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

Bakalářská práce

Generování automatizovaných funkcionálních testů

Plzeň 2019 Jitka Poubová

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 2. května 2019

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 research in the generation of automatized functional tests. It utilizes the graph of transitions and states of the tested application. For this research a web application already exists, serving for System Under Test (SUT). It is possible to intentionally inject errors inside this web application and hereby verify the quality of tests. The graph of the tested application is made in an application called Oxygen which can also generate test cases. According to these test cases it is possible to manually create functional tests. The aim of this thesis is the possibility of automatic test generation according to these test cases. This would eliminate the manual work of testers as well as it would increase the test coverage of given application.

Abstrakt

Práce se zabývá výzkumem v oblasti generování automatizovaných funkcionálních testů. Využívá graf přechodů a stavů testované aplikace. Pro tento výzkum již existuje webová aplikace, sloužící jako System Under Test (SUT), do které lze záměrně zanést chyby a tím ověřovat kvalitu testů. Graf testované aplikace je vytvořen v aplikaci Oxygen, která také dokáže vygenerovat testovací případy, dle kterých lze manuálně vytvářet funkcionální testy. Cílem práce je automatické vygenerování těchto testů dle zmíněného grafu, což by odstranilo manuální práci testerů a zároveň by zvýšilo protestovanost dané aplikace.

Obsah

1 Úvod

Na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni (KIV ZČU) byla úspěšně obhájena diplomová práce *Aplikace s možností injekce chyb pro ověřování kvality testů*. Jejím výsledkem je pseudorealistická webová aplikace, představující univerzitní informační systém – University Information System (UIS). Předpokládané použití této aplikace je pro výzkum v oblasti testování, a z toho důvodu do ní lze zanést různé softwarové chyby. Tímto způsobem je možné vytvořit poruchové klony UIS, na kterých lze jednoduše ověřit kvalitu a pokrytí testů.

K této aplikaci bylo též vytvořeno velké množství funkcionálních testů. Jsou volně dostupné, stejně jako jejich podpůrné knihovny, a společně s UIS tvoří projekt Testbed UIS.

Existuje nástroj Oxygen, vyvíjený na Fakultě elektrotechnické Českého vysokého učení technického (FEL ČVUT), ve kterém lze vytvořit graf přechodů a stavů pro testovanou aplikaci. Z tohoto grafu je možné automaticky vygenerovat testovací případy, kterými se může řídit tester při manuálním testování aplikace.

Dále byla na KIV úspěšně obhájena bakalářská práce *Generování testovacích datasetů*. Dokázala na triviální webové aplikaci, že je možné s využitím výše popsaných možností Oxygenu automaticky generovat testy.

Cílem této práce je využít všechny výše zmíněné dostupné nástroje, aplikace a postupy a vytvořit program, který dokáže samostatně vygenerovat kompletní funkcionální testy. Ty metodou procházení cest mají zcela pokrýt všechny existující cesty v testované aplikaci. Vstupem programu bude graf aplikace a testovací případy, obojí vytvořené v Oxygenu, a výstupem spustitelné funkcionální testy. Testovanou aplikací bude UIS.

2 Testovaná aplikace a používané nástroje

2.1 TbUIS

2.1.1 Obecné informace

Testbed University Information System (TbUIS) je projekt, jenž se skládá z webové aplikace University Information System (UIS) a z velkého množství funkcionálních testů včetně jejich podpůrných knihoven. Součástí tohoto projektu je také *Error Seeder* – nástroj pro zanášení defektů do zdrojového kódu.

Webová aplikace UIS je vytvořená za účelem výzkumu v oblasti testování. Z toho důvodu je její důležitou vlastností možnost zanést do ní díky výše zmíněnému *Error Seederu* různé softwarové chyby a tím vytvářet poruchové klony, na kterých se následně snadno ověří kvalita testů a jejich pokrytí. Aplikace byla vyvinutá v rámci diplomové práce *Aplikace s možností injekce chyb pro ověřování kvality testů* na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni (KIV ZČU) [4]. Tato práce byla úspěšně obhájena v roce 2018. Na ní dále navazuje aktuální diplomová práce *Rozhraní pro administraci aplikace s možností injekce chyb*.

2.1.2 Webová aplikace UIS

Jak již název UIS napovídá, jedná se o napodobeninu univerzitního informačního systému. Proto jsou zde tyto tři hlavní entity – student, učitel a předmět. Uživatel se může přihlásit jako student nebo jako učitel a podle toho existují různé případy užití. Student si může například zapsat předmět či přihlásit se na zkoušku, učitel má například možnost zrušit zkouškový termín nebo ohodnotit studenta. Všechny možné případy užití jsou popsány v tabulce 2.1.

Celý program je napsán v Javě s využitím technologií Java Server Pages (JSP), Bootstrap, Hibernate a Spring. Program využívá relační databázi. Z důvodu rozličnosti testování obsahuje aplikace mnoho různých běžných webových elementů jako jsou tlačítka, modální okna, zaškrtávací políčka, výběrové seznamy, tabulky nebo menu [6].

UIS je dobře dokumentovaná aplikace, připravená pro automatické funk-

cionální testování – každý element má vlastní ID, každá operace oznámí svůj výsledek a také lze kdykoli obnovit databázi do původního stavu. Na úvodní stránce jsou též odkazy na nápovědy – nastavení databáze či různá omezení daná specifikací. Pro usnadnění testování jsou v aplikaci již přednastavena testovací data v dostatečné míře a rozmanitosti, takže testování může začít okamžitě. Přednastavená data existují v různých kombinacích, aby pokrývala co nejširší spektrum nastavení a testy tak mohly ověřovat i speciální krajní situace.

Na první pohled se může zdát, že je to opravdu jednoduchá aplikace, ale po nakreslení grafu přechodů a stavů bylo zjištěno, že obsahuje 92 různých stavů a dále 120 různých přechodů mezi nimi (uvedené počty odpovídají grafu, který byl předpřipraven na začátku této práce).

2.1.3 Případy užití

Existují celkem tři typy uživatelů – nepřihlášený, přihlášený student a přihlášený učitel. Případy užití pro jednotlivé uživatele jsou v tabulce 2.1.

typ uživatele	případy užití
nepřihlášený	procházení webu přihlášení import/export databázových dat obnovení databáze do původního stavu
přihlášený (student/učitel)	odhlášení změna osobních dat
přihlášený student	zápis dalšího předmětu odzápis svého předmětu přihlášení na zkouškový termín ze svého předmětu odhlášení se ze zkoušky
přihlášený učitel	zápis nového předmětu k vyučování odebrání svého předmětu zrušení svého zkouškového termínu vytvoření nového zkouškového termínu vytvoření nového ohodnocení studenta změna ohodnocení studenta vypsání seznamu učitelů s jejich předměty

Tabulka 2.1: Případy užití dle typu uživatele UIS

2.2 Existující funkcionální testy UIS

K aplikaci UIS bylo také vytvořeno velké množství funkcionálních testů, k nimž existují podpůrné knihovny. Obojí je napsáno v Javě a samotné testy využívají technologii JUnit 4 a framework Selenium WebDriver. Samotné testy UIS nelze pro tuto bakalářskou práci použít, ale lze využít pomocného balíku `support`.

2.2.1 Struktura testů

Vytvořené testy jsou organizovány do tří skupin – pasivní, aktivní a negativní.

Kategorie pasivních testů ověřuje správné zobrazení webových stránek včetně jednotlivých elementů, u kterých kontroluje také jejich obsah. Patří sem například test, který ověřuje, zda-li se po přihlášení studenta v jeho menu nacházejí správné položky. Jiný test ověřuje, že položky v tabulce zapsaných předmětů studenta odpovídají datům z databáze. Tato kategorie obsahuje celkem 79 souborů – testů a testovacích sad.

Aktivní testy ověřují funkcionalitu aplikace pomocí provádění alespoň jedné aktivity. Samotné testy jsou rozděleny do tří skupin – testování krajních situací, testování pomocí jediné aktivity a testy, ve kterých se provádí více aktivit po sobě. Do testování krajních situací patří například studentova změna křestního jména na jméno, které má nejkratší povolenou délku. Příklad testu s jednou aktivitou je (po přihlášení učitele) vypsání nového zkouškového termínu z vyučovaného předmětu. Testem, ve kterém se provádí více aktivit, je například dokončení dvou zkoušek studenta a jejich následné ohodnocení oběma učiteli. Aktivní testy obsahují celkem 54 souborů – testů a testovacích sad.

Negativní test je takový test, který neočekává pozitivní výsledek. Zkouší provádět aktivity, které by měly skončit chybovým hlášením. Patří sem například změna uživatelského příjmení – uživatel zkusí zadat nové příjmení, které překračuje maximální povolený počet znaků. Celkem bylo vytvořeno 16 negativních testů včetně jedné testovací sady.

2.2.2 Balík `support`

Pro lepší organizaci testů je oddělen kód testů od kódu seleniových příkazů. Nejdříve bylo nutné si připravit podpůrné metody na základní nastavení, entity a opakující se akce. Tato knihovna (balík) má název `support`.

V tomto balíku se vyskytují třídy pro samotné entity student, učitel a předmět. Dále jsou zde pomocné třídy pro jednotlivé databáze těchto en-

tit. Pro funkčnost jsou zde třídy na načtení nastavení (konfigurace), všechna ID z testované aplikace a veškeré konstanty. Struktura celého balíku je znázorněna na popisu 2.1.

```
uis.support
    |_ basic
        |_ Configurations
        |_ Const
        |_ Id
        |_ Txt
        |_ UISLogger
    |_ db
        |_ entities
            |_ Student
            |_ Subject
            |_ Teacher
            |_ User
        |_ storage
            |_ DbStudents
            |_ DbSubjects
            |_ DbTeachers
            |_ DbUsers
        |_ support
            |_ ExamDates
            |_ ExamDatesGrades
            |_ Grade
            |_ Parametres
            |_ TeacherSubject
        |_ DbInicialization
    |_ test
        |_ categories
            |_ ActiveTest_Category
            |_ DbContentTest_Category
            |_ ModalContentTest_Category
            |_ ModalDbContentTest_Category
            |_ PageContentTest_Category
            |_ SmokeTest_Category
        |_ Abstract_Set_Tests
    |_ utils
        |_ application
            |_ Actions
            |_ CheckStatus
            |_ CheckStu
            |_ CheckTea
            |_ InfoStu
            |_ InfoTea
            |_ LogInOut
        |_ web
```

```
|_ Click
|_ EvalTable
|_ Check
|_ JavaScript
|_ SubTable
|_ Table
|_ Utils
```

Listing 2.1: Struktura balíku `support`

2.3 JUnit 4

2.3.1 Obecné informace

JUnit 4 je jednoduchý, open source framework pro psaní a spouštění opakovatelných jednotkových testů. Je to natolik rozšířený a podporovaný framework, že je využíván nejen pro jednotkové testy, ale i pro další typy testů z kategorie funkcionálního testování. Jednotlivé testy mohou být organizovány do testovacích sad, tzv. *test suites*.

Inspirací pro JUnit je knihovna xUnit, kterou vytvořit Kent Beck nejprve pro Smalltalk. Tato knihovna existuje v různých variantách pro mnoho programovacích jazyků, pro Javu existuje právě JUnit. Jeho původními tvůrci jsou Erich Gamma a Kent Beck [2]. Ve skutečnosti existuje několik testovacích frameworků na podobné bázi, ovšem JUnit je používán jako de facto průmyslový standard.

2.3.2 Základní anotace

- `@Test`

  Značí metodu, ve které se provádí samotné testování. Tato anotace působí, že jí označená metoda může být zařazena jako testovací případ a pomocí JUnit spouštěna. Hlavička metody, představující jeden testovací případ, začíná vždy `public void`, případně `public final void`. Název metody typicky začíná slovem `test` a pokračuje názvem testované entity, například v případě názvu metody `setValue` vypadá celá hlavička testu takto: `public void testSetValue()`.

- `@Before`

  Označuje metodu, která se provede před spuštěním každého testu. Metoda je typu `public void`, celý tvar `public void setUp()`. Tato me-

toda je vhodná pro opakované nastavování podmínek pro každý testovací případ.

- `@After`

  Metoda s anotací `@After` se provede po ukončení každého testu. Celá hlavička metody má typicky tvar `public void tearDown()`. Její použití je například pro vymazání dočasných souborů nebo logování chyby.

- `@BeforeClass`

  Takto značená metoda se provede před spuštěním všech testů této třídy. Hlavička metody začíná `public static void`, typicky `public static void setUpBeforeClass()`. Vhodné například pro nastavení počátečního URL nebo nastavení statických proměnných.

- `@AfterClass`

  Označuje metodu, jež se provede až po skončení všech testovacích případů této třídy. Celá hlavička metody má typicky tvar `public static void tearDownAfterClass`. Použije se například pro uvolnění zdrojů či statistiku proběhlých testů.

2.4 Oxygen

2.4.1 Obecné informace

Oxygen je aplikace, vyvíjená na Fakultě elektrotechnické na Českém vysokém učení technickém v Praze. Je to open freeware nástroj, napsaný jako platformně nezávislý program v jazyce Java [5]. V tomto programu lze snadno připravit graf stavů a přechodů pro širokou škálu aplikací. Na základě tohoto grafu umí nástroj vygenerovat testovací případy, ty pak mohou výrazně usnadnit manuální psaní testovacích případů.

2.4.2 Formáty výstupních souborů

Projekty vytvořené v Oxygenu se ukládají do souborů se speciální příponou *.prj*. Každý projekt může obsahovat několik grafů. Graf lze z Oxygenu vyexportovat v těchto formátech: XML, CSV, JSON a SVG. Naopak import lze provést pomocí souborů ve formátu XML nebo CSV. Co se týče testovacích případů, nazývaných *test situations*, ty lze vyexportovat v XML, CSV nebo JSON.

Pro výše zmíněnou testovanou aplikaci (UIS) byl graf již předpřipravený. V rámci této práce byl postupně aktualizován a výrazně doplněn (viz dále). Další větší změny se nyní neočekávají, minimálně do změny verze UIS.

2.4.3 Generování testovacích případů

Po vytvoření grafu lze přejít ke generování testovacích případů. Při zpracovávání grafu je možné využít různá kritéria pokrytí jako například pokrytí uzlů či pokrytí hran [1]. Algoritmus používaný Oxygenem je založený na Test Depth Level (TDL) kritériu pokrytí. Čím je TDL vyšší, tím více testovací případy pokrývají aplikaci.

Oxygen dokáže generovat testovací případy pomocí dvou různých algoritmů:

- Process Cycle Test (PCT) – algoritmus, založený na TDL a rozšířený o TDL redukci v cyklech grafu
- Prioritized Process Test (PPT) – algoritmus, kombinující TDL s kritériem priority pokrytí

Poznámka: Z hlediska dalšího zpracování této práce není použitý algoritmus podstatný.

2.4.4 XML soubor s testovacími případy

Pro tuto práci byly používány testovací případy ve formátu XML (popis tohoto formátu do této práce nepatří, bližší informace lze nalézt například v knize [3]). Oxygen dokáže dle grafu testovací případy vygenerovat. Hlavní element ve vytvořeném XML souboru má název `test_situations`. V jeho atributu `name` jsou dodatečné informace jako například použitý algoritmus pro vygenerování testovacích případů.

Uvnitř tohoto hlavního elementu se nacházejí jednotlivé testovací případy jako jednotlivé elementy s názvem `test_situation`. Mají jeden atribut (`selected_row`), který avšak pro tuto práci není podstatný. Obsah těchto elementů tvoří pole čísel, která jsou oddělena pomlčkami. Tato čísla reprezentují názvy hran, přes které má daný testovací případ procházet.

Příklad výše popsaného XML souboru s testovacími případy:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<test_situations name="Test situations 1, TDL= 1, ALG= PCT."
   note="">
<test_situation selected_row="0">01 - 02 - 03 - 04 - 100 -
   103 - 08 - 09</test_situation>
```

```
<test_situation selected_row="1">01 - 02 - 03 - 04 - 100 -
    151 - 152 - 153 - 155 - 157 - 158 - 101 - 131 - 132 - 133
    - 134 - 135 - 137 - 139 - 1312 - 101 - 104 - 06 - 100 -
    103 - 08 - 09</test_situation>
</test_situations>
```

Poznámka: Během práce byl aktualizován nástroj Oxygen a jeho nová verze 2.3 generuje testovací případy včetně názvů uzlů. Ale protože načítání XML souboru s testovacími případy bylo v době aktualizace Oxygenu již hotové pro názvy hran, nebylo zapotřebí pracovat i s názvy uzlů. I proto nakonec manuálně vytvořené testovací případy obsahují pouze názvy hran.

2.5 Selenium WebDriver

2.5.1 Obecné informace

Framework Selenium je sada nástrojů pro automatizování činnosti webových prohlížečů [7]. Používá se především k automatizovanému testování webových aplikací. Jeden z nástrojů, který automatizaci webového prohlížeče umožňuje, má název *Selenium WebDriver*. Je programovatelný a díky němu je možné simulovat chování uživatele v prohlížeči.

Mezi podporované jazyky patří kromě Javy také například Python, C# či JavaScript. V podporovaných prohlížečích najdeme například Firefox, Google Chrome nebo Safari [2].

2.5.2 Možnosti a funkce

Mezi hlavní možnosti a funkce tohoto nástroje patří:

- zobrazení konkrétní webové stránky
- vyhledávání prvků na webové stránce (například podle atributu `id`, `name` či podle samotného názvu značky (`tagname`))
- získání textového obsah daného prvku stránky
- zápis uživatelského vstupu
- zaškrtnutí checkboxu
- zvolení položky z výběrového seznamu
- odesílání formuláře (klik na tlačítko)

Selenium WebDriver toho umí ještě daleko více, ale pro účely této práce postačily výše zmíněné funkce.

3 Graf vytvářený Oxygenem

3.1 Obecné informace

Cesty průchodů jakoukoliv aplikací lze překreslit v Oxygenu jako graf. Jedná se o graf přechodů (hran) a stavů (uzlů), které existují v dané aplikaci. Každá hrana má svůj počáteční a koncový uzel (vždy vede jedním směrem mezi dvěma uzly). Všechny elementy grafu (uzly i hrany) mají dva identifikátory. Jedním z nich je `id`, unikátní číslo, a druhým je název elementu (označovaný jako `name`). Vzhledem k přítomnosti identifikátoru `id` sice již není zapotřebí mít druhý identifikátor, ale zároveň není pro přehlednost grafu vhodné, aby se v grafu vyskytovaly dva elementy (hrana nebo uzel) stejného názvu. Z toho důvodu je atribut `name` také unikátní. Dále má každý element atribut `description`, popis. Na tomto atributu je tato práce založena.

Přímo v nástroji Oxygen existuje funkce pro spuštění validace daného grafu. Kromě jiného tato validace ověří, že žádné dva elementy (hrana či uzel) nemají stejný název ani stejné ID [8].

3.2 Uzel (stav)

V XML souboru, jenž je vygenerován z Oxygenu a popisuje celý graf modelované aplikace, je uzel reprezentován následujícím elementem (pro lepší přehlednost bylo přidáno odřádkování):

```
<node
    description="id=loginPage.userNameInput&#10;
                 type=input&#10;
                 text=username"
    height="40.0"
    id="3"
    limitedConnectionProbability="0.0"
    name="Tea Username"
    priority="(not defined)"
    style="STYLE_ACTIVITY_NODE"
    width="80.0"
    xpos="280.0"
    ypos="20.0"
/>
```

Většina z těchto atributů je pro budoucí generování testů nadbytečná. Tato práce využívá pouze atributy `id` (identifikační číslo) a `description`

(popis). Atribut `description` je velmi důležitý, neboť z něj se získávají informace pro následné generování testů. Tento atribut obsahuje další položky a jejich hodnoty. Jako oddělovač jednotlivých položek slouží nová řádka, *line feed*, která je reprezentována v XML souboru jako `
`.

3.2.1 Typy uzlů v grafu

Graf jakékoliv aplikace se může skládat ze čtyř hlavních typů uzlů:

- Start (*STYLE_ACTIVITY_START*)
  Počáteční uzel, kde každý testovací případ začíná. V grafu je tento uzel znázorněn jako černě vyplněný kruh.
- End (*STYLE_ACTIVITY_END*)
  Koncový uzel aplikace, ve kterém končí každý testovací případ. V grafu má tvar kruhu, který má černý střed a ohraničení.
- Activity (*STYLE_ACTIVITY_NODE*)
  Uzel, kde se provádí určitá aktivita (například přihlášení). Má tvar obdélníku se zaoblenými rohy.
- Decision (*STYLE_ACTIVITY_DECISION*)
  Rozhodovací uzel, ze kterého vychází více hran k různým uzlům. Zobrazí se v grafu jako kosodélník.

3.3 Hrana (přechod)

V XML souboru, jenž je generován Oxygenem a popisuje celý graf modelované aplikace, je hrana popisována následovně (pro lepší přehlednost bylo přidáno odřádkování):

```
<edge
    description=""
    id="125"
    limitedConnectionProbability="0.0"
    name="0202"
    priority="(not defined)"
    source="3"
    target="4"
/>
```

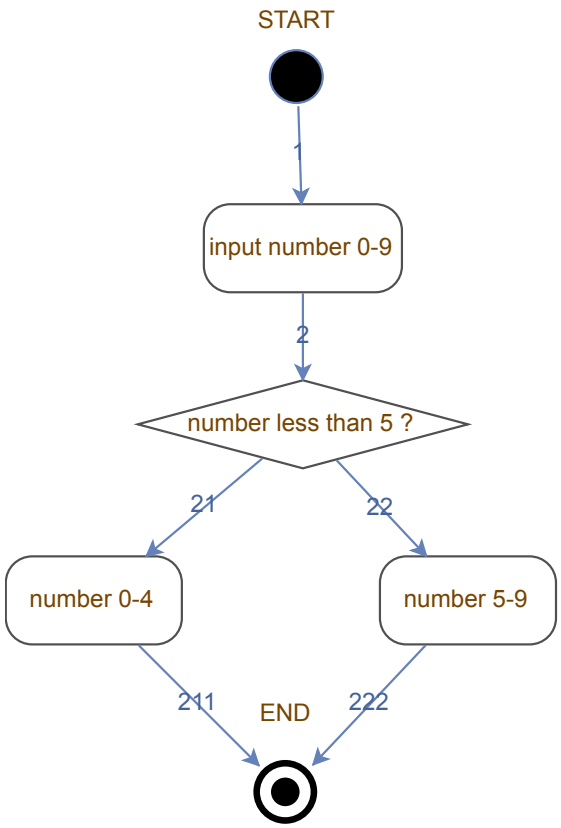
Z těchto atributů byly pro práci využity pouze atributy `name`, `source` a `target`. Z výše uvedené definice vyplývá, že v grafu neexistují dvě hrany stejného jména, tudíž atribut `name` je použit jako unikátní identifikátor

hrany. Atributy `source` a `target` obsahují ID počátečního a koncového uzlu. Atribut `description` se zde vyskytuje stejně jako u uzlu, nicméně pro tuto práci nebyl tento atribut u hrany vůbec použit.

3.4 Graf jednoduché aplikace

Pro znázornění grafu v podobě SVG i XML bude dále používán graf jednoduché aplikace. Tento graf můžeme vidět na obrázku 3.1. V tomto grafu se nacházejí všechny typy uzlů, které Oxygen nabízí.

Jednoduchá aplikace znázorňuje zadání čísla v rozsahu 0 až 9 a jeho následné zpracování. Počáteční uzel má název `START`. Z tohoto uzlu vycházejí všechny cesty grafem. Uzel s názvem `input number 0-9` reprezentuje aktivitu, kdy je do vstupního pole zadáno číslo v daném rozsahu. Poté po hraně s názvem `2` následuje rozhodovací uzel, z nějž vycházejí dvě hrany. Lze tedy přejít buď do uzlu `number 0-4` nebo do uzlu `number 5-9`. Z obou těchto uzlů pak vedou hrany do koncového uzlu `END`, který je tedy vždy posledním uzlem na každé cestě.



Obrázek 3.1: Graf jednoduché aplikace

XML formát grafu je možné si prohlédnout v popisu 3.1. Hlavním elementem je `graph`. V jeho atributech lze najít například jeho název či typ. Pro účel této práce byl používán jen typ grafu `ACTIVITY_DIAGRAM`, ačkoliv Oxygenu umožňuje vytváření i jiných typů grafů. Uvnitř elementu `graph` se nacházejí pouze již výše zmíněné elementy – `node` a `edge`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graph description="" id="6" linkage="" name="Graph-01" type=
   "ACTIVITY_DIAGRAM" version="">
<node description="" height="25.0" id="2"
   limitedConnectionProbability="0.0" name="START" priority="
   " style="STYLE_ACTIVITY_START" width="25.0" xpos="260.0"
   ypos="40.0"/>
<edge description="" id="8" limitedConnectionProbability="0.0
   " name="1" priority="" source="2" target="3"/>
<node description="" height="40.0" id="3"
   limitedConnectionProbability="0.0" name="input number 0-9"
    priority="" style="STYLE_ACTIVITY_NODE" width="90.0" xpos
   ="230.0" ypos="110.0"/>
<edge description="" id="9" limitedConnectionProbability="0.0
   " name="2" priority="" source="3" target="4"/>
<node description="" height="40.0" id="4"
   limitedConnectionProbability="0.0" name="number less than
   5 ?" priority="" style="STYLE_ACTIVITY_DECISION" width="
   150.0" xpos="200.0" ypos="190.0"/>
<edge description="" id="10" limitedConnectionProbability="
   0.0" name="21" priority="" source="4" target="5"/>
<edge description="" id="11" limitedConnectionProbability="
   0.0" name="22" priority="" source="4" target="6"/>
<node description="" height="40.0" id="5"
   limitedConnectionProbability="0.0" name="number 0-4"
   priority="" style="STYLE_ACTIVITY_NODE" width="80.0" xpos=
   "140.0" ypos="270.0"/>
<edge description="" id="12" limitedConnectionProbability="
   0.0" name="211" priority="" source="5" target="7"/>
<node description="" height="40.0" id="6"
   limitedConnectionProbability="0.0" name="number 5-9"
   priority="" style="STYLE_ACTIVITY_NODE" width="80.0" xpos=
   "310.0" ypos="270.0"/>
<edge description="" id="13" limitedConnectionProbability="
   0.0" name="222" priority="" source="6" target="7"/>
<node description="" height="35.0" id="7"
   limitedConnectionProbability="0.0" name="END" priority=""
   style="STYLE_ACTIVITY_END" width="35.0" xpos="250.0" ypos=
   "360.0"/>
</graph>
```

Listing 3.1: XML popis jednoduché aplikace

3.5 Návrh na doplnění XML souboru s grafem

Formát XML je pro reprezentaci grafu ideální, uspořádání souboru je přehledné. Avšak vzhledem k tomu, že nedílnou součástí této práce bylo doplnění potřebných informací do atributu `description`, nepřijde mi vhodné, aby všechny tyto informace byly pouze uvnitř tohoto atributu. Zde uvádím pro ukázku uzel z výsledného grafu UIS (pro přehlednost s odřádkováním):

```
<node
    description="
        tableId=stu.mySubjects.enrolledSubjects.table&#10;
        buttonId=stu.mySubjects.enrolledTable.
           unenrollSubjectButton-&#10;
        expectedId=stu.mySubjects.unenrollSubjectModal.
           unenrollSubjectButton&#10;
        subject=enrolledSubject&#10;
        type=tableClick"
    height="40.0"
    id="10"
    limitedConnectionProbability="0.0"
    name="StuMs Unenroll"
    priority="(not defined)"
    style="STYLE_ACTIVITY_NODE"
    width="80.0"
    xpos="180.0"
    ypos="420.0"
/>
```

Místo takovéto reprezentace by mohl být atribut `description` nahrazen jednotlivými položkami, které se uvnitř něj nacházejí. Z těchto položek by se tedy staly samostatné atributy, což by mohlo vypadat takto:

```
<node
    tableId="stu.mySubjects.enrolledSubjects.table"
    buttonId="stu.mySubjects.enrolledTable.
       unenrollSubjectButton"
    expectedId="stu.mySubjects.unenrollSubjectModal.
       unenrollSubjectButton"
    subject="enrolledSubject"
    type="tableClick"
    height="40.0"
    id="10"
    limitedConnectionProbability="0.0"
    name="StuMs Unenroll"
    priority="(not defined)"
    style="STYLE_ACTIVITY_NODE"
```

```
    width="80.0"
    xpos="180.0"
    ypos="420.0"
/>
```

Avšak tato reprezentace by byla aplikačně závislá. Mohlo by se například stát, že v testované aplikaci se nebude nacházet žádná tabulka, tudíž atribut `tableId` by nebyl vůbec zapotřebí. Z toho vyplývá, že je nutná důkladnější analýza problému, jak by měl XML soubor s grafem vypadat.

4 Analýza aplikace

4.1 Požadavky

Tato práce má dle zadání obsahovat zejména návrh a následnou realizaci modulární aplikace, která bude s využitím dalších datových struktur generovat automatizované funkcionální testy.

Jednotlivé požadavky na aplikaci:

- Programovacím jazykem bude Java. Výsledné zdrojové kódy vygenerovaných testů budou též v Javě a budou využívat JUnit 4 a framework Selenium WebDriver.
- Vstupními daty jsou dva XML soubory – graf testované aplikace a testovací případy. Obojí se vygeneruje z Oxygenu.
- Výstupem budou vygenerované funkcionální testy dle jednotlivých případů. Testy bude možné spouštět také dohromady jako sadu. Jejich součástí bude též logování každé prováděné aktivity.
- Testovanou aplikací je UIS.
- Je doporučené použít balík `support`, který byl vytvořený jako součást testů UIS.

4.2 Kritérium úspěchu testu

Jako první kritérium úspěšnosti testu bylo zvoleno jen to, že test dokáže projít přes všechny dané uzly testovacího případu. Toho bylo docíleno díky kontrole URL po každé jeho změně.

Avšak ne každá aktivita změní URL webové aplikace, proto jako další kritérium úspěchu byla ještě přidána kontrola, zda se po provedení nějaké aktivity zobrazilo oznámení o jejím úspěšném vykonání. Jedná se o kontrolu, jestli se objevil element (hlášení) s příslušným ID.

4.3 Průběh příprav a experimentů

Nejprve bylo nutné řádně prostudovat výchozí technologie a nástroje. Dalším krokem bylo ruční napsání jednoho testu dle grafu a testovacího případu.

Tím bylo zjištěno, jaké informace bude nutné dodat k uzlům do jejich atributu `description`. Následně bylo zapotřebí dopsat ke všem uzlům v grafu potřebné informace do tohoto atributu. Poté se mohlo přejít na samotné vygenerování testů ze vstupních dat, tedy ze XML souborů grafu a testovacích případů. Před vygenerováním testů se vždy ručně zjistilo, pro jaká vstupní data se může daný test provést.

Dalším krokem tedy bylo zautomatizovat výběr vstupních dat pro každý testovací případ. Toho bylo docíleno doplněním dalších informací do atributu `description`. Díky tomu se odhalilo, že testovací případy vygenerované Oxygenem mají nereálné požadavky na vstupní data a tudíž mnoho testů se dle nich nemohlo ani vygenerovat. Bylo tedy zvoleno náhradní řešení – manuálně napsané testovací případy ve stejném formátu XML souboru, jako jsou ty vygenerované Oxygenem. Poslední částí této práce bylo obarvování SVG grafu podle úspěšnosti testů (testovací sady).

4.4 Testovaná webová aplikace UIS

Před testováním webové aplikace UIS bylo nejprve nutné ji důkladně prozkoumat a ručně vyzkoušet. Zjistilo se, že aplikace obsahuje základní prvky webové stránky jako je například menu, tlačítko, výběrový seznam či zaškrtávací pole. Všechny tyto prvky mají dané identifikační číslo (ID), podle kterého je lze jednoznačně najít a provádět s nimi různé akce.

Složitějším prvkem aplikace je základní tabulka a také tabulka, která je rozdělena na více částí. Základní tabulku si lze prohlédnout na obrázku 4.1. Tabulka, která je rozdělena na více částí, je uváděna pod názvem *subtabulka*. Její příklad lze vidět na obrázku 4.2. Rozdělení tabulky je tvořeno pomocí řádků s jednotlivými nadpisy, v příkladu se jedná o názvy předmětů. Tyto nadpisy v řádcích jsou nazývány jako *subheader*, podnadpis. Příkladem celého subheaderu z obrázku je `Subject: Computation Structures`.

4.5 Graf UIS

Pro aplikaci UIS byl graf v nástroji Oxygen již předpřipravený. Očekávalo se, že samotný graf nebude dále upravován, že se pouze dopíší příslušné informace do atributů uzlů, popřípadě hran. Tento graf obsahoval celkem 92 uzlů a 120 hran. Velikost XML souboru toho grafu byla 39 kB.

Při práci se ovšem zjistilo, že v grafu úplně chybí některé stavy, které mohou v aplikaci UIS nastat. Kvůli tomu musel být graf upravován a pozměňován, za celou dobu výzkumu bylo postupně vytvořeno 24 verzí tohoto

Enrolled Subjects

#	Name	Teacher(s)	Credits	Unenroll subject
1	Computation Structures	Thomas Scatterbrained	5	X
2	Computer System Engineering	Alice Pedant	6	X
3	Programming in Java	Peter Strict	4	X

Obrázek 4.1: Snímek obrazovky UIS – ukázka jednoduché tabulky

Other Available Exam Dates for My Subjects

#	Teacher(s)	Date & Time	Participants	Register
Subject: Computation Structures				
1	Thomas Scatterbrained	2019-04-24 22:28	Participants (7/10)	Register
Subject: Computer System Engineering				
1	Alice Pedant	2019-04-24 22:28	Participants (3/10)	Register
Subject: Programming in Java				
1	Peter Strict	2019-04-24 22:28	Participants (8/10)	Register

Obrázek 4.2: Snímek obrazovky UIS – ukázka subtabulky

grafu. Nynější verze obsahuje 122 uzlů a 167 hran. Velikost XML souboru je 53 kB. Ilustrační obrázek této verze si lze prohlédnout na obrázku 4.3.

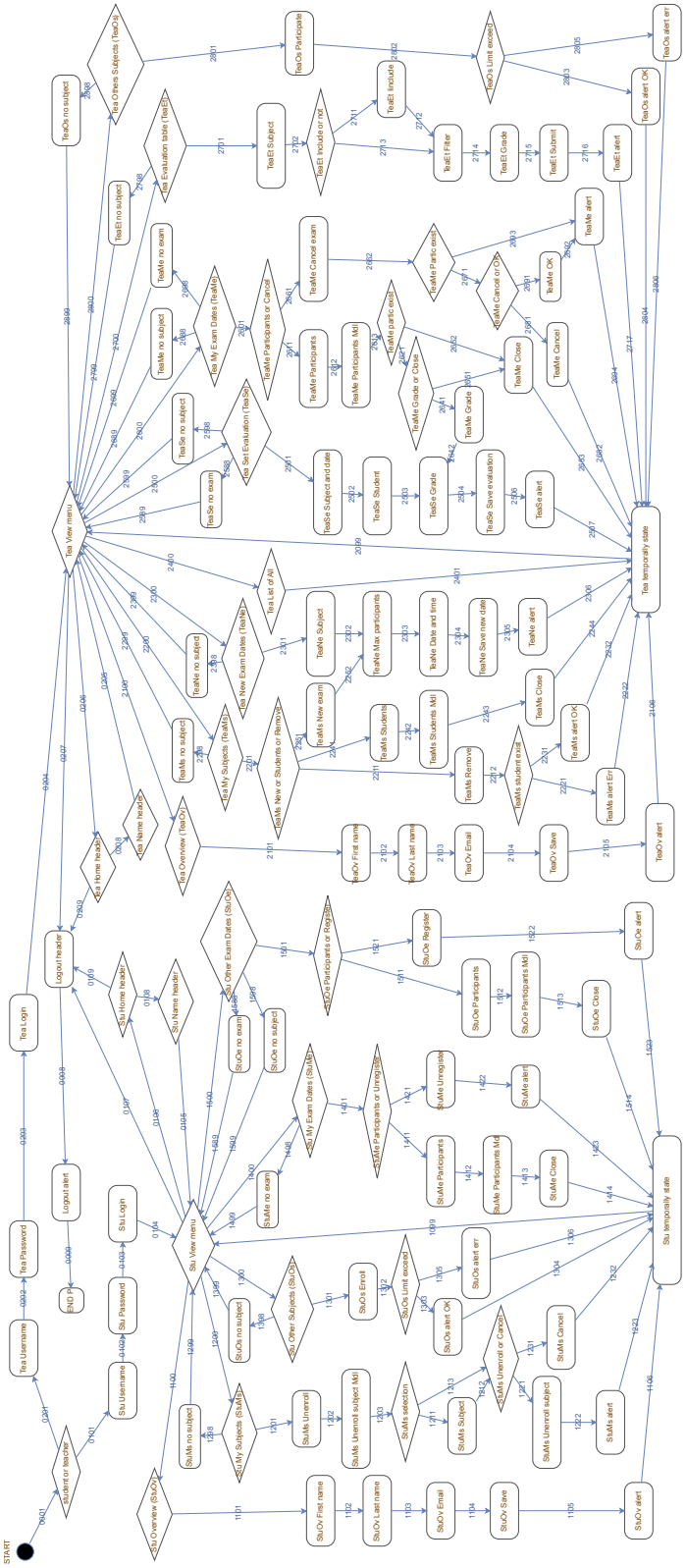
4.6 XML atribut `description`

Atribut `description` se nachází v XML souboru grafu u elementů `node` i `edge`, avšak data byla doplňována pouze k elementu `node`. Tento atribut je zcela obecný. Jednou z nejdůležitějších počátečních aktivit byla analýza, jaké všechny položky je nutné doplnit pro úspěšné generování testů. I přes důkladnou analýzu bylo později nutné tento seznam položek doplňovat.

Typy položek, které lze zapsat do atributu `description`, lze rozdělit do dvou skupin. První skupinu tvoří ty, které jsou obecné a nijak přímo nesouvisí s testovanou aplikací UIS. Příkladem je `buttonId`, neboť existence tlačítka se předpokládá u široké škály aplikací. Ve druhé skupině se nacházejí takové položky, které jsou silně aplikačně závislé. Jedná se například o `tableId`, které není zařazeno mezi obecné položky, neboť ne každá aplikace obsahuje tabulku. Položka `examDate` je ovšem již přímo závislá na testované aplikaci UIS.

Obecné typy položek, které lze zapsat do atributu `description`:

- `type` – typ aktivity, která se má vykonat
- `id` – ID elementu, nad kterým se má vykonat daná aktivita
- `url` – URL stránky
- `buttonId` – ID tlačítka
- `expectedId` – ID elementu, který se má po vykonání aktivity zobrazit
- `subject` – entita, nad kterou se má vykonat daná aktivita (například student)
- `text` – text, který má být vložený do pole
- `boolValue` – hodnota true/false (pro zaškrtnutí checkboxu)
- `db` – pomocná položka, omezuje vstupní data, nad kterými může být daný test provedený (průchod přes daný uzel je omezený vstupními daty)
- `dbSubject` – pomocný atribut k atributu `db`



Obrázek 4.3: Ilustrační obrázek pro ukázku složitosti nynějšího grafu UIS

Aplikačně závislé položky:

- `tableId` – ID tabulky
- `subheaderId` – ID podnadpisu uvnitř tabulky
- `examDate` – datum zkoušky

4.6.1 Možnosti položky `type`

Nejdůležitější položkou je `type`, neboť podle ní se určuje, jaké další položky musí popis daného uzlu obsahovat. Možnosti včetně dalších povinných položek k dané hodnotě naleznete v tabulce 4.1.

Možnosti položky `type` lze rozdělit do dvou kategorií. Do první kategorie spadají všechny možnosti, které jsou obecné a tudíž by bylo možné je uplatnit i u jiných testovaných aplikací, než jen u UIS. Druhou kategorii tvoří ty možnosti, které jsou silně aplikačně závislé (na UIS).

Obecné možnosti položky `type`:

- `url` – přejde na dané URL stránky, které následně zkontroluje
- `inputText` – vloží text do pole s daným ID
- `click` – klikne na element s daným ID
- `button` – klikne na tlačítko s daným ID a zkontroluje výsledné URL
- `buttonWaitId` – klikne na element s daným ID a počká, dokud se neobjeví očekávané ID
- `buttonAndWaitHideModal` – klikne na element s daným ID a počká, dokud nezmizí modální okno
- `tableClick` – klikne na tlačítko s daným ID prefixem[1], které se nachází v tabulce
- `subtableClick` – klikne na tlačítko s daným ID prefixem, které se nachází v subtabulce
- `isDisplayed` – ověří, že element s daným ID je zobrazený
- `isDisplayedIdPrefix` – ověří, že element s daným ID prefixem je zobrazený

[1]Prefix je předpona slova.

- `selection` – zvolí možnost z výběrového seznamu s daným ID
- `checkBox` – zaškrtne/odškrtne checkbox s daným ID
- `confirmAlert` – potvrdí vyskakovací okno
- `cancelAlert` – zruší vyskakovací okno

Možnosti položky `type` závislé na UIS:

- `input` – do pole s daným ID vloží příslušný text (záleží, zda-li se jedná o studenta či učitele, a také, jestli má být vloženo uživatelské jméno/heslo/...)
- `grade` – ohodnotí daného studenta známkou ze zkoušky (podle studentova jména a názvu předmětu najde příslušný řádek v tabulce, kde se nachází výběrový seznam s dostupnými známkami, a provede výběr známky)
- `submitGrade` – uloží ohodnocení studenta (podle studentova jména a názvu předmětu najde příslušný řádek v tabulce, kde se nachází tlačítko pro uložení ohodnocení studenta, které je následně stisknuto)

4.6.2 Možnosti položky `db`

Jak již bylo zmíněno výše, průchod grafem silně závisí na vstupních datech. V této části jsou popsány skupiny nastavení dat.

Níže jsou uvedeny jednotlivé hodnoty v části grafu, ve které se pracuje se studentem:

- `stuMySubjectsEmpty` – student nesmí mít žádný zapsaný předmět
- `stuOtherSubjectsEmpty` – pro studenta neexistuje žádný další předmět, který by si mohl zapsat
- `stuMyExamDatesEmpty` – student není přihlášen na žádný zkouškový termín
- `stuOtherExamDatesEmpty` – pro studenta neexistuje zkouškový termín, na který by se mohl přihlásit
- `stuOtherSubjectsLimit` – student má zapsán maximální povolený počet předmětů (dosáhl limitu)

možnost	další položky	
	nutné	nepovinné
inputText	id, text	–
input	id, text	–
url	url	–
button	id, url	–
click	id	–
buttonWaitId	id, expectedId	–
tableClick	subject, tableId, buttonId	expectedId
subtableClick	subject, tableId, buttonId, subheaderId	expectedId
buttonAndWaitHideModal	id	–
isDisplayed	id	–
isDisplayedIdPrefix	id	–
selection	id, subject	examDate
checkBox	id, boolValue	–
grade	subject	–
submitGrade	subject	–
confirmAlert	–	–
cancelAlert	–	–

Tabulka 4.1: Možnosti položky `type`

Hodnoty v části grafu, kde se pracuje s učitelem:

- `teaMySubjectsEmpty` – učitel neučí žádný předmět
- `teaMyExamDatesEmpty` – učitel nemá vypsaný žádný zkouškový termín
- `teaOtherSubjectsLimit` – učitel učí maximální povolený počet předmětů (dosáhl limitu)
- `teaOtherSubjectsEmpty` – pro učitele neexistuje další předmět, který by mohl učit

4.6.3 Možnosti položky `subject`

Přítomnost této položky v uzlu grafu značí, že v daném uzlu se s touto položkou pracuje. Například pokud si chce student odebrat studovaný předmět, pak se v uzlu, kde se předmět odebírá, nachází položka s příslušnou hodnotou: `subject="enrolledSubject"`.

Hodnoty, kterých může položka nabývat v části grafu, kde se pracuje se studentem:

- `enrolledSubject` – předmět, který student studuje
- `unenrolledSubjectWithTeacher` – předmět, který má učitele a zároveň ho daný student nestuduje
- `subjectWithExamDate` – zapsaný předmět, u kterého existuje vypsaný zkouškový termín, na který je daný student zapsán
- `subjectWithoutStudent` – zapsaný předmět, který má vypsaný alespoň jeden zkouškový termín a zároveň na žádném zkouškovém termínu z tohoto předmětu není daný student zapsán

Hodnoty položky, které se mohou nacházet v části grafu, kde se pracuje s učitelem:

- `taughtSubject` – předmět, který učitel učí
- `othersSubject` – předmět, jenž učitel neučí
- `taughtSubjectWithExamDate` – předmět, který je vyučován daným učitelem (a nikým jiným), existují dva typy:
  - daný předmět má vypsán alespoň jeden zkouškový termín
  - daný předmět má vypsán alespoň jeden zkouškový termín, na kterém je zapsán alespoň jeden student
- `examDate` – existují dvě možnosti:
  - pokud má být využit i `student` (popř. `fullName`), vezme se pouze daným vyučujícím vyučovaný předmět, ze kterého je vypsán zkouškový termín, na který je daný student přihlášen
  - jakýkoliv zkouškový termín z jakéhokoliv vyučovaného předmětu
- `taughtSubjectToRemove` – vyučovaný předmět, který je dále upřesněný položkou `dbSubject`:
  - `dbSubject=withStudents` – předmět studuje alespoň jeden student
  - položka `dbSubject` není uvedena – předmět nestudují žádní studenti

- `taughtSubjectWithExamDateToRemove` – vyučovaný předmět (pouze daným učitelem) s vypsaným zkouškovým termínem, který je dále upřesněný položkou `dbSubject`:
  - `dbSubject=withStudents` – na všech zkouškových termínech je zapsán alespoň jeden student
  - položka `dbSubject` není uvedena – na žádném zkouškovém termínu nejsou zapsáni žádní studenti
- `student` – student, který má zapsaný předmět, jenž je vyučován daným učitelem, a také je přihlášený na zkouškový termín z tohoto předmětu
- `fullName` – celé jméno studenta
- `grade` – jakákoliv známka, kterou může učitel udělit

4.6.4 Možnosti položky `text`

Vyskytuje se ve dvou případech, buď se jedná o hodnotu, která je nezávislá na jakýchkoliv jiných atributech, nebo o hodnotu, která přímo souvisí s atributem `student` případně `teacher`.

Nezávislé:

- `maxParticipants` – zadání maximální počtu účastníků (využívá se při vypisování nového zkouškového termínu jako maximální možný počet studentů)
- `newExamDateInput` – zadání nového zkouškového termínu (datum a čas ve formátu `YYYY-MM-DD HH:mm`, například `2019-09-29 17:40`)

Závislé možnosti, jedná se o data společná pro jakéhokoliv přihlášeného uživatele:

- `username` – uživatelské jméno
- `password` – heslo
- `firstName` – křestní jméno
- `lastName` – příjmení
- `email` – e-mailová adresa

4.7 Zpracování vstupních dat

Před samotným generováním testů je nutné nejprve zpracovat vstupní soubory – graf a testovací případy. Protože se jedná o XML soubory, existuje více možností pro práci s nimi.

Mezi dvě hlavní technologie zpracování (*parsování*) XML souborů patří:

- SAX (*Simple API for XML*) – Funguje na principu proudového čtení. Sekvenčně zpracovává celý soubor, je tedy nutné načtená data průběžně zpracovávat nebo ukládat.
- DOM (*Document Object Model*) – Celý dokument je najednou načtený do paměti ve stromové stuktuře. Výhodné, pokud je zapotřebí XML soubor nejen číst, ale také do něj zapisovat. Avšak nevhodné pro velké soubory, protože zabírá mnoho místa v paměti.

Kromě těchto dvou základních technologií, které mají naprosto odlišné přístupy, existuje také například StAX, *Streaming API for XML*. Ten, stejně jako SAX, čte soubor sekvenčně. Mezi výhody StAXu oproti technologii SAX patří například to, že umí dokument nejen vytvářet, ale i do něj zapisovat. Další výhodou je, že při čtení je textový obsah vracen najednou [3].

4.8 Problémy během řešení

Při výzkumu a zjišťování, jak bude generování fungovat, se narazilo na jisté neočekávané překážky a problémy, které bylo nutno řešit.

4.8.1 Vhodný výběr vstupních dat

Výběr dat, nad kterými se budou jednotlivé testy spouštět.

Zjistilo se, že každý testovací případ je unikátní ve smyslu, že nelze spustit nad jakýmikoliv daty. Například někdy bylo nutné, aby student měl zapsaný alespoň jeden předmět, ale v jiném případu bylo naopak potřeba, aby student neměl zapsaný žádný předmět.

Řešením bylo dopsat do `description` jednotlivých uzlů informaci, která by upřesňovala data, se kterými bude možno projít přes daný uzel v grafu.

4.8.2 Neodpovídající model aplikace UIS

Při experimentech se pracovalo se zkouškovými termíny jednotlivých předmětů a bylo zapotřebí vědět, který učitel vypsal daný termín. Zjistilo se však,

že model aplikace v balíku `support` není vůbec navržen tak, aby zajistil poskytnutí této informace. Zkouškový termín byl spojený jen s předmětem, ale už ne s učitelem. Problém tedy nastal u předmětů, které jsou vyučovány více než jedním učitelem.

Náhradní řešení problému bylo, že pokud bylo zapotřebí vědět, který učitel vypsal daný zkouškový termín, předměty s více než jedním učitelem se jednoduše neuvažovaly (přeskakovaly).

4.8.3 Nedostatečná vstupní data v databázi UIS

Databáze aplikace UIS je již naplněná vhodnými daty. Při práci se ale zjistilo, že dostatečně nepokrývá všechny krajní situace, které v aplikaci existují. Například neexistuje student, kterému by se zobrazila prázdná tabulka předmětů k zapsání. Tudíž tento testovací případ nemůže být ověřen a test není možné vygenerovat.

4.8.4 Vygenerované testovací případy z Oxygenu

Ke generování testů se nejprve používaly testovací případy, které byly vygenerovány z Oxygenu. Avšak jakmile začala být vybírána vstupní data dle předzpracování cesty daného testovacího případu, zjistilo se, že některé testovací případy mají nereálné nároky na vstupní data. Například byl vygenerován testovací případ, kdy se student přihlásil a poté měl procházet přes uzel, kde neměl zapsán žádný předmět, ale následně přes uzel, kde se měl podívat na účastníky u nějakého svého zkouškového termínu. Není ale možné, aby student nestudoval ani jeden předmět a zároveň byl přihlášený na nějaký zkouškový termín.

V rámci této práce bylo zvoleno náhradní řešení – vstupními testovacími případy nebudou ty, které jsou generovány z Oxygenu. Místo nich byly ručně vytvořeny testovací případy, které projdou celý graf, ale zároveň budou mít reálné nároky na vstupní data. Pokud se ovšem i přesto najdou případy, kdy nebudou k dispozici žádná vstupní data, splňující dané podmínky, test nebude vygenerován.

4.8.5 Úprava grafu UIS

Při zadávání této práce se očekávalo, že předpřipravený graf aplikace UIS, vytvořený v Oxygenu, nebude nadále upravován. Respektive že jedinou úpravou bude pouze doplnění atributu `description` o nezbytné informace.

Nicméně během výzkumu se zjistilo, že vytvořený graf má jisté nedostatky. Neobsahoval sice chyby, ale chyběly v něm nějaké cesty. Jednalo se

hlavně o krajní případy různých situací, například chyběla cesta, kdy student bez jakéhokoliv studovaného předmětu nahlédne do záložky `My Subjects`. Další podrobnosti ohledně tohoto problému byly již uvedeny v části 4.5.

5 Implementace

Pro přehlednost bude vytvářená aplikace dále nazývána Generátor.

5.1 Struktura aplikace

Vytvořená aplikace je dle zadání v jazyce Java. Vygenerované testy využívají technologií JUnit 4 a Selenium WebDriver.

Veškeré nově vytvořené třídy aplikace Generátor se nacházejí uvnitř balíku `uis.generator`. V balíku `uis` se kromě toho nachází též balík `support`, který je převzatý z již existujících testů aplikace UIS. Jeho struktura byla již popsána v části 2.2.2. Během práce se zjistilo, že tento převzatý balík `support` nebude potřeba celý, tudíž z něj byly odstraněny všechny nepotřebné třídy. Nakonec byl odstraněn celý balík `uis.support.test` a také celý balík `uis.support.utils.application`.

Vytvořená aplikace Generátor má následující strukturu:

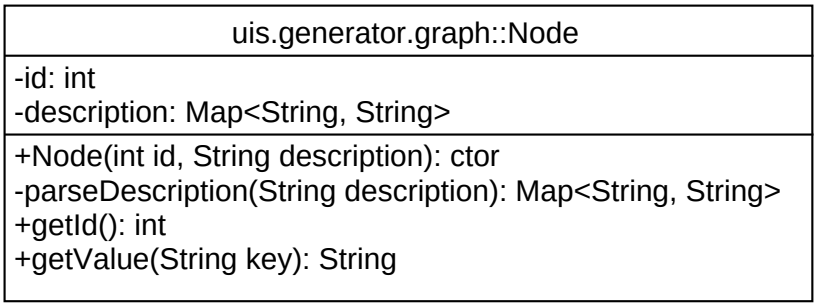
```
uis.generator
    |_ graph
        |_ Edge
        |_ Node
        |_ Graph
    |_ support
        |_ AbstractGeneratedSuite
        |_ AbstractGeneratedTest
        |_ Code
        |_ StringConstants
        |_ DbSupport
        |_ Support
    |_ utils
        |_ StudentGenerator
        |_ TeacherGenerator
        |_ TestGenerator
        |_ SVGEditor
        |_ XMLLoader
    |_ MainClass
```

5.2 Balík `graph`

Graf je v Generátoru reprezentován pomocí tří tříd – `Node` (uzel grafu), `Edge` (hrana grafu) a `Graph` (celý graf jako seznam uzlů a hran).

5.2.1 Třída `Node`

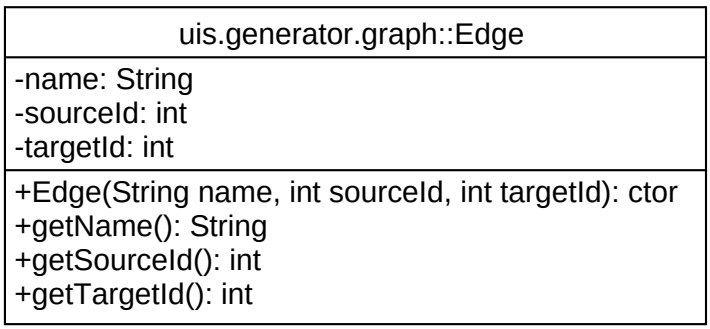
V Generátoru existuje třída `Node`, která slouží k reprezentaci jednotlivých uzlů grafu. Její UML class diagram lze vidět na obrázku 5.1.



Obrázek 5.1: UML class diagram třídy `Node`

5.2.2 Třída `Edge`

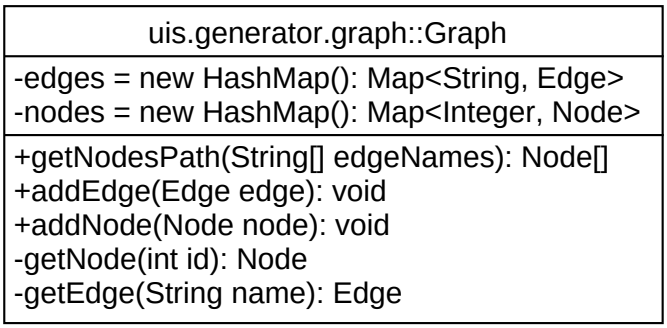
V Generátoru se nachází třída `Edge`, která reprezentuje jednotlivé hrany grafu. Její UML class diagram včetně všech atributů a metod je zobrazen na obrázku 5.2.



Obrázek 5.2: UML class diagram třídy `Edge`

5.2.3 Třída `Graph`

Graf aplikace je popisován pomocí dvou hashovacích map, z nichž jedna je používána pro hrany (klíčem je jméno) a druhá pro uzly (klíčem je číselné ID). UML class diagram této třídy je uveden na obrázku 5.3.



Obrázek 5.3: UML class diagram třídy `Graph`

Za zmínku stojí metoda `getNodesPath`, která jako argument bere pole jmen hran a vrací pole uzlů. Metoda byla napsána z toho důvodu, že po vygenerování testovacích případů z Oxygenu vznikne soubor, ve kterém jsou jednotlivé případy popsány pomocí jmen hran. Program ale potřebuje znát hlavně uzly, přes které případ prochází, proto bylo vytvoření této metody nezbytné.

5.3 Balík `support`

V tomto balíku se nacházejí různé typy podpůrných tříd. Jsou zde dvě abstraktní třídy, jedna pro vygenerované testy a druhá pro vygenerovanou sadu testů. Dále je zde třída pouze s řetězcovými konstantami a třída, která vrací pouze jednořádkové řetězce s úsekem kódu. Další třídou je `DbSupport`, která obsahuje metody pro výběr entit z databáze dle různých kritérií. Poslední třídou v tomto balíku je `Support`, která obsahuje všechny další pomocné metody (bude podrobněji vysvětleno dále).

5.3.1 Třída `AbstractGeneratedTest`

Aby byl kód generovaných testů kratší a zároveň se neopakoval v každém testu, bylo vhodné vytvořit abstraktní třídu `AbstractGeneratedTest`. Tuto třídu poté každý vygenerovaný test dědí.

Jako své atributy obsahuje všechny entity, které jsou ve všech testech používány. Je zde například:

```
protected static Student student;
protected static Subject enrolledSubject;
protected static Subject subjectWithExamDate;
...
```

Kromě toho jsou zde dva pomocné atributy, které jsou potřeba k následnému vygenerování vizualizace výsledku ve formátu SVG. Těmito atributy jsou:

```
protected static int index;
protected static List<String> edges;
```

V seznamu `edges` se nachází názvy všech hran, přes které vede cesta grafem daného testovacího případu. Atribut `index` slouží v případě neúspěšného testu ke zjištění, před kterou hranou v cestě grafem došlo k selhání.

5.3.2 Třída `AbstractGeneratedSuite`

Uvnitř této třídy je definována pouze činnost, jež se má vykonat před spuštěním sady testů a také po jejich ukončení. Před spuštěním sady testů se vytvoří složka pro ukládání výsledků testů v SVG formátu. Po ukončení sady se vytvoří finální agregované obarvené SVG, reprezentující výsledky celé testovací sady jako celku.

5.3.3 Třída `StringConstants`

Tato třída uchovává pouze veřejné statické řetězcové konstanty, neobsahuje tedy žádné metody. Tyto konstanty obsahují úseky kódu, které nevykonávají žádnou aktivitu. Patří sem například importování jiných tříd nebo hlavičky metod.

5.3.4 Třída `Code`

V této třídě se nacházejí metody, které vracejí vždy řetězec, jenž představuje jednu řádku kódu. Názvy jednotlivých metod jsou totožné s možnostmi položky `type` atributu `description`.

Příklad takové metody:

```
public static String click(String id) {
    return "Support.clickElement(\"" + id + "\");\n";
}
```

5.3.5 Třída `DbSupport`

Zde se nachází všechny pomocné metody, které pracují s databází. Metody bylo nutné vytvořit, neboť každý testovací případ je citlivý na data, která mohou být použita, aby test byl realizovatelný. Všechny metody v této třídě vracejí jako návratovou hodnotu množinu dané entity. Většinou je to množina předmětů, v jedné metodě množina studentů.

Aktuálně není sice nutné, aby tyto metody vracely celou množinu dané entity, stačil by jeden zástupce. Ale protože se v budoucnosti očekává, že se budou generovat parametrizované testy, bylo vhodné už vytvářené metody na tuto alternativu připravit.

5.3.6 Třída `Support`

Při práci se zjistilo, že balík `support`, který byl vytvořený k funkcionálním testům UIS, není dostatečný. Například některé jeho metody byly příliš komplexní – konaly více aktivit najednou. Bylo tedy nutné vytvořit jednodušší elementární metody. Za tímto účelem byla stvořená třída `Support`. Obsahuje takové metody, které jsou velmi podobné těm, jenž se nacházejí v balíku `support.utils.web`. Příklad metody třídy `Support`:

```
public static void elementIsDisplayed(String id) {
    UISLogger.testLogger.debug("Support.elementIsDisplayed(
        id = ["  id + "])");
    WebElement element = Utils.findWebElement(id);
    assertTrue("Element with ID=" + id + " is not displayed",
        element.isDisplayed());
}
```

5.4 Balík `utils`

Uvnitř tohoto balíku se nachází třída pro zpracování vstupních XML souborů, třída pro úpravu grafu v SVG formátu a další tři třídy pro generování testů.

5.4.1 Třída `XMLLoader`

Prvním krokem po spuštění Generátoru je načtení vstupních XML souborů – grafu a testovacích případů. Pro zpracování XML souborů bylo zvoleno rozhraní Streaming API for XML, StAX.

Metody pro načítání vstupních XML souborů se nacházejí ve třídě s názvem `XMLLoader`. Ke zpracování souboru s grafem slouží metoda s hlavičkou `public void parseGraphXML(InputStream file)`. Jejím parametrem je XML soubor s grafem jako `InputStream`. Metoda postupně prochází celý soubor po jednotlivých elementech a pokud narazí na element `Node` nebo `Edge`, vezme si z něj potřebné atributy a uloží si ho jako novou instanci třídy `Node` (`Edge`) do svého seznamu uzlů (hran).

5.4.2 Třída `SVGEditor`

Tato třída pracuje s grafem ve formátu SVG, ve kterém obarvuje hrany testovacích případů.

Metoda `colorBlack` má jako jeden z parametrů pole názvů hran. Tyto hrany následně obarví černou barvou a výsledek uloží. Tato metoda se provádí ještě před vygenerováním daného testu, tudíž pokud se test nevygeneruje, lze si snadno prohlédnout plánovanou cestu tohoto testovacího případu.

Další metodou je `colorGreenOrRed`. Jejími dvěma hlavními parametry je seznam hran a dále `int index`. Obarvení probíhá tím způsobem, že se postupně prochází seznam hran a pokud je aktuální index menší než hodnota parametru `index`, hrana se obarví zeleně. Pokud je větší či roven, hrana se obarví červenou barvou. Tuto metodu lze jednoduše využít i k obarvení hran pouze červeně či pouze zeleně – pro zelenou barvu stačí, aby měl parametr `index` například hodnotu `Integer.MAX_VALUE`, a pro červenou například -1.

Metoda `colorSuite` obarvuje agregovaný SVG graf podle právě dokončené celé sady testů. Obarvování grafu vychází z SVG souborů, které vznikly po dokončení jednotlivých testů. Je to sjednocení všech těchto souborů do jednoho grafu. Metoda prochází složku se všemi jednotlivě obarvenými grafy, ve kterých hledá obarvené hrany. Ukládá si je do dvou seznamů podle barev (červené, zelené). Jakmile se načtou data ze všech souborů, ze seznamu červených hran se odeberou všechny hrany zelené. Tedy pokud existuje alespoň jeden případ, který prošel danou hranou bez problémů, je tato hrana zelená. Z výsledného SVG lze také zjistit, kolikrát přes danou hranu všechny testovací případy dohromady prošly. Pokud přes hranu prošlo více testovacích případů, hrana nejprve zvětšuje svou šířku do daného maxima, a pokud už je její šířka maximální, začne se hrana obarvovat tmavějším odstínem zelené (případně červené).

5.4.3 Třídy `StudentGenerator` a `TeacherGenerator`

Jednotlivé testovací případy se vždy týkají buď jen učitele, nebo jen studenta. Ke kontrole a následnému výběru slouží třídy `StudentGenerator` a `TeacherGenerator`. Obě třídy fungují na stejném principu, který bude popsán na příkladu výběru studenta.

Třída `StudentGenerator` má několik veřejných atributů booleovského typu. Při vytvoření instance této třídy jsou všechny tyto atributy nastavené na *false*. Zde je jejich seznam:

- `stuMySubjectsEmpty`
- `stuOtherSubjectsEmpty`
- `stuMyExamDatesEmpty`
- `stuOtherExamDatesEmpty`
- `stuOtherSubjectsLimit`
- `enrolledSubject`
- `unenrolledSubjectWithTeacher`
- `subjectWithExamDate`
- `subjectWithoutStudent`

Ověření, že existuje student, se kterým se může daný test provést, probíhá následovně:

1. Postupně se prochází cesta testovacího případu po jednotlivých uzlech.
   (a) Pokud se v atributu popisu uzlu vyskytne položka `db` s hodnotou z výše uvedeného seznamu, nastaví se tento atribut třídy `StudentGenerator` na *true*.
2. Dle atributů nastavených na *true* se následně vygeneruje množina studentů, nad kterými může být test provedený.

Pokud je výsledná množina prázdná, neexistují vstupní data, se kterými by mohl být daný test provedený.

5.4.4 Třída `TestGenerator`

Uvnitř této třídy se nacházejí metody pro samotné vytváření jednotlivých souborů se zdrojovým kódem testů. Je zde pouze jedna veřejná metoda:

```
generateAllTests(Graph graph, List<String[]> testSituations)
```

V této metodě dochází k vygenerování všech testů a také testovací sady, vše dle parametru `testSituations`. Pro každý testovací případ v parametru se provádí tyto kroky:

1. obarvení cesty testovacího případu černou barvou a uložení jako SVG
2. ověření, zda-li existují data, se kterými lze projít cestu daného testovacího případu
3. vygenerování potřebných dat pro provedení testu (pokud byl předchozí krok úspěšný)
4. vytvoření nového `.java` souboru a zapsání veškerého kódu do něj

Za zmínku stojí také privátní metoda `generateTestCode`, ve které se generuje veškerý kód testu, jenž vykonává nějakou aktivitu. Tato metoda postupně prochází uzly na cestě testovacího případu a u každého uzlu čte položku `type` v atributu `description`. Podle hodnoty této položky zavolá příslušnou stejnojmennou metodu ze třídy `Code` se všemi patřičnými parametry.

5.5 Generované testy

Jak již bylo zmíněno, každý vygenerovaný test dědí od abstraktní třídy `AbstractGeneratedTest`. Všechny vygenerované testy mají také dost podobnou strukturu – všechny obsahují metodu s anotací `@Before` a metodu s anotací `@After`. Poslední metodou v každém vygenerovaném souboru je samotná testovací metoda, která má anotaci `@Test`. Příklad vygenerovaného testu je níže:

```
import org.junit.*;
import uis.generator.support.AbstractGeneratedTest;
import uis.generator.utils.SVGEditor;
import uis.support.basic.Const;
import uis.support.utils.web.Click;
import uis.support.utils.web.Utils;
```

```
import uis.generator.support.Support;
import java.lang.invoke.MethodHandles;
import java.util.ArrayList;
import java.util.Arrays;

public class GeneratedTest_02 extends
   AbstractGeneratedTest {

    @Before
    public void setUpBefore() {
        index = 0;
        edges = new ArrayList<>(Arrays.asList
           ("0001", "0101", "0102", "0103", "0104",
           "1200", "1298", "1299", "0107", "0008",
           "0009"));
        student = dbStudents.getStudent("magenta");
    }

    @After
    public void tearDownAfter() {
        SVGEditor.colorGreenOrRed(edges, index,
           MethodHandles.lookup().lookupClass().
           getName());
    }

    @Test
    public void test_1() {
        Support.getURLAndWaitCheck(url_base + "login
           ");
        index++;
        index++;
        Utils.setText("loginPage.userNameInput",
           student.getUserName());
        index++;
        Utils.setText("loginPage.passwordInput",
           Const.CORRECT_PASSWORD);
        index++;
        Support.clickElement("loginPage.
           loginFormSubmit");
        index++;
```

```
        index++;
        Support.clickAndWaitURLCheck("stu.menu.
           mySubjects", url_base + "student-view/
           mySubjects");
        index++;
        Support.elementIsDisplayed("stu.mySubjects.
           enrolledTable.NoRecords");
        index++;
        index++;
        Support.clickAndWaitURLCheck("header.link.
           logout", url_base + "login?logout");
        index++;
        index++;
        index++;
    }
}
```

5.6 Logování

Bylo by velmi vhodné, aby generované testy poznamenávaly každý vykonaný krok. Díky tomu by bylo jednoduché následně manuálně vyzkoušet daný testovací případ. Rozhodlo se, že generované testy budou tedy všechny své aktivity logovat.

Během práce se zjistilo, že psát jednotlivé logovací zprávy do generovaných testů není efektivní a pouze se tím zbytečně navyšuje velikost vygenerovaných souborů. Místo toho se logovací zprávy nyní nacházejí přímo v metodách s danou aktivitou. To znamená, že logování bylo doplněno (kromě vlastních tříd) do balíku `uis.support.utils.web`. Tím se tedy sekundárním efektem ihned umožnilo logování také u již existujících funkcionálních testů k aplikaci UIS.

Logovací výpis vypadá například takto:

```
DEBUG: Utils.setText(id = [loginPage.userNameInput], newText
   = [purple])
```

5.7 Testovací případy

V Oxygenu lze dle grafu vygenerovat testovací případy. Tyto případy ale mají nereálné požadavky na vstupní data, což bylo do detailů rozvedeno

v části 4.8.4. Z toho důvodu bylo pro tuto práci zvoleno náhradní řešení – manuálně napsané testovací případy. Byly psány dle grafu UIS a v každém testovacím případu se provádí pouze jedna aktivita – například student si zapíše další předmět či učitel vypíše nový zkouškový termín. Celkem obsahuje výsledný soubor 44 testovacích případů.

Zde je ukázka z výše popsaného XML souboru s testovacími případy (celý soubor lze nalézt v příloze A):

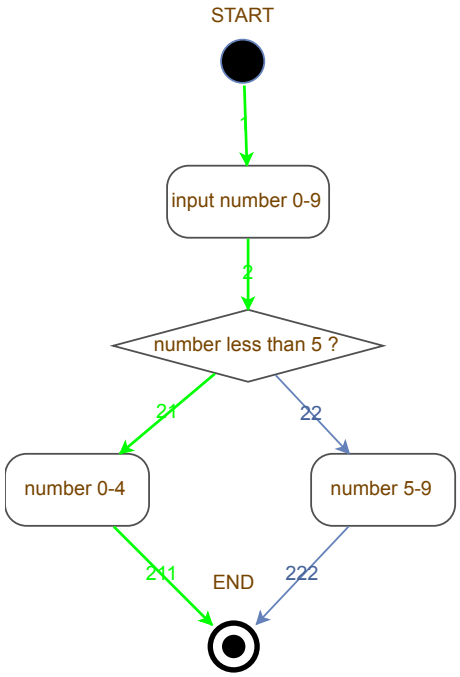
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><
    test_situations name="Test situations 3, TDL= 1, ALG= PCT.
    " note="">

<!-- STUDENT - BEGINNING -->

<!-- Stu Overview -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1100 -
    1101 - 1102 - 1103 - 1104 - 1105 - 1106 - 1099 - 0107 -
    0008 - 0009</test_situation>

<!-- Stu My Subjects -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
    1298 - 1299 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
    1201 - 1202 - 1203 - 1211 - 1212 - 1221 - 1222 - 1223 -
    1099 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
    1201 - 1202 - 1203 - 1211 - 1212 - 1231 - 1232 - 1099 -
    0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
    1201 - 1202 - 1203 - 1213 - 1221 - 1222 - 1223 - 1099 -
    0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
    1201 - 1202 - 1203 - 1213 - 1231 - 1232 - 1099 - 0107 -
    0008 - 0009</test_situation>
```

5.8 Vyobrazení výsledků testů

Jednotlivé testy se generují na základě grafu z Oxygenu, který je možné reprezentovat pomocí XML nebo SVG souboru. Exportovaného grafu ve formátu SVG bylo možné jednoduše využít při reprezentaci výsledků jednotlivých testů či celé testovací sady.

5.8.1 Úspěšný test

Pokud test skončí úspěšně, je celá cesta takového testovacího případu následně obarvená zelenou barvou. Příklad takto obarveného grafu je na obrázku 5.4.



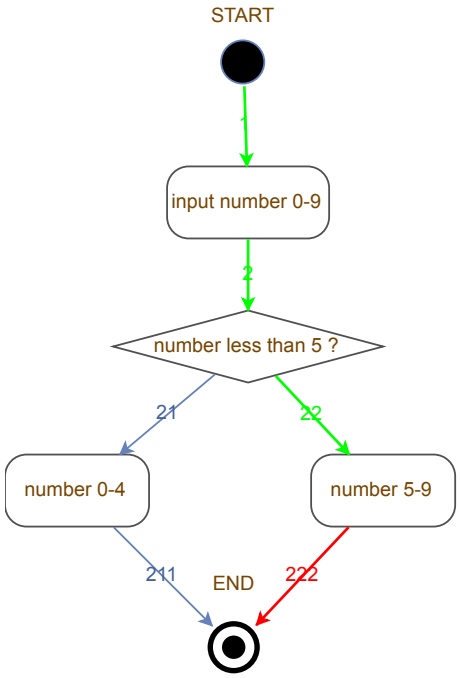
Obrázek 5.4: Výsledek úspěšného testu

5.8.2 Neúspěšný test

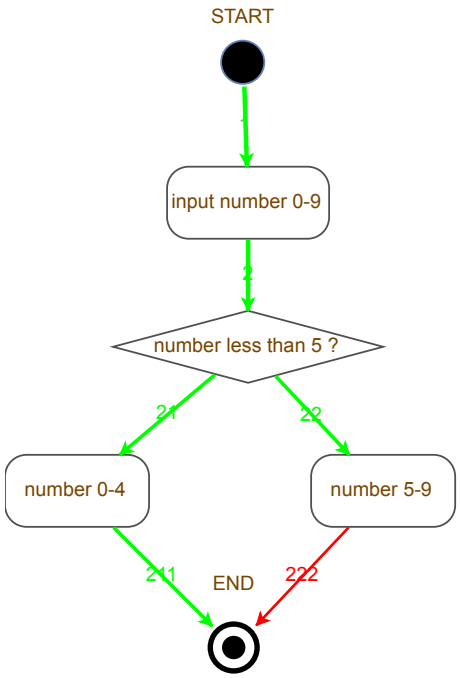
Pokud v průběhu testu dojde k chybě, celkově se test označí jako neúspěšný. Cesta tohoto testu je obarvená zeleně až do uzlu, ve kterém chyba nastala. Od hrany, vycházející z tohoto uzlu, má cesta červenou barvu. Z příkladu na obrázku 5.5 lze jednoduše zjistit, že chyba nastala v uzlu s názvem `number 0-9`.

5.8.3 Sada testů

Obarvování grafu po skončení celé sady testů vychází z SVG souborů, které vznikly po dokončení jednotlivých testů. Postup obarvování byl již popsán v části 5.4.2. Příklad obarveného grafu dle výsledků celé testovací sady je na obrázku 5.6.



Obrázek 5.5: Výsledek neúspěšného testu



Obrázek 5.6: Vyznačení výsledku celé testovací sady

6 Testování

6.1 Testovací prostředí

Veškeré testování probíhalo pod operačním systémem Windows 10. Verze Javy byla 1.8.161. Selenium WebDriver používal driver pro prohlížeč Google Chrome – Chrome driver verze 2.42. Testování neprobíhalo v *headless* módu, což znamená, že daný webový prohlížeč byl vždy plně zobrazený. Webová aplikace byla vždy spuštěná lokálně.

6.2 Ověření na bezporuchové UIS

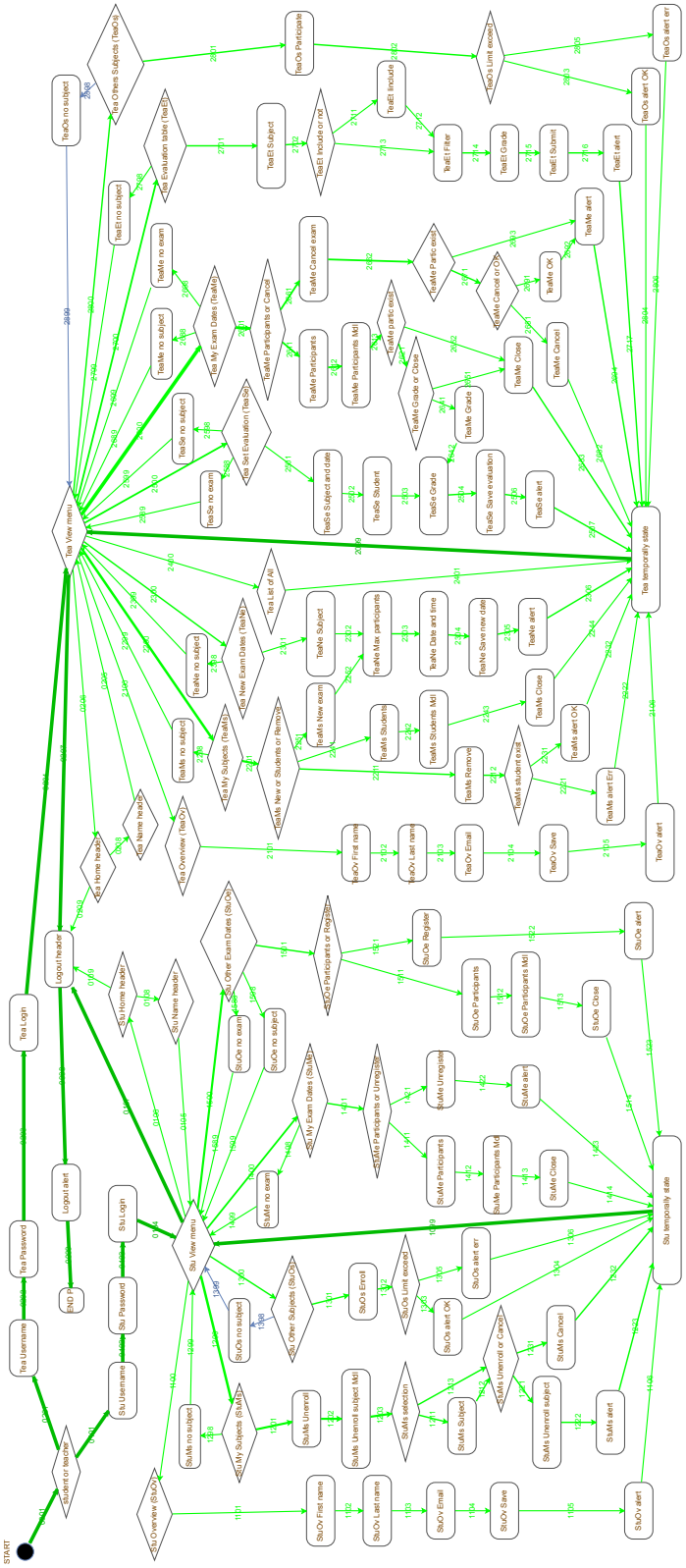
Ověřování vygenerovaných testů probíhalo během výzkumu na bezporuchové (defect free) nejnovější verzi UIS aplikace (verze 1.5.1). Na obrázku 6.1 je vidět, že výsledné testy v aplikaci neodhalily žádnou chybu. V obrázku je patrné, že zde existují cesty (hrany), které zůstaly v modré neutrální barvě. Tyto hrany představují testovací případy:

1. Přihlášený student přejde do záložky `Other Subjects` a v ní se mu objeví prázdná tabulka – neexistují předměty, které ještě nestuduje a mohl by si je zapsat.

2. Přihlášený učitel přejde do záložky `Others' Subjects` a v ní se mu objeví prázdná tabulka – neexistují předměty, které ještě neučí a mohl by je začít učit.

Testy pro tyto testovací případy nebyly vygenerovány, protože pro přednastavená data v databázi neexistuje žádný takový vhodný učitel ani student. Respektive žádný takový ani pro aktuální nastavení databáze existovat nemůže – pro studenta (učitele) je limit, kolik může mít studovaných (vyučovaných) předmětů. Není tedy možné, aby měl nějaký student (učitel) zapsané všechny předměty a tudíž prázdnou tabulku v záložce dalších předmětů k zapsání.

Kromě výše zmíněných neobarvených hran jsou všechny ostatní hrany zelené – vygenerované testy prošly graf a neodhalily žádnou chybu v testované webové aplikaci UIS.



Obrázek 6.1: Ilustrační obrázek výsledků sady testů

6.3 Ověření na poruchových klonech

Výsledné vygenerované testy byly vyzkoušeny na dvou poruchových klonech – *E01StudentService* a *E01TeacherService*. Obě tyto poruchové verze patří pod UIS verzi 1.5.0, což znamená, že zde chybí vstupní data pro dva krajní případy. Prvním je případ, kdy si student, který má zapsán maximální počet předmětů, zkusí zapsat další předmět. Druhý případ pro učitele je obdobný – učitel, který učí již maximální počet předmětů, si chce zapsat ještě další předmět. Z tohoto důvodu nebylo možné pro tyto případy vygenerovat testy. Dále nebyly testy vygenerovány ještě pro dva další případy, stejné jako u bezporuchové verze UIS (viz výše).

6.3.1 Poruchový klon *E01StudentService*

Chybová funkcionalita tohoto poruchového klonu spočívá v tom, že v samotné webové aplikaci UIS metoda `getStudiedSubjectsList()` vrací vždy `null` namísto seznamu studovaných předmětů daného studenta.

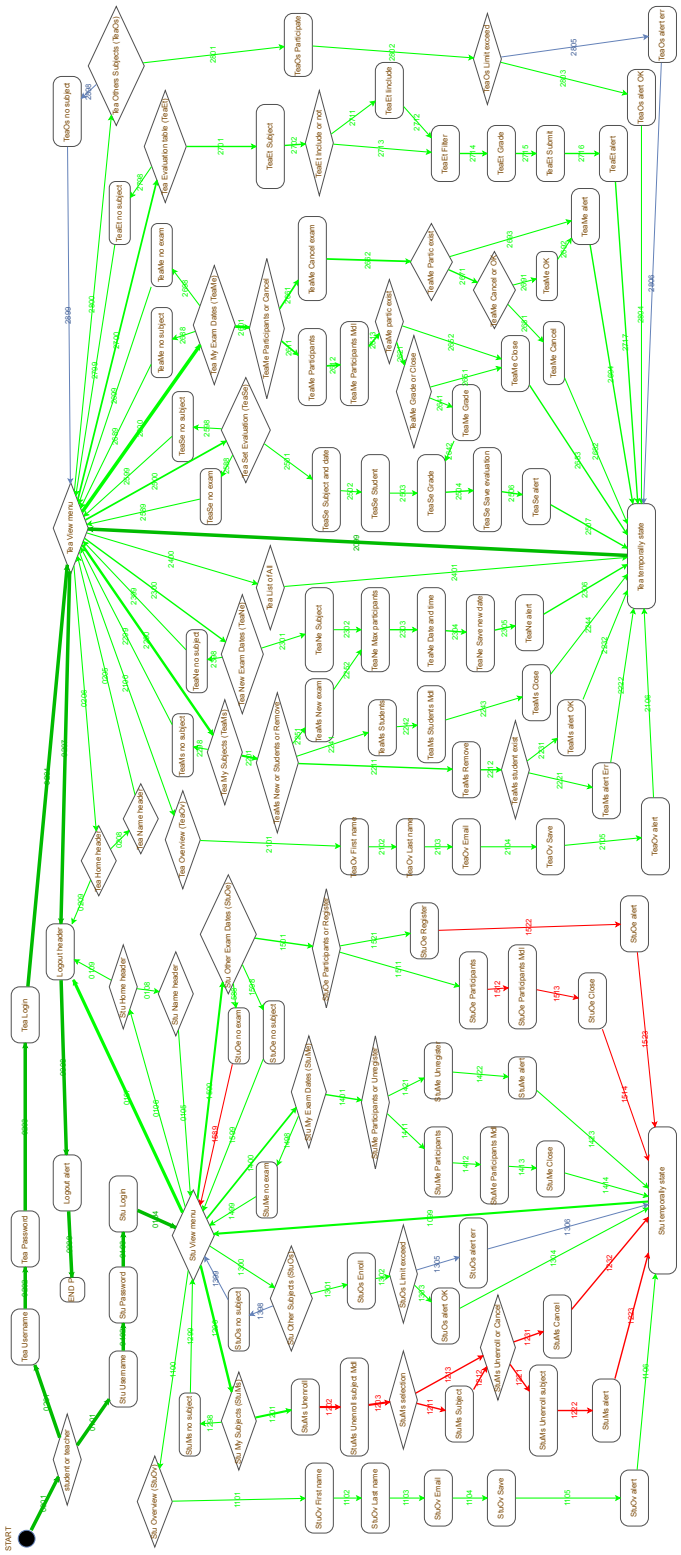
Výše popsaná chyba se projevila v těchto testovacích případech:

1. Student začal provádět odzápis svého předmětu, ale nakonec tuto akci zrušil.
2. Student chtěl provést celkový odzápis svého předmětu.
3. Student, který studuje několik předmětů, ale nemá dostupný žádný zkouškový termín, na který by se mohl přihlásit, navštívil záložku s dostupnými zkouškovými termíny.
4. Student si chtěl prohlédnout další účastníky zkouškového termínu, na kterém je on sám přihlášený.
5. Student se chtěl přihlásit na zkouškový termín ze svého předmětu.

Výsledky testování tohoto poruchového klonu jsou na obrázku 6.2.

6.3.2 Poruchový klon *E01TeacherService*

Tento poruchový klon se týká pouze uživatele v roli přihlášeného učitele. Díky tomu je výsledný graf obarven červeně jen v té části, kde se pracuje s učitelem. Defekt tohoto poruchového klonu UIS spočívá v tom, že metoda webové aplikace UIS `getTaughtSubjectsList()` vrací vždy `null` namísto seznamu vyučovaných předmětů daného učitele.

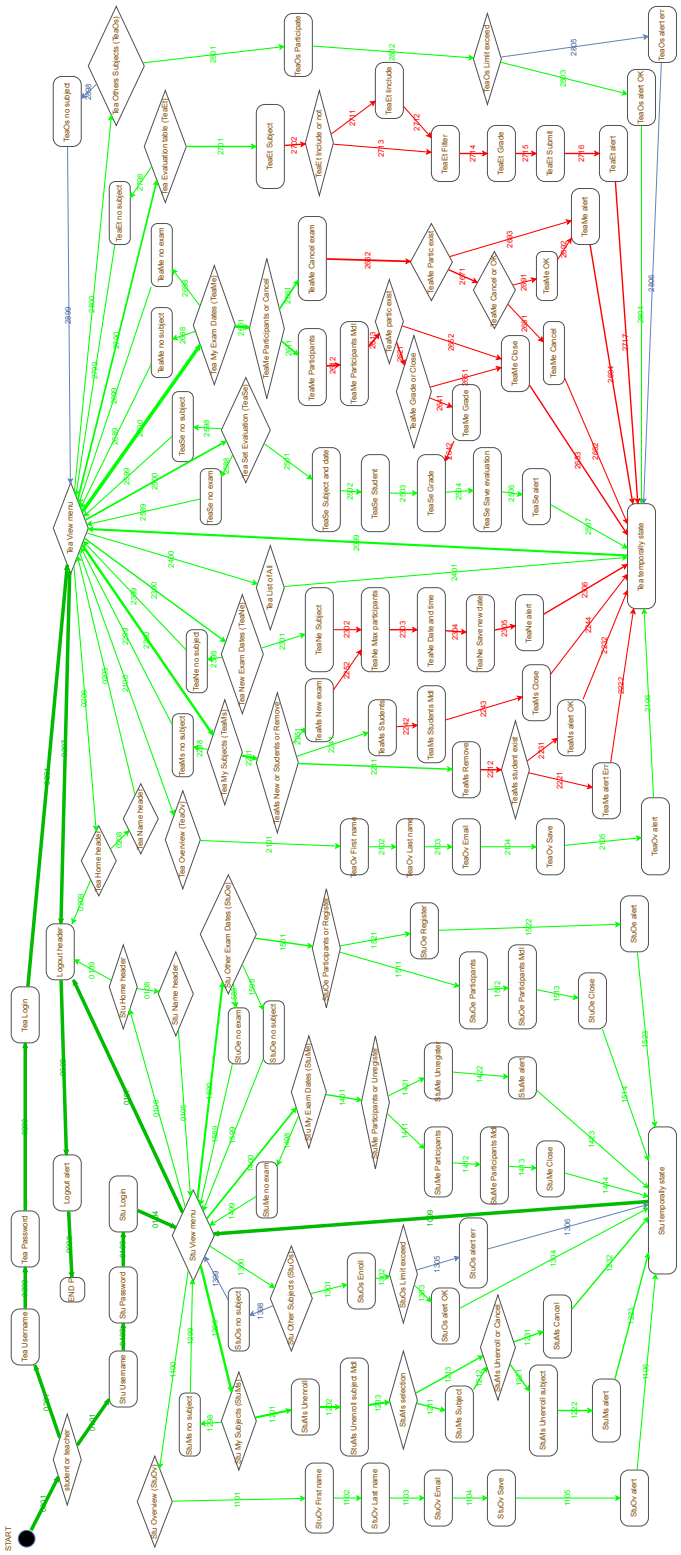


Obrázek 6.2: Ilustrační obrázek výsledků sady testů pro poruchový klon *E01StudentService*

Výše zmíněná chyba se projevila u těchto testovacích případů:

1. Učitel si chtěl odebrat vyučovaný předmět (ať už se studenty či bez nich).
2. Učitel chtěl nahlédnout do seznamu studentů u jeho předmětu.
3. Učitel chtěl vypsat nový zkouškový termín svého předmětu.
4. Učitel chtěl nahlédnout do seznamu studentů, kteří jsou přihlášeni na daný zkouškový termín.
5. Učitel chtěl oznámkovat studenty z daného zkouškového termínu.
6. Učitel chtěl zrušit svůj zkouškový termín (ať už s přihlášenými studenty nebo bez nich).
7. Učitel chtěl ohodnotit studenty svého předmětu.

Výsledky testovací sady, spuštěné nad tímto poruchovým klonem, jsou k nahlédnutí na ilustračním obrázku 6.3.



Obrázek 6.3: Ilustrační obrázek výsledků sady testů pro poruchový klon *E01TeacherService*

7 Závěr

Byla vyvinutá modulární aplikace Generátor, která dokáže na základě dvou XML souborů (grafu a testovacích případů) vygenerovat zdrojové kódy funkcionálních testů. Velikost vytvořené aplikace činí celkem 15 tříd, jejichž celková velikost je 91 kB. Práce byla vyvíjená a testovaná nad experimentální webovou aplikací UIS.

Součástí práce bylo nejprve prostudování všech nástrojů, které budou využívány, a také prozkoumání testované aplikace UIS. Následně byla provedena analýza možných obsahů atributu `description` u uzlů v grafu a do tohoto atributu u všech uzlů grafu byly dopsány všechny nezbytné informace pro generování testů. Poté začal samotný vývoj aplikace.

Během práce se narazilo na jisté problémy. Prvním z nich byl nepřesný graf aplikace UIS, který musel být několikrát doplňován o další uzly a hrany. Dále se zjistilo, že testovací případy, které generuje Oxygen, nejsou pro účel generování testů použitelné. Mají totiž nereálné požadavky na vstupní data. Proto byl pro tuto práci manuálně vytvořený vlastní XML soubor s testovacími případy, které pokrývají celý graf testované aplikace UIS. Nedostatky byly nalezeny také v samotné webové aplikaci UIS – chyběla zde vstupní data na otestování několika krajních situací, většina těchto dat však byla následně doplněna.

Nad rámec zadání práce byla vytvořená vizualizace výsledků jednotlivých testů i celé sady pomocí SVG souboru s grafem testované aplikace. Tím se výrazně zvýšila přehlednost výsledků.

Práce splnila všechny body zadání. Úspěšně se prokázalo, že lze generovat funkcionální testy pro netriviální webové aplikace, a to pomocí pouze grafu této aplikace a testovacích případů. Tento postup lze zobecnit i pro jiné typy aplikací.

Pracnost přípravy kompletního grafu je značná, ale ukazuje se, že je velmi uspokojivě kompenzována výsledným téměř stoprocentním protestováním všech cest v aplikaci. Lze odhadovat, že pro kritické a nebo často vylepšované aplikace je tento postup výhodný. Navíc se prokázalo, že důsledná příprava grafu je jedním z dalších způsobů, jak odhalit existující chyby v testované aplikaci.

Seznam zkratek

CSV	Comma Separated Values – formát pro reprezentaci tabulkových dat
DOM	Document Object Model – model XML, založený na stromové reprezentaci
JSON	JavaScript Object Notation – formát zápisu dat
PCT	Process Cycle Test – algoritmus pro generování testovacích případů v Oxygenu
PPT	Prioritized Process Test – algoritmus pro generování testovacích případů v Oxygenu, uvažující úrovně priorit
SAX	Simple API for XML – rozhraní pro čtení XML souborů, založené na sekvenčním zpracování
StAX	Streaming API for XML – rozhraní pro zpracování XML souborů
SVG	Scalable Vector Graphics – grafický formát souboru využívající XML
TbUIS	Testbed University Information System – projekt pro účely výzkumu testování
TDL	Test Depth Level – kritérium pokrytí testů
UIS	University Information System – webová aplikace
UML	Unified Modeling Language – grafický jazyk pro vizualizaci softwarového programu
XML	eXtensible Markup Language – obecný značkovací jazyk

Literatura

[1] Ammann, P. – Offutt, J. *Introduction to Software Testing.* Cambridge University Press, 2008. ISBN 978-0-521-88038-1.

[2] Herout, P. *Testování pro programátory.* Kopp, 2016. ISBN 978-80-7232-481-1.

[3] Herout, P. *Java a XML.* Kopp, 2012. ISBN 978-80-7232-307-4.

[4] Matyáš, J. *Aplikace s možností injekce chyb pro ověřování kvality testů.* Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2018. Vedoucí práce Pavel Herout.

[5] *Oxygen project (formerly PCTgen)* [online]. STILL – Software Testing IntelLigent Lab, FEL, ČVUT, 2019. [cit. 2019/04/04]. Dostupné z: `http://still.felk.cvut.cz/oxygen/`.

[6] *Automatické testování webových aplikací: souhrnný přehled výsledků* [online]. Root.cz, 2019. [cit. 2019/04/04]. Dostupné z: `https://www.root.cz/clanky/automaticke-testovani-webovych-aplikaci-souhrnny-prehled-vysledku/`.

[7] *Selenium – Web Browser Automation* [online]. Seleniumhq.org, 2019. [cit. 2019/04/17]. Dostupné z: `https://www.seleniumhq.org/`.

[8] Šimečková, L. *Generování testovacích datasetů.* Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2018. Vedoucí práce Pavel Herout.

A Použité testovací případy

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<test_situations name="Test situations 3, TDL= 1, ALG= PCT."
   note="">

<!-- STUDENT - BEGINNING -->

<!-- Stu Overview -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1100 -
   1101 - 1102 - 1103 - 1104 - 1105 - 1106 - 1099 - 0107 -
   0008 - 0009</test_situation>

<!-- Stu My Subjects -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
   1298 - 1299 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
   1201 - 1202 - 1203 - 1211 - 1212 - 1221 - 1222 - 1223 -
   1099 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
   1201 - 1202 - 1203 - 1211 - 1212 - 1231 - 1232 - 1099 -
   0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
   1201 - 1202 - 1203 - 1213 - 1221 - 1222 - 1223 - 1099 -
   0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1200 -
   1201 - 1202 - 1203 - 1213 - 1231 - 1232 - 1099 - 0107 -
   0008 - 0009</test_situation>

<!-- Stu Other Subjects -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1300 -
   1398 - 1399 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1300 -
   1301 - 1302 - 1303 - 1304 - 1099 - 0107 - 0008 - 0009</
   test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1300 -
   1301 - 1302 - 1305 - 1306 - 1099 - 0107 - 0008 - 0009</
   test_situation>

<!-- Stu My Exam Dates -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1400 -
   1498 - 1499 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1400 -
   1401 - 1411 - 1412 - 1413 - 1414 - 1099 - 0107 - 0008 -
   0009</test_situation>
```

```
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1400 -
   1401 - 1421 - 1422 - 1423 - 1099 - 0107 - 0008 - 0009</
   test_situation>

<!-- Stu Other Exam Dates -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1500 -
   1598 - 1599 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1500 -
   1588 - 1589 - 0107 - 0008 - 0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1500 -
   1501 - 1511 - 1512 - 1513 - 1514 - 1099 - 0107 - 0008 -
   0009</test_situation>
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 1500 -
   1501 - 1521 - 1522 - 1523 - 1099 - 0107 - 0008 - 0009</
   test_situation>

<!-- Stu Header -->
<test_situation>0001 - 0101 - 0102 - 0103 - 0104 - 0106 -
   0108 - 0105 - 0106 - 0109 - 0008 - 0009</test_situation>
<!-- STUDENT - END -->

<!-- TEACHER - BEGINNING -->

<!-- Tea Overview -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2100 -
   2101 - 2102 - 2103 - 2104 - 2105 - 2106 - 2099 - 0207 -
   0008 - 0009</test_situation>

<!-- Tea My Subjects -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2200 -
   2298 - 2299 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2200 -
   2201 - 2211 - 2212 - 2221 - 2222 - 2099 - 0207 - 0008 -
   0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2200 -
   2201 - 2211 - 2212 - 2231 - 2232 - 2099 - 0207 - 0008 -
   0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2200 -
   2201 - 2241 - 2242 - 2243 - 2244 - 2099 - 0207 - 0008 -
   0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2200 -
   2201 - 2251 - 2252 - 2303 - 2304 - 2305 - 2306 - 2099-
   0207 - 0008 - 0009</test_situation>

<!-- Tea New Exam Dates -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2300 -
   2398 - 2399 - 0207 - 0008 - 0009</test_situation>
```

```
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2300 -
   2301 - 2302 - 2303 - 2304 - 2305 - 2306 - 2099 - 0207 -
   0008 - 0009</test_situation>

<!-- Tea List of All -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2400 -
   2401 - 2099 - 0207 - 0008 - 0009</test_situation>

<!-- Tea Set Evaluation -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2500 -
   2588 - 2589 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2500 -
   2598 - 2599 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2500 -
   2501 - 2502 - 2503 - 2504 - 2506 - 2507 - 2099 - 0207 -
   0008 - 0009</test_situation>

<!-- Tea My Exam Dates -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2688 - 2689 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2698 - 2699 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2611 - 2612 - 2613 - 2621 - 2641 - 2642 - 2504 -
   2506 - 2507 - 2099 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2611 - 2612 - 2613 - 2621 - 2651 - 2653 - 2099 -
   0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2611 - 2612 - 2613 - 2652 - 2653 - 2099 - 0207 -
   0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2661 - 2662 - 2671 - 2681 - 2682 - 2099 - 0207 -
   0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2661 - 2662 - 2671 - 2691 - 2692 - 2694 - 2099 -
   0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2600 -
   2601 - 2661 - 2662 - 2693 - 2694 - 2099 - 0207 - 0008 -
   0009</test_situation>

<!-- Tea Evaluation Table -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2700 -
   2798 - 2799 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2700 -
   2701 - 2702 - 2713 - 2714 - 2715 - 2716 - 2717 - 2099 -
   0207 - 0008 - 0009</test_situation>
```

```
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2700 -
   2701 - 2702 - 2711 - 2712 - 2714 - 2715 - 2716 - 2717 -
   2099 - 0207 - 0008 - 0009</test_situation>

<!-- Tea Others Subjects -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2800 -
   2898 - 2899 - 0207 - 0008 - 0009</test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2800 -
   2801 - 2802 - 2803 - 2804 - 2099 - 0207 - 0008 - 0009</
   test_situation>
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 2800 -
   2801 - 2802 - 2805 - 2806 - 2099 - 0207 - 0008 - 0009</
   test_situation>

<!-- Tea Header -->
<test_situation>0001 - 0201 - 0202 - 0203 - 0204 - 0206 -
   0208 - 0205 - 0206 - 0209 - 0008 - 0009</test_situation>
<!-- TEACHER - END -->

 </test_situations>
```

B Uživatelská příručka

B.1 Předpoklady

K úspěšnému překladu a spuštění programu dle návodu je zapotřebí:

- operační systém Windows (testováno na verzi Windows 10)
- Java 8 a vyšší (testováno na verzi 1.8.161)
- Maven (potřeba pouze k překladu, použit ve verzi 3.5.3)

Na jiných operačních systémech nebyl program testován, avšak po úpravě příkazů, které se nacházejí v dávkových `bat` souborech, je jistě možné program přeložit a spustit i na jiném operačním systému, než je Windows.

B.2 Příprava

B.2.1 Webová aplikace UIS

Pokud UIS nechceme spouštět na adrese *http://oks.kiv.zcu.cz:10008*, je nutné si připravit databázi a umístit soubor s příponou `war`. Provedeme tedy následující kroky:

1. Nejprve si nainstalujeme nástroj XAMPP.
2. Poté provedeme nastavení databáze v *phpMyAdmin* – založíme novou databázi s těmito údaji:
   - Jméno databáze: uis-web-db
   - Porovnání: UTF8_general_ci
   - Username uživatele: uis-web
   - Heslo uživatele: uis
   - Hostname: localhost
3. V phpMyAdmin v záložce Oprávnění přidáme nového uživatele s výše vypsanými údaji, kterému garantujeme všechna dostupná oprávnění.
4. Vložíme soubor s názvem `uis.war` do složky `xampp\tomcat\webapps\`.
5. Spustíme XAMPP a v něm MySQL a Tomcat.

Nakonec je UIS přístupná na adrese *http://localhost:8080/uis/*

B.2.2 Konfigurační soubor

Do výsledného spustitelného `jar` souboru se vkládá také konfigurační soubor a všechna další vstupní data. Je tedy nutné si před překladem programu vše připravit – všechny tyto soubory se nacházejí ve složce `src\main\resources`.

Uvnitř této složky se nachází graf ve formátech XML i SVG, testovací případy ve formátu XML a dále textové soubory s obsahem databáze UIS. Posledním souborem je konfigurační soubor `configurations.txt`. V něm si lze nastavit, která URL adresa UIS má být použita (lokální nebo *http://oks.kiv.zcu.cz:10008*), jaký webový prohlížeč má být spouštěný a cesty k driverům jednotlivých webových prohlížečů. Nakonec se dá nastavit, zda má být prohlížeč spuštěný v *headless* módu (což znamená, že by se nezobrazoval, jen by pracoval na pozadí) a zda se mají ukládat snímky obrazovky, pořizované při selhání testu.

B.3 Překlad a spuštění

Uvnitř adresáře, kde se nachází složka se zdrojovými kódy (`src`) a soubor `pom.xml`, spustíme soubor `compile.bat`.

Po jeho úspěšném provedení vznikne složka `target`, kde se nacházejí další podadresáře a také soubor `TbUIS-generator-jar-with-dependencies.jar`. Pro generování testů stačí tento `jar` soubor spustit.

Jakmile se ukončí generování testů, najdeme v dané složce dva nové adresáře. V prvním, který má předponu `generator_path_svgs`, se nacházejí jednotlivé SVG soubory (pro každý testovací případ právě jeden). V každém souboru je vyznačena cesta daného testovacího případu. Druhý nově vytvořený adresář má předponu `generated_tests`. Nachází se zde všechny vygenerované testy včetně *test suite*. Přímo dovnitř tohoto adresáře přesuneme soubor `run_test.bat`. Jeho spuštěním začne překlad vygenerovaných testů a zároveň jejich spuštění jako sady.

Po skončení testování se v dané složce nacházejí logovací soubory. Dále se zde nachází nová složka s předponou `after_test`, která obsahuje SVG soubory s vizualizacemi výsledků jednotlivých testů i celé testovací sady.

C Obsah CD

- `Generator` - obsahuje vytvořenou aplikaci, tedy veškeré zdrojové kódy se vstupními daty (databáze, graf, testovací případy) a přeložené soubory se spustitelným `jar` souborem
- `UIS` - obsahuje čtyři `war` soubory, jeden je nezávadná UIS verze 1.5.0, druhý je nezávadná UIS verze 1.5.1 a zbylé dva soubory jsou poruchové klony
- `drivery` - obsahuje drivery pro prohlížeče Google Chrome, Firefox a Operu
- `text_prace` - obsahuje zdrojové soubory textu a také PDF soubor s textem bakalářské práce
- db_UIS_1-5-1 - obsahuje databázová data pro UIS verze 1.5.1