VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKA POKROČILÝCH KONSTRUKCÍ JAZYKA PYTHON NA ZÁKLADĚ POSKYTOVÁNÍ ZPĚTNÉ VAZBY KE STUDENTSKÝM KÓDŮM

TEACHING ADVANCED PYTHON THROUGH AUTOMATIC FEEDBACK TO STUDENT CODES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PAVEL LETÝ
AUTHOR

VEDOUCÍ PRÁCE — Doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Letý Pavel, Bc.**

Obor: Počítačové sítě a komunikace

Téma: **Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům**

Teaching Advanced Python through Automatic Feedback to Student Codes

Kategorie: Informační systémy

Pokyny:

1. Seznamte se se způsoby poskytování zpětné vazby programátorům v oblasti vysoceúrovňových dynamických jazyků a existujícími implementacemi obecných postupů.
2. Zpracujte přehled typů příkladů, které se používají v kursech zaměřených na výuku programování, aby si studenti osvojili pokročilé konstrukce těchto jazyků.
3. Navrhněte a implementujte systém, který bude schopen analyzovat zaslaný studentský kód, ověřit správnost a poskytnout zpětnou vazbu k nevhodně zvoleným konstrukcím, s případným odkazem na materiály vysvětlující vhodnější řešení.
4. Vyhodnoťte vytvořený systém v interakci s reálnými studenty, seznamujícími se s programováním v Pythonu.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle dohody s vedoucím

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT

Datum zadání: 1. listopadu 2016

Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Diplomová práce se věnuje problematice výukových systémů a jejich praktickému využití při výuce. V teoretické části práce je představena architektura těchto systémů společně s charakteristikami a příklady jednotlivých typů. Dále je popsána aplikace těchto systémů v oblasti výuky programování. Na základě těchto znalostí je navržena aplikace pro výuku jazyka Python s využitím zpětné vazby ke studentským projektům. V praktické části práce je představena implementace celého systému nasledovaná prezentací výsledků testování získaných v interakci s reálnými uživateli.

Abstract

This master thesis is focused on tutoring systems and their practical usage in education. In the theoretical part of this work is introducted architecture of this systems together with characteristics of types and examples. Next there is a description of systems in programming courses. Based on this knowledge application for teaching Python through feedback to student codes is proposed. Implementation and presentation of tests results, gained in interaction with real users, are introduced in the practical part of the work.

Klíčová slova

Python, inteligentní výukové systémy, automatizace učení, poskytnutí zpětné vazby

Keywords

Python, intelligent tutoring systems, learning automatization, feedback provision

Citace

LETÝ, Pavel. *Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Pavel Letý
23. května 2017

Poděkování

Rád bych poděkoval vedoucímu mé práce panu doc. RNDr. Pavlu Smržovi, Ph.D. za poskytnutou pomoc, konzultace a trpělivost při tvorbě této diplomové práce.

Obsah

Kapitola 1

Úvod

Se zvyšujícím se počtem studentů, kteří mají zájem o studium programování, roste potřeba vývoje automatizovaných nástrojů, které studentům nabídnou další možnost, jak se v dané problematice programování zdokonalovat nad rámec standardní výuky. Nespornou výhodou výukových systémů je, že umožňují individuální přístup, umí se adaptovat na tempo učení daného studenta a student je může využívat kdykoli a kdekoli. Zároveň tyto systémy šetří čas vyučujícím, kteří se pak mohou věnovat konzultacím nebo pořádání cvičení navíc.

Tato práce se zabývá vývojem aplikace pro automatizaci výuky konstrukcí jazyka Python ve všech úrovních obtížnosti. Výsledky analýz studentských řešení jsou prezentovány pomocí zpětné vazby, již tvoří analýza korektnosti, stylu a navržení zlepšení. Poslední uvedená část má v aplikaci největší význam. Student získá informaci, jak některé části svého řešení napsat lépe a kde se o dané problematice může dočíst více. Výhoda tohoto přístupu spočívá v cílenosti výuky na témata, která studentovi nejdou, a poskytuje mu možnost studia ze zdrojů zaměřujících se přímo na danou problematiku.

V kapitole 2 jsou obecně popsány systémy pro automatizaci učení, jejich architektura, dělení a příklady implementace. Popisuje nároky na návrh takového systému a podobu zpětné vazby. Dále tato kapitola pojednává o aplikaci těchto systémů v oblasti programování. Rozebírá především formu výuky kurzů programování na vysokých školách a vymezuje místo, které v nich tyto systémy mají. Kapitola 3 je věnována návrhu aplikace realizující systém pro automatizaci učení. V této kapitole je popsána architektura systému a v kapitole 4 její implementace. Kapitola 5 prezentuje výsledky testů implementace. V závěru práce jsou uvedeny dosažené výsledky společně s návrhy pro další vývoj systému. O tom pojednává kapitola 6.

Kapitola 2

Systémy pro automatizaci učení

V této kapitole je představena typická architektura systémů pro automatizované učení a její použití. Dále zde jsou uvedeny jednotlivé typy systémů s příklady implementací. V závěru kapitoly je diskutována jejich aplikace v oblasti programování a nároky kladené na návrh. V závěru jsou uvedeny způsoby jejich využití ve výuce.

2.1 Architektura

Systém ITS (Intelligent tutoring system) poskytuje možnost individuálního přístupu při učení se dané problematiky. Hlavním účelem takového systému je zlepšit proces učení. To lze realizovat různými způsoby jako jsou simulace nebo hra. ITS se snaží být alternativou k tradičním způsobům výuky, nabízí možnost studovat kdekoli a kdykoli pomocí interaktivní a často i zábavné formy výuky, a tím cílí na zájem studentů o danou problematiku.

Použitím těchto systémů lze snížit náklady na výuku tím, že již není nutná fyzická přítomnost studentů ve školách, což v konečném důsledku znamená snížení nákladů na provoz v prostorách školy. Tyto prostředky pak mohou sloužit pro nákup novějšího vybavení potřebného pro výuku [45].

ITS nalézají využití v mnoha odvětvích, ve kterých pomáhají studentům s porozuměním problémů z různých oblastí jako jsou algebra, astronautika, programování či logika [20]. Lze je nalézt v armádě, průmyslu a dalších odvětvích [45].

Dnešní systémy se typicky skládají z následujících komponent [35]:

- Expertní modul (expert module) je část ITS, která pokrývá znalost domény, do které je systém zasazen. V praxi to znamená, že expertní modul ITS systému pro výuku matematické logiky musí být schopný pokrýt relevantní znalosti z této problematiky, např. axiomy výrokové logiky, logické spojky, formule či dokazovací systém výrokové logiky.

- Znalost studujícího (student module) má za úkol modelovat studentovy vědomosti. Každý student má vlastní model, který se v průběhu výuky mění. Na základě znalosti obsažené v tomto modelu jsou doporučovány materiály k prostudování. Modul poskytuje funkci pro generování problémů k řešení odpovídajících studentovým znalostem. Aktualizace modelu je závislá na správnosti studentových řešení předložených problémů.

- Znalost strategie učení (tutor) je zodpovědná za generování problémů k řešení, v závislosti na odpovědích studenta určuje učební strategii. V případě špatných odpovědí

opakuje látku, přidává další cvičení či podává podrobnější vysvětlení. Pokud student odpovídá správně, tak určuje, co je vhodné studovat dál.

Autorka knihy [45] toto dělení rozšiřuje následovně:

- Schopnost systému generovat úlohy k řešení, poskytovat nápovědu a pomoc úměrně studentovým znalostem. V některé literatuře [39] též označováno jako *Curiculum*.
- Schopnost systému vylepšovat sama sebe v učebních strategiích na základě získaných zkušeností s výukou studentů.

a dále diskutuje základní stavební kameny, které umožnily vznik těchto systémů.

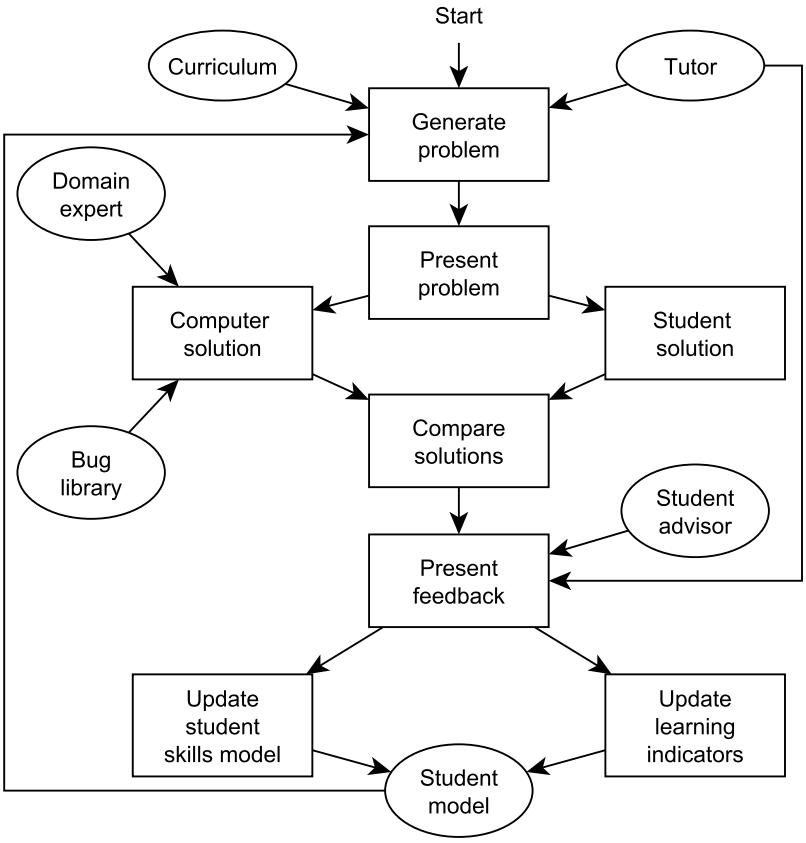
- Umělá inteligence – Marvin Minski[1] definoval umělou inteligenci jako vědu o vytvoření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který by byl považován za inteligentní v případě, že by danou úlohu měl vykonat člověk [30].
- Kognitivní věda – Transdisciplinární obor, který se věnuje problematice porozumění, jak lidé či stroje získávají znalosti a jak je dále používají k tomu, aby se chovali inteligentně. Jako vstupní data oboru slouží většinou výzkumy. Z hlediska ITS je tato věda vhodná pro porozumění cílové skupině lidí a jednotlivcům. Z toho plyne, že pokud ITS dokáže chápat, jak daný student myslí, nebo přistupuje k řešení problému, tak může navrhnout optimální strategii učení.
- Internet – Neomezený zdroj informací, který není závislý na čase či místě. Internet je nicméně jednoduše zneužitelný pro různé typy plagiátorství a nedovolenou spolupráci.

Na obrázku 2.1 je znázorněno základní schéma funkcionality ITS. Učení je založeno na řešení úloh studentem. Úlohy musí být vhodně zvoleny tak, aby byly jednoznačné a srozumitelné. Celý proces začíná vyhodnocením a vymodelováním znalostí studenta (v obrázku označeno jako Student model). Při modelování musí systém uvažovat, jaké jsou studijní prerekvizity pro řešení úloh z dané problematiky (Curriculum). Z těchto prerekvizit musí systém být schopen vybírat a následně zvolenou prerekvizitu vhodně prezentovat (Tutor). Prezentace probíhá formou výběru, nebo generováním úlohy, která je předložena studentovi k řešení. ITS si interně připraví svoje řešení tím, že danou úlohu vyřeší, nebo použije již předpřipravené řešení (Domain expert). Pak systém čeká, než student dokončí svoje řešení a vloží jej do systému. Jakmile se tak stane, ITS porovná svoje řešení s řešením odevzdaným studentem. Na základě porovnání je generována zpětná vazba, v které jsou uvažovány okolnosti řešení z předchozích pokusů, resp. co se v řešení studenta změnilo od poslední poskytnuté zpětné vazby (Student advisor). Poté dojde k aktualizaci modelu studenta (Student model). V modelu jsou aktualizovány studentovy schopnosti a indikátory pokroku učení daného studenta.

Využití ITS je založeno na získávání praktických zkušeností z dané problematiky, na základě kterých se studenti mohou naučit jak abstraktně myslet (abstrahovat řešení). Existuje mnoho způsobů, jak docílit správného výsledku z pohledu funkcionality. Úkolem ITS je asistovat studentovi, případně mu pomoci docílit optimálního výsledku [14, 31].

ITS přispívají mimo jiné k udržení zájmu studentů. Je totiž známo, že zadání náročných úloh bez pomoci studenty spíše odrazuje. Dokladem může být například studie [9], ve které 29 studentů bez předchozích zkušeností z programování mělo řešit zadané úlohy.

[1] Americký vědec, který se zabýval umělou inteligencí.



Obrázek 2.1: Obecné schéma inteligentního systému. Převzato z [39]

Zpočátku, během seznamování se s úlohami, se udržoval stav rovnováhy. Později následoval stav nerovnováhy (zmatení), kdy si studenti nevěděli rady s některými úlohami. Tento stav u některých přecházel až ve frustraci z toho, že nejsou schopni danou úlohu bez pomoci vyřešit. Po čase došli tito studenti až do stavu, kdy se přestali snažit hledat řešení. Z toho plyne, že student, který se dostává do úzkých při řešení úloh, může snadno přijít o motivaci danou problematiku dál studovat.

2.2 Typy systémů

V následujícím textu jsou představeny jednotlivé typy ITS. Dělení bylo převzato z [31].

- **Systémy založené na dialogu** – Student a tutor mezi sebou komunikují na základě otázek. Oba mají možnost položit otázku, a tím mezi nimi vzniká dialog. Tutor musí být schopen porozumět přirozenému jazyku studenta a na základě otázek generovat relevantní odpovědi i za předpokladu, že otázky studenta mohou být vágní. Tutor musí dále korektně vyhodnotit, jak student porozuměl problematice, a na základě toho navrhnout další postup. Základem těchto systémů bývá databáze dialogů, která je používána pro generování otázek a odpovědí. Příkladem takto založeného systému je například *PROPL* [29].

- **Systémy založené na zpětné vazbě** – Učení v těchto systémech je realizováno pomocí zpětné vazby generované pro programy psané studenty. Cílem je, aby dodaná zpětná vazba byla užitečná, pomohla zlepšit programovací schopnosti studenta a byla schopná doporučit zlepšení v případě, že dané řešení problému není optimální. Tyto systémy vyžadují, aby aplikace byly spustitelné. Příkladem může být systém *JITS* [43].

- **Systémy založené na příkladech** – Studentům jsou vysvětleny ukázkové problémy s řešením a pak jsou vyzváni k řešení podobných problémů stejného typu samostatně. Příkladem tohoto přístupu je systém *NavEx* [46], webová aplikace, která prezentuje vybrané příklady společně s textovým vysvětlením pro každý důležitý řádek zdrojového kódu. Tento způsob umožňuje studentům přistupovat k příkladu řádek po řádku, nebo se rovnou přesunout k řádku, který obsahuje problematickou část kódu. Další možností tohoto přístupu jsou nástroje, které studentovi prezentují programátorský problém společně se šablonou řešení, kterou musí doplnit. Příkladem je systém *ADAPT* [19] pro vyučování jazyka Prolog.

- **Systémy založené na simulaci** – Simulačně založený přístup učebního systému se zabývá dvěma hlavními problémy. Prvním je dynamičnost programu (například obsah proměnné se může měnit v čase běhu programu). Druhým problémem je, že studenti si často nedokáží představit, jak jednotlivé části programu fungují (například pro řešení problému byla použita velká míra abstrakce, které student neporozuměl). Simulačně založený systém řeší tyto problémy pomocí vizualizací částí řešeného algoritmu a tím snižuje míru abstrakce daného řešení. Příkladem je systém *OOP-AMIN* [13].

- **Systémy založené na analýze programu** – Na rozdíl od přístupů, které vyžadují syntézu programu či algoritmu, tento přístup umožňuje studentům přiblížení problematiky na základě analýzy programu. Příkladem je systém, který vytvořil Amruth N. Kumar [28]. Tento systém slouží k výuce programovacího jazyka C++ a k učení používá analýzu/debuggování segmentů zdrojového textu. Tyto systémy se častěji zabývají výukou programových konstruktů, než výukou programování jako takového.

- **Systémy založené na spolupráci** – Všechny výše zmíněné techniky jsou individuální. To znamená, že student pracuje samostatně pouze s pomocí tutora. Tato technika předpokládá učení na základě spolupráce mezi skupinou studentů. Studenti se tak naučí prezentovat a prosazovat svoje názory ve skupině, naučí se ze všech předložených nápadů složit jeden společný, který si z každého dílčího nápadu bere to nejlepší. Nevýhodou může být pasivní člen, který snižuje celkovou výkonnost skupiny, a tím negativně ovlivňuje její výsledky. Příkladem je systém *HABIPRO* [44].

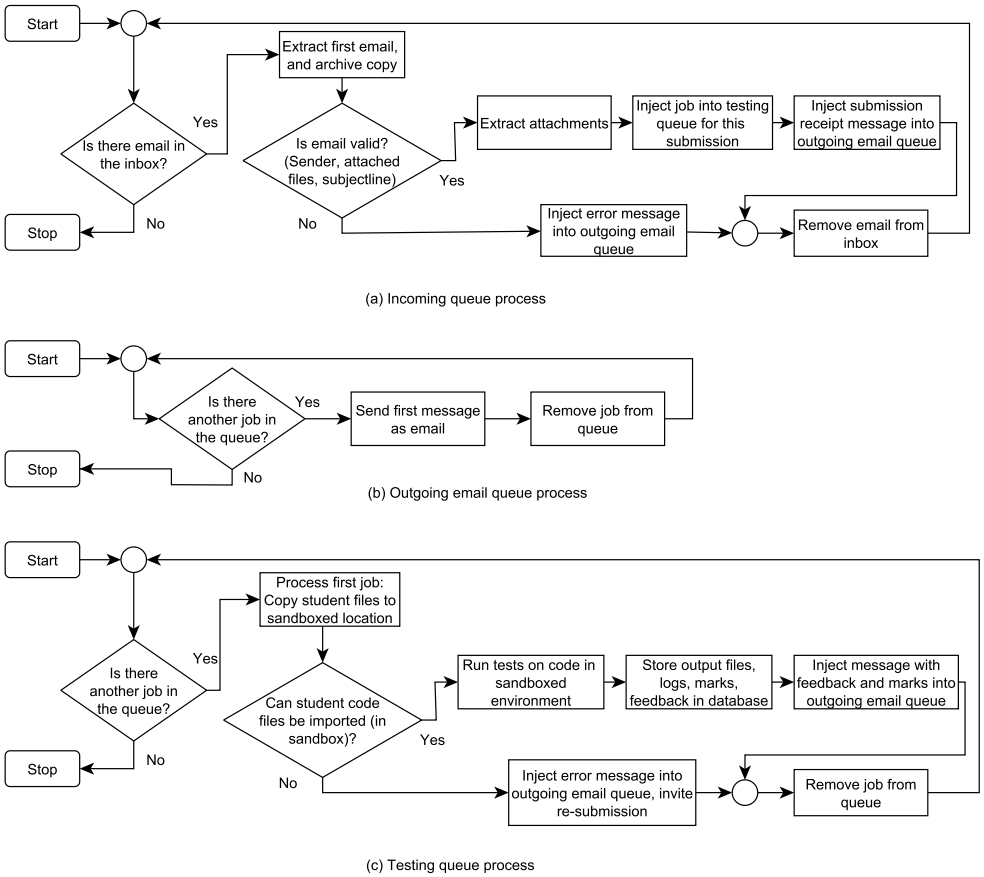
2.2.1 Příklady realizací

Prvním představeným systémem je systém určený pro automatické vyhodnocování studentských projektů napsaných v programovacím jazyce Python. Následující popis byl převzat z [14].

Tento systém vznikl na univerzitě v Southamptonu[2] jako další podpůrný prostředek pro výuku úvodních kurzů programování. Má za cíl pomoci začínajícím programátorům s domácími úlohami a zároveň šetřit čas učitelů, který by jinak strávili opravováním těchto

[2] Viz http://www.southampton.ac.uk/

úloh. Tento čas pak učitelé mohou využít k pořádání extra přednášek, demonstračních cvičení, případně mají více času na konzultace se studenty, kteří potřebují více individuálního přístupu. Systém je založen na technice Test-first, která je podrobně představena v 2.3.3. Vyhodnocování probíhá na základě učitelem předem připravených testů. Dále je kontrolován styl zdrojového textu. Jelikož jako implementační jazyk je použit Python, je kontrolováno, jak odevzdaný zdrojový text splňuje doporučení PEP8 normy[3]. Studenti svá řešení odevzdávají jako přílohu emailu, který je zaslán na speciální, k tomuto účelu zřízenou, emailovou adresu. Zpětná vazba je doručována emailem zpravidla do několika minut po odevzdání. Systém je periodicky spouštěn pomocí nástroje `cron`[4]. Obrázek 2.2 demonstruje funkcionalitu tohoto systému.



Obrázek 2.2: Vývojové diagramy zobrazující funkčnost systému představeného autory Hans Fangohr a Neil O'Brien. Převzato z [14]

Zavedením tohoto systému studenti mohou získat cennou zpětnou vazbu pro jejich laboratorní řešení v řádu několika minut, a případně svoje řešení na základě této zpětné vazby upravit. Zpětná vazba obsahuje výsledky testů, kontroly stylu zdrojového textu a předpokládanou finální známku z kurzu.

[3]Jedná se o souhrnná doporučení pro psaní zdrojového textu v Pythonu, více viz https://www.python.org/dev/peps/pep-0008/

[4]Linuxový nástroj pro periodické volání zvolených úloh, více viz https://linux.die.net/man/1/crontab

Druhý systém byl vytvořen na Massachusettském technologickém institutu. Vyhodnocuje studentské projekty na základě referenčního řešení, které implementuje učitel. Společně s referenční implementací tento systém počítá s chybovým modelem, který říká, jakých typů chyb se studenti nejčastěji dopouštějí během řešení daného úkolu. Systém vychází z toho, že specifikace problému (řešený úkol) je předem známá, tudíž lze předpovědět, jakých chyb se studenti budou pravděpodobně dopouštět. Pro modelování chyb byl navrhnut speciální jazyk, v převzaté literatuře označený jako EML (Error model language). Chybový model se skládá ze sekvence pravidel. Cílem je najít minimální počet oprav tak, aby studentské řešení odpovídalo referenčnímu [40].

2.3 Aplikace výukových systémů v oblasti programování

Tato sekce se věnuje využití výukových systému v oblasti programování. Diskutuje kritéria jejich návrhu společně s důvody, proč je vhodné je v tomto prostředí používat.

2.3.1 Výuka a samovzdělávání

V této sekci je věnována pozornost metodám výuky vybraných kurzů z různých oborů vyučovaných na univerzitách. Největší zastoupení kurzů zde má FIT VUT v Brně[5], protože se jedná o mateřskou univerzitu autora této práce. V následujícím přehledu jsou diskutovány formy výuky/aktivity, které dnes nejčastěji student může využít k učení. Jako základní dělení se nabízí dvě možnosti. První možností je jakákoli forma výuky na univerzitě (prezenční, dálková), kdy student dochází na univerzitu a absolvuje kurzy. Druhou možností je samostudium, kdy student prohlubuje svoje znalosti nad rámec znalostí získaných pomocí první možnosti, nebo se snaží naopak doplnit mezery v probírané látce. Nejdříve představíme formy výuky na univerzitách, čerpáme ze struktury kurzu Základy programování (IZP) [27] a z [14, 40].

- Přednáška – jeden z hlavních zdrojů nových poznatků, kdy vyučující vysvětluje problematiku velkému počtu studentů, a proto není možné, aby se věnoval každému studentovi zvlášť. Protože přednášek je omezený počet v semestru (13 týdnů), obvykle se jedná o jednu 2-3 hodinovou přednášku za týden, je nutné veškerou plánovanou látku odpřednášet v požadovaném čase. To leckdy pro vyučujícího znamená zvolit určité tempo při přednášení a redukovat množství detailů. Ve výsledku někteří studenti mohou mít problém s tempem nebo mírou detailů, a tak nemusí pochopit některé důležité koncepty.

- Cvičení – využívají se k doplnění a procvičení odpřednášené látky. Obvyklý rozsah je 1 až 2 hodiny týdně. Často jsou rozlišovány dva typy cvičení: laboratorní, kde student procvičuje probíranou látku prakticky, a numerická, sloužící k ověření teoretických znalostí, například různé důkazy.

- Konzultace – pokud je student v úzkých takovým způsobem, že nedokáže daný problém sám vyřešit, je obvykle k dispozici možnost konzultace s vyučujícím daného předmětu.

- Projekty – ve většině předmětů nedílná součást hodnocení. Slouží především k procvičení probírané látky. V některých předmětech jsou zadávány i týmové projekty, kdy je kladen důraz na spolupráci jednotlivých řešitelů.

[5] Viz http://www.fit.vutbr.cz/

Dále je zmíněn souhrn možností samovzdělávání.

- Kurzy na internetu – většinou se jedná o kurzy MOOC[6] označující masivní otevřené kurzy přes internet. Příkladem mohou být Coursera[7] a Udacity[8]. Jedná se o interaktivní kurzy, které se sestávají z předem nahraných videí vysvětlujících danou problematiku, kvízů a projektů. Kvízy a projekty slouží pro ověření a procvičení předpokládaných znalostí získaných po shlédnutí vysvětlujícího videa. Typická je rovněž online diskuze mezi zapsanými uživateli. Po úspěšném dokončení kurzu uživatel obdrží certifikát [25].

- ITS – nabízí jedinečnou možnost individuálního přístupu a adaptace na tempo učení každého studenta. Je obvykle možné mít přístup k systému z domu, například při vypracovávání projektů nebo řešení úloh z laboratoří. ITS může být rovněž použito pro průběžné ohodnocování studentů a k výpočtu finálních známek z kurzů.

- Ostatní – různé typy doplňkových aktivit jako například studium doporučené literatury ke kurzům, experimentování s programovacími jazyky a další.

V následující tabulce 2.1 jsou prezentovány vybrané kurzy společně s formami výuky, které se v těchto kurzech používají.

Univerzita/Fakulta	Předmět	Přednášky	Počítačová cvičení	Projekty
VUTBR/FIT	Základy programování (IZP) [27]	13 (39h)	12 (24h)	4
VUTBR/FIT	Skriptovací jazyky (ISJ) [41]	13 (26h)	–	8
VUTBR/FIT	Jazyk C (IJC) [33]	13 (39h)	–	2
MU/FI	Úvod do programování (IB111) [24]	13 (26h)	13 (26h)	–
ČVUT/FIT	Programování a algoritmizace I (BI-PA1) [4]	13	13	–
VUT/FIT	Logika (LOG) [47]	13 (26h)	13 (26h)	–
VUT/FIT	Matematické struktury v informatice (MAT) [48]	13 (39h)	13 (1h)	–

Tabulka 2.1: Tabulka aktivit v rámci kurzů z relevantních oblastí na univerzitách

Nyní jsou uvedeny některé metody výuky, které jsou běžně používány ve školách. Rozdělení převzato z [45].

- **Pasivní metody** – metody založené převážně na výkladu, kdy vyučující přednáší danou problematiku studentům a typicky je iniciátorem všech otázek, které směřuje ke studentům. Studenti pasivně naslouchají. Takto založené kurzy bývají velice dobře organizovány, mají většinou pevně daný řád, který se zpravidla moc nemění, a kurz má stále stejnou strukturu. To může v některých případech znamenat i to, že je

[6] Z anglického spojení Massive Open Online Course
[7] Viz https://www.coursera.org
[8] Viz https://www.udacity.com

přednášena neaktuální látka. Samotné kurzy nemusí být špatné, ale forma výuky může znatelně ovlivnit studentův zájem. Takto založené metody nejsou moc efektivní a studenti většinou vynaloží úsilí pouze k tomu, aby daný předmět úspěšně absolvovali.

- **Aktivní metody** – oproti pasivním metodám mají velkou výhodu. Aktivně zapojují studenta do výuky a dělají pro něj předmět zajímavějším. Příkladem takových metod je spolupráce v týmu, tzn. metoda dotazování, která je založena na vysvětlení principů a možných scénářů dané problematiky. Nutí studenty vytvářet vlastní otázky, experimentovat, analyzovat a dohledávat informace navíc.

Mezi aktivní metody dále patří i tzn. výuka 1 na 1, kdy se vyučující může studentovi věnovat v plné míře, přizpůsobit výklad jeho tempu a chápání. Takto koncipovaná výuka se dle Blooma [8] jeví jako nejefektivnější. Nicméně aktivní metody nejsou v klasické struktuře výuky, tak jak je nastíněna v tabulce 2.1, možné. Důvody jsou hlavně organizačního charakteru, kdy není možné v rámci přednášek s velkým počtem studentů přizpůsobit výklad každému z nich, nýbrž je nutné látku vysvětlovat s určitým množstvím detailů, jak již bylo diskutováno. Zde právě vzniká prostor pro výukové systémy, které tuto funkcionalitu mohou v různých formách nabídnout a tím zefektivnit samotnou výuku.

2.3.2 Přehled nejčastějších chyb v programech

V této sekci jsou představeny nejčastější chyby, kterých se studenti dopouštějí při řešení úloh. Autoři článků [3, 10] představují experiment, kdy k vytvoření klasifikace chyb využili přes 250 000 studentských řešení v jazyce Java sesbíraných z různých institucí. Cílem experimentu bylo zanalyzovat frekvenci chyb, jejich šíření mezí řešeními a čas potřebný k nalezení a opravě těchto chyb. Autoři uvádějí, že výsledky experimentu mohou být použity například pro zvážení změn v návrhu kurzů zabývajících se programováním. Dále mohou být prospěšné pro autory učebnic a různých online kurzů. Zjištěné třídy chyb jsou uvedeny v tabulce 2.2.

Třída chyb	Příklad
Nepochopení/neznalost syntaxe	`if (a = b))`
Typové chyby	`list.get("abc")`
Další sémantické chyby	`if (a == "start")`

Tabulka 2.2: Tabulka tříd nejčastějších chyb v programech

Výše uvedená tabulka představuje tři hlavní kategorie chyb, které se nejčastěji vyskytovaly v rámci výzkumu. První třída reprezentuje čistě chyby způsobené neznalostí syntaxe. Krom příkladu z tabulky, kde je použit nesprávný operátor porovnání, jsou to dále špatně používané závorky ve výrazech, smyčkách či podmínkách. Dále chybně používané logické operátory nebo nesprávné použití středníků. Druhou třídu tvoří typové chyby, kdy nejčastěji byly volány metody s argumenty špatných typů, jak je uvedeno v příkladu výše, nebo byly použity nekompatibilní typy mezi proměnnou a vrácenou hodnotou z funkce. Sémantické chyby jsou poslední třídou v tomto průzkumu. Nejčastěji se vykytovaly v podobě funkcí s definovaným návratovým typem, které neměly nastavenou žádnou návratovou hodnotu. Případně se jednalo o chyby, kdy řetězce byly porovnávány na základě operátoru `==` namísto funkcí k tomu určených, jak je demonstrováno v tabulce.

Podobné dělení představili i autoři ve článku [20]. Rozlišují zde chyby vzniklé nedbalostí, nedostatkem znalostí a na chyby způsobené nepochopením konceptu. Syntaktické chyby

mají tu výhodu, že jsou oznámeny interpretem/kompilátorem a student je může opravit. V případě chyb založených na nepochopení konceptu či nedostatku znalostí nastává riziko vzniku skrytých chyb. To znamená, že se chyby nemusí projevit pro každý vstup, ale projevují se pouze pro specifickou kombinaci vstupů, což vyúsťuje v nedefinované chování programu, či jeho pádu.

2.3.3 Techniky programování

V následující sekci jsou uvedeny vybrané techniky, které jsou často použity v průběhu vývoje nového softwaru.

- Extreme Programming je agilní metodika vývoje software, která je považována za efektivní, týmovou, flexibilní a s nízkou mírou rizika. Obsahuje několik fází a metod, z nichž je zajímavá metoda párového programování (Pair programming), kde samotný produkční kód píší dva lidé. Oba se snaží programovat současně, vedou mezi sebou dialog o designu, testech a snaží se ze společných myšlenek vybrat tu nejlepší [7].

- Code Review je technika programování, kdy změny ve zdrojovém kódu mohou být diskutovány ostatními, tzn. lze k nim vznést připomínky. Typická realizace vypadá tak, že existují dva účastníci, autor a QA inženýr (Quality assurance). Autor provede změnu ve zdrojovém kódu, kterou pak na základě připomínek QA inženýra opraví. Pokud kód dosahuje požadované kvality je začleněn do aktuální verze produktu [36].

- Black Box, White Box, Gray Box – Black box je metoda testování, kdy programátor, který píše testy, neví nic o tom jak vypadá návrh a struktura testované aplikace. Tato metoda je využívána především pro testování na vyšších úrovních, například pro akceptační testy[9]. White Box metoda je přesný opak metody Black box, struktura programu je předem známa a programátor je schopen napsat podrobnější testy. Gray box je kombinace obou dříve zmíněných metod [5].

- Z hlediska metodiky vytváření testů se v dnešní době nejvíce používají dvě techniky Test-first a Test-last. Pro obě techniky platí, že jsou využívány při aplikaci inkrementálního modelu vývoje software[10] Hlavním rozdílem mezi těmito dvěma technikami je doba psaní testů. V případě Test-first jsou testy napsány ještě před samotným produkčním kódem. Naopak v případě Test-last jsou testy napsány až po implementaci produkčního kódu. Zbylé rozdíly přehledně zobrazuje následující tabulka2.3 [12].

	Test-First	**Test-Last**
Who writes and runs tests?	Programmer	Programmer
When are tests written?	Before production code	After production code
When are tests run?	While writing production code, frequently	After writing production code, less frequently
Incremental development?	Yes	Yes
Regression testing?	Yes	Yes

Tabulka 2.3: Souhrn rozdílů mezi technikami Test-first a Test-last. Převzato z [12]

[9] Více viz http://softwaretestingfundamentals.com/acceptance-testing/
[10] Podrobnosti lze najít na: http://www.testingexcellence.com/incremental-model/

2.3.4 Kritéria návrhu výukového systému

Následující dělení bylo převzato z [14].

- **Korektnost** – řešení je ověřeno z hlediska správné funkcionality. Pro různorodé vstupy jsou kontrolovány korektní výstupy, většinou na základě testů. Pro tvorbu testů mohou být použity různé strategie. Nejčastěji se jedná od techniky Test-first a Test-last představené v 2.3.3. Implementačně lze tyto techniky realizovat různými způsoby. Programátor si může testy napsat sám, nebo použít již existující mechanismy jako jsou například jednotkové testy[11]. Jednotkové testy jsou založeny na White Box mechanismu uvedeném v 2.3.3. Testy jsou vytvářeny pokud možno pro všechny komponenty (jednotky) programu tak, aby bylo možné ověřit, zda jejich funkčnost odpovídá návrhu. Výhodou jednotkových testů je, že tento přístup je implementován pomocí různých knihoven/balíků pro většinu programovacích jazyků. Programátor, který si tento způsob osvojí, jej může využít napříč programovacími jazyky. Příkladem může být implementace pro Python[12], C++[13] nebo Java[14].

- **Rychlost implementace** – toto kritérium nám říká, jak rychle dané studentské řešení dokáže spočítat výsledek. Uvažujeme dobu běhu programu (Execution Time), kterou můžeme použít pro prvotní odhad efektivity daného řešení. Tento údaj můžeme získat přímo z výstupu generovaného pomocí jednotkových testů nebo v Linuxu pomocí příkazu `time`[15].

- **Pamětové nároky** – na základě pamětových nároků programu dokážeme odhalit vhodnost použitých datových struktur a případné chyby při práci s pamětí (například špatné uvolnování paměti). Pro tento účel lze použít celou řadu nástrojů. V Linuxu je asi nejznámější *valgrind*[16].

- **Robustnost** – nám říká, jak je dané řešení odolné vůči nekorektním/chybějícím vstupům, jak je schopno se vyrovnat s obsáhlými soubory a jak kvalitní jsou použité regulární výrazy apod.

- **Elegance, styl, znovupoužitelnost, dokumentace** – toto kritérium se snaží odhalit nedostatky v použitých datových strukturách nebo nevhodně zvolených řídících konstrukcích. Dále odhaluje nedostatečnou dokumentovanost zdrojového textu, zda je řešení správně abstrahované a další. Pro tento účel je většinou nutná manuální inspekce zdrojového textu. Kontrola stylu zdrojového textu je dalším kritériem sloužícím především k tomu, aby si student osvojil vhodné návyky pro psaní zdrojového textu. Pro tyto účely lze využít automatizované nástroje, které jsou pro různé programovací jazyky připravené, a studenti o nich často nevědí. Příkladem takového nástroje může být pro Python flake8[17] či pylint[18]. Pro jazyk Perl existuje projekt Perltidy[19] nebo

[11] Podrobné informace lze nalézt zde: http://softwaretestingfundamentals.com/unit-testing
[12] Viz https://docs.python.org/3.4/library/unittest.html
[13] Viz http://cppunit.sourceforge.net/doc/cvs/class_test_runner.html
[14] Viz http://junit.org/junit4/
[15] Viz https://linux.die.net/man/1/time
[16] Viz http://valgrind.org/
[17] Více zde: http://flake8.pycqa.org/en/latest/
[18] Podrobnosti zde: https://www.pylint.org/
[19] Domovská stránka projektu: http://perltidy.sourceforge.net/

multiplatformní nástroj Vera++[20]. Další možností je využití chytrého vývojového prostředí, které nabízí integrovanou funkcionalitu výše zmíněných nástrojů. Příkladem mohou být populární prostředí NetBeans nebo Eclipse.

- **Bezpečnost** – pokud uvažujeme ITS, které provádějí analýzu na spustitelném studentském řešení, je nezbytné přijmout jistá bezpečnostní opatření, která zabrání ať už nechtěnému, nebo chtěnému útoku na ITS [14]. Jelikož uvažujeme spustitelný kód, je nutné zajistit prostředí pro spuštění tohoto kódu. Toto prostředí je většinou realizováno pomocí pískoviště (sandbox), do kterého je vloženo studentovo řešení. Pískoviště je dedikované místo na disku se speciálními bezpečnostními nastaveními, které je určeno pro testování/spouštění potenciálně nebezpečného kódu. Příkladem těchto nastavení jsou uživatelská práva, spuštění programu pod speciálním účtem, kvóty na diskový prostor či požadavky na zdroje (POSIX resource limits[21]). Lze tedy předejít vedlejším účinkům programů, které mohou být i destruktivního charakteru. Dále je nezbytné skrýt strukturu systému před uživateli a umožnit jim interakci se systémem pouze pomocí definovaného rozhraní [14, 42].

2.3.5 Zpětná vazba

Výsledkem výukového systému by měla být kvalitní zpětná vazba poskytovaná studentovi. Ta může nabývat různých podob. V případě systému založených na dialogu představených v 2.2 to mohou být vhodně zvolené otázky pokládané studentovi, nebo správně zvolená strategie učení založená na studentových znalostech. Výhodou této zpětné vazby je, že je interaktivní. V případě systémů založených na poskytování zpětné vazby uvedených v 2.2 zpětná vazba není interaktivní, nýbrž se většinou jedná o textovou zprávu, která je pro daného studenta vygenerována na základě vloženého řešení. To zvyšuje požadavky na její kvalitu a hlavně srozumitelnost [31, 20].

V článku [21] byly představeny dvě hlavní kategorie strategií poskytovaní zpětné vazby:

- `Giving-Answer` – tato kategorie strategií funguje tak, že učitel pomocí dalších dílčích strategií vede studenta k tomu, aby dokázal sám identifikovat chybu. K tomuto využívá podobu finálního řešení. Mezi dílčí strategie patří *Opakování*, kdy učitel poukáže na chybnou část ve studentově odpovědi, či *Přeformulování*, kdy se učitel snaží vysvětlit nalezenou chybu z jiného úhlu pohledu. Další možností je tzn. *Přímá odpověď*, v případě, kdy student nerozumí dané problematice a je nutné ho přímo nasměřovat.

- `Prompting-Answer` – učitel pokládá dotazy studentovi, pomocí těchto dotazů se snaží studenta nasměřovat k nalezení chyby. Dotazy jsou formulovány tak, aby neobsahovaly přímo finální řešení. Z dílčích strategií této kategorie lze uvést *Vyjasnění požadavku*, učitel se formou otázky snaží studenta navést k tomu, že jeho řešení je špatné. Další možností je poskytnout část správného řešení a nechat studenta doplnit chybějící část.

Výše byly představeny dvě obecné skupiny strategií pro poskytování zpětné vazby. Některé typy ITS představené v 2.2 využívají kombinace těchto dvou přístupů, např. systémy založené na dialogu nebo příkladech. Co se týká systémů založených na poskytování zpětné vazby, ty využívají především možnosti první vysvětlené skupiny strategií.

Dalším požadavkem na zpětnou vazbu je především srozumitelnost [21]. Je vhodné zpětnou vazbu rozdělit do dílčích logických celků, které budou obsahovat výsledky. V případě

[20] Viz https://bitbucket.org/verateam/vera/wiki/Home

[21] Nastavením těchto limitů lze například předejít nekonečným smyčkám.

programovacích úloh mohou logické celky odpovídat kritériím zmíněných v 2.3.4, s výjimkou kritéria bezpečnosti. Zpětná vazba by měla obsahovat závěr, kde budou uvedena doporučení (co je vhodné nastudovat, kde nalézt další informace k problematice) [14].

Kapitola 3

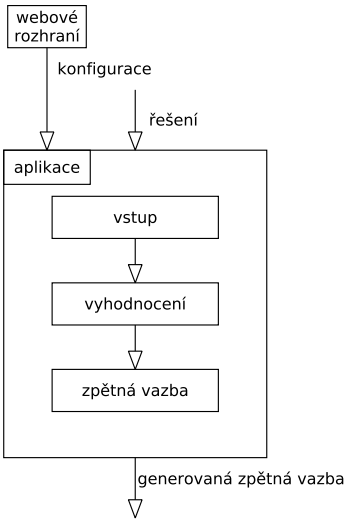
Návrh řešení

V této kapitole je představen návrh aplikace pro výuku pokročilých konstrukcí v jazyce Python. Výuka probíhá na základě generované zpětné vazby, která se skládá z několika částí, viz dále.

V první části kapitoly je naznačeno schéma celého systému spolu s popisem návrhu jednotlivých částí aplikace. V závěru kapitoly je uvedeno shrnutí spolu s diskuzí dalších možných postupů.

Navrhovaný systém přebírá a rozšiřuje některé myšlenky sytému navrženého autory Hans Fangohr a Neil O'Brien představeného v sekci 2.2.1. Zejména jde o metody testování korektnosti a stylu odevzdaného řešení. Dále některé myšlenky z oblasti zaručení bezpečnosti při spouštění cizího programu (studentského řešení). Vše je detailně popsáno u jednotlivých částí aplikace.

Obrázek 3.1 má za úkol prezentovat navrhovanou aplikaci v co největší míře obecnosti. Lze si všimnout, že aplikace se skládá ze tří hlavních částí *vstup*, *vyhodnocení* a *zpětná vazba*. Vstupem aplikace jsou studentská řešení a výstupem korespondující zpětná vazba, která je studentovi zpětně zaslána. Celý systém lze konfigurovat pomocí webového rozhraní a současně je možné ho využít jako zdroj řešení.



Obrázek 3.1: Obecná struktura navrhovaného systému

3.1 Vstup

Vstupem systému jsou studentská řešení zadaných projektů. Protože jsou tato řešení zpracovávána automaticky, je nutné definovat zadání projektu tak, aby výsledná řešení byla strojově zpracovatelná. To zahrnuje:

- **Specifikace zadání** – přesný popis problému, který má student v rámci projektu řešit. Specifikace by také měla obsahovat definice prototypů hlavních funkcí, které realizují řešení zadaného problému. Pro úspěšné vyhodnocení je nezbytné, aby prototypy funkcí ve studentském řešení odpovídaly prototypům uvedeným v testovacím skriptu.
- **Množina vstupů** – definuje možné vstupy, které jsou uvažovány při vyhodnocování.
- **Výstupy** – přesná definice výstupu společně s jeho typem (slovník, řetězec).
- **Příklady** – je dobrou praxí uvést do zadání příklady vstupů a k nim korespondujících výstupů. Na základě příkladů student může lépe porozumět specifikaci zadání.
- **Další informace** – obvykle obsahují další doporučení na literaturu, povolené knihovny a podobně.

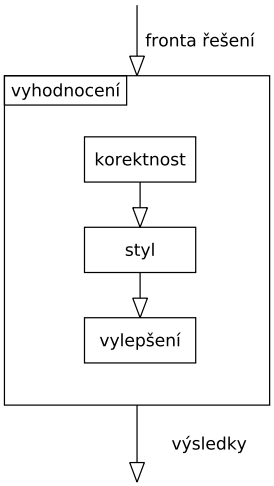
Kromě tvorby testů, které se vytvářejí na základě specifikace uvedené výše, je výhodné uvažovat další parametry projektů. Navrhované rozšiřující parametry projektů jsou popsány v následujícím výčtu.

- Definice seznamu studentů, kterým je umožněno vyhodnocovat svoje řešení. Lze realizovat na základě přihlašovacího jména studenta.
- Definice časového rozpětí, kdy lze daný projekt vyhodnocovat. Z toho vyplývá, že studentovi nebude umožněno odevzdat svoje řešení k vyhodnocení po uplynutí stanoveného času.
- Definice bodů pro jednotlivé úlohy v rámci projektu, viz níže.
- Z hlediska bezpečnosti aplikace uvažuje nastavení maximální doby, typicky několik sekund, za kterou se studenské řešení musí vyhodnotit. Dále je uvažováno nastavení prostředků operačního systému, jehož je dovoleno využívat při výpočtu řešení. Detailnější pohled na tyto parametry je diskutován v sekci 3.5.
- Nastavení nástrojů a doporučení, které mají být aplikovány na odevzdané řešení. Pomocí tohoto nastavení lze docílit specifické konfigurace každého projektu na základě zadání. Podrobnější informace jsou uvedeny v 3.2.2 a 3.2.3.
- Definice kroků, jež mají být provedeny v rámci vyhodnocování řešení. Například pro některé projekty není vyžadováno testování korektnosti, ale pouze kontrola stylu a doporučení.

Projekt se může skládat z více úloh, což umožňuje procvičení více témat v rámci jednoho souboru se zdrojovým textem.

3.2 Vyhodnocení

Tato část systému je používána pro všechny činnosti, které souvisí s vyhodnocením studentského řešení. Na vstupu je fronta řešení, tudíž jednotlivá řešení jsou vyhodnocována postupně. Základní vyhodnocení odpovídá kritériím popsaným v 2.3.4 a tvoří je kontrola korektnosti, stylu a navržení vylepšení. Grafické znázornění prezentuje obrázek 3.2.



Obrázek 3.2: Detailnější pohled na blok *vyhodnocení*

3.2.1 Korektnost

Kontrola korektnosti probíhá na základě testů, které vytvoří vyučující a následně nahraje do aplikace. Z hlediska aplikace je důležité, aby testy byly dodány v předstihu, což odpovídá metodice *test first* zmíněné v 2.3.3. Pro realizaci byl vybrán mechanismus využívající jednotkové testy, popsaný v 2.3.4. Hlavním důvodem výběru tohoto mechanismu je jeho rozšířenost a podpora v mnoha programovacích jazycích.

Navrhovaná aplikace předpokládá, že jednotlivé úlohy v projektu jsou definovány jako funkce, které lze volat z příslušného testovacího skriptu. Testovací skript pak bude obsahovat testovací sadu pro jednotlivé funkce (úlohy), která by měla pokrýt všechnu požadovanou funkcionalitu úlohy. V příkladu testovacího skriptu uvedeného níže je znázorněna testovací sada pro dvě úlohy, které jsou ve studentském řešení reprezentovány pomocí dvou funkcí: `balanced_paren` a `ceasar_list`. Studentské řešení je do testovacího skriptu importováno příkazem `import`.

```
import unittest

import proj04

class balanced_paren(unittest.TestCase):
    #Testovaci sada pro projekt 4: balanced_paren max 1b
    def test_case1(self):
```

```
        #Ocekavam true - zadne zavorky
        self.assertTrue(proj04.balanced_paren('123'))

    def test_case2(self):
        #Korektni poradi zavorek - ocekavam true
        self.assertTrue(proj04.balanced_paren('[{}]'))

    def test_case3(self):
        #Korektni poradi zavorek - ocekavam true
        self.assertTrue(proj04.balanced_paren('12<4<[a]b>>5'))

    def test_case4(self):
        #Nekorektni poradi zavorek - ocekavam false
        self.assertFalse(proj04.balanced_paren('<(>)'))

    def test_case5(self):
        #Nekorektni poradi zavorek - ocekavam false
        self.assertFalse(proj04.balanced_paren('{1<2(>3)}'))

class caesar_list(unittest.TestCase):
    #Testovaci sada pro projekt 4: caesar_list max 2b
    def test_case1(self):
        #Pouze mala pismena anglicke abecedy. Ocekavam vyjimku ValueError
        self.assertRaises(ValueError, proj04.caesar_list, "aAa", [1,2,3])

    def test_case2(self):
        #Priklad ze zadani s pretecenim
        self.assertEqual('ace', proj04.caesar_list("xyz", [3,4,5,6,7]))

    def test_case3(self):
        #Zkouska pouziti defaultniho klice
        self.assertEqual('yac', proj04.caesar_list("xyz"))

if __name__ == '__main__':
    unittest.main()
```

Kontrola korektnosti v sobě dále zahrnuje bodové hodnocení odevzdaného řešení. Bodový podíl se vztahuje ke každé úloze. V případě, že testovací sada dané úlohy selže, dostává student 0 bodů za danou úlohu. Pokud všechny funkce v testovací sadě úlohy skončí úspěchem, student dostává plný počet bodů za danou část. Celkový počet bodů pak tvoří součet všech obdržených bodů z dílčích úloh. Body pro jednotlivé úlohy lze nastavit dvěma způsoby: přímo v konfiguračním souboru daného projektu, nebo pomocí webového rozhraní.

3.2.2 Styl

Jak již bylo zmíněno v 2.3.4, je účelné kontrolovat styl psaného zdrojového textu, aby si studenti vštěpili dobré programovací návyky.

V navrhované aplikaci jsou za tímto účelem použity automatizované nástroje třetích stran podporující jazyk Python, jako je například `flake8`, zmíněné jsou rovněž zde 2.3.4.

Do této kategorie spadají i nástroje, které jsou schopny analyzovat výkonnostní metriky daného řešení jako dobu běhu programu či paměťové nároky.

Aplikace umožňuje spustit dané nástroje nezávisle na sobě s možností zvolení libovolných parametrů, viz výčet níže. Toho je dosaženo pomocí konfiguračních souborů pro každý nástroj. Tímto způsobem lze docílit obecnosti a jednoduchosti budoucích rozšíření aplikace. Navržené parametry pro nastavení nástroje jsou uvedeny níže.

- Cesta ke spustitelnému programu.
- Seznam parametrů pro tento program.
- Seznam výstupních souborů, které má daný program vygenerovat.
- Informační text, který je umístěn do zpětné vazby spolu s odkazem na výsledné soubory.

3.2.3 Navrhování vylepšení

Tato část aplikace má za úkol vygenerovat doporučení pro odevzdané studentské řešení a poskytnout odkazy na relevantní literaturu. Zde si student může o problematice přečíst více a v konečném důsledku tak zlepšit svoje znalosti, což je také jeden z cílů práce, viz 2.3.1.

Aby bylo možné generovat doporučení, je nutné aplikaci naučit, jak rozpoznat problematický zdrojový text. Jak bylo zmíněno v 2.3.2, existuje mnoho různých typů chyb, kterých se začínající programátoři mohou dopouštět, a jejich detekce není triviálním problémem. Často se nemusí jednat přímo o chyby, ale spíše o nevhodně zvolené konstrukce, řídicí struktury nebo datové struktury. S tímto faktem se snaží navrhovaná aplikace vypořádat pomocí modulární struktury, kdy báze znalostí aplikace je založena na modulech, které obsahují implementace detekcí pro různé typy chyb. Nevýhodou tohoto řešení je víceúrovňový průchod studentským řešením a delší čas zpracování. Nicméně v porovnání s obecností a rozšiřitelností, kterou toto řešení nabízí, jsou tyto nevýhody akceptovatelné. Navíc pokud jednotlivé moduly obsahují detekce více chyb stejného typu, lze dopad zmíněných nevýhod výrazně redukovat.

V následujícím textu rozšíříme klasifikaci chyb zmíněnou v 2.3.2 o některé základní typy chyb specifických pro jazyk Python [18]. Jejich přehled je uveden v tabulce 3.1.

Typ	Příklad
AttributeError	volání metody nad špatným typem objektu
SyntaxError	chybí dvojtečka za bloky **def**, **if**, **for**, nebo špatný počet otevíracích/zavíracích závorek
TypeError	použití operace nad špatným typem objektů; volání metody/funkce se špatným počtem nebo typem argumentů
IndentationError	použití tabulátorů a mezer společně
NameError	volání funkce před její definicí, chybějící **import**
IOError	otevírání souboru, který neexistuje
KeyError	vyhledávání klíče ve slovníku, který neexistuje

Tabulka 3.1: Tabulka některých druhů chyb a jejich označení v jazyce Python

Výše zmíněné chyby lze detekovat pomocí zpráv, které generuje překladač, ve většině případů jsou studenty opraveny ještě před odevzdáním projektu.

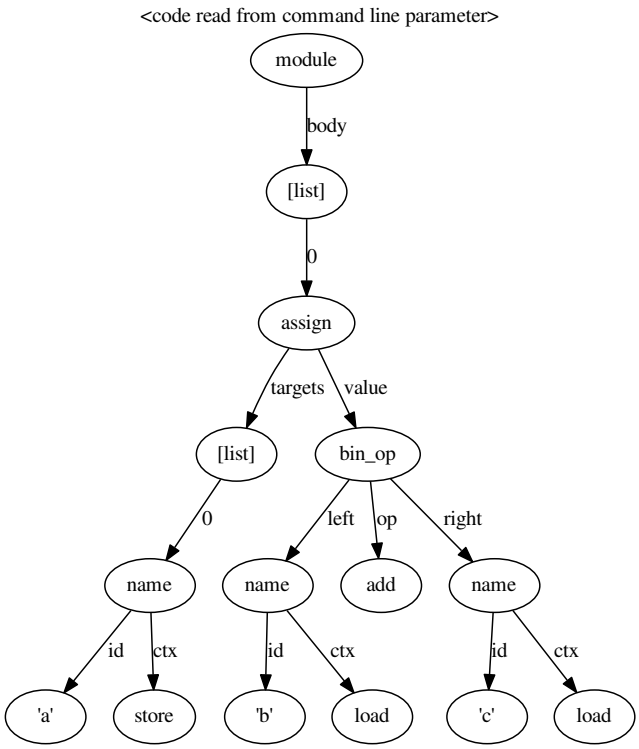
V sekci 2.3.2 byly dále zmíněny chyby, jejichž příčina spočívá v nepochopení konceptu, či neznalosti programovacího jazyka. Tyto chyby nemusí být detekovány překladačem a tudíž program funguje. Nicméně existuje pravděpodobnost vzniku skrytých chyb. Pro detekci takto zvolených konstrukcí je nutné v rámci aplikace vytvořit bázi znalostí.

Báze znalostí je vytvářena na základě studia literatury a zkušeností vyučujících. Příkladem vhodné literatury jsou knihy, které již uvažují jisté prerekvizity čtenáře, obsahující doporučení pro pokročilejší konstrukce. Jedná se o tituly [11, 2, 6]. Velké části chyb se lze vyhnout, pokud programátor uvažuje při implementaci řešení použití idiomů daného programovacího jazyka. Pro jazyk Python je k dispozici kniha [26], která obsahuje sbírku idiomatických konstrukcí a prezentuje jejich použití na příkladech.

Pro každou nevhodnou konstrukci nacházející se v bázi znalosti je nutné navrhnout detekční modul, který realizuje její detekci ve studentském kódu. V následujícím textu jsou prezentovány jednotlivé části modulu.

První částí, kterou každý detekční modul musí obsahovat, je takzvaný **detekční kód**, jehož úkolem je odhalit nevhodnou konstrukci ve studentském řešení. Tento kód lze vytvořit několika způsoby v závislosti na charakteru hledané konstrukce:

- **Analýza abstraktního syntaktického stromu (AST)** je konečný strom reprezentující syntaktickou strukturu zdrojového programu. Jeho vnitřní uzly označují operátory a listy jsou operandy [34]. Syntaktická struktura musí být vyjádřena pomocí gramatiky. Gramatika syntaxe jazyka Python je uvedena v [16]. Na základě této gramatiky lze provádět rekurzivní průchody stromem a detekovat potenciálně nevhodné konstrukce na základě vzorů. Na obrázku 3.3 je uvedena grafická reprezentace AST stromu pro jednoduché přiřazení.



Obrázek 3.3: Grafická prezentace abstraktního syntaktického stromu

V tabulce 3.2 je dále pro zadanou konstrukci uvedena AST reprezentace.

Konstrukce	AST
a = b + c	Module(body=[Assign(targets=[Name(id='a', ctx=Store())], value=BinOp(left=Name(id='b', ctx=Load()), op=Add(), right=Name(id='c', ctx=Load())))])

Tabulka 3.2: Konstrukce a k ní příslušný abstraktní strom

- **Regulární výrazy** jsou další technikou hledání nevhodných konstrukcí na základě vzorů. Regulární výraz lze definovat množinou řetězců (jazyk), které mají být v textu vybrány. Vyjadřovacím prostředkem regulárních jazyků je konečný automat [32]. V tabulce 3.3 je demonstrováno použití regulárního výrazu pro detekci nevhodné konstrukce. Příkladem je otevření souboru pro čtení. Namísto konstrukce použité v tabulce 3.3 je vhodné použít kontextový manager `with open('soubor.txt', 'r')`. Tento konstrukt zajistí, že otevřený soubor bude korektně zavřen, nehledě na to jak skončí vyhodnocení příkazů uvnitř bloku `with`.

Regulární výraz	Zdrojový kód
.*=.*open\(	fp = open('soubor.txt', 'r')

Tabulka 3.3: Konstrukce a k ní příslušný detekční regulární výraz

- **Strojové učení** je statistická metoda, jejíž cílem je získat data/vědomosti z dat. Nejčastějším příkladem uváděným v literatuře je filtr nevyžádané pošty. Hledání výskytu jednoho slova nemá takovou přesnost korektního odhalení nevyžádané pošty, jako když se uvažuje souvislost mezi vícero slovy. Odhalení této souvislosti lze dále kombinovat s dalšími faktory (např. délka zprávy) a na základě takto sestaveného kontextu by systém měl být schopný rozhodnout, zda se jedná o vyžádanou, či nevyžádanou poštu. Další aplikace strojového učení lze nalézt v systémech určených pro rozpoznávání řeči nebo v systémech pro přepis ručně psaného textu. Aby všechny zmíněné aplikace strojového učení fungovaly korektně, je nejdříve nutné systém učit. Pro učení se používá množina trénovacích dat. Existují dva základní přístupy: učení s učitelem a bez učitele. V případě učení s učitelem je pro vstupní data znám správný výstup. V případě výše popsaného filtru nevyžádané pošty jsou výstupy **je spam/není spam**. Metoda učení bez učitele znalost výstupu nemá. Další dělení se vztahuje k aktivitě/pasivitě učícího se systému. Aktivní systém komunikuje s prostředím, pokládá otázky. Pasivní systém pouze čeká na informace, které dostane od prostředí. Základní fáze procesu učení jsou: shromáždění dat, příprava vstupních dat, analýza vstupních dat, trénování algoritmu, testování algoritmu a použití algoritmu [23, 38].

Je-li nevhodná konstrukce nalezena, je zapotřebí ji popsat a uložit za účelem pozdějšího zpracování. Pro pozdější zpracování konstrukce je nutné uchovat následující informace:

- **Číslo řádku**, na kterém byla nevhodná konstrukce lokalizovaná v odevzdaném řešení.
- **Nadpis** slouží pro obecný popis nalezené konstrukce.
- **Vysvětlení** je určeno k detailnímu popisu, proč systém považuje nalezenou konstrukci za nevhodnou.

- **Odkazy** na doplňující materiály.
- **Závislosti** určují prerekvizitní znalosti, které by studenti měli znát, aby byli schopni dané problematice porozumět.

3.2.4 Detekce plagiátů

S rostoucím počtem studentů roste i pravděpodobnost výskytu plagiátů. Autoři [22] diskutují příčiny, proč se studenti uchylují k opisování. Dále v textu uvádí porovnání dostupných nástrojů pro detekci podobností ve zdrojovém textu. Následující text vychází z výše uvedené práce.

V obecné rovině lze plagiát implementace zdrojového textu popsat jako snahu vydávat převzatou implementaci, nebo její část, za svoje vlastní dílo bez uvedení zdroje. Často se lze s plagiáty setkat na univerzitách v rámci řešení projektů. Nejčastější příčinou opisování je čas, množství projektů či neznalost. Lze rozlišit plagiátorství úmyslné a neúmyslné. Manuální kontrola odevzdaných projektů v případě ročníků o stovkách studentů a při několika zadaných projektech za semestr je nemyslitelná. Zde vzniká potřeba automatizovaných nástrojů.

Autoři práce dále vysvětlují implementační strategie, které jsou využívány při tvorbě takovýchto nástrojů. První velmi rozšířenou strategií je transformace odevzdaného zdrojového kódu na lexikální jednotky (identifikátory, smyčky). Při této transformaci dojde k odstranění takových částí zdrojového textu, které nemají vliv na sémantiku řešení (komentáře). Lexikální jednotky jsou pak porovnávány mezi sebou a jsou vyhledávány podobnosti. Další strategie je založena na porovnání struktury programů, kdy jsou používány algoritmy pro pokročilé porovnávání řetězců. Testované nástroje byly například porovnávány z hlediska podporovaných jazyků, rozšiřitelnosti, prezentace výsledků či dostupnosti. Ze všech uvedených nástrojů pouze jeden obsahuje nativní podporu jazyka Python. Jedná se o nástroj *MOSS*[1], který je vyvíjen na Stanfordské univerzitě. Tento nástroj je implementován jako webová služba, kdy studentská řešení jsou nahrána na výpočetní server. Server provede porovnání řešení a jako odpověď zašle odkaz na webové stránky obsahující grafickou vizualizaci výsledků. Nástroj *MOSS* porovnává řešení na základě otisků jednotlivých souborů obsahujících implementaci. Linuxová platforma je rovněž podporována, stejně tak jako nastavení citlivosti detekce [37].

Na obrázku 3.4 je uveden příklad vyhodnocení dvou studentských řešení, které simulují dvě nejčastější techniky při opisování: přidání/odebrání/změnu komentářů nebo změnu názvu proměnných.

[1] Domovská stránka projektu: `http://theory.stanford.edu/~aiken/moss/`

moss_test/xnovak00.py (97%)		moss_test/xnovak01.py (97%)	
4-20		3-18	

moss_test/xnovak00.py

```
#!/usr/bin/env python3


class TooManyCallsError(Exception):
    """Exception"""
    pass


def limit_calls(max_calls=2, error_message_tail='called_too_often'):
    def decorator(function):
        def limited_function(*args, **kwargs):
            limited_function.calls += 1  #call counter increment
            if limited_function.calls <= max_calls:  #call counter test with max_calls
                return function(*args, **kwargs)
            else:
                error_message = 'function "{0}" - '.format(function.__name__) + error_message_tail
                raise TooManyCallsError(error_message)
        limited_function.calls = 0 #call counter init
        return limited_function
    return decorator
```

moss_test/xnovak01.py

```
#!/usr/bin/env python3


class TooManyCallsError(Exception):
    """Exception for calling method too many times"""
    pass

def limit_calls(max_calls=2, error_message_tail='called too often'):
    def limit_calls_decorator(function):
        def function_call(*args, **kwargs):
            function_call.calls += 1        #increment call counter
            if function_call.calls <= max_calls:        #check if function should be called or exception raised
                return function(*args, **kwargs)
            else:
                specific_error_message = 'function "{0}" - '.format(function.__name__) + error_message_tail
                raise TooManyCallsError(specific_error_message)         #raise exception
        function_call.calls = 0         #set call counter
        return function_call
    return limit_calls_decorator
```

Obrázek 3.4: Příklad vyhodnocení plagiátů nástrojem MOSS

Jelikož jsou studentská řešení nahrávána na cizí server, je nezbytné zajistit ochranu a soukromí odevzdaných řešení. V přehledu nejčastějších otázek uživatelů k používání nástroje[2] je uvedeno, že tyto problémy nástroj řeší generováním unikátních odkazů vedoucích k výsledné analýze. Rovněž zavádí časovou platnost odkazů, která je stanovena na 14 dní.

Autoři dále zdůrazňují potřebu manuální kontroly řešení, která jsou nástrojem označena jako plagiáty. Tato kontrola má zajistit, aby nedošlo k neoprávněnému obvinění daných studentů [1].

[2]K nalezení zde http://moss.stanford.edu/general/faq.html

3.3 Struktura zpětné vazby

Očekávaným výstupem této práce je textová zpětná vazba, která je zaslána studentovi po vložení jeho řešení do systému. V kapitole 2.3.5 byly diskutovány dvě techniky poskytování zpětné vazby. Z hlediska charakteru systému, kdy je nutné generovat statickou zpětnou vazbu bez možnosti přímého dialogu se studentem, je možné uvažovat pouze techniku `Giving-Answer`. Pokud tuto metodu vztáhneme k problematice zpětné vazby, která je vyžadována v této práci, forma a struktura generovaných výstupů by měla mít takovou vypovídající hodnotu, aby dokázala odpovědět na všechny potenciální otázky kladené studentem. Dále je nezbytné zajistit dobrou logickou stavbu a přehlednost.

Z těchto požadavků plyne, že zpětná vazba tvořená pouze generovaným prostým textem by nebyla dostačující. Nabízí se použití značkovacích jazyků, jejichž použití zvýší vyjadřovací schopnost aplikace. Za tímto účelem se jeví jako vhodná možnost jazyk HTML[3]. Díky velké rozšířenosti tohoto jazyka značné množství nástrojů pro analýzu zdrojových textů podporuje generování výsledných reportů právě do formátu HTML. Lze tedy poměrně snadno tyto reporty začleňovat do výsledné zpětné vazby. Jelikož zpětná vazba je modelována jako webové stránky, je možné vytvořit dobrou logickou stavbu. Nyní bude uveden seznam požadavků na strukturu zpětné vazby.

- **Vyhodnocení testů** poskytuje přehled testů, na základě kterých byla kontrolována korektnost řešení. V případě neúspěšných testů má za úkol zobrazit detaily, proč daný test neuspěl.
- **Vyhodnocení stylu** – hlavním smyslem této části zpětné vazby je předat studentovi konkrétní doporučení k zapsanému zdrojovému textu z hlediska stylu. Jedná se například o doporučení pro odsazování či detekci nepoužitých proměnných.
- **Doporučení** – z hlediska použitých konstrukcí konkrétně označených na odevzdaném zdrojovém textu. Každá detekovaná konstrukce musí povinně obsahovat vysvětlující text spolu s relevantními odkazy, kde student může nalézt další informace. V této části by zpětná vazba měla dále obsahovat obecná doporučení (tipy), která lze uplatit pro implementaci/návrhu jakéhokoli problému.

Na základě těchto kriterií byl vytvořen návrh formy zpětné vazby, která se sestává z celkem pěti HTML stránek.

Obrázek 3.5 ukazuje úvodní stránku celé zpětné vazby, která obsahuje základní informace o odevzdaném řešení společně s počtem získaných bodů. Důležitou částí je sekce *Poznámky*, která je vytvářena dynamicky v závislosti na nastavení použitých nástrojů v daném projektu. Zpravidla tato sekce zahrnuje seznam poznámek s odkazy na soubory, které obsahují výsledky zvolených nástrojů. Více informací lze najít v kapitole 3.2.2. Výjimku zde tvoří nástroj `flake8`, který je uvažován implicitně a má svoji vlastní HTML stránku. Detail stránky je zobrazen na obrázku 3.7.

[3]Specifikaci lze nalézt: https://html.spec.whatwg.org/multipage/

Přehled Testy Styl Doporučení Tipy

Shrnutí odevzdaného řešení

Login	xnovak00
Projekt:	proj04
Datum odevzdání:	17:54:44 05.05.2017
Získáno bodů:	5

Detaily řešení si můžete prohlédnout v záložkách výše.

Poznámky:

- Vaše řešení bylo testováno na pokrytí kódu tzn.procento zdrojového kódu, který byl proveden během spuštění aplikace. Výsledky si můžete vizualizovat zde.
- Vaše řešení bylo profilováno. Výsledek si můžete stáhnout zde. Řešení si můžete vizualizovat pomocí aplikace pyprof2calltree. Návod naleznete zde"

Vaše řešení můžete vizualizovat a ladit zde.

Obrázek 3.5: Úvodní stránka zpětné vazby

První stránka obsahuje informace o korektnosti odevzdaného řešení na základě provedených testů. Výsledky jsou zobrazeny v přehledné podobě pomocí tabulky. V případě, že některý test skončí neúspěšně, je tato skutečnost indikována označením příslušného testu červenou barvou. U neúspěšných testů jsou také uvedeny detaily, které blíže specifikují důvody neúspěchu testu. Obrázek 3.6 zobrazuje takto navrženou stránku.

Přehled Testy Styl Doporučení Tipy

Overview

Class	Fail	Error	Skip	Success	Total
test_proj04.balanced_paren	0	0	0	5	5
test_proj04.caesar_list	0	0	0	4	4
test_proj04.caesar_varnumkey	1	0	0	3	4
Total	1	0	0	12	13

Failure details

test_proj04.caesar_varnumkey (1 failures, 0 errors)

test_case5: builtins.AssertionError

Traceback

Details

```
'yac' != 'xyz'
- yac
+ xyz
```

All tests

test_proj04.balanced_paren (0 failures, 0 errors)

- test_case1
- test_case2
- test_case3
- test_case4
- test_case5

Obrázek 3.6: Stránka obsahující přehled provedených testů

Přehled | Testy | Styl | Doporučení | Tipy

PEP 8 Report

PEP8 Errors: 11, PEP8 Warnings: 2, Other: 1

proj04.py

Line	Code	Pos	Description
9	F401	1	'string' imported but unused
14	E501	80	line too long (83 > 79 characters)
14	W291	84	trailing whitespace
22	E501	80	line too long (88 > 79 characters)
27	E501	80	line too long (100 > 79 characters)
36	E501	80	line too long (96 > 79 characters)
38	E501	80	line too long (98 > 79 characters)
48	E231	27	missing whitespace after ','
48	E501	80	line too long (85 > 79 characters)
52	E501	80	line too long (94 > 79 characters)
59	E501	80	line too long (96 > 79 characters)
61	E501	80	line too long (95 > 79 characters)
67	E501	80	line too long (87 > 79 characters)
68	W391	1	blank line at end of file

Statistics

1. E501: 10
2. F401: 1
3. W291: 1
4. E231: 1
5. W391: 1

Obrázek 3.7: Vyhodnocení stylistických vlastností řešení

Třetí HTML stránka má název „*Doporučení*“ a je navržena k prezentaci nalezených nevhodných konstrukcí. Řádek, který obsahuje nalezenou konstrukci, je podbarven žlutou barvou. Na konci řádku se nachází tlačítko s názvem „*Klikni!*“. Po kliknutí na toto tlačítko je zobrazen vysvětlující text. Tento text splňuje kritéria navržená v sekci 3.2.3. Detail stránky společně s ukázkou popisu dvou doporučení je uveden na obrázku 3.8.

Přehled | Testy | Styl | Doporučení | Tipy

Pokud se Vám nezobrazuje u žlutě označených řádků tlačítko "Klikni!". Zkuste prosím načíst stránku ještě jednou.

```
 1. import sys                Klikni!
 2. from os import *                Klikni!
 3.
 4. def test_doc_string():
 5.     pass
 6.
 7. def test_regexp():
 8.     a = [1, 2, 3]
 9.     while len(a):
10.         print('Loop')
11.         break
12.     fp = open('file.txt', 'w')          Klikni!
13.     for n in range(len(a)):
14.         print(n)
15.     try:
16.         a = b/0
17.     except:              Klikni!
18.         pass
19.     b = 'a'
20.     string = b + 'b'
21.     name = 'Pavel'
22.     score = 10
23.     print("Total score for %s is %d" % (name, score))
24.     if type(a) == list:         Klikni!
25.         print("Found list")
26.     if a == None:               Klikni!
27.         print("a is None")
28.
29. def test_list_comph():
30.     a = [10, 10, 20]
31.     b = []
32.     for x in a:
33.         if x == 10:
34.             b.append(x)         Klikni!
35.
36. class StaticMethod(objec
37.     def can_be_static_me
38.         return a + b
39.
40. def test_remove():
41.     a = dict()
42.     del a['pokus']
43.
44. def test_nested_loops():                Klikni!
45.     a = [1]
46.     b = [2]
47.     c = [3]
48.     for x in a:          Klikni!
```

Kontrola importů
Pro modul os definujte, které funkce mají být importovány. Případně definujte alias, nebo použijte přímo import module_name.
Budete tak vždy vědět z jakého modulu je která funkce volána
Link: 1

Otevření souboru
Pro otevření souboru je vhodné použít context manager: with open(file, mode):.
Link: 1
Link: 2

Kontrola None
Pro kontrolu hodnoty None pro daný objekt jsou v Pythonu ustálené konstrukce if object is None nebo if object is not None.
Link: 1

List comprehension
Zkontrolujte, zda pro inicializaci seznamu lze použít list comrehension
Link: 1
Link: 2

Chybějící komentáře u funkcí, tříd
Definujte komentář(docstring) zajistíte tak lepší čitelnost vašeho kódu.
Link: 1

Kontrola dekorátoru pro statické metody
Metoda by měla být uvozena @staticmethod dekorátorem
Link: 1
Link: 2

Obrázek 3.8: Doporučení k odevzdanému zdrojovému kódu

Na této stránce se dále nacházejí odkazy na doplňující materiály, které spadají do kontextu nalezené konstrukce.

Náplní poslední stránky „*Tipy*“ je zprostředkovat obecná doporučení společně s příklady, které studenti mohou využít při implementaci dalších projektů. Výhodou této stránky je, že na ni lze umístit konstrukce, pro které není implementován detekční modul. Příklad stránky je uveden na obrázku 3.9.

Přehled **Testy** **Styl** **Doporučení** **Tipy**

```
1. for user in users:
2.         if check_login(user):
3.                 print('User is not valid!')
4.                         break
5. else:
6.         print('All users are valid!')
```

Pro konstrukci for lze použít konstrukt "else", jehož kód bude proveden v případě, že smyčka for je úspěšně provedena.

```
1. (login, access, _, _) = get_student_info(student)
2. if access:
3.         output = 'Student login: {login} does not access'.format(login=login)
```

Pokud nepotřebujete v daném bloku kódu všechnz prvky tuple, který vrací vaše funkce můžete tyto prvky ignorovat pomocí _

```
1. def funkce(student):
2.         return (student.login, student.age, student.access)
```

Funkce může vracet tuple obsahující vícero prvků

```
1. class Celsius:
2.     def __init__(self, temperature = 0):
3.         self._temperature = temperature
4.
5.     def to_fahrenheit(self):
6.         return (self.temperature * 1.8) + 32
7.
8.     @property
9.     def temperature(self):
10.        print("Getting value")
11.        return self._temperature
12.
13.    @temperature.setter
14.    def temperature(self, value):
15.        if value < -273:
16.            raise ValueError("Temperature below -273 is not possible")
17.        print("Setting value")
18.        self._temperature = value
```

Použití @property dekorátoru namísto getter/setter funkcí. Použitím této konstrukce lze v pozdějších fází vývoje aplikace jednoduše modifikovat hodnoty např. třídních proměnných

Obrázek 3.9: Obecná doporučení k implementaci

3.4 Webové rozhraní

Webové rozhraní má dva hlavní úkoly z hlediska funkcionality. Prvním úkolem je poskytnout administrátorovi možnost konfigurovat aplikaci. Druhým úkolem je zajistit pro studenty možnost vložení jejich řešení do systému. Z toho plyne, že webové rozhraní má celkem dva typy uživatelů: studenta a administrátora. Funkcionalita je popsána příslušným UML[4] diagramem užití.

3.4.1 Případy užití

Jak bylo zmíněno výše, systém má celkem dva účastníky. Prvním je student, rovněž ho lze definovat jako standardního uživatele bez speciálních práv, kterému je dovoleno pouze odevzdání vypracovaného řešení. K tomuto účelu je navržen formulář, do kterého lze vložit řešení. Řešením se rozumí textový soubor obsahující zdrojový text vypracovaného projektu v jazyku Python. Takto odevzdané řešení musí dále splňovat zadanou konvenci pro název souboru. Nutnost použití konvence je opět dána strojovým zpracováním, kdy je nutné z názvu souboru vyčíst informace, které jsou kritické pro další zpracování. V navrhované aplikaci byla zavedena následující konvence **isj_<projekt>_<login>.py**, kde:

- **projekt** – reprezentuje jméno projektu, jehož řešení odevzdaný soubor obsahuje.
- **login** – jednoznačný identifikátor studenta.

[4]Specifikaci jazyka lze nalézt v http://www.uml.org/

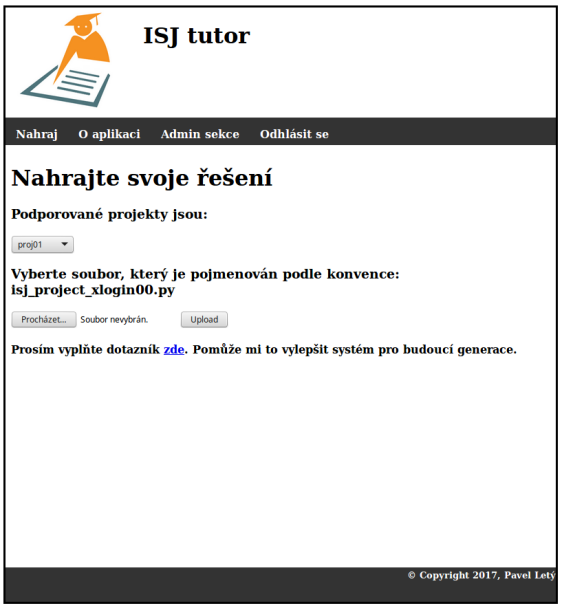
Nutností je, aby webové rozhraní bylo schopno zajistit kontrolu takto zvolené konvence. Lze tak předejít omylům při odevzdávání řešení.

Druhým účastníkem systému je administrátor, který je ve většině případů reprezentován vyučujícím. Možnosti administrátora v rámci webového rozhraní jsou znatelně vyšší než pro studenta. Jedná se zejména o konfiguraci aplikace. Pro potřeby konfigurace je nutné vytvořit speciální sekci webových stránek s omezeným přístupem. Takto vytvořená sekce poskytne potřebné rozdělení rolí mezi studentem a administrátorem. Omezení přístupu lze realizovat na základě znalosti hesla.

Diagram 3.10 popisuje navrhované případy užití webového rozhraní. Významné případy užití jsou pro jednotlivé účastníky popsány.

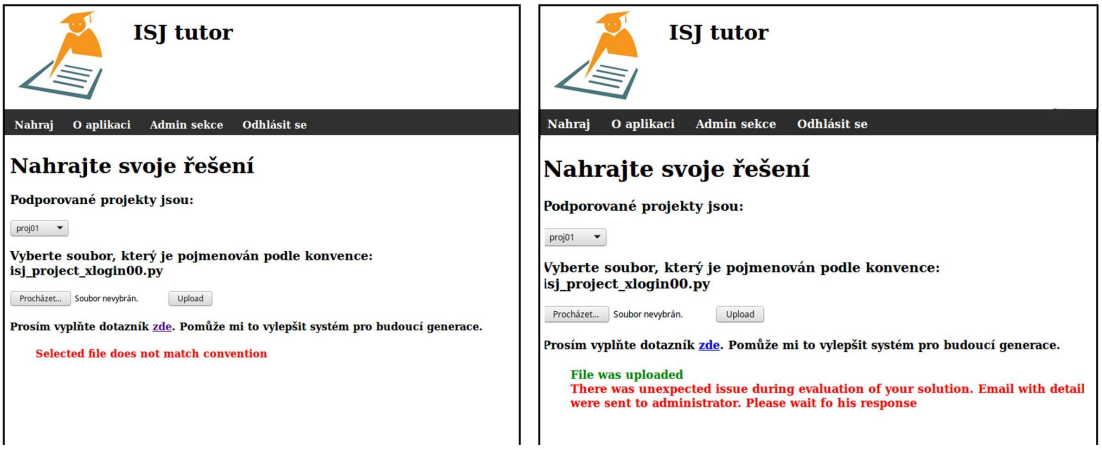
Obrázek 3.10: Diagram případu užití

Na obrázku 3.11 je prezentován formulář, který studenti používají pro odevzdání svého řešení. Soubor určený pro vložení do formuláře musí splňovat konvenci zavedenou zde 3.1.



Obrázek 3.11: Formulář pro odevzdání řešení

Pokud soubor tuto konvenci nesplňuje, je vypsána chyba. V případě, že dojde během zpracovávání řešení k chybě, je tato skutečnost rovněž indikována uživateli. Zároveň je zaslán email na adresu uvedenou v konfiguraci aplikace obsahující podrobnost chyby a studentské řešení, které chybu způsobilo. Příklad emailové zprávy je přiložen v příloze C. Oba zmíněné chybové stavy prezentuje obrázek 3.12.

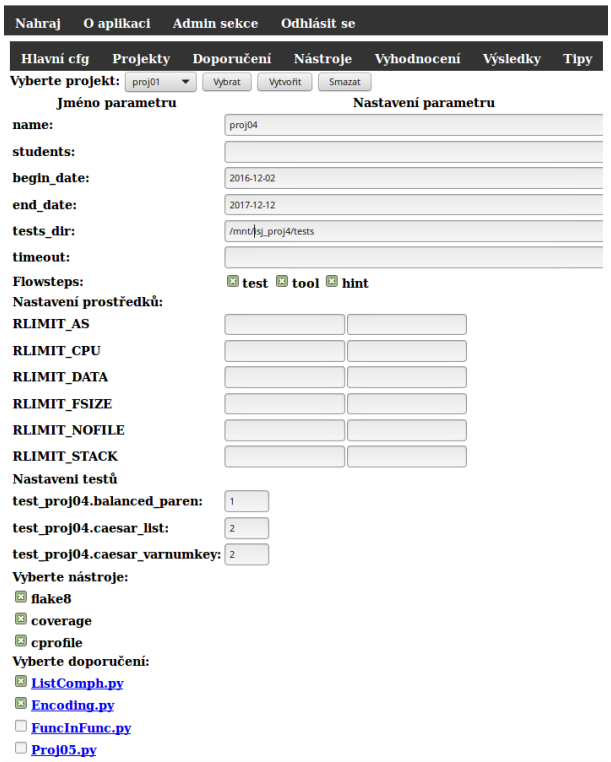


Obrázek 3.12: Levá část obrázku zobrazuje chybový stav, který nastane pokud jméno odevzdaného souboru xpokus00.py nesplňuje zadanou konvenci. V pravé části je zobrazena indikace chyby nalezené při zpracovávání řešení

Za účelem konfigurace je ve webovém rozhraní navržena speciální sekce, do které lze vstoupit pouze na základě znalosti hesla. Heslo je z bezpečnostních důvodů uloženo v souboru s vhodně zvolenými přístupovými právy. Tato sekce poskytuje možnost plnohodnotné konfigurace aplikace bez nutnosti manuálního přístupu do konfiguračních souborů. Přihlašovací formulář je uveden na obrázku 3.13.

Obrázek 3.13: Přihlašovací formulář

Administrátorská sekce zpřístupňuje veškerou funkcionalitu účastníka „*Správce*“ v diagramu užití uvedeného výše 3.10. Tato sekce obsahuje celkem sedm záložek, z nichž ústřední částí je záložka „*Projekty*“. Ta obsahuje souhrn všech nastavení pro zadané projekty. Ukázkové nastavení projektu prezentuje obrázek 3.14. Lze si všimnout, že záložka je složena z několika částí, které odpovídají kritériím představených v 3.1.



Obrázek 3.14: Záložka obsahující nastavení projektu

Na začátku stránky se nacházejí obecná nastavení.

- `name` – jméno projektu
- `students` – jednoznačné identifikátory studentů, kteří jsou oprávněni vkládat řešení.
- `begin_date` – počáteční datum, od kterého je možné vkládat řešení pro vyhodnocení.
- `end_date` – koncové datum, do kterého je možné vkládat řešení.

- `tests_dir` – složka obsahující testy pro daný projekt.
- `timeout` – čas v sekundách, který je přidělen procesu vykonávajícímu vyhodnocení korektnosti řešení.
- `flowsteps` – jsou hlavní kroky vyhodnocení, které mají být v rámci projektu provedeny.

Další nastavení zahrnuje konfiguraci prostředků operačního systému přidělenou pro daný projekt. Jedná se o bezpečnostní nastavení zabraňující útokům a případným specifickým chybám (zacyklení). Podrobnější informace jsou diskutovány v sekci 3.5. Následující konfigurace se týká nastavení bodového hodnocení úloh. Hodnocení lze zadat ke všem úlohám, které jsou v rámci testovacích skriptů uložených v *tests_dir* zapsány jako `unittest.TestCase`[5]. Tato problematika byla uvedena v sekci 3.2.1. Následují dva zaškrtávací seznamy, kde lze zvolit doporučení a nástroje, jež jsou použity během procesu vyhodnocování studentského řešení. Dílčí nastavení pro nástroje a doporučení lze najít ve stejnojmenných záložkách. Na obrázku 3.15 jsou zobrazeny možnosti pro konfiguraci nástroje. Položky nastavení odpovídají navrhovanému řešení v sekci 3.2.2.

Obrázek 3.15: Možnosti pro nastavení nástroje

Záložka *Doporučení* má dva hlavní cíle. Prvním cílem je umožnit administrátorovi nahrát soubor s detekčním modulem. Struktura detekčního modulu byla diskutována v sekci 3.2.3. Druhý cíl zahrnuje správu závislostí. Závislosti jsou tvořeny z tematických okruhů, které tvoří logické celky při výuce programovacího jazyka Python (funkce, datové struktury, generátory). K jednotlivým tématům lze skrze webové rozhraní přiřadit odkazy na literaturu, kurzy, dokumentaci a zároveň lze definovat prerekvizitní témata nutná pro porozumění dané problematice. Po přidání příslušných okruhů témat do systému je možné přiřadit témata přímo k příslušným detekčním modulům. Tento mechanismus umožňuje tematicky vymezit nevhodné konstrukce a poskytnout tak nejen vysvětlující materiály k dané konstrukci, ale i k celému tematickému celku a případným dalším prerekvizitním okruhům. Nastavení závislostí mezi tematickými okruhy může být inspirováno pořadím výkladu jednotlivých okruhů v literatuře [11, 2, 6], nebo z obsahů online kurzů[6].

[5] Více viz https://docs.python.org/3/library/unittest.html
[6] Například: https://www.codecademy.com/learn/python

Pro úplnost je uveden příklad. Nechť je detekována nevhodná konstrukce, která spadá do kategorie funkcí. Systém zkontroluje závislosti pro kategorii *Funkce* a načte všechny prerekvizitní kategorie společně s nastavenými odkazy. Tyto odkazy jsou posléze umístěny do karty *Doporučení* ve výsledné zpětné vazbě.

Administrátorská sekce v záložce *Hlavní cfg* dovoluje konfigurovat všechna potřebná nastavení nutná k chodu aplikace. Jednotlivé položky konfigurace jsou popsány v implementační dokumentaci na přiloženém DVD. Záložka *Vyhodnocení* zobrazená na obrázku 3.16 dovoluje vyučujícímu vyhodnotit všechny projekty najednou bez nutnosti postupného vkládání projektů přes formulář na obrázku 3.11. Zároveň je vyučujícímu pomocí volby *Zkontrolovat podobnost řešení* dovoleno zapnout kontrolu podobnosti řešení. Pro kontrolu podobnosti je využit algoritmus *MOSS* blíže přiblížený v sekci 3.2.4. Projekty musí být zabaleny v archivu a je nezbytné, aby jednotlivá řešení splňovala konvenci definovanou pro název odevzdaného souboru zavedenou v sekci 3.1. Ve vloženém archivu mohou být řešení všech nastavených projektů. Vyhodnocování pak probíhá v souladu s nastavením každého projektu stejným způsobem jako v případě, kdy je řešení vloženo přes formulář pro odevzdání.

Obrázek 3.16: Formulář pro vyhodnocení balíku řešení

Výsledek celého procesu je zobrazen na obrázku 3.17. Skládá se z tabulky obsahující identifikátor studenta, jméno projektu a bodového hodnocení. Identifikátor studenta je navrhován jako tlačítko, které po stisknutí otevře zpětnou vazbu v novém panelu internetového prohlížeče. Ta odpovídá zpětné vazbě, kterou dostává student. Pod tabulkou jsou umístěna dvě tlačítka: *Stáhnout CSV* a *Zobrazit podobnost řešení*. V případě prvního tlačítka lze stáhnout výsledné bodové hodnocení ve formátu CSV (Comma-separated values). Druhé tlačítko odkazuje na stránky, kde jsou umístěny výsledky analýzy podobnosti odevzdaných řešení. Příklad této stránky je uveden na obrázku 3.4.

Obrázek 3.17: Vyhodnocená řešení

Záložka *Výsledky* je určena pro mazání, či stahování vygenerovaných výsledků. Vyučující může vyhodnotit všechna řešení, stáhnout archiv s výsledky a ty rozeslat studentům.

Poslední záložku tvoří *Tipy*. Jak bylo uvedeno 3.3, obecná doporučení (tipy) jsou určeny k vysvětlení obecných konceptů. Protože jsou tato obecná doporučení zařazována do každé vygenerované zpětné vazby, tak k nim student má neustálý přístup. To zvyšuje pravděpodobnost, že některý z uvedených konceptů použije při řešení dalších projektů. Jednotlivé tipy se skládají ze dvou částí:

- **Soubor** – obsahuje příklady zdrojového kódu vysvětlující koncept/konstrukci. Zdrojový text je vhodné komentovat.
- **Popis** – zahrnuje doplňující informace k uvedenému příkladu.

Po vložení tipu dojde k jeho přidání do tabulky, kde je příslušné pole podbarveno červenou barvou. Ta značí, že daný tip zatím není přidán ve výsledném HTML souboru, který se vkládá do zpětné vazby. Pro vložení tipu do HTML stránky je nezbytné daný tip označit pomocí zaškrtávacího pole a tlačítkem *Přidat* potvrdit. Pole v tabulce se podbarví zelenou. Posledním krokem je stisknutí tlačítka *Vygenerovat*, které dokončí proces vytváření HTML stránky. Tento mechanismus má poskytnout vyučujícímu kontrolu nad aktuálně zobrazovanými tipy ve zpětné vazbě. Například za účelem dočasné deaktivace některých tipů.

3.5 Bezpečnost

Z kritérií uvedených v 2.3.4 plyne nutnost návrhu bezpečnostních opatření. Studentská řešení jsou při analýze používána dvěma způsoby:

- Spuštění studentského řešení za účelem ověření korektnosti.
- Použití studentského řešení jako argumentu pro detekční moduly, nástroje, které vykonávají statickou analýzu.

Tato opatření mají za cíl chránit aplikaci a její okolí před bezpečnostními riziky, které jsou spojeny se spouštěním studentských řešení. Příkladem mohou být nekonečné smyčky, použití příkazů pro mazání nebo neefektivní datové struktury.

V navrhované aplikaci jsou aplikována následující opatření dle 2.3.4:

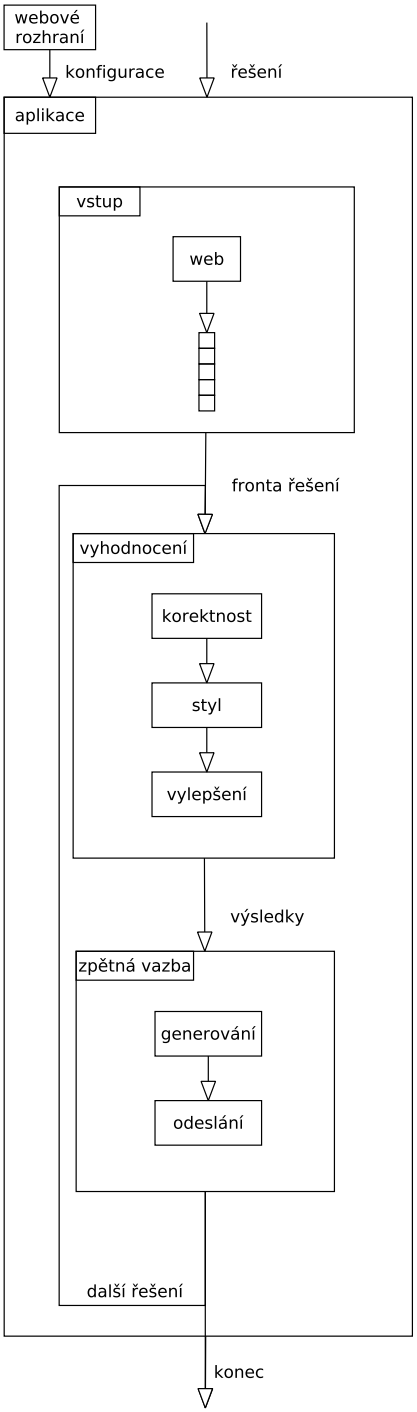
- **Uživatel a skupina** – pro spuštění aplikace se doporučuje vytvořit nového uživatele a případně i skupinu, které jsou dedikovány pouze pro účely provozu aplikace. Toto řešení umožňuje aplikaci pracovat pouze se soubory, které sama vytvořila. Není tedy možné, aby došlo například ke smazání souborů ostatních uživatelů hostitelského stroje při vyvolání mazacího příkazu z odevzdaného řešení. Pro vytvoření skupiny nebo uživatele jsou v Linuxu definovány příkazy `groupadd` a `useradd`.
- **Nastavení minimálních práv** - jsou minimální práva udělená pro adresáře, které jsou v rámci aplikace využívány. V Linuxu je možné práva nastavit přes program `chmod`.
- **Pískoviště (Sandbox)** – je složka, do které je umístěno studentské řešení před spuštěním. Na tuto složku jsou aplikovány restrikce uvedené v předchozích bodech. Studentské řešení je izolováno pouze v této složce a není mu dovoleno vykonávat jakoukoli činnost mimo ni.

- **Nastavení systémových prostředků** – předchozí opatření vymezují prostor pro spuštění aplikace. Kromě těchto restrikcí je důležité přesně vymezit maximální systémové prostředky, které mohou být při vyhodnocování studentského projektu alokovány. Tyto limity se vztahují zejména na nastavení maximální doby běhu programu, maximální využití diskového prostoru a virtuální paměti [14]. Pomocí těchto limitů lze zajistit, že řešení obsahující skryté chyby nebudou blokovat prostředky na výpočetním serveru. V Linuxu lze tyto nastavení vynutit příkazem `ulimit`.

3.6 Shrnutí

V této kapitole byl popsán výsledný stav návrhu aplikace pro výuku pokročilých konstrukcí v jazyku Python. Obrázek 3.18 prezentuje celkové schéma navrhované aplikace, které se sestává ze všech výše popsaných části.

Vstupem aplikace jsou studentská řešení a konfigurace aplikace. Řešení jsou vkládána do aplikace skrze formulář představený na obrázku 3.11. Proces vyhodnocení je podmíněn konfigurací jednotlivých projektů. Ta mimo jiné zahrnuje nastavení nástrojů, detekčních modulů, bezpečnostních opatření a kroků, které se mají v rámci vyhodnocení provést. Hlavní kroky jsou celkem tři. Prvním krokem je vyhodnocení korektnosti popsané v 3.2.1. Pro kontrolu korektnosti byla vybrána metoda využívající tzv. jednotkové testy. Druhý krok poskytuje možnost spustit nástroje třetích stran, či vlastní skripty nad odevzdaným řešením a tím rozšířit zpětnou vazbu o materiály specifické pro daný projekt. Tato problematika byla diskutována zde 3.2.2. Třetí krok hledá v odevzdaném řešení nevhodně použité konstrukce. Ty jsou definovány na základě detekčních modulů. Hlavní zdroj detekčních modulů představuje literatura, idiomy jazyka Python, či zkušenosti vyučujících. Implementace modulů je v závislosti na charakteru konstrukce realizována prostřednictvím analýzy AST, nebo regulárních výrazů viz 3.2.3.



Obrázek 3.18: Kompletní schéma systému

Kapitola 4

Implementace

V této kapitole je popsána implementace celého systému pro generování zpětné vazby na základě odevzdaných studentských projektů definovaná v kapitole 3. Celý systém se skládá ze dvou hlavních částí. První část tvoří aplikace realizující samotnou analýzu odevzdaného studentského řešení. Druhou část tvoří webové rozhraní, které poskytuje možnost odevzdání řešení a možnost konfigurace aplikace vyučujícím.

Na začátku kapitoly jsou uvedeny nástroje využité při implementaci řešení. V dalším průběhu kapitoly následuje postupně popis implementace aplikace a popis implementace webového rozhraní.

4.1 Použité nástroje

Pro implementaci aplikace byl vybrán jazyk Python[1] ve verzi 3.6. Důvodem výběru tohoto skriptovacího jazyka je jeho rozšířenost, dostupná dokumentace a podpora objektově orientovaného přístupu k návrhu. Python dále obsahuje podporu jednotkových testů 3.2.1, které jsou důležitým předpokladem pro implementační jazyk.

Pro tvorbu programové dokumentace je použit nástroj Sphinx[2]. Aplikace je schopna automatizovaně generovat dokumentaci ze zdrojových kódů. Generování probíhá na základě komentářů uvedených přímo ve zdrojovém textu. Tyto komentáře mají definovanou strukturu značek popsanou jazykem `reStructuredText`[3].

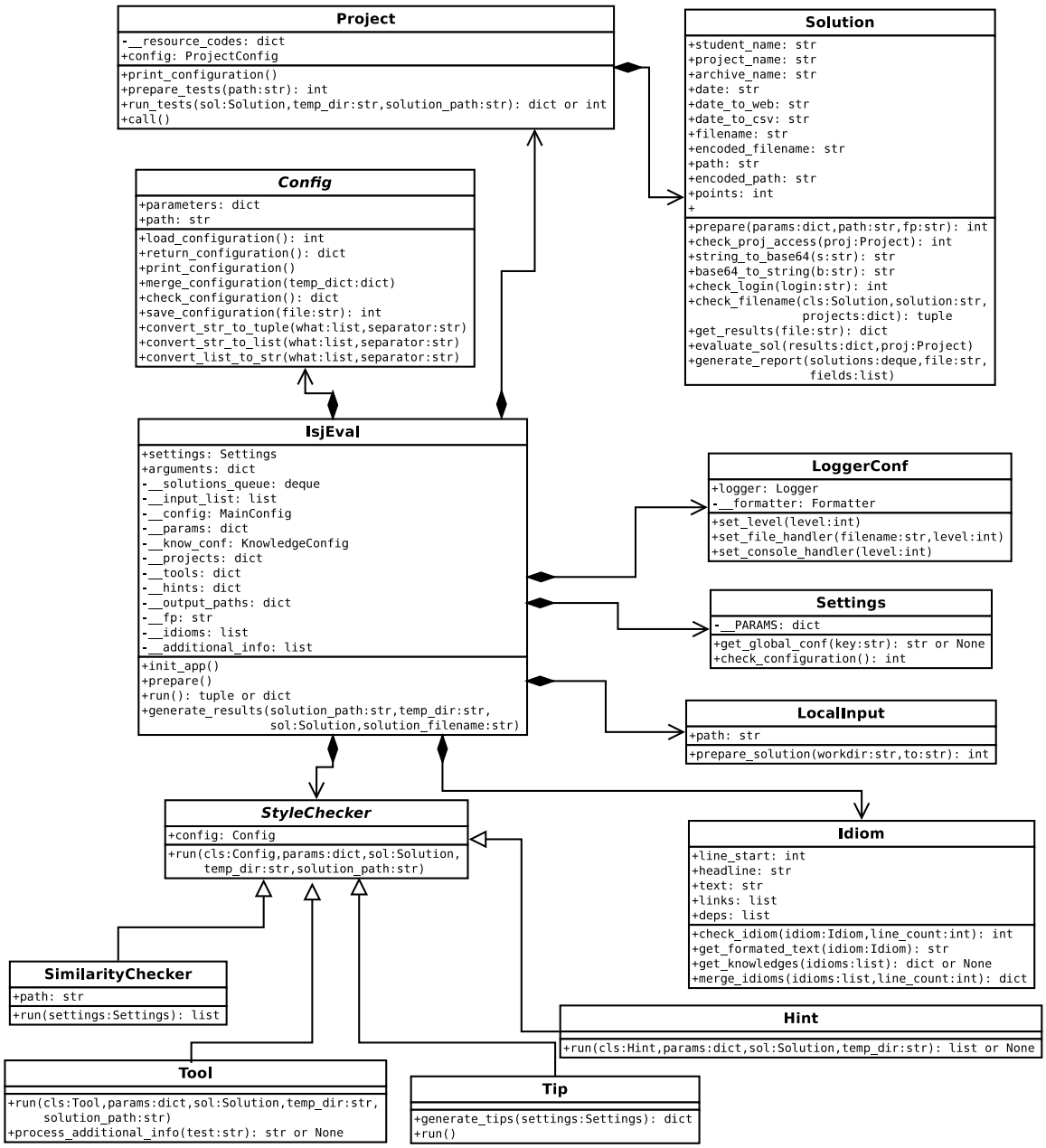
V rámci webového rozhraní byl použit volně dostupný nástroj Flask[4]. Jedná se o nástroj napsaný v jazyce Python, jenž dovoluje zcela nahradit jazyk PHP[5] při vytváření webové prezentace. Přitom je zachována veškerá kompatibilita s ostatními prostředky pro tvorbu webové prezentace jako jsou kaskádové styly (CSS), jazyk JavaScript, HTML. Značnou výhodou tohoto nástroje je možnost využití veškerých prostředků jazyka Python při komunikaci webového rozhraní s aplikací realizující vyhodnocení studentských projektů. Lze tak například zachytit výjimky produkované aplikací nebo pracovat s datovými strukturami bez nutnosti konverze. Dalšími prostředky využitými při řešení byly jazyk HTML doplněný o kaskádové styly[6], jež dovolují nastavit definice stylů pro značky HTML a tím vytvořit výslednou vizualizaci webové prezentace. Posledním nástrojem užitým v rámci webového

[1] Viz https://www.python.org/
[2] Domovské stránky projektu: http://www.sphinx-doc.org/en/1.5.1/
[3] Detaily k projektu jsou dostupné zde: http://docutils.sourceforge.net/rst.html
[4] Domovská stránka projektu zde: http://flask.pocoo.org/
[5] Více informací zde: https://secure.php.net/
[6] Referenční příručka viz https://www.w3schools.com/cssref/

rozhraní byl jazyk Javascript, použitý pro vytvoření dynamického chování webové prezentace na straně klienta a knihovna jQuery[7].

4.2 Aplikace pro analýzu studentského řešení

V této sekci je blíže popsána implementace aplikace realizující automatizované vyhodnocení studentských projektů, která byla diskutována v 3. Diagram tříd 4.1 zachycuje celou strukturu navržené aplikace.



Obrázek 4.1: Diagram tříd navrženého systému

[7]Stránky projektu: https://jquery.com/

Hlavní funkce `main` je umístěna v souboru `main.py`, kde je rovněž instanciován objekt třídy `IsjEval`. Tento objekt abstrahuje celou aplikaci. Chování aplikace lze ovlivnit pomocí parametrů příkazové řádky, které lze zadat při spuštění aplikace. Parametry jsou zpracovávány modulem `argparse`. Mezi nejdůležitější parametr patří přepínač `-c`, který určuje cestu k hlavnímu konfiguračnímu souboru viz 4.2.1. Další parametry jsou popsány v implementační dokumentaci na přiloženém DVD.

Po spuštění aplikace jsou prováděny následující úkony:

- Instanciace třídy `LoggerConf`, která v sobě zapouzdřuje objekt třídy *Logger* určený pro zaznamenávání aktivity aplikace. Podrobnosti jsou uvedeny v sekci 4.4.
- Instanciace třídy `Settings`. Tato třída obsahuje soukromý slovník, ve kterém jsou uloženy globální definice viz 4.2.1.
- Kontrola argumentů příkazové řádky.
- Instanciace třídy `MainConfig`, která obsahuje slovník správcem definovaných parametrů.
- Vytvoření pracovní složky (pískoviště) v souladu s bezpečnostními nároky diskutovanými v sekci 3.5 funkcí `prepare_workdir`.
- Načtení nastavení projektů `ProjectConfig`.
- Nastavení nástrojů `ToolConfig`.
- Nastavení doporučení `HintConfig`.
- Vložení studentského řešení do fronty. Fronta je implementována datovou strukturou `deque` z modulu `collections`. Lze tedy využít všechny předpřipravené funkce pro práci s touto datovou strukturou bez nutnosti implementace funkcí vlastních.

Po provedení výše zmíněných kroků je na řadě hlavní smyčka, která postupně vyhodnocuje řešení čekající ve frontě. Vyhodnocení je závislé na konfiguraci projektu a zahrnuje následující činnosti.

- Ověří na základě parametrů příkazové řádky, zda nebyl aktivován režim *evaluate* přepínačem *-e* a přizpůsobí tomuto faktu jméno výstupního souboru, který posléze vytvoří. Dále ověřuje, zda cesta k vygenerované zpětné vazbě má být kódována, či nikoli. Více informací lze nalézt v 4.2.2.
- Před samotným vyhodnocením je řešení zkopírováno do dočasného adresáře nacházejícího se v lokaci pískoviště. Dočasný adresář z modulu `tempfile` je vytvořen instanciací třídy `TemporaryDirectory`. Volání konstruktoru této třídy vrací *context-manager*[8]. Po ukončení používání dočasného adresáře je adresář automaticky smazán. Adresář je vytvořen s unikátním jménem a následující podmínkou. Není dovolen souběh, přístup/zápis/čtení adresáře má dovoleno pouze uživatel, který jej vytvořil [15].
- Na studentské řešení jsou postupně aplikovány definované testy, nástroje, doporučení viz příslušné sekce níže.

[8] Více informací viz https://docs.python.org/3/reference/datamodel.html

- Po dokončení všech kroků definovaných nastavením projektů jsou na základě výsledků vygenerovány HTML stránky `nosetests.html`, `feedback.html`, `hints.html` a `overview.html`. Všechny tyto soubory jsou posléze zkopírovány do dříve vytvořeného souboru s výsledky.

- Volitelně, pokud je definován přepínač příkazové řádky *-e*, dojde k vygenerování CSV souboru funkcí `generate_report`. Je-li definován přepínač *-ch* vykoná se instanciace třídy `SimilarityChecker` určená pro analýzu podobnosti řešení.

4.2.1 Konfigurace

Konfigurace aplikace je rozdělena do dvou částí.

První část tvoří globální definice, které jsou určeny zejména pro strukturální nastavení systému. Jsou to nastavení, která se mění pouze zřídka. Jedná se zejména o definice adresářů vytvářených při vyhodnocování řešení. Hlavním účelem je všechna tato nastavení centralizovat do jedné lokace a tím ve zdrojovém textu eliminovat napevno nastavené proměnné. Potenciální změnu parametrů lze provést velmi jednoduše na jednom místě ve zdrojovém textu. Z pohledu implementace se parametry nacházejí v privátním slovníku `__PARAMS` v modulu `settings.py`. K těmto parametrům lze přistupovat pouze přes funkci `get_global_conf`, která vrací hodnotu parametru pro zadaný klíč, nebo `None` v případě, že klíč v slovníku neexistuje. V modulu je dále implementována funkce **check_configuration**, jež je použita k ověření některých parametrů. Nastavení obsažená ve výše zmíněném slovníku není možné měnit skrze webové rozhraní. Příklad konfiguračního slovníku je uveden v příloze B.

Druhou část tvoří konfigurační soubory pro jednotlivé části systému (nástroje, doporučení, projekty). Jedná se o nastavení, která je možno měnit dynamicky pomocí webového rozhraní. Konfigurační soubory jsou uloženy v pracovním adresáři aplikace ve formátu odpovídající jazyku YAML[9]. Tento značkovací jazyk byl zvolen především kvůli jednoduchosti práce s tímto formátem v Pythonu a přehledné syntaxi, tzn. jde jednoduše měnit hodnoty parametrů ručně v případě potřeby. V aplikaci je dílčí konfigurační soubor reprezentován slovníkem, který je zapouzdřen do třídy. Rodičovská třída má název `Config`. Implementuje základní metody pro práci s konfiguračními soubory a zejména obsahuje abstraktní metodu `check_configuration`, která je zodpovědná za kontrolu parametrů umístěných ve slovníku a musí být implementována pro všechny potomky. Jednotlivé konfigurační soubory jsou pak potomky rodičovské třídy `Config`, kterou na základě svého určení vhodně rozšiřují. V tabulce 4.1 jsou uvedeny používané konfigurační soubory.

Jméno třídy	Popis
`MainConfig`	Hlavní konfigurace aplikace
`ToolConfig`	Konfigurace nástroje, skriptu
`HintConfig`	Konfigurace doporučení (detekční modul)
`KnowledgeConfig`	Konfigurace znalostí
`TipsConfig`	Konfigurace tipů
`ProjectConfig`	Zahrnuje všechna nastavení pro jednotlivé projekty

Tabulka 4.1: Tabulka tříd nejčastějších chyb v programech

[9]Stránka projektu: http://yaml.org/

4.2.2 Vstup

Hlavním vstupem aplikace je studentské řešení. Každé řešení je v aplikaci reprezentováno objektem třídy `Solution`. Tato třída v sobě zapouzdřuje atributy a metody nutné k práci/identifikaci řešení. Důležitými atributem je jméno projektu, na základě kterého je řešení vyhodnocováno. Dále jsou zde obsaženy časové údaje o odevzdaném řešení, bodové hodnocení a cesta k výslednému adresáři, do kterého budou uloženy všechny výstupy. Cesta musí být v lokaci, kde k ní webová část systému může přistoupit. Cestu k adresáři je možné kódovat viz sekce 4.2.6.

4.2.3 Korektnost

Korektnost řešení zahrnuje kontrolu řešení definovanými testy a na základě úspěšnosti testů také bodové hodnocení.

Povinným vstupem je konfigurační soubor projektu `ProjectConfig` daného řešení viz 4.2.1. Po spuštění aplikace dojde k načtení všech projektů z konfiguračních souborů. Posléze je vykonána instanciace třídy `Project` ze souboru `project.py` s příslušnou konfigurací projektu jako parametrem konstruktoru. Všechny projekty je nezbytné načíst, jelikož řešení čekající ve frontě mohou náležet jakémukoli projektu korektně nastavenému v systému.

V dalším kroku je řešení zkopírováno do pískoviště spolu s příslušnými testy. Následuje volání metody `run_tests`, která zprostředkuje spuštění testů nad studentským řešením s definovanými prostředky viz 4.2.6. Samotné spuštění testů je implementováno voláním nástroje `nose`[10]. Nástroj `nose` implementuje jednoduché rozhraní nad prací s funkcemi z modulu `unittest`. Rovněž podporuje export výsledků do HTML souboru, což umožňuje přímé použití tohoto souboru ve výsledné zpětné vazbě, která se sestává z HTML souborů viz 3.3. Příklad vygenerovaného výstupu je uveden na obrázku 3.6.

Extrakce bodového hodnocení probíhá na základě analýzy HTML výstupu. Souhrn získaných bodů z jednotlivých úloh je uložen v HTML tabulce obsažené ve vygenerovaném výstupu viz níže. Z této tabulky je extrahován počet úspěšných testů funkcí `get_results`. Pokud celkový počet úspěšných testů odpovídá celkovému počtu aplikovaných testů pro danou úlohu, je studentovi přičten plný počet bodů pomocí funkce `evaluate_sol`.

```
<table>
    <tr>
        <th>Class</th>
        <th>Fail</th>
        <th class="failed">Error</th>
        <th>Skip</th>
        <th>Success</th>
        <th>Total</th>
    </tr>
        <tr>
            <td>test_proj04.balanced_paren</td>
            <td>0</td>
            <td class="failed">1</td>
            <td>0</td>
            <td>4</td>
            <td>5</td>
```

[10] Stránka projektu: http://nose.readthedocs.io/en/latest/

```
        </tr>
    <tr>
        <td><strong>Total</strong></td>
        <td>0</td>
        <td class="failed">1</td>
        <td>0</td>
        <td>4</td>
        <td>5</td>
    </tr>
</table>
```

Po vyhodnocení všech řešení ve vstupní frontě lze vygenerovat CSV report. Report obsahuje informaci o celkovém počtu bodů, které student získal z odevzdaného řešení. V aplikaci je za tímto účelem navržena funkce `generate_report`, jež pro zápis využívá standardní modul `csv`.

4.2.4 Styl

V následující sekci jsou popsány třídy realizující kontrolu stylu 3.2.2 a podobnosti řešení 3.2.4. Ústřední prvkem je abstraktní třída `StyleChecker`. Základním parametrem je objekt třídy `Config`, který obsahuje specifická nastavení pro daný nástroj, doporučení apod. Důležitou součást představuje abstraktní metoda `run`, která slouží pro spouštění a zpracování výsledků. Tato metoda je v potomcích přetěžována. Potomci této třídy jsou popsány níže.

- `Tip` – je použita pro práci s tipy viz 3.3. Obsahuje metody pro načtení a ukládání tipů.
- `Tool` – zapouzdřuje nástroje použité při vyhodnocení. Spouští nástroj a příslušné výstupy kopíruje do složky obsahující výslednou zpětnou vazbu.
- `Hint` – podrobně popsána v sekci 4.2.5.
- `SimilarityChecker` – použit nástroj *MOSS* představený v 3.2.4. Řešení jsou nahrána na server, který realizuje analýzu pomocí speciálního skriptu[11]. Jakmile je nahrávání dokončeno aplikace aktivně čeká na vyhodnocení výsledků. Ty jsou poskytnuty ve formě odkazu na webové stránky. Tento odkaz je extrahován a poskytnut vyučujícímu k nahlédnutí.

4.2.5 Doporučení

Doporučení k odevzdanému studentskému kódu jsou generována na základě detekčních modulů blíže popsaných v 3.2.3. Každý detekční modul obsahuje detekční kód, pomocí kterého je hledána daná konstrukce. Je-li konstrukce nalezena, je vytvořen speciální záznam. Záznam je tvořen instancí třídy `Idiom` a jeho příklad je uveden níže. Povinnými položkami seznamu jsou číslo řádku, nadpis, vysvětlující text. Pokud tyto údaje nejsou zadány, doporučení je přeskočeno viz funkce `check_idiom`. Jednotlivé záznamy jsou ukládány do seznamu. Seznam je po vrácen zpět do hlavní aplikace, kde je dále zpracováván. V případě, že některý detekční modul selže, je tato skutečnost zaznamenána do logu a daný modul se přeskočí.

[11] Implementace k nalezení zde: http://moss.stanford.edu/general/scripts/mossnet

```
idioms.append(idiom.Idiom(i + 1,
  'Otevření souboru',
  'Pro otevření souboru použijte context manager: with open(file, mode):.',
  ['https://docs.python.org/3/library/functions.html#open',
  'http://preshing.com/20110920/the-python-with-statement-by-example/']))
```

Po vykonání všech nastavených detekčních modulů je v aplikaci dostupný seznam, který obsahuje všechny nalezené konstrukce. Jelikož na jednom řádku může být detekováno více konstrukcí, je nutné dané konstrukce sloučit tak, aby bylo dosaženo formátu navrženého v 3.3. Tuto funkcionalitu implementuje funkce `merge_idioms`, která na základě čísla řádku sloučí případné vícenásobné výskyty doporučení na jednom řádku. Výsledný seznam je posléze předán na vykreslení do HTML stránky viz 4.2.7.

Do každého záznamu (instanci třídy `Idiom`) může být přidán seznam témat, jenž je považován za prerekvizitní znalost pro danou konstrukci. Témata jsou tvořena názvem a odkazy na relevantní zdroje pro studium dané problematiky. Jejich nastavení je v příslušném konfiguračním souboru `KnowledgeConfig`. Pokud je seznam obsahující prerekvizitní témata ke studiu zadán, dojde k jeho zpracování funkcí `get_knowledges`, která vrátí odpovídající odkazy, ty jsou následně předány k vykreslení do HTML.

4.2.6 Bezpečnost

V sekci 3.5 byly diskutovány bezpečnostní kritéria kladená na aplikaci. V této sekci je popsána jejich implementace v aplikaci.

Vytvoření pískoviště implementuje funkce `prepare_workdir`, která vytváří příslušnou složku spolu s nastavením uživatelských práv. Dále je nastaven uživatel, případně skupina, pod kterými jsou studentská řešení spouštěna.

Správa systémových prostředků je implementována funkcemi z modulu `resource`[12]. Toto řešení bylo zvoleno zejména proto, že lze s nastavením limitů pro prostředky pracovat přímo v jazyku Python použitím příslušných funkcí. Není tedy nutné volat externí linuxové příkazy, například `ulimit`. Podporované prostředky, které lze nastavit, jsou vypsány v tabulce 4.2, kde je uveden název makra reprezentující prostředek a jeho popis. Popis byl převzat z [17].

Název	Popis
RLIMIT_CPU	Maximální procesorový čas zadaný v sekundách, který může proces použít.
RLIMIT_FSIZE	Maximální velikost souboru, který může proces vytvořit.
RLIMIT_DATA	Maximální velikost dat, které může proces uložit na haldu (Heap).
RLIMIT_STACK	Maximální velikost volání zásobníku.
RLIMIT_NOFILE	Maximální počet otevřených souborů.
RLIMIT_AS	Maximální velikost adresového prostoru, kterou může proces použít.
RLIMIT_NPROC	Maximální počet procesů, které mohou být vytvořeny během běhu programu.

Tabulka 4.2: Přehled podporovaných prostředků

[12] Viz https://docs.python.org/3/library/resource.html

Pro každý prostředek se nastavují dva limity [17].

- *soft* – je aktuální limit, který může být snižován, či zvyšován maximálně do výše *hard* limitu.
- *hard* – může být snižován na jakoukoli hodnotu větší než *soft* limit.

Limity se nastavují funkcí `set_resources` na základě konfigurace projektu a jsou nastaveny pouze pro proces, který spouští testy nad studentským řešením.

Dobu běhu studentského řešení lze krom výše zmíněného limitu `RLIMIT_CPU` omezit nastavením parametru `timeout` v hlavním konfiguračním souboru `MainConfig`. Tento parametr je poté předán funkci `communicate` z modulu *subprocess*.

Další bezpečnostní prvek představuje kódování cesty k adresáři obsahující výslednou zpětnou vazbu. Toto opatření bylo zavedeno za účelem zajištění soukromí studentů a předejití opisování, tzn. aby si student nemohl prohlížet zpětnou vazbu dalšího studenta. V případě, že je kódování cesty zapnuto (položka *encoded* v hlavním konfiguračním souboru `MainConfig`), je název adresáře kódován pomocí *Base64*[13] algoritmu. Název adresáře se sestává z unikátního identifikátoru studenta, data a času odevzdání. Pro šifrování/dešifrování je používán modul `base64`[14]. Na následujícím příkladu jsou ukázány obě možné podoby názvu adresáře.

```
xnovak00_proj04_2017-05-13-21-12-23
eG5vdmFrMDBfcHJvajA0XzIwMTctMDUtMTMtMjEtMDEtMDE%3D
```

4.2.7 Generování zpětné vazby

Během vyhodnocování studentského řešení dochází k vytváření výstupů, které tvoří zpětnou vazbu. Tyto výstupy jsou kopírovány do společného adresáře. Zpětná vazba se sestává z HTML stránek, z nichž některé jsou generovány nástroji třetích stran a přímo zkopírovány beze změny do adresáře s výsledky. Zbylé stránky jsou dynamicky generovány pomocí šablon napsaných v jazyku *Jinja*[15]. Šablona se sestává z HTML kódu, do kterého jsou vkládány speciální značky. Příklad šablony generující stránku pro nahrávání řešení je uveden níže.

```
<title>Nahrajte řešení</title>
<h1>Nahrajte svoje řešení</h1>
{% if projs %}
<form method=post enctype=multipart/form-data>
    <h3>Podporované projekty jsou:</h3>
        <select name="project">
            {% for proj in projs %}
                <option value={{ proj }}>{{ proj }}</option>
            {% endfor %}
        </select>
        <h3>Vyberte soubor, který je pojmenován podle konvence:
            isj_project_xlogin00.py</h3>
        <input type=file name=file>
        <input type=submit value=Upload>
```

[13] Specifikace zde: https://tools.ietf.org/html/rfc3548.html
[14] Viz https://docs.python.org/3/library/base64.html
[15] Viz http://jinja.pocoo.org/

```
        </form>
{% else %}
    <h3 class="red_text">Není nastavený žádný projekt</h3>
{% endif %}
```

Značky obsahují klasické řídící struktury, je tedy možné generovat stránku na základě kritérií, nebo předat slovník (`projs`) z hlavního programu a jeho obsah vypsat jako položky rolovacího seznamu. Základní podoba zpětné vazby čítá následující soubory:

- `hints.html` – zobrazuje načtený studentský kód s vyznačenými řádky, kde jsou uvedena příslušná doporučení viz obrázek 3.8. Zdrojový kód odevzdaného řešení je přehledně formátován přes volně dostupný JavaScript framework `code-prettify`[16]. Kód je vložen do speciální značky, viz níže, a po načtení stránky ve webovém prohlížeči přehledně naformátován.

  ```
  <pre class="prettyprint linenums">
  print("Jsem vlož zdrojový text")
  </pre>
  ```

  Řádky jsou označeny na základě vygenerovaného slovníku s doporučeními 4.2.5. Tato stránka je generována ze stejnojmenné šablony.

- `index.html` – rozcestník zpětné vazby, automaticky načítá strukturu menu a vytváří klikací odkazy v menu. Také automaticky načítá stránku `overview.html` popsanou níže. Stránka je opět dynamicky generována na základě šablony `feedback.html`.

- `overview.html` – obsahuje základní přehled o odevzdaném řešení společně s počtem získaných bodů. Dále je zde sekce *Poznámky*, která může obsahovat odkazy na extra nástroje, či výsledky uživatelských skriptů specifických pro daný projekt. Vzhled stránky odpovídá obrázku 3.5. Stránka je generována ze stejnojmenné šablony.

- `style.html` – tato stránka je vygenerována nástrojem `pepper8`[17], který dokáže převést výsledky stylistické analýzy provedené nástrojem `flake8` do HTML reportu. Celý proces je popsán skriptem `flake8.sh`, jenž je spuštěn v rámci kroku *tools*. Detail stránky je zachycen na obrázku 3.7.

- `styles.css` – je soubor se základními definicemi kaskádových stylů (CSS), které vytváří grafiku zpětné vazby.

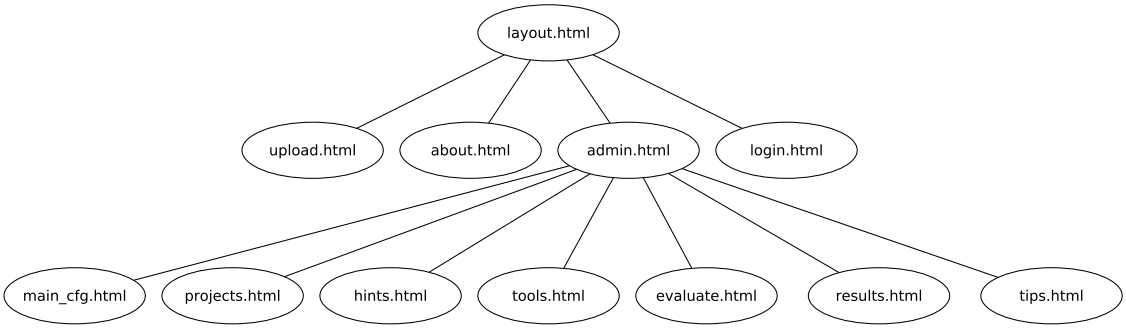
- `tips.html` – obsahuje výpis tipů, které tvoří seznam dvojic, zdrojový kód společně s vysvětlujícím textem. Zdrojový kód je formátován přes frameworku `code-prettify`. Pro generování stránky je použita stejnojmenná šablona. Vygenerovaná podoba stránky je k dispozici na obrázku 3.9.

- `xlogin00.html` – je soubor pojmenovaný unikátním identifikátorem studenta a obsahuje výsledky testů aplikovaných na řešení. Stránka je vygenerována nástrojem `nose`.

[16] Stránka projektu zde: https://github.com/google/code-prettify
[17] Více informací zde: https://pypi.python.org/pypi/pepper8/1.0.4

4.3 Webové rozhraní

Webové rozhraní implementuje skript `web_isj_eval.py`. Všechny potřebné soubory jsou uloženy v adresáři `web`. Tento skript vykresluje šablony jednotlivých webových stránek napsaných v jazyku *Jinja* představeném v sekci 4.2.7. Vykreslování a funkcionalita stránek jsou implementovány pomocí frameworku *Flask* představeného v sekci 4.1. Hierarchie stránek je zobrazena na obrázku 4.2. Hlavním souborem je `layout.html`, v něm se nachází blok, kam jsou načítány všechny stránky úrovně dvě ve vyobrazeném stromu. Stránka `admin.html` implementuje administrátorskou sekci a po přihlášení zpřístupňuje všechny stránky úrovně tři stromu.



Obrázek 4.2: Hierarchie webových stránek

Jak již bylo uvedeno, stránky jsou implementovány nástrojem *Flask*, kdy každá stránka je definována samostatnou funkcí. Tato funkce v sobě typicky obsahuje příkaz pro vykreslení webové stránky (šablony) s požadovanými parametry. Pro metody `GET` *Flask* implementuje speciální obslužné funkce.

Vzhled je konfigurován kaskádovými styly, které jsou načítány ze souboru `style.css`.

Spolupráce webového rozhraní a aplikace vyhodnocující studentská řešení je založena na volání této aplikace z webového rozhraní. Chování aplikace lze přizpůsobovat na základě parametrů příkazové řádky. Webové rozhraní pak provádí interpretaci definovaných výstupů získaných z aplikace podle zadaných parametrů. V aktuální implementaci existují dva režimy volání aplikace a to:

- Vyhodnocení jednoho řešení vloženého přes formulář 3.11. Iniciováno vyučujícím, nebo studentem.
- Vyhodnocení balíku řešení. Může být iniciováno pouze vyučujícím.

Tato funkcionalita je zajištěna pomocí importu hlavního modulu aplikace `main.py` do webového skriptu spolu s pomocnými moduly pro práci s konfiguračními soubory a globálním nastavením 4.2.1.

4.3.1 Administrace a bezpečnost

Administrace aplikace skrze webové rozhraní je možná po přihlášení. Pro přihlášení je vyžadováno zadání hesla do formuláře v šabloně, která se nachází v souboru `login.html`. Samotné heslo je uloženo z bezpečnostních důvodů v textovém souboru na disku. Soubor s heslem by měl mít minimální oprávnění, avšak aplikace musí mít tento soubor zpřístupněn pro čtení. Samotné heslo je načítáno pomocí metody `get_password`.

Přístup do administrátorské sekce je podmíněn nastavením `SESSION` proměnné. Tato proměnná je nastavena po úspěšném zadání hesla. Každá stránka z úrovně tři stromu na obrázku 4.2 kontroluje před vykreslením obsah této proměnné.

Dalším bezpečnostním opatřením je kontrola nahrávaných souborů. To znamená, že zejména studentům je dovoleno nahrát pouze soubory s definovanou koncovkou `.py`. Nahrávaný soubor je ještě před uložením předán funkci `secure_filename`[18], která odstraní případné nebezpečné znaky z názvu souboru.

4.4 Chybové stavy

Možné problémy při vyhodnocování studentských projektů jsou zaznamenávány do logu. Pro implementaci sběru logů je využita třída `Logger`. Tato třída poskytuje funkce pro vytváření logů a jejich vypisování do souboru, či konzole. Dalšími výhodami, které přináší používání této třídy, je možnost použití ve vícevláknových aplikacích a možnost nastavení úrovně logů tzn. lze vymezit, které zprávy mají být zapisovány na výstup. V aplikaci je nastaveno logování do souboru i konzole.

V případě chyby při vyhodnocování na straně webového rozhraní je příslušná výjimka zachycena. Detailní informace společně s řešením jsou zaslány na definovaný emailový účet. Toto chování implementuje funkce `send_email_to`. Na základě přijaté zprávy může vyučující analyzovat logy a zjistit tak příčinu neúspěchu vyhodnocení. Typickým problémem, který se při testování vyskytl, jsou řešení, která obsahují chyby v zarovnání zdrojového textu.

[18] Specifikace funkce: http://werkzeug.pocoo.org/docs/0.12/utils/#werkzeug.utils.secure_filename

Kapitola 5

Testování

V této kapitole je popsána metodika testování aplikace a webového rozhraní. Testování probíhalo ve třech hlavních částech. U všech variant je uvedeno testovací prostředí, cíle a výsledky.

První část tvořilo základní testování, jehož hlavním cílem bylo ověřit elementární funkcionalitu aplikace a webového rozhraní. To zahrnuje testování celého procesu vyhodnocení např. detekčních modulů na připravených vstupech, generování zpětné vazby či detekci chybových stavů.

Druhá část měla za cíl ověřit implementaci na reálných řešeních projektů, které v rámci kurzu ISJ studenti postupně vypracovávali. Na rozdíl od základního testování, kde byla struktura vstupu předem známa, různorodost studenských projektů nabízela velký potenciál k vývoji nových detekčních modulů a zároveň ověření těch stávajících.

Cílem třetí části testování bylo poskytnout výsledný systém studentům za účelem vyhodnocení implementovaného řešení a sběru nových nápadů pro budoucí vylepšení. Jako forma zpětné vazby byl zvolen dotazník.

5.1 Základní testování

Jako testovací prostředí byl zvolen interní webový server, který nabízí *Flask* společně se zapnutým *debug* módem. Webový server byl nakonfigurován na průměrném notebooku s 8GB RAM a 2-jádrovým procesorem.

Cílem testování bylo ověřit korektnost implementace a webového rozhraní, v dalším textu jsou uvedena jednotlivá testovací kritéria.

5.1.1 Vyhodnocení korektnosti řešení

Vyhodnocení korektnosti v aplikaci zajišťuje nástroje *nose*. Z hlediska aplikace je nutné zajistit korektní vstupy pro tento nástroj a ošetřit situace, kdy proces vyhodnocení selže. Typické příčiny selhání procesu vyhodnocení jsou řešení, která nelze vyhodnotit v zadaném časovém rozmezí tj. obsahují nekonečné smyčky. Dále řešení, jenž nejsou spustitelná (typicky nekonzistentní odsazení). Obě tyto situace byly testovány pomocí připravených vstupů, které je simulovaly.

5.1.2 Detekční moduly

Detekční moduly tvoří významnou část implementovaného systému. Proto bylo nutné ověřit, zda detekční kód v daných modulech funguje korektně. Za tímto účelem byl vytvořen speciální soubor, který obsahoval všechny konstrukce, pro jejichž detekci existuje detekční modul. Soubor byl vložen do aplikace přes webové rozhraní. Po dokončení vyhodnocení bylo kontrolováno, zda všechny konstrukce byly identifikovány korektně. Testovací soubor společně s vygenerovanou zpětnou vazbou jsou přiloženy na DVD.

5.1.3 Webového rozhraní

Funkcionalita webového rozhraní byla testována na nejnovějších verzích těchto prohlížečů Internet Explorer, Mozzila Firefox, Opera, Google Chrome. Testované platformy byly Linux Mint a Windows 7. Kontrola zahrnovala především test formulářů na správnost zadávaných hodnot a schopnost vyrovnat se s chybovými stavy.

5.2 Studentské projekty

V další fázi byl systém testován na reálných řešeních studentských projektů. Tyto projekty byly postupně zadávány v kurzu Skriptovací jazyky (ISJ). Celkem bylo během semestru zadáno 8 projektů, z nichž v rámci testování aplikace bylo vyhodnoceno prvních 7 projektů. Testovací prostředí bylo stejné jako v případě základního testování popsaného výše.

Hlavním cílem této fáze testování bylo najít možné skryté chyby v systému a znovu prověřit korektnost detekčních modulů. Rozdílem oproti základnímu testování byla různorodost odevzdaných řešení, tzn. i když všechna řešení vedla ke stejnému cíli, jednotlivé implementace byly značně odlišné. Díky tomuto faktu bylo možné zjistit, že některé moduly se za určitých podmínek nechovají korektně, což bylo záhy opraveno.

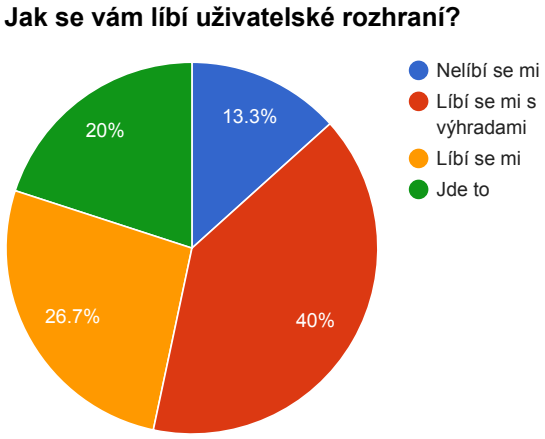
5.3 Dotazník

Nejvyšší vypovídající hodnotu faktické správnosti řešení má testování systému v interakci s reálnými uživateli. Za tímto účelem byl celý systém nainstalován na školní server `athena1`, který je přístupný i mimo adresný rozsah školní sítě. Přes webové rozhraní bylo zpřístupněno vyhodnocení prvních sedmi projektů, které studenti řešili postupně během semestru v rámci kurzu ISJ. Hlavním cílem bylo získat připomínky k navrženému webovému rozhraní a hlavně k vygenerované zpětné vazbě pro odevzdaná studentská řešení. Dalším cílem bylo shromáždit nápady pro budoucí vylepšení implementovaného systému.

Jako prostředek pro sběr těchto informací byla zvolena forma dotazníku. Dotazník byl studentům přístupný přes odkaz umístěný na hlavní stránce webového rozhraní a rovněž jim byl zaslán emailem. V jednotlivých otázkách byl kladen důraz především na vyhodnocení stránek obsahujících zpětnou vazbu.

Po zaslání emailu systém vyzkoušelo 39 studentů, z nichž 17 vyplnilo přiložený dotazník. Nasbíraná data zachycují následující grafy.

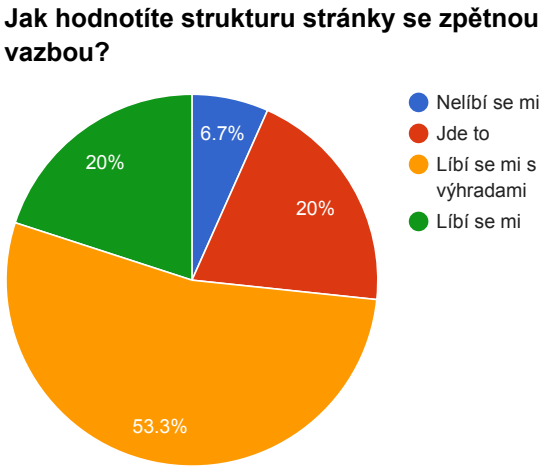
V první otázce studenti hodnotili vzhled uživatelského rozhraní. Většina studentů hodnotí uživatelské rozhraní kladně. Pouze čtrnácti procentům dotázaných se uživatelské rozhraní nelíbí.



Obrázek 5.1: Graf hodnotící vzhled uživatelské rozhraní

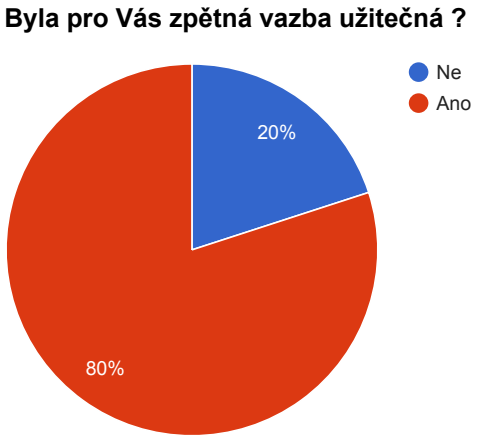
Uživatelskou přívětivost stránek by bylo možné vylepšit přidáním nových dynamických prvků či úpravou grafiky.

Druhá otázka hodnotila strukturu stránky se zpětnou vazbu, měla za cíl zjistit, zda jsou studenti spokojeni s rozdělením a obsahem jednotlivých stránek vygenerované zpětné vazby. Početné skupině studentů 54% se struktura stránek líbila s výhradami a pouze 7% se nelíbila vůbec.



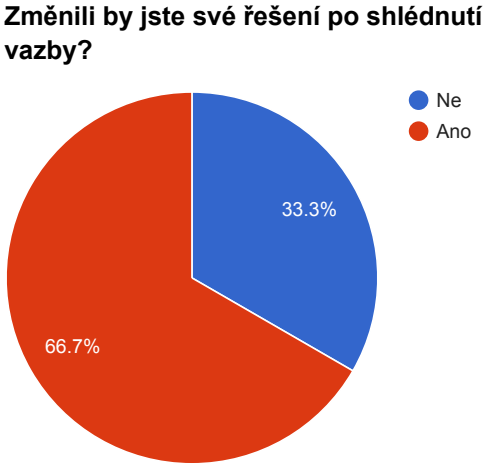
Obrázek 5.2: Graf hodnotící strukturu stránek se zpětnou vazbou

Cílem třetí otázky bylo zjistit, zda zpětná vazba jako celek je pro studenty užitečná. Kladné odpovědi (80% studentů zvolilo možnost *Ano*) dokazují, že systém nalézá své uplatnění jako nástroj doplňující klasickou konvenční výuku.



Obrázek 5.3: Graf zobrazující užitečnost zpětné vazby jako celku

Toto tvrzení podporují i výsledky, které jsou prezentovány v grafu 5.4. Tento graf zobrazuje procento studentů, kteří by po shlédnutí zpětné vazby změnili svoje řešení. Tato otázka se vztahovala na všechny komponenty zpětné vazby tzn. testy, doporučení, kontrola stylu a navržené tipy.



Obrázek 5.4: Graf popisující procento studentů, kteří by po shlédnutí zpětné vazby upravili své aktuální řešení

V páté otázce byli studenti dotazováni zda-li by využili možnost prostudovat si doporučené materiály. Studenti mohou využít individuální materiály, které souvisejí přímo s nalezenou konstrukcí, či materiály vysvětlující celá témata (datové struktury, funkce, generátory). Pouze mírně nadpoloviční většina (60% studentů) by tuto možnost využila. Zjištěné údaje naznačují, že by doporučené materiály měly být atraktivnější (např. zařazení videí, vizualizačních nástrojů), čímž by se obliba poskytnutých materiálů mohla zvýšit.

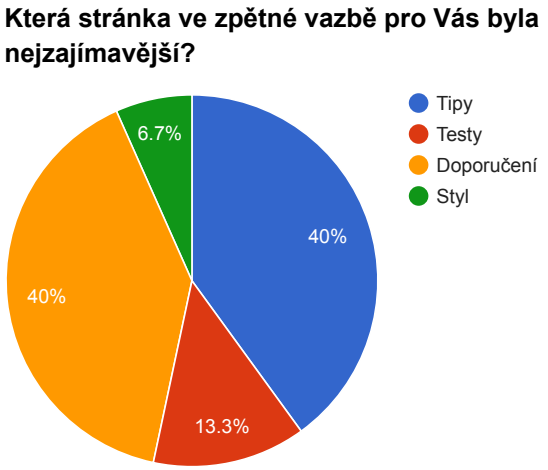
Obrázek 5.5: Graf prezentující procento studentů, kteří by využili nabízené doporučené materiály k prostudování

Předposlední otázka cílila na hodnocení stránky *Tipy*, která se nachází v každé zpětné vazbě, viz 3.3. Tato stránka slouží především k prezentaci často používaných konstrukcí, ale i těch méně známých, které jsou nevhodně nahrazovány jinými. Dalším cílem je pokrytí konstrukcí, pro které neexistuje detekční modul. Výsledné hodnocení této stránky je velmi pozitivní. Všichni studenti, kteří vyplnili dotazník tuto stránku považují za užitečnou.

Obrázek 5.6: Graf popisující užitečnost stránky zpětné vazby *Tipy*

Dokonce 40% dotázaných tuto stránku považuje za nejužitečnější, viz výsledky zobrazené na grafu 5.7. Z těchto výsledků lze vyvodit nutnost rozšíření této stránky o další tipy.

Poslední otázka vyhodnocovala, která stránka (komponenta) zpětné vazby se jeví studentům jako nejzajímavější. Z nasbíraných výsledků vyplynulo, že shodně po 40% získaly stránky *Tipy* a *Doporučení*. Nejméně pak stránky *Styl* a *Testy*. Nižší oblíbenost těchto stránek lze přičíst faktu, že bylo možné zvolit pouze jednu stránku a tudíž studenti vybírali pouze tu, která je nejvíce zaujala.



Obrázek 5.7: Graf prezentuje oblíbenost jednotlivých stránek zpětné vazby

Součástí dotazníku byly rovněž otevřené otázky. Hlavním cílem těchto otázek byl sběr nápadů a připomínek, které by napomohly budoucímu vývoji systému. Nejčastěji se objevovaly připomínky týkající se stránky zpětné vazby *Testy*. Studenti na této stránce postrádali podrobnější informace o testech, které selhaly. Dále se objevily hlasy volající po navýšením počtu tipů na stránce *Tipy*. Z pohledu uživatelského rozhraní studenti navrhli tyto úpravy:

- Automatický výběr souboru s řešením při přetažení souboru myší do prostoru stránky *Nahraj*.
- Automatické zavření všech otevřených oken s doporučeními pro nalezené konstrukce.

5.4 Shrnutí

V této kapitole byla popsána metodika testování implementovaného systému. Nejdříve bylo provedeno základní testování aplikace a webového rozhraní na připravených vstupech, které simulovaly různé situace, jež mohou během procesu vyhodnocení nastat. V druhé fázi byl systém testován na základě poskytnutých řešení studentských projektů. Tyto projekty umožnily objevit nedostatky v implementaci detekčních modulů díky různorodosti řešení. Poslední nejvíce důležitou součástí bylo vyhodnocení systému v interakci se studenty, kteří se teprve seznamují s jazykem Python. Interakce probíhala na základě zaslaného dotazníku. I když studentů, kteří vyplnili dotazník, nebylo mnoho, nasbíraná data i tak přinesla žádané poznatky. Z výsledků vyplynulo, že většina studentů systém hodnotí kladně. Objevily se také konstruktivní připomínky vedoucí k dalšímu možnému vývoji a vylepšení systému. Příkladem mohou být požadavky na detailnější přehled stránky s výsledky testů, či větší dynamičnost procesu odevzdávání řešení. Celkové hodnocení pak nabádalo k dalšímu pokračování ve vývoji systému.

Kapitola 6

Závěr

V dnešní době je vývoj nových automatizovaných výukových systému stále nezbytnější. Z důvodu velkého počtu studentů v jednom ročníku je vyučující nucen nastavit určité tempo při výkladu látky, které nemusí všem studentům vyhovovat. To může pro mnohé studenty v průběhu kurzu znamenat nemalé problémy s vypracováváním projektů, či skládáním zkoušek. Právě zde je možné nalézt ideální prostor pro automatizované výukové nástroje. Ty si kladou za cíl být doplňkem klasické konvenční výuky a poskytnout tak studentovi možnost dodatečného vysvětlení látky, která mu není jasná. Cílem této práce bylo navrhnout a implementovat výukový systém, který je na základě vloženého studentského řešení schopen vygenerovat zpětnou vazbu.

Při návrhu a implementaci této práce byly využity nastudované teoretické znalosti z oblasti inteligentních výukových systémů. Některé myšlenky byly rovněž převzaty z existujících implementací především [14].

Celý systém se sestává ze dvou hlavních částí aplikace realizující vyhodnocení studentského řešení a webového rozhraní. Webové rozhraní slouží jako rozhraní pro odevzdání řešení a pro potřeby konfigurace aplikace. Obě části systému byly vyhodnoceny ve třech fázích. V první fázi byla testována funkčnost implementace na připravených vstupech a schopnost webového rozhraní vyrovnat se s nekorektními vstupy. V druhé fázi byly jako vstupy použity reálná studentská řešení projektů. Testování bylo provedeno celkem na sedmi projektech. Díky různorodosti odevzdaných řešení bylo možné odhalit nedostatky v detekčním kódu některých modulů. V poslední fázi byl celý systém nainstalován na veřejný server, kde byl přístupný reálným uživatelům. Cílem bylo obdržet zpětnou vazbu od budoucích potenciálních uživatelů. Forma zpětné vazby byla reprezentována dotazníkem. Na základě nasbíraných dat bylo provedeno vyhodnocení. To ukázalo, že studenti navržený systém vnímají pozitivně a je pro ně přínosem. Zároveň ale upozornili na nedostatky v oblasti vzhledu a ve stránkách zpětné vazby, kde jsou umístěny výsledky testů.

Při dalším vývoji práce by bylo možné začít od připomínek, které vyplynuly z nasbíraných dat. Především se jednalo o poskytnutí podrobnějších informací k neúspěšným testům, přidání více dynamického chování do webového rozhraní či rozšíření databáze tipů.

Literatura

[1] Aiken, A.: Moss A System for Detecting Software Similarity. [online]. 2014 [cit. 2017-05-17].
URL http://theory.stanford.edu/~aiken/moss/

[2] Alchin, M.: *Pro Python, 2nd edition.* Apress, 2014, ISBN 978-1484203354.

[3] Altadmri, A.; Brown, N. C. C.: *37 Million Compilations: Investigation Novice Programming Mistakes in Large-Scale Student Data. Technická zpráva, Canterbury: School of Computing, University of Kent, 2016.*

[4] Balík, M.: Programování a algoritmizace I. [Online; navštíveno 30.11.2016].
URL https://www.fit.cvut.cz/predmety/bi-pa1

[5] Barabas, M.: Bezpečnost informačních systémů: System and Network Security. [přednáška]. 2015 [cit. 2017-01-04].

[6] Beazley, D.; Jones, B. K.: Pro Python, 3nd edition. O'Reilly Media, 2013, ISBN 978-1449340377.

[7] Beck, K.: Extreme Programming Explained. Addison-Wesley Professional, 2000, ISBN 201-61641-6.

[8] Bloom, B. S.: The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. Technická zpráva, University of Chicago and Northwestern University, 1984.

[9] Bosch, N.; D'Mello, S.: Sequential Patterns of Affective States of Novice Programmers. Technická zpráva, Nortre Dame: Departments of Computer Science and Psychology, University of Notre Dame, 2013.

[10] Brown, N. C. C.; Altadmri, A.: Investigating Novice Programming Mistakes: Educator Beliefs vs Student Data. Technická zpráva, Canterbury: School of Computing, University of Kent, 2014.

[11] Danjou, J.: The Hacker's Guide to Python, 3rd edition. Lulu.com, 2016, ISBN 978-1-329-98474-5.

[12] Erdogmus, H.; Morisio, M.; Torchiano, M.: On the Effectiveness of the Test-First Approach to Programming. Technická zpráva, Pittsburgh: Department of Computer Science, University of Pittsburgh, 2005.

[13] *Esteves, M.; Mendes, A. J.: A simulation tool to help learning of object oriented programming basics. In* 34th Annual Frontiers in Education, 2004. FIE 2004.*, Oct 2004, ISSN 0190-5848, s. F4C–7–12 Vol. 2, doi:10.1109/FIE.2004.1408649.*

[14] *Fangohr, H.; O'Brien, N.: Teaching Python programming with automatic assessment and feedback provision. Technická zpráva, Southampton: Faculty of Engineering and the Enviromment, University of Southampton, 2015.*

[15] *Foundation, P. S.: 11.6. tempfile — Generate temporary files and directories. [online]. 2017 [cit. 2017-05-08].*
URL https://docs.python.org/3/library/tempfile.html

[16] *Foundation, P. S.: 32.2. ast — Abstract Syntax Trees. [online]. 2017 [cit. 2017-05-01].*
URL https://docs.python.org/3/library/ast.html

[17] *Foundation, P. S.: 35.11. resource — Resource usage information. [online]. 2017 [cit. 2017-05-01].*
URL https://docs.python.org/3/library/resource.html

[18] *Foundation, P. S.: 5. Built-in Exceptions. [online]. 2016 [cit. 2017-01-08].*
URL https://docs.python.org/3/library/exceptions.html

[19] *Gegg-Harrison, T. S.:* Exploiting Program Schemata in a Prolog Tutoring System*. Dizertační práce, Durham: Department of Computer Science, Duke University, 1993.*

[20] *Gross, S.; Strickroth, S.; Piknwart, N.; aj.: Towards Deeper Understanding of Syntactic Concepts in Programming. Technická zpráva, Clausthal-Zellerfeld: Department of Informatics, Clausthal University of Technology, 2013.*

[21] *Gutierrez, F.; Atkinson, J.: Adaptive feedback selection for intelligent tutoring systems. Technická zpráva, Concepcion: Department of Computer Sciences, Universidad de Concepcion, 2010.*

[22] *Hage, J.; Rademaker, P.; van Vugt, N.: A comparison of plagiarism detection tools. Technická zpráva, Utrecht: Department of Information and Computing Sciences Utrecht University, 2010.*

[23] *Harrington, P.: Machine learning in Action. Manning Publications Co., 2012, ISBN 978-1617290183.*

[24] *Hladká, E.: Úvod do programování. [Online; navštíveno 30.11.2016].*
URL http://is.muni.cz/predmet/fi/podzim2016/IB111

[25] *Inc., C.: About. [Online; navštíveno 20.4.2017].*
URL https://about.coursera.org/

[26] *Knupp, J.: Writing Idiomatic Python 3.3. CreateSpace Independent Publishing Platform, 2013, ISBN 978-1482374810.*

[27] *Kreslíková, J.: Základy programování. [Online; navštíveno 30.11.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=11499

[28] *Kumar, A. N.: Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. Technická zpráva, Mahwah: Ramapo College of New Jersey, 2006.*

[29] *Lane, H. C.; VanLehn, K.: A Dialogue-Based Tutoring System for Beginning Programming. Technická zpráva, Canada: National Research Council of Canada, 2004.*

[30] *Lažanský, J.; Mařík, V.; Štěpánková, O.: Umělá inteligence (1). Academia, 2000, ISBN 80-200-0496-3.*

[31] *Le, N. T.; Strickroth, S.; Gross, S.; aj.: A Review of AI-Supported Tutoring Approaches for Learning Programming. Technická zpráva, Clausthal-Zellerfeld: Department of Informatics, Clausthal University of Technology, 2013.*

[32] *Mitkov, R.: The Oxford handbook of computational linguistics. Oxford University Press Inc., 2003, ISBN 978-0-19-927634-9.*

[33] *Peringer, P.: Jazyk C. [Online; navštíveno 30.11.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=7187

[34] *Pietrzak, B.: Abstract Syntax Tree. [online]. 2017 [cit. 2017-05-01].*
URL http://www.cs.put.poznan.pl/bpietrzak/sat/AbstractSyntaxTree.pdf

[35] *Polson, M. C.; Richardson, J. J.: Foundations of Intelligent Tuoring Systems. Lawrence Erlbaum Associates Inc., 1988, ISBN 0-8058-0053-0.*

[36] *Reidingerl, J.: Code reviews v praxi. [Online; navštíveno 30.11.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=11536

[37] *Schleimer, S.; Wilkerson, D. S.; Aiken, A.: Winnowing: Local Algorithms for Document Fingerprinting. Technická zpráva, Chicago: University of Illinois and UC Berkeley: Computer Science Division, 2003.*

[38] *Shalev-Shwartz, S.; Ben-David, S.: Understanding machine learning from theory to algorithms. Cambridge University Press, 2014, ISBN 978-1107057135.*

[39] *Shute, V. J.; Psotka, J.: Intelligent Tutoring Systems: Past, Present, and Future. Technická zpráva, Texas: Armstrong Laboratory, Brooks Air Force Base; Virginia: U.S. Army Research Institute, 1994.*

[40] *Singh, R.; Gulwani, S.; Solar-Lezema, A.: Automated Feedback Generation of Introductory Programming Assignments. Technická zpráva, Cambridge: MIT CSAIL; Redmond: Microsoft Research; Cambridge: MIT CSAIL, 2013.*

[41] *Smrž, P.: Skriptovací jazyky. [Online; navštíveno 30.11.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=6826

[42] *Stevens, W. R.; Rago, S. A.: Advanced Programming in The UNIX Environment, 3rd Edition. Addison-Wesley Professional, 2013, ISBN 978-0321637734.*

[43] *Sykes, E. R.: Qualitative Evaluation of the Java Intelligent Tutoring System. Technická zpráva, Oakville: School of Applied Computing and Engineering Sciences, 2006.*

[44] *Vizcaíno, A.; Alarcos, G.: A Simulated Student Can Improve Collaborative Learning. Technická zpráva, Paseo de la: Escuela Superior de Informática, 2005.*

[45] *Woolf, B. P.: Building intelligent interactive tutors: student-centered strategies for revolutionizing e-learning. Elsevier Inc., 2009, ISBN 978-0-12-373594-2.*

[46] *Yudelson, M.; Brusilovsky, P.: NavEx: Providing Navigation Support for Adaptive Browsing of Annotated Code Examples. Technická zpráva, Pittsburgh: School of Information Science, University of Pittsburgh, 2005.*

[47] *Šlapal, J.: Logika. [Online; navštíveno 30.06.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=11539

[48] *Šlapal, J.: Matematické struktury v informatice. [Online; navštíveno 30.11.2016].*
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=11536

Příloha A

Obsah přiloženého paměťového média

- `doc` – zdrojové kódy textové části diplomové práce,
- `isj_eval1` – zdrojové soubory implementovaného systému,
- `isj_eval1/doc` – implementační dokumentace,
- `README.txt` – návod k instalaci a použití,
- `outputs` – výsledky experimentů,
- `poster.pdf` – plakát prezentující cíle práce.

Příloha B

Konfigurační soubor

```
self.__PARAMS = {'upload_dir' : '../workdir/uploads',
    'hint_directory' : '../workdir/hint_dir',
    'proj_conf_dir' : '../workdir/proj_cfg',
    'report_dir' : os.path.abspath('../web/static/downloads'),
    'test_directory' : os.path.abspath('../workdir/sandbox'),
    'tips_dir' : '../workdir/tips_dir',
    'tool_conf_dir' : '../workdir/tool_cfg',
    'workdir' : '../workdir',
    'knowledge_cfg' : '../cfg/knowledge.yaml',
    'allowed_extensions' : set(['py']),
    'allowed_extensions_admin' : set(['zip', 'py']),
    'secret_key' : '',
    'tips_file' : '../isj_eval1/feedback/tips.html',
    'wis_csv_fields' : ['Id','Jmeno','Login','Body','Celk','Datum','Kdo'],
    'moss_execuable' : 'perl',
    'moss_params' : ['../scripts/moss','-l', 'python', '-m', '4'],
    'web_pass' : 'test'}
```

Příloha C

Příklad emailové zprávy

```
Solution: ../workdir/uploads/isj_proj04_xpokus00.py

Reason:
Traceback (most recent call last):
File "web_isj_eval.py", line 102, in upload
  paths = main.main(sys.argv)
File "../isj_eval1/main.py", line 33, in main
  results = main_app.run()
File "../isj_eval1/isjeval.py", line 167, in run
  self.generate_results(solution_path, temp_dir, sol, solution_filename)
File "../isj_eval1/isjeval.py", line 186, in generate_results
  source_code = read_file(source_path)
File "../isj_eval1/procedures.py", line 191, in read_file
  content = fh.read()
File "/python_install/lib/python3.6/codecs.py", line 321, in decode
  (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe1 in position 1594:
invalid continuation byte
```