**UNIVERZITA KOMENSKÉHO V BRATISLAVE**

**FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY**

**Framework na vytváranie testových úloh v Pythone**

Bakalárska práca

**2017 Ondrej Husár**

**UNIVERZITA KOMENSKÉHO V BRATISLAVE**

**FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY**

**Framework na vytváranie testových úloh v Pythone**

Bakalárska práca

Študijný program: Aplikovaná informatika

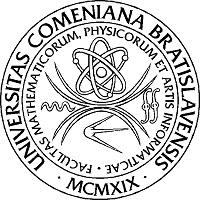
Študijný odbor: 9.2.9. aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: RNDr. Andrej Blaho, PhD.

**Bratislava 2017 Ondrej Husár**

6263106



Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky



**PRIHLÁŠKA NA ZÁVEREČNÚ PRÁCU**

|  |  |  |
| --- | --- | --- |
| **Meno a priezvisko študenta:** | Ondrej Husár |  |
| **Študijný program:** | aplikovaná informatika (Jednoodborové štúdium, bakalársky |  |
| **Študijný odbor:** | I. st., denná forma) |  |
| 9.2.9. aplikovaná informatika |  |
| **Typ záverečnej práce:** | Bakalárska |  |
| **Jazyk záverečnej práce:** | Slovenský |  |
| **Sekundárny jazyk:** | Anglický |  |

**Názov:** Framework na vytváranie testových úloh v Pythone

**Cieľ:** Analyzovať množinu testových úloh, ktoré sa používali v minulých školských rokoch a na tomto základe navrhnúť pythonovské moduly, ktoré umožnia efektívne generovať istú skupinu testov pre úvodný kurz programovania.

**Vedúci:** RNDr. Andrej Blaho, PhD.

**Katedra:** FMFI.KAI - Katedra aplikovanej informatiky

**Vedúci katedry:** prof. Ing. Igor Farkaš, Dr.

**Dátum schválenia:** 25.10.2016



podpis študenta

**Čestné vyhlásenie**

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím uvedených zdrojov.

V Bratislave .................................

**Poďakovanie**

Ďakujem svojmu školiteľovi Andrejovi Blahovi za poskytnutú pomoc, dôveru a ochotu poradiť počas celého priebehu tvorby mojej bakalárskej práce.

**Abstrakt**

Motiváciou tejto práce bola potreba vyvinúť pomocné prostriedky na efektívnejšiu tvorbu testových úloh. Zameriava sa na analýzu aktuálneho procesu testovania zadaní na akademickej pôde a najmä na existujúce nedostatky.

Okrem zhodnotenia daných nedostatkov prináša viacero riešení. Cieľom práce je vyvinúť nástroj, ktorý napomôže jednoduchej tvorbe testových modulov k úlohám pre študentov. Zároveň sa snaží ponúknuť jedinečné a inovatívne spôsoby testovania prvkov, ktoré sú v súčasnosti netestovateľné. Sú nimi rôzne dátové štruktúry, ale aj testovanie grafických komponentov. V neposlednom rade práca prináša intuitívne nástroje na ponúknutie štruktúrovaného a prehľadného výstupu z testovania.

**Kľúčové slová:** *framework, testovanie, PEF*

**Abstract**

The motivation of this bachelor’s thesis was the need to develop ancillary means for creating tests for programming tasks. It focuses on analyzing current process of assignment testing on academic grounds and on existing shortcomings. In addition to assessing these deficiencies, it provides several solutions. The aim of this thesis is to develop a tool that will enable simple creation of test modules for tasks given to students. At the same time, it tries to offer unique and innovative ways to tests parts of curriculum, that are currently not being tested. In particular, various data structures, as well as testing of graphic components. Finally, the work brings intuitive tools that offer users to provide structured and clear output from tests.

**Keywords:** *framework, testing, PEF*

Obsah

[Úvod 10](#_Toc483256490)

[1 Možnosti hodnotenia programátorských úloh 12](#_Toc483256491)

[1.1 Portály určené na samoštúdium 12](#_Toc483256492)

[1.1.1 Portály založené na komunite 12](#_Toc483256493)

[1.1.2 Portály vyvíjané jednotlivcom, alebo malým tímom 12](#_Toc483256494)

[1.1.3 Zhodnotenie 13](#_Toc483256495)

[1.2 Testovač využívaný na akademickej pôde 13](#_Toc483256496)

[1.3 Systémy agregujúce viacero funkcií 14](#_Toc483256497)

[1.3.1 Cieľ takýchto systémov 14](#_Toc483256498)

[1.3.2 Súčasti takýchto systémov 15](#_Toc483256499)

[1.3.3 Existujúce systémy 16](#_Toc483256500)

[2 Nástroje na tvorbu testových modulov 19](#_Toc483256501)

[2.1 Python Unittest Framework 19](#_Toc483256502)

[2.2 PEP8 19](#_Toc483256503)

[3 Nedostatky dostupných testovacích frameworkov 21](#_Toc483256504)

[3.1 Testovanie štruktúr 21](#_Toc483256505)

[3.2 Kontrola programov 21](#_Toc483256506)

[3.3 Nastavenie výstupu 22](#_Toc483256507)

[4 Návrh 23](#_Toc483256508)

[4.1 Časti frameworku 24](#_Toc483256509)

[4.1.1 Kontrola modulu 24](#_Toc483256510)

[4.1.2 TestCase API 26](#_Toc483256511)

[4.1.3 Testovanie grafických komponentov 30](#_Toc483256512)

[4.1.4 Výstup pre študentov 36](#_Toc483256513)

[5 Implementácia 39](#_Toc483256514)

[5.1 Použité technológie 39](#_Toc483256515)

[5.1.1 Unittest 39](#_Toc483256516)

[5.1.2 PEP8 40](#_Toc483256517)

[5.1.3 Python Imaging Library 40](#_Toc483256518)

[5.1.4 SSIM 41](#_Toc483256519)

[6 Výsledky 42](#_Toc483256520)

[6.1 Používateľská príručka 42](#_Toc483256521)

[6.2 Existujúce úlohy 42](#_Toc483256522)

[6.3 Feedback od študentov 43](#_Toc483256523)

[6.3.1 Všeobecný názor 43](#_Toc483256524)

[6.3.2 Práca s testovačom 44](#_Toc483256525)

[6.3.3 Názor na PEF 44](#_Toc483256526)

[7 Možnosti rozšírenia 46](#_Toc483256527)

[8 Záver 47](#_Toc483256528)

[9 Použitá literatúra a prílohy 48](#_Toc483256529)

[9.1 Použitá literatúra 48](#_Toc483256530)

[9.2 Prílohy 48](#_Toc483256531)

# Úvod

Jedným z dôležitých prvkov vyučovania programovania v začiatkoch štúdia je samostatné riešenie zadaní a domácich úloh. Pre vyučujúceho táto potreba prináša viacero výziev. Študent by sa mal pri riešení a súbežne pri odovzdávaní dozvedieť informáciu o správnosti jeho riešenia. Dosiahnuť to konzistentne a s efektívnym využitím času je náročná úloha, ktorú sa prakticky nedá vykonávať manuálne.

Jeden z hlavných dôvodov je ten, že programátorské úlohy, jedno v akom jazyku, zvyknú mať veľkú mieru voľnosti v spôsobe implementovania riešenia. Tým pádom, jednoduché porovnávanie odovzdaných riešení so vzorovým riešením neprodukuje zmysluplný výsledok použiteľný na hodnotenie študentov. (1) Je možné, že odovzdaný program spĺňa požadované aspekty zadania, ale využíva markantne iný spôsob a postup pri jeho riešení. Preto iba manuálna kontrola od skúseného lektora, spojená s prihliadnutím do vzorového riešenia môže viesť ku spravodlivému hodnoteniu.

Manuálne hodnotiť všetky riešenia bohužiaľ nie je fyzicky možné. Študenti potrebujú priebežné hodnotenie čo najskôr a čas programovania si určujú sami. Z toho vyplýva požiadavka na kvalitné automatické testovanie a vyhodnocovanie odovzdaných riešení. Pripravovať takéto testovanie pre každú preverovanú tému je ale tiež časovo náročné. Z istej miery je to spôsobené nedostačujúcimi, respektíve neexistujúcimi podpornými nástrojmi na tvorbu takýchto testových sád.

Preto by bolo vhodné navrhnúť podporný nástroj, vďaka ktorému sa čo najviac zefektívni generovanie testov pre skupiny úloh. Taktiež by bolo ideálne, aby sa vyvinuli, resp. použili ďalšie podporné techniky na hodnotenie odovzdaných programov študentov, ktoré by zvýšili kvalitu odozvy od učiteľa. Napríklad, aby pri nesprávnosti svojho riešenia vedel študent zistiť informáciu, kde presne spravil chyby.

Tiež treba uvážiť, či sa vďaka takémuto frameworku, neukážu aj ďalšie možnosti testovania problémov, ktoré súčasný systém neumožňuje. Napríklad súčasný systém funguje len pre testovanie textových výstupov (konzolový režim) a nezvláda testovať grafické výstupy.

Aktuálne riešenia sa taktiež nezaoberajú syntaktickou formou kódu, dodržiavaním konvencií a inými, nie funkčnými požiadavkami. Neposkytujú dostatočne detailný feedback pre študentov, ktorí sa prvý krát stretávajú s chybami vo svojich programoch.

Zároveň sa dá zamerať na fakt, že väčšia skupina študentov rieši rovnaké zadanie. Treba sa zamyslieť nad spôsobmi, ako by sa dali analyzovať podobnosti logických postupov a nájsť spôsob poskytnutia kvalitnej odozvy od profesora.

Táto bakalárska práca sa bude snažiť analyzovať aktuálne používané riešenia pri výučbe programovania. Bude hľadať ich výhody, nedostatky a pokúsi sa navrhnúť riešenie, ktoré umožní efektívnejšie vytvárať domáce zadania. Pokúsi sa tak napomôcť kvalitnejšej výučbe, najmä behu kurzov pre začínajúcich programátorov na FMFI.

# Možnosti hodnotenia programátorských úloh

## Portály určené na samoštúdium

Jednou z možností pre ľudí, ktorý sa chcú naučiť programovať sú portály a webové stránky, ktoré sú určené priamo na tento účel. Ľudia ich zvyčajne využívajú ako doplnok ku štúdiu, ale aj ako hlavný zdroj zadaní z vlastného záujmu o programovanie. Portály, o ktorých táto práca hovorí, sú založené práve na princípe programátorských úloh so sadou testov. Pozrieme sa na to, aké riešenia používajú pri generovaní cvičení a hodnotení ich správnosti.

Z môjho pohľadu, sa dajú rozdeliť do dvoch kategórií. Portály založené na komunite a portály vyvíjané jednotlivcom, resp. tímom ľudí.

### Portály založené na komunite

Sila a úspech takýchto portálov je ukrytá v programátorskej komunite, ktorá ich navštevuje. Portál sám o sebe poskytuje funkcionalitu ktorá obsahuje nasledovné nástroje:

* Možnosť pridať programátorskú úlohu.
* Možnosť pridať sadu testov, ktoré určujú správnosť riešení.
* Možnosť riešenia úloh na stránke otestovať.
* Možnosť diskutovať o riešeniach.

Jedným z takýchto portálov je napríklad CodeWars ([www.codewars.com](http://www.codewars.com)).

### Portály vyvíjané jednotlivcom, alebo malým tímom

Princíp týchto portálov je o čosi odlišný a značne jednoduchší. S pravidla obsahujú sadu dopredu definovaných úloh, testov a vzorových riešení. Vo svojej podstate sú niečo, ako náhrada domácich úloh.

Príkladom môže byť Practice Python (<http://www.practicepython.org/>).

### Zhodnotenie

Aj keď sa obe spomenuté riešenia zameriavajú na zjavne podobný problém, ich riešenia neprinášajú inšpiráciu pre výučbu programovania na univerzite. Pri prvom riešení nie je schopný jednotlivec nahradiť komunitu a pri druhom riešení je problémov viac. Po prvé, také niečo sú momentálne vyučujúci nútení robiť sami. Po druhé, nemôžu študentom pred vypracovaním poskytnúť vzorové riešenia.

To necháva stále otvorené otázky, ako zefektívniť vytváranie zadaní a testov a ako poskytnúť pri odovzdávaní čo najlepšiu odozvu.

## Testovač využívaný na akademickej pôde

Existujúce riešenia je vhodné hľadať na akademickej pôde. Predmety, ktoré sa zaoberajú výučbou programovania musia tento problém prirodzene riešiť. Pri výučbe na našej fakulte sa profesori často uchyľujú k manuálnemu kontrolovaniu kódu, respektíve, ku kombinácií manuálneho spúšťania sady testov a následnému nahliadnutiu do riešenia. Takýto spôsob opravovania je s pravidla časovo náročný a preto býva často delegovaný na cvičiacich predmetu.

Samozrejme, na fakulte existuje aj ďalší typ hodnotenia. O uľahčenie sa starajú zväčša automatické testy, ktoré sa spúšťajú po odovzdávaní na serveri. Týmto spôsobom sa testujú najmä I/O (input – output) testy, čiže testy ktoré poskytnú dáta na štandardný vstup a očakávajú výsledok zo štandardného výstupu.

Spätná väzba od týchto testov je však veľmi nezhovorčivá. Študentom povie iba informáciu, či majú riešenie správne, alebo nie. Takýto typ testovania sa používa aj na programátorských súťažiach. Ideológia týchto testovacích modulov je ale odlišná od tej našej. Ich primárnym cieľom je:

* Otestovať znalosti a schopnosti študenta.
* Prinútiť študenta, aby sa zamyslel nad dostatočne efektívnym riešením.



Obrázok č. 1 - Príklad výstupu modelového testovača v kurzoch na FMFI

Na obrázku č. 1 môžeme vidieť neúspešné testovanie. Riešenie prešlo cez prvú skupinu testov, ale zlyhalo na druhej. Ako môžete vidieť, nie je poskytnutá informácia o type chyby, o vstupe a ani o očakávanom výstupe.

Hlavne pre programátorov začiatočníkov je táto informácia veľmi nekompletná. Tieto riešenia z uvedených dôvodov nie sú ideálnym nástrojom pre naše potreby.

## Systémy agregujúce viacero funkcií

Na akademickej pôde, vo svete aj u nás, sa pri administrácií kurzov taktiež často využíva e-learningový nástroj LMS (z anglického názvu Learning Management System). Takýchto systémov je samozrejme viac. Existujú voľne dostupné riešenia, komerčné plány a samozrejme aj systémy vyvinuté na špecifické účely vedúcimi kurzov, alebo študentami.

### Cieľ takýchto systémov

Cieľ všetkých LMS systémov je vo svojej podstate jednotný. Integrovať služby potrebné na vedenie kurzov na jedno miesto. Pre čo najjednoduchšiu dostupnosť je v drvivej väčšine prípadov prístupové miesto internetový portál.

Portál má byť prvým miestom kam príde študent, ak hľadá aktuálne informácie o kurze. Taktiež slúži ako miesto odovzdávania zadaní a nahrádza vsúvanie papierov popod dvere kancelárie profesorov. Pri kurzoch, ktoré sa týkajú programovania, sú zadania jednou z hlavných súčastí výučby. Dovolím si tvrdiť, že v programátorských kurzoch je potreba na množstvo úloh väčšia, ako pri iných predmetoch. To je samozrejme náročné na čas profesorov a tak s príchodom e-learningu prirodzene prišla možnosť zjednodušiť proces zadávania a hodnotenia úloh.

### Súčasti takýchto systémov

Aby sa docielil čo najefektívnejší priebeh kurzov programovania, systémy mimo iných funkcií obsahujú aj tieto stavebné časti.

* Sklad úloh

V programovaní sa často stretávame s veľmi kreatívnymi zadaniami. Profesori vynakladajú snahu zatraktívniť úlohy a schovať požadované algoritmy za príbeh. Systém musí zaručiť, aby tieto zadania nezapadli prachom, alebo sa niekde nestratili. Musia byť ľahko dostupné a použiteľné pre ďalšie generácie študentov. Preto LMS systémy obsahujú sklad úloh.

* Testovač

Programátorské zadania majú jednu veľkú výhodu. Na rozdiel od napríklad humanitných vied, kde je práce nutné čítať a posúdiť, programátorské úlohy plnia veľmi konkrétne zadania a to prináša možnosť testovať ich automaticky. Na to samozrejme neexistuje lepšie miesto, ako e-learningová platforma, kde sú dané úlohy zadané.

* Diskusia

Posledná z vitálnych stavebných častí LMS systémov je možnosť diskusie. Študenti musia mať možnosť otázky klásť a profesor odpovedať. Tak isto pri nezhodách, neúspešných riešeniach, alebo nejasných hodnoteniach musí mať profesor možnosť informovať študenta o tom, čo bolo príčinou neúspechu.

### Existujúce systémy

Pri výskume som natrafil na riešenia, ktoré vyvinuli tímy na univerzitách po svete, ale aj na komerčné riešenia. Pre potrebu tejto bakalárskej práce spomeniem príklady z pôdy akademickej.

#### xLX

xLX, alebo eXtreme e-Learning eXperience je nástroj vyvinutý na University of Muenster v Nemecku. Je to integrácia nasledovnej funkcionality:

* mailový list pre daný kurz
* diskusné fórum pre účastníkov kurzu
* študijné materiály
* “pieskovisko”, resp. miesto na skúšanie malých kúskov kódu.
* elektronické odovzdávanie a hodnotenie úloh

Ako si môžete všimnúť, ide o príkladný prípad LMS systému. Naviac ponúka spomínané pieskovisko, čo môže byť ďalší z dôvodov, prečo študent nemusí opustiť systém a napĺňa tak ciel – jednotné miesto pre účasť v kurze.

#### LIST

Long-term Internet Storage of Tasks, alebo skrátene LIST je systém, ktorý navrhol a vytvoril Andrej Jursa v roku 2013 ako bakalársku prácu na Fakulte matematiky, fyziky a informatiky. Systém sa rozširoval ako diplomová práca a prešiel aj mnohými ďalšími rozšíreniami. Pôvodne systém vznikol ako nasledovník LaMSfET-u, ktorý nespĺňal všetky potrebné požiadavky na LMS systém. (2)

Ako vyplýva z anglického názvu LISTU, systém poskytuje priestor na zadávanie programátorských zadaní a ich uchovávanie. V dnešnej podobe je systém rozsiahla webová aplikácia, ktorá ponúka širokú funkcionalitu.

Mimo pridávania zadaní a množstva ďalšej funkcionality umožňuje aj:

* Administrovať viacero kurzy.
* Automaticky testovať riešenia zadaní a vyhodnocovať ich.
* Manuálne editovať body študentov.
* Viesť diskusiu k zadaniam.

Tento systém je veľmi taktiež veľmi dobrým príkladom LMS a aj preto je podobný systému xLx. Pre náš cieľ to prináša inšpiráciu. Čo si ale treba uvedomiť je fakt, že spomínané riešenia sa zaoberajú zefektívnením zadávania, testovania a hodnotenia úloh a aj zlepšením interakcie medzi účastníkmi. Naopak sa ale nezaoberajú zjednodušením vytvárania samotných testov ku zadaniam, čo je primárny cieľ mojej práce.

##### Aktuálne využitie

LIST je v prítomnosti stále v prevádzke. Z tradičných funkcií LMS systému slúži najmä na skladovanie úloh, automatické testovanie a hodnotenie študentov. LIST aktuálne aktívne využíva viacero predmetov z našej fakulty. Napomáha tomu aj fakt, že systém podporuje testovanie úloh vo viacerých programovacích jazykoch. Menovite sú to Java, Go, C++, Haskell a v neposlednom rade Python, ktorého podpora je nevyhnutná pre moju prácu.

##### Možnosti testovania úloh v Pythone v LISTe

LIST podporuje dva typy testovania programov v Pythone. Tak ako testovač, ktorý bol spomenutý v úvode práce, aj LIST podporuje Input/Output testovanie. Čo je ale zaujímavejšie, podporuje taktiež unit testing.

Pri I/O testovaní ide o typ testovania softvéru (v našom prípade programu) takzvanou čiernou skrinkou. Testovací skript vychádza zo zadania alebo požiadaviek na program, bez vedomosti o vnútornej štruktúre. (3) Testovaný systém predstavuje čiernu skrinku, ktorá je určitým spôsobom nakonfigurovaná a prijíma vstupy, v tomto prípade na štandardný vstup. Testovač skúša zadávať rôzne vstupy a sleduje výstupy, ktoré porovnáva z očakávanými výsledkami.

Unit testing je metóda testovania nazývaná aj biela skrinka. Toto testovanie sa zameriava aj na vnútornú logiku a štruktúru programu. Testovaný program si môžeme predstaviť ako priehľadnú skrinku, do ktorej vidíme a poznáme všetky jej súčasti. Takéto testovanie vyžaduje znalosť kódu aplikácie. (3)

Ako som už spomínal, LIST podporuje obe tieto techniky. Z pohľadu profesora je využiteľnejšia technika unit testovanie. Môže skúmať menšie celky programov a ich funkčnosť. Toho výsledkom je dôkladnejšia kontrola programu a možnosť poskytnutia kvalitnejšej spätnej väzby študentovi.

Spätná väzba pri testovaní v LIST-e sa delí na dve časti. Interakcia prebieha prostredníctvom hodnotiacej triedy. Tá musí byť importovaná a musí byť vytvorená jej inštancia v testovacom skripte. V nej sa nasledovne nastavujú body, ktoré vie LIST vyextrahovať. Zároveň, všetok obsah, ktorý počas behu skriptu dostane štandardný výstup a chybový výstup sa po dobehnutí testu zobrazí vo výsledku testu. To ponúka možnosť študentom poskytnúť bližšie informácie o priebehu testovania.

V nasledujúcich častiach budem rozoberať nástroje, ktoré sú schopné pomôcť pri vytváraní testovacích skriptov. Aj keď LIST ponúka jednoduchú spoluprácu a automatické spúšťanie testov, stále ostáva profesorom úloha dané testy pripraviť.

# Nástroje na tvorbu testových modulov

## Python Unittest Framework

Python unittest je najrozšírenejší unit testing modul vstavaný v jazyku Python. Jeho dizajn je postavený na XUnit framework-u navrhnutým Kent Beckom a Erichom Gammom. Rovnaký model je možno vidieť vo viacerých jazykoch vrátane napríklad jazyka C a Java.

Tento framework podporuje tzv. fixtures, test suites a test runner. (2) Jedna fixture je v podstate trieda, ktorá dedí triedu z modulu unittest. Viacero takýchto tried a súborov troví test suite. Následne unittest ponúka spomínaný test runner, ktorý na základe pár pravidiel spúšťa a vyhodnocuje dané testy.

Tak isto tento framework poskytuje základnú sadu pomocných metód, ktoré uľahčujú testovanie kódu. Metódy ktoré vracajú pozitívnu odozvu, že test prešiel a zároveň chybovú hlášku, ak testová metóda zlyhá.

Tu sa ukazuje priestor, ako by mohlo fungovať riešenie časti tejto bakalárskej práce. Stavanie na základoch stabilnej a rokmi otestovanej knižnice. Po istej analýze zadaní pre základnú výučbu programovania by sa taktiež mohli vyvinúť rozširujúce testovacie metódy. Pri dodržaní architektúry tohoto frameworku a inšpirovaní sa konvenciami, ktoré dodržiava sa dá zaručiť sľubný dizajn a integrácia novej funkcionality.

## PEP8

Okrem testovania správnosti kódu existuje aj iný pohľad na odovzdávane programy. Ide o jeden z najviac zanedbávaných prvkov pri výučbe programovania. Je ním pohľad na interné kvality odovzdávaných programov. Samozrejme, pri úvodných kurzoch netreba zaťažovať študentov časovou a pamäťovou náročnosťou do takej miery, ako bude v konečnom dôsledku pravdepodobne potrebné.

No aj napriek tomu je nutné, aby si študenti začali uvedomovať niektoré aspekty kvalitného programátora už na začiatku ich štúdia. Robert C. Martin vo svojej knihe vysvetľuje nasledovné. Funkcionalita, ktorú vytvárate dnes, bude v ďalšej verzií zmenená, ale čitateľnosť vášho kódu bude mať zásadný vplyv na všetky zmeny, ktoré kedy budete vykonávať. Štýl programovania a čitateľnosť vytvára precedens, ktorý bude aj naďalej ovplyvňovať udržateľnosť a rozšíriteľnosť ešte dlho potom, čo sa pôvodný kód zmení k nepoznaniu. Váš štýl a disciplína prežije, aj keď kód nie. (4)

Čo robí tento aspekt ešte dôležitejším, je fakt, že keď sa spýtate Python programátorov čo sa im najviac páči na jazyku Python, často budú spomínať vysokú čitateľnosť. Vysoká čitateľnosť je vskutku zabudovaná v jadre tohoto jazyka, rozoznávajúc známy fakt, že kód je o mnoho častejšie čítaný, ako písaný. (5)

Programátor by si mal vybrať jednoduchú sadu pravidiel na formátovanie a tvorbu svojho kódu a konzistentne sa nimi riadiť. Ak je súčasťou tímu, potom by si daný tím mal vytvoriť pravidlá a dodržovať ich. Pomáha prítomnosť automatizovaného nástroja, ktorý vie dodržiavanie pravidiel kontrolovať. (4)

Študenti väčšinou pri odovzdávaní zadaní pracujú samostatne. Ak sa ale budú chcieť v budúcnosti stať Python programátormi, budú patriť do veľkej komunity, ktorá už isté pravidlá dodržuje. Tieto pravidlá sú spísané na oficiálnej stránke Python-u a sú súčasťou tzv. Python Enhancement Proposals. Tieto PEP opisujú a dokumentujú ako sa jazyk Python vyvíja. Taktiež poskytujú referenčný bod pre programátorov. Celok zaoberajúci sa štýlom kódu je PEP8 a je to istá príručka formátovania a štýlu programovania v jazyku Python. Bola vyvinutá aby pomohla písať programátorom čitateľný kód.

Ďalšia pozitívna správa je tá, že existujú nástroje na kontrolu kódu voči väčšine pravidiel PEP8. Takáto kontrola by mohla naviesť študentov na cestu správneho písania kódu a naučila ich prispôsobovať sa programátorským konvenciám a pravidlám tímu.

# Nedostatky dostupných testovacích frameworkov

Vstavaný modul na testovanie v jazyku Python je veľmi kvalitný a užitočný. Je ale prirodzené, že v špecifických prípadoch sú na mieste ďalšie rozšírenia. Programátorské zadania, ktoré vznikajú na akademickej pôde spadajú do tejto kategórie. Úlohy, ktoré sa týkajú základov programovania, často obsahujú jednoduché dátové štruktúry a pracujú so súbormi. Vo všeobecnosti je rozsah tém pri úlohách počas kurzu pomerne malý.

Tak isto treba prihliadať na fakt, že na programy študentov sú kladené rôzne obmedzenia. Pri množstve študentov, ktorí sú súčasťou kurzov, sa vždy nájdu jedinci, ktorí si chcú svoju prácu uľahčiť. Aj keď je to niečo, čo je v bežnej praxi veľmi cenné, pri výučbe programovania musí vedieť profesor zabezpečiť to, že študent implementuje funkcionalitu vlastným pričinením.

V nie poslednom rade musí mať profesor úplnú kontrolu nad spätnou väzbou. Musí pre neho byť jednoduché ponúknuť zmysluplný a štruktúrovaný výstup. Zároveň je nutnosť mať možnosť neprezradiť veľa a poskytnúť len to, čo uzná za vhodné.

## Testovanie štruktúr

Python Unittest Framework ponúka porovnávanie a manipuláciu so základnými štruktúrami jazyka ako sú napríklad pole, množina, slovník a tak ďalej. V úvodných kurzoch programovania sa študenti stretávajú mimo týchto elementárnych štruktúr aj s ďalšími bežnými dátovými štruktúrami. Môžu to byť napríklad spájaný zoznam, rad, binárny strom, či rôzne iné.

V úvode žiadneho programátorského kurzu sa týmto štruktúram nedá vyhnúť a prirodzene sa vyskytujú aj v množstve úloh. Bohužiaľ, aj napriek tomu, žiadny testovací framework neposkytuje pomocné testovacie metódy pre porovnávanie týchto dátových štruktúr.

## Kontrola programov

Pri kontrole programov, ktoré študenti odovzdávajú, treba analyzovať najmä:

* Nepovolené použitie knižníc.
* Nesprávnu hierarchiu tried.

Prvý bod je jasný. V jazyku Python je množstvo užitočných vstavaných tried, ktoré môžu uľahčiť študentom ich robotu. Prirodzene preto vzniká potreba jednoducho takéto použitie zakázať a vedieť nakonfigurovať povolené pomôcky. Táto požiadavka je veľmi špecifická a netýka sa priamo unit testovania, preto je pochopiteľné, že táto funkcionalita nie je v dostupných frameworkoch podporovaná.

V druhom bode ide o nasledujúcu vec. Pred samotným overením funkcionality programu a tried v ňom definovaných je pri študentoch vždy dobré skontrolovať, či splnili základné podmienky. Tou môže napríklad byť vytvorenie potrebných tried. Profesor potrebuje čo najviac nástrojov, ktorými je schopný analyzovať kód študentov. Potrebuje docieliť dodržanie stanovených pravidiel a nie len správnu funkcionalitu.

## Nastavenie výstupu

Všetky predchádzajúce body strácajú zmysel, ak nie sú v symbióze s dobrou spätnou väzbou. Bez možnosti pekne prezentovať všetky informácie, ktoré boli testy schopné zistiť o programe študenta, by bol náš testovací skript ako človek s obrovským množstvom vedomostí, neschopný ich prezentovať a posunúť ďalej.

Táto potreba je umocnená tým, že toto je jediná časť testovania s ktorou prichádzajú do styku aj naši „koncoví zákazníci“. Tými sú na akademickej pôde jednoznačne študenti. Je na profesorovi, či im chce prácu čo najviac uľahčiť, alebo z nej spraviť väčšiu výzvu. Dobrý framework by mal ale zaručiť to, že sa pri jeho snažení nestane presný opak.

Pri existujúcich riešeniach sa zakladá skôr na konvenciách a jednotnosti testovacieho výstupu. Opäť je to pochopiteľné, pretože sú mierené na širokú programátorskú verejnosť. Táto požiadavka teda znova spadá do špecifík testovania na akademickej pôde.

Všetky tieto nedostatky sa moja práca snaží do iste miery vyriešiť a ponúknuť riešenia, ktoré zjednodušia proces tvorby testov k úlohám pre študentov.

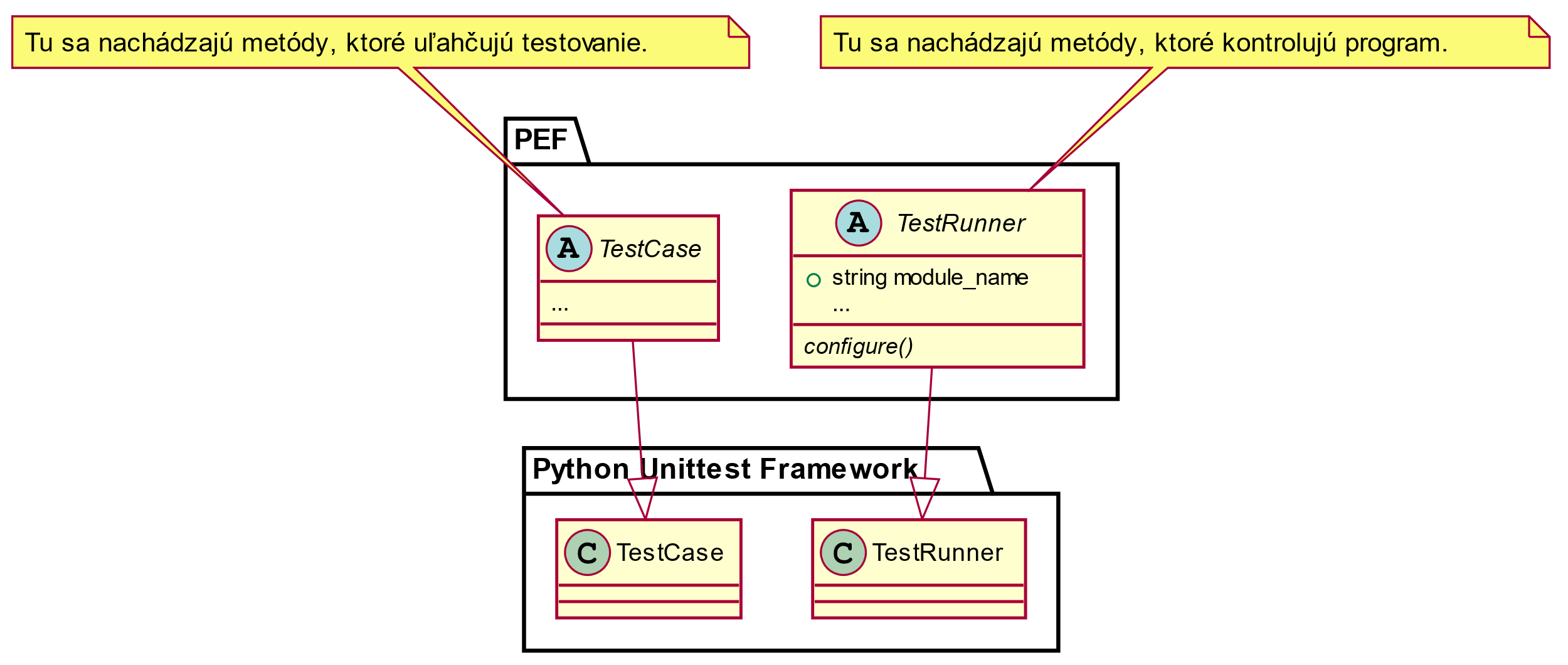
# Návrh

V tejto kapitole spomeniem a rozoberiem stavebné časti mojej práce, ktorou je Programming Exercises Framework, alebo skrátene PEF. Tento framework bude dostupný ako knižnica v jazyku Python.

Podobne ako framework Unittest, aj môj framework bude obsahovať TestCase (voľne preložené ako sada testov). Zdedením tejto triedy sa používateľ vytvorí takzvanú fixture a v tejto triede bude implementovať testy k existujúcemu zadaniu. Testy v zdedenej triede musia dodržovať konvenciu, ktorú predpisuje Python Unittest Framework. Každá metóda, ktorá má byť testom musí začínať prefixom test\_. Tieto metódy budú automaticky rozpoznané ako test a samostatne spúšťané.

Zdedenie triedy TestCase teda ponúka priestor pre tvorbu testov. Na konfiguráciu a nastavenie obmedzení pre celý študentský program sa bude využívať iná trieda. Táto trieda bude dediť triedu TestRunner z PEF. V nej sa bude okrem iného konfigurovať názov testovaného modulu.

Z dôvodu, že implementovaný modul bude musieť zbehnúť a otestovať program študenta po klasickom konzolovom spustení je nutné pozmeniť tradičné správanie knižnice Unittest. Pre používateľa to v praxi znamená, že bude musieť vytvoriť inštanciu triedy Runner z PEF. Táto inštancia bude v sebe obsahovať výsledky testu, ako aj informáciu o získaných bodoch.



Obrázok č. 2 – zjednodušený triedny diagram

Po pochopení tejto základnej architektúry sa môžeme pozrieť hlbšie do funkcionality knižnice PEF a odhaliť zmysel jej architektúry.

## Časti frameworku

Framework je navrhnutý modulárne. Každý modul slúži na jednu konkrétnu vec, ale je možné kategorizovať ich do 4 častí podľa ich účelu.

### Kontrola modulu

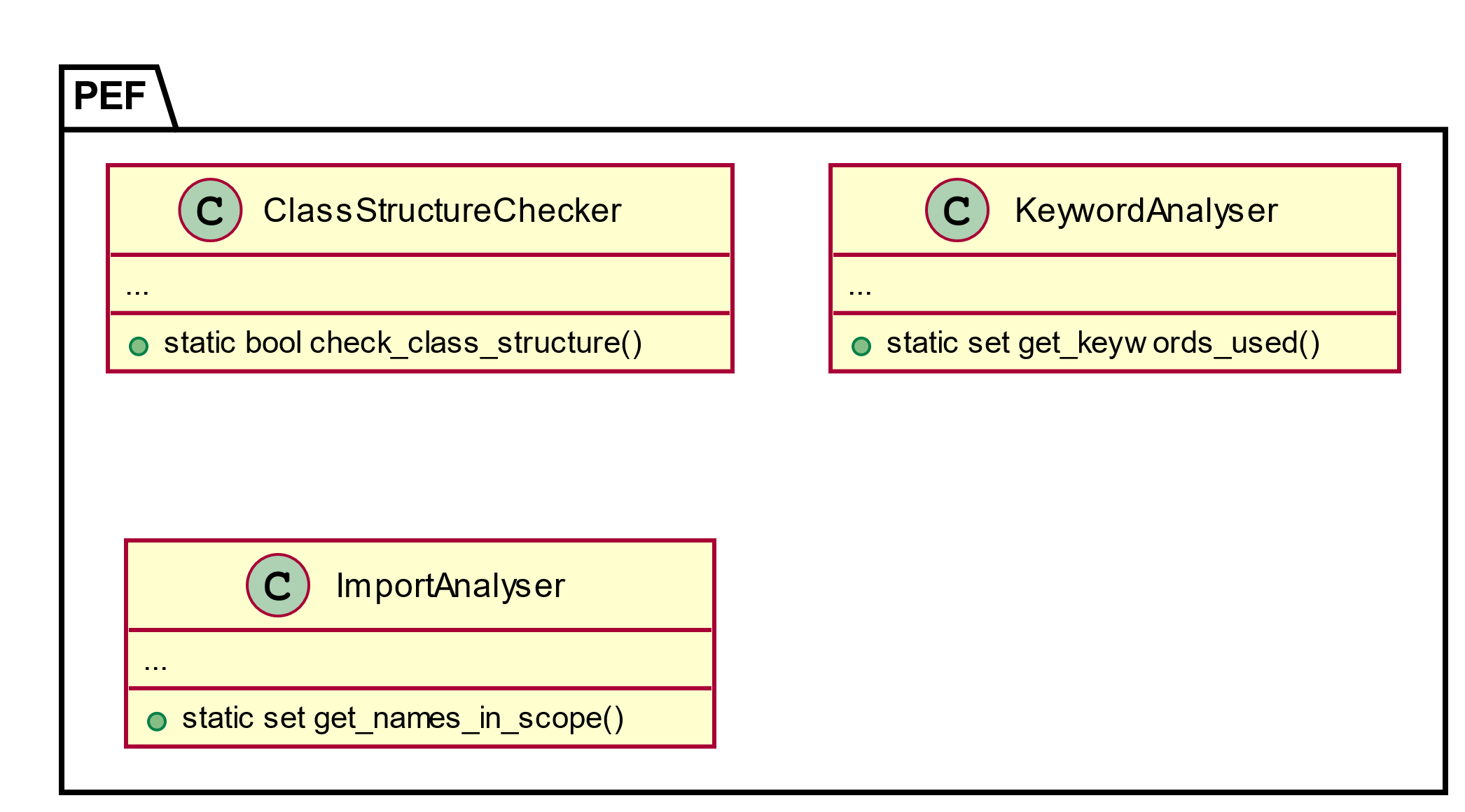
Kontrola modulu je zabezpečovaná a konfigurovaná z triedy, ktorá dedí triedu TestRunner. Konkrétne sa dajú nakonfigurovať nasledovné nastavenia:

1. kontrola štruktúry tried
2. kontrola použitých kľúčových slov
3. kontrola importovaných knižníc
4. kontrola dodržania PEP štandardov

O prvý z bodov sa stará modul ClassStructureChecker. Ten poskytuje jedinú metódu, ktorá dostane ako vstup hierarchiu tried a ich metód. Táto trieda testovaný modul skontroluje a vráti binárnu odpoveď – áno / nie – na otázku, či modul hierarchiu spĺňa.

Ďalej, kontrola použitých kľúčových slov. Tento framework slúži najmä na interakciu s úvodným kurzom programovania. Je preto potrebné, aby mal profesor možnosť obmedziť isté prvky jazyka. Môže tým napríklad vynútiť u študentov iný typ rozmýšľania. Túto funkcionalitu zabezpečuje KeywordAnalyser, ktorý ponúka metódu, ktorá vracia množinu použitých kľúčových slov.

Pri treťom bode sa využíva ImportAnalyser. Ten podobne ako KeywordAnalyser poskytuje metódu, ktorá vracia množinu všetkých importovaných knižníc v module.

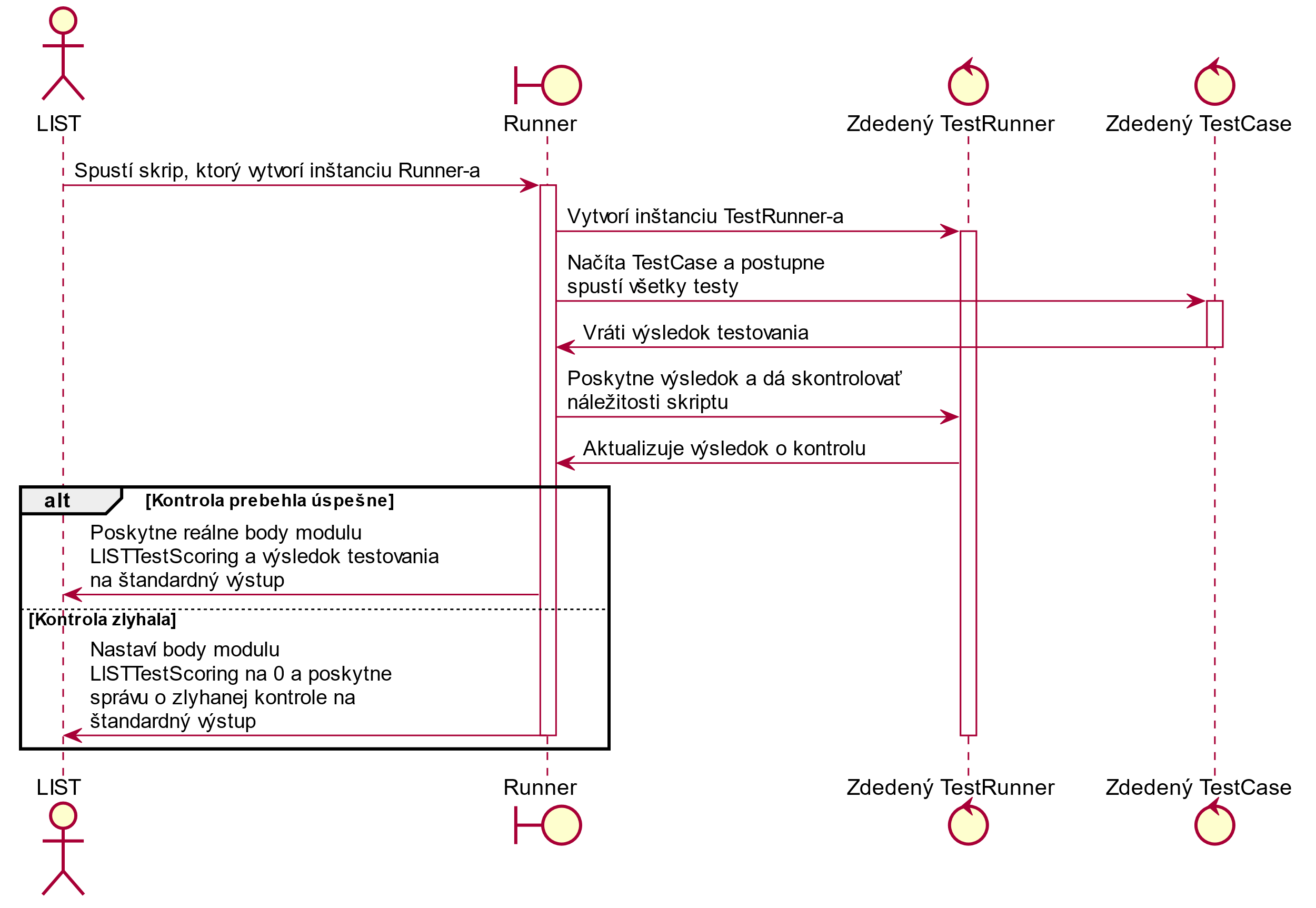


Obrázok č. 3 – Zjednodušený triedny diagram kontrolných tried

Posledný modul, ktorý sa využíva na kontrolu programu je nainštalovaná knižnica pep8 - Python style guide checker. Tento modul sa využíva na kontrolu štýlu testovaného kódu.

Samozrejme, bežný užívateľ frameworku PEF s týmito triedami nepríde do priameho kontaktu. Všetka konfigurácia prebieha v triede, ktorá dedí triedu TestRunner. Tam si užívateľ jednoducho nastaví, čo chce povoliť a čo zakázať. Defaultne sú zakázané všetky importy, povolené všetky kľúčové slová a nie je vyžadovaná žiadna konkrétna štruktúra tried. Možností konfigurácie je samozrejme oveľa viac a taktiež sa dá ľubovoľná z kontrol vypnúť.

Je nutné povedať to, že ak zlyhá ľubovoľná z troch kontrol, výsledok testovania sa nastaví na 0 bodov a študentovi sa neukáže výsledok jednotlivých testov. Toto správanie je navrhnuté z dôvodu, aby si študenti nemohli napríklad skontrolovať, či im program funguje s nepovolenou knižnicou, alebo inak kreatívne obchádzať testovanie. Proces kontroly modulu je zobrazený sekvenčným diagramom na obrázku č. 4.



Obrázok č. 4 – Sekvenčný diagram, ktorý sa zameriava na kontrolu programu

Výnimkou je kontrola dodržiavania PEP štandardov. Pri neúspešnej kontrole študentov framework iba upozorní na ich prehrešky. Pri úspešnej ich na druhú stranu pochváli. Tento prístup má viesť k vytvoreniu povedomia o štýle písania kódu.

### TestCase API

Ďalšou stavebnou súčasťou môjho frameworku sú testovacie metódy, ktoré nadstavujú nad Python Unittest Framework a poskytujú nadštandardnú funkcionalitu. Samozrejme, dostupné sú aj všetky metódy zo vstavaného frameworku. Z dôvodu konzistentnosti sú všetky metódy, ktoré slúžia na použitie užívateľom, pomenované technikou tzv. „CamelCase“. Samotný, voľne preložený názov – ťavia veľkosť písmen – evokuje to, že obrys textu písaného v CamelCase má „ťavie hrby“. Táto technika nie je v programovaní nijak výnimočná a spomínam to iba z dôvodu, že sa to vymyká štandardom programovania v Pythone.

Metódy, ktoré poskytuje PEF, sa dajú ďalej rozdeliť do troch kategórií.

#### Porovnávanie dátových štruktúr

V úvodných kurzoch programovania je skupina základných dátových štruktúr, ktoré musí spoznať každý študent. Úlohy s rozličnými zadaniami často smerujú k využitiu týchto dátových štruktúr. Bolo by dobré vedieť tieto dátové štruktúry porovnávať.

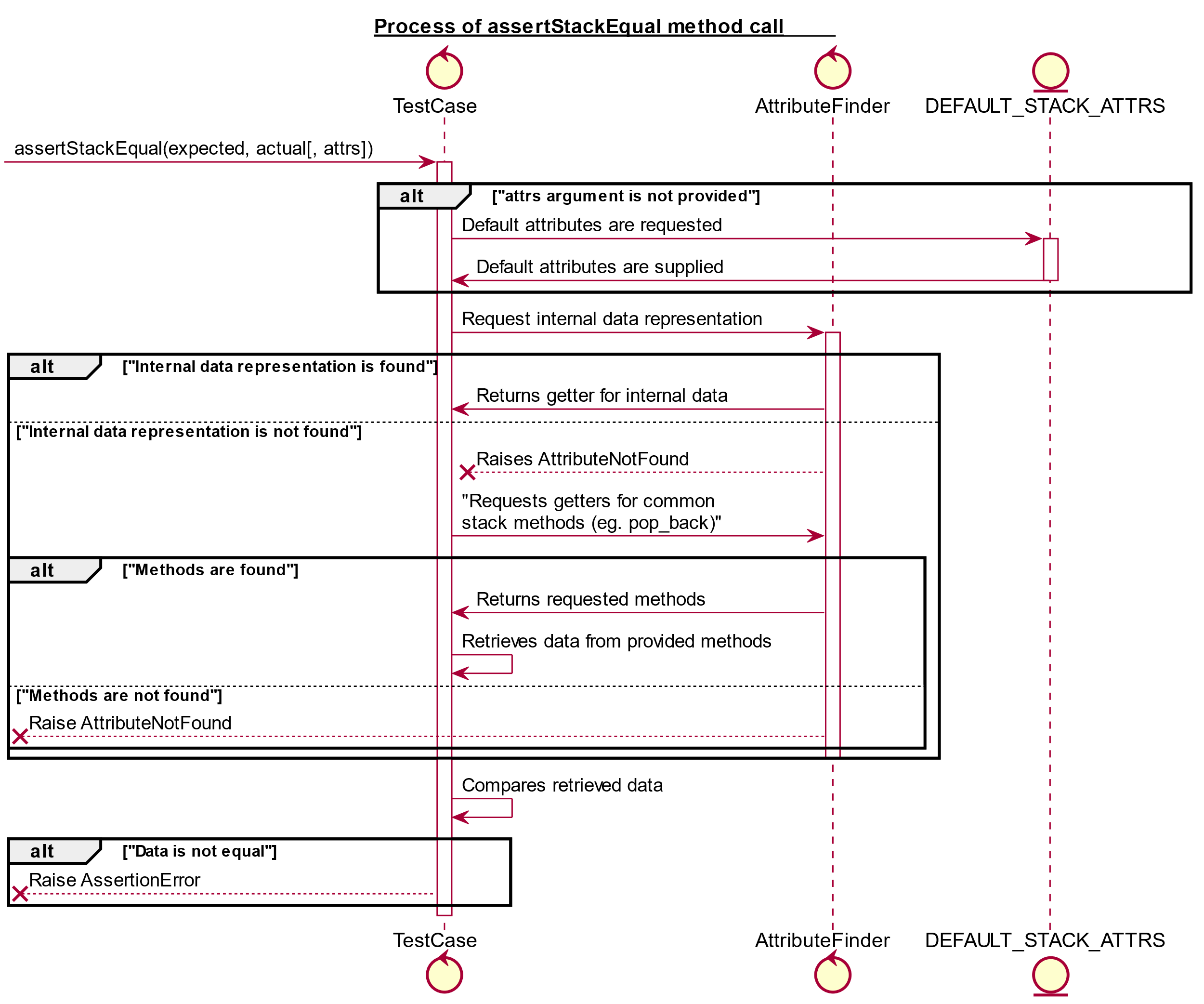
Tu ale prichádzame ku kľúčovému problému takéhoto porovnávania. Dátová štruktúra reprezentovaná študentom implementovanou triedou, môže byť navrhnutá inak, ako trieda zo vzorového riešenia. Porovnávať musíme inštancie dvoch rozličných tried. PEF framework sa snaží tento problém riešiť.

Pri porovnávaní dátových štruktúr pracujem s nasledujúcim predpokladom. Ak študent implementuje svoju verziu dátovej štruktúry, často používa rozličné názvy. Niekto programuje po slovensky, niekto po anglicky. Niekto dodržuje konvencie jazyku Python a privátne premenné nazýva s podčiarkovníkom ako prefixom. Niekto možno používa getery a iný si uchováva dáta len v atribútoch. Isté je ale to, že pri programovaní základných štruktúr všetci musia implementovať vitálne metódy na to, aby zaručili požadovanú funkcionalitu danej triedy.

S porovnávaním štruktúr pomáha najmä modul AttributeFinder, ktorý je schopný vyextrahovať potrebné metódy z poskytnutej inštancie triedy. Požadované metódy sú definované v slovníku. Kľúčmi sú názvy metód, ktoré sú potrebné pre otestovanie danej dátovej štruktúry. Hodnotami sú polia možných názvov daných metód. Nájdené výsledky poskytne AttributeFinder taktiež vo forme slovníku, kde kľúčmi ostávajú názvy a hodnotou sa stáva funkcia, ktorá vracia žiadané dáta.

Treba podotknúť, že nie vždy je potrebné, aby mala inštancia definované všetky atribúty / metódy. Pri snahe vytiahnuť z inštancie dáta, ktoré sa dajú porovnať, framework skúša viacero spôsobov. Napríklad pri porovnávaní zásobníka sa najprv pokúsi vytiahnuť interná reprezentácia a pri neúspechu sa pomocou metód, ktoré by mal zásobník mať, vyextrahujú dáta týmto spôsobom.

Taktiež, pri porovnávaní štruktúr má užívateľ dve možnosti. Možné názvy pre potrebné metódy môže poskytnúť ako argument samotnej porovnávacej metóde, alebo môže nastaviť predvolené možné názvy, ktoré sa použijú, ak argument neposkytne. Táto možnosť je preferovaná a programátor by sa mal uchyľovať k použitiu argumentu len v krajných prípadoch. Proces príkladnej porovnávacej metódy je zobrazený na obrázku č. 5.



Obrázok č. 5 – Proces volania porovnávacej metódy

Zdedením triedy TestCase sa stanú dostupnými metódy na porovnávanie dátových štruktúr, ktoré majú spoločných zopár argumentov a tými sú:

* **expected** – Tento argument očakáva inštanciu, ktorá slúži ako vzor.
* **actual** – Tento argument očakáva inštanciu, ktorá sa bude porovnávať voči vzoru.
* **attrs** – Tento argument je dobrovoľný a obsahuje slovník, v ktorom sú uvedené možné názvy pre potrebné metódy. Upravovať sa ale odporúčajú predvolené hodnoty na príslušnom mieste.

Konkrétne sú dostupné metódy:

* assertTreeEqual
* assertNodeEqual

Slúžia na porovnávanie stromov. Sú schopné porovnať stromy binárne a aj stromy všeobecné. Taktiež majú jeden spoločný argument – print\_tree – ktorý určuje, či sa pri neúspešnom porovnaní vygeneruje do pomocnej správy ASCII reprezentácia porovnávaných stromov. O vygenerovanie ASCII reprezentácie stromu sa stará trieda TreeVisualisation. Pri nastavení hodnoty print\_tree sa do výstupu vygeneruje reprezentácia celého stromu. Ak užívateľ potrebuje väčšiu kontrolu nad takýmto porovnaním, k dispozícií je aj metóda priamo v PEF a to getNodeASCIIComparison. Pri používaní tejto metódy je možné nastaviť maximálnu hĺbku, do akej strom vypisovať. Tak isto treba podotknúť fakt, že výstup najlepšie vyzerá pri konzistentných dĺžkach hodnôt vo vrcholoch. Dá sa teda nastaviť aj maximálna dĺžka hodnoty, ktorá ak bude presiahnutá, príde k jej orezaniu.

Algoritmus funguje na rekurzívnom spájaní štvorcových blokov. Najprv rekurzívne dôjde na spodok stromu, alebo do maximálnej hĺbky. Nasledovne v rodičovi pospája svojich potomkov so svojou hodnotou do štvorcového viacriadkového reťazca. Každý vrchol teda môže predpokladať, že musí pospájať dokopy štvorcové reťazce, ktoré reprezentujú jeho potomkov.

* assertLinkedListEqual
* assertLinkedListNodeEqual
* assertInLinkedList

Tieto metódy slúžia na porovnávanie spájaného zoznamu. Taktiež je možné porovnať spájaný zoznam celý, alebo začať od vrcholu, ktorý užívateľ vyberie za vhodný. Pri tretej spomenutej metóde, assertInLinkedList, nie sú rovnaké argumenty, ako pri predchádzajúcich. Tu je prvý argument hodnota a metóda zisťuje, či sa nachádza v spájanom zozname, ponúknutom v druhom argumente.

* assertStackEqual
* assertInStack
* assertQueueEqual
* assertInQueue

Tieto metódy sú ďalšími z množiny tých, ktoré porovnávajú dátové štruktúry. Ako môžete vidieť, ide o zásobník a rad. Pri porovnávaní sa najprv framework pokúsi nájsť internú reprezentáciu týchto štruktúr, ale taktiež sa spolieha na metódy, ktoré bežne tieto štruktúry majú obsahovať.

Framework ponúka porovnanie dvoch inštancií, ako aj nájdenie prvku v nich, tak ako pri spájanom zozname.

#### Agregované porovnávanie

Pri analýze úloh, ktoré boli zadávané v minulosti na úvodných kurzoch Programovania sa často pri kontrole správnosti vyskytoval istý druh testovania. Tým bolo opakované porovnávanie skupiny metód.

Metód assertEqualResultsForMethods ponúka uľahčenie práve takejto situácie. V argumentoch dostane dve inštancie a ľubovoľný počet názvov metód. Následne postupne porovnáva výsledky daných metód. Samozrejme, je možné poslať metódam aj argumenty. V takomto prípade sa posiela ako argument nie názov metódy, ale pole, ktoré obsahuje na prvej pozícií názov a v zbytku argumenty, s ktorými sa metóda zavolá.

Taktiež je možné nastavovať tzv. vlajku log\_commands, ktorá ak je zapnutá, tak sa uchováva história zavolaných príkazov, ktoré môžu byť zobrazené vo výstupe.

#### Porovnávanie textových súborov

Ďalšia často preberaná látka je manipulovanie so súbormi. Pred tým, ako programátori začínajú interagovať s rôznymi typmi databáz, musia sa naučiť čítať a zapisovať údaje do súboru v požadovanom formáte. Samozrejme, kontrolovať takéto niečo nie je náročné. Napriek tomu, v mojej práci som sa pokúsil uľahčiť aj takéto činnosti zadávateľa.

PEF teda ponúka dve metódy na kontrolu súborov. Jedna z nich porovnáva rovnosť obsahu dvoch súborov a druhá porovnáva obsah súboru, voči reťazcu poskytnutému v argumente:

* assertFileContentEqual
* assertFileContentEqualTo

### Testovanie grafických komponentov

Cieľ mojej bakalárskej práce bol nie len uľahčiť vytváranie testových modulov, ale aj pokúsiť sa nájsť ďalšie možnosti testovania problémov, ktoré súčasný systém neumožňuje.

Jedným z nedostatkov aktuálne dostupných nástrojov je chýbajúca možnosť automaticky testovať grafický výstup. Časť kurzov sa veľmi frekventovane zaoberá grafickými programami, pretože študent si môže ľahko vizualizovať to, čo naprogramoval. Pre profesora to ale znamená, že tiež musí výstup z programu vidieť, aby vedel posúdiť jeho správnosť.

Moja bakalárska práca sa pokúša tento problém adresovať. Pre možnosť otestovania takýchto programov sú potrebné 2 veci. Po prvé musíme vedieť získať výstup z programu bez toho, aby sa musel niekde zobraziť. Po druhé musíme vedieť vyhodnotiť, či je výstup správny.

#### Testovateľný Canvas

Pri programovaní grafických komponentov v kurzoch programovania v jazyku Python sa skoro vždy využíva vstavaná knižnica tkinter. V tejto knižnici sa nachádza taktiež komponent Canvas, ktorý slúži ako plátno na kreslenie. Študent naň môže kresliť čiary, geometrické tvary, alebo aj vykresľovať obrázky.

Moja idea testovateľného Canvasu je nasledovná. Študenti dostanú zadanie, ktoré budú musieť splniť. Pri svojej práci budú využívať triedu Canvas zo vstavanej knižnice tkinter. Následne, po odovzdaní svojho riešenia na server im bude podstrčená trieda Canvas z frameworku PEF. Toto je nutné, pretože knižnica tkinter nie je schopná vytvoriť obrázok bez toho, aby sa vykreslila.

Táto trieda bude naopak využívať inštalovateľnú knižnicu Python Imaging Library. Miesto vykreslenia teda poskytne metódy, ktoré budú navonok identické s metódami z tkinter-u, ale zabezpečia vygenerovanie obrázku a jeho uloženie v súborovom systéme.

Takýto obrázok potom bude možné porovnať napríklad s obrázkom, ktorý týmto spôsobom vygenerujeme zo vzorového riešenia. Ak nebudú identické, môžeme študentom poskytnúť obe reprezentácie a ukázať im tak názorne chybu v ich riešení.

Python Imaging Library samozrejme neponúka rovnakú funkcionalitu akú nájdeme v triede Canvas. Je teda nutné stanoviť študentom isté obmedzenia a množinu funkcionality s ktorou môžu pracovať.

Zároveň, ak sa pozrieme na možnosť integrácie s portálom LIST, musíme poznamenať dva poznatky. Na jednej strane správu pozitívnu. Tou je fakt, že nám systém umožňuje vytvárať počas testovania nové súbory. S ukladaním obrázku problém nenastane. Druhý poznatok je ten, že sa všetky testované súbory po vyhodnotení testu vymažú. Ostane len textový výstup, ktorý sa zobrazí študentovi.

Tento problém sa dá jednoducho vyriešiť tým spôsobom, že prekonvertujeme obrázok do formátu Base64. To je dátový formát zobrazujúci binárne dáta pomocou tlačiteľných ASCII znakov. Týmto pádom môžeme obrázok poskytnúť HTML elementu img, ktorý môže byť pri výstupe vyrenderovaný.

#### Porovnávanie obrázkov

Po vyextrahovaní obrázkov z riešenia študenta a zo vzorového riešenia nám stále ostáva druhá časť problému. Musíme z poskytnutých dvoch obrázkov zistiť, či sú rovnaké. Na tento účel slúži viacero algoritmov. Pri mojej práci som zvažoval dva spôsoby porovnávania – MSE a SSIM.

##### Mean Squared Error (MSE)

MSE je najstaršia a jedna z najjednoduchších rozdielových metód hodnotenia kvality obrazu. V slovenčine sa prekladá aj ako stredná kvadratická odchýlka. V teórii pravdepodobnosti a štatistike je to meradlo štatistickej disperzie. Jednoducho povedané MSE hovorí o tom, ako široko sú rozložené hodnoty v množine.

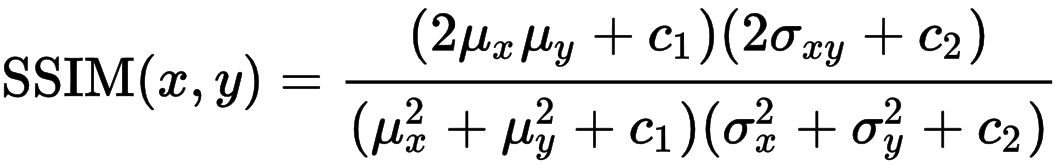
V našom prípade ide o porovnávanie dvojrozmerných obrazov a hodnota MSE sa dá vypočítať pre dva obrázky nasledovnou rovnicou.

Tá sa dá do jazyku Python pretransformovať veľmi jednoducho a vzniká tak veľmi jednoduchá funkcia. MSE naberá hodnoty od 0 vyššie. 0 indikuje identické obrázky.

Veľkou výhodou tejto techniky je jej jednoduchosť. Na druhej strane prináša menšiu presnosť ako porovnávanie výpočtom SSIM.

##### Structural Similarity Index (SSIM)

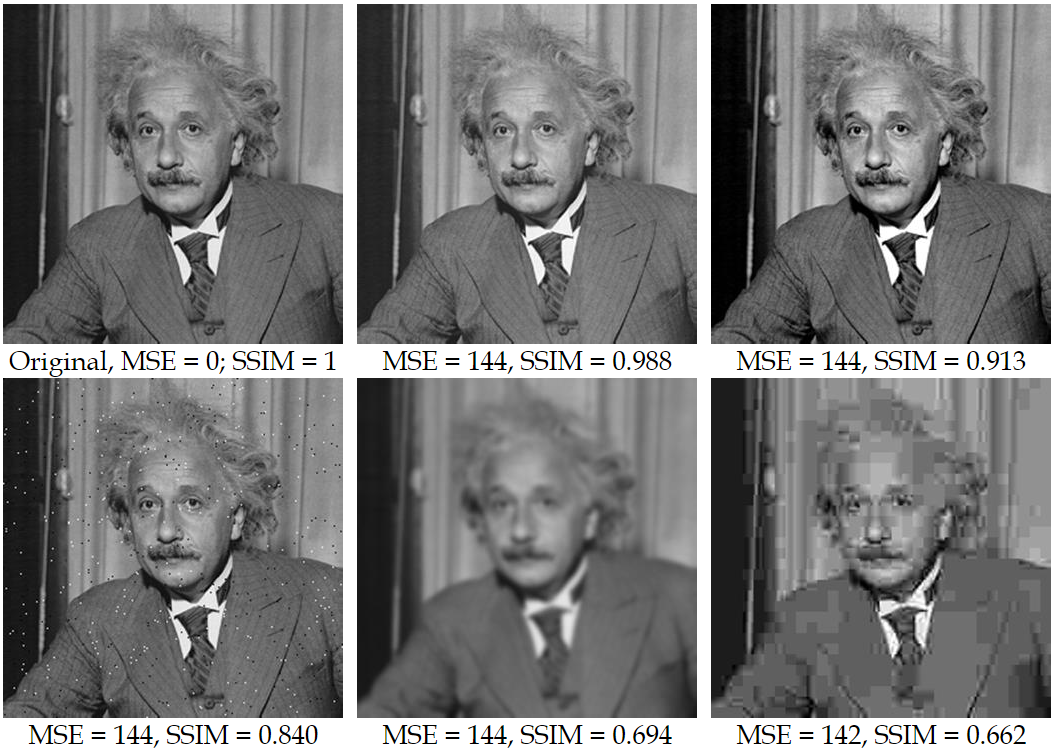
SSIM je metóda na predpovedanie vnímanej kvality digitálneho obrazu. Používa sa na meranie predikcie kvality, ale aj na porovnávanie obrázkov. Na rozdiel od MSE je takýto výpočet zložitejší. Tu je vzorec.



Moja práca sa nesnaží ponoriť vás do tajomstva týchto techník, iba znázorniť prvotný pohľad na tieto techniky. Vzorec vo svojej podstate pracuje s tromi zložkami, ktorými sú jas, kontrast a rozloženie obrazu. Pre porozumenie vzorcu je samozrejme nutné definovať všetky premenné. To pre jednoduchosť robiť nebudeme, ale spomenieme, že sú nimi najmä priemerné hodnoty, odchýlky a spoločné rozptyly.

Pre žiadne zrejmé rozdiely je hodnota SSIM rovná jednej a s narastajúcimi rozdielmi sa výstupná hodnota približuje k nule.

##### Porovnanie

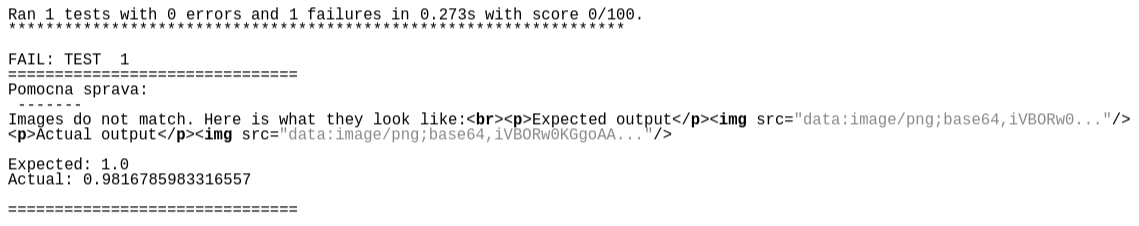


Obrázok č. 6 – MSE a SSIM porovnanie

Na obrázku č. 6 môžeme vidieť vypočítané hodnoty MSE a SSIM. Keďže pri našom porovnávaní pôjde najmä o geometrické tvary a jednoduché obrázky, technika SSIM sa ukazuje ako o mnoho vhodnejšia. Najmä ak sa na obrázku č. 6 pozrieme na tretí a šiesty obrázok, vidíme, že MSE je skoro identické, ale vizuálne sú obrázky veľmi odlišné. Pri nastavovaní odchýlky, ktorú sme ochotný povoliť študentovi by takéto niečo spôsobovalo veľké nepresnosti.

Preto je v našom prípade jasnou voľbou technika porovnávania pomocou výpočtu SSIM. Spojením porovnania obrázkov a toho, že sme schopní vygenerovať obrázok zo vstavanej triedy Canvas sa otvárajú nové možnosti testovania grafických programov.

Vo frameworku sú na porovnávanie obrázkov poskytnuté metódy assertImageEqual a assertImageAlmostEqual. Jediný rozdiel medzi nimi je ten, že pri druhej spomenutej užívateľ nastavuje odchýlku, ktorú je ochotný akceptovať pri porovnaní SSIM. Do výstupu sa pri nesprávnom porovnaní taktiež pridá zobrazenie porovnávaných obrázkov vo forme HTML tagov s obrázkami v Base64 formáte, napríklad taký, aký môžete vidieť na obrázku č. 7. Na ďalšom obrázku, č. 8, môžete vidieť, ako vyzerá príklad výstupu, ktorý sa vyrenderuje.



Obrázok č. 7 – Nevyrenderovaný výstup z testovania



Obrázok č. 8 – Vyrenderovaný výstup z testovania

### Výstup pre študentov

Žiadna zo súčasti frameworku spomenutá v predchádzajúcich častiach by nemala poriadny zmysel, ak nedokážeme funkcionalitu premietnuť aj do odozvy z testovania. PEF sa snaží ponúknuť prehľadný, štruktúrovaný a čo najpodrobnejší popis. Zároveň je samozrejmé, že veľká ťarcha leží na užívateľovi, ktorý framework využíva. Cieľom je mu jeho prácu zjednodušiť.

O manipuláciu s výstupom sa stará trieda MessageBuilder, aj keď užívateľ interaguje stále priamo s triedou, ktorá dedí triedu TestCase. Užívateľ pred testom ideálne nastaví ako prvú vec meno testu. Ak test prebehne úspešne, zobrazí sa iba meno testu a správa, že bolo všetko s testom v poriadku. Pri neúspešnom teste je výstup z testovania ďalej rozdelený na nasledovné časti.

* hlavná správa
* príkazy pred testom
* pomocná správa
* chybová správa

Hlavnú správu tak isto nastavuje užívateľ. Je to správa, ktorá sa zobrazí ako prvá vo výpise. Zadávateľ môže oznámiť, na čo sa test zameriava, poskytnúť informáciu s akými súbormi sa bude pracovať, alebo naopak nepovedať nič navyše. Táto správa ponúka jednoduchý priestor pre vlastnú predstavu toho, čo je nutné spomenúť.

Ďalšia sekcia, ktorou sú príkazy pred testom, je jedna z najdôležitejších pre študentov. Táto sekcia im umožní jednoduchšie hľadať chyby vo svojich programoch. Túto časť nenastavuje užívateľ manuálne. Cez pomocné metódy môže pridávať príkazy s argumentami. Buď po jednom, alebo po skupinách. Taktiež môže príkazy jednoducho vymazať. Zároveň, niektoré metódy umožňujú zapnúť vlajku, ktorá určuje, či sa volané príkazy majú zaznamenávať.

Pomocná správa je miesto, kde sa uchovávajú správy po neúspešnom porovnávaní. Túto časť plní výhradne framework a snaží sa ponúknuť správu o tom, prečo porovnanie zlyhalo. Nachádza sa v ňom klasické porovnanie očakávanej a reálnej hodnoty, textová reprezentácia dátových štruktúr, ale aj napríklad spomínané porovnanie obrázkov.

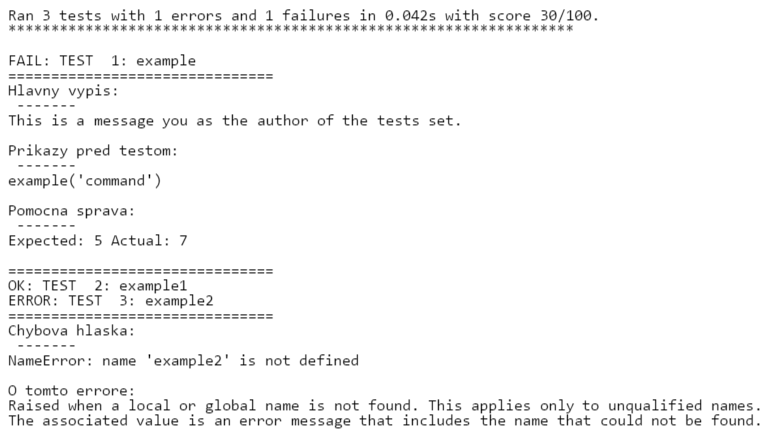
Jedna informácia, ktorú treba taktiež spomenúť, je tá, že pri každom zlyhanom teste je poskytnutá hneď pri názve informácia, či padol na chybe – ERROR – alebo zlyhalo porovnanie – FAIL. Pri padnutí na chybe sa zobrazí posledná vyššie uvedená sekcia. V nej sa zobrazí typ chyby, najrelevantnejšia správa, ktorá hovorí o tom, čo chybu spôsobilo a taktiež vysvetlivka, v ktorej je vysvetlené, v akých situácia sa bežne táto chyba stáva.

Na obrázku č. 9 môžete vidieť príklad triedy, ktorá dedí triedu TestCase a obsahuje 3 testy. Ako môžete vidieť v prvom teste sa nastavuje meno, správa od učiteľa a taktiež sa pridáva príkaz pred testom. V druhom teste sa nastaví iba meno a body, keďže neobsahuje žiadny príkaz, ktorý by vyvolal výnimku, určite test prejde. V treťom teste sa nachádza testovanie funkcie, ktorú modul neobsahuje. To samozrejme z obrázku nemôžeme vedieť, ale slúži to na ukážku toho, ako vyzerá chybová správa vo výstupe.



Obrázok č. 9 – Príklad triedy s testami

Na obrázku č. 10 vidíte, ako môže vyzerať výstup po spustení testovania z obrázku č. 9. Prvý test zlyhal na porovnaní, druhý prešiel a tretí padol na chybe.



Obrázok č. 10 – Výstup z testovania

Každá zo spomenutých sekcií sa vypíše iba ak niečo obsahuje. Ak by ale užívateľ potreboval explicitne pri testoch ľubovoľnú zo sekcií neposkytnúť, prirodzene, má takú možnosť.

Posledné dve veci, ktoré treba spomenúť, sú, že pred výstupom z konkrétnych testov sa zobrazí správa, koľko testov prebehlo celkovo, koľko úspešne a koľko na druhú stranu neprešlo, alebo padlo na chybe. Zároveň sa vypíše údaj, koľko testovanie trvalo. Netreba zabúdať ani na to, že ak neprejde kontrola modulu, jediný feedback, ktorý je poskytnutý je ten o danom zlyhaní.

Framework PEF ponúka nástroje, ktoré pri správnom využívaní môžu uľahčiť prácu ľuďom, ktorí zadania vytvárajú a zároveň poskytnúť konzistentný a nápomocný výstup pre študentov.

# Implementácia

## Použité technológie

V rámci mojej práce som sa snažil čo najmenej využívať externé knižnice, na ktorých by bol môj framework závislí. Do veľkej miery sa mi to podarilo. Snažil som sa držať tzv. single responsibility principle, čo je voľne preložené princíp jednej zodpovednosti. Snažil som sa tak zaistiť, aby každá trieda mala len jednu úlohu z väčšieho celku. Tomuto sa samozrejme vymyká trieda TestCase, v ktorej je agregovaná sila celého frameworku a poskytuje viacero služieb užívateľom.

Moju prácu som implementoval modulárne a pre väčšinu modulov som taktiež písal testy. Percentuálne pokrytie testami by mohlo by mohlo byť výrazne väčšie, ale na konkrétne čísla som nedbal. Testy som písal najmä k netriviálnemu kódu.

### Unittest

O Unitteste frameworku, ktorý je základnou časťou Python programovacieho jazyka som sa už v tejto práci zmienil a základné informácie som poskytol. V tejto časti chcem objasniť, ktoré časti z tohto frameworku vlastne využívam.

Po prvé, treba povedať, že som všetky úpravy do tried z unittest-u implementoval pomocou dedenia a následnej úpravy niektorých metód. Využil som architektúru triedy TestCase a ponechal správanie také, že každá metóda, ktorá začína prefixom „test“, je jeden samostatný test. Každý jednotlivý test sa spúšťa sám a inštancia triedy TestCase po každom behu zaniká. To zaručuje izolovanie od ostatných testov, ale vytvára to potrebu uchovať si potrebné informácie inde. Taktiež, dostupné sú všetky testovacie metódy z pôvodnej triedy TestCase, ale interne je upravený ich výstup, čo si bežný užívateľ nepotrebuje všímať.

Miesto, kde sa priebežné informácie uchovávajú je trieda, ktorú som ako druhú potreboval zdediť a upraviť k mojim potrebám. Jej názov je TestResult a uchovávajú sa v nej priebežné výsledky, čas behu, chybové správy a napríklad aj body. Unittest ponúka viac tried, ktoré môžu slúžiť ako TestResult, ale nijaká nevyhovovala našim potrebám.

Jedna z posledných tried, ktorej správanie bolo treba modifikovať, bola TestRunner. Z dôvodu, že sa trieda TestCase vytvára opakovane, TestRunner je miesto, kde museli prebehnúť celkové kontroly testovaného modulu.

Posledná výzva bolo spustenie testovania. To sa dá vykonať pomocou viacerých príkazov. Ja som si vybral cestu manuálneho pozbierania tried TestCase a TestRunnera a ich následnom spustení.

### PEP8

PEP8 je balíček v Pythone, ktorý slúži ako utilita na kontrolu dodržania konvencií písania Python kódu. Implementovať takéto niečo by bola téma pre samostatnú bakalársku prácu. Naviac by to bolo zbytočné, pretože kvalita tejto knižnice je dostatočná.

Jej architektúra zabezpečuje, že sa dá podľa potreby doplniť ľubovolná kontrola. Zároveň je to veľmi minimalistická knižnica, nevyžaduje externé balíčky a prichádza s veľkou sadou testov, ktoré jej pridávajú na dôveryhodnosti.

Cieľom mojej práce bolo taktiež upovedomiť začínajúcich programátorov o tom, že existuje niečo ako štýl písania kódu. Preto je využitie tejto knižnice tiež dôležitým celkom mojej práce.

### Python Imaging Library

Python Imaging Library, alebo skrátene PIL je knižnica, ktorá pridáva interpreteru jazyka Python schopnosti spracovávať obrázky. Táto knižnica podporuje širokú škálu formátov, stavia na efektívnej internej reprezentácií a poskytuje pomerne silné nástroje na spracovanie obrázkov.

Knižnica je navrhnutá tak, aby poskytovala rýchly prístup k údajom uloženým v niekoľkých základných pixelových formátoch. Poskytuje pevný základ pre nástroj na spracovanie obrazu.

V práci sa využíva táto knižnica na generovanie obrázku. Pomocou poskytnutých metód, ktoré sú dostupné v triede ImageDraw, simulujeme správanie triedy Canvas z knižnice tkinter a dokážeme tak vygenerovať obrázok, bez nutnosti využitia grafického zobrazovacieho systému.

### SSIM

SSIM je knižnica, ktorú vyvinul francúzsky profesor Antoine Vacavant a ktorá implementuje výpočet štrukturálnej podobnosti obrázkov. Táto knižnica je distribuovaná pod licenciou MIT a je možné ju využiť aj v mojej bakalárskej práci.

Na rozdiel od jednoduchého výpočtu strednej kvadratickej chyby je výpočet štrukturálnej podobnosti o mnoho zložitejší. Táto knižnica umožňuje jednoducho daný výpočet outsourcovať a zabezpečiť presnejšie výsledky pri porovnávaní obrázkov. To zabezpečuje možnosť testovať grafické aplikácie začínajúcich programátorov.

# Výsledky

Výsledkom mojej práce je knižnica PEF, ktorá obsahuje takmer všetku navrhnutú funkcionalitu. Okrem danej knižnice vznikli aj ďalšie dokumenty, ktoré sú relevantné pre moju prácu.

## Používateľská príručka

Prvým z nich je používateľská príručka. Táto príručka slúži pre potencionálnych užívateľov knižnice PEF, ktorých nezaujímajú dôvody a ideológia za touto knižnicou, ale ich cieľom je efektívne túto knižnicu využiť.

Táto príručka obsahuje jednoduché príklady využitia knižnice a taktiež prehľad možností konfigurácie jednotlivých častí. Príručka má rozsah približne ôsmich strán a poskytuje tak jednoduchý zdroj informácií pre využitie knižnice.

## Existujúce úlohy

Popri vývoji knižnice PEF sa v dvoch fázach podarilo dostať ku použiteľnej beta verzií. Vďaka tomu bolo možné ku dvom domácim zadaniam v kurze Programovanie 2 na FMFI implementovať testovací modul pomocou knižnice PEF a otestovať tak knižnicu v praxi.

Pred nasadením prvého domáceho zadania prebehlo jedno interné testovanie. Vďaka tomu, že zadanie domáceho zadania bolo identické so zadaním z pred dvoch rokov, bolo možné stiahnuť si predchádzajúce riešenia študentov. S týmito riešeniami som následne spúšťal testovací skript, vyvinutý pomocou knižnice PEF. Výsledky testovaní sa automaticky ukladali do textového súboru, ktorý som po skončení testovania kontroloval a posudzoval, či bola ponúknutá dostatočne presná spätná väzba.

Následne boli nasadené dve úlohy do systému LIST, ktoré používali beta verziu mojej bakalárskej práce. Študenti riešili dané zadania a odhalili viacero nedostatkov frameworku. Zároveň som sa aj ja pri tvorbe reálnych úloh mohol vžiť do role užívateľa a zistiť, ktoré prvky tvorby testov musím vylepšiť a poskytnúť intuitívnejší prístup.

Domáce zadania prebehli úspešne – nevznikla radikálna vzbura zo strany riešiteľov. Po pár týždňoch od skončenia doby riešenia som vytvoril dotazník, ktorý som odoslal riešiteľom nasadených úloh.

## Feedback od študentov

Dotazník bol dostupný online a bol rozposlaný 43 študentom, ktorí riešili obe zadania, ktoré boli testované knižnicou PEF. Z oslovených 43 študentov vyplnilo tento dotazník 16, čo považujem za decentnú vzorku.

Dotazník bol rozdelený do viacerých častí. Pozostával z vyjadrení, ktoré mali študenti ohodnotiť na škále 1 až 5, podľa miery ich súhlasu. Ďalej kládol otázky, na ktoré bolo nutné odpovedať priamo – áno/nie. Taktiež ponúkal priestor študentom, ktorí cítili väčšiu potrebu vyjadriť sa a kládol obšírnejšie otázky, kde mali možnosť poskytnúť dlhšie textové odpovede.

Ideologicky sa dotazník zameriaval na 3 oblasti. Prvá bola zameraná na všeobecný názor na programátorské domáce zadania a ich hodnotenie. Druhá bola zameraná na prácu s automatickým testovačom a skúsenosti študentov. Posledná, pre moju prácu dôležitá, sa týkala ich konkrétneho názoru na úlohy, ktoré boli testované s knižnicou PEF.

### Všeobecný názor

Prvá informácia, ktorú treba spomenúť je fakt, že študenti nepreferujú osobné hodnotenie voči automatickému testovaniu, ale ani naopak. Až 44% študentov malo k tejto otázke neutrálny postoj. Ostatní študenti boli rovnakou mierou rozdelení do skupiny, ktorá preferuje automatické testovanie a do takej, ktorá preferuje osobné hodnotenie.

Prekvapivo, podobné výsledky vyšli aj pri názore na vyjadrenie, že by mali byť zverejnené všetky testy. Celkový výsledok bola známka 3.38, pričom hodnota 5 vyjadruje absolútny súhlas a hodnota 1 nesúhlas. Treba ale dodať, že pri tomto vyjadrení bola vysoká smerodajná odchýlka a názory študentov sa veľmi líšili.

Na čom sa študenti takmer jednohlasne zhodli je to, že výsledky neúspešného testovania by mohli poskytnúť viac informácií a že im nestačí informácia, či test prešiel, alebo nie. Pociťujú dôležitosť feedbacku ktorý dostávajú a často priebežne odovzdávajú riešenia, aby zistili, kde majú chybu.

Zároveň sa názor študentov zhoduje v tom, že jednou z najdôležitejších vecí pri domácich úlohách je kvalita samotného zadania. Môj framework sa zameriava na uľahčenie písania testov k úlohám a snaží sa zjednodušiť prácu profesorovi. Zároveň ale samozrejme ostáva veľká časť práce na nich a na tvorbe samotných úloh.

### Práca s testovačom

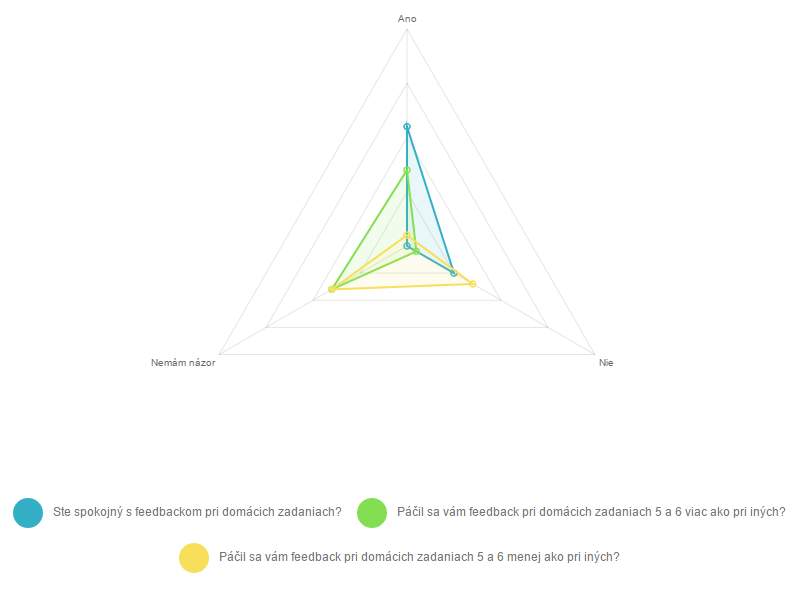
Respondentmi boli študenti, z ktorých drvivá väčšina študuje druhý semester v odbore Aplikovaná informatika. Všetci uviedli, že si zvykli na prácu s automatickým testovačom. 56% odpovedalo áno na otázku, či podľa nich niekedy strávili na zadaní primoc veľa času, kvôli chybe testovača, ako napríklad zavádzajúci feedback.

Študenti veľmi oceňujú, ak sú zverejnené príkazy, ktoré sa vykonali pred testovaním. Vadí im, že často nie je vidieť celé vstupy do funkcii, pretože sú primoc dlhé. Najviac študentov spomenulo problém, že ak ich program beží primoc dlho (alebo niekde vojde do nekonečného cyklu), nevedia zistiť príčinu. V tom prípade daný beh zastaví samotný skript, ktorý testy spúšťa a poskytne im iba informáciu, že skript bežal primoc dlho.

### Názor na PEF

Jedna z častí dotazníka sa zaoberala čisto názorom na úlohy, ktoré boli testované s pomocou PEF. Väčšina študentov, ktorí odovzdávali túto úlohu, z nej získala plný počet bodov. Samozrejme bolo aj pár výnimiek, ale nebolo ich veľa. Napriek tomu si študenti nemysleli, že zadania 5 a 6 boli ľahké. Opäť na stupnici 1 (nesúhlas) až 5 (súhlas) bola priemerná hodnota 2.87.

Na priamo položené otázky, či sa im páčil feedback pri úlohách 5 a 6 viac, ako pri ostatných odpovedali väčšinou neutrálne, alebo pozitívne voči mojej práci. Graf odpovedí aj s legendou môžete vidieť na obrázku č. 7.



Obrázok č. 11 – Spokojnosť s odozvou pri použití PEF

Tieto výsledky boli pre mňa pozitívnou správou. Po prvé indikujú, že som sa v mojej práci vybral správnym smerom. Študenti cítia potrebu kvalitných domácich zadaní s testami a moja práca smeruje k tomu, aby to umožnila. Zároveň sa ukázalo, že prvé stretnutia s mnou navrhnutým výstupom sa študentom páčili, alebo im aspoň nevadili. V neposlednom rade načrtli študenti ďalšie výzvy a názor, že stále je čo zlepšovať.

# Možnosti rozšírenia

Je viacero možností rozšírenia mojej práce. Jednou z nich je určite ladenie. V praxi pri vytváraní úloh sa najlepšie zisťujú potreby užívateľa. Aj keď sa s mojou prácou sekundárne stretla skupina študentov, PEF ešte len čaká na prvých užívateľov, ktorí pomocou neho vytvoria testový modul. Vtedy sa ukáže potreba niektoré veci implementovať iným spôsobom, možnosť detailnejšej konfigurácie, alebo niektoré veci spraviť viac intuitívne. Prácu som implementoval najlepšie, ako som vedel, ale z programátorskej praxe je jasné, že prevádzka je prvá pravá skúška.

Ďalšie rozšírenie môže samozrejme prísť rozšírením prítomnej funkcionality. Či už je to možnosť testovať ďalšie dátové štruktúry, mimo momentálne podporovaných, alebo pridanie metód do triedy testovatelného Canvasu, pre väčšiu podporu grafických komponentov. Taktiež je na mieste uvažovať nad možnosťou implementovania modulu, ktorý dokáže simulovať interakcie myši/klávesnice a umožniť tak testovať aj užívateľské interakcie.

Taktiež sa dá inšpirovať názorom študentov a vyvinúť nástroj, ktorý by dokázal limitovať čas bežania jednotlivých testov a nie len samotnej sady. Študenti by tak mohli dostať výsledok testov, ktoré dovtedy stihli prejsť a o neprekročenie času by sa nemusel starať externý skript.

Posledná evidentná možnosť rozšírenia je pridanie úplne novej funkcionality. Tú samozrejme neviem konkretizovať, pretože ak by som vedel, bola by v istej forme prítomná už teraz. Určite ale existujú ďalšie výzvy a oblasti výučby programovania, ktoré sa ťažko automaticky testujú a čakajú na nové nápady.

# Záver

Motiváciou tejto bakalárskej práce bolo vytvoriť framework, ktorý umožní jednoduché a efektívne vytváranie testov k úlohám v programovacom jazyku Python. Zároveň mala preskúmať oblasti, ktoré boli v doterajších podmienkach automaticky netestovateľné a pokúsiť sa nájsť riešenia, ktoré by to umožnili.

V práci sme analyzovali systémy, ktoré sa používajú na testovanie a odovzdávanie úloh na akademickej pôde. Rozoberala možnosť integrácie so systémom LIST, ktorý je aktuálne využívaný úvodnými kurzami programovania na FMFI. Taktiež sme sa pozreli na najrozšírenejší testovací framework jazyka Python a jemne nazreli do jeho architektúry.

Zistili sme nedostatky existujúcich frameworkov a navrhli sme riešenie, ktoré poskytuje nad rámec bežných nástrojov na testovanie aj možnosť istej kontroly testovaného modulu. Moja bakalárska práca sa zameriava na testovanie zadaní na akademickej pôde a umožňuje nastaviť limitácie a požiadavky na testovaný program.

Vyvinutý výsledok je výnimočný viacerými aspektami. Umožňuje testovať rôzne dátové štruktúry bez toho, aby museli spĺňať jednotné konkrétne predpisy. Zároveň sme splnili aj sekundárny cieľ a to tým, že práca priniesla riešenie na testovanie grafických programov. Mimo tieto výsledky poskytla jednoduchšiu prácu s výstupom z testovania a rozdelila tento výstup do viacerých častí, čím umožnila modulárny a štruktúrovaný feedback.

Na existujúcich úlohách sme si overili použiteľnosť mojej práce a odskúšali sme si testovanie s reálnymi študentami. Pomocou spätnej väzby sme zistili názor študentov. Popri tom sme postupne prácu zlepšovali, alebo zisťovali, čo ešte v práci chýba a čo študenti považujú za hlavné nedostatky automatického testovania.

Vďaka novému frameworku PEF sa rozšíria možnosti profesorov vytvárať kreatívne zadania pre svojich študentov a zároveň sa im uľahčí práca so samotným vytváraním. Moja práca je samozrejme len jednou prísadou kvalitných domácich zadaní, ale pri jej správnom zakomponovaní vie určite pomôcť vygenerovať výsledok, s ktorým budú spokojní študenti a aj profesori.

Ak by som mal zhrnúť hlavné prínosy mojej práce, tak by som medzi ne určite zaradil možnosť jednoducho testovať dátové štruktúry, jednoducho poskytnúť prehľadný feedback študentom a v neposlednej rade testovať niečo, čo sa predtým testovať nedalo.

# Použitá literatúra a prílohy

## Použitá literatúra

1. *Using Software Testing Techniques for Efficient Handling of Programming Exercises in an eLearning Platform.* **J, Schwieren.** 2006, The Electronic Journal of e-Learning Volume 4 Issue 1 available online at www.ejel.org, s. 87-94.

2. **Jursa, Andrej.** NOVÝ DLHODOBÝ VIACÚČELOVÝ SKLAD ÚLOH NA CVIČENIA. *Bakalárska práca.* Bratislava : UNIVERZITA KOMENSKÉHO V BRATISLAVE, 2013. 60082964-3a57-4d56-b396-dfd76815f051.

3. **Nepšinský, Martin.** Testovanie biela a čierna skrinka. *Slovenská technická univerzita Fakulta informatiky a informačných technológií, Manažment projektov softvérových a informačných systémov.* [Online] 1. Október 2007. [Dátum: 22. Máj 2017.] http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2008/essays/msipapersource51-nepsinsky.pdf.

4. **Martin, Robert C.** *Clean Code: A Handbook of Agile Software Craftsmanship.* Upper Saddle River, NJ : Prentice Hall PTR, 2008.

5. **Rosebrock, Adrian.** pyimagesearch. [Online] 15. September 2014. http://www.pyimagesearch.com/2014/09/15/python-compare-two-images/.

6. **HUNT, Andrew a David THOMAS.** Programátor pragmatik: jak se stát lepším programátorem a vytvářet kvalitní software Vyd. 1. Brno : Computer Press, 2007, s. 266.

7. **Schlusser, Kenneth Reitz a Tanya.** The Hitchhiker’s Guide to Python 1st Edition. *The Hitchhiker’s Guide to Python.* California : O'Reilly Media, Inc., 2016.

## Prílohy

1. Zdrojový kód frameworku PEF
2. Používateľská príručka
3. Kompletné výsledky dotazníku