VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FRAMEWORK PRO TESTOVÁNÍ STUDENTSKÝCH PROJEKTŮ

FRAMEWORK FOR TESTING STUDENT PROJECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. NATÁLIA DIŽOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání diplomové práce

Studentka: **Dižová Natália, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Softwarové inženýrství
Název: **Framework pro testování studentských projektů**
Framework for Testing Student Projects
Kategorie: Analýza a testování softwaru
Zadání:

1. Nastudujte metody testování na úrovni integračních testů a testů jednotek. Nastudujte vzory pevného okolí testované jednotky (test-fixture). Nastudujte použití linuxových kontejnerů.
2. Analyzujte požadavky pro testování studentských projektů v předmětech IZP, IOS, IPS a ITS. Navrhněte systém pro testování a hodnocení studentských projektů. Uvažujte projekty v různých programovacích jazycích, různých prostředích (příkazová řádka, síťová služba, různé distribuce), historie testů, uvažujte s automatickými testy i ruční revizí kódu a dalších souborů. Systém by měl také zahrnovat tvorbu reportů se základními statistikami.
3. Implementujte navržený systém s využitím technologie linuxových kontejnerů.
4. Ověřte framework pomocí automatických integračních testů.

Literatura:

- P. Ammann, J. Offutt. *Introduction to Software Testing*, Cambridge University Press, 2008. ISBN 978-0-511-39330-3.
- Rosen, R.: Linux Containers and the Future Cloud. Linux Journal. 2014-06-10. URL: http://www.linuxjournal.com/content/linux-containers-and-future-cloud
- Domovská stránka projektu Docker. URL: https://www.docker.com/

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Táto diplomová práca sa zaoberá návrhom a implementáciou frameworku, ktorého cieľom je zvýšiť efektivitu a zjednodušiť prácu pri testovaní a hodnotení študentských projektov. Potrebné štúdium je popísané v teoretickej časti, ktorá je zameraná na základné princípy a typy testovania softvéru. Venuje sa aj problematike technológie linuxových kontajnerov. Ďalej sú v práci analyzované požiadavky na testovanie študentských projektov v rôznych predmetoch. Následne je navrhnutý a implementovaný systém, ktorý pokrýva analyzované požiadavky. Posledná časť práce sa zaoberá overením správnej funkčnosti systému a popisom ďalších možností rozšírenia práce.

Abstract

This Master's Thesis is about design and implementation of a framework, whose target is to improve effectiveness and simplify student project's evaluation process. Theoretical part of this Thesis is dedicated to software testing fundamentals and used principles. It also describes Linux containerization technology. In the next part, Thesis contains analysis of requirements for student project testing in various University courses. Core of the Thesis describes design and its implementation of a system, which satisfies analyzed requirements. Last part shows how implemented system was verified and shows possible future extensions of this work.

Kľúčové slová

framework, verifikácia systému, statická analýza, dynamická analýza, revízia kódu, testovanie softvéru, integračné testy, jednotkové testy, linuxové kontajnery, python, ncurses, textové užívateľské rozhranie, študentské projekty

Keywords

framework, system verification, static analysis, dynamic analysis, code review, software testing, integration tests, unit tests, linux containers, python, ncurses, text-based user interface, student's projects

Citácia

DIŽOVÁ, Natália. *Framework pro testování studentských projektů*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Framework pro testování studentských projektů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením Ing. Aleša Smrčku, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Natália Dižová
16. mája 2022

Poďakovanie

Rada by som sa poďakovala vedúcemu tejto diplomovej práce Ing. Alešovi Smrčkovi, Ph.D za jeho podporu, odborné konzultácie, užitočné rady a pripomienky počas vypracovávania tejto práce.

Obsah

Kapitola 1

Úvod

Oblasť informačných technológii sa neustále posúva dopredu a je čím ďalej tým viac rozšírená vo svete. Čoraz viac mladých ľudí sa vyberá práve smerom k informačným technológiám a má záujem vzdelávať sa v tomto obore. Štúdium informačných technológii je z obrovskej časti zamerané na prax v podobe tvorby študentských projektov. Počas štúdia sa dostávame do kontaktu s rôznymi oblasťami informatiky, pričom každá vyžaduje rôzny prístup k tvorbe projektov a tým pádom aj k testovaniu výsledných projektov.

Testovanie softvéru tvorí neodmysliteľnú súčasť vývoja informačných technológii. Ide o komplexnú disciplínu, ktorá vyžaduje množstvo času a úsilia pre vyprodukovanie vhodných výsledkov. V prípade študentských projektov vyžaduje testovanie nadmernú pozornosť a trpezlivosť. Nejde len o zhodnotenie istej kvality projektu či vytvorenie bodového hodnotenia, ale ide najmä o snahu vyučujúcich predať študentom s každým jedným projektom čo najviac skúsenosti a spätnej väzby. Vďaka tomu sa môžu študenti stále posúvať dopredu, tvoriť kvalitnejšie programy a odniesť si z každého vytvoreného projektu ďalšie poznatky a skúsenosti, ktoré následne využijú aj v profesnom živote.

Touto diplomovou prácou chcem podporiť vyučujúcich a všetkých, ktorí sa podieľajú na opravovaní študentských projektov, a zjednodušiť im prácu pri testovaní a hodnotení projektov z rôznych informačných oblastí.

Potrebná teória k tejto práci je popísaná v kapitolách 2 a 3, ktoré sa zaoberajú rôznymi metódami a prostriedkami pre overovanie softvéru. Cieľom kapitoly 2 je priblížiť základné typy testovania z pohľadu jednotlivých fáz vývoja softvéru a popísať proces vykonávania testu. V kapitole 3 je následne predstavená technológia linuxových kontajnerov ako prostriedok pre izoláciu vhodnú na testovanie softvéru.

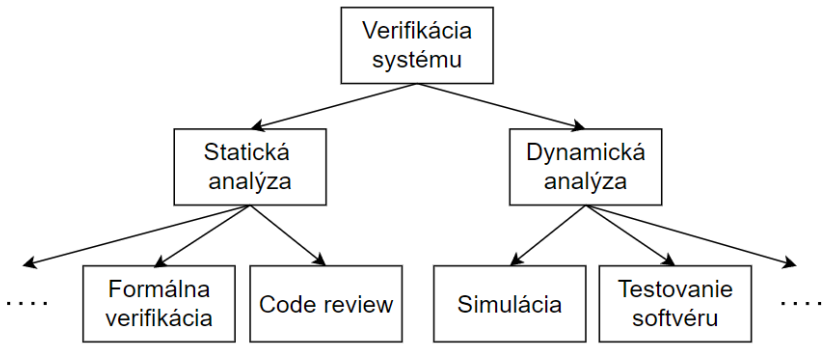
V druhej časti tejto práce je predstavený nový framework pre testovanie študentských projektov. Kapitola 4 je zameraná analýzu požiadaviek a špecifík rôznych projektov z rôznych vyučovaných predmetov. Ďalej sa táto kapitola zaoberá výberom vhodných technológii a návrhom riešenia systému pre testovanie študentských projektov, ktorý pokrýva analyzované požiadavky. V kapitole 5 je následne popísaná implementácia jednotlivých častí systému vytvoreného podľa návrhu. Kapitola 6 sa zaoberá overením správneho fungovania implementovaného systému s využitím automatických integračných testov. Na záver sú v práci zhrnuté dosiahnuté výsledky spolu s priblížením ďalších možností rozšírenia.

Kapitola 2

Testovanie softvéru

Táto kapitola sa zaoberá základnými princípmi a metódami pri overovaní softvéru. Zameriava sa najmä na testovanie softvéru kde popisuje jednotlivé fázy vykonávania testov a rozdelenie na rôzne typy testovania. Nakoniec sú predstavené aj niektoré metódy testovania na úrovni integračných testov a testov jednotiek.

Na proces overovania systému možno celkovo nahliadnuť z pohľadu verifikácie alebo validácie. Validácia sa zaoberá otázkou, či je budovaný správny systém (teda vyhovuje potrebám koncového užívateľa). Verifikácia rieši otázku, či je systém budovaný správne (teda vyhovuje zadanej špecifikácii) [13]. Metódy verifikácie ďalej delíme podľa ich prístupu na statickú a dynamickú analýzu (viď obrázok 2.1).



Obr. 2.1: Klasifikácia metód verifikácie systému [11]

Statická analýza skúma vlastnosti softvéru bez jeho spustenia. Dá sa teda efektívne využiť už v počiatočných fázach vývoja softvéru (viď obrázok 2.3), kedy ešte nie je dostupný jeho funkčný prototyp. Táto technika sa používa napríklad pre analýzu a kontrolu zdrojového kódu tzv. *code review* alebo kontrolu špecifikácie požiadaviek napríklad pomocou formálnej verifikácie.

Dynamická analýza naopak overuje vlastnosti softvéru na základe spustenia jeho kódu. Využíva sa v neskorších vývojových fázach, kde sa už vyžaduje spustiteľná verzia softvéru. Formou dynamickej analýzy môže byť napríklad simulácia, ktorá sa vykonáva nad modelom systému alebo testovanie softvéru, ktoré sa vykonáva nad samotným spustiteľným programom sledovaním jeho behu.

Podľa odbornej literatúry možno testovanie softvéru definovať ako proces vykonávania programu za účelom nájdenia chýb [3]. Existencia softvérovej chyby je podmienená splnením aspoň jednej z nasledujúcich podmienok:

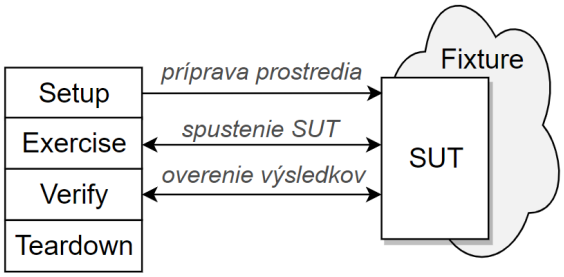
- softvér nerobí niečo, čo by podľa špecifikácie robiť mal,
- softvér robí niečo, čo by podľa špecifikácie robiť nemal,
- softvér robí niečo, o čom špecifikácia nehovorí,
- softvér nerobí niečo, o čom špecifikácia nehovorí, ale mala by hovoriť,
- softvér je náročný na pochopenie, ťažko sa s ním pracuje, je pomalý alebo ho koncový používateľ nebude považovať za správny [7].

Cieľom testovania je teda odhaliť čo najviac chýb čo najskôr a zvýšiť tak kvalitu testovaného systému, ktorému sa v kontexte testovania hovorí tiež SUT (system under test).

2.1 Fázy vykonávania testu

Základný princíp vykonávania testov je členený na štyri základné fázy. Pôvodne bolo toto rozdelenie definované pre jednotkové testovanie, avšak dá sa aplikovať na všetky úrovne testovania (viď sekcia 2.2) [11]. Na obrázku 2.2 sú znázornené jednotlivé fázy vykonávania testu a ich interakcia s SUT:

- **Setup**: Prípravná fáza, ktorá pozostáva z nastavenia kontextu alebo vytvorenia stáleho prostredia pre SUT. Výsledkom sú externe pripravené umelé dáta (testovacie vstupy) potrebné pre testovanú jednotku. Pripravenému prostrediu, spolu so všetkými vstupnými dátami, sa hovorí *test fixture* (viď sekcia 2.1.1).
- **Exercise**: Spočíva v prenechaní riadenia testovanému subjektu SUT, spolu s pripravenými testovacími vstupmi z časti setup. SUT následne vykonáva svoju činnosť a prípadne z nej vráti aj nejaký výsledok.
- **Verify**: Overovacia fáza, počas ktorej sa kontrolujú dosiahnuté výsledky a zmenený stav SUT. Porovnávajú sa očakávané výsledky s reálnymi výsledkami, ktoré SUT vyprodukovalo nad zadanými vstupmi z časti setup.
- **Teardown**: Čistiaca fáza, počas ktorej sa vykonáva čistenie vedľajších účinkov z testu.



Obr. 2.2: Interakcia jednotlivých fáz testu so systémom [6]

2.1.1 Vzory pevného okolia testovanej jednotky

Pri testovaní softvéru, resp. pri vytváraní samotného testu je dôležité, aby spomínaná prípravná fáza setup zaručila stálosť testu. Test sa považuje za stály, ak jeho opakovaným vykonávaním dopadne test vždy rovnako, s rovnakými výsledkami. Preto sa zavádzajú takzvané vzory pevného okolia testovanej jednotky, nazývané tiež *test fixture*.

Test fixture predstavuje všetky vstupné dáta pre SUT také, aby bolo možné spustiť test opakovane a to bez ohľadu na aktuálny kontext [11]. Zahŕňa kolekciu všetkých vstupných dát, ktoré by mohli nejakým spôsobom ovplyvňovať priebeh daného testu.

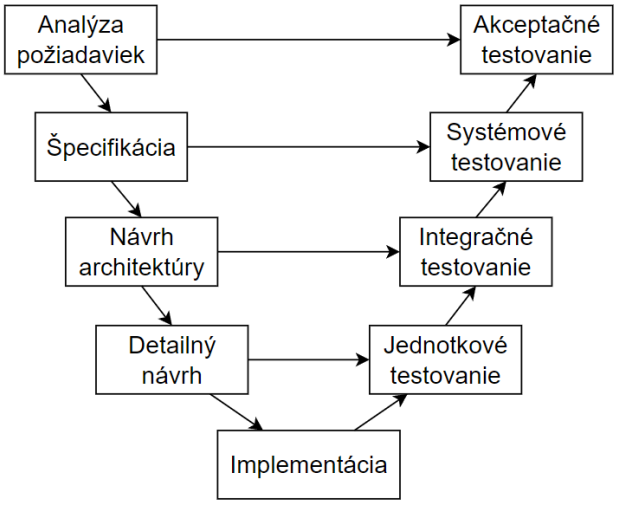
Na pojem test fixture existuje niekoľko rôznych pohľadov a môže mať viacero významov. Pre účely tejto diplomovej práce môžeme chápať test fixture ako objekt, voči ktorému prebieha test, pričom tento objekt musí pred každým testom zostať v známom pevnom stave aby produkoval rovnaký výsledok [5].

Pre prípravu pevného okolia testovanej jednotky existuje niekoľko stratégii [11]:

- **Transient Fresh Fixture**: Stratégia kde sa pri každom teste vytvára nový fixture (vo fáze setup) a po teste celý zaniká (vo fáze teardown).
- **Persistent Fresh Fixture**: Pre každý test je fixture unikátny, ale existuje pred aj po vykonaní testu a slúži tak pre niekoľko rôznych testov, ktoré majú rovnaký cieľ.
- **Shared Fixture Strategies**: Rôzne stratégie, ktorých cieľom je znovupoužitie fixture pre viacero testov, ktoré tak majú zdieľané okolie, ale navzájom sa neovplyvňujú.

2.2 Úrovne testovania z pohľadu vývoja softvéru

Základné úrovne testovania sú definované podľa toho, v akej fáze vývojového cyklu softvéru sa testovanie vykonáva. Na obrázku 2.3 je znázornený jeden z najznámejších vývojových modelov nazývaný V-model, podľa ktorého delíme úrovne testovania na jednotkové, integračné, systémové a akceptačné testovanie. Nasledujúci popis jednotlivých úrovní vychádza najmä z knihy *Software Testing: Principles and Practices* [13].



Obr. 2.3: Vývojový V-model [8]

2.2.1 Jednotkové testovanie

Na najnižšej úrovni V-modelu (viď obrázok 2.3) sa vykonáva testovanie jednotiek, nazývané tiež *unit testing*. Existuje niekoľko definícií pre unit testing, pričom všetky zahŕňajú nasledujúce tri najpodstatnejšie atribúty. Jednotkové testovanie je (väčšinou automatizovaný) proces, ktorý:

- overuje malý kus kódu (jednotku, fragment, modul),
- vykonáva sa rýchlo,
- vykonáva sa izolovaným spôsobom [5].

Ide teda o overenie správnej implementácie najmenších stavebných blokov programu, resp. fragmentov kódu, akými sú napríklad procedúra alebo funkcia v procedurálnych jazykoch, jednoduché triedy alebo metódy v objektovo orientovaných programovacích jazykoch, klauzule v Prologu a podobne [11].

Tvorba jednotkových testov je zvyčajne realizovaná priamo programátormi jednotiek, pričom ich cieľom je odhaliť čo najviac chýb na najnižšej úrovni a zmenšiť tak rozsah testovania na vyšších úrovniach. Jednotkové testovanie zvyčajne prebieha opakovaným spúšťaním jednotlivých fragmentov nad rôznymi vstupmi a overuje sa správnosť výstupov podľa vopred definovaných očakávaní.

V niektorých literatúrach sa naviac uvádza samostatná úroveň testovania modulov, ktorá sa ale väčšinou zahŕňa pod úroveň jednotkového testovania. Modul je kolekcia súvisiacich jednotiek, ktoré sú zoskupené v jednom súbore, triede, balíčku alebo rozšírení. Cieľom testovania modulov je porovnať funkciu modulu s funkčnou špecifikáciou alebo špecifikáciou rozhrania, ktorá modul definuje [3].

2.2.2 Integračné testovanie

Integrácia je definovaná ako súbor interakcií medzi komponentami. Testovanie interakcie medzi modulmi sa nazýva integračné testovanie a jeho predpokladom je správne fungovanie samostatných jednotiek systému. Integračné testovanie je zamerané na overenie rozhrania modulov v podsystéme, ich vzájomnú spoluprácu a komunikáciu.

Počas integračného testovania by sa mali brať do úvahy všetky typy rozhraní, či už interné, externé, implicitné alebo explicitné. Interné rozhranie poskytuje komunikáciu modulov interne v rámci projektu alebo produktu. Externé rozhranie je viditeľné aj mimo produktu pre vývojárov tretích strán. Explicitné rozhranie je riadne popísané a zdokumentované, zatiaľ čo implicitné rozhranie je interne známe softvérovým inžinierom, ale nie je zdokumentované.

2.2.3 Systémové testovanie

Táto fáza testovania sa vykonáva až po jednotkovom testovaní a integračnom testovaní. Predpokladom je, že jednotlivé časti systému pracujú správne.

Pri systémovom testovaní sú spojené jednotlivé subsystémy a vykonáva sa analýza a testovanie systému ako celku. Cieľom je overiť, či softvér ako celok spĺňa špecifikáciu požiadaviek a odhaliť tak prípadné chyby v návrhu a špecifikácii. V tejto fáze sa overujú ako funkčné tak aj nefunkčné aspekty. Po funkčnej stránke sa testovanie systému zameriava na reálne používanie produktu zákazníkom. Po nefunkčnej stránke sa zameriava na parametre týkajúce sa výkonu, rozšíriteľnosti, spoľahlivosti, prevádzkyschopnosti a podobne.

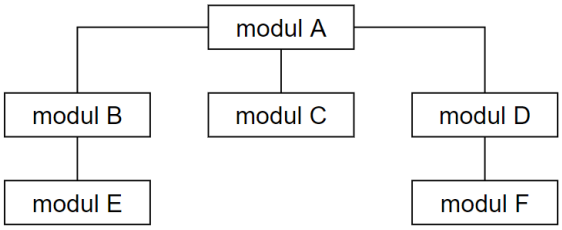
2.2.4 Akceptačné testovanie

Na najvyššej úrovni V-modelu (viď obrázok 2.3) sa vykonáva akceptačné testovanie. Vykonáva ho priamo zákazník, prípadne zástupca zákazníka a realizuje sa na základe vopred dohodnutých prípadov použití od zákazníka.

Cieľom akceptačného testovania je overiť, či výsledný produkt pracuje správne v reálnom nasadení a či spĺňa požiadavky zákazníka. Zameriava sa napríklad na overenie komplexnej funkčnosti, testujú sa obchodné scenáre, používateľské scenáre, dodržiavanie právnych zmlúv a podobne.

2.3 Metódy testovania

Každý systém je zložený z viacerých komponentov alebo modulov (viď obrázok 2.4), ktoré je potrebné otestovať. Pre vytvorenie testovacej sady, ktorá pokrýva všetky komponenty a rozhrania je vhodné si zaviesť správne poradie a prístup testovania. V tejto sekcii sú popísané základné metódy testovania na úrovni jednotkových, modulových a integračných testov. Rozlišujeme dva základné prístupy testovania: inkrementálny (*incremental testing*) a neinkrementálny (*nonincremental testing*).



Obr. 2.4: Príklad štruktúry systému zloženého zo šiestich modulov

2.3.1 Neinkrementálne testovanie

Neinkrementálny prístup testovania, nazývaný tiež *big bang testing*, je založený na dvoch fázach. V prvej fáze sa testuje každý modul izolovane a v druhej sa testuje jeden celok, ktorý je zložený zo všetkých modulov. Moduly môžu byť testované súčasne (paralelne) alebo postupne (sériovo). Testovanie každého modulu si vyžaduje špeciálny ovládací modul pre riadenie testovania (predávanie vstupných argumentov, zobrazovanie výsledkov, prenos testovacích prípadov cez modul...) a jeden alebo viac modulov *stub* pre simuláciu funkcii iného modulu v systéme [3].

Obrázok 2.4 znázorňuje príklad systému, ktorý pozostáva zo šiestich modulov A až F. Takýto systém by sa neinkrementálnou metódou testoval nasledovne. Všetkých šesť modulov sa najskôr otestuje jednotlivo ako samostatné jednotky. Nakoniec sa moduly kombinujú a integrujú aby vytvorili jeden program. Integrácia zahrňujúca všetky moduly sa nazýva systémová integrácia alebo tiež *final integration testing* [13].

Výhodou neinkrementálneho prístupu je, že v prvej fáze poskytuje priestor pre paralelné testovanie modulov. Testovanie všetkých modulov súčasne prináša obrovskú výhodu najmä vo veľkých projektoch skladajúcich sa z mnohých modulov [3].

2.3.2 Inkrementálne testovanie

Princípom inkrementálneho testovania je postupné pridávanie ďalších modulov do sady testovaných modulov. Namiesto testovania každého modulu izolovane sa nasledujúci testovaný modul skombinuje so sadou modulov, ktoré už boli testované. Podľa poradia v akom sa testujú moduly v rámci systému, rozlišujeme:

- **Testovanie zdola nahor**: V prípade systému na obrázku 2.4 je prvým krokom testovanie samostatných modulov E, C a F (paralelne alebo sériovo). V ďalšom kroku sa testujú moduly B a D tak, že sú kombinované s modulmi E a F (teda sa netestujú izolovane) [3]. Tento prístup vyžaduje päť ovládacích modulov, ale žiadne stub moduly.
- **Testovanie zhora nadol**: Začína sa počiatočným modulom v systéme, teda v prípade systému na obrázku 2.4 je prvým krokom testovanie modulu A. Keďže v tomto štádiu nie sú ešte overené moduly B, C a D, musia sa nahradiť modulmi stub, ktoré budú simulovať ich funkcie [3]. Ďalej sa pokračuje testovaním smerom k spodným modulom, pričom predtým testované moduly budú použité namiesto stub modulov.

Výhodou inkrementálneho testovania je, že sa implementované chyby v rozhraní prejavia skôr ako pri neinkrementálnom prístupe, kde sa moduly kombinovane testujú až na konci.

Kapitola 3

Linuxové kontajnery

Pri vývoji softvéru ako aj pri jeho testovaní sa často stretávame s pojmom kontajner. V tejto kapitole je v krátkosti predstavený princíp linuxových kontajnerov so zameraním na ich využitie. Následne sú spomenuté niektoré hlavné výhody použitia kontajnerov a základný rozdiel oproti virtuálnym strojom. Na konci tejto kapitoly je stručný popis jedného z nástrojov, slúžiacich pre prístup ku kontajnerom, nazývaný Docker.

3.1 Princíp linuxových kontajnerov

Kontajnerová infraštruktúra založená na Linuxe poskytuje svojím používateľom prostredie blížiace sa čo najviac k štandardnej distribúcii Linuxu [9]. Hlavnou myšlienkou kontajnerov je poskytnúť minimálnu a čo najviac prenosnú sadu požiadaviek, ktoré vyžaduje aplikačný alebo softvérový balík na spustenie [12]. Ide teda o prostriedok na balenie aplikácií, rámcov, knižníc a iných závislostí štandardizovaným spôsobom. S každým kontajnerom sa dá vďaka štandardizácii zaobchádzať rovnako, bez ohľadu na to, aká aplikácia v ňom beží [10].

Kontajnery sú špeciálne zapuzdrené a zabezpečené procesy bežiace na hostiteľskom systéme, ktoré môžu mať izolovaný súborový systém, sieť a menný priestor pre procesy. Všetky procesy bežiace v kontajneroch zdieľajú rovnaké linuxové jadro základného hostiteľského operačného systému [10]. Tým sa kontajnery najviac odlišujú od virtuálnych strojov, ktoré majú každý svoj vlastný plnohodnotný operačný systém. Oproti virtuálnym strojom majú kontajnery tiež menšiu réžiu a doba spustenia typického kontajnera je výrazne menšia.

Použitie kontajnerov pri vývoji softvéru rieši niekoľko problémov:

- reprodukovateľnosť: zabalená aplikácia s prostredím sa môže jednoducho zdieľať medzi produkciami, teda je zaručená konzistencia,
- prenositelnosť: aplikáciu je takýmto spôsobom možné spustiť na rôznych distribúciách a v rôznych prostrediach,
- vývojové prostredie: využitie kontajneru rieši problémy spojené s rôznymi verziami knižníc v lokálnom vývojárskom prostredí,
- simulácia produkcie: vývojár nemusí kvôli testovaniu aplikácie inštalovať potrebné nástroje, databázy a pod. k sebe, ale stačí vytvoriť kontajner s požadovaným prostredím a spustiť aplikáciu v ňom,
- čistenie po testovaní: po vykonaní testu sa rýchlo a jednoducho odstráni použitý kontajner a vývojársky počítač ostáva čistý,

- izolácia: všetky procesy bežia izolovane v kontajnery, vďaka čomu zostáva lokálne prostredie chránené, čisté a neporušené [2].

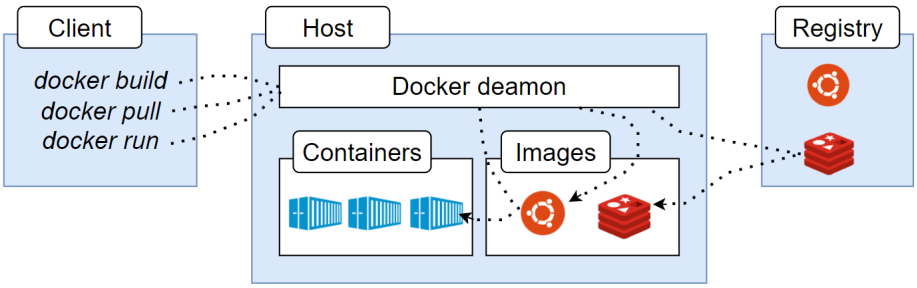
3.2 Docker kontajnery

Docker predstavuje sadu nástrojov a rozhraní používaných pre vývoj, správu, posielanie a spúšťanie aplikácii vo forme kontajnerov. Umožňuje oddeliť aplikácie od infraštruktúry za účelom rýchleho a konzistentného dodania softvéru [1]. Vďaka kontajnerom, ktoré obsahujú všetko potrebné na spustenie aplikácie sa netreba spoliehať na to, čo je aktuálne nainštalované na hostiteľskom systéme. Kontajnery možno jednoducho zdieľať s istotou, že bude fungovať u každého rovnakým spôsobom (či už na osobnom počítači vývojára, fyzickom alebo virtuálnom počítači v dátovom centre alebo v rôznych prostrediach) [1].

Docker využíva rozhrania linuxového jadra, pričom najdôležitejšie z nich sú rozhranie menného priestoru (*namespaces*) a riadiacich skupín (*control groups*) [12]. Namespace je špeciálna dátová štruktúra Linuxového jadra, vďaka ktorej sa vytvára logická izolácia aplikácii a riadi viditeľnosť zdrojov pre procesy [4]. Control groups sú ovládacie prvky, ktoré poskytujú zdieľanie zdrojov pre procesy v rámci jedného namespace. Control groups sa vytvára samostatne pre každý zdroj, ktorý chceme ovládať.

Vo svete Docker kontajnerov sa často vyskytujú nasledujúce pojmy [12]:

- *image* - statický objekt poskytujúci šablónu, ktorá slúži na vytváranie kontajnerov,
- *container* - kontajner vytvorený z nejakého *image* ako jeho spustiteľná inštancia,
- *register* - úložisko, ktoré slúži na ukladanie a distribúciu objektov *image*,
- *host* - hostiteľský systém, na ktorom je nainštalovaná aplikácia Docker,
- *client* - klient poskytujúci interakciu užívateľa s Dockerom na vytváranie príkazov pre správu kontajnerov, ktoré sa potom predávajú démonom na vykonanie,
- *deamon* - démon slúži pre správu objektov na hostiteľovi a prijíma príkazy od klienta,
- *dockerfile* - konfiguračný súbor popisujúci kroky potrebné na zostavenie objektu *image*.



Obr. 3.1: Architektúra systému Docker [1]

Na obrázku 3.1 sú znázornené spomínané pojmy a ich vzájomné vzťahy a prepojenie v rámci architektúry Docker. Docker je založený na architektúre klient-server. Klient komunikuje s jedným alebo viacerými démonmi prostredníctvom rozhrania API, na ktorom démon prijíma od klienta požiadavky. Na hostiteľskom systéme beží démon, ktorý vykonáva príkazy napríklad pre správu objektov image, pre vytvorenie kontajneru z daného image alebo pre prácu s registrami a podobne [1].

Kapitola 4

Testovanie študentských projektov

Táto kapitola sa zaoberá riešením problému spojeného s opravovaním a hodnotením študentských projektov na *Fakulte informačních technológii VUT v Brně*. Na úvod je predstavená krátka motivácia, ciele a analýza existujúcich riešení. Následne sú dôsledne analyzované požiadavky na nový testovací framework a na záver je popísaný návrh na riešenie a pokrytie týchto požiadaviek.

Proces testovania a hodnotenia študentských projektov spočíva z viacerých aktivít. Pred samotným testovaním je potrebné skontrolovať správny formát odovzdaných súborov či ich pomenovanie, následne overiť správnu funkčnosť projektu prostredníctvom testov, ohodnotiť zrozumiteľnosť a čitateľnosť kódu a dokumentácie, vytvoriť celkové ohodnotenie projektu, vložiť ohodnotenie do systému prípadne hodnotenie poslať e-mailom študentovi s prípadnou ďalšou spätnou väzbou a mnoho ďalších aktivít. Celý tento proces je časovo náročný a niektoré činnosti vyučujúci častokrát vykonávajú opakovane. Niektoré akcie sa však dajú automatizovať čo vedie k myšlienke využiť pre testovanie študentských projektov vhodný framework.

Existuje niekoľko testovacích frameworkov, no väčšina z nich je zameraná práve na jeden programovací jazyk alebo jedno prostredie, teda by sa dali využiť len na jeden konkrétny typ projektu. V rámci štúdia sa však stretávame s projektami v rôznych predmetoch, ktoré sú koncipované rôznym spôsobom a sú zamerané na rôzne oblasti a problematiky. Preto by mal framework, vhodný pre testovanie študentských projektov, podporovať projekty v rôznych programovacích jazykoch a v rôznych prostrediach. Treba brať do úvahy, že pre jedno zadanie projektu existuje veľa odlišných riešení a nie len jedno dopredu dané riešenie alebo len nejaký vzor riešení.

Motiváciou pre vytvorenie nového frameworku bolo hlavne zjednodušiť a spríjemniť vyučujúcim prácu pri opravovaní študentských projektov. Cieľom je čo najviac zefektívniť prácu minimalizovaním vykonávania opakovaných akcií, resp. umožniť automatické vykonávanie opakovaných úkonov nad projektami a tiež umožniť vykonávanie hromadných akcií nad viacerými projektami. Veľký dôraz je kladený aj na užívateľské ovládanie, ktoré musí byť intuitívne pre vyučujúcich a tiež čo najefektívnejšie.

Výsledný framework pomôže nielen vyučujúcim ušetriť ich čas, ale aj študentom, ktorí vďaka tomu dostanú rýchlejšiu spätnú väzbu s ohodnotením a ďalšími poznámkami k vytvoreným projektom.

4.1 Analýza požiadaviek

V tejto sekcii je poskytnutý celkový prehľad požiadaviek pre testovanie študentských projektov. Na začiatok sú popísané špecifiká typických študentských projektov v rôznych vyučovaných predmetoch. Následne sú analyzované funkčné požiadavky pre testovací framework a na záver sú zhrnuté aj základné nefunkčné požiadavky.

4.1.1 Špecifiká študentských projektov vo vybraných predmetoch

V rámci úvodnej štúdie pre túto prácu bolo potrebné zoznámiť sa s procesom testovania a hodnotenia študentských projektov na *Fakulte informačních technológii VUT v Brně*. Cieľom bolo analyzovať špecifické požiadavky typických projektov so zameraním predovšetkým na predmety *Základy programování*, *Operační systémy*, *Programovací seminář* a *Testování a dynamická analýza*:

- **Základy programování** (IZP) je jeden z predmetov, s ktorými sa študenti stretnú hneď v prvom semestri bakalárskeho štúdia. Väčšina z nich sa práve v tomto predmete prvýkrát zoznamuje so základmi programovania a majú možnosť si prvýkrát vyskúšať riešiť rôzne problémy a navrhovať jednoduché algoritmy na ich riešenie. Cieľom projektov a cvičení v tomto predmete je typicky vytvorenie jednoduchej funkcie, prípadne doplnenie preddefinovanej kostry programu. Pri opravovaní takýchto projektov ide teda väčšinou o testovanie malých jednotiek kódu, kde sa typicky využíva automatizácia jednotkového testovania. Framework pre testovanie študentských projektov by teda mal poskytnúť jednoduché spustenie automatických testov spolu s automatickým priradením výsledkov testov k danému projektu.

  V predmete IZP sa študenti učia nielen programovacie konštrukty a algoritmy, ale aj základné princípy, ktoré by mal správny programátor dodržiavať. Preto je potrebné kontrolovať okrem funkčnosti projektov aj príslušnú dokumentáciu, komentovanie kódu a podobne. Framework by mal teda ponúkať aj možnosť vytvorenia poznámky k projektu, prostredníctvom ktorej môže vyučujúci predať študentovi rady, prípadne poukázať na nedostatky kódu. Celkovo tak vznikne možnosť vykonávať statickú analýzu projektu (code review) a predať tak študentom spätnú väzbu nielen vo forme bodového ohodnotenia ale aj vo forme textového vyjadrenia.

- Jeden z typických projektov v predmete **Operační systémy** (IOS) je zameraný na synchronizáciu procesov v prostredí UNIX. Tento predmet je vyučovaný v prvom ročníku bakalárskeho štúdia, kedy študenti stále nemajú dostatok skúsenosti s programovaním. Veľmi často sa teda stáva, že študenti v projekte vytvoria program, ktorý vykonáva nesprávnu synchronizáciu procesov alebo po sebe nečistí a môže zanechávať neporiadok v systéme. Ak následne vyučujúci testuje takýto program na reálnom stroji, musí potom vynaložiť špeciálne úsilie pre čistenie a pre navrátenie do pôvodného stavu. Preto je dôležité aby testovanie študentských projektov prebiehalo v izolovanom prostredí, čím sa výrazne zjednoduší čistiaca fáza testovania projektov.

- V predmete **Programovací seminář** (IPS) sú projekty typicky zamerané na prácu s vláknami a procesmi, ich synchronizáciu a paralelizáciu, dynamické prideľovanie pamäte a podobne. Tento predmet sa vyučuje až v druhom ročníku bakalárskeho štúdia, teda sa predpokladá, že študent už ovláda základy programovania, algoritmizácie a základné synchronizačné primitíva. Z toho vyplýva, že projekty sú o čosi náročnejšie

(napr. oproti predmetu IOS), čím sa otvára väčší priestor na chyby študentov pri práci na projektoch, čo vedie k ešte väčšej potrebe izolácie projektov počas ich testovania.

- Projekty z predmetu **Testování a dynamická analýza** (ITS) sa zaoberajú najmä návrhom a implementáciou automatizovanej testovacej sady. Pri tvorbe testovacej sady sa typicky definujú rôzne testované scenáre prostredníctvom textu. Pre ohodnotenie projektu je teda potrebné prechádzať textové súbory a prípadne k nim pridávať adekvátne poznámky.

Väčšinou sa v rámci všetkých študentských projektov vyžaduje aj tvorba príslušnej dokumentácie k projektu. Bežne sa stáva, že študenti neodovzdajú požadovanú dokumentáciu, prípadne nestihnú vypracovať projekt v plnej podobe a odovzdajú len čiastočné riešenie. V takýchto prípadoch je potrebné zohľadniť chýbajúce časti riešenia v hodnotení projektu. Preto okrem vytvorenia automatického hodnotenia pomocou automatizovaných testov a prípadného ďalšieho hodnotenia a vytvárania poznámok v rámci code review, je požadované mať počas testovania študentských projektov možnosť pridať či upraviť hodnotenie projektu aj manuálne.

4.1.2 Funkčné požiadavky

Medzi požiadavkami pre testovanie študentských projektov v rôznych predmetoch sa podarilo nájsť niektoré zhodné a niektoré špecifické požiadavky a ich zjednotením vytvoriť zoznam funkčných požiadaviek pre nový framework.

Funkčné požiadavky možno rozdeliť do štyroch základných kategórii, kde sa následne rozdeľujú do ďalších menších podkategórii. Medzi základné kategórie funkčných požiadaviek patrí:

- užívateľské ovládanie a prechádzanie dát (viď tabuľka 4.1),
- správa testov (viď tabuľka 4.2),
- typy testov a spôsob ich vykonávania (viď tabuľka 4.3),
- reportovanie (viď tabuľka 4.4).

Následne sú v tejto sekcii analyzované a popísané jednotlivé požiadavky rozdelené medzi spomínané základné kategórie.

Požiadavky na užívateľské ovládanie a prechádzanie dát

Prvá kategória požiadaviek je zameraná na to, ako užívateľ systém ovláda a s akými dátami pracuje. Zoznam týchto požiadaviek je štruktúrovaný do tabuľky 4.1, kde má každá požiadavka svoje jedinečné číslo, ktorým sa dá identifikovať.

Vyučujúci spomínaných predmetov si zvykli používať pre testovanie študentských projektov buď príkazový riadok, skripty alebo súborový manažér *Midnight Commander*. Nový testovací framework by mal poskytnúť jednoduché textové užívateľské prostredie (podobné spomínanému nástroju *Midnight Commander*). Pre interakciu užívateľa so systémom sa požaduje využívať primárne klávesnicu so zaužívanými klávesovými skratkami. Podpora ovládania myšou nie je vyžadovaná, nakoľko myš nie je vhodná na rýchlu prácu, pretože by užívateľ musel zbytočne vynaložiť ďalšie úsilie aby presúval ruku medzi klávesnicou a myšou, čo by spomalilo celý proces práce s frameworkom.

Pre prechádzanie a zobrazovanie dát (či už súborov alebo adresárov) je potrebné zvažovať viaceré užívateľské pohľady, ktoré sa líšia najmä v tom, aké dáta užívateľ v danú chvíľu vidí. Základným pohľadom je zobrazenie obsahu adresárovej štruktúry, kde užívateľ prechádza medzi adresármi a prezerá ich obsah (napríklad adresáre študentov, ktoré obsahujú súbory odovzdaných projektov). Ďalším pohľadom je zobrazenie obsahu samotného súboru, pričom užívateľ môže vidieť buď len rýchly náhľad do súboru (typicky pár riadkov súboru) alebo sa zobrazí obsah celého súboru. V režime rýchleho náhľadu nie je možné nijak upravovať daný súbor a užívateľ pomocou šípok na klávesnici prechádza medzi súbormi, pričom sa mu vedľa zobrazuje náhľad súboru, ktorý má zrovna vyznačený. V prípade zobrazenia celého obsahu súboru je možné aj upravovať obsah tohto súboru či pridávať k nemu poznámky a užívateľ v tomto režime pomocou šípok prechádza obsah len tohto jedného otvoreného súboru.

Jedným z najväčších prínosov tohto frameworku je zavedenie tzv. systému značiek. Značka (tag) nesie užitočnú informáciu a môže byť pridaná k nejakému súboru. Každá značka je identifikovaná pomocou špeciálneho znaku „`#`“ a má svoj jedinečný názov, ktorý ju charakterizuje. Typy značiek sa rozlišujú na parametrické (s jedným alebo viacerými parametrami) a neparametrické (bez parametrov).

Zavedenie značiek prináša mnohé výhody pri testovaní študentských projektov. Je možné vďaka nim jednoducho označiť projekt napríklad ako samostatný, skupinový (kde parametrom môže byť identifikátor skupiny), prípadne ho označiť za plagiátorský (napr. pomocou neparametrickej značky `#plagiat`). Tiež je možné značky využiť pre pridávanie bodového hodnotenia po vykonaní testu k projektu (napr. pomocou značky `#test_1(5)`, kde parameter 5 udáva počet bodov za daný test). Framework by mal teda podporovať automatické pridávanie značiek z výsledkov testov a manuálne značkovanie. Správa značiek potom zahŕňa pridanie či odstránenie značky alebo zmena hodnoty parametra existujúcej značky.

Značky je možné následne využiť aj pre filtráciu súborov. Pri testovaní študentských projektov sa môže stať, že niektorý z definovaných testov má v sebe chybu a je potrebné ho zmeniť a opraviť. Táto chyba sa môže odhaliť až po vykonaní testov a pridelení bodového hodnotenia študentom. V takomto prípade treba znovu vykonať testy a skontrolovať, ktoré bodové hodnotenia sa zmenili a upraviť ich. S využitím značiek sa dá jednoducho filtrovať len tie projekty, ktorých sa zmena v danom teste týka a netreba tak zdĺhavo prehľadávať celú štruktúru testov a projektov, aby sa prípadné nejasnosti napravili.

Framework by mal umožňovať filtrovanie na základe spomínaných značiek a mal by tiež poskytnúť možnosť rekurzívneho filtru nad súbormi v adresárovej štruktúre. Filter nad súbormi môže byť zadaný buď podľa cesty (kde sa vyhľadajú súbory v rámci adresárovej štruktúry, ktorých cesta odpovedá zadanému výrazu) alebo podľa obsahu (kde sa vyhľadajú súbory, ktoré obsahujú zadaný reťazec).

Filtrované súbory je potom možné prechádzať a vykonávať nad nimi rôzne skupinové akcie, ako napríklad pridanie poznámky, pridanie značky alebo spustenie testov nad skupinou súborov (projektov).

Nakoľko študenti väčšinou odovzdávajú projekty vo forme archívu, je potrebné aby framework vedel pracovať aj s archívom a poskytoval možnosť rozbaliť archív (či skupinu archívov). Okrem toho by mal systém poskytovať aj základné funkcie pre správu súborov ako je vytvorenie nového súboru či adresára alebo zmazanie či úprava súboru.

Ďalšie požiadavky na výsledný framework sa týkajú spôsobu zobrazovania informácii. U každého študentského projektu by mala byť možnosť zobrazovať aspoň základné informácie, medzi ktoré patrí:

- status projektu, ktorý hovorí o tom, či už bol projekt testovaný alebo nie,
- dátum a čas posledne vykonaného testu,
- typ projektu, ktorý definuje, či je projekt skupinový alebo samostatný (prípadne aj zoznam študentov v jednej skupine),
- v prípade plagiátorského projektu aj informácia o tejto skutočnosti spolu s identifikátorom plagiátorskej skupiny.

Okrem týchto základných informácii by u každého projektu mala byť možnosť zobraziť si zoznam absolvovaných testov spolu s farebným vyznačením podľa ich výsledku (prípadne podľa spôsobu zlyhania). Čo sa týka vytvárania poznámok k súborom projektu, framework by mal poskytnúť možnosť zobrazenia zoznamu všetkých poznámok k danému súboru. Následne pri otvorení, resp. zobrazení obsahu súboru s poznámkami, by mala byť možnosť farebného zvýraznenia všetkých riadkov, ktoré obsahujú nejakú poznámku.

Zobrazovanie informácii by malo byť konfigurovateľné, aby si užívateľ mohol sám zvoliť ktoré informácie budú zobrazené a ako. Pri každej informácii je vhodné ponechať aj možnosť voľby jej reprezentácie (slovo, skratka, číslo, špeciálny znak a podobne).

Id	**Podkategória**	**Požiadavka**
1	ovládanie	ovládanie pomocou klávesnice
2	prechádzanie dát	pohľad na obsah do adresárovej štruktúry
3	prechádzanie dát	pohľad na obsah súboru – rýchly náhľad
4	prechádzanie dát	pohľad na obsah súboru – zobrazenie celého súboru
5	filtrovanie súborov	filtrovanie podľa cesty
6	filtrovanie súborov	filtrovanie podľa obsahu
7	filtrovanie súborov	filtrovanie podľa značky
8	práca s filtrovanými súbormi	prechádzanie medzi filtrovanými súbormi
9	práca s filtrovanými súbormi	pridať poznámku skupine
10	práca s filtrovanými súbormi	pridať značku skupine
11	práca s filtrovanými súbormi	spustiť testy nad skupinou súborov
12	správa súborov	vytvorenie súboru
13	správa súborov	zmazanie súboru
14	správa súborov	úprava súboru
15	práca s archívom	rozbaliť archív
16	značkovanie súborov	rozlíšenie typu značky – podľa počtu parametrov
17	značkovanie súborov	automatické značkovanie z výsledkov testov
18	značkovanie súborov	manuálne značkovanie
19	správa značiek	pridať značku k súboru
20	správa značiek	odstrániť značku zo súboru
21	správa značiek	zmeniť hodnotu parametra existujúcej značky
22	zobrazenie informácii	zobraziť náhľad základných informácii o projekte
23	zobrazenie informácii	farebné vyznačenie testu – podľa spôsobu zlyhania
24	zobrazenie informácii	farebné vyznačenie riadku s poznámkou v súbore
25	zobrazenie informácii	zobraziť zoznam všetkých poznámok k súboru

Tabuľka 4.1: Požiadavky na užívateľské ovládanie a prechádzanie dát

Požiadavky na správu testov

Framework by mal v rámci požiadaviek na správu testov poskytovať možnosť tvorby nového testu, mazania testov, úpravy testov a spustenia testov (viď tabuľka 4.2). Pri tvorbe nového testu treba brať do úvahy rôzne typy testov ako je napríklad *smoke test* (ktorý testuje, či projekt ide vôbec preložiť a spustiť) alebo *sanity test* (ktorý testuje, či projekt niečo robí a je implementovaná nejaká základná funkcionalita bez ohľadu na to či toto chovanie odpovedá zadaniu projektu). Takéto testy môžu odhaliť nefunkčnosť študentského projektu už na začiatku testovania a priradiť tak k projektu príslušnú značku, napríklad, že sa projekt nedá ani spustiť a teda nemá zmysel na ňom spúšťať ďalšie testy.

Niektoré projekty môžu mať časť zadania voliteľnú a implementácia tejto časti tak nie je povinná. V takýchto prípadoch treba pri tvorbe testov kontrolovať aj niektoré závislosti, či už na úspešnom alebo neúspešnom vykonaní iného testu alebo závislosť na existencii implementácie tohto rozšírenia.

Pri modifikácii vytvorených testov je vyžadované uchovávať históriu testov. Každý test teda musí mať svoju verziu a pri každej úprave sa vytvorí nová verzia daného testu. Následne bude možné spúšťať rôzne verzie jednotlivých testov (alebo celé sady testov).

Framework by mal poskytnúť možnosť selekcie, teda filtrovania testov podľa značky (napr. pre konkrétnu verziu, interval hodnotenia, výsledok testu, a podobne). Výslednú sadu vybraných testov by malo byť možné následne hromadne spravovať (mazať, modifikovať, spúšťať).

Id	Podkategória	Požiadavka
26	správa jedného testu	tvorba nového testu
27	správa jedného testu	mazanie testu
28	správa jedného testu	modifikácia testu a verzovanie
29	hromadná správa	filtrovanie testov podľa značky

Tabuľka 4.2: Požiadavky na správu testov

Požiadavky na typy testov a spôsob ich vykonávania

Táto kategória požiadaviek zahŕňa spôsoby a druhy testov či kontrol, ktoré treba zohľadniť pri hodnotení študentských projektov (viď tabuľka 4.3). Niektoré metódy na overovanie systému sa dajú automatizovať. Framework by mal teda umožniť vykonávanie automatického testovania, ktoré pozostáva zo statickej alebo dynamickej analýzy. Ako bolo spomenuté v kapitole 2, dynamická analýza spočíva v spustení potenciálne škodlivého zdrojového kódu. Preto je potrebné, aby tieto testy prebiehali v izolovanom prostredí.

S tým prichádza ďalšia požiadavka na framework a tou je možnosť izolácie vykonávania testov, spolu nastavením izolovaného prostredia (napr. vytvorenie sieťovej infraštruktúry).

Nie všetky projekty sú zamerané na tvorbu programu a zdrojových kódov, ktoré sa dajú hodnotiť automatickými testami. Treba vykonávať aj ručnú revíziu kódu (code review) a ďalších odovzdaných súborov. Mnoho študentských projektov zahŕňa aj rôzne textové súbory, dokumentácie formátu .pdf, obrázky, či ďalšie iné formáty súborov. Pre textové súbory (programové zdrojové kódy, .txt formát, .md formát a pod.) by mal framework poskytnúť interný prehliadač súborov. V prípade programových zdrojových kódov by mal interný prehliadač podporovať aj zvýraznenie syntaxe (syntax highlighter) pre základné

programovacie jazyky používané v typických študentských projektoch. Pre súbory s iným ako textovým formátom je možné využiť externý prehliadač definovaný užívateľom.

Interný editor by mal poskytnúť možnosť vytvárania poznámok do externého reportu. Pri vytváraní poznámok je vhodné rozlišovať medzi špecifickými poznámkami a typickými poznámkami. Užívateľ by mal mať možnosť vytvoriť si poznámku (napríklad k častej chybe, ktorá sa opakuje pravidelne u viacerých študentov) a uložiť túto poznámku ako typickú. Neskôr pri zadávaní ďalšej poznámky bude mať možnosť vybrať jednu z uložených typických poznámok alebo napísať novú, špecifickú poznámku.

Ako už bolo spomínané, jednou z požiadaviek je, aby sa testy farebne líšili podľa toho ako dopadol ich výsledok. Zlyhanie testu ale môže mať viacero dôvodov a niekedy je potrebné tieto prípady rozlíšiť. Z toho dôvodu vznikla ďalšia požiadavka na voliteľnú možnosť definovať ako rozpoznávať typy zlyhania testov. Príkladom môže byť zlyhanie testu na tzv. *assert error*, kedy test zlyhá na tom, že sa testuje niečo, čo daný študentský projekt vykonáva nesprávne. Oproti tomu môže byť zlyhanie testu na tzv. *runtime error*, kedy ide o chybu frameworku, teda projekt vyžaduje niečo, čo v testovacom prostredí nie je k dispozícii (napríklad typicky niektoré knižnice, ktoré študent využíval pri riešení projektu).

Id	**Podkategória**	**Požiadavka**
30	automatické testy	statická analýza
31	automatické testy	dynamická analýza
32	ručné code review	interný prehliadač súborov
33	ručné code review	poznámkovanie do externého reportu
34	ručné code review	rozlíšenie typických a špecifických poznámok
35	ručné code review	syntax highlighter v prehliadači súborov
36	spôsob izolácie vykonávania testov	možnosť nastavenia izolovaného prostredia
37	rozlíšenie typu zlyhania testu	voliteľná možnosť definovať ako rozpoznať typy zlyhania testov

Tabuľka 4.3: Požiadavky na typy testov a spôsob ich vykonávania

Požiadavky na reportovanie

Požiadavky na spôsob tvorby záverečnej správy hodnotenia projektu (report) či inej spätnej väzby, sú zobrazené v tabuľke 4.4. Po tom ako vyučujúci dôkladne otestuje študentský projekt, je potrebné vytvoriť záverečné hodnotenie s prípadnými poznámkami a celkovým bodovým skóre. Celkové ohodnotenie projektu sa skladá z viacerých častí. Hlavnú časť tvorí automatický report generovaný z automatických testov. Následne by mal mať vyučujúci možnosť ručne upravovať tento report. Ručná úprava poskytuje pridanie vlastného hodnotenia, ktoré nesúvisí s automatickým hodnotením a ide teda o pridávanie nezávislej poznámky či bodového ohodnotenia (napríklad za chýbajúcu dokumentáciu). Oproti tomu, druhý spôsob ručnej úpravy reportu spočíva v manuálnej úprave automatického ohodnotenia, kde je možné pridať poznámku k automatickým testom.

Z výsledkov testov a hodnotení sa následne vytvoria štatistiky a histogramy. Je požadovaná tvorba výsledného grafu bodového hodnotenia, ktoré zobrazuje rozloženie, koľko študentov spadá do danej bodovej kategórie. Generovanie histogramov je zamerané na testy, ktoré by mali zobrazovať, ktorý test bol ako hodnotený. Na základe týchto histogramov je možné niekedy odhaliť podozrivé testy (napríklad ak nikto nemal z daného testu ani jeden bod, je možné, že je v ňom chyba).

Id	Podkategória	Požiadavka
38	automatický report	generovanie reportu z automatických testov
39	ručná úprava reportu	pridanie vlastného hodnotenia
40	ručná úprava reportu	úprava automatického hodnotenia
41	štatistiky a histogramy	graf bodového hodnotenia
42	štatistiky a histogramy	histogramy

Tabuľka 4.4: Požiadavky na reportovanie

4.1.3 Nefunkčné požiadavky

Jednou z hlavných nefunkčných požiadaviek, na ktoré je kladený silný dôraz je intuitívnosť ovládania systému. Cieľom celej práce je čo najviac zjednodušiť a zefektívniť prácu vyučujúcim pri opravovaní a hodnotení študentských projektov. Preto je potrebné čo najviac ponechať zaužívané ovládacie prvky, aby sa vyučujúci nemuseli dlho zoznamovať s novým prostredím a rýchlo si naň zvykli.

S tým súvisí ďalšia požiadavka a tou je použiteľnosť, teda jednoduchosť používania systému. Systém by mal byť schopný poskytnúť používateľom nápovedu na ovládanie, prípadný zoznam klávesových skratiek a podporovanej funkcionality. V rámci zaistenia použiteľnosti sa tiež predpokladá dokumentácia k výslednému systému.

Ďalšou veľmi dôležitou nefunkčnou požiadavkou je rozšíriteľnosť. Táto požiadavka sa týka predovšetkým dobrej čitateľnosti a zrozumiteľnosti zdrojového kódu, vhodného využitia komentárov či používania vhodných a aktuálne podporovaných knižníc. Systém by mal byť udržateľný a mal by poskytnúť možnosť rozšírenia o ďalšie funkcionality.

Výsledný systém by mal byť spustiteľný na linuxových strojoch, nemal by používať žiadne zastaralé knižnice, mal by byť bezpečný a mal by poskytovať užívateľom rýchlu odozvu.

4.2 Výber technológii

Táto sekcia popisuje zvolené technológie pre implementáciu frameworku spolu s odôvodnením ich výberu vzhľadom na analyzované požiadavky.

Pre implementáciu som sa rozhodla použiť jazyk Python a to najmä pre jednoduchú rozšíriteľnosť, čitateľnosť, výbornú podporu a tiež kvôli lepšej flexibilite a rýchlosti vývoja oproti jazykom vyžadujúcich kompiláciu.

4.2.1 Knižnica ncurses

Pre vytvorenie užívateľského rozhrania systému som zvažovala viacero nástrojov, medzi ktoré patrili FINAL CUT, termbox, newt a ncurses. Hlavnou požiadavkou na výsledný framework bolo poskytovanie textového užívateľského rozhrania, aby bolo možné testovať projekty aj na vzdialených počítačoch pomocou textového *ssh* spojenia.

FINAL CUT je moderný a populárny nástroj, ktorý poskytuje širokú podporu pri tvorbe textového užívateľského rozhrania, avšak je dostupný len pre jazyk C++. Termbox je jednoduchý nástroj pre tvorbu textových rozhraní, no od roku 2020 už nie je udržiavaný, preto som ho nepovažovala za vhodný. Newt je knižnica založená prevažne na grafických ovládacích prvkoch a skôr sa využíva pre tvorbu grafických rozhraní než textových. Okrem toho má táto knižnica niekoľko nedostatkov, mnohé nastavenia a zmeny vykresľovaného prostre-

dia sú ťažko realizovateľné až nemožné. Knižnica ncurses je dostupná aj pre jazyk Python, je veľmi rozšírená, stále udržovaná a poskytuje užívateľské rozhranie, kde je možné opakovane nastaviť vykreslené prostredie. Preto som zvolila knižnicu ncurses ako najvhodnejšie riešenie.

4.2.2 Knižnica Pygments

Pre zaistenie požiadavky na podporu zvýraznenia syntaxe textových súborov, som zvažovala rôzne podporné knižnice ako je napríklad Syntect alebo Pygments. Syntect je knižnica, ktorá je pôvodne implementovaná pre jazyk Rust, avšak existuje aj podpora pre Python. Obe knižnice sú stále udržiavané a podporujú zvýraznenie syntaxe pre mnoho rôznych jazykov. Hlavným rozhodovacím faktorom medzi nimi bola závislosť na spomínanej vybranej technológii ncurses. Nakoľko ncurses používa na vykresľovanie textu vlastné definované farby, bolo potrebné vybrať takú knižnicu, ktorá by sa dala tomuto prispôsobiť. Z toho dôvodu som zvolila knižnicu Pygments.

Pygments podporuje formátovanie textu (s podporou viac ako 530 jazykov) a následné sfarbenie podľa vybraného štýlu. Okrem preddefinovaných formátovaní a štýlov poskytuje tiež možnosť implementovať a využiť vlastné spracovanie textu.

4.2.3 Šablónovací systém Jinja

Finálny report s celkovým hodnotením projektu môže pre každý projekt vyzerať inak. Bolo by teda veľmi obmedzujúce poskytnúť len jeden formát výslednej správy z testovania projektu. Preto som sa rozhodla pri tvorbe reportu využiť šablónovací systém, aby si užívateľ mohol sám zvoliť, ako bude vyzerať výsledné hodnotenie. Pre tento účel som zvolila systém Jinja pre jazyk Python, ktorý je prehľadný, dobre rozšíriteľný, veľmi jednoducho sa používa a je rýchly. Alternatívami pre šablónovací systém sú napríklad Mako alebo Chameleon, ktoré sú oba tak tiež pre jazyk Python avšak Jinja je oproti nim populárnejšia a viac udržovaná.

4.2.4 Linuxové kontajnery

Ďalšiu veľmi dôležitou technológiou sú linuxové kontajnery. Ako už bolo spomenuté v kapitole 3, využitie kontajnerov prináša počas testovania softvéru hromadu výhod. Hlavným dôvodom použitia tejto technológie v rámci frameworku pre testovanie študentských projektov je práve možnosť izolácie prostredia, v ktorom sa vykonajú samotné testy. Vďaka jednoduchému vytváraniu a odstráneniu kontajnerov sa minimalizuje réžia spojená s prípravou a čistením prostredia pred a po vykonaní testu.

Ďalším dôvodom pre využitie linuxových kontajnerov je, že pri vytváraní objektov image sa rovno vytvorí aj test fixture pre samotný jeden test alebo pre celú testovaciu sadu. Image je možné potom použiť na testovanie projektov na ľubovoľnom stroji, pričom je zaručená stálosť testov.

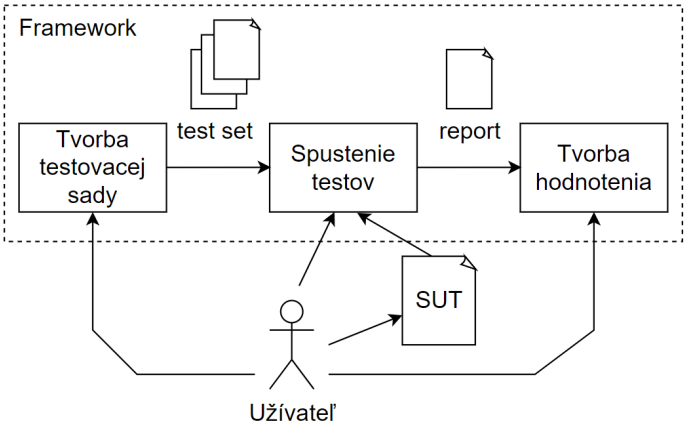
Na druhú stranu linuxové kontajnery prinášajú aj isté nevýhody, ako napríklad, že vytvorenie kontajneru trvá určitý čas kým sa v ňom nastaví celé prostredie s potrebnými závislosťami. Oproti virtuálnym strojom, ktoré by tak isto poskytli požadovanú izoláciu prostredia, však vytvorenie kontajneru vyžaduje podstatne menej času.

4.3 Návrh systému

Táto sekcia sa zaoberá návrhom riešenia systému tak, aby pokrýval analyzované požiadavky zo sekcie 4.1. Pre lepšie pochopenie zmyslu celého systému a jeho využitia je v prvej časti najskôr priblížený proces testovania študentských projektov z pohľadu vyučujúcich. Ďalej je popísaný návrh systému, ktorý by mal poskytnúť textové užívateľské rozhranie (TUI – *text-based user interface*) pre podporu tohto procesu.

4.3.1 Proces testovania študentských projektov

Proces testovania študentských projektov pozostáva z viacerých akcii, ktoré vyučujúci vykonáva. Na obrázku 4.1 je zobrazená interakcia užívateľa (vyučujúceho) so systémom počas hodnotenia študentského projektu. Systém poskytuje tri základné bloky procesov: tvorba testovacej sady, spustenie testov a tvorba hodnotenia.



Obr. 4.1: Interakcia užívateľa so systémom počas testovania študentských projektov

Tvorba testovacej sady pozostáva z vytvorenia, mazania, modifikácie testov, prípadne filtrovania a selekcie testov, pričom výsledkom tejto aktivity je testovacia sada (*test set*), ktorá sa bude následne spúšťať nad SUT (študentským projektom).

Pred spustením testovacej sady je niekedy potrebné najskôr pripraviť SUT na testovanie, napríklad rozbalením archívu odovzdaného projektu. Užívateľ teda musí mať možnosť siahnuť na SUT. Následne je možné spustiť testovaciu sadu nad SUT, pozastaviť testovaciu sadu, prípadne môže vyučujúci upravovať testovaný systém, ak v ňom odhalí nejakú chybu, ktorú chce napraviť a následne nad ním znovu spustiť testy.

Výstupom fáze spúšťania testov je report, ktorý obsahuje výsledky automatických testov. Vyučujúci potom vytvára hodnotenie projektu. Hlavná časť hodnotenia je tvorená generovaným reportom z automatických testov, ku ktorým je možné pridávať ďalšie poznámky. Následne hodnotenie obsahuje poznámky z ručnej revízie (code review) a prípadné ďalšie nezávislé poznámky k projektu.

Detailnejší proces testovania je zobrazený pomocou diagramu aktivít v prílohe A. Jednotlivé aktivity diagramu reprezentujú funkcie poskytované systémom pre užívateľa.

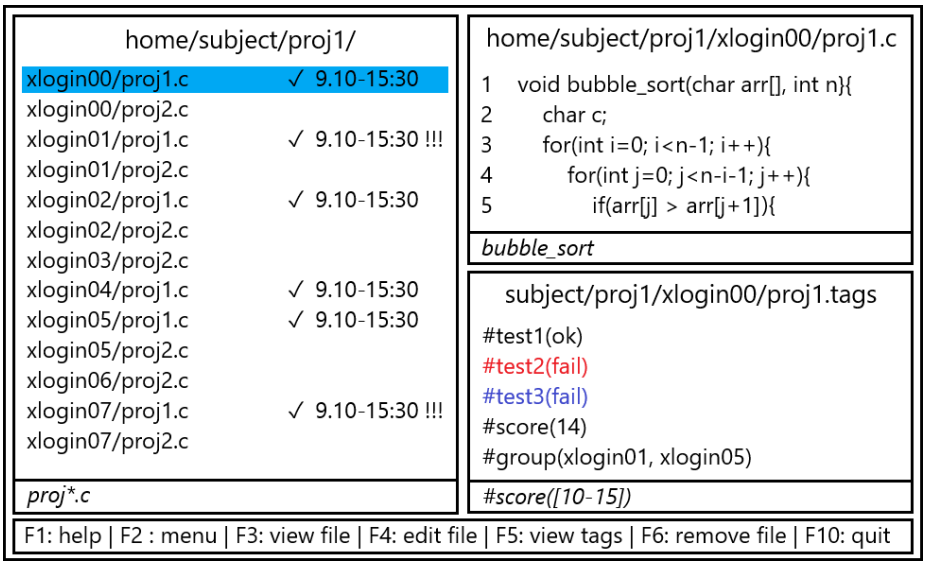
4.3.2 Zobrazenie dát a rýchly náhľad do súboru

Čo sa týka užívateľského ovládania a prechádzania dát je systém navrhnutý tak, aby poskytoval niekoľko základných užívateľských pohľadov. Na obrázku 4.2 je načrtnutý návrh systému, ktorý poskytuje textové užívateľské rozhranie rozdelené na viacero okien.

Na ľavej strane sa nachádza okno pre prechádzanie obsahu adresárovej štruktúry (v tomto prípade sa prechádza adresár *home/subject/proj1/*).

Na pravej strane je okno rozdelené na ďalšie dve časti. V hornej časti sa zobrazuje rýchly náhľad do súboru, ktorý má aktuálne užívateľ zvolený (v tomto prípade je zvolený modro zvýraznený súbor *xlogin00/proj1.c*). V spodnej časti sa nachádza zoznam značiek k tomuto súboru. Niektoré značky môžu byť farebne vyznačené, napríklad podľa spôsobu zlyhania testu (červená a modrá značka s parametrom `fail`).

V spodnej časti sa nachádza ešte jedno malé okno, ktoré slúži len ako nápoveda na ovládanie základných funkcii pomocou klávesových skratiek. Obsah tohto okna sa mení podľa toho, v akom režime/okne sa zrovna užívateľ pohybuje (napr. správa poznámok, code review, prechádzanie adresára, správa značiek a podobne).



Obr. 4.2: Návrh TUI systému - prehľadávanie adresára a náhľad do súboru

4.3.3 Filtrovanie súborov

Pod každým oknom je priestor na rôzne druhy filtrovania súborov. V okne s adresárovou štruktúrou je zadaný filter *proj*.c*, ktorý filtruje súbory podľa cesty, teda zobrazuje len súbory, ktorých cesta/názov sa zhoduje s daným výrazom. V okne s obsahom súboru je zadaný filter *bubble_sort*, ktorý po filtrovaní zobrazí len tie súbory, ktoré obsahujú zadaný reťazec. V okne so značkami je zadaný filter podľa značky, ktorý v tomto prípade filtruje súbory, ktoré majú pridanú značku s názvom *score*, s hodnotou parametru v intervale od 10 do 15. V adresárovej štruktúre sa potom zobrazia len súbory, ktoré vyhovujú všetkým zadaným filtrom. Užívateľ následne môže medzi týmito súbormi prechádzať.

4.3.4 Zobrazenie informácii o projekte

Ako je možné vidieť na obrázku 4.2, u každého súboru pri prechádzaní adresárovej štruktúry sú na konci riadkov zobrazené nejaké informácie. Táto časť je konfigurovateľná a v tomto príklade sú reprezentované a zobrazené nasledujúce informácie. Zľava je najskôr použitý symbol „check“, ktorý informuje o vykonanom teste nad daným súborom. Môžeme si všimnúť, že tento symbol je len pri súboroch *proj1.c*, teda ostatné súbory projektu ešte neboli testované. Ďalej za týmto symbolom nasleduje dátum a čas posledne vykonaného testu a na konci úplne vpravo je u niektorých súborov symbol troch výkričníkov, čo hovorí o podozrení na plagiátorské projekty. Ide len o rýchle zobrazenie informácii na prvý pohľad, detailné informácie sa ukladajú v špeciálnych súboroch, v podobe spomínaných značiek.

4.3.5 Užívateľské ovládanie

Systém je navrhnutý tak aby ovládanie okien bolo realizované pomocou klávesnice. Šípky slúžia na prechádzanie medzi súbormi alebo riadkami v súbore. Tabulátor slúži na prepínanie medzi aktívnymi oknami a ďalšie klávesové skratky budú využité pre ostatnú funkcionalitu, ktorá môže byť odlišná v závislosti od toho, v ktorom okne sa užívateľ zrovna pohybuje.

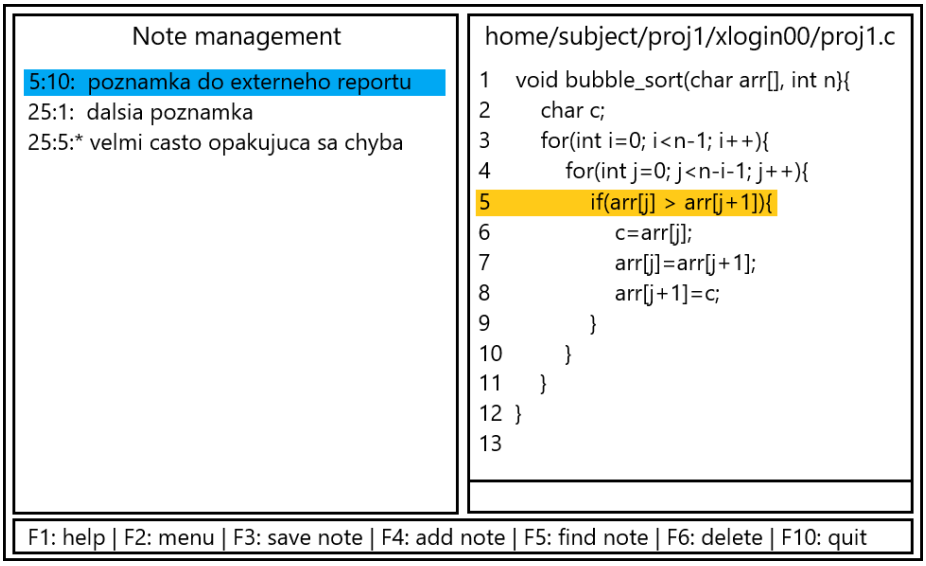
V každom režime budú klávesové skratky využité primárne pre najčastejšie vykonávané akcie špecifické pre daný režim. Napríklad v režime prechádzania adresárovej štruktúry to budú funkcie týkajúce sa správy súborov, správy testov či filtrovanie. V režime správy značiek pôjde o funkcie pre vytvorenie novej značky, odstránenie značky či zmena hodnoty parametra značky. V režime prezerania obsahu súboru budú najdôležitejšie funkcie pre úpravu súboru a správu poznámok.

Všetky ostatné funkcie, pre ktoré nebude dostupná klávesová skratka, budú poskytnuté užívateľovi cez menu v hlavnom režime (t.j. režim prehľadávania adresárovej štruktúry). V menu bude možné prechádzať šípkami medzi zoznamom funkcii a spúšťať ich.

4.3.6 Code review a poznámky

Ďalší návrh systému zahŕňa ručné code review, kde užívateľ potrebuje prechádzať celý obsah súboru. V tomto režime sa využije celá pravá strana na zobrazenie obsahu súboru, kde bude možné pomocou klávesových skratiek vypnúť alebo zapnúť zobrazenie číslovania riadkov alebo farebné vyznačenie riadku s poznámkou. Obrázok 4.3 ukazuje príklad zobrazenia súboru s poznámkou na riadku 5 (viď žltý zvýraznený riadok). Na ľavej strane tejto ukážky je zoznam všetkých poznámok k danému súboru, medzi ktorými je možné prechádzať pomocou šípok.

Každá poznámka v súbore je identifikovaná pomocou čísla riadku a čísla stĺpca vo forme `riadok:stĺpec`, za ktorým nasleduje samotný obsah poznámky vo forme textu. Pri identifikátore môže mať poznámka špeciálny symbol hviezdičku, ktorá značí, že je táto poznámka uložená ako typická a často sa opakuje (viď obrázok 4.3, posledná poznámka). Týmto spôsobom sa rozlišuje špecifická poznámka od typickej. Každá špecifická poznámka sa môže kedykoľvek uložiť ako typická. V prípade, že užívateľ chce pridať novú poznámku a vybrať tak jednu z typických poznámok, môže na to využiť menu, kde si vyberie zo zoznamu uložených poznámok, ktorú chce pridať.

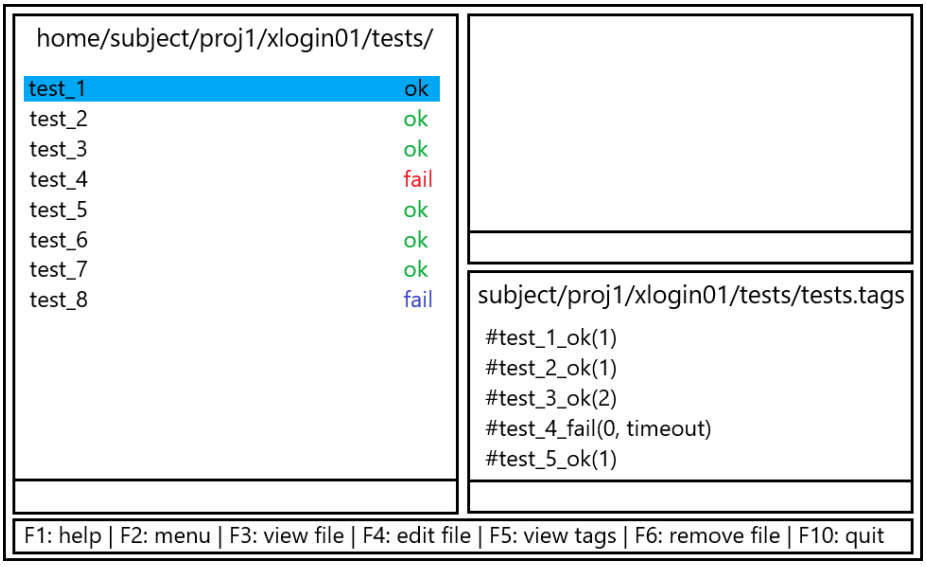


Obr. 4.3: Návrh TUI systému - code review a prehľad poznámok

4.3.7 Zobrazovanie výsledkov z automatických testov

Výsledky z automatických testov sa ukladajú do samostatného adresára kde sú ďalej štruktúrované podľa jednotlivých testov (viď obrázok 4.4, ľavá strana). V tomto adresári si každý test počas spustenia ukladá svoje výstupy, záznamy a ďalšie súbory týkajúce sa výsledkov testu pre dané jedno študentské riešenie projektu. Okrem týchto výsledkov sa počas testovania generujú automatické značky s výsledkom testu (prešiel/neprešiel) a s bodovým ohodnotením za daný test (viď obrázok 4.4, pravá strana).

Na základe týchto značiek sa potom zobrazujú informácie o výsledkoch jednotlivých testov, ktoré je možné zobrazovať a prechádzať medzi nimi v samostatnom okne (viď obrázok 4.4, ľavá strana).



Obr. 4.4: Návrh TUI systému - prehľad výsledkov automatických testov

4.3.8 Tvorba výsledného hodnotenia

Výsledný report z hodnotenia projektu sa generuje zo šablóny, ktorú si užívateľ môže prispôsobiť. V tejto šablóne je možné zahrnúť informácie z viacerých častí hodnotenia projektu, pričom každá sa ukladá trochu iným spôsobom. Jednotlivé časti hodnotenia sa ukladajú v adresári *reports* (viď obrázok 4.5, ľavá strana) v samostatných súboroch:

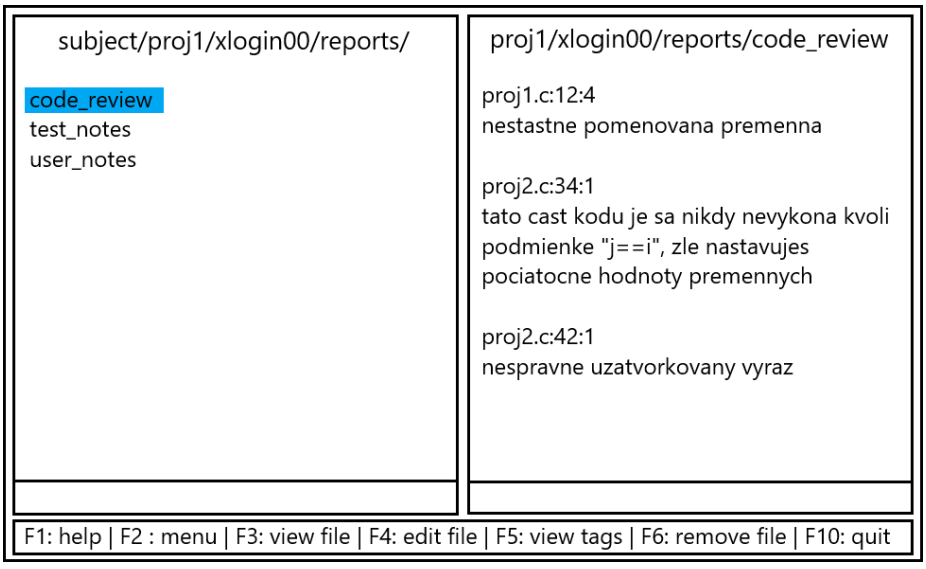
- `code_review`: uchováva poznámky z code review,
- `test_notes`: uchováva poznámky k automatickým testom,
- `user_notes`: uchováva ďalšie nezávislé samostatné poznámky k hodnoteniu.

Okrem poznámok sú k dispozícii aj výsledky z automatických testov, ktoré sa generujú po spustení testov a ukladajú sa pomocou značiek. Tieto značky v sebe nesú informácie o vykonanom teste a jeho výsledku, teda typicky: názov testu, výsledok testu (prešiel/neprešiel) a hodnotenie testu (počet bodov).

Poznámky k automatickým testom, ktoré užívateľ zadáva manuálne sa ukladajú do samostatného súboru. Pridať poznámku k automatickým testom je možné kdekoľvek v adresári projektu (cez menu), prípadne priamo v súbore s týmito poznámkami.

Ďalšia časť hodnotenia zahŕňa výsledky z code review. Code review vykonáva užívateľ prechádzaním jednotlivých súborov projektu v internom editore, pričom vykonáva revíziu obsahu týchto súborov a pridáva k nim poznámky. Z týchto poznámok sa neskôr vygeneruje celkový report v jednom súbore, ktorý obsahuje všetky poznámky z code review (viď obrázok 4.5, pravá strana). Poznámky sú v tomto súbore oddelené prázdnymi riadkami a každá je identifikovaná názvom súboru, ku ktorému bola pridaná, číslom riadku a číslom stĺpca.

Pre ďalšie hodnotenie projektu mimo automatických testov a mimo code review je možné pridať samostatnú nezávislú poznámku a to kdekoľvek v adresári projektu daného študenta (cez menu). Tieto poznámky sa ukladajú do zvláštneho súboru „nikam nepatriacich“ poznámok. Môže obsahovať poznámky ako napríklad: „chýbajúca dokumentácia“, „nesprávne pomenovaný odovzdaný archív“ alebo „pekne okomentovaný a efektívny kód“.



Obr. 4.5: Návrh TUI systému - prehľad výsledných reportov a code review

Kapitola 5

Implementácia systému

Táto kapitola sa zaoberá popisom implementácie frameworku pre testovanie študentských projektov, ktorý bol vytvorený na základe analyzovaných požiadaviek z kapitoly 4.

Systém je implementovaný v jazyku Python s využitím nástrojov vybraných v sekcii 4.2. Je spustiteľný na bežných linuxových strojoch (testovaný na distribúciách Fedora 35 a Raspbian 10). Vďaka tomu, že poskytuje textové užívateľské rozhranie vytvorené podľa definovaného návrhu, je možné tento systém využívať aj na vzdialených strojoch.

5.1 Ovládanie systému a konfigurácia

Systém je ovládaný len pomocou klávesnice a väčšina funkcií systému je mapovaná na klávesové skratky (ostatné sú dostupné cez menu). Motiváciou k vytvoreniu mapovania funkcií na klávesy, bolo poskytnúť užívateľom možnosť konfigurovať si klávesy podľa vlastnej potreby. Nakoľko je celé užívateľské prostredie (vrátane ovládania) postavené na knižnici ncurses, ktorá má vlastnú reprezentáciu kláves, bolo potrebné tieto klávesy sprístupniť užívateľom v zrozumiteľnej podobe. Z toho dôvodu bolo vytvorené mapovanie, ktoré je implementované vo viacerých súboroch:

- *control.yaml*: Definuje ovládanie systémových funkcií klávesami. Funkcie aj klávesy sú reprezentované vo forme znakového reťazca.
- *controls/functions.py*: Obsahuje identifikátory všetkých implementovaných funkcii, ktoré sú reprezentované ako číselné konštanty. Okrem toho mapuje systémové funkcie zo súboru *control.yaml* na číselnú reprezentáciu funkcii.
- *controls/control.py*: Implementuje objekt, ktorý uchováva mapovanie číselnej reprezentácie funkcii na klávesy zo súboru *control.yaml*. Využíva sa na zistenie odpovedajúcej funkcie pre stlačenú klávesu, pričom rovno mapuje reprezentáciu kláves knižnice ncurses na internú textovú reprezentáciu kláves.

Prispôsobovanie ovládania funkcii klávesami možno realizovať v súbore *control.yaml*, kde je možné využívať len implementované reprezentácie kláves (zoznam týchto kláves je vypísaný v komentároch na začiatku súboru *control.yaml*). Okrem toho je možné jednoducho rozšíriť implementáciu o ďalšie klávesy.

```
# control.yaml
delete_file: 'F8'

# controls/functions.py
DELETE_FILE = 103
{'delete_file': DELETE_FILE}

# controls/control.py
{'F8': DELETE_FILE}
if key == curses.KEY_F8: return 'F8'
```

Výpis 5.1: Príklad mapovania funkcie na klávesy

Vo výpise 5.1 je farebne vyznačený príklad mapovania funkcie pre zmazanie súboru na klávesu F8. Sivé riadky vyznačujú, v akom súbore sú dané mapovania definované. Samotná funkcia je následne v systéme implementovaná pod číselnou konštantou 103.

Okrem zmeny kláves je možné systému nastaviť jeho počiatočný stav, do ktorého sa nakonfiguruje. Na to slúži súbor *config.yaml*, v ktorom sú definované nasledujúce nastavenia (v zátvorke je uvedený počiatočný stav, ktorý môže užívateľ zmeniť):

- režim rýchleho náhľadu do súboru (zapnuté),
- zobrazenie značiek (zapnuté),
- zobrazovanie záznamov (zapnuté),
- zobrazovanie informácii o projekte (zapnuté),
- zvýraznenie riadkov s poznámkou v editore (zapnuté),
- zobrazenie číslovania riadkov v editore (zapnuté),
- veľkosť horného a dolného okraju okna (1),
- veľkosť tabulátoru (4).

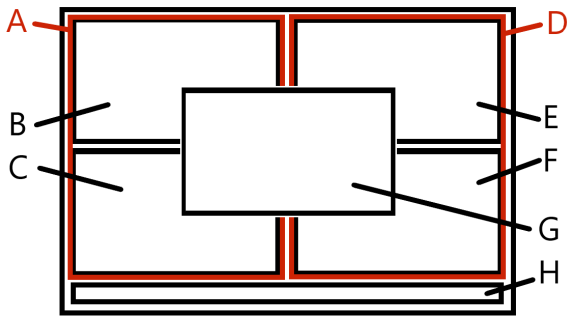
Nastavenia z tohto súboru sa načítajú na začiatku spustenia programu (rovnako tak aj spomínané mapovanie kláves na funkcie). Systém musí vedieť v každom okamžiku v akom stave sa nachádza a akými klávesami sa ovláda. Preto bol vytvorený objekt `Environment`, ktorý slúži na uchovávanie potrebných informácii o stave a prostredí systému. Inštancia tohto objektu sa vytvorí hneď na začiatku spustenia programu a medzi jednotlivými funkciami sa predáva ako parameter `env`.

5.2 Vykresľovanie obrazoviek

Knižnica ncurses už pri inicializácii vytvorí hlavnú obrazovku (objekt *window*[1]) o veľkosti terminálu, vrámci ktorej možno zobrazovať dáta. Táto obrazovka slúži aj na interakciu užívateľa so systémom, teda sa pomocou nej získavajú stlačené klávesy.

[1] https://docs.python.org/3/library/curses.html#window-objects

Aby bolo možné zobrazovať dáta v rôznych častiach obrazovky, bolo potrebné vytvoriť viac takýchto objektov (viď obrázok 5.1). Každá samostatne zobrazovaná časť tak má vlastnú obrazovku na vykresľovanie dát. Rozloha a umiestnenie jednotlivých pod-obrazoviek v rámci celej hlavnej obrazovky je znázornené na obrázku 5.1. V spodnej časti sa nachádza obrazovka H, slúžiaca pre zobrazovanie stručnej nápovedy na využívanie klávesových skratiek. Obrazovky A a D majú rovnakú veľkosť a sú ďalej ešte rozdelené na dve menšie obrazovky (B,C a E,F). V strede sa nachádza jedna menšia centrálna obrazovka G, ktorá sa využíva na zobrazenie komplexnej užívateľskej nápovedy, zobrazenie menu, či zadávanie vstupu užívateľom.



Obr. 5.1: Znázornenie rozmiestnenia obrazoviek systému

Vykresľovanie obsahu obrazoviek bolo pôvodne riešené tak, že sa po každom stlačení klávesy prepísali všetky obrazovky. Pri častom a rýchlom stláčaní kláves tak bolo vidieť preblikávanie obrazoviek, čo pôsobilo rušivo. Tento jav sa čiastočne eliminoval tým, že sa po stlačení klávesy prekreslí len nevyhnutná časť obrazovky, ktorej sa zmenil obsah.

5.2.1 Problematika kurzoru

Obrazovky knižnice ncures slúžia na zobrazovanie dát, no treba vedieť v každý okamžik čo presne sa má vykresľovať a ako. Preto bolo potrebné vytvoriť ďalší objekt `Window`, pre správnu orientáciu vrámci obrazovky a správne vykresľovanie dát. Pre každú zobrazovanú časť teda prislúcha jedna obrazovka a jedna inštancia objektu `Window`.

`Window` si udržiava vždy aktuálnu pozíciu kurzoru (riadok a stĺpec), podľa toho ako sa užívateľ pohybuje v danom okne a pamätá si tiež posuv obrazovky (pre prípad, ak sa dostaneme mimo maximálny rozsah obrazovky, celý obsah okna sa posunie o tento posuv).

Pre správne vykresľovanie a posuv kurzoru bolo potrebné vysporiadať sa s tabulátormi. Knižnica ncurses vykresľuje znak **\t** tak, že text zarovná vždy na štvrtú pozíciu, čo nie je úplne najšťastnejšie riešenie, najmä ak je potrebné sa v texte pohybovať kurzorom po jednom znaku. Implementovaný systém rieši tento problém tak, že pred vykresľovaním nahradí všetky tabulátory v texte vopred definovaným počtom medzier, pričom ale nemení originálny text (napríklad v súbore). Prechádzanie kurzorom cez tabulátor potom vyzerá tak, že ak deteguje prítomnosť znaku **\t**, systém vie, že sa má posunúť kurzorom o definovaný počet medzier. Výpočet správnej pozície kurzoru prebieha vrámci objektu `Window` a jeho metód, pričom berie do úvahy:

- posuv o tabulátory v danom riadku - metóda `calculate_tab_shift`,
- vertikálny posuv riadkov - metóda `vertical_shift`,
- horizontálny posuv stĺpcov - metóda `horizontal_shift`.

Kurzor si naviac pamätá poslednú pozíciu stĺpca, ktorú využíva, aby sa držal intuitívne stále v rovnakej línii pri prechádzaní obsahu súboru. Ak sa teda posunie kurzor napríklad z riadku 4, stĺpca 10 na riadok 5, ktorý je prázdny, bude nová hodnota stĺpca 0. Ak sa následne kurzor posunie naspäť na riadok 4, využije zapamätanú poslednú pozíciu stĺpca s hodnotou 10 a nová hodnota stĺpca tak bude 10, nie 0.

Ďalšou problematickou časťou pri vykresľovaní obrazoviek bolo udržanie správnej pozície kurzoru pri zmene veľkosti obrazovky. Systém je implementovaný tak, aby pri zmene veľkosti okna držal pozíciu kurzoru relatívne v strede. Podľa aktuálnej pozície kurzoru sa okno pomyselne rozdelí na dve časti (nad kurzorom a pod kurzorom) a následne sa pri zmenšovaní okna uberá zobrazovaný riadok vždy z väčšej časti. Obdobne tak pri zväčšovaní okna sa pridáva zobrazovaný riadok súboru do menšej časti.

5.2.2 Zmena pozície obrazovky

Spomínané obrazovky (objekty knižnice ncurses) sa dajú posúvať, teda je možné meniť ich pozíciu. Túto vlastnosť využíva centrálna obrazovka G (viď obrázok 5.1), ktorú si môže užívateľ posunúť napravo, naľavo alebo na stred. Aktuálnu pozíciu vykresľovanej obrazovky si táto zobrazovaná časť pamätá pomocou objektu `Window`, atribútom `position`, ktorý môže mať číselnú hodnotu: 1 (pozícia vľavo), 2 (pozícia v strede) alebo 3 (pozícia vpravo).

Túto funkcionalitu ocení užívateľ najmä v prípade, že pre účely vykonania zamýšľanej akcie s daným centrálnym oknom potrebuje informácie, ktoré toto okno čiastočne či úplne zakrýva.

5.2.3 Užívateľské režimy

Každý režim má pridelenú svoju obrazovku na vykresľovanie (*screen*) a svoje okno (*window*) na orientáciu vrámci obrazovky. Všetky obrazovky a okná sú v systéme prístupné cez hlavnú triedu `Environment`, ktorá okrem iného aj definuje poradie prepínania medzi režimami. Základné režimy, medzi ktorými môže užívateľ prepínať pomocou tabulátoru, sú:

- prechádzanie adresárovej štruktúry - obrazovka A alebo B (pri zapnutom zobrazovaní záznamov),
- prechádzanie obsahu súboru - obrazovka D alebo E (pri zapnutom zobrazovaní značiek),
- prechádzanie poznámok z code review - obrazovka A,
- prechádzanie značiek - obrazovka F,
- prezeranie záznamov - obrazovka C.

V každom režime je možné ukončiť systém pomocou klávesy F10 alebo zobraziť nápovedu pomocou klávesy F1. Zobrazenie nápovedy je realizované pomocou obrazovky G, kde sa vykresľuje pomôcka k ovládaniu daného režimu vo forme dvojice: klávesa a jej odpovedajúca funkcia. Pôvodne bol výpis nápovedy fixný, no neskôr bol prerobený tak, aby sa táto dvojica zistila automaticky podľa konfigurácie a mapovania ovládania systému (viď sekcia 5.1). Pri zmene ovládania systému teda netreba túto časť implementácie meniť. Ak sa celá nápoveda nezmestí do zobrazovanej časti, môže užívateľ prechádzať šípkami hore a dolu, aby videl aj zvyšok nápovedy.

5.3 Adresárová štruktúra

Po zapnutí systému sa užívateľ nachádza v režime prechádzania adresárovej štruktúry, kde vidí obsah jeho aktuálneho adresára, odkiaľ systém spúšťal. Ak je adresár prázdny, zobrazí sa len hláška s touto skutočnosťou. Medzi položkami v jednom adresári sa prechádza pomocou šípok hore a dole. Zmena adresára sa vykonáva šípkami doprava (vojde do vyznačeného pod-adresára) a doľava (vyjde z aktuálneho adresára do rodičovského).

Obsah adresára je reprezentovaný pomocou objektu `Directory`, ktorý uchováva zoznam pod-adresárov a súborov v aktuálnom adresári. Ak je aktuálny adresár zároveň koreňovým adresárom nejakého projektu, načíta sa konfigurácia tohto projektu a vytvorí sa inštancia objektu `Project`, ktorá sa uloží do `Directory`. Vďaka tomu je možné v každý okamžik systému zistiť, či aktuálny adresár je v nejakom pod-adresári projektu, čím sa otvárajú ďalšie funkcie systému.

Objekt `Project` okrem projektovej konfigurácie (viď 5.3.1) uchováva tiež informácie o študentských riešeniach, ktoré sú reprezentované pomocou objektu `Solution`. Zoznam študentských riešení (inštancií objektu `Solution`) je dostupný cez atribút `solutions` triedy `Project`. Študentské riešenia sú adresáre, ktoré musia byť uložené v koreňovom adresári projektu a musia zodpovedať identifikátoru riešenia projektu definovaného v konfiguračnom súbore projektu.

Objekt `Solution` vznikol až v neskoršej fáze implementácie, keď sa ukázalo, že práca so systémom je pri veľkom počte študentských riešení pomalá, čo bolo spôsobené opakovaným načítavaním súborov (dát) študentských riešení. Preto bolo potrebné niektoré súbory načítať dopredu a udržiavať ich v pamäti počas práce nad jedným projektom. Pre každé študentské riešenie sa teda načíta a ukladá zoznam značiek k riešeniu, výsledky testov, poznámky k testom a ďalšie poznámky k riešeniu.

5.3.1 Založenie nového projektu

Podmienkou pre založenie nového projektu je, že sa užívateľ musí nachádzať v nejakom adresári, ktorý ešte nie je (pod)adresárom žiadneho projektu. Následne pomocou menu (klávesa F2) zvolí možnosť pre založenie nového projektu, čím sa na mieste vytvorí konfiguračný súbor *proj_conf.yaml*, čím sa tento adresár stáva koreňovým adresárom pre nový projekt. Konfiguračný súbor obsahuje nasledujúce nastavenia, ktoré je možné prispôsobiť pre konkrétny projekt:

- `name`: názov projektu,
- `created`: dátum založenia projektu,
- `solution_id`: identifikátor riešenia projektu (typicky xlogin00),
- `max_score`: maximálny počet bodov za projekt,
- `sut_requested`: očakávaný názov súboru s riešením (ktoré sa bude testovať),
- `sut_ext_variants`: zoznam podporovaných variánt názvu súboru s riešením,
- `solution_info`: zobrazované informácie o riešení,
- `tests_info`: zobrazované informácie o výsledkoch testov.

Po založení nového projektu sa okrem konfiguračného súboru vytvorí adresár *tests* pre definíciu testov a v ňom sa vytvoria súbory:

- *scoring* pre definíciu skóre (bodového hodnotenia) za každý test,
- *sum* pre definíciu rovnice, podľa ktorej sa spočíta celkové bodové hodnotenie projektu,
- *testsuite.sh* pre definíciu testovacej stratégie.

Ďalej sa vytvorí adresár *history* pre uchovávanie histórie testov, spolu so súborom *testsuite_history.txt*, ktorý zaznamenáva udalosti (vykonané zmeny) v testovacej sade, ako napríklad vytvorenie nového testu, modifikácia testu alebo zmazanie testu (viď sekcia 5.11.3).

Ako posledný sa vytvorí adresár *reports* so súborom *report_template.j2*, pre definíciu šablóny na vytvorenie celkového reportu z hodnotenia projektu. Tento súbor obsahuje predvolenú štruktúru šablóny, ktorú môže užívateľ zmeniť s využitím šablónovacieho systému Jinja. Ak sa užívateľ nachádza v režime zobrazenia obsahu tohto súboru, môže si pomocou klávesy F4 zobraziť podporované dáta pre tvorbu šablóny.

```
proj1/
   - history/
      - testsuite_history.txt
   - reports/
      - report_template.j2
   - tests/
      - scoring
      - sum
      - testsuite.sh
   - xlogin00/
   - xlogin01/
   - xlogin02/
   - proj_conf.yaml
```

Výpis 5.2: Typická adresárová štruktúra projektu

Vo výpise 5.2 je zhrnutá štruktúra novovytvoreného adresára projektu, s pridanými odovzdanými riešeniami študentov „xlogin00“, „xlogin01“ a „xlogin02“. Riešenia študentov musia byť vždy uložené v koreňovom adresári projektu.

5.4 Interný editor

Pre prechádzanie obsahu súboru je implementovaný interný editor. Po otvorení súboru v internom editore sa načíta jeho obsah a vytvorí sa nová inštancia objektu `Buffer`, kde sa obsah súboru ukladá po riadkoch. Obsah súboru je možné v internom editore meniť. `Buffer` si pamätá vždy aktuálnu verziu obsahu súboru, ktorá je dostupná cez atribút `lines`. Okrem toho si `Buffer` udržiava ďalšie dve verzie súboru:

- atribút `original_buff` uchováva originálny obsah súboru, ktorý sa uloží pri jeho prvom načítaní (otvorení),

- atribút `last_save` uchováva posledne uložený obsah súboru, ktorý sa aktualizuje vždy keď užívateľ tento súbor uloží (klávesou F2).

Klávesou F8 je možné obnoviť obsah súboru z posledného uloženia (`last_save`) a klávesovou skratkou CTRL+R sa obnoví úplne originálny obsah súboru (`original_buff`).

Interný editor má dva režimy:

- režim správy (manažmentu) súboru, ktorý sa zapína klávesou ESC,
- režim úpravy (editovania) súboru, ktorý sa zapína klávesou „a“.

V režime editovania je možné ľubovoľne meniť obsah súboru, pričom sa tieto zmeny vykonávajú len interne vrámci triedy `Buffer`, preto je potrebné upravený obsah súboru aj uložiť (klávesou F2), čím sa prepíše pôvodný obsah súboru za nový.

Režim manažmentu slúži na vytváranie poznámok do externého reportu (viď 5.4.2) a na filtrovanie súborov podľa ich obsahu (viď sekcia 5.6).

5.4.1 Reprezentácia poznámok

Po otvorení súboru v internom editore sa okrem obsahu tohto súboru načíta aj súbor s poznámkami, ktorý k nemu patrí. Súbor s poznámkami je pomenovaný podľa názvu súboru s pridanou príponou „_report.yaml“. Príklad obsahu súboru s poznámkami možno vidieť vo výpise 5.3, kde každá poznámka je identifikovaná riadkom a stĺpcom v súbore. Napríklad poznámka „zle pomenovana premenna“ patrí na riadok 8, stĺpec 4.

```
3:
    42:
    - chyba bodkociarka ;
8:
    4:
    - zle pomenovana premenna
```

Výpis 5.3: Obsah súboru s uloženými poznámkami

Poznámky k súboru sa interne uchovávajú pomocou objektu `Report`, kde sú uložené ako zoznam inštancii objektu `Note`. `Report` si podobne ako `Buffer` uchováva aj pôvodnú originálnu kópiu poznámok, ktoré sa prípadne obnovujú spolu s obsahom súboru. `Note` je veľmi jednoduchý objekt s troma atribútmi: riadok, stĺpec a text poznámky.

Rozlišujú sa dva typy poznámok: typické a špecifické. Typické poznámky sú také, ktoré užívateľ zadáva často a sú spoločné pre všetky projekty. Zoznam typických poznámok sa ukladá v súbore *data/typical_notes.txt*, pričom sa neviažu na konkrétny riadok ani stĺpec (ukladá sa len text). Poznámke je možné zmeniť typ len v režime prechádzania poznámok, kam sa užívateľ môže prepnúť z interného editoru klávesou F7.

5.4.2 Vytváranie poznámok vrámci code review

Poznámkovanie do externého reportu je možné len v dvoch režimoch:

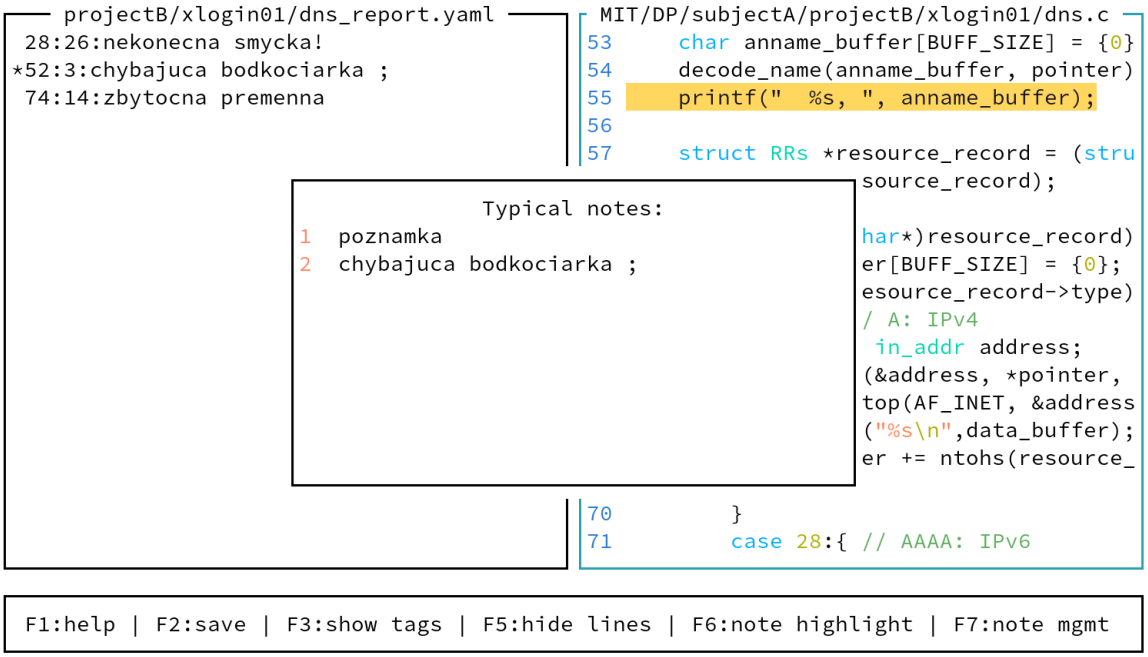
- v režime prechádzania obsahu súboru (v internom editore, režim správy súboru),
- v režime prechádzania poznámok.

Nová poznámka sa pridá vždy pre riadok a stĺpec aktuálnej pozície kurzoru v otvorenom súbore. Kurzor sám o sebe nie je dostatočne viditeľný, preto systém vždy zvýrazní riadok, ku ktorému sa poznámka pridáva, aby mal užívateľ jasne vyznačené k čomu sa poznámkou vyjadruje.

Pre pridanie typickej poznámky bolo pôvodne implementované samostatné menu so zoznamom uložených poznámok. Medzi týmito poznámkami mohol užívateľ prechádzať, jednu z nich vybrať a tá sa pridala do súboru. Princípom typických poznámok je, že sa používajú často a užívateľ teda nechce zakaždým prehľadávať celé menu aby našiel to, čo potrebuje. Preto bol systém prerobený tak, aby každá typická poznámka mala svoj vlastný index, pomocou ktorého sa môže rovno pridať do súboru. Tieto indexy sú v intervaloch čísiel 1 až 9 a písmen A až Z. Nevýhodou oproti pôvodnej implementácii je, že sú typické poznámky obmedzené na maximálny počet 35 (čo by sa dalo prípadne rozšíriť buď kombináciou spomínaného menu alebo pridaním ďalších komplexnejších indexov).

Ak chce teda užívateľ pridať typickú poznámku, zadá ju jednoducho klávesou, ktorá odpovedá indexu danej poznámky. Napríklad ak vie, že má na indexe 6 uloženú poznámku XY, nemusí ju zdĺhavo hľadať cez menu, ale má ju dostupnú hneď pod klávesou 6. Ak si užívateľ nepamätá, ktorá poznámka je uložená pod ktorým indexom, môže si zobraziť zoznam typických poznámok pomocou klávesy F9 a následne jednu z nich pridať.

Pre pridanie špecifickej poznámky je vyhradená klávesa s číslom 0, kedy sa užívateľovi otvorí centrálna obrazovka pre zadanie vstupu. Pre uchovávanie a spracovanie užívateľovho vstupu sa využíva objekt `UserInput`, ktorý má vlastný atribút `pointer`, pomocou ktorého sa orientuje v zadanom texte. Ak je zadaný text dlhší než je obrazovka, zalomí sa na nový riadok. Po zadaní vstupu sa text uloží ako poznámka k aktuálne otvorenému súboru a zvýrazní sa riadok s novou poznámkou (viď 5.4.3). Ako už bolo spomenuté, centrálnu obrazovku pre zadanie vstupu je možné posúvať vpravo alebo vľavo (napríklad ak cez obrazovku nie je vidieť riadok, ku ktorému sa poznámka pridáva).



Obr. 5.2: Zobrazenie typických poznámok

Na obrázku 5.2 je znázornená ukážka vloženia typickej poznámky. V pravom okne je možné vidieť otvorený súbor *dns.c*. V tomto okne je žlto zvýraznený riadok 55, na ktorý užívateľ pridáva novú poznámku. V centrálnom stredovom okne je zobrazený zoznam uložených typických poznámok s indexom 1 a 2. Ak užívateľ teraz stlačí klávesu 2, pridá sa poznámka „chybajuca bodkociarka ;“ na riadok 55 v súbore *dns.c*. V ľavom okne je zobrazený zoznam pridaných poznámok k tomuto súboru. Vidíme, že už v tomto súbore sú pridané tri poznámky (na riadku 28, 52 a 74).

Pre zobrazenie všetkých pridaných poznámok k danému súboru musí užívateľ zapnúť režim prechádzania poznámok. Potom sa mu na ľavej strane obrazovky zobrazí zoznam s pridanými poznámkami, pričom každá je identifikovaná ako riadok:stĺpec:text (viď obrázok 5.2, ľavé okno). Medzi poznámkami je možné prechádzať šípkami a prípadne ich mazať alebo editovať. Z tohto režimu je možné skočiť do otvoreného súboru priamo na riadok s aktuálne vybranou poznámkou (klávesou F5).

5.4.3 Zvýraznenie poznámky v editore

Jednou z požiadaviek na systém bolo, aby v súbore boli vyznačené riadky, ku ktorým sa vzťahuje nejaká poznámka. Pôvodne bolo zvýraznenie poznámky riešené tak, že sa riadok s poznámkou zvýraznil celý žltou farbou (podobne ako keď sa pridáva nová poznámka). To však začalo pri väčšom počte poznámok pôsobiť rušivo. Preto bola zvolená alternatíva, kedy sa zvýrazní len číslo riadku s poznámkou, prípade ak je číslovanie riadkov vypnuté, zvýrazní sa na začiatku riadku jedna medzera.

naty/MIT/DP/subjectA/projectB/xlogin01/dns.c

```
16 #define BUFF_SIZE 1024
17 char buffer[BUFF_SIZE];
18
19 // Function for parsing NAME, QNAME, RDATA
20 void decode_name(char *n_buff,char **recv){
21     char *name = n_buff;
22     while(1){
23         unsigned char size = **recv;
24         (*recv)++;
25         if(size==0){
```

naty/MIT/DP/subjectA/projectB/xlogin01/dns.c

```
16 #define BUFF_SIZE 1024
17 char buffer[BUFF_SIZE];
18
19 // Function for parsing NAME, QNAME, RDATA
20 void decode_name(char *n_buff,char **recv){
21     char *name = n_buff;
22     while(1){
23         unsigned char size = **recv;
24         (*recv)++;
25         if(size==0){
```

Obr. 5.3: Zvýraznenie poznámky (vľavo pôvodné riešenie, vpravo aktuálne riešenie)

Na obrázku 5.3 sú ukážky oboch implementácii. Vľavo je pôvodné riešenie s vyznačeným celým riadkom 23 a vpravo je aktuálne riešenie s vyznačeným číslom 23.

Medzi riadkami s poznámkami je možné skákať pomocou klávesových skratiek CTRL a šípka hore alebo dole. Napríklad ak by v súbore boli len dve poznámky na riadku 5 a 10, pričom kurzor by sa aktuálne nachádzal na riadku 7, potom po stlačení kláves CTRL a šípka dole, by sa kurzor presunul na najbližší spodný riadok s poznámkou, čo je riadok 10.

5.4.4 Zvýraznenie syntaxe

Interný editor podporuje zvýraznenie syntaxe pre mnohé jazyky a textové formáty. Využíva k tomu knižnicu pygments, kde bolo potrebné vytvoriť vlastné formátovanie a štýl tak aby sa výsledok prispôsobil knižnici ncurses.

Knižnica pygments poskytuje funkcie pre spracovanie textu na tokeny podľa zadaného lexeru. Lexer je špecifický pre každý jazyk či textový formát. Pygments dokáže na základe názvu súboru vybrať vhodný lexer, ktorý sa následne použije pre spracovanie obsahu súboru.

Následne je potrebné vybrať vhodný formáter, ktorý na základe zadaného štýlu naformátuje jednotlivé tokeny na odpovedajúce farby. Nakoľko knižnica pygments podporuje len farby zadané hexadecimálnou hodnotou, bolo potrebné vytvoriť nový štýl, ktorý používal farby, s ktorými vie pracovať aj knižnica ncurses. Ncurses dokáže vykresľovať len farby, ktoré má dopredu definované, no vytvoriť špecifické farby pomocou hexadecimálnych hodnôt je veľmi náročné, pretože to priamo táto knižnica nepodporuje.

Pomocou knižnice ncurses sa najskôr definovali farby pre zvýraznenie syntaxe. Tieto farby sú identifikované celými číslami v rozmedzí 40 až 60. Rovnaké čísla sa následne použili v novovytvorenom štýle knižnice pygments a to tak, že sa z nich umelo vytvorili hexadecimálne hodnoty vo formáte „#0000xx“. Teda napríklad číslo 45 je reprezentované ako „#000045“.

Výsledkom teda je, že pygments postupne priradí tokenom farby s hodnotami „#000040“ až „#000060“ podľa novovytvoreného štýlu. Okrem nového štýlu bolo následne potrebné vytvoriť aj vlastný formater, ktorý z hodnoty „#000045“ dokázal spätne vytvoriť hodnotu 45, pod ktorou má ncurses uloženú vlastnú farbu.

```
from pygments import highlight, lexers
def parse_code(file_name, code):
    lexer = lexers.get_lexer_for_filename(file_name, stripln=False)
    curses_format = CursesFormatter(style='ncurses')
    text = highlight(code, lexer, curses_format)
    return text
```

Výpis 5.4: Využitie knižnice pygments pre zvýraznenie syntaxe

Výpis 5.4 ukazuje využitie nového formateru `CursesFormatter` a nového štýlu `ncurses`. Vstupom je obsah súboru uložený v premennej `code` a názov súboru uložený v premennej `file_name`, a výstupom je naformátovaný text s pridaným štýlom. Najskôr sa pomocou knižnice pygments vybral správny lexer podľa názvu súboru, následne sa vytvoril vlastný formater využívajúci nový štýl a nakoniec sa pomocou funkcie `highlight` vykoná samotné parsovanie textu a formátovanie.

5.5 Značkovanie

Zavedenie systému značiek prináša mnohé výhody a možnosti využitia pri hodnotení študentských projektov. Pomocou značky je možné označiť študentský projekt a priradiť tak k nemu určité informácie. Značkovanie je obmedzené len vrámci nejakého (pod)adresára existujúceho projektu.

Značky sa ukladajú len v špecifických súboroch, podľa toho k čomu sa vzťahujú. Značka sa môže vzťahovať:

- k študentskému riešeniu (súbor *solution_tags.yaml*),
- k výsledkom z automatických testov (súbor *tests_tags.yaml*),
- ku konkrétnemu testu (súbor *test_tags.yaml*),
- k celej testovacej sade (súbor *testsuite_tags.yaml*).

Vo výpise 5.5 je znázornené rozmiestnenie súborov so značkami (modro vyznačené súbory) v typickom adresári projektu. Adresár *proj1/tests/* obsahuje dva testy (*test_1* a *test_2*), pričom ku každému testu prislúcha samostatný súbor *test_tags.yaml* so značkami k tomuto testu. Typicky ide o značky definujúce verziu testu alebo typ testu. Užívateľ môže pomocou týchto značiek filtrovať testy. Okrem toho je v adresári *proj/tests/* ešte jeden súbor so značkami (*testsuite_tags.yaml*), ktorý obsahuje značky vzťahujúce sa k celej testovacej sade (napríklad verzia testovacej sady).

Ku každému adresáru študentského riešenia prislúcha jeden samostatný súbor so značkami (napríklad *proj/xlogin01/solution_tags.yaml*). V tomto súbore sa ukladajú značky súvisiace s daným riešením. Typicky ide o značky, pomocou ktorých sa napríklad označí riešenie ako skupinové, plagiátorské, neskoro odovzdané či nesprávne pomenované a podobne. Adresár *proj1/xlogin01/tests/* obsahuje výsledky z automatických testov *tests_tags.yaml*, do ktorého jednotlivé testy automaticky ukladajú značky s hodnotením či výsledkami testu.

```
proj1/
   - history/
   - reports/
   - tests/
      - test_1/
      - test_2/
           dotest.sh
           test_tags.yaml
      - testsuite.sh
      - testsuite_tags.yaml
   - xlogin00/
   - xlogin01/
      - tests/
           test_1/
           test_2/
           tests_tags.yaml
      - solution_tags.yaml
   - proj_conf.yaml
```

Výpis 5.5: Rozmiestnenie súborov so značkami v adresári projektu

Pre správu značiek je vyhradený samostatný režim prechádzania značiek, ktorý pracuje s obrazovkou F (viď obrázok 5.1). Značky sú uchovávané pomocou objektu `Tags`, kde sú uložené v dátovej štruktúre slovník. Každá značka sa skladá z jedného názvu (kľúč v slovníku) a niekoľkých parametrov, ktoré sú uložené vo forme zoznamu (môže byť aj prázdny).

5.5.1 Manuálna správa značiek

V režime prechádzania značiek je možné manuálne spravovať značky, čo zahŕňa tvorbu nových značiek, úpravu existujúcich značiek či ich mazanie. Vytvorenie novej značky (klávesa F3) je realizované pomocou centrálneho okna s využitím objektu `UserInput`. Zadáva sa vo forme: `názov_značky parametre`, pričom názov značky nesmie obsahovať medzeru. Parametre sú voliteľné a sú oddelené medzerou, pričom ich môže byť maximálne 10. Parameter obalený úvodzovkami je považovaný za jeden samostatný parameter aj v prípade, ak obsahuje medzery.

5.5.2 Automatické značkovanie

Systém môže v niektorých prípadoch pridávať značky automaticky. Napríklad:

- pri modifikácií testov sa automaticky zmení značka s verziou testu,
- pri nesprávnom pomenovaní študentského projektu sa pridá značka `#renamed_sut` informujúca o tejto skutočnosti,
- pri počítaní celkového skóre za projekt sa automaticky pridá značka `#score` s bodovým hodnotením,
- po ukončení testovania sa automaticky pridá značka `#last_testing` s dátumom testovania
- a počas automatického testovania sa pridávajú prípadne ďalšie značky.

Automatické značkovanie počas testovania spočíva v tom, že pri spustení automatických testov sa k študentskému riešeniu pridávajú značky s výsledkami. Testy implementované ako shell skripty môžu využívať pripravenú funkciu `add_test_tag`, ktorá pridá značku do súboru *tests_tags.yaml*. Tejto funkcii je potrebné predať minimálne jeden parameter pre názov značky a ďalšie voliteľné parametre funkcie reprezentujú parametre značky. Typicky sa automatické značkovanie používa na pridávanie značiek pre hodnotenie projektu a informácie o tom ako test dopadol. Napríklad po spustení testu *test_1* sa automaticky vytvoria značky `#test_1_ok()` a `#scoring_test_1(2)`, ktoré hovoria o tom, že test prebehol úspešne a počet bodov za tento test je 2.

5.6 Filtrovanie súborov

Filtrácia súborov bola implementovaná podľa návrhu tak, aby boli poskytnuté tri typy samostatných filtrov (v zátvorke je uvedený režim, v ktorom sa daný filter spravuje):

- filter podľa cesty (v režime prechádzania adresárovej štruktúry),
- filter podľa obsahu (v režime prechádzania obsahu súboru - režim správy súboru),
- filter podľa značky (v režime prechádzania značiek).

Každý filter je možné zadať po stlačení klávesy „/“, čím sa v príslušnej obrazovke vyhradí posledný riadok pre zadanie filtra užívateľom. Všetky tri typy filtrov sa ukladajú pomocou objektu `Filter` v atribútoch `path`, `content` a `tag` v podobe textu.

Filter podľa cesty sa zameriava len na súbory, ktoré odpovedajú zadanej ceste. Napríklad filter *tests/**/dotest* zobrazí len súbory, ktoré sa nachádzajú v (pod)adresári *tests* a ich názov začína prefixom *dotest*. Vo výpise 5.6 je ukážka implementácie filtra podľa cesty, kde vstupom je textový parameter *dest_path* so zadaným filtrom a výstupom je zoznam odpovedajúcich súborov. Na prehľadávanie adresárovej štruktúry sa využíva knižnica *glob*, ktorá rekurzívne filtruje obsah adresára podľa zadanej cesty *dest_path* a vráti zoznam ciest, ktoré odpovedajú danému vstupu. Tieto výsledky sa následne filtrujú a vyberajú sa z nich len súbory.

```
import glob
import os
def get_files_by_path(dest_path):
    path_matches = []
    for file_path in glob.glob(dest_path, recursive= True):
        if os.path.isfile(file_path):
            path_matches.append(file_path)
    return path_matches
```

Výpis 5.6: Filtrovanie súborov podľa cesty

Filter podľa obsahu pozerá na obsah všetkých súborov (prípadne len súborov už filtrovaných po zadaní filtra podľa cesty) a hľadá v nich zadaný reťazec. V prípade, že filter prehľadáva väčšie množstvo súborov, môže tento proces trvať dlhšiu dobu, nakoľko sa každý jeden súbor číta a hľadá sa v ňom filtrovaný reťazec.

Filter podľa značiek je mierne náročnejší a vyžaduje viac času. Pre každý súbor je najskôr potrebné zistiť, ktorý súbor so značkami k nemu patrí, respektíve je treba zistiť, kde sú uložené značky vzťahujúce sa k tomuto súboru. Ako je naznačené v adresárovej štruktúre vo výpise 5.5, značky sa ukladajú do rôznych súborov podľa toho, k čomu sa vzťahujú. Podľa cesty súboru je možné zistiť či k nemu nejaký súbor so značkami patrí a ak áno, tak ktorý. Značky študentských riešení (tj. súbory *solution_tags.yaml* a *tests_tags.yaml*) sú dopredu načítané a uložené v triede `Solution` pod atribútmi `tags` a `test_tags`. Ostatné súbory so značkami sa musia dynamicky načítať, čo môže pri väčšom množstve súborov trvať dlhšie. Ak už sú značky súboru načítané, hľadá sa v nich počadovaná značka zadaná cez filter.

Vyhľadávaná značka môže byť zadaná vo forme výrazu a to v rôznych podobách:

- je zadaný len názov značky,
- je zadaný názov značky a číslo parametra,
- je zadaný názov značky, číslo parametra a porovnanie parametra.

Ak je zadaný len názov značky, v jednotlivých súboroch sa hľadá táto značka bez ohľadu na to, či má nejaké parametre alebo nie. Pri hľadaní zadanej značky sa kontroluje len prefix značiek, teda napríklad pre vyhľadanie značky `#documentation` stačí do filtra zadať napríklad prefix `docu`.

Značka s parametrom sa zadáva vo forme „`tag.param`“, kde `tag` je názov značky (alebo jej prefix) a `param` je celé číslo väčšie ako 0, ktoré udáva index parametra. Ak je zadaný názov značky s parametrom, pre nájdenú značku sa naviac kontroluje, či existuje aj parameter so zadaným indexom. Napríklad pre značku `#score(12, bonus, 15/2)` bude výsledkom výrazu „`score.1`“ hodnota 12. Keďže v tomto prípade na hodnote parametra nezáleží, kontroluje sa len, či značka a parameter s daným indexom existuje.

Hodnoty parametrov značky je možné porovnávať zadaním celého výrazu vrátane porovnávacieho znamienka a cieľovej hodnoty parametra. Pre porovnanie je možné použiť len znamienka $<$, $>$ alebo $=$. Výraz s porovnaním hodnoty parametra sa zadáva vo forme „`tag.2 < hodnota`“. Ak by sme teda chceli filtrovať napríklad všetky súbory, ktoré majú značku s názvom `score`, s prvým číselným parametrom väčším ako 5 bodov, zadali by sme filter v podobe „`score.1 > 5`“.

5.6.1 Agregácia

Výsledkom filtrovania je zoznam súborov. Niekedy však potrebuje vyučujúci vidieť len zoznam študentov (adresárov so študentskými riešeniami), ktorí prešli zadaným filtrom. Preto systém ponúka možnosť agregácie filtrovaných súborov. Agregácia prebieha na základe spoločných súborov so značkami.

Vo výpise 5.7 je ukážka filtrovaných súborov pred agregáciou a po agregácii. Súbory *file1.txt*, *file2.txt* a *file3.txt* sa všetky vzťahujú k jednému riešeniu, ktoré má značky uložené v jednom súbore *solution_tags.yaml*. Preto sa po agregácii tieto súbory zlúčili na spoločný adresár daného riešenia.

```
# výsledok filtrovania pred agregáciou
xlogin00/file1.txt
xlogin00/file2.txt
xlogin00/file3.txt
xlogin01/file1.txt
xlogin01/file3.txt
xlogin02/file1.txt
xlogin02/file2.txt
# výsledok filtrovania po agregácii
xlogin00/
xlogin01/
xlogin02/
```

Výpis 5.7: Ukážka agregácie filtrovaných súborov

5.7 Zobrazenie informácii o projekte

Ako bolo spomínané v sekcii 5.3.1, pri založení nového projektu sa vytvorí konfiguračný súbor, v ktorom sú okrem iného definované aj požiadavky na zobrazovanie informácii o projekte. Informácie sa zobrazujú v adresárovej štruktúre vždy v riadku pri adresári študentského riešenia a pri adresári s výsledkom testu. Celkovo sú teda pre projekt definované dve skupiny informácii: `solution_info` a `tests_info`.

Definovanie zobrazovaných informácií vymedzuje, čo sa má pri danom adresári zobraziť a za akých podmienok. Každá zobrazovaná informácia má definovanú nasledujúcu štruktúru:

- `identifier`: celé číslo,
- `visualization`: znak, text alebo odkaz na parameter značky,
- `length`: celé číslo,
- `description`: text,
- `predicates`: zoznam predikátov.

`Identifier` je identifikátor informácie, ktorému možno priradiť celé číslo unikátne vrámci jednej skupiny informácii. Identifikátor určuje prioritu informácie, podľa čoho sa určuje poradie, v akom sa informácie zobrazujú v riadku s adresárom. Počet zobrazovaných informácií

záleží aj od veľkosti obrazovky. Zobrazujú sa primárne informácie s najvyšším identifikátorom a radia sa sprava, pričom medzi jednotlivými informáciami je vždy jedna medzera. Ak sa niektoré informácie už nezmestia do riadku s adresárom, nebudú zobrazované.

`Visualization` je vizualizácia informácie, teda určenie toho, čo sa má zobraziť. Ideálne je používať vo vizualizácii len jeden znak. Čím kratšie informácie sa zobrazujú, tým viac sa ich na obrazovku zmestí. Okrem znaku či textu je možné zobraziť aj hodnotu parametra značky. Pri odkazovaní na parameter značky je nutné použiť striktný formát (podobne ako pri filtrovaní podľa značky): `tag.param` kde `param` je celé číslo väčšie ako 0, ktoré udáva index parametra značky. V tomto prípade je zobrazenie informácie podmienené existenciou značky s daným parametrom.

Príklad využitia: Chceme u každého študenta vidieť, kedy bol naposledy testovaný a vieme, že po testovaní sa automaticky pridáva k riešeniu značka `#last_testing(date)`, potom môžme definovať zobrazenie informácie kde do vizualizácie zadáme odkaz na parameter značky vo forme „`last_testing.1`“. U každého študenta sa potom zobrazí dátum posledného testovania.

Takéto dynamické informácie môžu niekedy „rozbiť“ vizuálne zarovnanie zobrazovaných informácii, preto sa pri používaní odkazu na parameter značky odporúča zadať aj nasledujúci parameter `length`.

`Length` je voliteľná časť definície informácie, ktorá obmedzuje dĺžku vizualizácie. Zároveň sa vďaka definovanej dĺžke udržiava zarovnanie rovnakých informácii pod seba. Ak teda u niektorej položky určitá informácia chýba, zobrazí sa len medzera o veľkosti tejto dĺžky.

Príklad využitia: Chceme zobrazovať počet bodov za projekt u každého študenta a vieme, že maximálny počet bodov je 99, teda ide o maximálne dvojciferné čísla, potom môžme obmedziť vizualizáciu na dva znaky. Vďaka tomu budú ak jednociferné čísla zarovnané na dva znaky a rovnako tak aj riešenia, ktoré ešte hodnotené neboli, budú mať namiesto čísla zobrazenú medzeru o veľkosti dvoch znakov.

`Description` je ďalšia voliteľná časť informácie, ktorá slúži skôr pre užívateľa, aby vedel, čo daná informácia zobrazuje. Ide teda len o krátky popis informácie.

Posledná a najdôležitejšia časť je `predicates`. Ide o zoznam predikátov, ktoré určujú kedy sa má informácia zobraziť a s akou farbou. Každý predikát má dve položky:

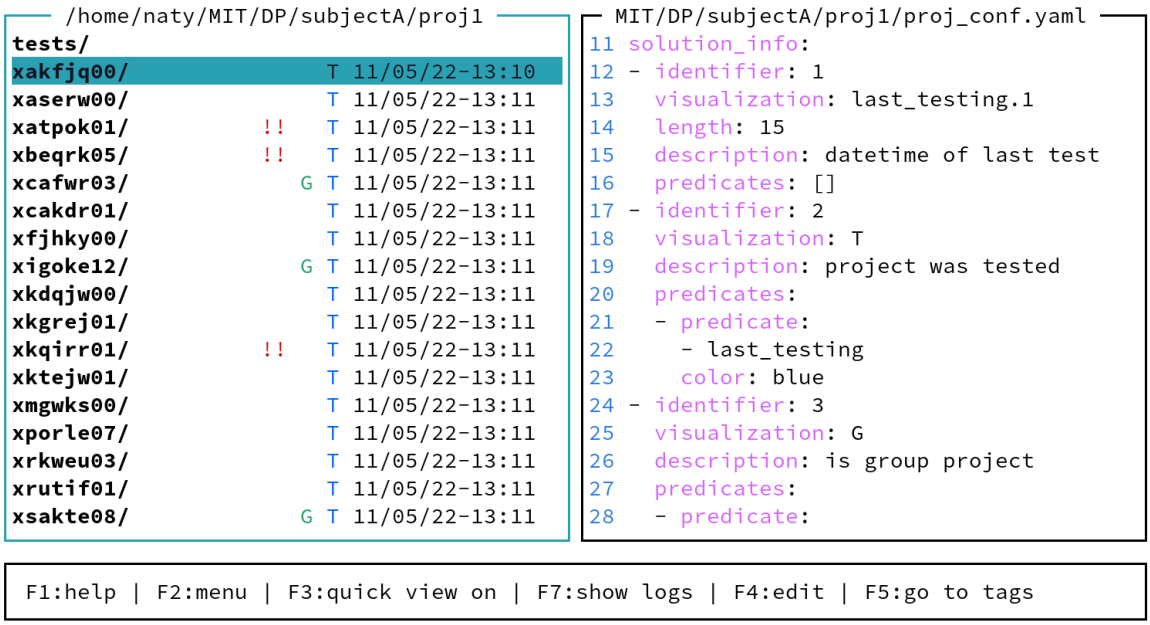
- `predicate`: zoznam podmienok,
- `color`: farba vizualizácie informácie.

Predikáty sa vyhodnocujú postupne za radom v zozname zhora dole, ako sú definované v konfiguračnom súbore projektu. Pre zobrazenie definovanej informácie musí byť splnený aspoň jeden predikát. Podľa toho, ktorý predikát je splnený, takou farbou sa zobrazí informácia. V prípade, že sa nájde prvý predikát, ktorý je splnený, na ostatné sa už nehľadí a použije sa farba tohto predikátu.

`Predicate` definuje zoznam podmienok pre zobrazenie informácie. Aby bol predikát splnený, musia byť splnené všetky podmienky z tohto zoznamu. Podmienky sú definované ako výrazy nad značkami (rovnako ako pri filtrovaní podľa značky):

- odkazuje sa na existenciu značky (bez ohľadu na jeho parametre),
- odkazuje sa na existenciu parametra značky (bez ohľadu na hodnotu parametra),
- odkazuje sa na hodnotu parametra značky, ktorá sa s niečim porovnáva.

`Color` určuje, akou farbou sa informácia zobrazí, ak je splnený daný predikát. Farba sa zadáva pomocou vopred definovaných hodnôt vo forme textu: `red`, `green`, `blue`, `cyan`, `yellow`, `orange` alebo `pink`. Prípadne je možné zadať prázdny reťazec (respektíve nechať túto položku prázdnu). V takom prípade sa informácia zobrazí základnou farbou textu.



Obr. 5.4: Zobrazenie informácii o projekte

Na obrázku 5.4 je príklad zobrazenia informácii o projekte. Na ľavej obrazovke je možné vidieť adresárovú štruktúru projektu so zobrazenými informáciami pri študentských adresároch. Na pravej obrazovke je zobrazený obsah konfiguračného súboru projektu, v ktorom sú definované zobrazované informácie. V tomto prípade sú u každého študenta zobrazované informácie (z pravej strany):

- dátum a čas posledného testu,
- písmeno „T“, ktoré značí, že riešenie bolo testované,
- písmeno „G“, ktoré značí, že je riešenie skupinové,
- reťazec „!!“, ktorý značí, že riešenie je plagiátorské.

Tieto základné informácie sú definované v konfiguračnom súbore každého novovytvoreného projektu. Konfiguračný súbor je možné doplniť o ďalšie zobrazované informácie, napríklad aby bolo u každého študenta vidieť rovno výsledok nejakého konkrétneho testu alebo celkový počet bodov za projekt.

V prípade definície zobrazovaných informácií pre výsledky testov je implementovaná špeciálna konštanta, ktorú je možné použiť ako súčasť značky v predikátoch alebo vo vizualizácii. Ide o konštantu `XTEST`, ktorá sa pri vyhodnocovaní predikátu nahradí za názov testu, ktorý sa aktuálne vyhodnocuje. Ak by sme teda chceli pri každom teste vidieť, či tento test prešiel, môžme v predikáte použiť túto konštantu ako súčasť značky: `XTEST_ok`.

5.8 Zobrazenie záznamov

Systém ponúka funkcie, ktoré sa vykonávajú na pozadí bez toho, aby užívateľ videl, že sa vykonali. V takýchto prípadoch je vhodné mať možnosť sledovať, čo sa v systéme deje a či sa vôbec daná akcia dokončila. Z toho dôvodu systém vytvára záznamy (anglicky *logs*), ktoré je možné zobraziť v samostatnej užívateľskej časti na obrazovke C (viď obrázok 5.1). Do režimu prezerania záznamov sa užívateľ môže prepnúť pomocou tabulátoru a šípkami v ňom môže prechádzať. Obrazovku so záznamami je možné schovať pomocou klávesy F7 v režime prechádzania adresárovej štruktúry.

Záznamy informujú prevažne o dlho trvajúcich akciách, respektíve o ich priebehu (napríklad pri spúšťaní testov), varujú o problematickom vykonaní akcie či zlyhaní akcie (napríklad nepodarené rozbalenie archívu študenta), prípadne informujú o úspešných akciách (napríklad vytvorený nový test alebo založený nový projekt).

Každý záznam obsahuje nasledujúce informácie:

- dátum a čas vytvorenia záznamu,
- typ záznamu (varovanie, chyba alebo informovanie),
- textová správa záznamu.

Záznamy sa ukladajú do špeciálneho súboru *logs.csv* a po ukončení systému sa nezmažú. Ak chce užívateľ premazať záznamy, môže tak spraviť pomocou klávesy F9 v režime prezerania záznamov. Nakoľko sa záznamy na začiatku spustenia systému načítavajú zo súboru, odporúča sa ich pred ukončením systému vždy premazať, aby systém nabehol rýchlejšie. Ak si však užívateľ chce uchovávať tieto záznamy aj pre budúce použitie systému, nie je potrebné ich mazať, avšak treba počítať s tým, že inicializácia systému bude trvať o čosi dlhšie.

```
11/3/22-14:42 | INFO    | new project created (proj1)
11/3/22-14:43 | INFO    | creating new test...
11/3/22-15:02 | INFO    | test 'test_1' created (scoring ok=1,fail=0)
11/3/22-15:30 | INFO    | extracting all solution archives...
11/3/22-15:30 | WARNING | problematic archives: {'xlogin03.7z'}
11/3/22-15:33 | INFO    | *** testing student 'xlogin00' ***
11/3/22-15:33 | INFO    | testing done
11/3/22-15:33 | INFO    | *** testing student 'xlogin01' ***
11/3/22-15:33 | INFO    | testing done
11/3/22-15:33 | INFO    | *** testing student 'xlogin02' ***
11/3/22-15:33 | ERROR   | FUT 'dns.c' doesnt exists in solution directory
11/3/22-15:33 | INFO    | testing done
11/3/22-15:33 | INFO    | testing all students done !!
```

Výpis 5.8: Ukážka výpisu záznamov

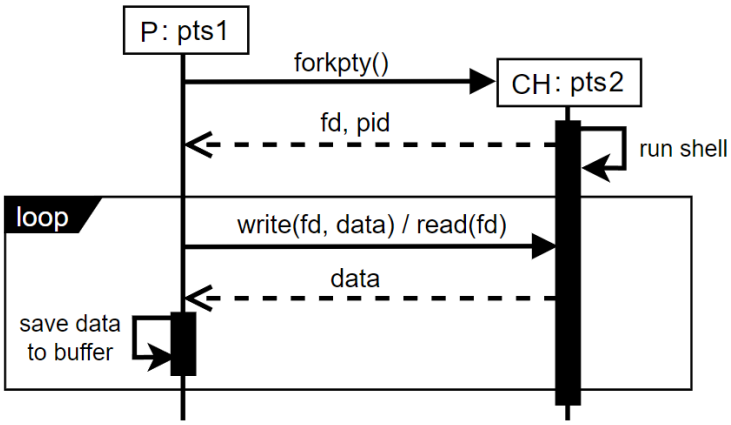
Vo výpise 5.8 je ukážka obsahu súboru *logs.csv*. Na každom riadku je zľava vyznačený dátum a čas vytvorenia záznamu. Následne je identifikovaný typ záznamu, či sa jedná o informovanie, varovanie alebo upozornenie na chybu pri vykonávaní danej akcie. Nakoniec je vypísaný samotný text záznamu, na základe ktorého je možné vidieť vykonávané akcie so systémom.

Na tomto príklade môžme vidieť, že užívateľ najskôr založil nový projekt `proj1` a vytvoril v ňom test `test_1`. Následne užívateľ rozbalil archívy študentov, z ktorých študenta `xlogin03` sa nepodarilo rozbaliť a systém o tom varuje užívateľa. Nakoniec užívateľ spustil automatické testy, pričom môžme vidieť, že študent `xlogin02`, neodovzdal správne pomenované riešenie s názvom „*dns.c*“ a teda nie je čo testovať. Ostatné študentské riešenia boli testované úspešne.

5.9 Prepnutie na interaktívny Shell

Z každého spomínaného režimu je možné sa kedykoľvek prepnúť na interaktívny shell a neskôr sa zase vrátiť naspäť. Prepínanie je realizované pomocou klávesovej skratky CTRL+O. Chovanie je inšpirované nástrojom Midnight Commander.

Pre vytvorenie samostatného procesu, v ktorom pobeží shell, bolo využité systémové volanie `forkpty()`[2], ktoré vráti otvorený súborový deskriptor `fd` pre komunikáciu s novovytvoreným samostatným pseudo-terminálom (viď obrázok 5.5). Rodičovský proces má na starosti riadenie systému a komunikáciu s novým terminálom. Komunikácia zahŕňa čítanie, zápis a ioctl funkcie. Ioctl systémové volanie bolo potrebné na prispôsobenie veľkosti pseudo-terminálu v prípade, že užívateľ menil veľkosť hlavného terminálu, v ktorom bežal framework. Čítanie a zápis prebieha cez súborový deskriptor pseudo-terminálu, pričom sa načítané dáta naviac ukladajú do samostatného bufferu pomocou vyčleneného vlákna. Tento buffer sa vždy vypíše na obrazovku, keď sa užívateľ prepne do interaktívneho shellu klávesovou skratkou CTRL+O (vďaka čomu užívateľ vidí aj históriu shellu).



Obr. 5.5: Sekvenčný diagram komunikácie medzi procesmi

Na obrázku 5.5 je znázornená komunikácia medzi rodičovským procesom `P`, ktorý beží v termináli `pts1` a jeho potomkom `CH`, ktorý beží v samostatnom termináli `pts2`. Proces `CH` spustí vo svojom termináli `shell` a čaká. `P` komunikuje s potomkom pomocou súborového deskriptoru `fd`, volaním funkcii `write` alebo `read`. V prípade čítania z `fd` potomok vráti požadované dáta, ktoré si následne rodič ukladá do bufferu.

[2] https://linux.die.net/man/3/forkpty

5.10 Tvorba izolovaného prostredia

Dynamické testovanie študentských projektov prebieha v izolovanom prostredí s využitím linuxových kontajnerov a nástroja Docker. Pre spustenie testov v izolovanom prostredí je potrebné, aby bol dopredu vytvorený Docker image s názvom „test“. Ak takýto image neexistuje, užívateľ ho môže vytvoriť pomocou systémovej funkcie cez menu (klávesa F2) v režime prechádzania adresárovej štruktúry. Image sa vytvára pomocou súboru `Dockerfile`, ktorý musí byť uložený v koreňovom adresári projektu. Ak takýto súbor neexistuje, je možné ho vytvoriť cez systémovú funkciu v menu, kedy sa užívateľovi zobrazí centrálna obrazovka pre zadanie základných údajov pre `Dockerfile`. Následne sa v koreňovom adresári projektu vytvorí súbor `Dockerfile` so zadanými údajmi, ktorý je možné prípadne upraviť a prispôsobiť konkrétny projekt.

```
FROM centos:7
RUN addgroup -g 1002 test 2>/dev/null || groupadd -g 1002 test
RUN adduser -D -u 1000 -G test test 2>/dev/null || useradd -u
1000 -g test test
USER test
```

Výpis 5.9: Ukážka súboru Dockerfile

Vo výpise 5.9 je ukázaný príklad novovytvoreného súboru `Dockerfile` po zadaní distribúcie *Centos 7*, identifikátora užívateľa *1000* a identifikátora skupiny *1002*. Podľa zadanej distribúcie sa vytvorí skupina „test“ s identifikátorom *1002*, pomocou funkcie `addgroup` alebo `groupadd`. Obdobne sa vytvorí užívateľ „test“ s identifikátorom *1000*, pomocou funkcie `adduser` alebo `useradd` a priradí sa mu vytvorená skupina „test“.

5.11 Testovanie projektov

Testovanie študentských projektov pozostáva z dvoch základných častí: code review a spúšťanie automatizovaných testov. Pre podporu code review slúži spomínaný interný editor, v ktorom je možné vytvárať poznámky do externého reportu. Pre spúšťanie testov sa využíva izolované prostredie, ktoré je možné vytvoriť pomocou ponúkaných systémových funkcii. Okrem toho systém poskytuje ďalšie funkcie pre podporu testovania študentských riešení, ako je napríklad:

- rozbalenie študentského archívu (podpora pre: `.zip`, `.tar`, `.tgz`, `.tbz` a `.tgz`),
- premenovanie študentského riešenia na požadovaný formát
- hromadné spúšťanie testov nad viacerými riešeniami,
- hromadné čistenie výsledkov testov.

Pri rozbaľovaní študentských archívov sa prehľadáva koreňový adresár projektu, v ktorom sú uložené študentské riešenia. Všetky archívy, ktorých názov sa zhoduje s identifikátorom riešenia (definovaný v konfiguračnom súbore projektu pod atribútom `solution_id`), sa rozbalia do samostatných adresárov v koreňovom adresári projektu. Následne je možné premenovať študentské riešenia na požadovaný formát. K tomu rovnako slúži konfiguračný súbor

projektu, v ktorom je definovaný požadovaný názov súboru s riešením (`sut_requested`) a zoznam podporovaných variánt tohto súboru (`sut_ext_variants`). Ak sa v adresári študentského riešenia nenachádza súbor s požadovaným názvom, hľadajú sa jeho podporované varianty. Ak obsahuje maximálne jeden súbor s podporovanou variantou názvu, tento súbor sa skopíruje a pomenuje sa podľa požadovaného názvu. Riešenia, ktoré neobsahujú žiaden súbor s podporovanou variantou riešenia, sú zobrazené v okne so záznamami (viď sekcia 5.8), aby mohol užívateľ tieto riešenia prípadne skontrolovať manuálne.

5.11.1 Tvorba testov a testovacej stratégie

Pre uchovávanie testovacej sady je koreňovom adresári projektu vyhradený samostatný adresár *tests*. Pre každý test musí byť v tomto adresári vytvorený samostatný adresár, ktorý obsahuje minimálne jeden súbor s názvom *dotest.sh* (napríklad vo výpise 5.5, súbor *proj1/tests/test_2/dotest.sh*).

Pre vytvorenie nového testu je implementovaná systémová funkcia dostupná cez menu v režime prechádzania adresárovej štruktúry. Zvolením tejto funkcie sa užívateľovi zobrazí centrálna obrazovka pre zadanie názvu nového testu. Ak užívateľ nešpecifikuje názov testu, vytvorí sa nový test s vygenerovaným názvom v podobe `test_i`, kde `i` je poradové celé číslo (väčšie ako 0).

S vytvorením nového testu sa v súbore *tests* vytvorí samostatný adresár s názvom nového testu a v tomto adresári sa vytvorí súbor *dotest.sh* pre definíciu testu. Tento súbor je možné priamo po vytvorení modifikovať. Pre implementáciu samotného testu sú poskytnuté podporné funkcie dostupné v zdrojovom súbore *data/tst.sh*. Zoznam podporných funkcií je možné zobraziť v režime úpravy súboru *dotest.sh* pomocou klávesy F4. Súbor *tst* s podpornými funkciami sa pre každý samostatný projekt nakopíruje do adresára *tests/src/* v koreňovom adresári projektu a je možné ho ľubovoľne upravovať pre potreby daného projektu. Je však nutné, aby tento súbor implementoval minimálne funkciu `get_fce_for_dotest()`, ktorá vráti vo forme textu po riadkoch oddelený zoznam podporných funkcii a ich stručný popis v podobe: „funkcia = popis“.

Na základe implementovaných testov je následne vhodné vytvoriť testovaciu stratégiu, k čomu slúži súbor *testsuite.sh* v adresári *tests*. Pre tvorbu testovacej stratégie je dostupná podporná funkcia `tst run`, ktorá berie ako parameter názov testu, ktorý spustí.

```
ulimit -f 2048
while read t; do tst run $t; done<<EOF
test_1
test_2
test_3
test_4
EOF
```

Výpis 5.10: Ukážka súboru s testovacou stratégiou

Vo výpise 5.10 je znázornená ukážka testovacej stratégie, kde sa najskôr nastaví limit na maximálnu veľkosť vytváraných súborov pomocou funkcie `ulimit`[3]. Následne sa spúšťajú sekvenčne testy `test_1` až `test_4` pomocou podpornej funkcie `tst run`, ktorá pre každý test spustí jeho skript *dotest.sh*.

[3] https://linuxcommand.org/lc3_man_pages/ulimith.html

5.11.2 Spúšťanie testov

Spúšťanie automatických testov sa realizuje pomocou menu v režime prechádzania adresárovej štruktúry. V menu sú dostupné dve funkcie pre testovanie. Jedna funkcia je zameraná na spustenie vytvorenej testovacej stratégie (súbor *testsuite.sh*). Druhá funkcia slúži na spustenie samostatných vybraných testov, kde sa užívateľovi zobrazí centrálna obrazovka so zoznamom implementovaných testov, z ktorých môže zvoliť jeden či viac testov, ktoré sa spustia sekvenčne. V zozname s testami sa prechádza šípkami a je možné označiť viacero testov buď pomocou klávesy F3, ktorá označí aktuálne zvolený test alebo pomocou klávesy s indexom pre daný test.

Spúšťať samostatné testy či celú testovaciu sadu je možné nad:

- jedným študentským riešením,
- všetkými študentskými riešeniami v koreňovom adresári projektu,
- vybranou množinou filtrovaných študentských riešení.

Spustenie testovacej sady pozostáva z viacerých krokov. V prvom rade systém skontroluje, či študentské riešenie obsahuje požadovaný testovaný súbor (definované v konfigurácii projektu pomocou atribútu `sut_requested`). Ak takýto súbor neexistuje, testovanie končí a systém o tom informuje užívateľa v obrazovke so záznamami. Okrem toho systém automaticky pridá k tomuto riešeniu značku `#scoring_missing_fut(0)`, čo môže neskôr užívateľ zahrnúť napríklad vo výpočte celkového hodnotenia (viď 5.12). Ak požadovaný súbor existuje a je spustiteľný, nasleduje čistiaca fáza, ktorá odstráni výsledky predošlého testovania. Následne sa skontrolujú potrebné súbory pre testovanie, ako je *tests/scoring*, *tests/sum*, *tests/testsuite.sh* a *tests/src/tst*.

Pred samotným testovaním je potrebné najskôr pripraviť izolované prostredie, v ktorom sa budú spúšťať testy. Potrebné dáta pre testovanie sa uložia do dočasného adresára *tmp/docker_shared*, ktorý obsahuje:

- adresár *sut* s kópiou adresára študentského riešenia,
- adresár *tests* s kópiou adresára s testami projektu,
- súbor *tests/run.sh*, ktorý riadi testovanie (spúšťa testovaciu sadu).

V tejto fáze sa predpokladá, že existuje docker image „test“, ktorý sa využije na vytvorenie samostatného docker kontajneru. Vytvorí sa teda kontajner, ktorému sa sprístupní obsah zdieľaného adresára *docker_shared* do adresára *opt* (viď výpis 5.11). Následne sa v kontajneri spustí skript */opt/tests/run.sh*, ktorý spúšťa samotné testy.

Výsledky testov sa ukladajú do adresára */opt/sut/tests*, kde sú organizované do samostatných adresárov podľa jednotlivých testov (generované záznamy či iné výstupy sa ukladajú do pod-adresára daného testu). Ak testy automaticky generujú značky s hodnotením, ukladajú sa do súboru */opt/sut/tests_tags.yaml*. Po ukončení testovania sa skopírujú výsledky testov zo zdieľaného adresára *docker_shared* do adresára študentského riešenia a odstráni sa celý kontajner. K študentskému riešeniu sa naviac automaticky vytvorí značka `#testsuite_version` s verziou spúšťanej testovacej sady a značka `#last_testing` s dátumom testovania. Poslednou fázou je výpočet bodového hodnotenia z automatických testov (viď sekcia 5.12).

Vo výpise 5.11 je možné vidieť prehľad adresárovej štruktúry izolovaného kontajneru, po ukončení testovania.

```
/opt/
   - tests/
      - test_1/dotest.sh
      - test_2/dotest.sh
      - testsuite.sh
      - run.sh
   - sut/
      - tests/
         - test_1/stdout
         - test_2/stdout
      - tests_tags.yaml
```

Výpis 5.11: Štruktúra zdieľaného adresára linuxového kontajneru

5.11.3 Verzovanie a história testov

Pre uchovávanie histórie testov bol zavedený veľmi jednoduchý verzovací systém testov s využitím značiek. Každý test si uchováva svoju aktuálnu verziu pomocou značky `#version` uloženej v súbore *test_tags.yaml* (viď prehľad súborov so značkami vo výpise 5.5). Rovnako tak sa udržiava verzovanie celej testovacej sady.

Pri vytváraní nového testu pomocou systémovej funkcie cez menu, sa novému testu priradí značka `#version` s počiatočnou verziou 1. Ako už bolo spomenuté, každý test musí obsahovať minimálne jeden súbor *dotest.sh*. Akákoľvek zmena tohto súboru sa považuje za modifikáciu testu. Pri modifikácii testu sa uloží kompletná kópia celého adresára testu do dočasného adresára *tmp*. Užívateľ tak má uloženú pôvodnú verziu testu a môže vykonávať potrebné úpravy. Po upravení a uložení súboru *dotest.sh* (klávesou F2 v režime prechádzania obsahu súboru) sa systém spýta užívateľa, či chce touto zmenou vytvoriť novú verziu testu. Ak užívateľ túto možnosť potvrdí, v adresári s históriou testov projektu sa vytvorí nový adresár, do ktorého sa skopíruje pôvodná verzia testu (uložená v dočasnom adresári *tmp*). Následne sa zvýši aktuálna verzia testu o 1 a rovnako tak sa zvýši verzia celej testovacej sady. Každou modifikáciou testu či pridaním alebo odstránením testu sa zvyšuje verzia testovacej sady.

História testov sa ukladá v adresári *history* v koreňovom adresári projektu (viď adresárová štruktúra projektu 5.2). Štruktúra adresára s uloženými historickými verziami testov je znázornená vo výpise 5.12.

```
proj1/history/
   - test_1/
      - version_1/
      - version_2/
      - version_3/
   - test_2/
      - version_1/
      - version_2/
   - testsuite_history.txt
```

Výpis 5.12: Typická adresárová štruktúra histórie testov

V adresári *history* sa okrem starých verzii testov uchováva aj súbor *testsuite_history.txt*, ktorý obsahuje zoznam vykonaných zmien vrámci testovacej sady. Medzi zaznamenávané typy zmien/udalostí patrí:

- vytvorenie nového testu,
- modifikácia testu,
- zmazanie testu.

Každá udalosť je v súbore uložená na samostatnom riadku vo forme „`i:test:udalosť`“, kde:

- `i` je číslo verzie novej testovacej sady, ktorá touto udalosťou vznikla,
- `test` je názov testu, ktorý bol vytvorený, modifikovaný či odstránený,
- `udalosť` je typ udalosti, ktorý popisuje čo sa dialo s daným testom.

Vďaka uchovávaniu kópii starých verzií testov v adresári *history* a uchovávaniu histórie jednotlivých udalostí v súbore *testsuite_history.txt*, je tak možné spätne obnoviť akúkoľvek verziu testovacej sady.

Vezmime napríklad do úvahy postupnosť udalostí vo výpise 5.13, kde je aktuálna verzia testovacej sady 15. Ak by sme sa chceli dostať spätne na verziu 10, vidíme, že by bolo potrebné vykonať tieto zmeny:

1. odstrániť `test_7` vytvorený vo verzii testovacej sady 11,
2. obnoviť `test_5` s verziou 1 z histórie, ktorý bol zmazaný vo verzii 12,
3. vrátiť test `test_3` na verziu 2 z histórie, ktorý bol modifikovaný vo verzii 13,
4. vrátiť test `test_4` na verziu 1 z histórie, ktorý bol modifikovaný vo verzii 14 a 15.

```
...
9:test_1:modify test (test version 1 -> 2)
10:test_6:create new test
11:test_7:create new test
12:test_5:remove test (version 1)
13:test_3:modify test (test version 2 -> 3)
14:test_4:modify test (test version 1 -> 2)
15:test_4:modify test (test version 2 -> 3)
```

Výpis 5.13: Ukážka zaznamenaných udalostí vrámci histórie testovacej sady

5.12 Výpočet bodového hodnotenia

Základom pre výpočet celkového bodového hodnotenia projektu je súbor *tests/sum*. Tento súbor slúži na definíciu rovnice pre výpočet celkového hodnotenia. Rovnica je v tomto súbore identifikovaná pomocou prefixu `SUM=`. V rovnici je možné používať len operácie pre sčítanie (`+`), odčítanie (`-`) alebo násobenie bodov (`*`). Hodnoty, s ktorými rovnica počíta sa získavajú zo značiek pridaných k študentskému riešeniu. Po spracovaní a vypočítaní rovnice sa vytvorí nová značka `#score`, ktorá obsahuje parameter s číselnou hodnotou výsledku rovnice, čo reprezentuje výsledné bodové hodnotenie projektu. Pokiaľ je výsledná hodnota rovnice väčšia ako je maximálny počet bodov za projekt, rozdiel týchto hodnôt sa uloží ako parameter novej značky `#score_bonus`.

Pri využívaní značiek v definícii rovnice s hodnotením je nutné dodržiavať nasledujúce obmedzenia a pravidlá:

- Je možné využívať len značky s prefixom `scoring_`, pričom nie je nutné tento prefix uvádzať v rovnici. Ak chceme do rovnice zahrnúť napríklad body za dokumentáciu, ktoré sú uložené pomocou značky `#scoring_documentation(4)`, do rovnice sa zadá len výraz `documentation`. Rovnica, ktorá počíta celkové skóre na základe bodov z dokumentácie a na základe výsledkov z automatických testov `testA` a `testB`, by vyzerala nasledovne:

  ```
  SUM= testA + testB + documentation
  ```

- Značky využité v rovnici sa nahradia vždy za hodnotu prvého parametra značky. Je preto nutné, aby si užívateľ sám ustrážil, či do rovnice zadáva značku, ktorá má (alebo bude mať) prvý parameter číselnú hodnotu. Pokiaľ sa v rovnici nachádza značka, ktorý neexistuje (alebo nemá ani jeden parameter), potom sa táto hodnota úplne ignoruje z rovnice.

- Zátvorky v rovnici nie sú podporované. Pre zložitejšie výpočty je potrebné vytvoriť najskôr manuálne značku s medzi-výpočtom, ktorá sa následne použije v rovnici `SUM`. Ako už bolo spomínané, značky sa ukladajú v súboroch formátu YAML na presne určených rovnakých miestach pre každé študentské riešenie. Je teda jednoduché sa k týmto značkám dostať a vytvoriť napríklad skript pre počítanie zložitejšieho medzivýpočtov, ktoré by sa následne automaticky uložili medzi značky a dali by sa tak použiť vo výslednej rovnici `SUM`.

- Okrem značiek je možné v rovnici použiť špeciálnu konštantu `SUM_ALL_TESTS`, ktorá sa v systéme vyhodnotí ako suma (sčítanie) bodových ohodnotení zo všetkých platných testov projektu definovaných v adresári *tests*. Bodové hodnotenia testov sa získavajú zo značiek s prefixom `scoring_`, nasledované názvom testu (napríklad značka `#scoring_test_1(2)`). Tieto značky sa automaticky pridávajú počas spúšťania testov (viď sekcia 5.5). Bodové hodnoty, ktoré sa ukladajú do týchto značiek počas automatických testov, sa získavajú zo súboru *tests/scoring*.

Súbor *tests/scoring* slúži na definíciu bodového hodnotenia pre každý test. Hodnotenie testov je v tomto súbore definované formou shellovského súboru, kde sa vytvárajú premenné, ktorým sa priraďujú bodové hodnoty. Premenné sú zložené vždy z názvu testu a sufixu `_ok` alebo `_fail`. Napríklad priradenie `test_1_ok=2` a `test_1_fail=0` definuje, že ak test s názvom `test_1` nad nejakým študentským riešením prejde úspešne, priradí sa mu značka `scoring_test_1` s hodnotou 2 a ak tento test zlyhá, priradí sa značka s hodnotou 0. Novovytvoreným testom sa v súbore *tests/scoring* automaticky vytvorí premenná s priradenými hodnotami 1 pre úspech a 0 pre zlyhanie testu.

5.13 Generovanie správy s hodnotením projektu

Pre tvorbu výslednej správy hodnotenia projektu sa využíva šablónovací systém Jinja. Výsledná správa je generovaná na základe šablóny uloženej v súbore *reports/report_template.j2* v koreňovom adresári projektu a ukladá sa do adresára študentského riešenia v súbore *reports/total_report*. V šablóne je možné zahrnúť nasledujúce dáta týkajúce sa jedného študentského riešenia:

- výsledky z automatických testov,
- poznámky z code review,
- poznámky k automatickým testom,
- ďalšie nezávislé poznámky k riešeniu.

Okrem týchto dát sú k dispozícii ďalšie informácie o projekte ako: názov projektu, maximálny počet bodov za projekt a dosiahnutý počet bodov študentského riešenia. Ďalej je možné v šablóne zahrnúť akékoľvek značky vzťahujúce sa buď k riešeniu (uložené v súbore *solution_tags.yaml*) alebo k výsledkom testov (uložené v súbore *tests_tags.yaml*). Zoznam spomínaných dát a informácii o projekte, ktoré môže užívateľ zahrnúť do šablóny, je možné zobraziť v režime prechádzania obsahu súboru so šablónou, pomocou klávesy F4.

Výsledky z automatických testov sú spracované zo značiek v súbore *tests_tags.yaml*. V šablóne sú výsledky testov dostupné cez zoznam `test_results`, kde každá položka má atribúty:

- `name`: názov testu,
- `description`: krátky popis testu,
- `result`: stav výsledku testu (prešiel/neprešiel),
- `score`: počet získaných bodov za daný test,
- `note`: poznámka k testu (napríklad spôsob zlyhania).

Poznámky z code review sú v šablóne dostupné cez zoznam `code_review`, kde každá poznámka má atribúty:

- `file`: názov súboru, v ktorom bola poznámka pridaná,
- `row`: riadok, na ktorom bola poznámka pridaná,
- `col`: stĺpec, na ktorom bola poznámka pridaná,
- `text`: text poznámky.

Z poznámok vytvorených vrámci code review v internom editore, je možné generovať samostatnú správu, ktorá obsahuje všetky poznámky zo všetkých súborov jedného študentského riešenia. Táto správa sa ukladá v adresári riešenia v súbore *reports/code_review*.

Počas testovania a hodnotenia projektov môže užívateľ pridávať ďalšie poznámky k riešeniu, ktoré sa rozlišujú na:

- poznámky k automatickým testom,
- nezávislé poznámky k riešeniu.

Poznámku vzťahujúcu sa k automatickým testom je možné pridať v režime prechádzania adresárovej štruktúry niektorého študentského riešenia, cez menu (klávesa F2). Užívateľovi sa tak zobrazí samostatná obrazovka pre zadanie vstupu (text poznámky), ktorý sa následne uloží do súboru *reports/test_notes* v adresári študentského riešenia. Nakoľko sa tieto poznámky vzťahujú k testom (respektíve k celej testovacej sade), vytvorené poznámky sa ukladajú vždy spolu s aktuálnou verziou testovacej sady. Vďaka tomu má užívateľ prehľad

o tom, ktorá poznámka sa vzťahuje ku ktorej verzii testovacej sady. Vo výslednej správe hodnotenia je tak možné zahrnúť len aktuálne poznámky k poslednej verzii testovacej sady. V šablóne sú tieto poznámky dostupné cez zoznam `test_notes`.

Nezávislé poznámky k riešeniu sa pridávajú tak isto v režime prechádzania adresárovej štruktúry, cez menu a ukladajú sa do súboru *reports/user_notes* v adresári študentského riešenia. Užívateľ pomocou týchto poznámok môže pridať napríklad komentár k celkovému dojmu z projektu, k efektivite kódu, štruktúre dokumentácie či akúkoľvek ďalšiu pripomienku k riešeniu. Pri tvorbe šablóny výsledného hodnotenia sú tieto poznámky dostupné v zozname `user_notes`.

5.14 Štatistiky a histogramy

Po testovaní všetkých študentských riešení systém ponúka aj tvorbu štatistík z výsledkov automatických testov. V režime prechádzania adresárovej štruktúry je cez menu možné generovať dva typy štatistík.

Prvá štatistika sa týka celkového bodového hodnotenia študentov a počíta sa na základe hodnôt parametra značky `#score` študentských riešení. Pre jednotlivé hodnoty celkového skóre sa počíta severita vrámci všetkých testovaných študentov. Výsledkom je graf bodového hodnotenia uložený formou textu v súbore *reports/scoring_stats* v koreňovom adresári projektu. Okrem toho sa vrámci tejto štatistiky počíta aj priemerná hodnota získaných bodov za projekt a modus (najčastejšia hodnota).

```
MIT/DP/subjectA/projectC/reports/scoring_stats
Maximum score: 15
-----------------------------------
Average: 8.2
Average (without zero): 9.76
Modus: 15
-----------------------------------
Scoring severity:
0:  **** 4
1:  ** 2
2:   0
3:  **** 4
4:   0
5:   0
6:   0
7:   0
8:  *** 3
9:   0
10: * 1
11:  0
12:  0
13: **** 4
14:  0
15: ******* 7
```

Obr. 5.6: Ukážka štatistík bodového hodnotenia projektu

Na obrázku 5.6 je ukážka generovaných štatistík a bodového hodnotenia projektu. Môžme vidieť, že maximálny počet získaných bodov za projekt je 15, priemerný počet bodov je 8,2 a bez nulových hodnotení je priemer 9,76. Najčastejší počet bodov je 15.

Ďalšie štatistiky, ktoré systém zobrazujú histogram rozloženia výsledkov testov, ktorý sa ukladá v súbore *reports/tests_stats*. Histogram ukazuje pre každý spustený test, koľko bodov za neho študenti získali, teda ako boli hodnotené. Na základe týchto štatistík je možné odhaliť podozrivé testy. Ak za niektorý test dostali všetci študenti plný počet bodov, môže byť tento test príliš jednoduchý a mal by sa upraviť. Naopak ak nejaký test neprešiel ani u jedného študenta, teda všetci za neho dostali nula bodov, je možné, že je v ňom chyba. Početnosť bodov za jednotlivé testy sa počíta na základe značiek, ktoré testy automaticky pridávajú k riešeniam počas testovania.

MIT/DP/subjectA/projectC/reports/tests_stats

Test name	0b	1b	5b
test1	20%	0%	80%
test2	20%	80%	0%
test3	20%	80%	0%
test4	20%	80%	0%
test5	56%	44%	0%
test6	56%	44%	0%
test7	20%	80%	0%
test8	20%	80%	0%
test9	24%	76%	0%
test10	40%	60%	0%
test11	20%	80%	0%
test12	60%	40%	0%
test13	80%	20%	0%
test14	40%	60%	0%
test15	40%	60%	0%
test16	40%	60%	0%
test17	40%	60%	0%
test18	0%	100%	0%

	0b	1b	3b	8b	10b	13b	15b
Total score	16%	8%	16%	12%	4%	16%	28%

Obr. 5.7: Ukážka štatistík bodového hodnotenia projektu

Na obrázku 5.7 je ukážka štatistík z výsledkov testov vo forme tabuľky (textového histogramu). Jednotlivé riadky reprezentujú testy a v stĺpcoch sú pre tieto testy zobrazené percentuálne početnosti ich bodových hodnotení. Na základe týchto informácií môžme vidieť, že posledný test `test18` bol u všetkých študentoch hodnotení jedným bodom, čo môže znamenať, že je tento test triviálny a mal by sa naň vyučujúci pozrieť. Rovnako tak za prvý test `test1` získalo 80% študentov až 5 bodov, čo je v porovnaní s ostatnými testami veľa.

Okrem výsledkov jednotlivých testov je v tejto štatistike zahrnuté aj percentuálne rozloženie celkového hodnotenia študentov. V ukážke 5.7 môžme vidieť, že až 16% študentov získalo z projektu nula bodov a 28% šikovných študentov získalo plný počet bodov.

Kapitola 6

Integračné testovanie systému

Táto kapitola sa zaoberá riešením a výsledkami integračného testovania frameworku. Vrámci testovania vytvoreného systému popísaného v kapitole 5, boli implementované automatické integračné testy v jazyku Python. Celkom bolo navrhnutých a realizovaných 10 komplexných testov, ktoré sú implementované v súbore *testing/run_tests.py*.

Pre každý test sa vytvára samostatný *test fixture* vrámci *setup* fáze (viď kapitola 2). Najskôr sa v každom teste vytvorí samostatný adresár, v ktorom sa pripravia dáta potrebné pre vykonanie testu. Následne sa v tomto adresári spustí testovaný systém so špecifickým vstupom pre daný test. Nakoniec sa overia výsledky a odstráni sa celý adresár. Vstup pre testovaný systém je definovaný ako postupnosť kláves, ktoré ovládajú systém a vykonávajú tak určité akcie.

Spustenie systému (frameworku) vrámci testu sa realizuje pomocou samostatného procesu, ktorý spustí samostatné vlákno s časovým limitom nastaveným na 200 sekúnd. Po vypršaní časového limitu sa proces zastaví a odstráni. V tabuľke 6.1 sú zobrazené hodnoty trvania jednotlivých testov v sekundách, ktoré boli spúšťané na systéme Fedora 35 s procesorom Intel i7-8700k a 32GiB operačnej pamäti.

Id	**Popis**	**Doba trvania [s]**
1	vytvorenie a modifikácia súboru	1.19
2	vytváranie poznámok	1.24
3	založenie projektu a tvorba testov	1.22
4	modifikácia testov (verzovanie a história)	1.23
5	vytvorenie súboru Dockerfile	1.23
6	generovanie záznamov	1.22
7	filtrovanie súborov a ich hromadná správa	1.23
8	vytváranie a mazanie značiek	0.21
9	spúšťanie testov a generovanie reportu	2.22
10	tvorba štatistík	11.26

Tabuľka 6.1: Priemerná doba trvania integračných testov

Následne sú v tejto kapitole stručne popísané jednotlivé testy.

6.1 Vytvorenie a modifikácia súboru

Testovanie vytvorenia nového súboru a jeho modifikácie nie je podmienené existenciou projektu. Jedinou počiatočnou podmienkou pre tento test je, že sa systém musí spúšťať v prázdnom adresári. Vrámci *setup* fáze sa teda vytvorí pre tento test nový adresár *test1*, v ktorom sa systém spustí. Následne po spustení systém na základe vstupných kláves vytvorí nový súbor s názvom *test* a uloží ho. Pomocou tabulátoru sa prejde do režimu úpravy novovytvoreného súboru (nakoľko je jediný súbor v aktuálnom adresári) a zapíše sa do tohto súboru reťazec „`This is\n...test`“, pričom sa pri zápise použije aj klávesa pre mazanie znaku a otestuje sa aj pridávanie nového riadku v súbore. Nakoniec sa upravený súbor uloží a systém skončí. Vo fáze *verify* sa overí existencia nového súboru a jeho obsah a vo fáze *teardown* sa zmaže celý adresár *test1*.

6.2 Vytváranie poznámok

Vrámci tohto testu sa kombinuje prechádzanie adresárovou štruktúrou, prepínanie medzi režimami úpravy súboru a správa poznámok. Systém sa spúšťa v adresári s troma súbormi *test1*, *test2* a *test3*. Obsah súboru s typickými poznámkami je prázdny. Systém sa spustí v pripravenom adresári s troma súbormi a šípkami prejde na súbor *test2*. Tento súbor sa otvorí a v režime prechádzania obsahu súboru systém prejde do režimu správy súboru. Následne pridá špecifickú poznámku s obsahom `note`, na prvý riadok v súbore. Ďalej systém otvorí správu poznámok a v toto režime uloží novovytvorenú poznámku ako typickú a skončí. Výsledkom tejto postupnosti akcií je, že súbor *test2* obsahuje poznámku `note` a táto poznámka je uložená v súbore s typickými poznámkami.

6.3 Založenie projektu a tvorba testov

Jedinou podmienkou tohto testu je prázdny adresár. Systém sa teda spustí v prázdnom adresári, v ktorom pomocou menu vytvorí nový pod-adresár s názvom `proj`. Vrámci prechádzania adresárovej štruktúry vojde do nového adresára a pomocou menu v ňom založí nový projekt. V tom istom adresári rovnako pomocou menu následne vytvorí nový test s názvom `testA` a systém skončí. Na konci tohto testu sa skontroluje existencia všetkých súborov a adresárov, ktoré sa podľa špecifikácie majú vytvoriť založením nového projektu (viď 5.3.1). Rovnako tak sa skontroluje správne vytvorenie nového testu s počiatočnou verziou jedna.

6.4 Modifikácia testov (verzovanie a história)

Systém sa spúšťa v prázdnom adresári, v ktorom okamžite založí nový projekt a vytvorí dve nové testy bez špecifikovaných názvov (názov sa vygeneruje automaticky pre oba testy). Následne pomocou menu zvolí možnosť modifikovať prvý test, do ktorého zapíše reťazec „`echo test`“ a uloží novú verziu testu, a skončí. Výsledkom týchto akcií by mali byť jednak vytvorené adresáre a súbory nového projektu, a vytvorené dva nové testy. Prvý test obsahuje zadaný reťazec a má zvýšenú verziu dva. Druhý test nebol upravený a mal by mať verziu jedna. Verzia celej testovacej sady sa zvýšila trikrát (dvakrát za vytvorený nový test a jedenkrát za modifikáciu testu). Okrem toho sa pôvodná verzia prvého testu uložila do

adresára s históriou a všetky udalosti týkajúce sa testovacej sady sa zaznamenali do súboru *history/testsuite_history.txt*, ktorý by mal obsahovať:

```
2:test_1:create new test
3:test_2:create new test
4:test_1:modify test (test version 1 -> 2)
```

6.5 Vytvorenie súboru Dockerfile

V prázdnom adresári systém po spustení testu založí hneď nový projekt. Následne cez menu zvolí možnosť vytvoriť súbor Dockerfile. Systém zadá požadované vstupy pre distribúciu `fedora:34`, pre identifikátor užívateľa `1000` a identifikátor skupiny `1000`, a skončí. Výsledkom by mal byť vytvorený súbor Dockerfile s obsahom:

```
FROM fedora:34
RUN addgroup -g 1000 test 2>/dev/null || groupadd -g 1000 test
RUN adduser -D -u 1000 -G test test 2>/dev/null || useradd -u 1000 -g test
test
USER test
```

6.6 Generovanie záznamov

Tento test je zameraný na vytváranie záznamov počas vykonávaných akcii so systémom. Systém po spustení teda ako prvé premaže súbor so záznamami. Následne v prázdnom adresári založí nový projekt a pomocou menu zvolí možnosť rozbaliť všetky študentské archívy. Nový projekt neobsahuje žiadne študentské archívy, preto by táto akcia mala vytvoriť záznam s varovaním. Ďalej systém vytvorí nový test bez špecifikovaného názvu, tento test hneď modifikuje a uloží pôvodnú verziu do histórie. Následne systém v adresári projektu vytvorí nový adresár s názvom `xlogin00` a spustí nad týmto adresárom (študentským riešením) vytvorený test. Nakoniec tento test odstráni, čím sa znovu uloží do histórie a systém skončí. Výsledkom všetkých týchto akcii by mal byť súbor obsahujúci nasledujúce záznamy:

```
... |INFO   |new project created (test6)
... |INFO   |extracting all solution archives...
... |WARNING|no solution archives found in project directory
... |INFO   |creating new test...
... |INFO   |test 'test_1' created (scoring ok=1,fail=0)
... |INFO   |file 'dotest.sh' saved
... |INFO   |old version (1) of test 'test_1' is saved in history
... |INFO   |*** testing student 'xlogin00' ***
... |ERROR  |FUT 'sut' doesnt exists in solution directory
... |INFO   |testing done
... |INFO   |testing all students done !!
... |INFO   |removing test 'test_1'...
... |INFO   |test 'test_1' (version: 2) is archived in history
```

Na začiatku každého záznamu je namiesto reťazca `...` dátum a čas vytvorenia záznamu. Tento údaj sa môže meniť a je pre účely tohto testu irelevantný, preto sa vrámci kontroly súboru so záznamami ignoruje a kontroluje sa len fixná časť záznamov.

6.7 Filtrovanie súborov a ich hromadná správa

Pre testovanie správneho filtrovania súborov je potrebné mať k dispozícii viacero súborov v adresári. Počiatočným stavom je teda adresár so súbormi v pod-adresároch:

- *xlogin00/file1*,
- *xlogin01/file2* (obsahuje reťazec `test`),
- *xlogin02/file3*,
- *xlogin03/test1*,
- *xlogin04/test2* (obsahuje reťazec `test`),
- *xlogin05/tmp1*,
- *xlogin06/tmp2* (obsahuje reťazec `test`).

Systém po spustení najskôr v hlavnom adresári vytvorí projekt. Následne v režime prechádzania adresárovej štruktúry zadá filter podľa cesty „`*e`“ a prepne sa do režimu prechádzania obsahu súboru, kde zadá filter podľa obsahu „`test`“. Nakoniec filtrované súbory agreguje, pridá k nim nezávislú poznámku s obsahom `ok` a skončí. Jediné súbory odpovedajúce obom zadaným filtrom sú *xlogin01/file2* a *xlogin04/test2*, teda len adresáre *xlogin01* a *xlogin04* budú obsahovať zadanú poznámku.

6.8 Vytváranie a mazanie značiek

Systém sa vrámci tohto testu spúšťa v adresári s dvoma pripravenými pod-adresármi *xlogin00* a *xlogin01*. Najskôr sa po spustení systému založí nový projekt. Následne sa v adresárovej štruktúre prejde na adresár *xlogin00* a prepne sa na režim správy značiek. Vytvorí sa nová značka `#test` a prejde sa na adresár *xlogin01*, ktorému sa pridajú dve značky `#test1` a `#test2`. Následne systém znovu prejde na správu značiek pre *xlogin00* a zmení parameter značky `#test` na hodnotu 2. Nakoniec systém odstráni značku `#test2` z *xlogin01* a skončí. Vo výsledku by mal adresár *xlogin00* obsahovať jednu značku `#test` s parametrom 2 a adresár *xlogin01* tak isto jednu značku `#test1` bez parametra.

6.9 Spúšťanie testov a generovanie reportu

Počas prípravnej fázy tohto testu sa vytvoril v adresári ďalší pod-adresár *xlogin00* so súborom *sut*. Systém v hlavnom adresári založí nový projekt a vytvorí nový test `test_1`, ktorý bude automaticky pridávať značky `#scoring_test_1(2)` a `#test_1_ok()`. To znamená, že tento test vždy prejde a pridá študentom k hodnoteniu dva body. Následne sa vytvorí testovacia stratégia, ktorá spustí vytvorený test `test_1`. Systém pridá k pripravenému adresáru s riešením *xlogin00* samostatnú nezávislú poznámku s textom `ok` a vrámci code review súboru *sut* pridá poznámku `note` na prvom riadku súboru. Ďalej systém spustí testovaciu sadu nad *xlogin00*, vygeneruje výsledný report a skončí. Výsledný report sa vytvára zo šablóny, ktorá po založení nového projektu nebola zmenená. Adresár *xlogin00* by mal na konci tohto testu obsahovať nezávislú poznámku `ok`, poznámku z code review `note` a značku s bodovým hodnotením testu. Generovaný report by teda podľa šablóny mal vyzerať nasledovne:

```
Scoring for project: project
Total score: 2 / 10
Bonus score: -
================================
2:ok:test_1:-
================================


Code review
================================
xlogin00/sut:1:0:note
================================


Additional notes
================================
ok
================================
```

6.10 Tvorba štatistík

Pre otestovanie správnej tvorby štatistík je potrebné vytvoriť v počiatočnom adresári testu niekoľko pod-adresárov reprezentujúcich študentské riešenia. Pre tento test boli teda vytvorené nasledujúce pod-adresáre a súbory s riešením:

- *xlogin00/sut* (obsahuje reťazec `test`),
- *xlogin01/sut* (obsahuje reťazec `txt`),
- *xlogin02/sut* (obsahuje reťazec `abc`),
- *xlogin03/sut* (obsahuje reťazec `def`),
- *xlogin04/sut* (obsahuje reťazec `ahoj`),
- *xlogin05/sut* (obsahuje reťazec `test`).

Systém po spustení založí nový projekt a vytvorí nový test, ktorý pridá k riešeniu značky `#scoring_test_1(0)` a `#test_1_ok()` (podobne ako predošlý test). Vytvorí sa testovacia stratégia, ktorá spustí vytvorený test `test_1`. Následne systém zadá filter podľa obsahu súboru s hľadaným reťazcom „`a`“ a výsledným riešeniam pridá značku `#scoring_bonus(3)`. Potom zmení filter podľa obsahu na reťazec „`e`“ a výsledným riešeniam pridá značku `#scoring_bonus(9)`. Ďalej tieto vytvorené značky pridá do rovnice pre výpočet celkového hodnotenia a spustí testovaciu sadu nad všetkými riešeniami. Nakoniec sa prepočíta skóre všetkých riešení, vygenerujú sa štatistiky a systém skončí. Výsledná štatistika ukáže, že:

- jedno riešenie dosiahlo celkové hodnotenie 0 bodov (len za vykonaný test `test_1`),
- dve riešenia dosiahli bodové hodnotenie 3 (pridané body za obsahujúci reťazec „`a`“),
- tri riešenia dosiahli bodové hodnotenie 9 (pridané body za obsahujúci reťazec „`e`“).

Kapitola 7

Možné rozšírenia

V tejto krátkej kapitole sú stručne popísané možné rozšírenia vytvoreného frameworku pre testovanie a hodnotenie študentských projektov. Väčšina týchto rozšírení sa týka len drobných úprav implementácie. V nasledujúcich bodov sú zhrnuté niektoré možné rozšírenia systému:

- Zobrazenie obsahu archívu bez jeho rozbalenia. Pri prechádzaní adresárovej štruktúry by sa nájdením na archív zobrazil v náhľade jeho obsah.
- Zvýraznenie adresára, ktorý je platným koreňovým adresárom nejakého projektu. Táto informácia sa dá zistiť podľa toho, či adresár obsahuje konfiguračný súbor projektu.
- Zvýraznenie riadku súboru, na ktorom je vykonaná (neuložená) zmena, čo by sa dalo realizovať podobne ako u zvýraznenia riadku s poznámkou.
- Zobrazenie aktuálnej pozície kurzoru (riadok:stĺpec) pri prechádzaní súboru (v dolnom pravom rohu v obrazovke s obsahom súboru).
- Dynamické čítanie súboru po riadkoch len pre zobrazovanú časť súboru. Súbor by sa tak nemusel načítať celý, v prípade, že sa zobrazuje len nejaká jeho časť (pár riadkov).
- Sledovanie modifikácie otvoreného súboru mimo framework (*linux file watcher*) a prípadná aktualizácia modifikovaného obsahu súboru. Toto rozšírenie by vyriešilo problém situácie kedy má užívateľ pomocou frameworku otvorený súbor v internom editore, pridal v ňom poznámku na riadok X a následne ten istý súbor modifikoval v inom editore. Framework o tejto skutočnosti nevie a pridanú poznámku tak neposunie na správny riadok, ku ktorému bola pridaná.
- Možnosť definície rovnice pre výpočet bonusových bodov. Momentálne sa počítajú bonusové body len v prípade, ak celková suma bodov prekročí maximálny počet bodov za projekt (rozdiel týchto hodnôt je bonus).
- Vytvoriť funkciu v menu, ktorá pridá bodové hodnotenie (tj. vytvorí značku s prefixom `#scoring_`, pridá hodnotenie do rovnice pre výpočet celkového bodového hodnotenia a pridá toho hodnotenie do šablóny výslednej správy s hodnotením).
- Možnosť uložiť často pridávané hodnotenie ako typické a neskôr ho pridať priamo jednou klávesovou skratkou (podobne ako typické poznámky).

- Komplexné filtrovanie podľa rôznych kritérií na jednom mieste. V samostatnej obrazovke by bolo možné zadať kombináciu filtrov (podľa cesty, obsahu či značky) s využitím logických spojok.
- Pri modifikácii testovacej sady upozorniť na študentské riešenia testované starými verziami testov.
- Automatické obnovenie starej verzie testu z histórie.
- Zobraziť zoznam testov, ktoré boli modifikované, pridané či odstránené od určitej verzie testovacej sady.
- Automatické posielanie e-mailov s výsledkami hodnotenia projektu a zadávanie bodov do informačného systému školy.
- Vytvoriť funkciu v menu pre zmenu ovládania frameworku.
- Vytvoriť funkciu v menu pre zmenu konfigurácie frameworku.
- Možnosť uložiť posledný stav prostredia frameworku do konfigurácie a znovu ho načítať v tomto stave.

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo navrhnúť a implementovať systém, ktorý zjednoduší a zefektívni prácu pri testovaní študentských projektov. V práci som sa na jednej strane venovala analýze procesu testovania softvéru zo všeobecnejšieho hľadiska a na druhej strane som analyzovala požiadavky na testovanie študentských projektov z pohľadu vyučujúcich.

V teoretickej časti som priblížila základy overovania softvéru so zameraním na rôzne metódy testovania najmä v oblasti integračných testov a testov jednotiek. Zaoberala som sa pojmom vzoru pevného okolia testovanej jednotky a venovala som sa aj problematike izolácie testov s využitím technológie linuxových kontajnerov.

V druhej časti práce som špecifikovala požiadavky na testovanie a hodnotenie študentských projektov v rôznych oblastí, na základe ktorých som navrhla testovací framework. Navrhnutý systém sa mi podarilo implementovať s využitím technológie linuxových kontajnerov a poskytnúť tak možnosť testovania projektov bezpečne v izolovanom prostredí. Vytvoreným systémom som poskytla vyučujúcim podporu aj pri revízií zdrojových kódov študentov a pri vytváraní poznámok do výsledného hodnotenia projektu. Pre overenie správneho fungovania systému som vytvorila desať komplexných integračných testov.

Do budúcna by som chcela výsledný systém rozšíriť minimálne o navrhnuté možnosti rozšírenia v tejto práci a postupne ho tak zdokonalovať. Prípadne by sa dalo ďalej pracovať na integrovaní funkcionality pre automatické rozpoznávanie plagiátorských projektov, na základe archívu starších odovzdaných projektov či iných voľne dostupných zdrojových kódov.

Literatúra

[1] *Oficiálna dokumentácia projektu Docker* [online]. Docker, Inc. [cit. 2022-01-7]. Dostupné z: https://docs.docker.com/.

[2] Beneš, E. *Advanced Linux Administration - Linux Containers* [online]. Neverejne dostupné v informačnom systéme FIT VUT v Brne, 2021 [cit. 2022-01-7].

[3] Glenford J. Myers, T. B. *The Art of Software Testing.* 3. vyd. Wiley Publishing, 2011. ISBN 978-1-118-03196-4.

[4] Jain, S. M. *Linux Containers and Virtualization: A Kernel Perspective.* Apress, 2020. ISBN 978-1-4842-6282-5.

[5] Khorikov, V. *Unit Testing Principles, Practices, and Patterns.* Manning Publications, 2020. ISBN 978-1-6172-9627-7.

[6] Meszaros, G. *XUnit Test Patterns: Refactoring Test Code.* Addison-Wesley Professional, 2007. ISBN 978-0-1314-9505-0.

[7] Patton, R. *Software Testing.* 2. vyd. Sams, 2005. ISBN 978-0-6723-2798-8.

[8] Paul Ammann, J. O. *Introduction to Software Testing.* Cambridge University Press, 2008. ISBN 978-0-511-39330-3.

[9] Rosen, R. *Linux Containers and the Future Cloud* [online]. Linux Journal, jún 2014 [cit. 2022-01-7]. Dostupné z: https://www.linuxjournal.com/content/linux-containers-and-future-cloud.

[10] Schenker, G. N. *Learn Docker - Fundamentals of Docker 18.x.* Packt Publishing, 2018. ISBN 978-1-78899-702-7.

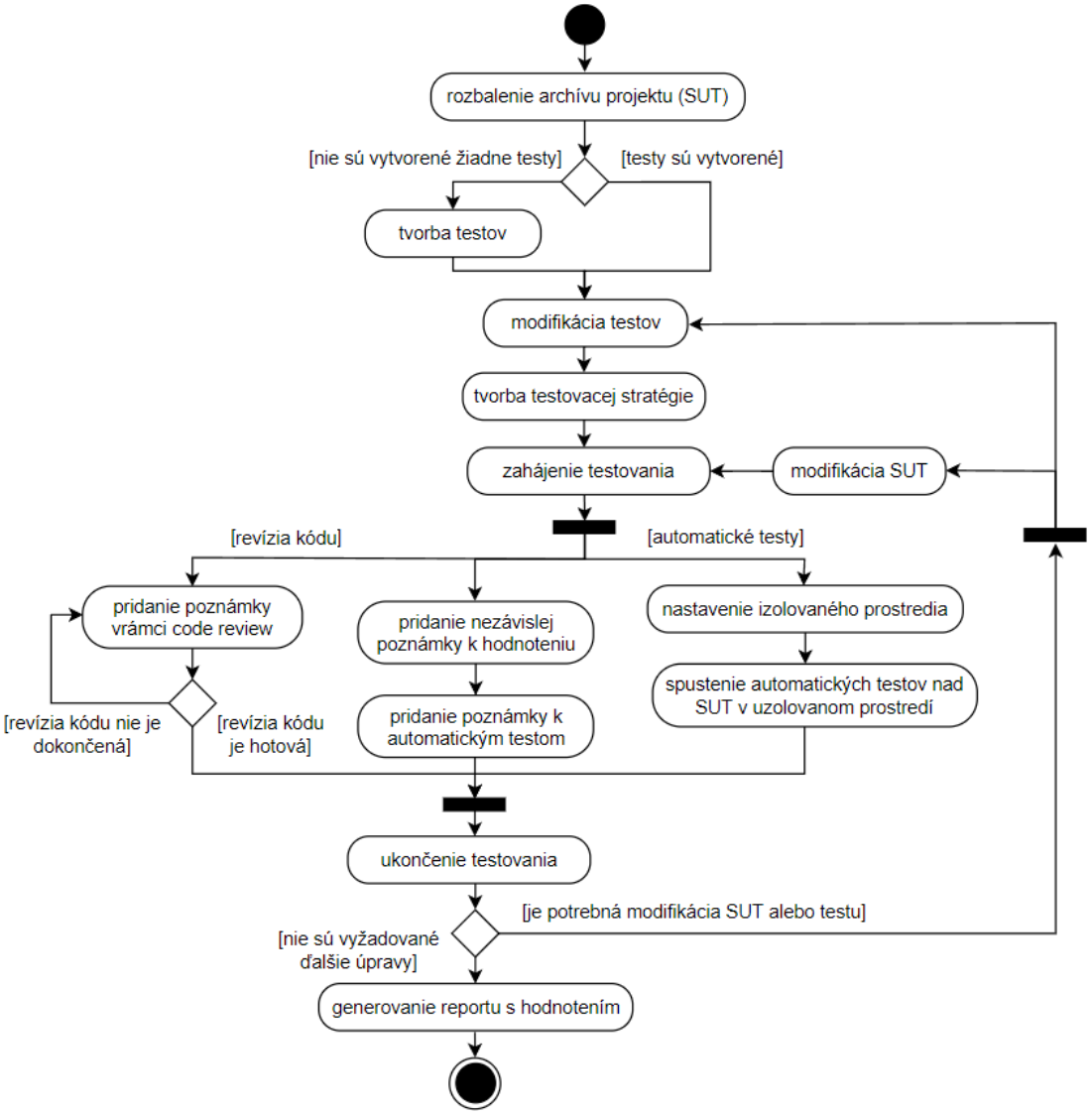
[11] Smrčka, A. *Testování a dynamická analýza - Úvod a pojmy v testování* [online]. Neverejne dostupné v informačnom systéme FIT VUT v Brne, 2020 [cit. 2022-01-3].

[12] Soyinka, W. *Linux Administration: A Beginner's Guide.* 8. vyd. McGraw-Hill, 2020. ISBN 978-1-26-044171-0.

[13] Srinivasan Desikan, G. R. *Software Testing: Principles and Practices.* Pearson India, 2006. ISBN 978-8-1775-8121-8.

Príloha A

Diagram aktivít procesu testovania



Obr. A.1: Diagram aktivít procesu testovania študentských projektov