Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Generování akceptačních testů

Plzeň, 2021 Jitka Poubová

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jitka POUBOVÁ**
Osobní číslo:	**A19N0043P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Generování akceptačních testů**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s projektem TbUIS a relevantními kvalifikačními pracemi na tomto projektu. Dále se seznamte s nástrojem Robot framework, zejména s jeho používaným formátem klíčových slov a testovacích případů.
2. Analyzujte dostupné informace z báze znalostí projektu. Na základě analýzy navrhněte metodiku, jak automatizovaně generovat akceptační testy aplikace UIS pro Robot framework. V případě nedostačujících dat navrhněte takovou sadu informací, která generování umožní.
3. Navrženou metodiku realizujte v programovacím jazyce Java.
4. Ověřte na bezporuchovém klonu aplikace a vhodně zvoleném vzorku poruchových klonů, že generované testy poskytují srovnatelné výsledky s již existujícími akceptačními testy.
5. Získané výsledky diskutujte a zveřejněte na webových stránkách projektu.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Pavel Herout, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **11. září 2020**
Termín odevzdání diplomové práce: **20. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 24. září 2020

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 19. května 2021

Jitka Poubová

Poděkování

Ráda bych poděkovala panu doc. Ing. Pavlu Heroutovi, Ph.D., za cenné připomínky a dobré rady během vypracovávání této práce.

Abstract

This thesis is focused on experimenting with automatized generation of acceptance tests. As System Under Test (SUT) is used *Testbed University Information System* (TbUIS) project which has been made especially for the purpose of development and verification of new testing methods. Master thesis follows thesis *Acceptance testing in project TbUIS* in which a set of manually written acceptance tests for Robot Framework has been created. This thesis' content is composed of automatized generation of acceptance tests for Robot Framework and then comparison of generated tests' results with results of manually written set of acceptance tests. The aim of this thesis is to experimentally verify on a real application if it is possible to automatically generate set of acceptance tests of comparable quality.

Abstrakt

Práce se zabývá experimentováním v oblasti automatizovaného generování akceptačních testů. Jako System Under Test (SUT) je využíván projekt *Testbed University Information System* (TbUIS), který byl vytvořen právě za účelem vyvíjení a ověřování nových testovacích metod. Diplomová práce navazuje na práci *Akceptační testování v projektu TbUIS*, ve které byla manuálně napsána sada akceptačních testů pro nástroj Robot Framework. Náplní této práce je automatizovaně generovat akceptační testy pro nástroj Robot Framework a následně porovnat jejich výsledky s výsledky již manuálně vytvořené sady akceptačních testů. Cílem práce je na reálné aplikaci experimentálně ověřit, zda je možné vygenerovat srovnatelně kvalitní sadu akceptačních testů automaticky.

Obsah

1 Úvod

Na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni (KIV ZČU) je řešen výzkumný projekt *Testbed University Information System* (TbUIS). Hlavním cílem tohoto projektu je výzkum nových testovacích metod a postupů. Projekt je aktivní od roku 2017 a v současné době představuje vyspělý ucelený systém, se kterým lze provádět různé experimenty v oblasti testování.

V akademickém roce 2019/20 byla na KIV ZČU úspěšně obhájena diplomová práce *Akceptační testování v projektu TbUIS* [9], v rámci které byla manuálně připravena (kódována) sada akceptačních testů pro nástroj Robot Framework. Díky této práci je k dispozici značné množství zkušeností a také publikované soubory ucelených výsledků, na kterých je možné založit další výzkum.

Cílem této práce je využít rozsáhlý projekt TbUIS společně s poznatky z výše zmíněné diplomové práce a prozkoumat možnost, jestli je možné akceptační testy pro Robot Framework generovat automaticky. Předpokladem je rozsáhlá báze znalostí projektu TbUIS, avšak v případě nedostačujících informací je cílem práce také rozšířit tuto bázi znalostí tak, aby bylo generování testů možné. Druhým cílem této práce je také posouzení kvality vygenerovaných akceptačních testů a jejich porovnání s již existujícími manuálně napsanými akceptačními testy.

Výsledky práce by měly být studií proveditelnosti generování akceptačních testů z různých textových podkladů (analytických, dokumentačních apod.) [2]. Druhým výsledkem by měl být seznam doporučení, jak zdokonalit přípravu zmíněných textových podkladů tak, aby bylo možné generování testů zčásti nebo úplně zautomatizovat.

2 Projekt TbUIS a používané nástroje

2.1 TbUIS

2.1.1 Základní informace

Projekt *Testbed University Information System* (TbUIS) se skládá z několika softwarových produktů, které dohromady jako celek tvoří kompletní řešení pro použití v oblasti experimentování a ověřování nových testovacích přístupů a metod [8]. Součástí TbUIS jsou tyto produkty:

- *University Information System* (UIS) – bezporuchová webová aplikace, jejíž součástí jsou také dokumentující soubory (případy užití, požadavky a testovací případy),
- poruchové klony UIS,
- jednotkové testy (součástí aplikace UIS),
- funkcionální testy (nezávisle vytvořené),
- akceptační testy (nezávisle vytvořené),
- knihovna *support* – podpůrná knihovna sloužící k testování UIS,
- *ErrorSeeder* – nástroj pro injektování poruch přímo do zdrojového kódu (slouží k vytváření poruchových klonů).

Projekt TbUIS díky nástroji *ErrorSeeder* podporuje přístup umělého vkládání chyb (*defect injection*) a vytváření poruchových klonů webové aplikace UIS. Tento projekt je open-source a může být použit pro ověření jakékoliv testovací techniky [3].

Detailní informace ohledně testované aplikace UIS jsem již popisovala ve své bakalářské práci *Generování automatizovaných funkcionálních testů* [4].

2.1.2 Knihovna *support*

Podpůrné třídy a metody byly nejprve jen součástí sady funkcionálních testů. Pro zvýšení přehlednosti a také za účelem snazší integrace (například do jiných experimentálních testovacích projektů) byly následně tyto podpůrné

třídy a metody odděleny do samostatného projektu. Tento projekt se nyní nazývá knihovna *support*. Tato knihovna obsahuje metody pro ovládání aplikace UIS pomocí nástroje Selenium a může být tedy využívána různými typy testů. Díky tomu se testy nemusí zabývat samotným ovládáním aplikace, stačí jim jen použít knihovnu *support*.

Součástí knihovny *support* je také kompletní testovací orákulum (*test oracle*). Aby totiž bylo možné rozhodovat o tom, jestli test prošel nebo selhal, je nutné mít k dispozici informace o očekávaném chování systému a způsobu, jak určit jeho správnost. Jako testovací orákulum je označován právě zdroj těchto informací [7]. Toto orákulum, které je součástí knihovny *support*, je možné využít v automatizovaných testech pro stanovení výsledku testu.

2.1.3 Sada akceptačních testů UIS

Na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni (KIV ZČU) byla v roce 2020 úspěšně obhájena diplomová práce *Akceptační testování v projektu TbUIS*. V rámci zmíněné diplomové práce byla vytvořena sada akceptačních testů s využitím nástroje Robot Framework. Celkem bylo vytvořeno 122 akceptačních testů (respektive testovacích případů pro Robot Framework), přičemž 98 z nich testuje základní funkcionalitu (dle specifikace případů užití) a zbylých 24 se zaměřuje na scénáře (speciální případy užití) [9].

Součástí diplomové práce bylo také vytvoření aplikace *WrapperGenerator*, která na základě knihovny *support* dokáže generovat metody (wrappery), které obalují metody z balíku `uis.support.services` této knihovny. Tyto wrappery nepoužívají jako parametry objekty z knihovny, ale pouze řetězce (například místo objektu třídy `Student` se předává řetězec s přihlašovacím jménem studenta). Dále přes vytvořený validátor entit (třída `EntityValidator` aplikace *WrapperGenerator*) se ověří, jestli je možné z daného řetězce vytvořit entitu, a pokud ano, je s takto vytvořenou entitou volána metoda z knihovny *support*. Metody z knihovny *support* tedy nejsou nikdy volány z akceptačních testů přímo, vždy jsou volány přes tyto vygenerované wrappery, aby byl pokaždé zajištěný průchod přes validátor entit.

Příklad souboru se zdrojovým kódem manuálně napsaných akceptačních testů v rámci zmíněné diplomové práce:

```
1 *** Settings ***
2
3 Resource          ../../baseKeywords.robot
4 Resource          ../../keywords/scenarios/exams/
     cancel/one.robot
```

```
5
6  Variables          ../data/student/UC07_CancelExam.
       yaml
7
8  Suite Setup        Prepare Suite without DB restore
9  Test Setup         Prepare Test with DB restore
10 Suite Teardown     End Test
11
12 *** Test Cases ***
13
14 TC.07.01 Student Cancel Exam Date
15     [Template]           Student Cancel Exam Date
16     [Tags]               UC.07  HappyDay  MAJOR
17     [Documentation]      In this test case student
           cancels registered Exam Date
18     ...
19     FOR    ${params}    IN    @{SuccessCancel}
20        @{params}
21     END
22
23
24 TC.07.02 Student Cancel Exam Date Failed
25     [Template]           Student Cancel Exam Date
           Should Fail
26     [Tags]               UC.07  HappyDay  MINOR
27     [Documentation]      In this test case student
           cancels registered Exam Date
28     ...
29     FOR    ${params}    IN    @{FailedCancel}
30        @{params}
31     END
```

Příklad odpovídajících vstupních dat, která jsou využívána ve výše uvedeném příkladu s akceptačními testy:

```
1  SuccessCancel:
2    -
3      - maroon
4      - Computation Structures
5    -
6      - green
```

```
7       - Computer System Engineering
8     -
9       - brown
10      - Programming in Java
11    -
12      - pink
13      - Mobile Applications
14
15  FailedCancel:
16    -
17      - blue
18      - Computation Structures
19    -
20      - pink
21      - Computer System Engineering
22    -
23      - blue
24      - Programming in Java
25    -
26      - red
27      - Programming in Java
28    -
29      - magenta
30      - Linear Algebra
```

2.1.4 Generátor klíčových slov

Aplikaci pro generování klíčových slov jsem vytvořila v rámci předmětu Oborový projekt (KIV/OPSWI) v prvním ročníku navazujícího studia. Tento projekt byl koncipován jako příprava na samotnou diplomovou práci. Zadáním bylo zanalyzovat stávající sadu akceptačních testů, popsanou v sekci 2.1.3, a poté navrhnout a vytvořit aplikaci, která by generovala klíčová slova pro Robot Framework.

Analýzou bylo zjištěno, že ručně psaná klíčová slova pro sadu akceptačních testů jen obalují metody, které generuje *WrapperGenerator*. Protože už existuje aplikace *WrapperGenerator*, bylo navrženo, že klíčová slova budou generována spolu s wrappery právě pomocí této aplikace. Stávající aplikace už má implementováno procházení tříd z knihovny *support* a předpřipravené informace o těchto třídách a jejich metodách, bylo by tedy zbytečné im-

plementovat novou aplikaci, která by stejným způsobem zpracovávala knihovnu *support*. Při procházení knihovny *support* se na základě každé třídy (souboru) z balíku `uis.support.services` této knihovny vygeneruje jeden soubor s wrappery a jeden soubor s klíčovými slovy.

Samostatná aplikace Generátor klíčových slov nikdy nevznikla. Pokud tedy bude dále zmíněna aplikace Generátor klíčových slov, bude se jednat o *WrapperGenerator*, který byl rozšířen o funkcionalitu generování klíčových slov.

Každý vygenerovaný soubor s klíčovými slovy obsahuje dvě části:

- část `Settings`, ve které je deklarována použitá knihovna (respektive wrapper),
- část `Keywords`, kde jsou implementována jednotlivá klíčová slova.

Implementace klíčového slova spočívá v deklarování argumentů, jejich zalogování a nakonec zavolání daného wrapperu s těmito argumenty.

Příklad vygenerovaného klíčového slova:

```
1  *** Settings ***
2
3  Library           uis.support.services.student.
      m3_othersubjects.
      Stu_OtherSubjects_Actions_Wrapper  WITH NAME
      Stu_OtherSubjects_Actions
4
5  *** Keywords ***
6
7  Student Enroll Subject
8      [Arguments]        ${arg0}  ${arg1}
9      Log                ${arg0}
10     Log                ${arg1}
11     Stu_OtherSubjects_Actions.Enroll Subject
          ${arg0}  ${arg1}
```

Poznámka: Takto vypadala vygenerovaná klíčová slova, když byl projekt v rámci předmětu KIV/OPSWI dokončen. Během vypracovávání této diplomové práce ale došlo k optimalizaci generování klíčových slov a jejich kód je nyní odlišný. Více je optimalizace (včetně příkladu finální podoby klíčového slova) popsána v sekci 5.2.

2.2 Robot Framework

2.2.1 Základní informace

Nástroj Robot Framework (RF) je generický open-source framework, který je možné použít k automatizaci testování nebo k robotické automatizaci procesů (*Robotic Process Automation* – RPA). Je to otevřený a rozšiřitelný nástroj a může být integrován s jakýmkoli dalším nástrojem za účelem vytvoření sofistikovaného automatizovaného řešení. Zápis testovacích případů v Robot Frameworku je díky své jednoduché syntaxi snadno použitelný a čitelný i pro člověka.

Robot Framework je nezávislý na operačním systému i aplikaci. Jádro frameworku je implementováno v jazyce Python, ale je spustitelné také v *Java Virtual Machine* (JVM) pomocí Jythonu nebo v .NET pomocí IronPythonu [5].

2.2.2 Formáty souborů

Robot Framework definuje několik formátů, které je možné používat při psaní zdrojového kódu. Při zpracovávání souboru Robot Framework jeho obsah nejprve rozdělí na řádky a poté řádky rozdělí na jednotlivé tokeny (jako například klíčová slova nebo argumenty). Robot Framework umožňuje používat tyto formáty souborů:

- formát dělený mezerami (*space separated format*) – jednotlivé tokeny musí být odděleny minimálně dvěma mezerami (nebo minimálně jedním tabulátorem),
- formát dělený svislou čarou (*pipe separated format*) – jednotlivé tokeny musí být odděleny symbolem svislé čáry (|) s mezerou před i za tímto symbolem.

Jeden Robot Framework soubor může kombinovat oba formáty, respektive jednotlivé řádky souboru mohou používat libovolný z těchto dvou formátů. Oba tyto formáty dělí tokeny v souboru do sloupců (pomyslných nebo díky možnosti více mezer/tabulátorů i opravdových), přičemž při dalším psaní v Robot Frameworku je nutné dodržovat, co má být uvedeno v jakém sloupci [6].

2.2.3 Klíčová slova

Klíčové slovo (*keyword*) v Robot Frameworku sdružuje jednotlivé kroky testovacího případu, respektive se jedná o funkci, která je volána buď z jiného

klíčového slova nebo přímo z testovacího případu. V Robot Frameworku se rozlišují dva typy klíčových slov:

- uživatelská klíčová slova (*user keywords*) – vysokoúrovňová klíčová slova, která jsou implementována v rámci vytváření testů a většinou kombinují dohromady více klíčových slov,
- knihovní klíčová slova (*library keywords*) – nízkoúrovňová klíčová slova, která jsou implementována uvnitř použitých knihoven (jedná se o způsob volání knihovních funkcí z Robot Frameworku).

Název daného klíčového slova se píše do prvního sloupce. Do druhého sloupce se píší názvy ostatních (volaných) klíčových slov, ovšem pokud volané klíčové slovo vrací nějakou hodnotu, jeho název se píše až do následného (tedy třetího) sloupce. Pokud je potřeba zavolat klíčové slovo s argumenty, tyto argumenty se píší za název klíčového slova (do následných sloupců).

Příklad syntaxe klíčových slov (zdroj: [6]):

```
*** Keywords ***
Open Login Page
    Open Browser    http://host/login.html
    Title Should Be    Login Page

Title Should Start With
    [Arguments]    ${expected}
    ${title} =    Get Title
    Should Start With    ${title}    ${expected}
```

2.2.4 Testovací případy

Testovací případy jsou tvořeny z dostupných klíčových slov, která mohou být importována buď z používaných knihoven nebo ze souborů se zdroji (*resource files*). Případně mohou být klíčová slova vytvořena přímo v samotném souboru s testovacími případy.

Syntaxe testovacích případů je v mnoha ohledech identická jako syntaxe klíčových slov. Rozdíl je v názvu, který sekci testovacích případů uvozuje (`*** Test Cases ***`) a který se píše do prvního sloupce. Názvy testovacích případů se také píší do prvního sloupce. Rozdělení do sloupců v kódu testovacího případu se řídí stejnými pravidly jako psaní kódu klíčového slova (více popsáno v předchozí části 2.2.3). Testovací případ začíná řádkou se

svým názvem a končí buď začátkem dalšího testovacího případu (jeho názvem) nebo na konci sekce s testovacími případy.

Příklad testovacího případu (zdroj: [6]):

```
*** Test Cases ***
Valid Login
    Open Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open

Setting Variables
    Do Something    first arg    second arg
    ${value} =    Get Some Value
    Should Be Equal    ${value}    Expected value
```

Testovací případ ale nemusí být vždy popsán jen výše zmíněným způsobem. Je možné totiž využít šablony (*template*), které Robot Framework podporuje. Testovací šablony mění dosavadní *keyword-driven* testy na *data-driven* testy. U *keyword-driven* testů, které jsou popsány výše, se těla těchto testů skládají z klíčových slov a jejich argumentů. Naproti tomu v případě *data-driven* testů se šablonou obsahují tyto testy pouze danou šablonu a argumenty pro ni. Díky šablonám je tedy snazší testu předat celou sadu vstupních dat.

Příklady obou typů testovacích případů (zdroj: [6]):

```
*** Test Cases **
Normal test case
    Example keyword    first arg    second arg

Templated test case
    [Template]    Example keyword
    first arg    second arg
```

Někdy je zapotřebí předat testovacímu případu více vstupních dat (běžně celou sadu dat). Za tímto účelem je možné použít šablonu v kombinaci s *for* cyklem. V případě manuálně psané sady akceptačních testů, která byla popsána v sekci 2.1.3, byl použit pro psaní kódu akceptačního testu právě přístup se šablonou (*template*) a *for* cyklem.

Příklad použití šablony a *for* cyklu (zdroj: [6]):

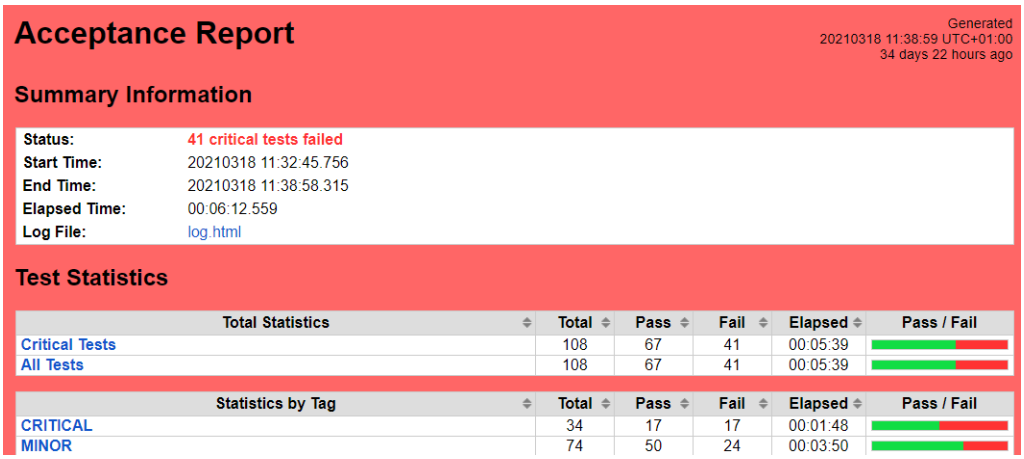
```
Template with for
    [Template]    Example keyword
    FOR    ${item}    IN    @{ITEMS}
        ${item}
    END
```

2.2.5 Reporty

Ačkoliv samotné vytváření a provádění testů přináší mnoho výhod, je také nutné (a velmi výhodné) výsledky testů přehledně zobrazovat. Robot Framework po dokončení testování vygeneruje dva HTML soubory, jeden s logy a druhý s reportem, přičemž tyto soubory se vzájemně doplňují. Soubor s logy obsahuje detailní informace o vykonání testů. Obsahuje také statistiku testů, avšak vyšší úroveň shrnutí do statistik je uvedená v report souboru. Robot Framework ještě vygeneruje také dva XML soubory, které obsahují metadata všech výstupů výsledků testů. [1]

Příklad začátku souboru s reportem je na obrázku 2.1. Soubor s reportem obsahuje následující části:

- Hlavička souboru – obsahuje název testu a čas, kdy byl report vygenerován.
- Souhrnné informace (*Summary Information*) – obsahují celkový souhrn (počet neúspěšných testů), čas začátku a konce testování, celkový čas testování a odkaz na soubor s logy.
- Statistiky (*Test Statistics*) – obsahují tabulku s celkovou statistikou, tabulku se statistikou podle tagu a tabulku se statistikou podle testovací sady. Všechny zmíněné tabulky obsahují sloupec s celkovým počtem testů, s počtem úspěšných a neúspěšných testů a čas testování. V posledním sloupci se nachází barevně znázorněný poměr testů úspěšných a neúspěšných.
- Detaily (*Test Details*) – obsahují podrobnější informace než statistiky (například časy začátku a konce jednotlivých testů nebo zprávu s podrobnostmi v případě neúspěšných testů).



Acceptance Report

Generated
20210318 11:38:59 UTC+01:00
34 days 22 hours ago

Summary Information

Status:	41 critical tests failed
Start Time:	20210318 11:32:45.756
End Time:	20210318 11:38:58.315
Elapsed Time:	00:06:12.559
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	108	67	41	00:05:39	
All Tests	108	67	41	00:05:39	

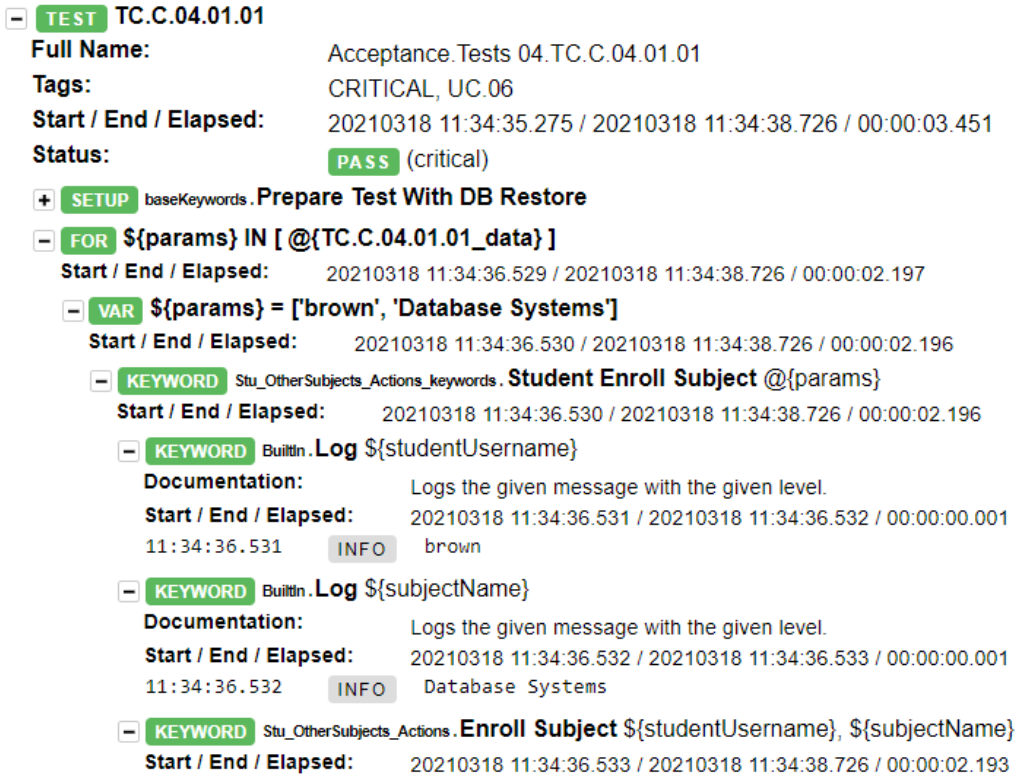
Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
CRITICAL	34	17	17	00:01:48	
MINOR	74	50	24	00:03:50	

Obrázek 2.1: Ukázka reportu

Příklad logů je na obrázku 2.2. Soubor s logy se skládá z těchto částí:

- Hlavička – zde se nachází název testu a čas vzniku souboru s logy.
- Statistiky (*Test Statistics*) – obsahují obdobné tabulky jako soubor s reporty (popsáno výše).
- Logy (*Test Execution Log*) – v hierarchické struktuře uložené veškeré logy do nejmenších detailů. U každého testu jsou vypsány informace jako jeho název, tagy, čas začátku a konce, uplynulý čas, konečný status (úspěšný nebo neúspěšný test). Každá metoda obsahuje informace jako hodnoty argumentů a všechny časy (začátek, konec, uplynulý čas).

  U každé metody je také barevná indikace, jestli metoda proběhla v pořádku (zelená barva) nebo jestli se při jejím vykonávání vyskytla chyba (červená barva). Díky této barevné indikaci je jednoduché na první pohled zjistit, ve kterém kroku test selhal.



Obrázek 2.2: Ukázka logů

3 Analýza

3.1 Analýza báze znalostí projektu

3.1.1 Báze znalostí

Báze znalostí projektu TbUIS se skládá z těchto částí:

- webová aplikace UIS (včetně JUnit testů ve zdrojovém kódu),
- poruchové klony UIS,
- vstupní data – seznam studentů, učitelů a předmětů, včetně vztahů mezi nimi,
- podpůrná knihovna *support* napsaná v Javě s využitím nástroje Selenium WebDriver,
- generátor obalovacích metod (*WrapperGenerator*) pro metody ze *support* knihovny rozšířený o funkcionalitu generování klíčových slov pro Robot Framework,
- sada funkcionálních testů (JUnit),
- sada akceptačních testů (Robot Framework),
- případy užití (*use cases* – UC),
- požadavky (*requirements* – RQM),
- testovací případy (*test cases* – TC).

Z tohoto výčtu je patrné, že báze znalostí TbUIS je poměrně rozsáhlá. Pravděpodobně rozsáhlejší, než u běžných komerčních projektů. Viz též v Diskuzi v kapitole 7.

3.1.2 Jak bude báze znalostí v práci využívána

Webová aplikace UIS (spouštěna s danými vstupními daty) a její poruchové klony budou použity k validaci vygenerovaných akceptačních testů. Podpůrná knihovna *support* bude generovanými akceptačními testy pravděpodobně přímo využívána. Vygenerované soubory z generátoru obalovacích

metod (tedy soubory s wrappery a soubory s klíčovými slovy) budou akceptačními testy pravděpodobně přímo využívány.

Pro případné doplňkové informace bude možné použít sadu funkcionálních testů. Sada akceptačních testů bude sloužit hlavně pro představu, jak by mohly výsledné generované akceptační testy vypadat. Zdokumentované výsledky této sady pak poslouží pro ověření vygenerovaných akceptačních testů

Funkcionalita aplikace UIS je popsána ve třech různých dokumentacích – případy užití, požadavky a testovací případy. Případy užití jednoduše popisují, jaké funkce aplikace UIS poskytuje, respektive jaké akce může koncový uživatel v aplikaci provádět. Požadavky popisují, jakou veškerou funkcionalitu musí aplikace UIS splňovat. Testovací případy se věnují už přímo konkrétním akcím a mají jasně daná vstupní data a také výsledek provedené akce (dají se tedy snadno přímo ověřit).

Všechny tyto dokumenty spolu souvisejí (je možné dopředné i zpětné trasování) – testovací případy jsou detailnější (konkrétnější) než požadavky a ty jsou detailnější než případy užití. Například pro jeden konkrétní případ užití *Změna osobních dat uživatele* existuje pět požadavků a čtrnáct testovacích případů.

3.1.3 Případy užití

Pro aplikaci UIS existuje celkem 22 případů užití (*use cases*). Popis případu užití obsahuje mnoho prvků – například primární aktér, omezení, podmínky před a po vykonání, scénář úspěchu a alternativní toky. Jednotlivé soubory s případy užití mají formát XML, ze kterého je též vygenerován formát HTML. Případy užití pro UIS se dají rozdělit do čtyř kategorií podle hlavního aktéra:

- nepřihlášený uživatel,
- obecný přihlášený uživatel (nezáleží, jestli student nebo učitel),
- přihlášený student,
- přihlášený učitel.

Případem užití v roli nepřihlášeného uživatele může být přihlášení do aplikace UIS. Nepřihlášený uživatel může také provádět speciální aktivity, určené zejména pro testování, jako je například obnovení databáze UIS do původního stavu. V roli obecného přihlášeného uživatele může být případem

užití například změna uživatelských dat. Jedním z případů užití v roli přihlášeného studenta je odepsání svého předmětu. Přihlášený učitel má případ užití například ohodnocení studenta po zkoušce (přidělení známky).

3.1.4 Požadavky

Požadavky a testovací případy byly vytvářeny v test management systému SquashTM. Dále zmiňované JSON soubory vznikly exportem z tohoto systému.

Požadavků (*requirements* – RQM) pro aplikaci UIS bylo sepsáno celkem 259 a jsou uloženy v jednom souboru ve formátu JSON. Pro lepší přehlednost se požadavky, stejně jako testovací případy, dělí do čtyř kategorií podle typu testů:

- typ A – ověřování statického obsahu stránky (celkem 185 požadavků),
- typ B – obsah databáze po její inicializaci (celkem 28 požadavků),
- typ C – bezchybná funkčnost aplikace (celkem 31 požadavků),
- typ D – negativní alternativní toky (popisují očekávanou reakci na nekorektní aktivity uživatele; celkem 15 požadavků).

Příklad zápisu jednoho požadavku:

```
[{
  "rqs": {
    "name": "errorless function - one change",
    "prefix": "RQS.C",
    "description": "",
    "rqsList": [{{
      "rqs": {
        "name": "student's Other Subjects",
        "prefix": "RQS.C.04",
        "description": "<ol><li>executes <a href='
           https://projects.kiv.zcu.cz/tbuis/web/
           files/uis/uc/en/html/use-case-06.html'
           target='_blank'>UC.06</a></li><ol>",
        "rqmList": [{
          "rqm": { // start of RQM
            "name": "enroll subject - boundary
               change",
```

```
            "prefix": "RQM.C.04.02",
            "description": "<ol><li>no subject
               enrolled so far</li><li>already 6
               subjects enrolled</li><li>the message
                TXT_ALERT_SUCCESS is always
               displayed</li><li>executes <a href='
               https://projects.kiv.zcu.cz/tbuis/web
               /files/uis/uc/en/html/use-case-06.
               html' target='_blank'>UC.06</a></li><
               ol>",
            "criticality": "minor",
            "category": "functional"
          } // end of RQM
        }]
      }
    }}]
  }
}]
```

3.1.5 Testovací případy

Stejně jako u požadavků, tak i testovací případy (*test cases* – TC) jsou všechny v jednom souboru ve formátu JSON (který vznikl exportem z test management systému SquashTM). Dělení testovacích případů je také stejné jako u požadavků (typ A, B, C, D). Co je ale rozdílné oproti požadavkům (z důvodu větší detailnosti) je počet testovacích případů – pro UIS jich bylo sepsáno celkem 563. Počty testovacích případů dle typu:

- typ A – ověřování statického obsahu stránky (celkem 376 testovacích případů),
- typ B – obsah databáze po její inicializaci (celkem 28 testovacích případů),
- typ C – bezchybná funkčnost aplikace (celkem 108 testovacích případů),
- typ D – negativní testy (mají za cíl vynutit očekávanou reakci na nekorektní aktivity uživatele; celkem 51 testovacích případů).

Příklad zápisu jednoho testovacího případu:

```
[{
  "ts": {
    "name": "errorless function - one change",
    "prefix": "TS.C",
    "description": "",
    "tsList": [{{
      "ts": {
        "name": "student's Other Subjects",
        "prefix": "TS.C.04",
        "description": "",
        "tsList": [{
          "ts": {
            "name": "enroll subject - boundary
               change",
            "prefix": "TS.C.04.02",
            "description": "",
            "tcList": [{
              "tc": { // start of TC
                "name": "none subject enrolled",
                "prefix": "TC.C.04.02.01",
                "prerequisite": "restoration of the
                   database",
                "description": "<ul><li>so far no
                   subject not enrolled</li><li><
                   code>Magenta, Database Systems</
                   code></li><li>the message
                   TXT_ALERT_SUCCESS is displayed</
                   li><ul>",
                "importance": "low",
                "nature": "functional",
                "type": "compliance",
                "steps": [{
                  "action": "default action",
                  "expected_result": "default result
                     "
                }],
                "verified_requirements": [{
                  "id": "C.04.02"
```

```
                }]
              } // end of TC
            }]
          }}]
        }
      }}]
    }
}]
```

Soubor s testovacími případy je strukturovaný do několika testovacích sad (`ts` – *test suite*), přičemž každá sada obsahuje buď další testovací sady (uvnitř prvku `tsList`) a nebo obsahuje testovací případy (uvnitř prvku `tcList`). Prvky s testovacími případy (`tc` – *test case*) obsahují atributy, které daný testovací případ popisují – název (`name`), prefix, popis (`description`) a další.

3.2 Návrh metodiky

Prvním krokem při návrhu metodiky byl výběr, na základě čeho se budou testy generovat. V úvahu připadaly tři základní zdroje – požadavky, případy užití a testovací případy. Zvoleny byly testovací případy, neboť obsahují nejvíce konkrétních informací. Každý popsaný testovací případ má vždy dáno, jaká akce se má provést, jaký je očekávaný výsledek a také jaká vstupní data mají být použita. Pokud akce vyžaduje přihlášeného uživatele, je dáno, který konkrétní uživatel to má být (jeho uživatelské jméno).

Generované akceptační testy pro Robot Framework budou používat stejnou strukturu testu (testovacího případu) jako již hotové akceptační testy (struktura je blíže popsána v části 2.1.3). Tedy že každý test bude mít danou šablonu (klíčové slovo), která bude zajišťovat provedení akce, a také bude mít daná data, se kterými bude spouštěn. Obě tyto znalosti se převezmou z dokumentu s testovacími případy. Šablonou bude klíčové slovo Robot Frameworku, které bude se zadanými argumenty volat metodu z generátoru obalovacích metod pro metody knihovny *support*.

Každý výsledný vygenerovaný akceptační test bude tedy volat jednu konkrétní metodu z knihovny *support* s konkrétními daty za použití Robot Frameworku.

Během experimentování bylo nutné tuto metodiku několikrát pozměnit. Přesný popis, jak se nakonec generují akceptační testy, se nachází v části 5.6.

3.3 Analýza vytvářené aplikace

3.3.1 Požadavky

Zde budou shrnuty všechny požadavky na aplikaci generující testy, které se již postupně objevovaly v předchozích kapitolách.

Předem dané požadavky:

- Programovacím jazykem aplikace bude Java.
- Výsledné akceptační testy budou vytvářené pro Robot Framework. Testy budou využívat klíčová slova, generovaná z Generátoru klíčových slov. Tato klíčová slova volají podpůrnou knihovnu *support*, která využívá framework Selenium WebDriver.
- Testovanou aplikací bude bezporuchový klon webové aplikace UIS a reprezentativní sada poruchových klonů.

Požadavky doplněné po návrhu metodiky:

- Vstupem aplikace generující testy budou testovací případy a požadavky (obojí ve formátu JSON).

4 Implementace

4.1 Struktura aplikace

Implementovaná aplikace je rozdělená do několika balíků. Hlavními dvěma balíky jsou `generator` a `settings`. V balíku `generator` se nachází všechny potřebné třídy sloužící k vygenerování akceptačních testů. Balík `settings` obsahuje pomocné třídy pro správné nastavení Robot Frameworku při spouštění testů.

Celková struktura vytvořené aplikace Generátor akceptačních testů je uvedena níže na popisu 4.1 (bez balíku `test`, jehož struktura je uvedena v části 4.7):

```
src/main/java/
    |_ generator
        |_ model
            |_ rqs
                |_ Requirement.java
                |_ RequirementParent.java
                |_ RequirementSuite.java
                |_ RequirementSuiteParent.java
            |_ ts
                |_ Step.java
                |_ TestCase.java
                |_ TestCaseParent.java
                |_ TestSuite.java
                |_ TestSuiteParent.java
                |_ VerifiedRequirement.java
        |_ object
            |_ Method.java
            |_ Parameter.java
            |_ ParamType.java
        |_ utils
            |_ Constants.java
            |_ FileLoader.java
            |_ FileSaver.java
            |_ GeneratorConfig.java
            |_ RequirementsPreprocessing.java
            |_ RequirementsSuiteDeserializer.java
            |_ TagSearcher.java
            |_ TestsPreprocessing.java
            |_ TestSuiteDeserializer.java
        |_ Generator.java
        |_ Main.java
```

```
    |_ settings
        |_ IConfigurable.java
        |_ TestSettingRobotFramework.java
```

Listing 4.1: Struktura balíku `java` Generátoru akceptačních testů

4.2 Balík `generator.model`

V tomto balíku se nacházejí třídy pouze s `public` atributy (dále jen „modelové třídy“), na které je mapován obsah JSON souboru. Tyto třídy byly vytvořeny přesně podle struktury JSON souboru buď s požadavky (balík `rqs`) nebo s testovacími případy (balík `ts`). Jejich účel je jen pro bezproblémové načtení a uložení dat z JSON souboru pomocí veřejné knihovny `com.google.gson`.

Všechny třídy v tomto balíku mají jen veřejné atributy a neobsahují žádné metody. Mezi atributy každé třídy spadají všechny podřazené elementy dle struktury daného JSON souboru. Některé třídy mají ale také atributy, které reprezentují nadřazené elementy.

Například třída `TestCase` má několik atributů, přičemž jeden konkrétní atribut `TestSuite parentTs` je výjimečný v tom, že se jedná o nadřazený element, do kterého daný `TestCase` spadá. Všechny ostatní atributy jsou už podřazené elementy daného testovacího případu. Pro příklad je zde uveden celý kód třídy `TestCase`:

```
1  package generator.model.ts;
2  import java.util.List;
3
4  public class TestCase {
5      public String name;
6      public String prefix;
7      public String prerequisite;
8      public String description;
9      public String importance;
10     public String nature;
11     public String type;
12     public List<Step> steps;
13     public List<VerifiedRequirement>
           verified_requirements;
14     public TestSuite parentTs;
15 }
```

4.3 Balík `generator.object`

V balíku `object` se nacházejí všechny třídy, jejichž instance reprezentují objekty důležité pro vygenerování testovacího případu. Patří sem třídy pro reprezentaci metody, parametru a typu parametru.

4.3.1 Třída `Method`

Instance třídy `Method` uchovávají tyto informace – název metody, názvy parametrů a počet povinných parametrů. Některé metody v knihovně *support* jsou totiž přetížené (mají stejný název ale různý počet parametrů). Toto je řešeno tak, že data do atributu s názvy parametrů se vezmou z té metody, která jich má nejvíce. Pokud je některý z parametrů nepovinný (existuje verze metody, která tento parametr nemá), do atributu s názvy parametrů se uloží se znakem „=“ na posledním indexu (tento princip rozlišování nepovinného parametru byl převzat z Robot Frameworku). Díky tomu je možné snadno rozlišit, který parametr je povinný, a tedy snadno vypočítat hodnotu atributu uchovávajícího počet povinných parametrů.

4.3.2 Třídy `Parameter` a `ParamType`

Třída `Parameter` má dva atributy – data (ukládaná vždy jako řetězec) a `ParamType` označující typ parametru. Výčtový typ `ParamType` uchovává všechny typy parametrů, se kterými je možné se v knihovně *support* setkat. Nejedná se o datové typy, ale o ručně nadefinované typy parametrů podle informace, jakou parametr nese.

V knihovně *support* se rozlišují tyto typy parametrů:

- `STUDENT` – uživatelské jméno studenta,
- `TEACHER` – uživatelské jméno učitele,
- `SUBJECT` – název předmětu,
- `FIRSTNAME` – křestní jméno (používané při změně uživatelských údajů),
- `LASTNAME` – příjmení (používané při změně uživatelských údajů),
- `EMAIL` – e-mailová adresa (používaná při změně uživatelských údajů),
- `CLICKABLE_SUBJECT` – název předmětu, na který se má kliknout (používaný ve specifickém testovacím případu),

- `ENROLLED_SUBJECT` – název předmětu, který má student zapsán,
- `PARTICIPANTS` – počet povolených účastníků na zkoušce,
- `GRADE` – známka z předmětu.

4.4 Balík `generator.utils`

Uvnitř balíku `utils` se nacházejí všechny pomocné třídy pro načítání nebo ukládání souborů a také pro zpracovávání jejich obsahu.

4.4.1 Třída `FileLoader`

Třída `FileLoader` umožňuje načítání souborů. Obsahuje celkem tři statické metody. Všechny tyto metody mají pouze jeden parametr – `String filename`, tedy řetězec s názvem souboru.

Každá z metod umí načíst jiný soubor:

- `TestSuiteParent[] loadTestCases` – dokáže načíst soubor v JSON formátu s testovacími případy,
- `RequirementSuiteParent[] loadRequirements` – dokáže načíst soubor v JSON formátu s požadavky,
- `List<Method> loadMethods` – dokáže načíst soubor se seznamem metod včetně jejich parametrů.

4.4.2 Třída `FileSaver`

Třída pro ukládání obsahu do souboru se nazývá `FileSaver`. Tato třída obsahuje pouze dvě veřejné statické metody, přičemž obě metody mají za parametr vždy jen list s řetězci.

Metoda `saveYamlData()` uloží data do několika souborů (formát *YAML*). Rozdělení do souborů je dáno prefixem testovacího případu, ke kterému data patří. Například soubor *data_01.yaml* obsahuje vstupní data pro testovací případy s prefixem `TC.C.01`, tedy pro testovací případy `TC.C.01.01.01`, `TC.C.01.01.02`, ..., až po `TC.C.01.05.02`.

Druhá metoda `saveTests()` ukládá zdrojové kódy akceptačních testů ve formátu testovacích případů pro Robot Framework. Rozdělení do souborů je stejné jako u dat. Všechny tyto soubory (data a akceptační testy) se ukládají přímo do složky `test`, aby bylo možné testy rovnou spustit.

4.4.3 Třída `GeneratorConfig`

Pro načítání obsahu konfiguračního souboru pro Generátor akceptačních testů existuje třída `GeneratorConfig`. Tato třída obsahuje metodu, která má jako parametr cestu ke konfiguračnímu souboru a dokáže daný konfigurační soubor načíst. Po načtení zmíněného souboru tato třída dané informace o konfiguraci poskytuje v rámci běhu celé aplikace.

4.4.4 Třídy s příponou `Deserializer`

Dále balík obsahuje dvě třídy – `RequirementSuiteDeserializer` a také `TestSuiteDeserializer`. Třídy implementují rozhraní `JsonDeserializer` z knihovny `com.google.gson`. Obě třídy obsahují jen jednu implementovanou metodu `deserialize()`, která slouží pro deserializaci JSON souboru (respektive jeho elementů) do Java objektů. Protože bylo zapotřebí, aby testovací případy a požadavky měly vždy referenci na nadřazený element, bylo nutné napsat vlastní implementaci metody `deserialize()`.

4.4.5 Třída `RequirementsPreprocessing`

Další třídou, která se nachází v balíku `utils`, je třída pro předzpracování požadavků (RQM) – `RequirementsPreprocessing`. Tato třída poskytuje jen jednu veřejnou metodu, jejíž hlavička vypadá takto:

```
public Map<String, Requirement> preprocessRqsParents
    (RequirementSuiteParent[] rqsParents)
```

Výše uvedená metoda prochází všechny elementy v `rqsParents`, ve kterých hledá jednotlivé požadavky, které následně ukládá do mapy. Klíčem v mapě je řetězec s prefixem požadavku (například `RQM.C.01.01`) a hodnotou příslušná instance třídy `Requirement`.

4.4.6 Třída `TestsPreprocessing`

Balík `utils` obsahuje také třídu `TestsPreprocessing`, která zpracovává testovací případy. Třída dokáže buď předzpracovat testovací případy z elementů načtených ze souboru, ale také dokáže v testovacím případu (nebo testovací sadě) najít potřebné informace.

Metoda pro předzpracování postupně prochází všechny instance třídy `TestSuiteParent` z parametru a hledá v nich jednotlivé testovací případy. Nalezené testovací případy ukládá jako instance třídy `TestCase` do seznamu, který poté vrací v návratové hodnotě. Metoda má tuto hlavičku:

```
public List<TestCase> preprocessTestSuiteParents
    (TestSuiteParent[] testSuiteParents)
```

Další veřejná metoda postupně prochází všechny nadřazené testovací sady daného testovacího případu a získává z těchto sad jejich názvy. Tyto názvy oddělené mezerou poté vrací jako návratovou hodnotu. Hlavička metody vypadá následovně:

```
public static String getAllTsNames(TestCase tc)
```

Poslední veřejnou metodou ve třídě `TestsPreprocessing` je metoda s názvem `getAllTcInformation()`. Tato metoda z daného testovacího případu (předaného jako parametr) získává veškeré informace potřebné pro nalezení správné metody z knihovny *support*. K získání všech informací je nutné projít všechny nadřazené elementy (testovací sady) daného testovacího případu.

4.4.7 Třída `TagSearcher`

Třída `TagSearcher` nabízí metody pro prohledávání HTML tagů a vyjímání některého obsahu z nich. Prohledávání HTML tagů je zapotřebí při zpracovávání atributu `description` z popisu testovacího případu z JSON souboru. Tento atribut totiž používá HTML tagy a jen díky nim je například možné získat vstupní data z tohoto atributu.

Metoda `findTextInsideCode()` prohledá řetězec z parametru a vrátí řetězec, který se nacházel uvnitř tagu `<code>`. Druhá metoda funguje na stejném principu, pouze nevyhledává v tagu `<code>`, ale v tagu `<a>`.

4.4.8 Třída `Constants`

Poslední třídou v balíku `utils` je třída `Constants`. V této třídě se nachází konstantní řetězce, používané pro generování zdrojových kódů testovacích případů a také dat. Nachází se zde také původně ručně vytvořená mapa všech správných názvů metod z knihovny *support*, které jsou vždy podle klíče (prefix testovacího případu) přiděleny danému testovacímu případu. Tato mapa sloužila pro generování testů v první fázi experimentu, nyní slouží pouze ke kontrole, jestli došlo k přiřazení správného názvu metody z knihovny *support*.

Třída `Constants` obsahuje také jednu veřejnou statickou metodu, která ale pouze vloží řetězec z parametru dovnitř dlouhého řetězce a ten poté vrací.

4.5 Nezařazené třídy v balíku `generator`

4.5.1 Třída `Generator`

Nejdůležitější třídou celé aplikace je třída `Generator`, jejíž hlavním úkolem je generování zdrojových kódů akceptačních testů a také generování obsahu souborů se vstupními daty pro jednotlivé testy. V konstruktoru třídy `Generator` se jako parametry předávají veškerá data načtená ze vstupních souborů, která jsou potřebná pro vygenerování všech výstupních souborů.

Hlavní metodou třídy `Generator` je metoda `generate()`, která postupně prochází jednotlivé testovací případy a generuje pro ně odpovídající akceptační testy a data. Pro každý testovací případ se provedou následující kroky (detailnější popis jednotlivých kroků se nachází v části 5.6):

1. nalezne odpovídající požadavek (z něhož použije informace pro budoucí Robot Framework tagy `criticality` a `UC`),
2. vyhledá v popisu testovacího případu (případně v popisu nadřazených testovacích sad) vstupní data,
3. vybere odpovídající metodu z knihovny *support* (respektive odpovídající klíčové slovo, které danou metodu volá),
4. uloží řetězec se vstupními daty pro daný testovací případ,
5. uloží řetězec se zdrojovým kódem akceptačního testu pro daný testovací případ.

Ve třídě `Generator` se nacházejí všechny pomocné metody, které jsou zapotřebí při provádění výše zmíněných kroků během generování akceptačních testů.

4.5.2 Třída `Main`

Ve třídě `Main` se nachází pouze jediná metoda `main()`, která je vstupním bodem celé aplikace. V této metodě dochází k propojení tříd, které dokáží načíst soubory a předzpracovat je, se třídou `Generator`. Ta zajišťuje samotné generování zdrojových kódů akceptačních testů pro Robot Framework. Nakonec jsou v metodě `main()` volány metody ze třídy `FileSaver`, které uloží vygenerované zdrojové kódy testů a vstupní data do souborů.

4.6 Balík `settings`

V balíku `settings` se nachází pomocné třídy pro správnou konfiguraci Robot Frameworku při spouštění akceptačních testů. Balík obsahuje pouze dva soubory – jedno rozhraní a jednu třídu, která toto rozhraní implementuje.

4.6.1 Rozhraní `IConfigurable`

Rozhraní `IConfigurable` obsahuje čtyři metody, které jsou standardně zapotřebí při testování. Jsou to:

- `void beforeSuite()` – jaké akce mají být vykonány před zahájením celé testovací sady,
- `void afterSuite()` – jaké akce mají být vykonány po skončení celé testovací sady,
- `void beforeTest()` – jaké akce mají být vykonány před zahájením testu,
- `void afterTest()` – jaké akce mají být vykonány po skončení testu.

4.6.2 Třída `TestSettingRobotFramework`

Tato třída implementuje rozhraní `IConfigurable`, přičemž ale ve skutečnosti využívá pouze metody `beforeSuite()` a `afterSuite()` (těla zbylých dvou metod nechává prázdná). Celá třída `TestSettingRobotFramework` vychází ze třídy s totožným názvem z práce [9]. Třída prošla pouze menšími úpravami (přejmenování metod, přidání rozhraní), přičemž výkonný kód metod zůstal buď stejný nebo byl zjednodušen.

4.7 Balík `test` (vygenerované soubory)

Uvnitř generovaného balíku `test` se nachází veškeré zdrojové kódy a soubory, potřebné pro vykonání akceptačních testů. Vygenerované akceptační testy se ukládají do balíku `robotframework.acceptance` a soubory s daty pro jednotlivé testy se generují do balíku `robotframework.data`. Posledním balíkem v balíku `test` je `robotframework.keywords`, který obsahuje klíčová slova.

Struktura balíku `test` je uvedena níže na popisu 4.2 (pro zjednodušení zde nejsou rozepsány soubory uvnitř balíků `login`, `student` a `teacher`):

```
src/main/test/
    |_ robotframework
        |_ acceptance
            |_ tests_01.robot
            ...
            |_ tests_13.robot
        |_ data
            |_ data_01.yaml
            ...
            |_ data_13.yaml
        |_ keywords
            |_ login
            |_ student
            |_ teacher
            |_ baseKeywords.robot
```

Listing 4.2: Struktura balíku `test`

4.7.1 Balík `test.robotframework.acceptance`

Vygenerované akceptační testy se ukládají přímo do složky pro testování (konkrétně `src/main/test/robotframework/acceptance`), aby mohly být rovnou spouštěny. Obsahy vygenerovaných souborů s testy mají vždy stejnou strukturu – v první části je uvedeno veškeré nastavení včetně potřebných knihoven a ve druhé části se nacházejí samotné výkonné kódy akceptačních testů. Příklad souboru s akceptačními testy je uveden v popisu 4.3.

```
1  *** Settings ***
2
3  Resource    ../keywords/baseKeywords.robot
4  Resource    ../keywords/student/studentKeywords.robot
5  Resource    ../keywords/teacher/teacherKeywords.robot
6
7  Variables   ../data/data_08.yaml
8
9  Suite Setup         Prepare Suite without DB restore
10 Test Setup          Prepare Test with DB restore
11 Suite Teardown      End Suite
12
13 *** Test Cases ***
14
15 TC.C.08.01.01
16     [Template]     Teacher Remove Subject
```

```
17      [Tags]          UC.10   CRITICAL
18      FOR     ${params}     IN     @{TC.C.08.01.01_data}
19          @{params}
20      END
21
22  TC.C.08.01.02
23      [Template]     Teacher Remove Subject
24      [Tags]          UC.10   CRITICAL
25      FOR     ${params}     IN     @{TC.C.08.01.02_data}
26          @{params}
27      END
28
29  TC.C.08.01.03
30      [Template]     Teacher Remove Subject
31      [Tags]          UC.10   CRITICAL
32      FOR     ${params}     IN     @{TC.C.08.01.03_data}
33          @{params}
34      END
```

Listing 4.3: Příklad vygenerovaného souboru s akceptačními testy

První část souboru s akceptačními testy (uvozena řádkem `*** Settings ***`) obsahuje nejprve tři knihovny s klíčovými slovy. Jedná se o základní klíčová slova pro nastavení Robot Frameworku (`baseKeywords.robot`) a pak o klíčová slova, pomocí nichž je volána knihovna *support* (`studentKeywords` a `teacherKeywords`). Dále jsou v první části importována data ze souboru ve formátu *YAML*, přičemž soubor s daty má v názvu vždy stejné číslo, jako soubor s akceptačními testy. Na konci první části jsou uvedeny metody z `baseKeywords.robot`, které se volají buď před spuštěním celé testovací sady (`Suite Setup`) nebo před spuštěním jednoho testu (`Test Setup`) nebo po skončení všech testů v testovací sadě (`Suite Teardown`).

Ve druhé části souboru, která je uvozena řádkem `*** Test Cases ***`, se nachází výkonný kód jednotlivých akceptačních testů. Na řádku začínajícím `[Template]` je uveden název klíčového slova, pomocí něhož je volána odpovídající metoda z knihovny *support*. Řádek s tagy (`[Tags]`) obsahuje vždy dva tagy – jeden označuje případ užití, ke kterému se daný akceptační test (testovací případ) vztahuje (například `UC.10`) a druhý tag označuje kritičnost (závažnost) akceptačního testu. Kritičnost může obecně nabývat buď hodnoty `CRITICAL` (velmi vysoká závažnost), `MAJOR` (vysoká závažnost) nebo `MINOR` (nízká závažnost). Protože v této práci se zpracovávají testovací případy a požadavky pouze typu C, vyskytují se ve vygenerovaných

akceptačních testech jen tagy `CRITICAL` a `MINOR` (typ C neobsahuje žádné testovací případy a požadavky s tagem `MAJOR`). Oba tagy jsou pouze převzaté z požadavku, ke kterému je daný test (testovací případ) vázán. Informace o daném požadavku se získávají ze vstupního *JSON* souboru s požadavky.

Výkonným kódem akceptačního testu je `for` cyklus. Ten načítá vstupní data z *YAML* souboru s daty pro daný testovací případ a pro jednotlivá data spouští test (volá klíčové slovo uvedené za slovem `[Template]`).

4.7.2 Balík `test.robotframework.data`

Společně s akceptačními testy se současně generují také soubory se vstupními daty, se kterými jsou dané akceptační testy spouštěny. Tyto datové soubory mají formát *YAML* a ukládají se přímo do složky pro testování (konkrétně `src/main/test/robotframework/data`).

Všechny vygenerované soubory s daty mají stejnou strukturu – vždy je nejprve uveden název datové sady (například `TC.C.08.01.01_data:`) a poté jsou na samostatných řádcích uvedeny jednotlivé hodnoty. Příklad takového datového souboru je uveden v popisu 4.4.

```
1  TC.C.08.01.01_data:
2    -
3      - pedant
4      - Operating Systems
5
6  TC.C.08.01.02_data:
7    -
8      - strict
9      - Operating Systems
10
11 TC.C.08.01.03_data:
12   -
13     - pedant
14     - Operating Systems
```

Listing 4.4: Příklad vygenerovaného souboru s daty

4.7.3 Balík `test.robotframework.keywords`

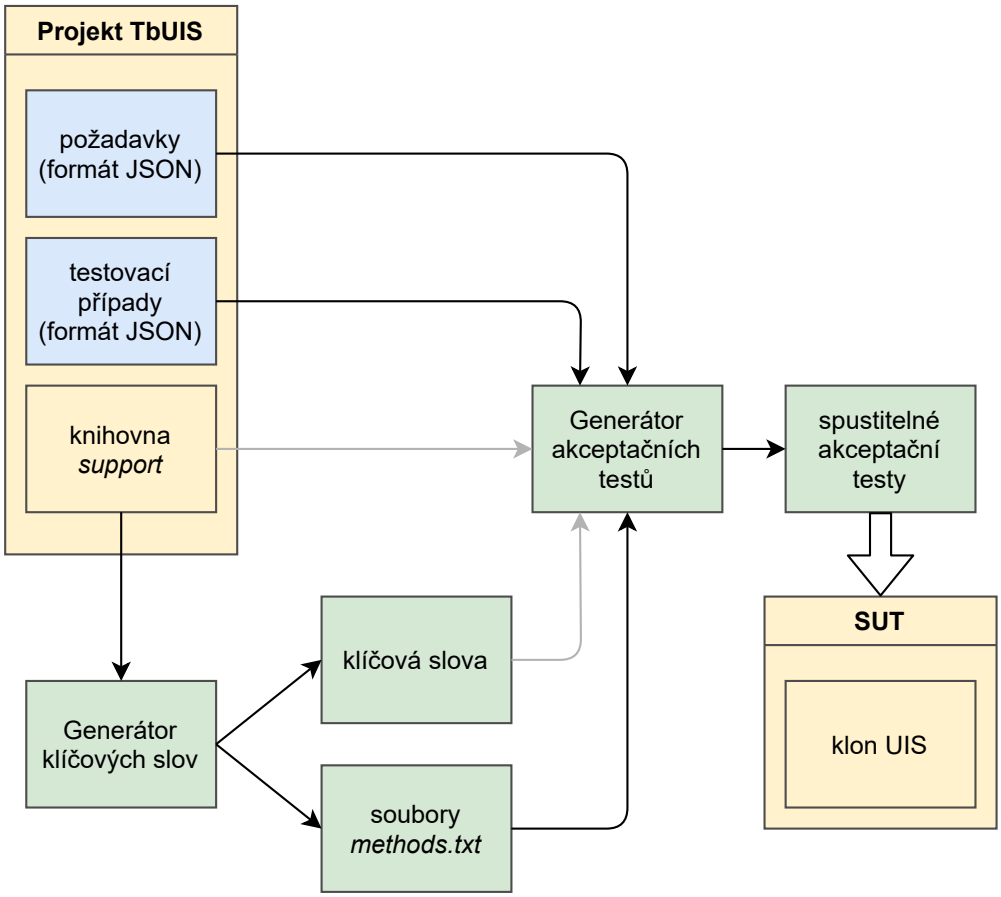
Poslední balík uvnitř balíku `test` se nazývá `keywords` a jeho struktura přímo kopíruje strukturu balíku `uis.support.services`, který se nachází v knihovně *support*. Oba balíky tedy obsahují tři balíky – `login`, `student`

a `teacher`. Uvnitř těchto balíků v balíku `keywords` se nachází vygenerovaná klíčová slova pro Robot Framework, která jsou podrobněji popsána v části 5.2. Generátor klíčových slov do každého balíku (`login`, `student` a `teacher`) také vygeneruje soubor *methods.txt*, kam se ukládají všechny názvy vygenerovaných klíčových slov včetně názvů jejich parametrů. Příklad takového souboru je uveden v části 5.2.

Uvnitř balíku `test.robotframework.keywords` se nachází také soubor `baseKeywords.robot`, což je jediný soubor v celém balíku `test`, který není generován. Tento soubor obsahuje několik klíčových slov, která slouží pro konfiguraci akceptačních testů. Soubor byl převzat z manuálně psané sady akceptačních testů a byl zjednodušen (byla vymazána některá nepotřebná klíčová slova).

4.8 Kontext aplikace

Na obrázku 4.1 je uvedené schéma kontextu celé práce. Zelenou barvou jsou vyznačeny části, které tvoří výsledky této diplomové práce – vytvořená aplikace Generátor akceptačních testů, který generuje spustitelné akceptační testy, a také zjednodušená aplikace Generátor klíčových slov, která generuje klíčová slova a soubory *methods.txt*. Aplikace přímo používá (značeno černou šipkou) jako vstupní soubory požadavky, testovací případy a také soubory *methods.txt*, kde jsou uvedeny všechny názvy klíčových slov včetně názvů jejich parametrů. Šedou šipkou je naznačena závislost Generátoru klíčových slov na knihovně *support* a také na vygenerovaných klíčových slovech. Dvojitou velkou šipkou je naznačen vztah, že vygenerované spustitelné akceptační testy testují webovou aplikaci UIS.



Obrázek 4.1: Kontext aplikace

5 Postup vývoje aplikace a experimentů

5.1 Fáze vývoje

Řešení celé práce probíhalo postupně následujícími fázemi:

1. Studium Robot Frameworku – především formátu klíčových slov a testovacích případů.
2. Prozkoumání diplomové práce *Akceptační testování v projektu TbUIS*, především vytvořených testů pro Robot Framework.
3. Analýza báze znalostí UIS.
4. Vylepšení generátoru klíčových slov (například logování nebo přizpůsobení i pro proměnlivý počet parametrů metody).
5. Experiment – vygenerování jednoho akceptačního testu na základě dat z dokumentu testovacích případů v JSON formátu. Data se získají z tagu `<code>`.
6. Vylepšení knihovny *support* – její metody budou provádět a rovnou i ověřovat výsledek akce.
7. První spuštění vygenerovaných akceptačních testů – úspěšné, ale bylo potřeba převzít několik ručně psaných souborů z diplomové práce *Akceptační testování v projektu TbUIS*.
8. Doplnění knihovny *support* o potřebné metody (například metoda pro změnu křestního jména i příjmení najednou).
9. Doplnění JSON souboru s testovacími případy (například o jméno uživatele, se kterým má být akce provedena, pokud tam tato informace chyběla).
10. Spuštění vygenerovaných akceptačních testů na jednom poruchovém klonu (konkrétně na klonu `13-C0.H1.M0.L0_G_T_D_01`).
11. Analýza převzatých manuálně psaných souborů – některé vymazány, některé začleněny do knihovny *support* a některé jen zjednodušeny. Pro úplnost zde budou uvedeny převzaté soubory:

- Do knihovny *support* byly začleněny soubory (respektive třídy) `EntityValidator` a `UnknownObjectException`.
- Převzaty do Generátoru akceptačních testů (a případně jen zjednodušeny) byly dva soubory – prvním z nich je soubor s názvem `TestSettingRobotFramework`, přičemž se jedná o Java třídu pro konfiguraci Robot Frameworku. Druhým je `baseKeywords.robot`, který připravuje testovací prostředí před a po testu nebo testovací sadě a využívá zmíněnou třídu `TestSettingRobotFramework`.

12. Analýza „přirozeného“ jazyka[1] v popisu testovacích případů.
13. Analýza počtu parametrů a jejich typu u metod z knihovny *support* a správné přiřazení dat testovacího případu do jednotlivých parametrů.
14. Ověření vygenerovaných akceptačních testů na bezporuchovém klonu a také všech poruchových klonech (celkem 28).
15. Porovnání výsledků vygenerovaných akceptačních testů s manuálně psanými akceptačními testy.

5.2 Vylepšení generátoru klíčových slov

Během vypracovávání této práce bylo také zapotřebí provádět změny v generátoru klíčových slov. První změnou bylo, že se klíčová slova budou generovat jen pro zdrojové soubory z knihovny *support* s příponou *_Actions* a už ne pro soubory s příponou *_Check*. Tuto změnu bylo možné provést až poté, co veškerá ověření zajišťovaná metodami ze souborů s příponou *_Check* byla volána rovnou ze souborů s příponou *_Actions* (úprava v knihovně *support*).

Velkou úpravou generátoru klíčových slov bylo odstranění jejich závislosti na wrapperech. Bylo totiž zjištěno, že je možné volat metody z knihovny *support* přímo, pokud se v ní provedou menší úpravy (více o úpravách knihovny *support* je uvedeno v části 5.4). Pro odstranění závislosti na wrapperech bylo také nutné vylepšit generátor klíčových slov, aby podporoval parametry typu `Integer`.

Dalším vylepšením bylo při generování klíčových slov vygenerovat také soubor *methods.txt*. Zprvu tento soubor obsahoval pouze názvy všech vygenerovaných klíčových slov v dané složce. Například:

[1] Dále bude používán termín „přirozený jazyk“. Tímto termínem je myšlen zápis testovacího případu v JSON souboru. Ve skutečnosti se jedná o velmi specifickou podmnožinu skutečného přirozeného jazyka.

```
    Student Change Email
    Student Register Exam Date
```

Poté ale bylo nutné, aby soubor obsahoval také názvy parametrů, takže nyní soubor vypadá takto:

```
    Student Change Email : studentUsername newEmail
    Student Register Exam Date : studentUsername
       subjectName examDate=
```

Název klíčového slova je oddělen dvojtečkou, za kterou následují názvy parametrů (oddělené mezerami). Některé metody v knihovně *support* jsou přetížené, tedy existují ve verzích s různými počty parametrů. Pro označení těchto parametrů, které jsou nepovinné, byla použita stejná syntaxe, jakou používá Robot Framework – tedy na konec názvu parametru se přidá symbol „=“ (uvedeno v příkladu výše na posledním řádku).

Posledním krokem při vylepšení generátoru klíčových slov bylo odstranění všeho nepotřebného z aplikace (všeho, co se týkalo wrapperů). Původně se totiž aplikace nazývala *WrapperGenerator*, tedy generátor wrapperů. Samotný generátor klíčových slov tedy neexistoval jako samostatná aplikace, jen jako rozšíření aplikace *WrapperGenerator*. Avšak po zjištění, že wrappery nejsou nutné, bylo z aplikace odstraněno vše, co se týkalo jejich generování. Posledním krokem bylo přejmenování aplikace *WrapperGenerator* na *KeywordGenerator* (Generátor klíčových slov). Aplikace ale stále obsahuje některé soubory z původního projektu, neboť princip čtení knihovny *support* zůstal v aplikaci *KeywordGenerator* kompletně zachován.

Příklad finální podoby vygenerovaného klíčového slova:

```
*** Settings ***

Library           uis.support.services.student.
   m3_othersubjects.Stu_OtherSubjects_Actions
   WITH NAME  Stu_OtherSubjects_Actions

*** Keywords ***

Student Enroll Subject
    [Arguments]        ${studentUsername}
        ${subjectName}
    Log                ${studentUsername}
    Log                ${subjectName}
```

```
12      Stu_OtherSubjects_Actions.Enroll Subject
          ${studentUsername}  ${subjectName}
```

Původní příklad klíčového slova před všemi úpravami generátoru během vypracovávání této práce byl uveden už v části 2.1.4, avšak pro rychlé porovnání je zde uveden ještě jednou:

```
1  *** Settings ***
2
3  Library           uis.support.services.student.
     m3_othersubjects.
     Stu_OtherSubjects_Actions_Wrapper  WITH NAME
     Stu_OtherSubjects_Actions
4
5  *** Keywords ***
6
7  Student Enroll Subject
8      [Arguments]        ${arg0}  ${arg1}
9      Log                ${arg0}
10     Log                ${arg1}
11     Stu_OtherSubjects_Actions.Enroll Subject
         ${arg0}  ${arg1}
```

5.3 Výběr metody z knihovny *support*

Pro vygenerování testu bylo nutné znát název metody z knihovny *support*, kterou daný test použije. V první fázi generování testů byla tato informace dohledána ručně, ale v další fázi se začalo experimentovat s automatizací (rozpoznávání přirozeného jazyka). Ale i pokud byl nalezen správný název metody, ještě bylo nutné zajistit, aby byly předány všechny skutečné hodnoty parametrů a to ve správném pořadí.

5.3.1 Postup nalezení správného názvu metody

Při generování klíčových slov pro Robot Framework na základě knihovny *support* se také vygeneruje několik souborů (pro každý podadresář z balíku `uis.support.services` jeden soubor). Jednotlivé řádky vygenerovaných souborů jsou tvořeny názvy vygenerovaných klíčových slov včetně názvů jejich parametrů dle metod, které se nachází v daném podadresáři. Tyto

vygenerované soubory slouží jako základ při automatizovaném přiřazovaní testovacího případu k názvu metody (respektive k názvu klíčového slova).

Přiřazení názvu klíčového slova k testovacímu případu probíhá v těchto krocích:

1. získání všech informací o testovacím případu,
2. předzpracování informací o testovacím případu,
3. ohodnocování shody názvu metody a informací o testovacím případu (počítání skóre),
4. výběr názvu metody dle dosaženého skóre – metoda s nejvyšším skóre je vybrána.

5.3.2 Optimalizace analýzy přirozeného jazyka

Bylo nutno provést několik experimentů, co všechno do informací o testovacím případu je nutné zahrnout a co tam naopak raději nedávat (příliš mnoho informací vedlo často k větší neúspěšnosti). Bylo možné tam zahrnout čtyři typy textů – popis testovacího případu, název testovacího případu, popis nadřazené testovací sady a název nadřazené testovací sady (přičemž každý testovací případ může mít několik nadřazených testovacích sad). Výběr těchto informací měl zásadní vliv na výsledné přiřazení názvů metod. Jako nejúspěšnější byla vyhodnocena kombinace názvu testovacího případu s názvem a popisem nadřazených testovacích sad.

Při předzpracování byly prováděny různé experimenty, co v těchto informacích změnit či odstranit. V jednom z experimentů byla z informací o testovacím případu vyjmuta slova, která nenesou mnoho významu, jako jsou předložky (například *in*, *on*) a spojky (například *and*). Toto odstranění slov napomohlo při hledání správného názvu metody. Také se experimentovalo s nahrazením některých slov jejich synonymy, což se osvědčilo jako úspěšná pomoc při hledání. Například aby se propojila metoda `participateInSubject` s popisem „signs up to teach a subject - common change“, bylo nutné v popisu nahradit slovo „teach“ slovem „participate“.

První základní úvahou ohledně ohodnocování shody názvu metody a informací o testovacím případu bylo nejprve přidání podmínky, že pokud se slovo vyskytuje v informacích o testovacím případu a také v názvu metody, přičte se do skóre názvu metody hodnota 2. Druhým experimentem bylo znevýhodnění těch metod, které ve svém názvu obsahují slova, která se v informacích o testovacím případu vůbec nevyskytují. Pokud takové slovo název metody obsahoval, skóre shody se snížilo o 1. Tento experiment se

znevýhodněním vedl k lepšímu nalezení názvu metody. Například pokud by toto znevýhodnění zavedeno nebylo, tak pro testovací případ `TC.C.02.02.01` s informacemi „*only one enrolled subject with exam date unenroll of subject - boundary change student's my subjects / enrolled subjects errorless function - one change*“ by skóre metod `Student Unenroll Subject` a `Student Skip To Unenroll Subject` bylo stejné (hodnota 7 – za slovo `unenroll` se přičte 2 a za slovo `subject` se přičte 5, neboť se vyskytuje dvakrát). Avšak když se použije znevýhodnění metod se slovy, které se v informacích o testovacím případu nevyskytují, hodnoty obou skóre dopadnou rozdílně. Pro metodu `Student Unenroll Subject` se nic nezmění, skóre bude stále 7, avšak pro metodu `Student Skip To Unenroll Subject` bude skóre už jenom 5 (skóre sníženo dvakrát o 1 kvůli slovům `Skip` a `To`). A protože jsou skóre rozdílná, Generátor snadno a jednoznačně zvolí správný název metody.

Dalším experimentem při ohodnocování bylo přidání podmínky, že pokud se shodné slovo vyskytuje v popisu testovacího případu vícekrát, tak při každém dalším výskytu se ke skóre přičte 1. Díky této podmínce se hledání správného názvu metody vylepšilo. Například pro testovací případ `TC.C.04.01.02` s následujícími informacemi „*one subject without registered exam date enroll subject - common change student's other subjects errorless function - one change*“ se správná metoda s názvem `Student Enroll Subject` ohodnotila se skóre 7. Slovo `Student` se při ohodnocování přeskakuje, dále za slovo `Enroll` se připočetlo skóre 2 a za slovo `Subject` se připočetlo 5 (vyskytuje se v popisu dvakrát, při prvním výskytu se připočte 2 a při druhém 3, protože se jedná o opakovaný výskyt).

Posledním experimentem při ohodnocování shody bylo zvýhodnění, pokud se shodná slova vyskytují za sebou. Pokud se nalezlo shodné slovo a zároveň poslední shodné slovo bylo to předchozí, skóre se navíc zvýšilo o 1. Toto zvýhodnění ale nakonec nevedlo k lepším výsledkům při hledání správného názvu metody. Například když se použilo zvýhodnění, pro testovací případ `TC.C.02.02.03` s informacemi „*only one student with exam date unenroll of subject - boundary change student's my subjects / enrolled subjects errorless function - one change*“ se pro metody `Student Register Exam Date`, `Student Unregister Exam Date` a `Student Unenroll Subject` spočítalo shodné skóre s hodnotou 4. Protože se v informacích o testovacím případu vyskytují slova `exam` a `date` za sebou, bylo prvním dvěma metodám zvýšeno skóre o 1. Ale protože názvy prvních dvou metod obsahují slova, které se v informacích o testovacím případu nevyskytují (`Register` a `Unregister`), skóre jim bylo sníženo o 1 a tak vzniklo shodné skóre pro všechny tři zmíněné metody. Avšak pokud se zvýhodnění metod se shodnými slovy vyskytujícími se za sebou nepoužije, dopadne skóre metod `Student Register Exam`

`Date` a `Student Unregister Exam Date` o 1 nižší, tedy hodnota 3. Díky tomu může být jednoznačně zvolena správná metoda (tedy metoda s názvem `Student Unenroll Subject`). Z tohoto důvodu se nakonec zmíněné zvýhodnění slov vyskytujících se za sebou nepoužilo.

5.3.3 Předání parametrů

Výběrem správné metody z knihovny *support* ale ještě nebylo možné skončit, neboť každá metoda má jasně určené parametry včetně jejich pořadí. V souboru s testovacími případy pak každý případ obsahuje konkrétní data, se kterými má být spuštěn. Bylo tedy zapotřebí tato data zanalyzovat a poté je ve správném pořadí použít při volání metody z knihovny *support*.

V první fázi generování akceptačních testů bylo předávání parametrů (jejich počet i pořadí) zkontrolováno nejprve manuálně. U každého testovacího případu byl zkontrolován počet parametrů včetně pořadí a pokud bylo pořadí nevyhovující, bylo pozměněno, aby bylo správné. Tyto změny byly prováděny buď v knihovně *support* nebo v JSON souboru s popisy testovacích případů.

Poté, co experiment s vyhledáním správného názvu metody z knihovny *support* byl úspěšný, se pokročilo o kus dál a začalo se pracovat na analýze dat, která jsou použita jako parametry. Při hledání správného názvu metody se vycházelo se souborů, které byly generovány souběžně s klíčovými slovy pro Robot Framework a obsahují názvy všech vygenerovaných klíčových slov. Kvůli parametrům bylo nutné, aby tyto soubory obsahovaly nejen názvy klíčových slov, ale také názvy jejich parametrů. Toho bylo docíleno nejprve úpravou knihovny *support*, která musí být překládána s argumentem `-parameters` (ten byl vložen do *pom.xml*), aby Generátor klíčových slov dokázal zjistit jména parametrů (do té doby jména parametrů známa nebyla a klíčová slova byla generována se syntetickými názvy parametrů jako `arg0`, `arg1` a podobně). Do souboru *pom.xml* dovnitř pluginu `maven-compiler-plugin` stačilo přidat tyto tři řádky:

```
<compilerArgs>
    <arg>-parameters</arg>
</compilerArgs>
```

Dalším krokem bylo zlepšení Generátoru klíčových slov, aby do souborů s názvy vygenerovaných klíčových slov ukládal také názvy jejich parametrů. Datový typ parametru zapotřebí nebyl, neboť naprostá většina parametrů byla typu `String` a jen několik z nich typu `Integer`, žádné jiné datové typy použity nebyly. Navíc nutno dodat, že uložení datového typu by nijak

nepomohlo, samotné klíčové slovo (stejně jako akceptační test, který klíčové slovo volá) nepotřebuje datový typ svých parametrů znát.

V tomto kroku byly k dispozici soubory s názvy metod (včetně názvů jejich parametrů) a také data ze souboru s testovacími případy. Následovala analýza dat z testovacích případů, aby se rozlišilo, jakého typu jsou jednotlivá data. Nejprve se rozlišovaly typy `STUDENT` (uživatelské přihlašovací jméno studenta ze známého seznamu studentů), `TEACHER` (uživatelské přihlašovací jméno učitele ze známého seznamu učitelů) a `SUBJECT` (název předmětu ze známého seznamu předmětů), neboť data byla porovnávána s databázovými konfiguračními soubory a bylo tedy snadné zjistit, jestli se jedná o studenta, učitele nebo předmět.

Kromě dat, která bylo možné porovnat s databázovými soubory, ale vstupují do metod i jiné typy parametrů. Při změně uživatelských dat se jednalo o křestní jméno, příjmení a nebo e-mailovou adresu. Protože křestním jménem i příjmením může být jakýkoli řetězec, byl zvolen přístup, že u těchto dat se přímo do testovacího případu (respektive do jeho popisu v JSON souboru s testovacími případy) napíše, jakého typu konkrétní data jsou. Před změnou vypadala data popisu testovacího případu v JSON souboru takto:

```
<code>Bart, Simpson</code>
```

a po změně takto:

```
<code>firstname:Bart, lastname:Simpson</code>
```

.

Stejný postup byl zvolen také u počtu účastníků, u e-mailové adresy a také pro rozlišení dvou parametrů typu předmět. Některé metody totiž mají za parametry dva předměty – jeden, který má student zapsán, a druhý, který se vybírá v tabulce.

Posledním rozlišovaným typem parametru je `GRADE` (známka – písmeno v rozsahu od A po F). Tento typ známka se rozlišuje regulárním výrazem, který ověřuje, že se skutečně jedná o jedno velké písmeno v daném rozsahu.

Pokud je tedy známo u všech dat napsaných v popisu testovacích případů, o jaký typ dat (parametru) se jedná, je poté možné snadno přiřadit tato data k jednotlivým parametrům podle názvu těchto parametrů. Přiřazení probíhá ověřováním, jestli název parametru v sobě obsahuje název typu dat. Například `studentUsername` v sobě obsahuje slovo *student*, takže se k němu přiřadí data s tímto typem.

Díky úspěšné implementaci vyhledání správného názvu metody a také správného přiřazení všech parametrů v daném pořadí se generování akceptačních testů stává plně automatizované.

5.3.4 Rizika analýzy parametrů

Experiment s přiřazením správných parametrů v jejich pořadí byl úspěšný, ale silně aplikačně závislý. Pokud by se jednalo o velmi odlišnou aplikaci, je velmi pravděpodobné, že žádný z typů parametru (`STUDENT`, `TEACHER`, `GRADE`, ...) by se tam nevyskytoval. V takovém případě by bylo nutné si vlastní typy parametrů doimplementovat. Druhou možností by bylo vždy psát typ parametru do popisu testovacího případu.

I v případě, že se na základě analýzy dat správně pozná, jakého jsou typu, ještě to nemusí nutně znamenat úspěch. Pokud by názvy parametrů metod v knihovně *support* byly napsané nedbale a nenesly by podstatné informace, přiřazení parametru by se tím velmi ztížilo. Avšak tato varianta není moc pravděpodobná, neboť programátor knihovny *support* si bude jistě vždy vědom důležitosti psaní čitelného kódu, když ví, že jeho knihovna bude používána i dalšími lidmi. Ale například i používání zkratek (byť jednoduše pochopitelných pro člověka) by mohlo automatizované přiřazování parametrů ztížit.

5.4 Doplnění knihovny *support*

V rámci práce byla vytvořena modifikace knihovny *support*, ve které byly provedeny následující akce:

- Do metod vykonávajících akci bylo dopsáno volání ověřovacích metod z této knihovny, které zjišťují, jestli daná akce byla provedena úspěšně. Díky tomu není nutné tyto ověřovací metody volat z Robot Frameworku.
- Doplněny metody pro změnu uživatelského jména a příjmení najednou.
- Dopsány metody bez parametru `examDate`. Doplněna podpora druhého a třetího termínu zkoušky při zakládání nového zkouškového termínu.
- Doplněna validace entit (třída `EntityValidator`). Všechny metody byly přetíženy tak, aby všechny jejich parametry byly jen základní datové typy (`String`, `Integer`, ...) a nebyly to objekty knihovny *support* (`Student`, `Teacher`, `Subject`).

5.5 Doplnění souboru s testovacími případy

JSON soubor s popisem testovacích případů vyžadoval drobné úpravy. Jednalo se o:

- Kde bylo potřeba, byl doplněn chybějící argument, s jakým uživatelem spouštět testovací případy provádějící změnu uživatelských dat (tedy změnu jména, příjmení a nebo e-mailové adresy).
- K nadřazeným elementům (testovacím sadám) přidáno jméno uživatele, s jakým mají být podřazené testovací případy spouštěny (konkrétně TS.C.01.01, TS.C.01.02, TS.C.01.03, TS.C.01.04, TS.C.01.05, TS.C.07.01, TS.C.07.02, TS.C.07.03, TS.C.07.04 a TS.C.07.05).
- Přidána čárka mezi jménem a příjmením u testovacích případů, které vykonávají změnu těchto uživatelských dat.
- U TC.C.08.01.01 změněn učitel.
- U evaluace doplněna známka do tagu `<code>`.
- Počet účastníků doplněn do tagu `<code>`.
- U TS.C.09.02 změněn učitel a předmět.

5.6 Průběh generování testů

Aktivity stručně popsané v části 4.5.1 (tedy průběh generování akceptačních testů) mají svoji detailní podobu následující:

1. Načtení všech vstupních souborů:
   - dva soubory *methods.txt* (jeden z adresáře *student* a druhý z adresáře *teacher*) obsahující názvy metod včetně parametrů (tyto soubory jsou vygenerovány společně s klíčovými slovy),
   - JSON soubor s požadavky (RQM) jen typu C,
   - JSON soubor s testovacími případy (TC) jen typu C.
2. Předzpracování JSON souborů s požadavky a s testovacími případy. Z načtené stromové struktury souborů vzniknou dva seznamy – jeden s požadavky a jeden s testovacími případy.

3. Pro každý načtený testovací případ:

   (a) Podle ID požadavku, které má každý testovací případ dané, se nalezne konkrétní požadavek, který se k testovacímu případu vztahuje. Z tohoto požadavku se vezmou informace o případu užití (jeho číslo) a úroveň kritičnosti (`criticality`). Oba tyto údaje se při generování samotného testovacího případu vkládají mezi tagy.

   (b) Podle HTML tagu `<code>` se naleznou vstupní data pro daný testovací případ. Tag `<code>` s daty se nachází buď uvnitř popisu testovacího případu nebo uvnitř popisu nadřazené testovací sady. Možná je také kombinace obojího – například uživatelské jméno se nachází v nadřazené testovací sadě, ale název předmětu se nachází v popisu konkrétního testovacího případu.

   (c) Nalezení odpovídající metody z knihovny *support*.

   (d) Seřazení argumentů, tj. skutečných hodnot parametrů, pro správné volání metody z knihovny *support*.

   (e) Uložení argumentů ve správném pořadí do souhrnné proměnné s daty.

   (f) Uložení kódu celého testovacího případu do souhrnné proměnné s testovacími případy.

4. Uložení všech dat ze souhrnné proměnné do jednotlivých souborů podle prvního čísla testovacího případu. Tedy všechna data pro testovací případy s prefixem `TC.01` jsou uložena do jednoho souboru *data_01.yaml* atd.

5. Uložení všech zdrojových kódů testovacích případů ze souhrnné proměnné do jednotlivých souborů podle prvního čísla testovacího případu. Tedy všechny kódy testovacích případů s prefixem `TC.01` jsou uloženy do jednoho souboru *tests_01.robot* atd.

6 Ověření

V první fázi generování akceptačních testů se výsledky ověřovaly pouze na bezporuchovém klonu UIS. Teprve když bylo dosaženo toho, že všechny vygenerované testy byly na bezporuchovém klonu úspěšně provedeny, se mohlo začít ověřovat také na poruchových klonech. Posledním krokem při ověřování kvality vygenerovaných akceptačních testů bylo porovnání jejich výsledků s výsledky testů z již existující manuálně psané sady akceptačních testů [9].

6.1 Konfigurace při testování

Při závěrečném ověřování akceptačních testů (konkrétní výsledky uvedeny v části 6.2) byla použita tato konfigurace:

- operační systém: Microsoft Windows 10 Pro
- verze operačního systému: 10.0.19042 Build 19042
- webový prohlížeč: Opera
- verze Opera Driveru: 89.0.4389.82
- URL: lokální UIS (*http://localhost:8080/uis*)
- headless mód prohlížeče: ne
- maximalizované okno prohlížeče: ne

Během celého vypracovávání této práce byly při spouštění testů zjištěny významné rozdíly mezi jednotlivými webovými prohlížeči a verzemi daných driverů. Některé prohlížeče (případně verze driverů) měly problémy se zadáváním symbolu „@“, jiné měly problémy s modálními okny. Je tedy nutné brát v potaz webový prohlížeč a verzi driveru. V případě jakýchkoliv problémů při testování může pomoci spuštění akceptačních testů v jiném prohlížeči nebo s jinou verzí driveru.

6.2 Porovnání se sadou stávajících akceptačních testů

Pro kompletní porovnání stávajících akceptačních testů [9] (dále pro názornost označovaných jako „manuální testy“) a vygenerovaných akceptačních

testů je nejprve nutné uvést jejich celkový počet. Manuálních akceptačních testů bylo vytvořeno celkem 122, přičemž 98 z nich ověřuje základní funkcionalitu dle případů užití a zbylých 24 se zaměřuje na speciální případy (scénáře). Oproti tomu vygenerovaných akceptačních testů je celkem 108. Toto číslo je totožné s počtem testovacích případů typu C, neboť akceptační testy jsou generovány na základě souboru s popisy testovacích případů typu C. Tedy ke každému testovacímu případu ze souboru je vygenerován právě jeden akceptační test.

K porovnání počtů testů manuálních a generovaných slouží tabulka 6.1. V ní jsou přehledně uvedené počty akceptačních testů, vždy patřící k jednomu případu užití. Jsou zde uvedené pouze ty případy užití, ke kterým byly akceptační testy vygenerovány (avšak sada manuálních akceptačních testů pokrývá všechny případy užití). To je dáno tím, že se akceptační testy generovaly jen na testovací případy typu C, které ověřují správnou funkcionalitu. Nepřipadají tedy v úvahu například případy užití, které jen kontrolují obsah tabulky, ale nevykonávají žádnou akci, tyto testy mají v seznamu všech testovacích případů označení B.

Kromě počtu testů jsou v tabulce 6.1 uvedeny také počty vstupů, respektive vstupních dat. Je důležité vzít i toto kritérium v potaz, neboť jak je z tabulky patrné, mezi manuálními a vygenerovanými akceptačními testy jsou v tomto ohledu velké rozdíly. U generovaných akceptačních testů jsou vstupní data brána z popisu testovacích případů, přičemž každý testovací případ má ve svém popisu pouze jedna vstupní data. Oproti tomu u manuálních akceptačních testů bylo těchto dat ručně sepsáno relativně mnoho.

případ užití	manuální AT		generované AT	
	počet testů	počet vstupů	počet testů	počet vstupů
UC.03	18	32	28	1
UC.04	3	12	13	1
UC.06	5	18	7	1
UC.07	2	9	6	1
UC.08	6	28	5	1
UC.10	3	5	3	1
UC.12	5	10	7	1
UC.14	9	13	9	1
UC.15	6	2	8	1
UC.16	12	2	14	1
UC.17	4	4	8	1

Tabulka 6.1: Celkové počty testů a vstupních dat

Poznámka: Protože budou dále popisovány výsledky experimentů na poruchových klonech (tedy klonech s injektovanými poruchami), znamená termín „selhání testu“ žádaný a očekávaný výsledek. Selhávající test znamená, že test injektovanou poruchu detekoval. Naopak neselhávající test znamená, že nebylo dosaženo detekce poruchy.

Tabulka 6.2 popisuje výsledky vygenerovaných akceptačních testů a porovnává je s výsledky manuálních akceptačních testů. Hodnoty v prvním sloupci jsou názvy jednotlivých klonů UIS. Ve druhém a třetím sloupci jsou uvedeny počty testů, které očekávaně selhaly při testování na daném klonu. Poslední sloupec dává do poměru počet selhávajících vygenerovaných akceptačních testů oproti počtu selhávajících manuálních akceptačních testů. Procentuální hodnota tedy vyjadřuje, v jaké míře pokrývají vygenerované akceptační testy ty manuální.

Procenta v posledním sloupci tabulky 6.2 přinášejí významnou informační hodnotu. Snadno lze vyčíst, že celkem u 6 poruchových klonů nebylo odhaleno selhání ani manuálními akceptačními testy, ani těmi vygenerovanými. Vygenerované akceptační testy navíc neodhalily žádné selhání u dalších 9 poruchových klonů, celkem tedy generované testy nerozpoznaly selhání u 15 poruchových klonů. Naproti tomu u 4 poruchových klonů selhalo více vygenerovaných testů než těch manuálních. Z toho je patrné, že některé poruchy jsou generovanými testy více pokryté a lépe otestované. U zbylých poruchových klonů se procentuální pokrytí pohybuje v rozmezí od 31 % do 93 %. Důvodem jsou zejména mnohem rozsáhlejší sady vstupních dat u manuálních testů – viz též tabulka 6.1.

Pro větší srozumitelnost zde budou vysvětleny všechny čtyři odchylky, kdy počet selhání vygenerovaných testů převyšoval počet selhání testů manuálních (hodnota vyšší než 100 %):

- `04-C0.H0.M0.L1_S_S_04` – Na daný případ užití (UC.04) obsahuje manuální sada 3 testy (z toho je 1 z nich negativní) a vygenerovaná sada 13 testů. Z manuální sady selhaly všechny testy kromě toho negativního (tedy 2 testy) a z vygenerované sady všech 13. Díky vyššímu počtu vygenerovaných akceptačních testů pro případ užití UC.04 vyšel poměr ku počtu selhávajících manuálních testů na 650 %.

- `19-C0.H1.M0.L0_S_S_10` – Opět se týká případu užití (UC.04), pro který obsahuje manuální sada 3 testy (z toho je 1 z nich negativní) a vygenerovaná sada 13 testů. Z manuální sady selhaly všechny testy kromě toho negativního (tedy 2 testy) a z vygenerované sady jich selhalo 9. Díky vyššímu počtu vygenerovaných testů tedy vyšel poměr ku manuálním testům na 450 %.

- `20-C0.H1.M0.L0_T_S_02` – Týká se případu užití UC.12, ke kterému patří z manuální sady testů 5 (přičemž u 2 testů se daná akce zrušení zkouškového termínu nakonec neprovede) a z vygenerované sady testů 7. Z manuální sady selžou 3 testy k UC.12 a 1 scénářový test (celkem 4 testy) a z vygenerované sady selžou testy všechny (7 testů). Kvůli tomu vyjde poměr na 175 %.

- `21-C0.H1.M0.L0_T_S_04` – Opět se týká případu užití UC.12, pro který obsahuje manuální sada 5 testů (z toho 2 nakonec testovanou akci neprovedou) a vygenerovaná sada 7 testů. Z manuální sady selže testů celkem 6 (3 z UC.12, 2 z UC.13 a 1 scénářový). Z vygenerované sady selžou všechny testy k UC.12, tedy všech 7 testů. Sada neobsahuje scénářové testy ani testy k UC.13. Díky zmíněným počtům testů nakonec vyjde poměr nad 100 %, konkrétně na 117 %.

Pro ještě lepší porovnání manuálních akceptačních testů a těch vygenerovaných byla vytvořena tabulka 6.3. U obou sad testů obsahuje každý akceptační test tag, který označuje, jaký případ užití daný test pokrývá. Díky tomuto tagu bylo možné sestavit tuto tabulku a zjistit, jak moc srovnatelné obě sady testů jsou. V každé buňce tabulky mohou být použity dva symboly „č“ a „V“, mezi kterými je symbol lomítka. Lomítko odděluje dvě sady testů od sebe – na levé straně od lomítka je indikace pro manuální sadu akceptačních testů a na straně pravé od lomítka se vyskytuje indikace pro vygenerované akceptační testy. Symbol „č“ (jako „část“) značí, že v dané sadě testů pro daný případ užití jen část z nich selhala. Oproti tomu symbol „V“ (jako „všechny“) označuje, že všechny testy v dané sadě, které se týkají daného případu užití, selhaly.

Pro názornost budou níže vysvětleny všechny kombinace hodnot, které by se v tabulce mohly vyskytnout:

- prázdná buňka – žádné testy ani z jedné sady neselhaly,

- č/ – část testů týkajících se daného UC z manuálně psané sady selhala,

- /č – část vygenerovaných testů týkajících se daného UC selhala,

- V/ – všechny testy týkající se daného UC z manuálně psané sady selhaly,

- /V – všechny vygenerované testy týkající se daného UC selhaly,

- č/V – část testů týkajících se daného UC z manuálně psané sady selhala a také všechny vygenerované testy týkající se daného UC selhaly,

- V/č – všechny testy týkající se daného UC z manuálně psané sady selhaly a také část vygenerovaných testů týkajících se daného UC selhala,
- č/č – část testů týkajících se daného UC z manuálně psané sady selhala a také část vygenerovaných testů týkajících se daného UC selhala,
- V/V – všechny testy týkající se daného UC z manuálně psané sady selhaly a také všechny vygenerované testy týkající se daného UC selhaly.

Z tabulky 6.3 je patrné, že jen v šesti případech neselhaly žádné generované testy, jen ty manuální. U těchto šesti případů se jedná o poruchy UIS, které jsou odhalitelné pouze při zadání nesprávných vstupních dat (například počet účastníků zkoušky byl nastaven na -1) nebo dokonce nezadání žádných vstupních dat. To ale představuje negativní testy, které jsou v seznamu testovacích případů v kategorii D. Případně některé z poruch jsou odhalitelné jen při ověřování dat u druhého typu uživatele (například učitel zruší zkoušku a součástí ověření testu je také přihlášení studenta a ověření, že v jeho seznamu zkoušek se zrušená zkouška nevyskytuje).

V prvním případě (špatná vstupní data) je zřejmé, proč vygenerované akceptační testy poruchu neodhalily – vygenerované testy totiž ověřují pouze správnou funkcionalitu, tedy vždy zadávají pouze validní vstupní hodnoty. Ve druhém případě (poruchy zjistitelné jen při přihlášení druhého typu uživatele) se jedná o takové poruchy, které by vygenerované akceptační testy byly schopné odhalit, pokud by ověřování výsledku testu probíhalo opravdu důsledně. Zde je tedy na místě zvážit, jestli by dané ověření nemělo být součástí knihovny *support*, tedy konkrétně součástí těch metod, které generované akceptační testy přímo využívají.

Ve všech ostatních případech (dle ostatních buněk v tabulce 6.3) selhaly ohledně daného případu užití alespoň některé testy z obou sad zároveň. Díky tomu je patrné, že generované akceptační testy jsou srovnatelné s těmi manuálními.

Poznámka: Detailní popisy jednotlivých poruchových klonů včetně charakteristik injektovaných poruch lze nalézt na stránkách projektu TbUIS (*https://projects.kiv.zcu.cz/tbuis/web/page/download*).

klon UIS	manuální AT	generované AT	poměr
C0.H0.M0.L0_ALL_OK	0	0	–
01-C0.H0.M0.L1_S_S_07	0	0	–
02-C0.H0.M0.L1_S_S_03	0	0	–
03-C0.H0.M0.L1_S_S_02	8	0	0 %
04-C0.H0.M0.L1_S_S_04	2	13	650 %
05-C0.H0.M0.L1_S_S_09	1	0	0 %
06-C0.H0.M0.L1_T_S_05	26	11	42 %
07-C0.H0.M1.L0_S_S_08	0	0	–
08-C0.H0.M1.L0_S_S_12	0	0	–
09-C0.H0.M1.L0_T_S_10	1	0	0 %
10-C0.H0.M1.L0_T_S_11	1	0	0 %
11-C0.H0.M1.L0_S_S_05	1	0	0 %
12-C0.H0.M1.L0_T_S_08	14	10	71 %
13-C0.H1.M0.L0_G_T_D_01	28	22	79 %
14-C0.H1.M0.L0_D_U_01	21	9	43 %
15-C0.H1.M0.L0_D_U_02	21	9	43 %
16-C0.H1.M0.L0_T_S_07	3	0	0 %
17-C0.H1.M0.L0_T_S_09	1	0	0 %
18-C0.H1.M0.L0_S_S_06	0	0	–
19-C0.H1.M0.L0_S_S_10	2	9	450 %
20-C0.H1.M0.L0_T_S_02	4	7	175 %
21-C0.H1.M0.L0_T_S_04	6	7	117 %
22-C1.H0.M0.L0_S_S_11	16	5	31 %
23-C1.H0.M0.L0_T_S_03	4	0	0 %
24-C1.H0.M0.L0_T_S_06	24	8	33 %
25-C1.H0.M0.L0_S_S_01	38	31	82 %
26-C1.H0.M0.L0_T_S_01	59	41	69 %
27-C1.H0.M0.L0_U_D_01	4	0	0 %
28-C2.H2.M1.L0_M_CR	86	80	93 %

Tabulka 6.2: Výsledky

klon	UC										
	03	04	06	07	08	10	12	14	15	16	17
00											
01											
02											
03											
04		č/V									
05											
06						č/V					č/V
07											
08											
09								č/			
10								č/			
11											
12										č/č	
13									V/V	V/V	
14								č/V			
15								č/V			
16								č/			
17								č/			
18											
19		č/č									
20							č/V				
21							č/V				
22					č/V						
23							č/				
24											č/V
25		V/V	V/V	V/V	č/V		č/				
26						V/V	V/V	V/V		V/V	č/V
27											
28		V/V	V/V	V/V	č/V	V/V	V/V	V/V	V/V	V/V	č/V

Tabulka 6.3: Porovnání na jednotlivých případech užití

7 Diskuze výsledků a doporučení

Součástí zadání (konkrétně bod 2) byla analýza dostupných informací, návrh metodiky a také v případě nedostačujících dat navržení takové sady informací, která generování akceptačních testů umožní. Během vypracovávání práce bylo však zjištěno, že žádná další sada informací potřeba není. Tedy že pro vygenerování akceptačních testů stačí knihovna *support*, popisy testovacích případů a vygenerovaná klíčová slova. Avšak je nutné vzít v potaz, že báze znalostí projektu TbUIS je poměrně rozsáhlá. U běžných komerčních projektů takto rozsáhlá být nemusí, takže u jiného projektu by se mohlo stát, že veškerá báze znalostí nebude dostačující a bude muset být doplněna o další informace.

Jedním z efektů této práce byla také celková optimalizace procesu. Na začátku experimentování se totiž používal generátor wrapperů a současně klíčových slov, přičemž všechny vygenerované soubory (wrappery a klíčová slova) byly využívány. Dále během vypracovávání práce bylo však zjištěno, že když se provedou menší úpravy v knihovně *support*, tak wrappery nakonec vůbec nejsou zapotřebí. Proces tedy byl optimalizován tím, že byl kompletně odstraněn krok generování wrapperů a jejich využívání.

Další formou optimalizace bylo také vylepšení generátoru klíčových slov. Jen díky přidání pouhých čtyř řádků do souboru `pom.xml` knihovny *support* bylo dosaženo toho, že se klíčová slova generují s totožnými názvy parametrů dle knihovny *support* (do té doby byla klíčová slova generována se syntetickými názvy jako například `arg0`). Toto vylepšení významně pomohlo při vypracovávání této práce, avšak přínos tohoto vylepšení může být použit i v dalších experimentech s projektem TbUIS a knihovnou *support*.

Tato diplomová práce ověřila, že je možné generovat akceptační testy pro Robot Framework z popisu testovacích případů. Avšak TbUIS (respektive využívaná knihovna *support*) je speciální, neboť již obsahuje orákulum. Metoda z knihovny *support* tedy nejen provede danou akci, ale také rovnou ověří, jestli se daná akce podařila. Ale v reálné praxi není existence orákula pravděpodobná (ve skutečnosti totiž prakticky představuje *N-version programming*). V takovém případě by bylo zapotřebí, aby v popisu testovacího případu bylo napsáno, jaký je očekávaný výsledek testu. Tento požadavek je ovšem relativně snadno (narozdíl od naprogramovaného orákula) splnitelný. Tedy pokud by orákulum nebylo obsaženo v knihovně *support*, muselo by

být zahrnuto v popisu testovacího případu.

Důležitým kritériem při posuzování výsledků této práce je porovnání generovaných akceptačních testů s manuálními akceptačními testy, vytvořenými v rámci diplomové práce [9]. Generované akceptační testy jsou srovnatelné s těmi manuálními, neboť v rámci možností poskytují obdobné výsledky. Manuální akceptační testy ale mají několik výhod. První výhodou je, že manuálních akceptačních testů je celkově o 14 více než těch generovaných, přičemž 24 z nich se zaměřuje na konkrétní speciální případy (scénáře). Druhou jejich výhodou je, že každý manuální akceptační test má svou (relativně velkou) sadu vstupních dat. Počet vstupních dat v jedné sadě se u jednotlivých testů pohybuje od 2 do 32. Oproti tomu pro každý vygenerovaný akceptační test existují pouze jedna vstupní data. Třetí výhodou manuálních akceptačních testů je fakt, že obsahují také negativní testy, jejichž cílem je vynutit očekávanou reakci na nekorektní aktivity uživatele. Tedy například cíleně zadat nesprávná vstupní data a poté ověřit, že se systém chová správně i při zadání nesprávných dat. Bližší porovnání s konkrétními výsledky je uvedeno v kapitole 6.

Je ovšem nutno zdůraznit, že tato práce měla za cíl ověřit, jestli je způsob generování akceptačních testů vůbec možný za předpokladu dosažení alespoň srovnatelných výsledků. Z porovnání z předchozí kapitoly je zřejmé, že manuální testy byly většinou úspěšnější než testy generované. Ovšem je zcela reálné, aby byl Generátor akceptačních testů ještě vylepšený. Například pokud by popisy testovacích případů byly obsáhlejší a obsahovaly by více vstupních dat, manuální akceptační testy by ztratily jednu ze svých výhod. Dále pokud by knihovna *support* poskytovala také metody, které by byly určeny pro negativní testy, bylo by reálné rozšířit generátor akceptačních testů natolik, aby zvládl generovat také negativní akceptační testy. Díky tomu by manuální testy ztratily další svou výhodu. Poslední výhodou manuálních testů je jejich větší počet, ovšem počet generovaných akceptačních testů je totožný s počtem testovacích případů ve vstupním souboru. Řešením je tedy buď rozšířit počet testovacích případů v souboru s jejich popisy nebo přidat podporu pro další typ testovacích případů (například podporu pro negativní testy).

Je třeba zmínit i další významný přínos této práce. Tím je praktické ověření a četné realizované a ověřené návrhy na vylepšení knihovny *support*. Tyto poznatky výrazně vylepšují metodiku přípravy knihovny *support*, ať už bude používána pro ručně psané akceptační testy nebo by sloužila i pro generované akceptační testy.

7.1 Doporučení pro knihovnu *support*

Poznámka: Některé informace uvedené v této části již byly uvedeny dříve. Zde se budou opakovat z důvodu přehledného shrnutí dosažených výsledků.

Knihovna *support* v projektu TbUIS je důsledně zpracována a tak nebylo zapotřebí významných změn. Samotná knihovna zajišťuje ovládání aplikace UIS (s použitím nástroje Selenium WebDriver), načtení databázových souborů a konfiguraci UIS. Díky tomu se generované akceptační testy mohou věnovat pouze akcím prováděným nad aplikací UIS a nemusí se zabývat nízkoúrovňovými záležitostmi (například načtení konfiguračních a databázových souborů, spuštění driveru, přechodu na konkrétní URL apod.).

Důležitou částí knihovny *support* je balík `uis.support.services`. V něm se nacházejí veškeré vysokoúrovňové metody, které zajišťují provedení hlavních úkolů v aplikaci UIS. Metody v tomto balíku jsou členěny do tří dalších balíků (`login`, `student` a `teacher`). V balíku `login` je například metoda pro přihlášení uživatele. Balík `student` obsahuje například metodu pro zapsání předmětu či přihlášení se na zkouškový termín. V balíku `teacher` je k dispozici například metoda pro ohodnocení studenta z konkrétního předmětu a zkouškového termínu. Celý balík `uis.support.services` je vygenerovanými akceptačními testy přímo využíván a jeho existence velmi usnadnila automatizované generování testů. Prvním doporučením je tedy zajistit, aby podpůrná knihovna obsahovala podobný balík, který bude poskytovat obdobné vysokoúrovňové metody pro ovládání aplikace.

Během vypracovávání této práce bylo zjištěno, že ve výše zmíněném balíku `uis.support.services` mají všechny metody parametry tvořené konkrétními objekty (například `Student`, `Subject`, `Teacher`). Avšak pro jednoduché propojení vstupních dat, která se načítají z textového souboru s popisy testovacích případů, a metod z knihovny *support* bylo výhodnější se objektům vyhnout. Z toho důvodu byla nakonec knihovna *support* (respektive jen balík `uis.support.services`) rozšířena o metody, které mají parametry jen ze základních datových typů (`String`, případně `Integer`). Znamenalo to pouze přetížit již existující metody a také dodat do knihovny *support* třídu `EntityValidator` (převzatou z práce [9]). Tato třída poskytuje metody, které z řetězce (například uživatelského jména) vytvoří instanci konkrétní třídy (například `Student`). Avšak pokud by knihovna *support* byla vytvořena důsledně, přetěžování metod a doplnění třídy `EntityValidator` by nebylo nutné, neboť obojí by již bylo součástí této knihovny. Druhým doporučením je tedy zkontrolovat, že poskytované vysokoúrovňové metody pro ovládání aplikace mají parametry tvořené pouze základními datovými typy (`String`, `Integer` apod.).

Posledním doporučením, které se týká výše zmiňovaného poskytování vysokoúrovňových metod, je důraz na jejich množství a také pojmenování (včetně pojmenování parametrů). Akceptační testy se generují na základě popisu testovacích případů a to v poměru 1:1, tedy na základě jednoho testovacího případu se vygeneruje jeden akceptační test. Proto je nutné, aby podpůrnou knihovnou poskytované metody odpovídaly akcím, které se mají vykonat na základě jednotlivých testovacích případů. Například pokud existuje testovací případ, ve kterém si student má zapsat předmět, musí existovat odpovídající metoda v knihovně *support*. Dále by tato metoda měla nést název, skládající se ze stejných slov, které jsou použité v popisu testovacího případu. Špatným příkladem je kombinace názvu metody `participateInSubject` a popisu testovacího případu „signs up to teach a subject - common change“ – zde je jasně vidět, že nejsou použita stejná slova (*participate* versus *teach*), což značně znesnadňuje nalezení odpovídající metody z knihovny *support* pro daný testovací případ.

7.2 Doporučení pro popisy testovacích případů

Popisy testovacích případů v souboru ve formátu JSON jsou napsané podrobně a zahrnují mnoho důležitých informací.

Prvním doporučením pro jejich vylepšení je v popisu testovacího případu uvádět vstupní data, se kterými má být proveden. Data lze uvést buď přímo v popisu konkrétního testovacího případu nebo také v popisu nadřazené testovací sady (v takovém případě se data použijí pro celou sadu). Je také důležité dbát na to, aby byla jednotlivá data od sebe oddělena (například čárkou), aby mohla být přímo použita jako parametr. Aktuálně se data uvádějí v atributu `description` a nacházejí se uvnitř tagu `<code>`, což vypadá například takto:

```
"description": "<ul><li>already many subjects
   enrolled, no with enrolled exam date</li><li><
   code>Blue, Database Systems</code></li><li>the
   message TXT_ALERT_SUCCESS is displayed</li><ul>"
```

Avšak pro usnadnění generování testů by bylo výhodnější, aby vstupní data nebyla v atributu `description`, neboť tento atribut je velmi obsáhlý. Bylo by vhodnější, aby byla vstupní data uváděna v samostatném atributu (například s názvem `data`). Ideálně také s rozlišením, o jaký typ dat se jedná.

Atribut se vstupními daty by tedy mohl vypadat například takto:

```
"data": [
    {
    "student":"Blue",
    "subject":"Database Systems"
    }
]
```

V tuto chvíli každý testovací případ obsahuje pouze jednu sadu vstupních dat, ale samozřejmě by mohl obsahovat sad i více, proto v návrhu obsahuje atribut `data` pole objektů. Generované akceptační testy jsou *data-driven* testy, tedy rovnou navržené tak, aby byly schopné přijímat sady vstupních dat. Stačilo by pouze upravit Generátor akceptačních testů tak, aby byl schopný zpracovávat soubor s popisy testovacích případů i při možnosti uvádění více sad vstupních dat u testovacích případů. Následně by jen Generátor ukládal všechny sady vstupních dat do souborů s daty, přičemž zdrojové kódy testů by zůstaly nezměněny.

Druhé doporučení přímo souvisí s knihovnou *support*. V předchozí sekci (7.1) je zmíněno doporučení klást důraz na pojmenovávání metod v této knihovně. Při generování testů se totiž na základě popisu testovacího případu (a nadřazených testovacích sad) hledá odpovídající metoda z knihovny *support*. Proto je důležité, aby se v popisu testovacího případu a nadřazených testovacích sad vyskytovalo co nejvíce stejných slov a naopak nepoužívala se vůbec synonyma, která generování jen znesnadní.

8 Závěr

Byla vytvořena modulární aplikace Generátor akceptačních testů, která na základě několika vstupních souborů dokáže vygenerovat soubory se zdrojovými kódy akceptačních testů pro Robot Framework a také soubory se vstupními daty pro tyto testy.

Nejprve bylo nutné prostudovat diplomovou práci *Akceptační testování v projektu TbUIS* [9], nástroj Robot Framework a analyzovat bázi znalostí projektu TbUIS.

Z výsledků prvotních experimentů bylo zjištěno, že informace z báze znalostí projektu jsou dostačující a tedy není zapotřebí doplňovat další informace. Bylo tedy možné vytvořit funkční Generátor akceptačních testů a při jeho tvorbě byly současně navrženy i změny v metodice vytváření dokumentačních souborů.

Posledním krokem bylo ověření vygenerovaných akceptačních testů (a jejich vstupních dat) na aplikaci UIS. Toto ověření proběhlo s využitím celé škály možností TbUIS, tj. na bezporuchovém klonu aplikace a také na všech aktuálně dostupných poruchových klonech (celkem 28). Výsledky vygenerovaných akceptačních testů byly podrobně a z několika hledisek porovnány s výsledky manuálně psaných akceptačních testů se závěrem, že oba druhy testů dávají srovnatelné výsledky.

Práce splnila všechny body zadání. Bylo úspěšně prokázáno, že je možné vygenerovat akceptační testy pro netriviální webovou aplikaci na základě vybraných dokumentačních vstupních souborů a s využitím podpůrné knihovny *support*. Je třeba zdůraznit, že se nejedná o generickou úlohu, která by šla přímo v tomto stavu převzít, použít na jinou aplikaci a očekávat podobně úspěšné výsledky. Tato práce však dokázala, že tento směr automatizovaného generování akceptačních testů je reálně proveditelný a že by mohlo být možné vytvořit obecnou metodiku, použitelnou pro více různých aplikací.

Seznam zkratek

HTML	Hypertext Markup Language – značkovací jazyk
JSON	JavaScript Object Notation – formát zápisu dat
RF	Robot Framework – nástroj pro automatizaci testů
TbUIS	Testbed University Information System – projekt pro účely výzkumu testovacích metod
UIS	University Information System – webová aplikace
XML	eXtensible Markup Language — obecný značkovací jazyk
YAML	YAML Ain't Markup Language – formát pro serializaci strukturovaných dat

Literatura

[1] Bisht, S. *Robot Framework Test Automation.* Packt Publishing Ltd., 2013. ISBN 978-1-78328-303-3.

[2] Bureš, M. et al. *Efektivní testování softwaru.* Grada Publishing, 2016. ISBN 978-80-247-5594-6.

[3] Bureš, M. – Herout, P. – Ahmed, B. S. *Open-source Defect Injection Benchmark Testbed for the Evaluation of Testing* [online]. IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020. [cit. 2021/05/03]. Dostupné z: `https://projects.kiv.zcu.cz/tbuis/web/files/uis/others/icst2020.pdf`.

[4] Poubová, J. *Generování automatizovaných funkcionálních testů.* Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2019. Vedoucí práce Pavel Herout.

[5] *Robot Framework Introduction* [online]. Robot Framework Foundation, 2021. [cit. 2021/03/22]. Dostupné z: `https://robotframework.org/`.

[6] *Robot Framework User Guide* [online]. Robot Framework Foundation, 2021. [cit. 2021/03/22]. Dostupné z: `http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html`.

[7] Roudenský, P. – Havlíčková, A. *Řízení kvality softwaru: průvodce testováním.* Computer Press, 2013. ISBN 978-80-251-3816-8.

[8] *TbUIS – About the Project* [online]. ReliSA – Reliable Software Architectures, KIV, ZČU, 2021. [cit. 2021/03/22]. Dostupné z: `https://projects.kiv.zcu.cz/tbuis/web/page/about`.

[9] Vais, R. *Akceptační testování v projektu TbUIS.* Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2020. Vedoucí práce Pavel Herout.

A Uživatelská příručka

V této uživatelské příručce je popsán návod pro překlad a spuštění Generátoru klíčových slov a Generátoru akceptačních testů včetně návodu na spuštění vygenerovaných testů. Jednotlivé kroky na sebe logicky navazují – nejprve je nutné vygenerovat klíčová slova, která se potom použijí pro vygenerování akceptačních testů, které se spustí. Avšak v odevzdaném ZIP souboru existují již soubory vytvořené při překladu a také jsou již jednotlivé kroky provedené. Díky tomu není nutné provést všechny kroky v daném pořadí, ale lze provést jakýkoli krok i bez provedení těch předchozích (je možné například rovnou spustit vygenerované akceptační testy).

A.1 Předpoklady

K úspěšnému překladu a spuštění aplikace a všech jejích částí jsou zapotřebí:

- Java 8 (testováno na verzi 1.8.0_181),
- Maven (testováno na verzi 3.6.3),
- nainstalovaná podpůrná knihovna *support* – ta se nachází v adresáři `Aplikace_a_knihovny/TbUIS-support` a pro její instalaci stačí v tomto adresáři pouze do příkazové řádky zadat příkaz:

```
mvn install
```

A.2 Generování klíčových slov

Celá aplikace Generátor klíčových slov se nachází uvnitř adresáře s názvem `Aplikace_a_knihovny/generator-keywords`.

A.2.1 Překlad

Pro překlad aplikace je nutné uvnitř výše zmíněného kořenového adresáře zadat do příkazové řádky příkaz:

```
mvn install
```

A.2.2 Spuštění

Pro spuštění aplikace je nutné přesunout vytvořený JAR soubor s názvem `uis-robotkeywordgenerator-1.0.0-jar-with-dependencies.jar` z adresáře `target` do kořenového adresáře a poté do příkazové řádky zadat příkaz:

```
java -jar uis-robotkeywordgenerator-1.0.0-jar-with-
    dependencies.jar
```

Pokud aplikace skončila úspěšně, vznikl v aktuálním adresáři podadresář `out-keywords`, ve kterém se nacházejí vygenerovaná klíčová slova.

Pokud má Generátor akceptačních testů použít vygenerovaná klíčová slova, je nutné vzít obsah adresáře `out-keywords/uis/support/services` (tedy adresáře `login`, `student` a `teacher`) a vložit jej do aplikace Generátor akceptačních testů, konkrétně do adresáře `generator-acceptance/src/test/robotframework/keywords`. V odevzdávaném ZIP souboru se vygenerovaná klíčová slova v daném adresáři již nacházejí, takže není zapotřebí tento krok udělat.

A.3 Generování akceptačních testů

Aplikace Generátor akceptačních testů se nachází uvnitř adresáře s názvem `Aplikace_a_knihovny/generator-acceptance`.

A.3.1 Překlad

Pro překlad aplikace je nutné uvnitř výše zmíněného kořenového adresáře zadat do příkazové řádky příkaz:

```
mvn install
```

A.3.2 Spuštění

Pro spuštění aplikace je nutné přesunout vytvořený JAR soubor s názvem `uis-generator-acceptance-1.0.0-jar-with-dependencies.jar` z adresáře `target` do kořenového adresáře a poté do příkazové řádky zadat příkaz:

```
java -jar uis-generator-acceptance-1.0.0-jar-with-
    dependencies.jar <config-file-path>
```

Parametr `<config-file-path>` označuje jediný parametr aplikace. Tento parametr je nepovinný a nese cestu k souboru s konfigurací (*generator-configurations.txt*). Pokud parametr není uveden, použije se výchozí hodnota pro cestu `configuration/generator-configurations.txt`.

Pro konfiguraci Generátoru akceptačních testů slouží soubor *generator-configurations.txt*, který se nachází v adresáři `configuration`. V tomto konfiguračním souboru je možné nastavit cesty k těmto vstupním souborům:

- `configurationDirectory` – cesta k adresáři obsahujícímu všechny konfigurační a datové soubory pro UIS,
- `studentMethodsFile` – cesta k souboru *methods.txt* s hlavičkami metod pro přihlášeného studenta,
- `teacherMethodsFile` – cesta k souboru *methods.txt* s hlavičkami metod pro přihlášeného učitele,
- `requirementsFile` – cesta k JSON souboru s požadavky,
- `testCasesFile` – cesta k JSON souboru s testovacími případy.

Pokud aplikace proběhla úspěšně, akceptační testy se vytvořily v adresáři `generator-acceptance/src/test/robotframework/acceptance` a vstupní data v adresáři `generator-acceptance/src/test/robotframework/data`.

A.4 Spuštění akceptačních testů

A.4.1 Předpoklady

Před spuštěním akceptačních testů je nutné zkontrolovat, že se v kořenovém adresáři (`Aplikace_a_knihovny/generator-acceptance`) nachází složka `configuration` a v ní soubor `configurations.txt`. V tomto souboru je uvedená veškerá konfigurace pro testování UIS, kterou akceptační testy použijí. Před samotným spuštěním testů je zapotřebí zkontrolovat konfiguraci uvedenou v tomto souboru. Zejména se jedná o výběr webového prohlížeče a jeho nastavení (cesta k driveru nebo možnost, že se driver automaticky stáhne). Také lze nastavit, na jaké URL adrese se nachází testovaná aplikace UIS (jestli se použije lokální instance UIS nebo veřejná). Pro lokální spuštění UIS je návod na *https://projects.kiv.zcu.cz/tbuis/web/page/uis#run-uis*.

A.4.2 Spuštění

S pomocí nástroje Maven a jeho pluginu pro Robot Framework, který je součástí aplikace Generátor akceptačních testů, je možné vygenerované akceptační testy spouštět. Stačí do příkazové řádky uvnitř kořenového adresáře aplikace (`Aplikace_a_knihovny/generator-acceptance`) zadat příkaz:

```
mvn org.robotframework:robotframework-maven-plugin:1.5.1:run
```

Po zadání příkazu se spustí webový driver (dle konfigurace) a začne se vykonávat celá sada akceptačních testů. Po jejich skončení vygeneruje Robot Framework soubory s reportem a logy, tyto soubory se nacházejí ve složce `generator-acceptance/target/robotframework-reports`.

Pokud je žádoucí provést jen určitou část akceptačních testů, je možné spustit jen podmnožinu testů zvolenou dle tagu pomocí přidaného parametru. Například pokud chceme spustit jen akceptační testy s tagem `UC.14` (ty se týkají jen případu užití s číslem 14), zadáme tento příkaz:

```
mvn org.robotframework:robotframework-maven-plugin:1.5.1:run
    -Dincludes=UC.14
```

B Obsah ZIP souboru

- `Text_prace`
  - `tex`

    Tento adresář obsahuje všechny zdrojové soubory textu diplomové práce.
  - `DP-Poubova.PDF`

    PDF soubor s textem diplomové práce.
- `Poster`

  Obsahuje oba soubory posteru (`.pub` a `.pdf`).
- `Aplikace_a_knihovny`
  - `generator-acceptance`

    Obsahuje všechny zdrojové kódy i přeložené soubory včetně spustitelného JAR souboru. Dále obsahuje také adresář s konfiguračními soubory a všechny vstupní soubory.
  - `generator-keywords`

    Obsahuje všechny zdrojové kódy i přeložené soubory včetně spustitelného JAR souboru.
  - `TbUIS-support`

    Obsahuje všechny zdrojové kódy i přeložené soubory, dále také obsahuje všechny databázové a konfigurační soubory.
- `Vstupni_data`

  Obsahuje pouze vysvětlující soubor *Readme.txt*.
- `Vysledky`

  Tento adresář obsahuje veškeré logy a reporty Robot Frameworku. Adresář je strukturovaný do podadresářů pojmenovaných jako názvy poruchových klonů. V každém podadresáři se nacházejí výsledné reporty Robot Frameworku (vždy pro konkrétní poruchový klon) vygenerované při testování generovanými akceptačními testy.
- `Readme.txt`

  V tomto souboru se nachází popis obsahu ZIP souboru.