1. **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 súbežnom odovzdávaní dozvedieť informáciu o správnosti jeho riešenia. Dosiahnuť to konzistentne a s efektívnym využitím času je náročná úloha, ktorú sa prakticky nedá vykonávať manuálne.

[4] Jeden z hlavných dôvodov je ten, že programátorské úlohy, jedno v akom jazyku, zvyknú mať veľkú mieru voľnosti pre študentov. Tým pádom, jednoduché porovnávanie odovzdaných riešení so vzorovým riešením neprodukuje zmysluplný výsledok použiteľný na hodnotenie študentov. 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 bohužial nie je fyzicky možné. Študenti potrebujú priebežné hodnotenie čo najskôr a čas programovania si určujú sami. Z toho vyplýva požiadavka na 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 učitela. 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, ktorý sa prvý krát stretávajú s chybami vo svojich programoch.

Zároveň sa dá zamerať na fakt, že väčsia 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 profersora.

Táto bakalárska práca sa bude snažiť analyzovať aktuálne používané riešenia pri výučbe programovania. Bude hladať ich výhody, nedostatky a pokúsi sa navrhnúť riešenie, ideálne pre kvalitnejšiu výučbu a beh kurzov pre začínajúcich programátorov na FMFI.

1. **Východiská**

Problematikou výučby programovania a zdokonaľovania sa v ňom sa dnes zaoberá viacero projektov. Ich účel je vždy rôzny. Nachádzajú sa medzi nimi komerčné projekty,

Pri analýze existujúcich riešení v tejto práci hľadíme najmä na ich charakteristické črty a niektoré z ich defektov.

* 1. **Internetové portály určené na samo-štúdium**

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, alebo ako hlavný zdroj zadaní z vlastného záujmu o programovanie. Portály, o ktorých táto práca hovorí, sú založené práve na princípe programátorských úloh so sadou testou. 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.

* + 1. **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)).

* + 1. **Portály vyvíjané jednotlivcom, resp. tímom ľudí**

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/).

Aj keď sa obe spomenuté riešenia zameriavajú na zjavne podobný problém, ich riešenia neprinášajú inšpiráciu pre učiteľa programovania. V prvom prípade nie je schopný jednotlivec nahradiť komunitu a v druhom prípade 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.

* 1. **Systémy využívané na akademickej pôde**

Pri výskume som narazil na riešenia, ktoré vyvinuli tímý na univerzitách po svete. Daní akademici taktiež evidujú potrebu na kvalitné podporné nástroje. Uvediem jeden príklad takéhoto systému.

* + 1. **xLx - eXtreme e-Learning eXperience**

Nástroj vyvinutý na University of Muenster v Nemecku. Je to vpodstate symbióza 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 vpodstate o kombináciu funkcionality, integrovanú pomocou portálu do jedného systému. Všetky veci z daného zoznamu prirodzene musia využívať všetky kurzy. Zaujímavé môže byť ale prepojenie elektronického odovzdávania a hodnotenia úloh, čo je náš hlavný cieľ aj s inými dôležitými aspektami výučby.

* 1. **Existujúce riešenia na FMFI**

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

Samozrejme, na fakulte existuje aj ďalší typ hodnotenia. O ulahčenia sa postarávajú zväčša automatické testy, ktoré sa spúšťajú po odovzdávaní na serveri.

Výstup z týchto testov je však veľmi nezhovorčivý. 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 iná, ako tá naša. Ich primárnym cieľom je:

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

*Príklad výstupu modelového testovača v kurzoch na FMFI:*



Na obrázku môžeme vidieť neúspešné riešenie. Riešenie prešlo cez prvú sadu testov, zlyhalo na druhej sade. Ako môžete vidieť, neposkytujú informáciu o type chyby, o vstupe a ani o očakávanom výstupe.

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

* + 1. **LIST**

Long-term Internet Storage of Tasks, alebo skrátene LIST je sytém ktorý navrhol a vytvoril Andrej Jursa v roku 2013 ako bakalársku prácu na Fakulte matematiky, fyziky a informatiky.

Ako vyplýva z jeho názvu, poskytuje priestor na zadávanie programátorských zadaní. V dnešnej podobe je sytém rozsiahla webová aplikácia ktorá ponúka širokú funkcionalitu.

Mimo pridávania zadaní umožňuje napríklad:

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

Tento nástroj je veľmi podobný systému xLx. Pre náš cieľ to prináša inšpiráciu. Čo si ale treba uvedomiť je fakt, že spomínané riešenia sa nezaoberajú zjednodušením vytvárania zadaní, čo je náš primárny cieľ. Riešia hlavne problém zefektívnenia interakcie medzi profesorom a študentami.

* 1. **Nástroje na tvorbu testových modulov**

[5] Keď hovoríme o testových moduloch a testovaní riešení študentov, hovoríme o takzvanom unit testingu. Pojem unit testing v tomto kontexte označuje automatické testovanie a overovanie funkčnosti implementácie zadania. Pod týmto pojmom sa zahrňujú nástroje, metodika a činnosť, ktorých cielom je správne overenie funkčnosti častí zdrojového kódu.

* + 1. **Python Unit Testing**

Python unittest je najrozšírenejší unit-testový modul vstavaný v Pythone. Jeho dizajn je postavený na XUnit frameworku navrhnutým Kent Beckom a Erichom Gammom. Rovnaký patern je možno vidieť vo viacerých jazykoch vrátane napríklad jazyka C a Java.

Tento framework podporuje tzv. fixtures, test suites a test runner. Jedna fixture je vpodstate 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 ulahčujú 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. Po istej analýze zadaní pre základnú výučbu programovania by sa mohli vyvynúť rozširujúce testovacie metódy, ktoré testujú napríklad zložitejšie dátové štruktúry, povolené kľúčové slová v riešení a uľahčili by vyučujúcim testovanie ich zadaní.

* 1. **Analytické pomôcky na opravovanie zadaní**

Ďalšia vec, týkajúca sa hodnotenia správnosti zadaní je odozva od učiteľa. Aj keď je to aspekt, ktorý smeruje k tomu, aby bol čo najviac automatizovaný, pre kvalitu vzdelania môže byť prínosom.

[1] Aj napriek tomu, že programy sa dajú skontrolovať, napríklad unit-testami, zväčša sú zadania komplexné a možno k nim pristúpiť rôznymi algoritmami, alebo stratégiami a často vyžadujú kreativitu. Preto, mnoho študentov môže veľa vyťažit aj z ľudskej odozvy, nie len binárnemu “správne vs. “nesprávne”.

Tu prichádzajú k dispozícií nové možnosti. Technológie ako CodeWebs, alebo OverCode ukazujú, ako by sa dala zefektívniť práaca s riešeniami od väčšieho množstva študentov.

[2] Elena Leah Glassman, prezentuje vo svojom článku projekt OverCode a prezentuje ho ako systém, ktorý slúži na vizualizáciu a skúmanie veľkého množstva riešení programátorských úloh. Použitím statickej a aj dynamickej analýzy vie zoskupiť podobné riešenia a umožňuje učiteľom rýchlejšie spoznať chápenie, či mylné predstavy študentov. Tým pádom môže profesor poskytovať relevantnejšiu odozvu svojim študentom.

Táto idea je určite hodná zamyslenia. Zadávateľ tak zistí, akými rôznymi spôsobmi študenti rozmýšlali. Môže rôznymi riešeniami inšpirovať iných. Zároveň je takáto analýza pri študentoch potrebná aj pre predídenie podvádzaniu.

* 1. **Python Enhancement Proposals**

Predchádzajúca sekcia ma privádza do ďalšieho bodu. Jeden z najprehliadavanejších bodov pri výučbe programovania sú interné kvality odovzdávaných programov. Samozrejme, pri úvodných kurzoch netreba zaťažovať študentov časovou a pamäťovou náročnosťou do takej miery, ako bude v konečnom dôsledku potrebné.

No aj napriek tomu je nutné, aby si študenti začali uvedomovať niektoré aspekty dobrého programátora už na začiatku ich štúdia. [3] Robert C. Martin vo svojej knihe vysvetluje nasledovné. Funkcionalita, ktorú vytvárate dnes, bude v ďalšej verzií zmenená, ale čitateľnosť vašeho kódu bude mať zásadný vplyv na všetky zmeny, ktoré kedy budete vykonávať. Štýl programovania a čitateľnosť vytvára precedens, ktorý bude aj naďalej ovplyvňovať udržovateľnosť a rozšíritelnost ešte dlho potom, čo sa pôvodný kód zmení k nepoznaniu. Váš štýl a disciplína prežije, aj keď kód nie.

Čo robí tento aspekt ešte dôležitejším, je fakt, že [Hitchhikers guide] 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.

[3] 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 dodržova ich. Pomáha prítomnosť automatizovaného nástroja, ktorý vie dodržiavanie pravidiel kontrolovať.

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 oficálnej stránke Python-u a sú súčasťou tzv. Python Enhancment Proposals. Tieto PEP opisujú a dokumentujú ako sa jazyk Python vyvýja. Taktiež poskytujú referenčný bod pre programátorov. Celok zaoberajúci sa štýlom kódu je PEP8 a je to istá príručka formátovania a štýlu programovania v Pythone. Bola vyvynutá aby pomohla písať programátorom čitateľný kód.

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

[1] A. Nguyen, C. Piech, J. Huang, L. Guibas: Scalable Homework Search, *Proceedings of the 23rd international conference on World Wide Web, Seoul, Korea, 2014*

[2] Elena Leah Glassman: Interacting with Massive Numbers of Student Solutions*, Doctoral Symposium, UIST’14, October 5–8, 2014, Honolulu, HI, USA*

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

[4] *Schwieren J et al (2006) “Using Software Testing Techniques for Efficient Handling of Programming Exercises in an eLearning Platform” The Electronic Journal of e-Learning Volume 4 Issue 1, pp 87-94, available online at www.ejel.org*

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