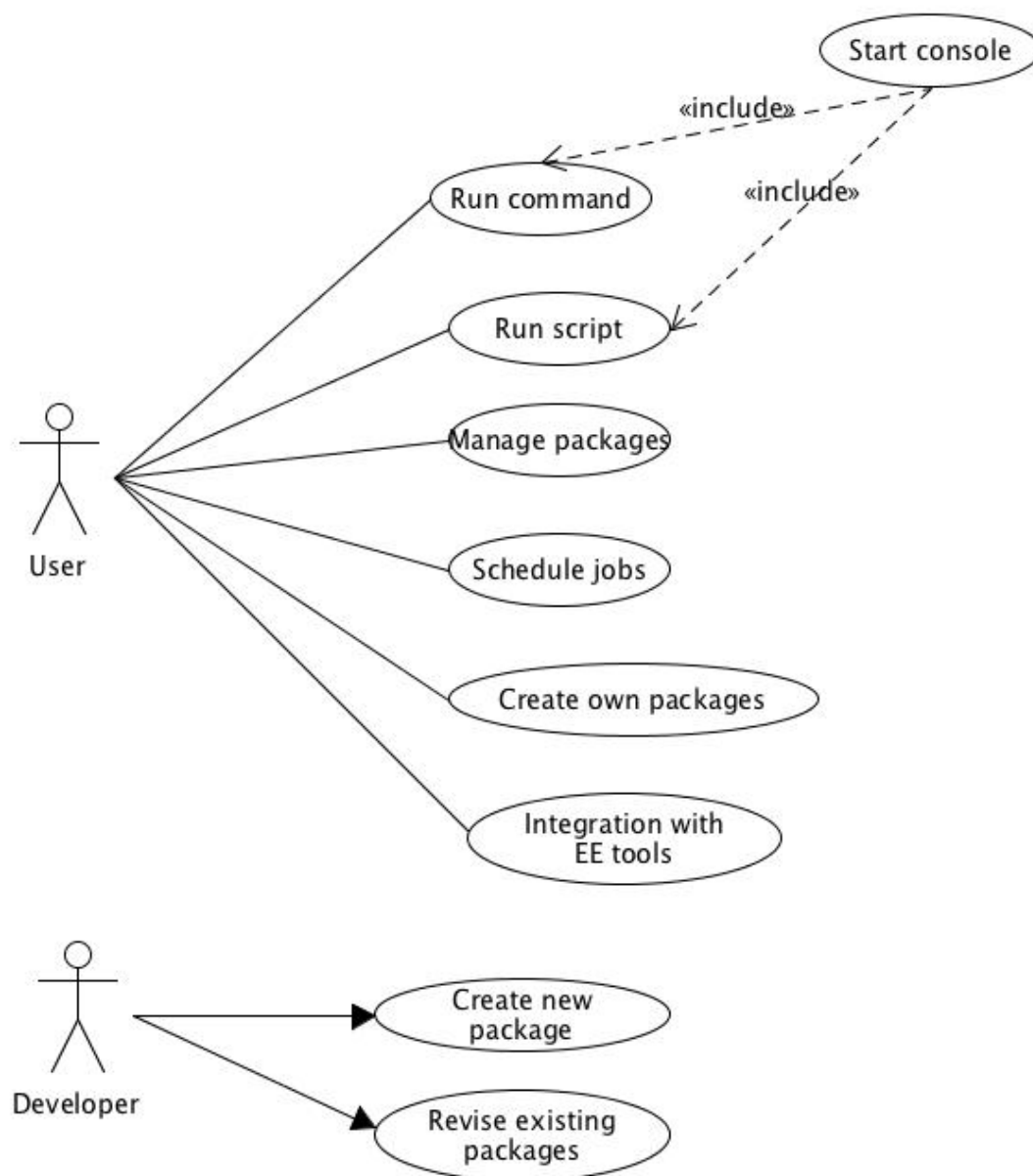
abeceda jazyka, popis ake pismena-slova rozpoznava, ake su vyhradene slova jazyka  
a bla bla Doplit daco o co sa bude dat opret v navrhú a arch.

### **6.4 Procedúry a algoritmy**

procedúra - konečná postupnosť inštrukcií, ktorá sa dá vykonať mechanicky. Doplit  
daco o co sa bude dat opret v navrhú a arch.

# 7 Návrh

## 7.1 Use case



Obrázok 6: Use case pre navrhovanú aplikáciu

### 7.1.1 Popis use casov

<b>Use case</b>	Run command
<b>Podmienky</b>	Shell aplikácia musí byť spustená
<b>Popis</b>	Používateľ zadá validný príkaz, následne získa výstup pre zadaný príkaz.

<b>Use case</b>	Run script
<b>Podmienky</b>	Shell aplikácia musí byť spustená a skript správne napísaný.
<b>Popis</b>	Vykonajú sa všetky príkazy tak ako sú popísané v p

<b>Use case</b>	Manage packages
<b>Podmienky</b>	Shell aplikácia musí byť spustená.
<b>Popis</b>	Používateľ bude schopný nahráť, zmazať, nahradiť vybraný package.

<b>Use case</b>	Schedule job
<b>Podmienky</b>	
<b>Popis</b>	cell8

<b>Use case</b>	Reťaziť príkazy
<b>Podmienky</b>	cell5
<b>Popis</b>	cell8

Doplním chýbajúce.

## 7.2 Prvotný nástrel

## 7.3 Oddelenie štruktúry



## 8 Architektúra aplikácie

Ako sme ukázali existuje veľké množstvo skriptovacích jazykov či, ktoré dokážu efektívne automatizovať dennodennú prácu avšak majú jeden spoločný nedostatok - nie sú multiplatformové. Táto vlastnosť môže byť pre niekoho nepodstatná, no pri veľkých projektoch kde sa mŕňa množstvo prostriedkov na automatizáciu to až tak zanedbateľný fakt nie je. Stačí si len predstaviť koľko času zabere tvorba automatizovaných skriptov pre jednu platformu a pripočítať rovnaké množstvo času pre každú ďalšiu. Niektorí by mohli namietat, že pre ďalšie platformy to len trochu času nezabere nakoľko logika skriptov je už definovaná. Tu treba brať ohľad na to, že nie každý jazyk poskytuje programátorovi rovnaké API a teda treba rátať s možnosťou, že niekde bude potrebné doimplementovať veci chýbajúce v jazyku. Preto vyššie spomenuté dôvody sme sa rozhodli pre vytvorenie nového jazyka, ktorý by bol jednoducho rozšíriteľný, manažovateľný, ľahko píšateľný a platformovo nezávislý.

### 8.1 Java

Je vyvíjaný spoločnosťou Oracle. Jeho syntax vychádza z jazykov C a C++. Zdrojové programy sa nekompilujú do strojového kódu, ale do medzistupňa, tzv. „byte-code“, ktorý nie je závislý od konkrétnej platformy. Táto vlastnosť Javy nám veľmi vyhovuje pre dosiahnutie cieľa platformovej nezávislosti. Ďalším veľmi podstatným faktom je, že v Jave programuje veľké množstvo developerov, tým pádom majú open source projekty veľkú šancu, že si ich komunita developerov osvojí a prispeje k ich postupnému zlepšovaniu.

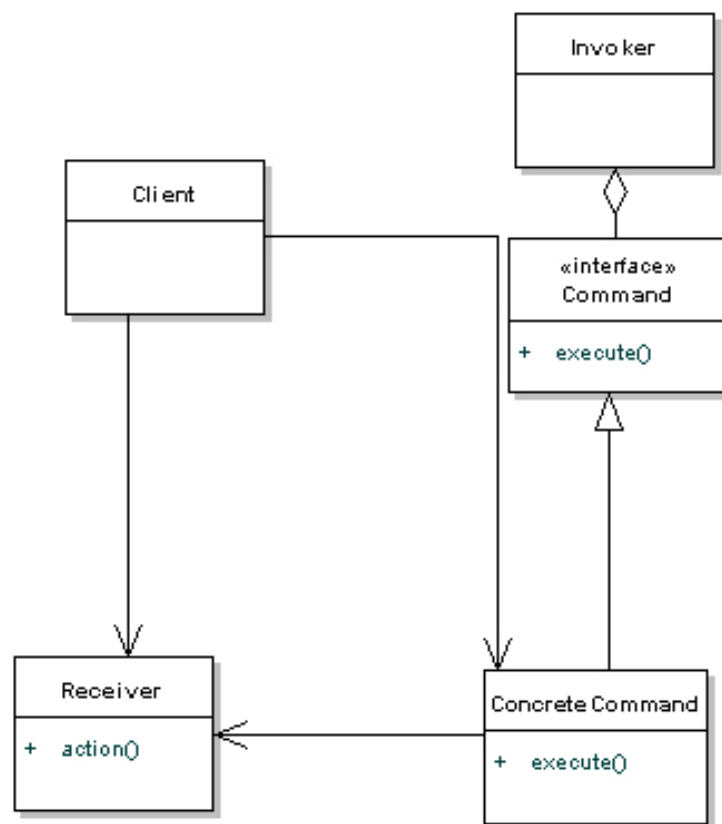
### 8.2 Použite navrhové vzory

Aby sme zaručili rozšíriteľnosť, manažovateľnosť a ďalšie zásady dobrého softvéru bolo potrebné zvoliť vhodnú architektúru, ktorú popisujú použité návrhové vzory.

#### 8.2.1 Command - príkaz

Command pattern je známy behaviorálny návrhový vzor, používa sa najmä na menovanie algoritmov, vzťahov a zodpovedností medzi objektami. Cieľom vzoru je zapuzdriť požiadavku(request) ako objekt tým pádom parametrizovať klienta s rôznymi požiadavkami a zabezpečiť operáciu spať.

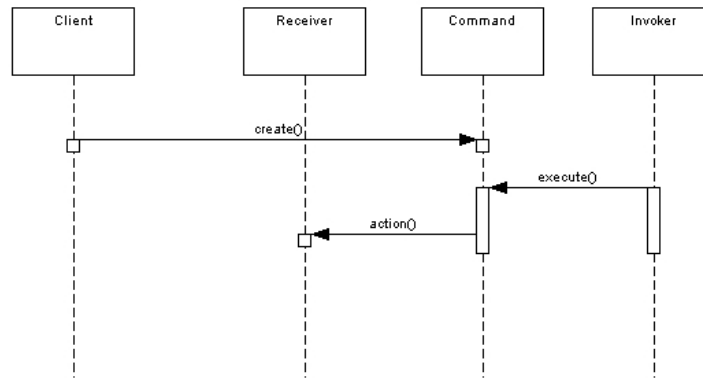
Command vzor deklaruje interface pre všetky budúce commandy a zároveň execute() metódu, ktorú s vypýta Receiver commandu aby splnil požadovanú operáciu. Receiver je objekt, ktorý vie ako požadovanú operáciu splniť. Invoker pozná command a pomocou implementovanej execute() metódy dokáže vyvolať požadovanú operáciu. Klient potrebuje



Obrázok 7: Class diagram Command návrhového vzoru

implementovať ConcreteCommand a nastaviť Receiver pre command. ConcreteCommand definuje spojenie medzi action a receiver. Keď Invoker zavolá execute() metódu na ConcreteCommand spustí tým jednu alebo viac akcií, ktoré budú bežať pomocou Receiveru.

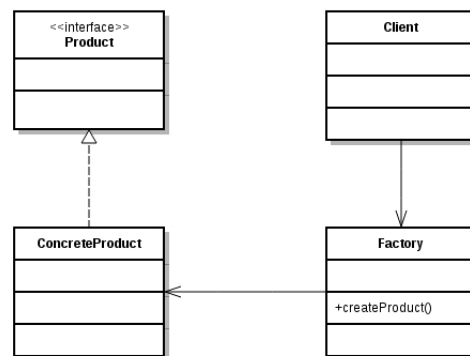
Pre lepšie pochopenie je proces zobrazený aj na sekvenčnom diagrame.



Obrázok 8: Sekvenčný diagram Command návrhového vzoru

### 8.2.2 Factory - továreň

Factory návrhový vzor patrí do sekcie vytváracích vzorov, pomocou tohoto vzoru budeme schopný vytvárať objekty bez toho aby sme prezradili logiku ich vytvárania klientovi. Diagram návrhového vzoru je možné vidieť na nasledujúcom obrázku.



Obrázok 9: Class diagram Factory návrhového vzoru

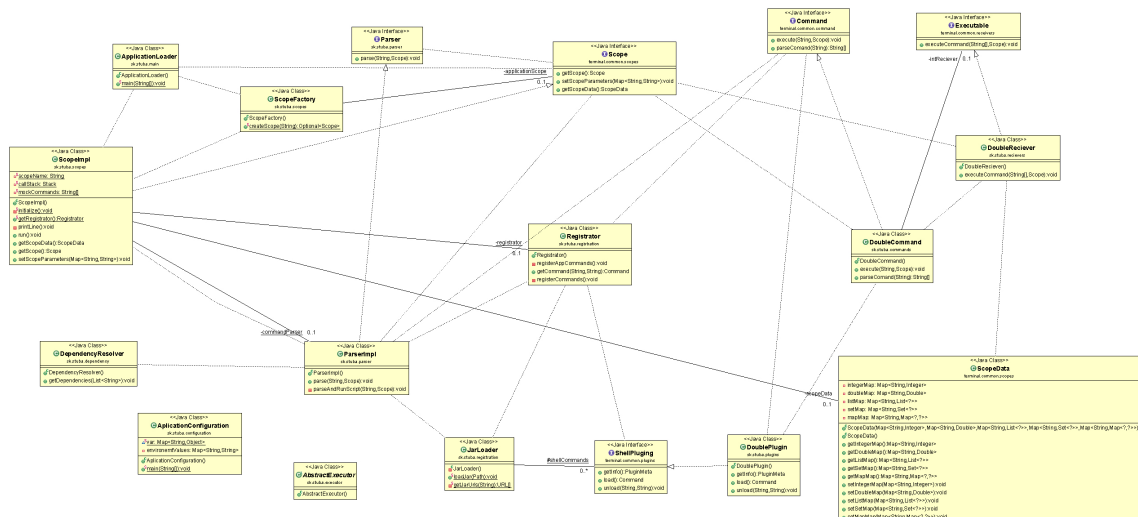
### 8.2.3 Interpreter

možno použiť

## 8.3 Komponenty aplikácie

Ako prvé bolo treba zistiť z akých komponentov sa bude aplikácia skladať. Bolo treba zamyslieť sa čo a ako to chceme dosiahnuť. v prvom návrhu sme identifikovali nasledovné komponenty. Rozhodli sme sa, že vytvoríme plugin systém kvôli tomu, čo najdem a napíšem do analyzy.

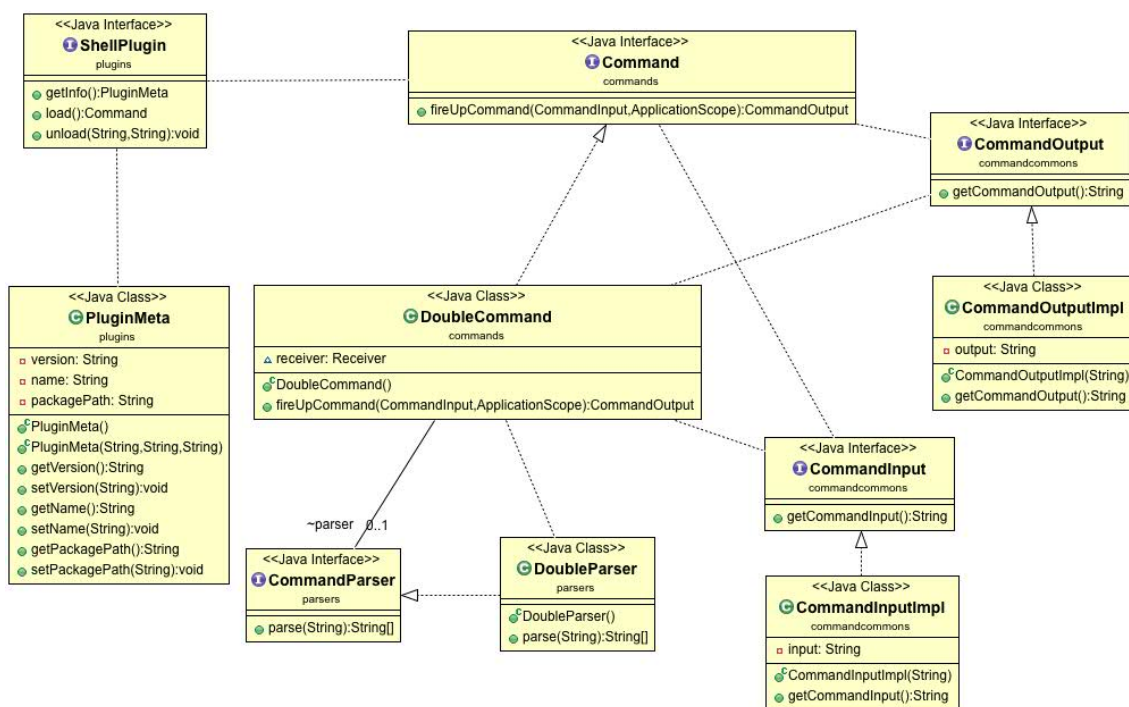
- Parser - vstupov aj výstupov
- Loader jar súborov
- Sťahovač dependencií - jarka ktoré momentálne produkt neobsahuje napr. cusotm riešenia
- Scoping



Obrázok 10: Prvé funkčné riešenie

Z nasledovného class diagramu nebolo na prvý pohľad zreteľne viditeľné aké komponenty v programe existujú preto bolo potrebné zamyslieť sa ako by sa dali tieto časti rozumne rodeliť. Z prvotného návrhu sme vytiahli plugin. Pre implementáciu pluginu sme sa rozhodli použiť architektúru command patternu. Class diagram implementácie je viditeľný na nasledovnom obrázku.

Popisat scoping Vytvaranie commandov



Obrázok 11: Class diagram pluginu

## 9 Zhodnotenie výsledkov

Zatiaľ sa toho nespravilo hodne ale verím, že sa to tu cele zaplní.

# Záver

Cieľom práce bolo zanalyzovať populárne konzolové rozhrania rovnako aj skriptovacie jazyky, ktoré sú často využívané pri administrácii počítačových systémov. Taktiež bolo treba nájsť jednotlivé výhody ako aj nedostatky jednotlivých riešení, zhodnotiť ich a nájsť medzi nimi rozumný prienik, ktorý bolo treba dostať do použiteľnej podoby. Kládli sme dôraz hlavne na to aby naše riešenie bolo čím najlepšie upraviteľné aby mohlo vyhovieť požiadavkam rôznych používateľov.

# Prílohy

A	CD s aplikáciou . . . . .	II
B	Návod na spustenie a používanie aplikácie . . . . .	III

# A CD s aplikáciou



## B Návod na spustenie a používanie aplikácie

Ako spustiť a používať app.