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

# Diplomová práce

# Akceptační testování v projektu TbUIS

Plzeň, 2020 Radek Vais

**Místo této strany bude zadání práce.**

# Prohlášení

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

V Plzni dne 19. května 2020

Radek Vais

# Poděkování

Je moji milou povinností poděkovat panu doc. Ing. Pavlu Heroutovi, Ph.D. za odborné vedení, ochotu, se kterou se mnou problematiku konzultoval, ale hlavně za čas, který mi při tom všem věnoval.

# Abstract

This thesis deals with acceptance testing and selection of a suitable tool for the implementation of automatic acceptance tests of the UIS application. Part of the tool selection is an overview of the properties of commonly used alternatives. Furthermore, the thesis describes the procedure of solution design, scenario design and implementation of a suite of automatic acceptance tests using the Robot Framework tool. Part of the project preparation is a technical solution for the integration of the already existing SupportLibrary. The implemented suite of acceptance tests was validated against 28 standard UIS defect clones. The suite of acceptance tests, as expected, reveals errors in the application's dynamic data. It has been verified that the use of a specialized framework for acceptance tests increases the speed and efficiency of test creation. It has been verified that Robot framework is handy tool for creating comprehensive test scenarios.



# Abstrakt

Práce se zabývá akceptačním testováním a výběrem vhodného nástroje pro realizaci automatických akceptačních testů existující aplikace UIS. Součástí výběru nástroje je přehled vlastností běžně používaných alternativ nástrojů AAT. Dále práce popisuje postup návrhu řešení a návrhu scénářů. Na základě toho je v práci realizována sada 122 automatických akceptačních testů s využitím prostředků nástroje Robot Framework. Součástí přípravy projektu je i technické řešení integrace již existující knihovny SupportLibrary. Výsledná sada akceptačních testů byla validována proti 28 standardním poruchovým klonům aplikace UIS. Sada akceptačních testů (dle očekávání) odhaluje chyby dynamických dat aplikace. Bylo ověřeno, že použití specializovaného frameworku pro akceptační testy zvyšuje rychlost a efektivitu tvorby testů. Dále se potvrdilo, že Robot Framework je velmi vhodný pro zápis rozsáhlých scénářů automatizovaných testů.

# Obsah

# 1 Úvod

Cílem této práce je prozkoumat dostupné postupy a nástroje pro akceptační testování software, zhodnotit jejich vlastnosti a vytvořit sadu automatických akceptačních testů pro konkrétní aplikaci.

Prvotním východiskem práce je výběr vhodného nástroje pro vytváření akceptačních testů, které budou sloužit k testování aplikace UIS z projektu TbUIS. Dále bude prostřednictvím zvoleného nástroje připravena sada akceptačních testů této aplikace. Tato sada bude vytvořena na základě use-case specifikace systému.

Vytvořená sada akceptačních testů pomůže k ověření stávající funkcionality aplikace UIS. Sada akceptačních testů vhodně doplní stávající test-mix aplikace. Pod pojmem test-mix rozumíme již existující soubor sad testů různých typů (jednotkové testy, funkcionální testy, zátěžové testy).

V případě, že při testování dojde k odhalení chyb v aplikaci UIS, bude třeba nalezené chyby opravit, aby byl stále k dispozici bezporuchový klon aplikace (etalon). Oprava takovýchto chyb bude jedním z výstupů této práce.

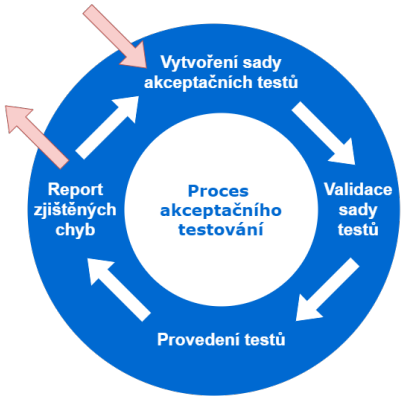
Výsledky práce by měly směřovat ke zhodnocení přínosu zvoleného nástroje pro automatické akceptační testování, a to jak z pohledu efektivity odhalování chyb, tak i z pohledu rychlosti vývoje a udržování testovacích scénářů.

# 2 State of the Art

## 2.1 Akceptační testování

Akceptační testování (dále též jen AT) je disciplína testování software, která „zastupuje zákazníka“ v procesu ověřování kvality produktu. Zastoupením zákazníka chápeme mechanismy ověření, že výsledný program obsahuje smluvenou funkcionalitu nebo vlastnosti. Hlavním cílem této disciplíny je minimalizace pravděpodobnosti, že software nepracuje správně. Doporučovanou praxí je provádění akceptačních testů zákazníkem. Přínosem takového postupu je vyšší byznysová kvalifikace[1] testerů aplikace.

Důležitost procesu akceptačního testování je podtržena faktem, že každý software musí mít jasně definován hotový stav (definition of done – DoD). Právě akceptační testy jsou vhodným nástrojem, jak tento stav snadno definovat a měřit. Základní metrikou DoD tedy může být procento úspěšně dokončených akceptačních testů. Je třeba připomenout, že akceptační testování je proces, který obsahuje následující cyklus činností:




Obrázek 2.1: Proces tvorby akceptačních testů.

Fakt, že akceptační testování je proces, je v konfliktu s využitím akceptačních testů jako DoD. Představme si následující situaci (blíže viz [10]): V prvním kole testování bylo úspěšně provedeno 22 % testů. Ve druhém kole pak bylo úspěšně provedeno pouze 15 % testů. Podle výše zmíněné naivní metriky se tedy stav aplikace zhoršil. Naivní metrika nijak nepokrývá celkový počet testů. V uvedeném případě nešlo o zhoršení stavu, ale o zvýšení

[1] Testeři z řad zákazníka přesně znají požadavky na chování aplikace. Testery také mohou být skuteční uživatelé aplikace.

pokrytí akceptačních testů (vyšší počet, nové scénáře). Z tohoto důvodu je vhodné sadu akceptačních testů ve správný moment fixovat, aby mohla sloužit jako metrika pro DoD. Z pohledu životního cyklu SW je zafixování sady AT mechanismus, jak jasně ohraničit release[2] aplikace.

### 2.1.1 Kategorie testů

Disciplína testování software zavádí řadu různých kategorií a kategorizací. Zpravidla se rozlišují následující typy testů:

- dle typu spuštění aplikace – statické/dynamické
- dle typu spuštění testu – manuální/automatické
- dle znalosti programu – black/gray/white box
- dle typu testu – jednotkové, integrační, systémové, smoke, regresní, výkonnostní, akceptační

Statické testy slouží k identifikaci „nebezpečných“ (potenciálně chybových) konstrukcí v programu již na úrovni jeho zdrojového kódu. Jedná se o kontrolu dodržení omezení standardu projektu, která mají za cíl předcházet chybám. V případě bezpečnostně kritických aplikací programovaných v jazyce C/C++ je běžným standardem zákaz alokace paměti na haldě (viz MISRA-C++ nebo JSF C++ [11]). Úlohou statických testů je odhalit tyto prohřešky. V případě použití jazyka Java, lze pro statické testy použít nástroj PMD, který provede statickou analýzu kódu a díky množině pravidel nástroj označí potenciálně chybová místa.

Kategorizace dle znalosti programu určuje, do jaké míry známe zdrojový kód aplikace. Testovaná aplikace je připodobňována ke krabičce. V případě tzv. černé skříňky (black box) je známo jen rozhraní aplikace, tester nemá k dispozici žádné informace o vnitřním fungování testované komponenty nebo aplikace. V případě tzv. šedé skříňky jde o obecnou znalost fungování komponenty např. jaký algoritmus byl použit. Tester stále nemá k dispozici přesný popis volaných metod a vnitřních procesů. Poslední kategorií je skříňka průhledná (white box). V tomto případě má tester k dispozici všechny informace o testovaném programu – jeho zdrojový kód.

Akceptační testy jsou zpravidla dynamické, výjimku tvoří požadavky na strukturu programu definované specifikací programu[3]. Pro definici vztahů typů testů k akceptačním testům je důležité připomenout základní vlastnosti některých typů testů.

[2] Vydání nové verze aplikace.

[3] Vynucená statická analýza kódu viz [11]

Jednotkové (unit) testy testují funkcionalitu jednotlivých tříd a funkcí programu (základních programových jednotek). Jednotkové testy zpravidla připravuje vývojář. Tyto testy jsou silně závislé na použité platformě. Existuje mnoho nástrojů (frameworků) pro implementaci jednotkových testů (např. JUnit, NUnit, MSUnit, ...).

Integrační testy slouží k ověření technologického propojení komponent systému. Propojované systémy nemusejí obsahovat finální funkcionalitu, ale je nezbytné, aby jejich rozhraní mělo finální podobu. Příkladem může být webová služba, která poskytuje statické datové výstupy. V rámci integračních testů neprobíhají testy konfigurace „produkčního“ prostředí.

Regresní testy lze považovat za podmnožinu akceptačních testů. Jedním z jasných požadavků nové verze je zachování původní funkcionality.

Systémové testy ověřují kompletní mechanismus nasazení a konfigurace systému. Běžně probíhají na straně výrobce SW. Jde o testy na plně integrovaném prostředí. V rámci systémových testů tedy probíhá i kontrola mechanismů nastavení prostředí.

Akceptační testy (též uživatelské akceptační testy[4]) tedy slouží k ověření korektního chování systému v prostředí velice blízkém produkčnímu.

### 2.1.2 Akceptační prostředí

Akceptační prostředí simuluje prostředí produkční pro potřeby testů. Všechny systémy v tomto prostředí by tedy měly maximálně odpovídat prostředí, ve kterém bude finální aplikace nasazena. V ideálním případě jde o klon aplikací a infrastruktury. Nicméně toto řešení je pro velké podniky velice nákladné. Zároveň složitost tohoto prostředí je přímo úměrná složitosti a závislostem aplikace.

V rámci tohoto rozboru budeme uvažovat pouze webové či instalované aplikace pro osobní počítače[5].

Uvažujeme-li standalone instalovanou aplikaci, pro akceptační prostředí je potřeba definovat pouze parametry počítače, protože standalone aplikace nemá žádné další závislosti. Přidáme-li do úvahy, že instalovaná aplikace používá sdílenou databázi entit[6], akceptační prostředí se rozšiřuje o nastavení verze databáze, fyzického spojení s databází a aplikační konfiguraci použité databáze. Obdobným způsobem se specifikace akceptačního prostředí

[4]Z anglického UAT – user acceptance tests

[5]Testování mobilních aplikací přináší mnoho problémů způsobených nepřeberným počtem mobilních zařízení a verzí operačních systémů. Pouhé prostředí osobních počítačů představuje dostatečnou rozmanitost pro rozbor problémů.

[6]Databáze entit obsahuje informace, které systém zpřístupňuje. V případě univerzitního informačního systému může jít o záznamy studentů, učitelů, ...

rozšiřuje v případě, že aplikace využívá nějakou službu[7].

V případě webové monolitické aplikace je situace obdobná (hardware, databázové spojení). Pro webovou aplikaci se konfigurace akceptačního prostředí rozšiřuje o definici webového prohlížeče. Doposud lze konfigurace zmiňovaných prostředí považovat za snadno řešitelné. Ale situaci značně komplikuje microservice[8] architektura aplikací. Pro distribuovaný systém je třeba udržovat konfiguraci využívaných a poskytovaných rozhraní, případně informaci o jejich kompatibilitě.

### 2.1.3 Příprava dat

Samostatnou kapitolou akceptačního testování je příprava testovacích dat. Pro přípravu dat existuje několik běžně používaných postupů. Různé postupy přinášejí specifické výhody a nevýhody. Výběr přístupu tedy závisí na typu testované aplikace, případně na konkrétním testovacím scénáři.

Prvním přístupem je konstrukce testů s předpokladem prázdné databáze entit testovaného systému. Každý jednotlivý test je vytvářen nezávisle na datech, potřebná data pro test jsou vytvořena v rámci přípravné fáze (setup). Tento přístup s sebou nese velkou výhodu čistoty testovacího prostředí. Čistotou testovacího prostředí chápeme fakt, že test není ovlivněn chybou dat (všechna byla vytvořena před spuštěním testu). Na druhou stranu tento přístup nese velkou nevýhodu. Všechna data je třeba připravit při spuštění scénáře. Toto vytváření dat zatěžuje test někdy zbytečnou závislostí na funkčnosti aplikace. Uvažujme prostředí UIS (viz kapitola 3). Pro korektní otestování přihlášení studenta na zkoušku je třeba úspěšně projít následující scénáře (UC – use cases): vytvoření studenta, učitele, předmětu, spuštění výuky předmětu, zapsání studenta, vypsání zkoušky. Až v tuto chvíli je možné skutečně otestovat scénář přihlášení na zkoušku. Tento přístup k přípravě dat je vhodný pro scénáře, které vytvářejí základní datové entity v aplikaci (např. založení uživatele).

Dalším přístupem je inicializace databáze do výchozího stavu. Během fáze spuštění testu proběhne nastavení dat aplikace mechanismem nezávislým na funkcionalitě testované aplikace (např. přímým záznamem do databáze). Tento mechanismus redukuje závislost testu na ostatních funkčnostech aplikace nahrazením aplikační tvorby dat, což je výhodou tohoto přístupu.

[7]Není zásadní, zda jde o webovou či jinou instalovanou službu.

[8]Softwarový architektonický styl, který využívá faktu, že síťové spojení je spolehlivé. Jednotlivé části systému (např. správa uživatelů, autorizace, kontakty, ...) jsou nezávislé aplikace někde „na síti“. K vyvolání funkcionalit jednotlivých komponent je použito síťové spojení. Tyto komponenty jsou zpravidla distribuované do různých částí sítě, případně jsou duplikované pro lepší dostupnost.

Nevýhodou tohoto přístupu je případná chyba v založení dat (důvodem je jiný mechanismus) a v některých případech nutnost korigovat časové značky v datech. Uvažujme následující situaci: Cílem testera je vytvořit scénář ověřující přihlášení na zkoušku, respektive nemožnost přihlášení. Nelze se přihlásit na zkoušku v minulosti. Očekávaný výsledek testera tedy je, že bude mít k dispozici přihlášení na následující termín, nebude mít možnost přihlášení na předchozí termín a zároveň učitel bude moci zadávat hodnocení k termínu v minulosti.

Výše popsaný scénář demonstruje základní problém „časových" dat. Existují domény, kde je řešení časových závislostí mnohem složitější. Příkladem může být bankovnictví, kde často nejde o komplexní závislosti, ale o dlouhé periody opakování. Například k dosažení stavu domicilace[9] je třeba připravit v historii plateb patřičné údaje s měsíční periodou.

Posledním přístupem je klonování produkčních dat. Tento přístup přináší nejvěrnější data pro testování aplikace („jak se opravdu používá"). Je vhodný a žádoucí pro dlouholeté aplikace, především pro testování zpětné kompatibility datového modelu. Výhodou je, že vývojáři, respektive testeři, mají přístup k datům skutečně vytvořeným aplikací, kde jsou časová data korektně vytvořená. Data přirozeně obsahují uživatelské chyby[10], což může být vnímáno pozitivně (odhalení chyb aplikace) i negativně (pády testů pro špatná data). Data pro jednotlivé testy je třeba hledat, tester tedy musí dokonale ovládat chování testované aplikace a obecné principy v doméně aplikace. V případě UIS je doménou aplikace vysokoškolská výuka. Tester si tak musí být vědom relací mezi studenty a učiteli nebo jiná přirozená omezení systému (např. je dán počet pokusů ke složení zkoušky).

Velkou nevýhodou klonování produkčních dat je nutnost tato data anonymizovat. Nejen v souvislosti s GDPR[11] je třeba přenášená data zbavit osobních údajů. Zároveň je důležité monitorovat, kdo k takové databázi má přístup (riziko při outsourcingu softwarových služeb).

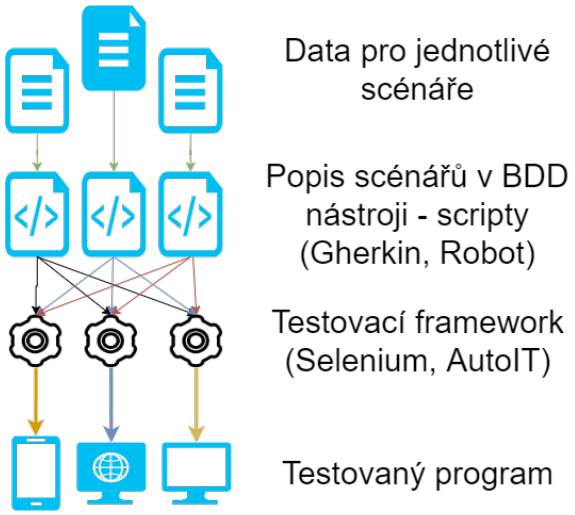
[9]Domicilace zavazuje klienta vůči bance převádět všechny příjmy na účet který mu banka poskytla. Zpravidla klient díky domicilaci získá výhodnější úrok na hypotéce.

[10]Uživatelské chyby mohou být způsobeny kopírováním textu ve špatném kódování, obcházením aplikace...

[11]Obecné nařízení o ochraně osobních údajů (angl. General Data Protection Regulation neboli GDPR) je nová legislativa EU, která výrazně zvyšuje ochranu osobních dat občanů.

### 2.1.4 Technika behavior-driven development

Behavior-driven development (BDD) je technika test-driven developmentu [12], kdy se do procesu přípravy testů významně zapojuje zákazník. Podle obecné test-driven metodiky se i pro BDD aplikuje metoda nezávislosti vývoje testů a aplikace. Testy vznikají i dříve než samotná aplikace (vice viz [1] nebo [2]).




Obrázek 2.2: Schéma vztahu elementů automatického akceptačního testování: data testů, popis scénářů, automatizační SW a testovaný systém.

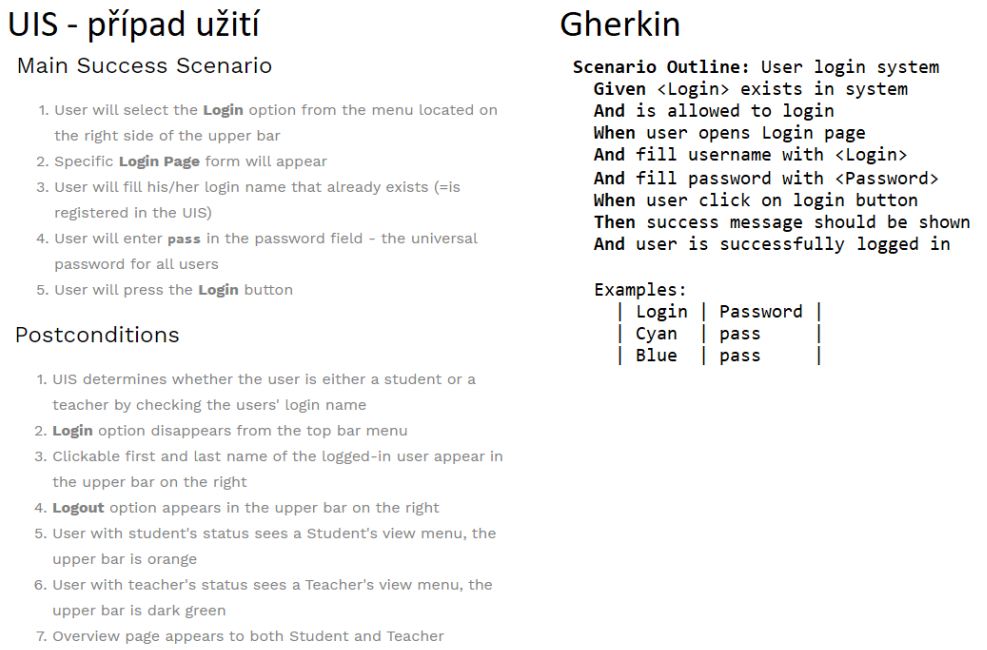
Jedním z cílů metodiky je maximální zapojení zákazníka do procesu ověření aplikace a přípravy testů. Z tohoto důvodu umožňují nástroje vyvíjené pro vytváření testovacích scénářů zápis scénářů téměř v přirozeném jazyce. Technická úroveň použitého jazyka závisí na úrovni připravených abstrakčních vrstev v rámci projektu. Za lépe připravené vrstvy považujeme ty, jejichž použití se blíží přirozenému jazyku. Z tohoto důvodu je žádané v rámci integrace vytvořit prostředí pro zápis scénářů pomocí akcí odpovídajících byznysovým akcím zákazníka, které jsou nezávislé na použité technologii výsledné aplikace.

Uvažujme příklad z praxe. Informační systém vysoké školy spravuje stále stejné procesy (přihlašování na zkoušky, zadávání známek, ...). Je tedy žádoucí definovat postup operací, scénáře, a jejich vstupní data (parametry a očekávané výsledky). Informační systém může mít více různých rozhraní (webová aplikace, mobilní aplikace, ...), které se musí chovat stejně, nebo

[12] Test-driven development je technika vývoje aplikace na základě provádění častých testů, které určují chování výsledku

alespoň konzistentně[13]. Díky možnostem BDD frameworků je snadné vytvořit integrační vrstvy pro jednotlivé cílové aplikace (web, mobil, ...). Na Obrázku 2.2 je znázorněna výsledná nezávislost dat na scénářích akcí a nezávislost scénářů akcí na automatizační vrstvě. Na diagramu znázorněné optimální nezávislosti lze v praxi jen těžko dosáhnout[14].

Napříč nástroji pro BDD je rozšířena syntaxe jazyka Gherkin, která je obecně považovaná za syntaxi zápisu BDD scénářů. Na Obrázku 2.3 můžeme porovnat dokumentaci aplikace – případ užití a stejný scénář zapsaný v jazyce Gherkin s použitím akcí z „byznysové“ abstrakční vrstvy.




Obrázek 2.3: Porovnání případu užití v přirozeném jazyce a scénáře pro test daného UC v jazyce Gherkin.

Nabízí se úvaha, že jednotlivé scénáře jsou kopií dokumentace aplikace a přináší jen další prostor pro chybu. Toto tvrzení lze jen těžko rozporovat, testovací scénáře jsou konkrétní kroky uživatele již dříve popsané dokumentací [5] (pro představu lze použít obdobnou abstrakci jako definice třídy podprogramu a její instance). Nicméně zapojení zákazníka do procesu přípravy

[13]Příklad rozdílného konzistentního chování: 1) V rámci mobilní aplikace lze zadat termín zkoušky nejpozději 24h před jejím začátkem. 2) V rámci desktopové aplikace je toto omezení zmenšeno na 1h před začátkem. Obě aplikace se chovají konzistentně – nelze vytvořit zkouškový termín v minulosti a zároveň jsou pro vytvoření zkouškového termínu potřeba stejná data.

[14]Důvodem může být odlišnost platforem (např. operační systém), nebo doba zpracování požadavků (např. mobilní zařízení čeká na WiFi připojení).

těchto scénářů má velký přínos i přes riziko chyb. V brzkém začátku procesu se odhalí zásadní chyby v návrhu použití aplikace, protože zákazník vidí, jakým způsobem se aplikace bude používat. Zároveň přípravou scénářů, které ověřují alternativní toky, a určením jejich závažností (criticality) zákazník určuje priority či další omezení, která nebyla odhalena v úvodní analýze.

## 2.2 Automatické akceptační testování

Tato část je věnována významu automatizace akceptačního testování. Nástroje automatického akceptačního testování (AAT) nabývají v posledních letech na důležitosti, protože mnoho softwarových produktů přechází na metodiky CI (continous integration). Tento přechod je způsobený vysokou dostupností nástrojů pro CI a jejich „vybaveností“. Dnes nejrozšířenější nástroje byly vydány přibližně před deseti lety: Jenkins (2011), TeamCity (2006), Team foundation server – TFS (2006). Význam těchto nástrojů se zvýšil s příchodem komplexních DevOps[15] (developer operations) nástrojů dostupných v podobě cloudových řešení. Příkladem může být integrace TFS na platformě Microsoft Azure, kdy vývojář jedním klikem ve vývojovém prostředí (IDE) provede nasazení, spuštění a testy aplikace.

Je prakticky nemožné každý den[16] otestovat kompletní funkcionalitu SW, jak vyžadují metodiky CI. Pomocným nástrojem jsou jednotkové testy, které zpravidla netestují kompletní nasazenou aplikaci, nýbrž její části častokrát v dočasné konfiguraci. Od automatických akceptačních testů očekáváme ověření funkčnosti aplikace v akceptačním prostředí (viz 2.1.2).

Zásadním problémem AAT je příprava testovacích dat. Tento problém je patrný především v aplikacích s dlouhým časovým průběhem (např. připočítávání měsíčního úroku). Testování takových funkcí je složité zařadit do CI testovací sady.

### 2.2.1 Dostupné nástroje

Následující tabulka (2.1) shrnuje dostupné nástroje pro AAT, které by měly být uvažovány jako vhodné varianty pro tvorbu akceptačních testů pro vytvářený projekt (viz [3] a [6]). V následující části bude podrobněji představen každý zmíněný produkt. Nástroje AAT slouží především ke snížení potřebných technických znalostí pro vytváření testu, snadné opakovatelnosti a oddělení scénáře od testovacích dat.

[15] Jedním z procesů DevOps je CI – continous integration

[16] V extrémním případě nejde jen o jeden den, nýbrž o každý přírůstek kódu (commit).

| Nástroj | Výrobce | Domovské stránky |
|---|---|---|
| Behat | community | behat.org |
| Cucumber | SmartBear | cucumber.io |
| FitNesse | community | fitnesse.org |
| Ranorex | Idera | ranorex.com |
| Robot Framework | community | robotframework.org |
| Screenster | Screenster | screenster.io |
| Testim | Testim | testim.io |

Tabulka 2.1: Nejznámější dostupné nástroje pro AAT.

### Behat

Jde o neoficiální verzi frameworku Cucumber (viz dále) pro platformu programovacího jazyka PHP. Je k dispozici zdarma. Pro zápis testů používá syntaxi scénářů Gherkin. Za vývojem nástroje stojí komunita open source vývojářů.

Poslední release byl v únoru 2020.

### Cucumber

Jde o testovací nástroj vyvíjený společností SmartBear. Do roku 2019 šlo o open source projekt, o který se starala početná komunita vývojářů. Zmíněná komunita vývojářů u projektu zůstala a podílí se na oficiálních, či neoficiálních částech frameworku Cucumber.

Existují integrace do technologických zásobníků běžně užívaných programovacích jazyků: Java, Kotlin (modifikace JVM), JavaScript, Ruby, Go, Lua. Pro jazyk Java je dostupná integrace nástroje Selenium k testování webových aplikací. V případě potřeby testování klasické GUI aplikace je třeba připravit vrstvu GUI automation.

Framework je v community verzi k dispozici zdarma, uživatel musí vytvářet scénáře „ručně“. Zároveň společnost SmartBear poskytuje program CucumberStudio pro správu testů, vytváření šablon a jejich spouštění.

Poslední release byl v únoru 2020.

### FitnNesse

Jde o webový nástroj umožňující paralelní spolupráci celého týmu zapouzdřující framework FIT (Framework for Integrated Testing) pro tvorbu AAT. Nástroj umožňuje snadnou spolupráci a správu akceptačních testovacích scénářů. Systém pracuje v prostředí Java a podporuje aplikace v běžně používaných programovacích jazycích: Java, .Net, Ruby, Python, C a PHP. Pro tes-

tování GUI[17] aplikací nebo webů je třeba doprogramovat abstrakční vrstvu ovládání (např použití Selenium frameworku).

Testy probíhají na serveru, nutná podmínka pro vytváření testů je webový prohlížeč. Tento nástroj je výhodný pro případ spolupráce velkého týmu.

Scénáře jsou zapisovány ve formě tabulek viz Obrázek 2.4. Nutnou podmínkou tohoto zápisu je vytvoření metod použitých v příkladu (*Fill, Check logged user, ...*), které zapouzdřují integrační vrstvu pro strojové ovládání aplikace.


Obrázek 2.4: Ukázka BDD scénáře v rámci nástroje FitNesse

Poslední release byl v únoru 2020, verze v20200205.

### Ranorex

Jde o komerční nástroj cílený na aplikace vytvářené v prostředí .Net C#. Nástroj umožňuje jak psaní scénářů kódem, tak nahrávání úkonů v aplikaci. Nástroj má silnou integraci v prostředí systému Microsoft Windows a vývojové prostředí MS Visual Studio.

Poslední release byl v únoru 2020, verze 9.3.0.

[17] Samostatné aplikace s uživatelským rozhraním.

### Robot Framework

Robot Framework je nástroj pro automatizaci akceptačních testů. Běhovým prostředím nástroje je Python. Jde o nástroj vytvářený komunitou, zároveň je založena nadace pro podporu Robot Frameworku, kam přispívají velké firmy na rozvoj systému. Použití nástroje není vázáno poplatky.

Díky Java interpretru Pythonu Jython lze Robot Framework úzce propojit s prostředím Javy, není třeba vytvářet marshallingovou[18] vrstvu a je možné z Robot Frameworku přímo spouštět Java třídy.

Komunita vytváří mnoho knihoven, které zpřístupňují běžně používané nástroje akceptačních testů. Základem je integrace na framework umožňující ovládání uživatelského rozhraní (UI automation): Selenium pro webové rozhraní, AutoIt pro GUI v prostředí Windows. Dále poskytuje knihovny pro práci s SSH, FTP nebo napojení na Testlink (systém pro správu testů).

Zápis testů probíhá syntaxí jazyka Python (python-like) klíčových slov obdobně jazyku Gherkin. Framework umožňuje vytvářet vlastní klíčová slova, která snadno zapouzdří danou funkcionalitu. Tím se odlišuje od syntaxe Gherkin, která vynucuje jasnou strukturu testovacího scénáře v duchu BDD. Scénář pro přihlášení do aplikace může vypadat následujícím způsobem:

```
Login Scenario
    [Arguments]  ${username}  ${password}
    Open Login Page
    Fill Username With  ${username}
    Fill Password With  ${password}
    Check Login Success Message
    Check Logged User
```

Poslední stable release[19] byl v květnu 2019, verze 3.1.2. Aktuální latest release[20] je betaverze 3.2 z února 2020.

### Screenster

Cloudové komerční řešení pro správu testovacích scénářů. Lze testovat pouze webové aplikace dostupné online. Lze vyjednat i self-hosted řešení, které umožní testovat aplikace bez nutnosti vystavení do internetu.

[18] Marshalling je proces transformace objektů v operační paměti, tak aby je bylo možné předat mezi různými přímo nekompatibilními částmi programu.

[19] Aktuálně vydaná a podporovaná verze aplikace.

[20] Poslední označená verze, která nemá oficiální podporu, ale je uvolněna veřejnosti např. z důvodu testování.

Aplikace umožňuje nahrávání testovacích scénářů i manuální programování prostřednictvím Selenium frameworku.

Poslední release 1.3 byl v zimě 2017.

### Testim

Jde o cloudové řešení pro správu testovacích scénářů. Scénáře je možné nahrávat z GUI nebo využít dostupný Javascript-like (syntaxí podobný jazyku Javascript) programovací jazyk pro psaní testů. Prostřednictvím tohoto nástroje je možné testovat pouze webové aplikace dostupné z internetu.

Pozitivem tohoto nástroje je snadná spolupráce v týmu.

Informace o aktuální verzi nejsou veřejně k dispozici. Poslední úpravy byly nasazeny během únoru 2020.

### Výběr vhodného nástroje

Z výše popsané rešerše dostupných nástrojů vyplývá, že nejlepším kandidátem pro tvorbu akceptačních testů systému UIS (viz 3) je Robot Framework. Jedním z hlavních důvodů této volby je integrovaná podpora automation frameworků (Selenium – web – aktuálně využívaný v projektu UIS, AutoIt – GUI – možnost rozšíření pro alternativní aplikaci) a dalších modulů.

Dalším důvodem je možnost spuštění v prostředí Javy a snadná integrace Java programů. Je nutné podotknout, že vhodnou alternativou by mohl být také nástroj FitNesse, který též poskytuje integraci Javy. V případě UIS je chybějící vrstva Selenium automation vytvořena již v rámci SupportLibrary (viz 3).

Nevýhodou FitNesse frameworku je ale nutnost spuštění serveru pro spuštění série testů. Tento fakt rozhodl o použití Robot Frameworku. V případě komerčního projektu je vhodné zvážit použití placeného nástroje Ranorex, který poskytuje elegantní mechanismy pro testování webových a .NET GUI aplikací.

# 3 Projekt TbUIS

TbUIS[1] je výzkumný projekt zpracovávaný výzkumnou skupinou ReliSA[2] na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Vedoucím projektu je doc. Ing. Pavel Herout, Ph.D. Základním cílem projektu je poskytnout zkušební prostředí (testbed) pro vývoj nových testovacích metod. Součástí výzkumu je ověřování jednotlivých metod testování a hodnocení jejich praktického přínosu.

Aktuálně se projekt TbUIS skládá z několika základních částí:

- UIS
- ErrorSeeder
- SupportLibrary
- Functional tests
- Acceptance tests

Základní částí TbUIS je smyšlený Univerzitní informační systém (UIS). Jde o webovou aplikaci založenou na technologii Java. Pro vývoj byl použit framework Spring, ORM[3] nástroj Hibernate, cílovou platformou je Tomcat servlet container. Aplikace je návrhem uzpůsobena pro cílené zanášení defektů. Tento proces je realizován prostřednictvím mechanismu dependency injection a je umožněn díky konfiguraci aplikace při spuštění. Více o problematice systému UIS viz [8].

Dalším nástrojem TbUIS je program ErrorSeeder, který slouží pro snadnou správu dostupných možností připravených defect verzí komponent aplikace a přípravu poruchových klonů. ErrorSeeder interně využívá nástroj Maven pro vytváření jednotlivých poruchových verzí. ErrorSeeder vznikl jako diplomová práce [8], možnosti nástroje významně rozšířila diplomová práce [9] vytvořením několika desítek nových typů defektů systému UIS, rozšířením katalogu defektů a kategorizací připravených poruchových klonů.

Základním kamenem pro následné testy je projekt SupportLibrary. V rámci tohoto projektu byl vytvořen nástroj pro automatické ovládání webové aplikace prostřednictvím frameworku Selenium a jeho návrhového vzoru

[1] Testbed University Information System – `https://projects.kiv.zcu.cz/tbuis`
[2] Reliable Software
[3] ORM – objektově relační mapování

PageObjects. Jde tedy o zapouzdření akcí v UIS pro prostředí Java aplikace. SupportLibrary obsahuje též oracle aktuálního stavu aplikace, které odráží stav databáze testované aplikace, respektive částí nezbytných pro provádění zvolených testů. Mechanismus využití testovacího oracle byl také publikován v souvislosti s akceptačním testování IoT[4] (internet of things) řešeních viz [7]. V případě zmíněného článku je testovacím oraclem konečný automat.

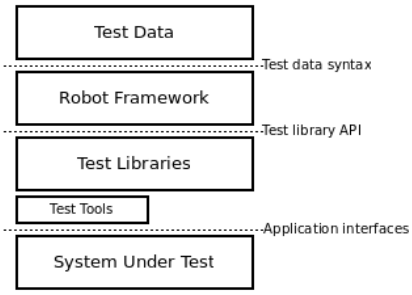
Prvním projektem pokrývajícím testování aplikace je projekt funkcionálních testů, který pokrývá jednotlivé funkce aplikace UIS. S využitím frameworku JUnit5 byly připraveny funkcionální testy pokrývající statické texty (nadpisy, položky menu, ...) aplikace a základní funkcionalitu. Komponentou projektu TbUIS je projekt akceptačních testů (acceptance tests), který je vyvíjen v rámci této práce. Bude sloužit především k ověření „navázané“ funkcionality, spouštění složitějších scénářů.

[4] *Internet věcí* (IoT) je označení pro síť fyzických zařízení, vozidel, domácích spotřebičů a dalších zařízení, která jsou vybavena elektronikou, softwarem, senzory, pohyblivými částmi a síťovou konektivitou, která umožňuje těmto zařízením se propojit a vyměňovat si data – Wikipedia – `https://cs.wikipedia.org/wiki/Internet_v%C4%9Bc%C3%AD` [online] 17.5.2020

# 4 Robot Framework

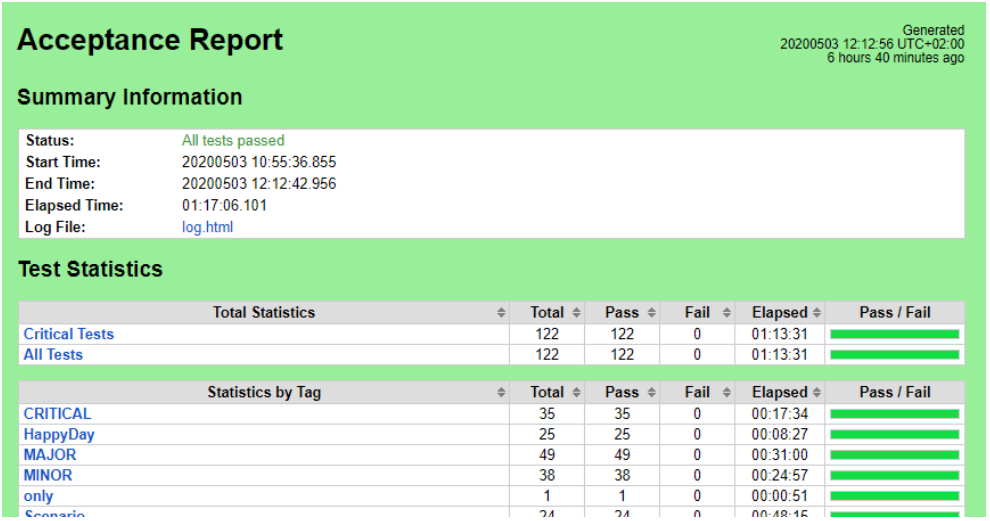
Jak již bylo popsáno, Robot Framework je nástroj pro automatizaci softwarových procesů specializovaný pro testování. Tato kapitola slouží pro získání obecného přehledu o vlastnostech frameworku. Konkrétní popisy použití jsou uvedeny v dokumentaci [12] či připravené příručce (viz přílohy DP).

Dle původního návrhu byl framework určen především pro spouštění automatických testů. V aktuální verzi se Robot Framework považuje za obecný nástroj automatizace procesů. Technicky není rozdíl mezi automatickým spuštěním testu nebo úkolu (task), jde pouze o rozlišnou terminologii. V obou případech dojde k interpretaci jednotlivých robot scriptů prostřednictvím Python interpretru. Framework je navržen ve vrstvené architektuře, která vynucuje oddělení jednotlivých částí programu (viz Obrázek 4.1). Při dodržení pokynů frameworku budou výsledné testy dobře udržovatelné, modulární a datově nezávislé.




Obrázek 4.1: Architektonické vrstvy použití Robot Frameworku. [12]

Velkou výhodou Robot Frameworku v porovnání s jednotkovými testy je úroveň reportingu. Standardním výstupem běhu testů je rozsáhlý záznam jednotlivých operací (viz Obrázek 4.4), včetně trasování jednotlivých parametrů a klíčových slov. Report (viz. Obrázek 4.2) je dělený dle uživatelsky definovaných skupin (mohou být i vnořené), zároveň je k dispozici jednoduchý mechanismus tagů (značek). Každý test nebo klíčové slovo může být označeno libovolným počtem značek. Podle každého tagu lze v test reportech snadno filtrovat. Zároveň lze použít tagy nebo skupiny pro výběr spouštěných testů. Informace o výsledku proběhlých testů je jednoznačně propagována uživateli. V případě selhání některého z testů bude zelená barva pozadí reportu nahrazena červenou.



**Acceptance Report**

Generated
20200503 12:12:56 UTC+02:00
6 hours 40 minutes ago

**Summary Information**

| | |
|---|---|
| Status: | All tests passed |
| Start Time: | 20200503 10:55:36.855 |
| End Time: | 20200503 12:12:42.956 |
| Elapsed Time: | 01:17:06.101 |
| Log File: | log.html |

**Test Statistics**

| Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| Critical Tests | 122 | 122 | 0 | 01:13:31 | |
| All Tests | 122 | 122 | 0 | 01:13:31 | |

| Statistics by Tag | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| CRITICAL | 35 | 35 | 0 | 00:17:34 | |
| HappyDay | 25 | 25 | 0 | 00:08:27 | |
| MAJOR | 49 | 49 | 0 | 00:31:00 | |
| MINOR | 38 | 38 | 0 | 00:24:57 | |
| only | 1 | 1 | 0 | 00:00:51 | |
| Scenario | 24 | 24 | 0 | 00:48:15 | |

Obrázek 4.2: Ukázka reportu úspěšné sady testů.

## 4.1 Klíčová slova

Základním vyjadřovacím prostředkem Robot Frameworku jsou klíčová slova. Klíčová slova můžeme vnímat jako metody, které mohou mít vstupní parametry a návratovou hodnotu.

Klíčová slova jsou podle mechanismu vzniku a použití dělena do dvou kategorií: knihovní (library) a uživatelsky definované (user defined). Z názvu je jasně patrný původ jednotlivých definic. Knihovní klíčová slova jsou definovaná v externích modulech, které je třeba specifickým způsobem inicializovat (klíčové slovo Library). Uživatelsky definovaná klíčová slova jsou ve zdrojových souborech (scriptech), které je možné zanořovat a vytvářet hierarchickou strukturu (klíčové slovo Resource).

Základní knihovny Robot Frameworku (built-in) obsahují více než 300 klíčových slov pro všechny běžné operace testování (podmínky, cykly, práce s řetězci...). Framework, stejně jako programovací jazyk Python, disponuje mechanismy pro práci se seznamy či slovníky.

Při vytváření klíčových slov je doporučeno být co nejvíce konkrétní. Není chybou, že název klíčového slova je věta. Nezapomeňme, že klíčová slova mohou zapouzdřovat složité kroky jednotlivých use cases, nebo dokonce celý use case.

Definice klíčového slova obsahuje několik sekcí, které ovlivňují různé části životního cyklu klíčového slova:

- *[Documentation]* – dokumentace klíčového slova
- *[Tags]* – seznam tagů klíčového slova

- *[Timeout]* – nastavení timeoutu běhu klíčového slova
- *[Teardown]* – klíčové slovo, které se provede po skončení klíčového slova
- *[Arguments]* – definice parametrů volání klíčového slova
- *[Return]* – definice návratové hodnoty klíčového slova

Výsledná definice klíčového slova s návratovou hodnotou může vypadat následovně:

```
Return One Value
    [Arguments]    ${arg}
    Do Something    ${arg}
    ${value} =    Get Some Value
    [Return]    ${value}
```

## 4.2 Test Case

Základní jednotkou testů v prostředí Robot Frameworku je test case. Tyto testy lze sdružovat do skupin. Na první pohled jsou definované sekce totožné s definicí klíčového slova (čtyři části jsou shodné). Test case nemůže pracovat s parametry a návratovou hodnotou a naopak disponuje částí Setup pro přípravu podmínek testu.

- *[Documentation]* – dokumentace testu
- *[Tags]* – seznam tagů klíčového slova
- *[Timeout]* – nastavení timeoutu běhu testu
- *[Teardown]* – klíčové slovo, která se provede po skončení testu
- *[Setup]* – klíčové slovo, které se provede před spuštěním testu
- *[Template]* – klíčové slovo šablony testu

V případě použití konfigurace *[Template]* tělo testu obsahuje pouze data a klíčové slovo uvedené v konfiguraci slouží k definici chování testu. Následující fragment scriptu RF (Obrázek 4.3) demonstruje použití šablony testu. Je definováno klíčové slovo pro logování dvou parametrů a testovací scénář, který využívá toto klíčové slovo jako šablonu. V rámci běhu testovacícho scénáře *TC Template* se dvakrát vykoná klíčové slovo šablony *Log Some Params*. Při prvním volání bude mít parametry *ToLog1* a *ToLog2* a při druhém volání pak *Another1* a *Another2*.

```
*** Keywords ***
Log Some Params
    [Arguments]  ${p1}  ${p2}
    Log  ${p1}
    Log  ${p2}

*** Testcases ***
TC Template
    [Templates]  Log Some Params
    ToLog1 ToLog2
    Another1 Another2
```

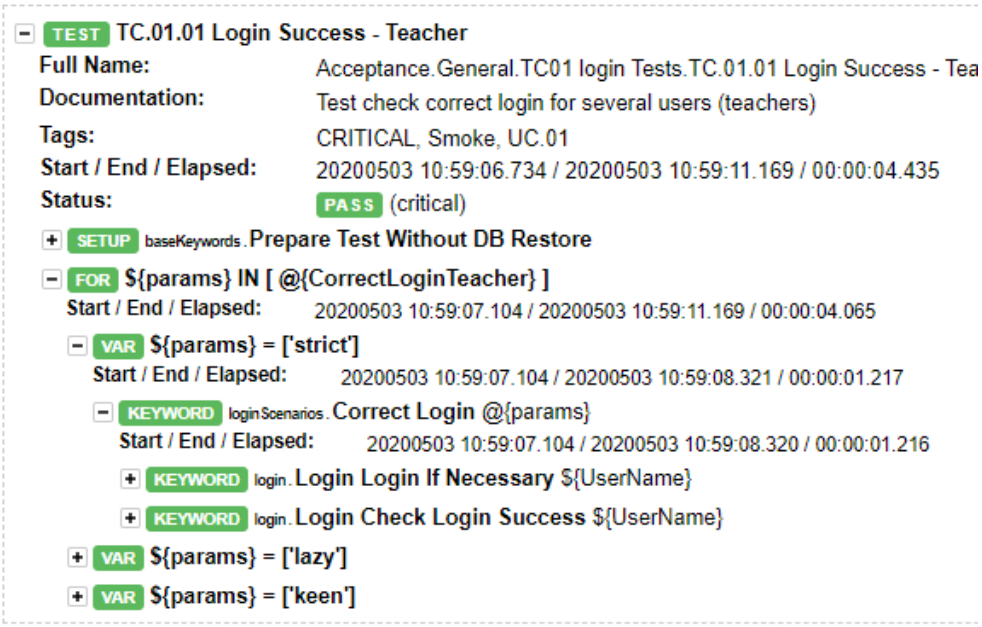
Obrázek 4.3: Ukázka použití šablony testovacího případu.

Je důležité zmínit, že běh test case pracuje s odlišným kontextem než klíčové slovo. V *[Teardown]* sekci testovacího scénáře můžeme používat konstrukce vztažené k výsledku testu (např. Run Keyword If Test Fails). Tyto konstrukce nejdou využít v kontextu keywords. Obdobná klíčová slova jsou definována i pro testovací sadu (test suite). Validní možností využití těchto klíčových slov je zapouzdření do uživatelem připraveného slova pro „úklid testu“, které je vyvoláno prostřednictvím [Teardown] testu. Takové volání už bude mít běhový kontext nastavený správně.

## 4.3 Data driven tests

Robot Framework poskytuje vestavěnou (built-in) podporu data driven testů. Prostřednictvím klíčových slov s argumenty a mechanismu šablonování testu (*[Template]*) je tvůrci testu umožněno v rámci testu pouze načíst data a spustit vybrané klíčové slovo. Tento mechanismus jasně odděluje data testu od jeho logiky. Na Obrázku 4.4 můžeme vidět, že v datové sadě `CorrectLoginTeacher` jsou tři záznamy. Na Obrázku 4.4 jsou vidět podrobné záznamy provedených operaci.

Built-in funkce umožňují načítat data z přiložených souborů (ve formátu YAML). Pokud vývojář testu použije některý z modulů, mohou být data načítána z databáze nebo souborů jiných formátů.




Obrázek 4.4: Ukázka záznamu úspěšného běhu (logu) scénáře přihlášení.

# 5 Zapojení Robot Frameworku

Tato kapitola popisuje způsob přípravy a technické provedení modulu akceptačních testů vytvořeného v rámci této práce. Kapitola obsahuje popis přípravy scénářů akceptačního testování s využitím nástroje Robot Framework (RF). Postupně bude představena zvolená metodika tvorby scénářů a následné technické provedení v prostředí Jython[1].

## 5.1 Postup tvorby scénářů

Scénáře pro jednotlivé kroky akceptačních testů UIS byly vytvářeny na základě use-case (UC) specifikace systému [5]. Výchozím podkladem připravených akceptačních testů jsou tedy ekvivalenty hlavních a vedlejších toků popsaných ve specifikaci. Dále bylo navrženo několik scénářů, které si kladou za cíl detailněji ověřit fungování aplikace. Tyto scénáře sdružují několik operací v jednom testu. Jako příklad může sloužit scénář „Pomalý student“, který simuluje dva studentovy neúspěchy na zkoušce a poslední úspěch. Tyto testy podle scénáře jsou důležité s ohledem na korektní vytváření testovacích dat.

Základní testy dle UC využívají výchozí stav databáze UIS, je tedy možné testovat akce nezávisle na ostatních UC. Jedinou výjimkou jsou nepřímo propojené UC. Jde o situace, kdy v rámci prvního UC je objekt (např. zkouškový termín) vytvořen a v rámci druhého zobrazen. V případě takového propojení mohou testy vykazovat chyby i v případě korektního dokončení prvního UC.

Testy podle scénáře vytvářejí data „od nuly“ postupným průchodem jednotlivých UC (např. pro přihlášení na zkouškový termín musí být tento termín nejprve vytvořen). Tento postup zajišťuje, že test bude mít pro svůj běh připravena správná data (viz kapitola 2.1.3).

### Analýza UC

Každý jednotlivý UC aplikace obsahuje několik standardních částí (*Scope*, *Level*, *Stakeholders*, *Constraints*,...). Pro potřeby analýzy testů jsou důležité

[1] Jde o implementaci Python interpretru v jazyce Java, který poskytuje automatické mechanismy pro volání knihoven v jazyce Java `https://www.jython.org/`.

především sekce hlavních (*Main Success Scenario*) a vedlejších (*Alternative Flows*) toků, předpokladů (*Preconditions*) a faktů (*Postconditions*). Tyto sekce popisují, jakým způsobem se má aplikace používat a jaké mají být její standardní reakce.

Uvažujme UC.16 – *Zapisování nebo oprava zápisu hodnocení zkoušky*[2] aplikace UIS jako příklad. Předpoklady UC jsou popsány následujícím způsobem:

1. učitel je již přihlášený do UIS
2. je zobrazena stránka **Evaluation Table**

Tyto základní předpoklady není třeba v popisu scénáře RF zahrnout, protože jsou již zahrnuty v systému služeb SupportLibrary (SL). Každá služba SL obsahuje mechanismus, který kontroluje právě přihlášeného uživatele a případně provede přihlášení. Dále služby SL obsahují mechanismus otevření žádané URL adresy a její kontrolu.

Script RF tedy popisuje provádění hlavního toku scénáře, uskutečnění hlavní akce UC. Provedení akce je v rámci UC popsáno několika kroky. Zpravidla jde o popis naplnění jednotlivých formulářových polí. Tyto jednotlivé kroky během provádění akce jsou opět svázané se službou SL. Služby SL odstiňují akceptační scénáře od faktického popisu formulářových polí, jejich ovládání či popisu. Scénář hlavního toku UC.16 tedy odpovídá následujícímu fragmentu kódu (řádky uvozené `#` jsou komentáře a řádky uvozené `...` jsou pokračováním předchozího řádku viz [12]):

```
Evaluate Using Table
    [Arguments]  ${Teacher}  ${Student}  ${Subject}  ${Grade}
#Arrange
    ${examDate}=  Dates Prepare Exam Date

#Act
    Teacher Evaluations Add New  ${Teacher}  ${Subject}
    ...  ${Student}  ${Grade}  ${examDate}

#Assert
    Teacher Check Save Evaluation Success Alert
    Teacher Check All Evaluations For Student
    ...   ${Teacher}  ${Student}

    Student Check Completed Subjects and Grades  ${Student}
```

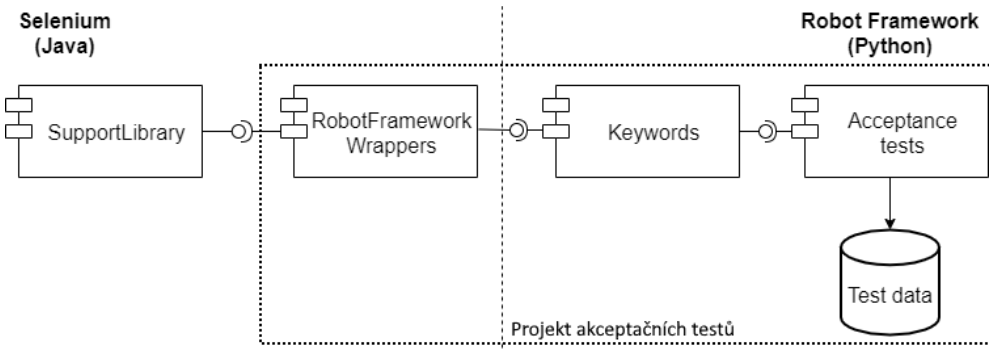
[2] `https://projects.kiv.zcu.cz/tbuis/web/files/uis/uc/cz/html/use-case-16.html`

Zápis scénáře je uvozen třemi slovy `#Arrange`, `#Act` a `#Assert`, jde o dobrý zvyk při psaní testů[3]. Komentář `#Arrange` uvozuje přípravu dat pro test. Testovaná akce je uvedena v sekci `#Act`. Nakonec je provedena kontrola faktů (`#Assert`), tedy reakcí systému. V případě UC.16 se jedná o zobrazení správného hlášení výsledku operace a dále o propsání informace do systému, kdy učitel i student musejí vidět dané změny v systému – zadané hodnocení. Je důležité připomenout, že zároveň s výše uvedeným pozitivním scénářem vzniká i jeho negativní verze. V negativní verzi scénáře se očekává, že hlavní akce selže nebo ji nebude možné provést (např. pro špatně vyplněný formulář).

Dalším vstupem pro tvorbu scénářů je sekce alternativních toků (Alternative flows), která popisuje, jakým způsobem se má systém chovat, pokud nejsou naplněny předpoklady základní akce. V ukázkovém případě UC.16 jde o situace, kdy učitel nevyučuje žádný předmět, neexistují zkouškové termíny apod. K ověření této situace bude použit připravený negativní scénář testu.

Jedním z posledních vstupů pro přípravu scénářů jsou fakta popsané v sekci *Postconditions*. V této sekci je popsán dopad akce na systém, který musí být prověřen. Z UC.16 je vidět, že UIS disponuje staticky[4] definovaným výčtem ohodnocení (A – F), které se vyhodnocuje odlišně. Je třeba ověřit chování a možnost využít jednotlivé hodnoty hodnocení a správné vyhodnocení pravidel.

Výše popsaným způsobem bylo analyzováno všech 22 UC aplikace UIS a na základě těchto mechanismů vznikly scénáře založené na případech užití.




Obrázek 5.1: Diagram komponent projektu akceptačních testů s využitím Robot Frameworku.

[3]V této práci je tento systém použit pouze u složitějších testovacích případů, u jednoduchých nemá praktický význam. Navíc tento systém nelze použít u scénářových testů, kde se jednotlivé bloky překrývají.

[4]Hodnoty výčtu nelze měnit akcemi v rámci chodu UIS.

## 5.2 Souslednost v projektu TbUIS

Pro akceptační testy realizované Robot Frameworkem vznikl v rámci TbUIS nový projekt TbUIS-RobotFramework, který využívá již připravenou knihovnu TbUIS-SupportLibrary (dále jen SupportLibrary nebo SL). Projekt TbUIS-RobotFramework je dělen do tří vrstev (komponent) viz Obrázek 5.1.

```
public static void enrollSubject(Student student, Subject subject) {
  Login_Actions.loginIfNecessary(student);
  Click.openUrlAndWait(Url.URL_STU_OTHERSUBJECTS);
  Stu_OtherSubjects_Page page = new Stu_OtherSubjects_Page();
  String subjectName = subject.getName();
  page.clickOnEnrollButton(subjectName);
  int number = student.getAllEnrolledSubjects().size();
  if (number < Const.STUDENT_MAX_SUBJECTS) {
    if (Stu_OtherSubjects_Check.checkEnrollSuccessAlert() == true) {
      student.enrollSubject(subjectName);
    }
  }
  else {
    Stu_Overview_Check.checkUpdateErrorAlert();
  }
  boolean correct = page.menu.isOtherSubjectsLinkSelected();
  String reason = "Actual menu item 'Other Subjects' is NOT selected";
  Check.checkCorrectness(correct, page.menu.OTHERSUBJECTS_LINK,reason);
}
```

Obrázek 5.2: Ukázka zdrojového kódu operace SupportLibrary.

Nejnižší vrstvou je integrace na SupportLibrary (operace viz Obrázek 5.2). Integrací rozumíme obalení dostupných metod SupportLibrary pomocí kódu (viz Obrázek 5.3), který zpřístupňuje metody do prostředí Robot Frameworku. V rámci nejnižší vrstvy je možné vytvářet i lokální podpůrné knihovny pro služby, které neposkytuje SupportLibrary. Typickým příkladem je generování různých forem datumů či ověřování vstupních dat pro SupportLibrary.

```
public void enrollSubject(String arg0Str,String arg1Str)
                throws AssertionError, UnknownObjectException  {
    Student arg0 = EntityValidator.getStudentByUserName(arg0Str);
    Subject arg1 = EntityValidator.getSubjectByName(arg1Str);

    Stu_OtherSubjects_Actions.enrollSubject(arg0,arg1);
}
```

Obrázek 5.3: Ukázka vygenerovaného kódu metody wrapperu – obalení metody SL `enrollSubject`.

Druhou vrstvu můžeme do určité míry chápat jako pomocnou. Tato vrstva definuje základní klíčová slova jako ekvivalenty SupportLibrary (viz Obrázek 5.4). Vrstva přináší snadnou možnost nahradit integrační vrstvu již na úrovni návrhu struktury akceptačních testů. Pokud vznikne potřeba zaměnit SupportLibrary, modifikuje se právě tato vrstva řídících klíčových slov.

```
Student Enroll Subject
    [Arguments]  ${name}  ${subject}
    Log  Stu_OtherSubjects_Action.Enroll Subject ${name} ${subject}
    Stu_OtherSubjects_Action.Enroll Subject  ${name}  ${subject}
```

Obrázek 5.4: Ukázka druhé vrstvy kódu akceptačních testů – integrace wrapperu SL do klíčového slova RF.

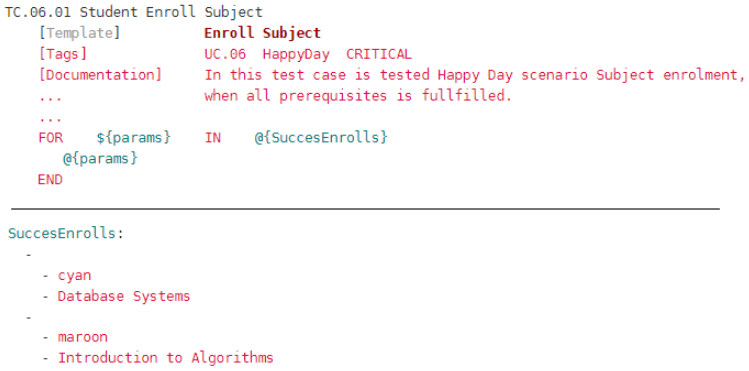
Nad touto vrstvou řídících klíčových slov jsou vytvářeny konkrétní akce v rámci UIS frameworku. Opět jde o definice klíčových slov. Tato připravovaná klíčová slova jsou buď kompozicí řídících klíčových slov nebo připraveným scénářem testu.

```
Enroll Subject
    [Tags]  UC.06
    [Arguments]  ${StudentUserName}  ${SubjectName}

    Student Enroll Subject  ${StudentUserName}  ${SubjectName}
    Student Check Enroll Success Alert
    Student Check Enrolled Subjects  ${StudentUserName}
    Student Check Other Subjects  ${StudentUserName}
```

Obrázek 5.5: Ukázka klíčového slova pro scénář UC.06.

Poslední vrstvou je příprava konkrétních testovacích scénářů. V rámci scénáře je provedeno spojení s šablonou testu a přiřazení odpovídajících datových souborů. V datových souborech jsou připraveny datové řady pro jednotlivé scénáře.




Obrázek 5.6: Ukázka instance TC dle dříve definovaného klíčového slova (scénáře) a datového `.yaml` souboru s datovou řadou.

## 5.3 Integrační komponenta

Nejnižší vrstvou projektu je integrace na SupportLibrary. Tato komponenta obsahuje obalové třídy (dále jen wrappery), které zpřístupňují SupportLibrary z prostředí RF. Vznik těchto wrapperů je podrobně popsán v sekci 5.3.1. Zároveň jsou součástí této komponenty podpůrné nástroje. Prvním je třída poskytující metody pro přípravu datumů ve správném formátu (dle požadavků UIS). Druhým nástrojem je validátor entit. Validátor entit slouží k „překladu“ názvů entit na instance objektů ze SupportLibrary. S existencí validátoru entit se počítá při automatickém generování.

### 5.3.1 Generování kódu

Na nejnižší vrstvě (wrapping) bylo potřeba vytvořit obalové třídy SupportLibrary pro prostředí RF, které provedou základní ověření a přípravu předávaných dat. V rámci RF je snadné pracovat s entitami řetězců nebo celých čísel. Zahrnout do RF ovládání instancí Java objektů se ukázalo jako zbytečně složité a nepotřebné pro vytváření testů. Knihovna SupportLibrary poskytuje rozhraní pro získávání instancí objektů entit (tj. studentů, učitelů, předmětů). Zároveň knihovna na mnoha místech vyžaduje tyto entity jako parametr volání. V rámci testu v prostředí RF není vhodné modifikovat použité instance, modifikace objektů je odpovědností SupportLibrary, která provádí změny v test oracle (viz dříve[5]). Je velmi pravděpodobné, že přímou

[5]Jde o odraz aktuálního systému, na základě kterého se vyhodnocují stavy stránek a jejich obsah (kapitola 3).

modifikací objektů v prostředí RF bychom způsobili nekonzistenci dat v test oraculu a následně chybné výsledky testů.

Z tohoto důvodu byla připravena jednoduchá statická validační třída, která prostřednictvím API SupportLibrary získá požadovanou instanci entity, kterou je možné bezpečně a jednoduše předat do RF. S využitím této validace je mechanismus vytváření obalových tříd SupportLibrary pro RF jednoduchá rutinní činnost. Při vytváření obalů se ukázalo jako efektivní vytvořit generátor, který připraví obaly automaticky.

V rámci automatického generování obalových tříd není potřeba „ruční“ syntaktické či sémantické analýzy zdrojového kódu. Hlavní pomůckou je Java Reflection API, tedy mechanismy pro zjištění programových vlastností[6] Java tříd. Hlavním úkolem generátoru je tedy analyzovat veřejné API tříd a vytvořit obal všech public metod. Přidanou hodnotou analyzátoru (může být označena za bázi znalostí) jsou mechanismy pro získání instancí objektů specifických typů (více viz 5.3.2).

Výhodou automatického generování klíčových slov je minimalizace rizika „copy paste“ chyb a rychlejší adaptace na změny či rozšíření SupportLibrary.

### 5.3.2 Nástroj WrapperGenerator

Pro automatické generování obalových tříd SupportLibrary byla vytvořena utilita WrapperGenerator. Důvodem volby samostatné utility namísto integrace do procesu sestavení a spuštění akceptačních testů byla vyšší kontrola vývojáře nad změnami provedenými generátorem. V případě automatického procesu by mohlo docházet k vytváření chyb přetypování. Z principu mechanismu převodu řetězců na entity, který připravuje WrapperGenerator, nemůže existovat mechanismus, který pozná, že došlo ke změně pořadí typů elementů. Proces přípravy generovaných tříd má tedy dva kroky. Prvním je samotné vygenerování zdrojových souborů wrapperů a druhým je integrace na vrstvu RF. Pro usnadnění manuální práce generátor řadí jednotlivé metody podle abecedy. Zároveň může dojít k situaci, kdy degradací složitých datových typů (`Student`, `Teacher`...) dojde k vygenerování metod s duplicitní signaturou[7]. Úkolem programátora v rámci integrace je rozhodnout, jakým způsobem bude tento fakt propagován do klíčových slov (odstranění, přejmenování metody).

Parametrem spuštění utility je název balíku/ů pro skenování a výstupní složka. V předaných balících, které musí být dostupné prostřednictvím Java

[6]Programovými vlastnostmi chápeme dostupné vlastnosti, metody a jejich parametry.

[7]V rámci aktuální implementace SupportLibrary (1.7.2) jde o jednu operaci pro přihlášení uživatele.

class-path, je provedeno vyhledání metod a následné vygenerování obalových tříd. V terminologii formálních jazyků a překladačů je možné označit algoritmus vygenerování obalových tříd za formu rekurzivního sestupu.

Pro validaci parametrů je klíčová třída `Parameter.java`, která nese definice zpracování a přípravy parametrů volání jednotlivých obalovaných tříd. Je zde připraveno zpracování pro každý jeden očekávaný vstupní typ obalované metody. Kód pro zpracování se liší dle datového typu parametru. K přípravě vstupních parametrů je použita třída `EntityValidator`, která je rozšířením SupportLibrary. Činnost WrapperGeneratoru končí vygenerováním Java zdrojového kódu. K překladu souborů dochází až v rámci kompilace projektu TbUIS-RobotFramework.

Výstupem této utility tedy jsou zdrojové kódy wrapperů pro RF. Tyto zdrojové kódy tvoří část integrační vrstvy, je v nich použit mechanismus validace parametrů z ručně implementované části integrační komponenty.

## 5.4 Vrstva klíčových slov

Jde o vrstvu, která přímo navazuje na vygenerované obalové třídy. Tato vrstva je manuálně vytvořená. V kontextu architektury Robot Frameworku jde o vrstvu na rozhraní test libraries a Robot Framework (viz Obrázek 4.1). Obal „výkonného kódu“ testu, který je umístěný v SupportLibrary, dodržuje jmenné konvence obalované knihovny a zároveň poskytuje agregované balíky pro jednotlivé skupiny operací. SupportLibrary je dělena do skupin podle způsobu autorizace operací. Autorizace může být tří typů: anonymní, student, učitel.

Na této vrstvě neprobíhá vykonávání žádné logiky programu.

Jedna z motivací pro zařazení této vrstvy do návrhu aplikace byla architektonická příprava pro nahrazení SupportLibrary jinou knihovnou, například přímým vyvoláním frameworku Selenium v rámci RF. Takové nahrazení by odpoutalo projekt od závislosti na specifickém prostředí Jython. Za předpokladu, že nahrazující knihovna dodrží stávající kontrakt SupportLibrary, nebude třeba modifikovat stávající testovací scénáře.

Alternativně lze této vrstvy programu využít, pokud existuje více aplikací dle stejné specifikace. Příkladem může být existence GUI a webové aplikace, které zpřístupňují stejné funkce, jen v jiném prostředí a jinak zobrazené. Pro testování je třeba použít různých nástrojů, ale scénáře mohou zůstat stejné.

## 5.5 Testovací scénáře

Tato komponenta odpovídá za přípravu klíčových slov RF pro vykonání jednotlivých operací v UIS během testu (testovacích scénářů). Základem pro tyto scénáře je použití operací z vrstvy klíčových slov (= obal operací SupportLibrary a její rozšíření na integrační vrstvě). Dále jsou některé scénáře doplněny o funkčnosti, které v knihovně nejsou definovány. Doplnění testů je použito především pro přípravu negativních verzí již vytvořených operací a pro různé kompozice operací. Příkladem kompozice operací může být přihlášení na zkoušku a jeho jednotlivé kroky: přihlášení studenta na předmět, výběr termínu, přihlášení na termín. Tato „kompoziční“ klíčová slova jsou použita při vytváření speciálních scénářů. Využití speciálních klíčových slov tedy zaručuje kontrolu stavu UIS před a po provedení akce. Speciální klíčové slovo je připraveno například pro zapsání předmětu studentem viz Obrázek 5.7.

```
Verify Student ${StudentUserName} Enroll Subject ${SubjectName}
    [Tags]  function
    Log  Verify Student ${StudentUserName} Enroll Subject ${SubjectName}
    ...  INFO

    Log  Student name: ${StudentUserName}
    Log  Subject name: ${SubjectName}

    Student Check Other Subjects  ${StudentUserName}
    Student Check Completed Subjects And Grades  ${StudentUserName}

    Student Enroll Subject  ${StudentUserName}  ${SubjectName}
    Student Check Enroll Success Alert

    Student Check Enrolled Subjects  ${StudentUserName}
    Student Check Other Subjects  ${StudentUserName}
    Student Check Completed Subjects And Grades  ${StudentUserName}
```

Obrázek 5.7: Ukázka speciálního klíčového slova. Stav UIS je kontrolován před i po provedení požadované operace (zapsání předmětu).

Je nutné připomenout, že výkonný kód scénářů testu je připraven v rámci klíčových slov. V tomto případě není rozdíl mezi klíčovými slovy pro scénářové testy nebo testy dle UC. Z pohledu RF nejde o test case, ale o klíčové slovo. Tato příprava je provedena z důvodu využití mechanismu šablonování testu. Odpovědností instancí testů pak je načíst data a spustit test (klíčové slovo) dle připravené množiny. Připravený scénář je znázorněný na Obrázku 5.8.

```
Teacher Cancels Exam Date Participants
    [Arguments]  ${TeacherUserName}  ${SubjectName}  @{Participants}
    [Tags]  UC.12
    Teacher Cancels Exam Date  ${TeacherUserName}  ${subjectname}
    FOR  ${participant}  IN  @{Participants}
        Student Check Other Exam Dates  ${participant}
        Student Check Exam Dates  ${participant}
    END
```

Obrázek 5.8: Ukázka scénáře pro kontrolu chování zrušení zkouškového termínu s přihlášenými studenty. Parametry testu je identifikace zkouškového termínu a seznam studentů, u kterých má být provedena kontrola seznamu zkouškových termínů.

Tento scénář (keyword) je použit jako šablona testovacího případu (test case), který načítá data pro běh testu. Data pro jednotlivé testy jsou připravena a načtena z přiložených datových souborů. Výhodou této implementace je přidávání nových testovacích variant bez nutnosti opakovat zdrojový kód scénáře či modifikovat soubor scénáře. Toto rozložení přispívá ke kontrole vývoje testovacích scénářů.

Vraťme se nyní do situace, kdy jsou společně se zákazníkem připravovány testovací scénáře. Připravit scénář testu s pomocí prostředků RF může být technicky složitější. Zákazník sice snadno chápe, co které klíčové slovo znamená, ale nemusí si být vědom všech technických konsekvencí. Tento postup je třeba podrobit vyšší úrovni kontroly než přípravu datových variací testu. Fyzické oddělení tedy umožňuje snazší zapojení kontrolních mechanismů. Programátor připravuje scénáře dle zadání od zákazníka. Zákazník má v plné režii datové řady testů.

### 5.5.1 Připravené speciální scénáře

Základní množinou testovacích scénářů jsou hlavní toky popsané ve specifikaci aplikace a základní alternativní toky, viz Tabulka 7.1. Pro ověření složitějších omezení, například více učitelů na jednom předmětu apod. jsou připraveny speciální scénáře. Většina scénářů se zaměřuje na ověření chování aplikace v případě, že v databázi nejsou připravena žádná data. V rámci připravených scénářů jsou připraveny tři sady TC, které sdružují logicky související scénáře. Jde o scénáře kooperace dvou učitelů, operace učitele v rámci zkouškového termínu a operace studenta v rámci zkouškového termínu. Zbylé scénáře jsou umístěné v samostatných souborech, případně umístěné v sadě se svou smoke variantou. Smoke varianta testu využívá pouze první položku

datové řady.

### Pomalý student

Scénář simuluje situaci, kdy student skládá zkoušky z předmětu na několik pokusů. V našem případě jde o dva pokusy (omezení stanovené UIS), během kterých je student hodnocen „nevyhověl“. Až poslední studentův pokus je hodnocen jako „vyhověl“ Scénář především ověřuje možnost zapsat si další zkouškový termín po případném nesložení zkoušky.

### Plná zkouška

Scénář simuluje situaci, kdy učitel vypíše termín a postupně se naplňuje jeho kapacita, až není další volné místo. Zde je vhodné připomenout, že pro základní scénáře jsou data připravena v rámci interního nastavení UIS databáze, kdežto nyní jsou vytvářena postupně. Cílem scénáře tedy je ověřit korektní zápis studentů na zkouškové termíny a zablokování přihlášení po dosažení jeho kapacity.

### Dva učitelé

Zde se jedná o sadu více scénářů. Scénáře simulují situaci, kdy dva vyučující učí jeden předmět. Oba učitelé vypisují všechny zkouškové termíny ve stejný čas. Cílem scénářů je ověřit, že systém dovolí paralelně vypsat termíny (tři, dle omezení UIS) oběma vyučujícím, na které se studenti mohou korektně přihlásit.

První varianta kontroluje chování bez účasti studentů, druhá pak zavádí postupné přihlašování studentů na jednotlivé termíny a ověřuje možnost přihlásit se na zkoušku k různým vyučujícím. Třetí pak kontroluje, že učitel nemůže zrušit zkouškový termín kolegovi.

### Hodnotím za kolegu

Scénář simuluje situaci, kdy se pokouší jeden vyučující zadat do systému hodnocení ze zkouškového termínu kolegy. Dle specifikace UIS toto chování není přípustné. V rámci scénáře je třeba ověřit, že programátor kontroluje vazbu vyučující – zkouškový termín (ne jen vyučující – předmět). Tento scénář má dvě varianty dle typu zadání hodnocení (tabulka/formulář).

Obrázek 5.9 je ukázáno použití kompozičních klíčových slov pro tvorbu scénáře. Jak bylo již dříve popsáno, kompoziční klíčová slova seskupují více kontrolních operací do jednoho klíčového slova.

```
Colleague Grades Form
    [Documentation]  Keyword which simulates when one teacher tries to grade anothers exam date.
    [Tags]           ScenarioKW
    [Arguments]      ${Teacher1UserName}  ${Teacher2UserName}  ${SubjectName}  ${StudentName}

    Prepare Test with DB restore

    Verify Teacher ${Teacher1UserName} Participate In Subject ${SubjectName}
    Verify Teacher ${Teacher2UserName} Participate In Subject ${SubjectName}

    Verify Student ${StudentName} Enroll Subject ${SubjectName}

    ${examDate} =  Dates Prepare Exam Date

    Verify Teacher ${Teacher1UserName} Create Exam For Subject ${SubjectName} On ${examDate} For 1

    Run Keyword And Expect Error  *
    ...  Teacher Set Evaluation  ${Teacher2UserName}  ${SubjectName}  ${StudentName}  A   ${examDate}
```

Obrázek 5.9: Ukázka použití kompozičních klíčových slov v rámci scénáře.

### Hodnotím v minulosti

Tento scénář ověřuje správné uzamčení hodnocení zkouškového termínu, pokud byl student ohodnocen v rámci dalšího termínu.

### Nerozhodný učitel

Tento scénář ověřuje správné chování UIS v případě, že učitel mění hodnocení předmětu. Předmětem kontroly je propsání změn hodnocení termínu a možnost jejich změny.

### Hodnocený termín

Tento scénář ověřuje správné chování UIS v případě, že se učitel snaží zrušit termín, v rámci kterého byl již hodnocen některý ze zúčastněných studentů. Předmětem kontroly je zablokování možnosti smazat hodnocení termínu.

### Zkouškové termíny – učitel

Tato skupina scénářů obsahuje dva odlišné scénáře. Simuluje pokus o vytvoření čtvrtého zkouškového termínu učitelem (pozitivní a negativní verzi). Dle omezení UIS je možné vytvořit pouze tři zkušební termíny. Druhým scénářem je ověření korektního odstraňování zkouškových termínů studentům (blokace přihlášení, zobrazení, . . . ), po jejich zrušení učitelem.

### Zkouškové termíny – student

Scénáře simulují pokusy o přihlášení studenta na zkouškový termín. Postupně jsou ověřeny omezení čtvrtého zkouškového termínu, nemožnost při-

hlášení na čtvrtou zkoušku a korektní chování v případě vícekrát opakované změny přihlášení na zkouškový termín (jiný datum a čas termínu).

#### Zápis předmětů

Scénář simuluje zápis studenta na předměty a ověření, že je v pořádku „zápis od nuly“. Scénář je vytvořen ze stejného důvodu jako Plná zkouška. Cílem ověření je korektní zvyšování počtu předmětů (bez ovlivnění manuálním insertem dat).

#### Příprava semestru

Další z řady scénářů pro ověření zadání dat od nuly. Tentokrát je ověřován počet předmětů vyučovaných jedním vyučujícím.

#### Kapacita předmětu

Další z řady scénářů pro ověření zadání dat od nuly. Tentokrát je ověřován počet studentů přihlášených na jednom předmětu. Toto omezení UIS neobsahuje, proto by se všichni studenti měli úspěšně přihlásit.

#### Speciální zadání známky

Tento scénář ověřuje korektní fungování „zkratky“ v zadání hodnocení zkouškového termínu. Scénář využívá tlačítka *Grade* na seznamu studentů zkouškového termínu, stisk tohoto tlačítka předvyplní formulář pro zadání známky na jiné stránce.

## 5.6 Rozšíření služeb SupportLibrary

Během analýzy jednotlivých případů užití aplikace bylo odhaleno, že služby SupportLibrary neobsahovaly všechny potřebné funkce. Pro potřeby akceptačních testů jsem SL rozšířil o následující operace:

- operace načtení vlastního datového souboru, včetně přenastavení interního oracle
- operace pro kontrolu seznamu účastníků zkoušky – učitel
- operace pro klik na tlačítko „*Grade*“ v seznamu účastníků zkoušky – učitel
- operace pro dokončení procesu oznámkování při použití tlačítka „*Grade*“

## 5.7 Projekt a spuštění

Projekt TbUIS-RobotFramework je sdílený Java/Robot Framework projekt, který je spravován nástrojem Maven. V sekci zdrojových kódů je umístěna část SupportLibrary wrapperů a v sekci testů je umístěn zdrojový kód RF scriptů.

Z důvodů závislosti na interpretru Jython, který není kompatibilní se současně užívanou LTS[8] verzí Javy (11), je pro překlad a spouštění potřeba využít Javu ve verzi 8. Pro snadné spuštění testů je použit doplněk (plugin) nástroje Maven: `robotframework-maven-plugin`. Tento nástroj využívá knihovnu `org.robotframework.robotframework`, která zapouzdřuje Jython a verzi Robot Frameworku pro Javu.

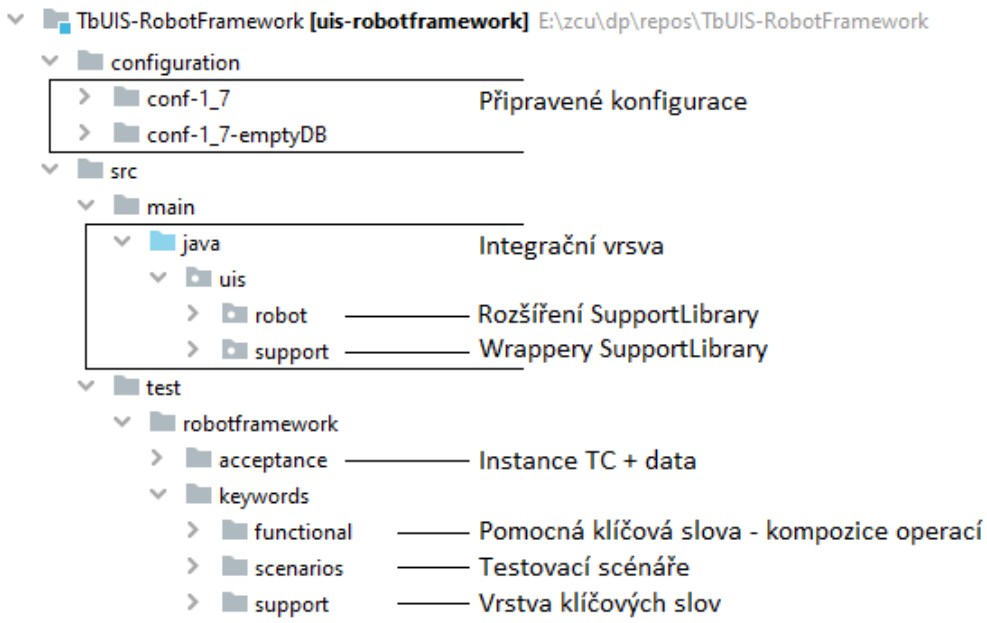
Využití `robotframework-maven-plugin` doplňku vynucuje umístit testy aplikace do Maven sekce pro zdrojové kódy testů (`src/test`) při výchozí konfiguraci RF Maven pluginu do podsložek `robotframework/acceptance`. Projekt tedy porušuje metodiku nástroje Maven, kdy by v sekci pro testy projektu měly být umístěny jen testy daného projektu, nicméně tento drobný prohřešek přináší několik výhod. První výhodou je automatické stažení knihoven (Robot Framework, Jython). Další výhodou je dodržení pořadí sestavení projektu. Před spuštěním testů (prostřednictvím cíle Maven životního cyklu verify) je v případě chybějící obalové knihovny tato automaticky vytvořena (překlad wrapperů).

### 5.7.1 Struktura projektu

Jak již bylo zmíněno, zdrojový kód projektu je strukturován dle metodiky Maven projektů. Ve složce `src/main/java` tedy najdeme definice balíku `uis.support`, kde jsou umístěny wrappery pro SupportLibrary a balík `uis.robot`, který obsahuje testovací klíčové slovo pro ověření integrace a doplňkové nástroje pro testování. Jde o třídu `DateUtils`, která poskytuje doplňkovou funkcionalitu pro získání datumů, a třídu `EntityValidator`, která poskytuje „překlad“ jmen na instance objektů (viz kapitola 5.3.2).

Scripty pro Robot Framework jsou umístěny ve složce `src/test/robotframework`. Zde jsou scripty rozděleny na dvě části, `acceptance` a `keywords`. V části `acceptance` jsou definice jednotlivých testovacích scénářů a odpovídajících datových řad. V části pro klíčová slova jsou připraveny šablony operací (`scenarios`) a integrace na SupportLibrary `support`. Pro usnadnění integrace SupportLibrary jsou připravené sjednocující soubory

[8]Ke dni 15.5.2020 je nejvyšší verzí javy 14 vydaná v březnu 2020, za long time support (LTS) verzi je považovaná 11 (září 2018).




Obrázek 5.10: Struktura projektu TbUIS-RobotFramework.

klíčových slov, které usnadňují vkládání jednotlivých operací do scénářů (`studentKeywords`/`teacherKeywords`).

Dále je v kořenu projektu umístěna složka `configuration`, která obsahuje složky jednotlivých konfigurací SupportLibrary s připraveným obsahem databáze, případně jinou změnou konfigurace UIS. Projekt TbUIS-RobotFramework byl vytvořen pro testování verze UIS 1.7, proto jsou konfigurace jednotlivých prostředí označené `-1_7`. Zároveň byl ve verzi 1.7 knihovny SupportLibrary připraven mechanismus načítání změny konfigurace z proměnných prostředí a operace pro import vlastního datového souboru, což umožnilo elegantní přepínání konfigurací testovacího prostředí. Za výchozí konfiguraci je považována `conf-1_7`, v případě výchozí konfigurace se obsah databáze resetuje standardním způsobem UIS – Restore Database. V případě jakékoli jiné než výchozí konfigurace je třeba do složky s konfigurací navíc umístit soubor `databaseContent.json`, který obsahuje data ve formátu JSON pro import do UIS (viz [8] a [9]). Pro reset aplikace se používají mechanismy UIS ImportDB. Datový soubor musí logickým obsahem odpovídat konfiguraci obsahu SupportLibrary, jinak budou testy poskytovat falešné výsledky.

### První spuštění

Prvním krokem je získání knihovny SupportLibrary. V aktuálním stavu je třeba tuto závislost připravit manuálně prostřednictvím lokální instalace

nástrojem Maven. Jde o kroky stažení zdrojových kódů, úprava projektu a instalace (`mvn install`). Významně se zde projevuje závislost na JDK8. SupportLibrary není připravena jako multi-target (spustitelná více verzemi JRE), je tedy třeba manuálně připravit artefakt `<verzeSL>-java8` (úpravou souboru `pom.xml`). Instalace musí proběhnout prostřednictvím nástroje Maven konfigurovaného na JDK8[9]. Pro SupportLibrary verze 1.7.2, budou upraveny následující části `pom.xml`:

```
<project>
    <version>1.7.2-java8</version>
    <properties>
        <java.lang.version>8</java.lang.version>
    </properties>
</project>
```

Tato závislost je připravena ve složce LIBS na přiloženém DVD a lze provést přímou instalaci bez nutnosti stahovat zdrojový kód SupportLibrary.

Druhým krokem je příprava správné konfigurace SupportLibrary. Tyto konfigurace se dle potřeby umisťují do zvláštních složek ve složce `configuration`. V této nové složce se očekávají následující soubory:

- `configuration.txt` – základní konfigurační soubor Support Library
- `students_list.txt` – seznam studentů a studovaných předmětů
- `subjects_list.txt` – seznam předmětů s kreditovým ohodnocením
- `teachers_list.txt` – seznam učitelů a vyučovaných předmětů
- `users_list.txt` – seznam uživatelů systému

V rámci akceptačních testů jsou připraveny konfigurace pro prostředí UIS verze 1.7 se základní databází a prázdnou databází.

Třetím krokem je spuštění instance UIS. Tento krok není nezbytný. Je možné využít instancí UIS spuštěných na serverech katedry (jeden poruchový a jeden bezporuchový klon). Nicméně doporučuji spuštění vlastní instance (buď lokálně a nebo na vlastním serveru). Na veřejných serverech by mohlo dojít k souběhu různých pokusů návštěvníků. Vlastní instalace také poskytne výrazně lepší odezvu aplikace. Vybranou instanci UIS je třeba uvést v konfiguraci jednotlivých prostředí (jako URL domovské stránky).

[9]Lze ověřit příkazem `mvn -version`.

Nyní je možné provést instalaci projektu akceptačních testů (jde o překlad wrapperů) prostřednictvím nástroje Maven (`mvn install`). Během instalace se spustí testy označené tagem `smoke`. Smoke testy mají za úkol ověřit integraci RF-Java (TC Test Basic Java Keyword) a RF-SupportLibrary. V případě, že smoke testy nejsou úspěšné, nedojde k instalaci kompilovaných obalových tříd a spuštění vybraných testů Maven pluginem `robotframework-maven-plugin` nebude možné.

### 5.7.2 Příprava nového scénáře

Hlavní motivace použití Robot Frameworku (specializovaného nástroje pro AT) bylo připravit prostředí pro rychlý vývoj scénářů akcí prováděných v aplikaci. Uvažujme tři možné situace: změna omezení aplikace (změna konfigurace), nový scénář (nová posloupnost akcí) a nová operace (zcela nová funkcionalita UIS).

Uvažujeme-li změnu konfigurace aplikace (změna počtu zkušebních termínů, maximální počet učitelů na předmět, ...) modifikace sady AT se dotkne pouze datových souborů. Jednotlivé testovací případy jsou strukturované dle specifikace. Pokud dojde ke změně v UC.03 – *Změna uživatelských personálních údajů*, bude se změna týkat především datových souborů svázaných s TC.03. Nakonec je třeba provést analýzu, zda změna konfigurace ovlivňuje scénářové varianty testů. Například snížení maximálního počtu učitelů na předmětu by pro scénářové testy bylo fatální – testují chování systém při spolupráci dvou učitelů. Stejný způsob úpravy se použije ve chvíli, kdy používání aplikace identifikuje datovou sadu, která způsobuje chybu[10]. Tester identifikuje scénář a datovou řadu doplní o potřebná data.

Druhým případem je vytvoření nové posloupnosti operací v UIS (nového scénáře). Opět je tato operace iniciována nalezením chyby během standardního používání aplikace. Nalezená chyba je způsobena specifickou posloupností operací, která není součástí již existujícího scénáře. Nejprve je potřeba určit, zda jde o kompletní nový scénář (prázdná databáze) nebo doplnění UC o specifickou situaci (výchozí stav databáze UIS). V představení procesu nyní pokračujeme složitější cestou – nový scénář. Programátor musí vytvořit tři nové soubory (v nejhorším případě). První soubor bude obsahovat datové řady testu. Druhý soubor bude obsahovat instance testu. Instancí testu rozumíme deklaraci testovacího případu, který určuje použitou šablonu, datovou řadu a název TC. Třetí soubor bude obsahovat popis scénáře (konkrétní

[10]Uvažujme akademický případ. Názvy předmětů jsou v paměti uloženy za sebou. Programátor neošetřil přetečení pole. Příliš dlouhý název předmětu tak přepíše následující text.

posloupnost akcí).

Při vytváření scénářů je vhodné vzít do úvahy souvislost s již existujícími skupinami a sjednocovat tak vykonávání testů do sad souvisejících scénářů (např. scénáře spolupráce dvou učitelů). Pokud existuje scénář se shodným výsledkem pro různá data, je vhodné vytvořit více instancí testu pro datové řady obsahující jiný typ chyby. Uvažujme UC.14 – *Vypsání nového termínu zkoušky*. V rámci tohoto případu užití se vyplňuje formulář různými daty (datum, počet účastníků, volba předmětu). Vyplnění formuláře tedy může selhat minimálně ve třech různých situacích (špatné datum, číslo, ...). Tuto skutečnost je žádoucí reflektovat v instancích testů, kdy pro jeden scénář („vyplním formulář a dojde k chybě“) je více instancí testů a datových řad. V případě UC.14 jsou výsledné TC následující:

- `TC.14.01 Teacher Create Exam`
- `TC.14.02 Teacher Create Exam - Custom Date`
- `TC.14.03 Teacher Create Exam - Custom Participants Number`
- `TC.14.04 Teacher Create Exam Failed - Not Teaching Subject`
- `TC.14.05 Teacher Create Exam Failed - Wrong Date`
- `TC.14.06 Teacher Create Exam Failed - Number Format Out Of Range`
- `TC.14.07 Teacher Create Exam Failed - Wrong Number Format`
- `TC.14.08 Teacher Create Exam Failed - Date In History`
- `TC.14.09 Teacher Create Exam Failed - Wrong Date Order`

Připravené TC.14.04 – 09 odpovídají možným chybám v zadání dat, která systém kontroluje a jejich výsledkem je neprovedení akce vytvoření zkoušky. Těchto šest negativních scénářů ověřuje restrikce systému. Tyto scénáře interně využívají stejný scénář – `Create Exam Term Should Fail`. Při dodržení tohoto dělení je jasně vidět, kterou nepřípustnou možnost systém umožnil. Zároveň zde zůstává jistá variabilita pro specifická data (např -1 i 11 jsou čísla mimo rozsah povolený UIS) a není třeba vytvářet speciální TC pro záporné hodnoty apod.

Posledním, nejnáročnějším případem je rozšíření o nové operace (viz 5.6). Důvodem zahájení tohoto procesu může být rozšíření funkcionality UIS (je třeba testovat nové akce). V takovém případě programátor musí odpovídajícím způsobem rozšířit knihovnu SupportLibrary, použít nástroj *Wrapper-Generator* pro aktualizaci obalových tříd a doplnit implementaci klíčových

slov pro nově „obalené“ funkce SL. Následně programátor musí postupovat stejně jako v případě vytváření nového scénáře.

### 5.7.3 Neočekávané překážky

Během vývoje sady testů bylo odhaleno několik neočekávaných problémů, které bylo třeba vyřešit. Prvním zjištěným problémem byly nahodilé chyby ve strojovém ovládání prohlížečů *Chrome* a *Firefox* pomocí dostupných webdriverů[11]. Pravděpodobně jde o chybu třetí strany, která je způsobena velmi rychlým vývojem jednotlivých prohlížečů. Naštěstí se tyto nahodilé chyby při použití prohlížeče *Opera* neobjevují. Z tohoto důvodu byly experimenty realizovány v prostředí prohlížeče *Opera*.

Druhým významným problémem bylo „zahazování“ chybových zpráv. V rámci integrace Robot Frameworku a Javy docházelo k „zahození“ výjimek typu `AssertionError`. V interpretu Jython došlo k zahození informace o chybě a testy aplikace byly falešně negativní – testy ukazují, že aplikace je bez chyby. Odstranění tohoto problému naštěstí nebylo složité. Technicky mechanismy Jython propagují výjimky korektně. Tato informace byla známá již z PoC[12] při návrhu aplikace. Zahození výjimek tedy bylo odstraněno doplněním možnosti vyhození `AssertionError` do deklarací všech metod obalových tříd. Tato změna byla zahrnuta do utility *WrapperGenerator* a bylo provedeno přegenerování obalových tříd. Tato zdánlivě bezvýznamná úprava zajistila korektní propagaci chyb do prostředí RF = odhalení žádaných chyb.

[11] Webdriver – komponenta třetí strany, která umožňuje strojové ovládání webového prohlížeče.

[12] Proof of Concept – ucelená funkční část aplikace, která dokazuje použitelnost navrženého řešení.

# 6 Test-bed Experiment

V této kapitole budou představeny a diskutovány výsledky použití vytvořené sady akceptačních testů. Experiment byl proveden s použitím 28 standardních poruchových klonů UIS[1] a jedním bezporuchovým klonem. Každý klon UIS byl otestován připravenou sadou. V názvu každého poruchového klonu (např. *28-C2.H2.M1.L0_M_CR*) je zakódována klasifikace chyb, které daný poruchový klon obsahuje, a jejich počet (C=critical, H=high, M=medium, L=low).

## 6.1 Popis experimentu

Průběh experimentu byl řízen scriptem a probíhal zcela automatizovaně (podrobně viz příloha B). Úvodní kroky scriptu připravily prostředí. Díky použití nástroje Git pro správu verzí bylo ověřeno, že je pro experiment použita žádaná verze aplikace `git checkout`. Nástroj Maven byl použit pro odstranění dočasných souborů sestavení (`mvn clean`). Řídící script experimentu je uložen ve složce runner na přiloženém DVD.

Po přípravě prostředí byly v cyklu řízeném seznamem poruchových verzí (viz projekt TbUIS[2]) prováděny následující činnosti:

- příprava poruchového klonu dle konfigurace, tj. vygenerování `.war`,
- nasazení `.war` na server (localhost),
- odstranění dočasných souborů akceptačních testů,
- spuštění sady akceptačních testů,
- uložení výsledků.

Výstupem řídícího scriptu tedy není jen soubor výsledků sady akceptačních testů, ale také výstup jednotlivých kroků přípravy (výstup programu Maven), aby bylo možné ověřit, zda nedošlo k chybě při přípravě experimentu.

[1] Konkrétní informace o jednotlivých poruchových klonech jsou uvedeny na webových stránkách projektu `https://projects.kiv.zcu.cz/tbuis`

[2] Viz kapitola 3 nebo webové stránky projektu: `https://projects.kiv.zcu.cz/tbuis/web/page/download`

## 6.2 Sada akceptačních testů

Vytvořená sada testů obsahuje 122 testovacích případů (test cases) UIS. Z celkového počtu je 24 TC zaměřeno na speciální případy užití UIS (scénáře) a 98 ověřuje základní chování dle UC specifikace. Každý scénář je spuštěn několikrát s různými datovými sadami. V rámci testování dojde až k 6008 porovnání očekávaných stavů objektů UIS. V případě nalezení chyby se počet porovnání snižuje, protože připravené scénáře nedosáhnou svého plánovaného konce.

Otestovat kompletní aplikaci trvá průměrně 1 h 18 min. Nejdelší experiment trval 1 h 35 min a nejkratší 26 min. Délka provádění jedné sady akceptačních testů závisí především na typu defektu, který obsahuje testovaný klon (čím dříve scénář selže, tím rychleji je testování sady dokončeno). Experimenty byly prováděny na lokální stanici. Čas provádění tedy není ovlivněn zpožděním síťové komunikace.

Lokální stanice pro experiment měla následující konfiguraci: Operační systém Windows 10, 8GB RAM, procesor Intel Core i7-5500U, SSD disk. Pro databázi byla použita lokální instance MySQL databáze 10.1.37-MariaDB. Jako web server byl použit Tomcat server verze 9.0.24 využívající interpreter Java 11 (11.0.2). Akceptační testy byly spuštěné Java interpretrem verze 8 (1.8.0_251) prostřednictvím Maven balíku `robotframework-maven-plugin` verze 1.5.1.

## 6.3 Odhalené chyby UIS – vývoj

V rámci vývoje sady akceptačních testů bylo odhaleno a opraveno několik chyb v aplikaci UIS. Následující dvě chyby byly nalezeny díky speciálně připravovaným scénářovým testům. Fakt, že tyto chyby byly odhaleny až nyní[3] v praxi dokazuje existenci „pesticidového paradoxu“. „Pesticidový paradox“ je situace, kdy sada testů neodhaluje určité chyby, ale aplikace je obsahuje. Z tohoto důvodu je třeba testovací sady neustále vyvíjet a připravovat nové typy testů (např. scénářové).

1) Student měl neomezený počet pokusů na složení zkoušky. Omezení deklarované ve specifikaci na počet zkouškových termínů ovlivňovalo pouze učitele, respektive počet studentových pokusů u jednoho učitele. Tuto chybu odhalil scénář maximálního počtu studentových pokusů.

2) Bylo možné smazat zkušební termín, kde již bylo uděleno hodnocení studenta, učitel tak ztratil možnost editovat udělenou známku. Toto cho-

[3] V rámci pátého realizovaného projektu TbUIS a tři roky od vytvoření aplikace.

vání nebylo popsáno ve specifikaci a k jeho objevení vedla „náhoda“ při vytváření možných variant zrušení zkouškového termínu (bez studentů, se studenty, s uděleným hodnocením). Popis správného chování byl doplněn do specifikace UIS a zároveň bylo upraveno chování aplikace UIS.

Celkem bylo v rámci této práce odbaveno 28 ticketů, ve kterých byly reportované jednotlivé chyby odhalené ostatními uživateli nebo návrhy na vylepšení UIS (projekty funkcionálních testů a zátěžových testů). Opravy a vylepšení UIS jsou řízeny prostřednictvím nástroje Gitlab nasazeném na jednom ze serverů katedry[4]. Nejrozsáhlejší opravy se týkaly těchto částí UIS:

- řazení všech seznamů (studentů, učitelů),
- vylepšení aktivity tlačítek (UX úpravy aplikace),
- opravy textací.

[4]https://gitlab.kiv.zcu.cz/herout/TbUIS-UIS/issues

## 6.4 Výsledky experimentu

Následující tabulka shrnuje výsledky experimentu dle testovaných poruchových klonů. Úspěšné testovací případy (test case – TC) jsou označovány *Pass*, neúspěšné TC *Fail*:

| Poruchový klon | Trvání testu [*h:m*] | Pass | Fail |
|---|---|---|---|
| 00-C0.H0.M0.L0_ALL_OK | 01:17 | 122 | 0 |
| 01-C0.H0.M0.L1_S_S_07 | 01:18 | 122 | 0 |
| 02-C0.H0.M0.L1_S_S_03 | 01:16 | 122 | 0 |
| 03-C0.H0.M0.L1_S_S_02 | 01:07 | 114 | 8 |
| 04-C0.H0.M0.L1_S_S_04 | 01:18 | 120 | 2 |
| 05-C0.H0.M0.L1_S_S_09 | 01:18 | 121 | 1 |
| 06-C0.H0.M0.L1_T_S_05 | 00:42 | 96 | 26 |
| 07-C0.H0.M1.L0_S_S_08 | 01:25 | 122 | 0 |
| 08-C0.H0.M1.L0_S_S_12 | 01:19 | 122 | 0 |
| 09-C0.H0.M1.L0_T_S_10 | 01:18 | 121 | 1 |
| 10-C0.H0.M1.L0_T_S_11 | 01:18 | 121 | 1 |
| 11-C0.H0.M1.L0_S_S_05 | 01:18 | 121 | 1 |
| 12-C0.H0.M1.L0_T_S_08 | 01:13 | 108 | 14 |
| 13-C0.H1.M0.L0_G_T_D_01 | 01:04 | 94 | 28 |
| 14-C0.H1.M0.L0_D_U_01 | 00:50 | 101 | 21 |
| 15-C0.H1.M0.L0_D_U_02 | 00:50 | 101 | 21 |
| 16-C0.H1.M0.L0_T_S_07 | 01:18 | 119 | 3 |
| 17-C0.H1.M0.L0_T_S_09 | 01:19 | 121 | 1 |
| 18-C0.H1.M0.L0_S_S_06 | 01:23 | 122 | 0 |
| 19-C0.H1.M0.L0_S_S_10 | 01:23 | 120 | 2 |
| 20-C0.H1.M0.L0_T_S_02 | 01:35 | 117 | 5 |
| 21-C0.H1.M0.L0_T_S_04 | 01:28 | 116 | 6 |
| 22-C1.H0.M0.L0_S_S_11 | 00:57 | 106 | 16 |
| 23-C1.H0.M0.L0_T_S_03 | 01:26 | 118 | 4 |
| 24-C1.H0.M0.L0_T_S_06 | 00:42 | 98 | 24 |
| 25-C1.H0.M0.L0_S_S_01 | 00:30 | 84 | 38 |
| 26-C1.H0.M0.L0_T_S_01 | 00:32 | 63 | 59 |
| 27-C1.H0.M0.L0_U_D_01 | 01:19 | 118 | 4 |
| 28-C2.H2.M1.L0_M_CR | 00:26 | 36 | 86 |

Tabulka 6.1: Výsledky experimentu.

Akceptační testy ukazují, do jaké míry je aplikace připravena plnit funkce požadované zákazníkem. Počet testovacích případů (TC), které selhaly, tedy určuje připravenost aplikace na předání zákazníkovi. Na základě výsledků akceptačních testů můžeme určit tři kategorie podle počtu TC, které selhaly. První kategorií jsou poruchové klony, kde nebyla sadou AT odhalena žádná chyba, byť poruchový klon defekt obsahuje.

Druhou kategorií je částečné selhání. Částečným selháním rozumíme chybové chování jednoho UC. Hrubým odhadem určíme poruchové klony spadající do druhé kategorie dle počtu selhaných testovacích případů. Průměrný počet TC testujících jeden UC je 5, z toho plyne, že do této kategorie zahrneme všechny poruchové klony s max pěti TC, které selhaly (*Failed*).

Třetí kategorie testů je „Aplikace je považována za nezpůsobilou k předání“. Do této kategorie spadají všechny ostatní klony, kde bylo sadou AT odhaleno více než pět chybných TC.

Z výsledků experimentu je patrné, že mezi závažností injektovaného defektu a počtem *Failed* TC není jednoznačný vztah. Naivním předpokladem by bylo očekávat, že se zvyšováním závažností defektu v poruchovém klonu bude selhávat více testovacích případů. Ve skutečnosti jsou jednotlivé TC vázané na případy užití (UC) aplikace, proto vztah „horší defekt = více selhání“ bude platit především v rámci scénářů pro jednotlivé UC. Hlavním faktorem počtu *Failed* TC je míra zapojení daného UC v rámci sady scénářových TC. Jako příklad použijeme operace UIS vytvoření a zrušení zkouškového termínu. Oba tyto UC jsou nepochybně součástí nějakého scénářového TC, ale operace zrušení je použita v mnohem menším počtu scénářových TC než operace vytvoření. Problém[5] v operaci vytvoření zkouškového termínu má za následek selhání více TC z důvodu, že více TC tuto operaci používá k ověření dalších vlastností aplikace.

### 6.4.1 Kategorie: Bez detekovaného selhání

Dle očekávání do této kategorie patří bezporuchový klon[6]. Dále jsou v této kategorii 4 poruchové klony obsahující defekty závažnosti low a medium a jeden klon obsahující defekt závažnosti high.

#### 01-C0.H0.M0.L1_S_S_07

V rámci testování tohoto klonu nebyla odhalena špatná barva tlačítka. Technologií automatických testů je velmi těžké odhalit výslednou vizuální stránku

[5] Nezáleží na tom, zda se jedná o injektovaný defekt či nově odhalené selhání.
[6] 00-C0.H0.M0.L0_ALL_OK

komponent, pokud nepoužíváme porovnání obrázků. Nejedná se tedy o nedostatek akceptační sady.

#### 02-C0.H0.M0.L1_S_S_03

V rámci testování tohoto klonu nebyl odhalen chybný nadpis stránky studenta *My Subjects*. Toto chování také nenarušuje funkcionalitu aplikace natolik, aby nemohla být akceptována. Všechny ostatní operace proběhly úspěšně. Chybu tohoto typu spolehlivě odhaluje sada funkčních testů, o požadované chování lze rozšířit další verzi SupportLibrary.

#### 07-C0.H0.M1.L0_S_S_08 a 08-C0.H0.M1.L0_S_S_12

V rámci testování těchto klonů nebyl odhalen student navíc v seznamu účastníků zkoušky. Od akceptační sady testů se očekává, že tento typ chyb bude odhalen. Během procesu přípravy sady akceptačních testů (navazování na operace SL a jejich rozšiřování) nebylo odhaleno, že operace SL pro kontrolu stránky *Registered Exam Dates* ověřuje pouze přítomnost aktuálně přihlášeného studenta.

Veřejné služby SupportLibrary, které využívá sada akceptačních testů, v tuto chvíli neobsahují ověření seznamů studentů registrovaných na zkouškovém termínu. Pro odhalení této chyby je nutné buď integrovat další část SL (*PageObjects*), nebo v budoucích verzích SupportLibrary v rámci vrstvy služeb implementovat novou metodu (operaci) pro kontrolu seznamů.

#### 18-C0.H1.M0.L0_S_S_06

V rámci testování tohoto klonu nebyl odhalen chybějící sloupec v tabulce na stránce učitele *Other Subjects*. Tato chyba je závažnější než špatný nadpis, nicméně typově se jedná o podobný defekt. Tabulka zobrazuje správný obsah, jen neúplný. Služba SL, která provádí kontrolu stránky, ověřuje pouze množinu očekávaných a zobrazených předmětů. Chybu tohoto typu odhaluje sada funkčních testů, o požadované chování lze rozšířit služby SupportLibrary v další verzi.

### 6.4.2 Kategorie: Menší počet selhání

Zde je třeba ověřit, zda v rámci případů, kdy selhalo malé množství scénářů, došlo ke skutečnému selhání, nebo jde o falešně pozitivní výsledek. V této kategorii jsou poruchové klony, pro které selhalo 5 nebo méně TC. V těchto případech se očekává selhání v rámci specifických podmínek daných UC. Do této kategorie spadá 11 konfigurací.

V rámci analýzy reportů akceptačních testů následujících konfigurací nedošlo k odhalení zajímavých či neočekávaných výsledků. Testy odhalily dle očekávání selhání v procesu (UC), který je ovlivněn injektovaným defektem.

- 04-C0.H0.M0.L1_S_S_04 – odhaleno chybné přesměrování po provedení akce
- 09-C0.H0.M1.L0_T_S_10 – odhaleno chybové chování pro záporný počet účastníků termínu
- 10-C0.H0.M1.L0_T_S_11 – odhaleno chybové chování pro záporný počet účastníků termínu
- 17-C0.H1.M0.L0_T_S_09 – odhaleno chybové chování pro záporný počet účastníků termínu
- 16-C0.H1.M0.L0_T_S_07 – odhalen chybný text hlášky aplikace
- 19-C0.H1.M0.L0_S_S_10 – odhalena chybná práce modifikací databáze
- 20-C0.H1.M0.L0_T_S_02 – odhalena chybná práce modifikací databáze
- 27-C1.H0.M0.L0_U_D_01 – odhalena chyba zobrazení tabulky (prázdná)

U následujících chybových klonů je vhodné popsat pozorované výsledky.

### 05-C0.H0.M1.L1_S_S_09

Detailnější zkoumání je vhodné věnovat poruchovému klonu *05*. Tento klon obsahuje podobný defekt jako poruchové klony *07* a *08* (z kategorie „bez detekovaného selhání“). V seznamu studentů se objevuje jeden navíc. Tentokrát se nejedná o virtuálního studenta, který je pouze zobrazen. Poslední přihlášený student je zdvojen a s tímto duplicitním záznamem aplikace pracuje. Pokud by se jednalo pouze o chybu zobrazení, jako u klonů *07* a *08*, chyba by nebyla odhalena. V tomto případě chyba není jen na úrovni zobrazení, ale také na vrstvě služeb UIS (viz [8]). Tato chyba tedy neovlivní jen zobrazený seznam, ale také vyhodnocení aktivity tlačítka „*Register*“, tlačítko při naplnění kapacity není aktivní. Je vhodné připomenout, že defekty v *07* a *08* modifikují jinou tabulku účastníků (pro stránky *My Exam Dates*).

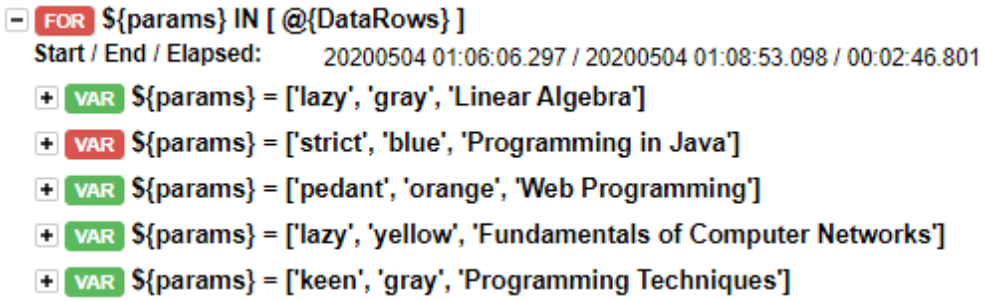
Chyba ovlivňuje dostupnost provedení akce, proto selže scénář pro kontrolu možnosti přihlášení maximálního počtu studentů na zkouškovém termínu. Již bylo řečeno, že rozšíření SupportLibrary o funkce kontrolující stav

tabulky a seznam účastníků by odhalilo chybu dříve. Z pohledu akceptačních testů je důležité reportovat stav, kdy očekávaná akce nemohla být provedena, což se stalo v tomto případě.

U tohoto klonu je vhodné věnovat pozornost faktu, že scénář, který odhalil selhání, má i variantu testu pro smoke testy (pouze jedna datová sada), který selhání neodhalil. Při přípravě smoke testu byla totiž náhodně vybrána konfigurace s právě jedním studentem, který se přihlásí na termín s právě jedním volným místem. V tuto chvíli aplikace funguje dle očekávání (nelze zdvojit posledního studenta), proto smoke test neodhalil selhání dle injektovaného defektu.

### 11-C0.H0.M1.L0_S_S_05

Při analýze reportu tohoto chybového klonu bylo zjištěno, že jde o falešně pozitivní výsledek (false positive). Došlo k selhání právě jednoho testovacího scénáře a to právě v jedné kombinaci dat. Pouze tento fakt nestačí k jasnému určení, zda jde o chybu testu, či nikoliv.




Obrázek 6.1: Log testovacího scénáře *TC Teacher Changes Evaluation Until Satisfaction*.

Při bližší analýze selhání testovacího scénáře bylo zjištěno, že se jedná o selhání, kdy nebylo zobrazeno oznámení o úspěšném přihlášení. Nesouvisí tedy s procesem změny hodnocení studenta. Krok přihlášení je zahrnut v každé operaci SupportLibrary. Nicméně testovací případy ověřující funkčnost scénáře přihlášení uživatele neodhalily selhání. Zároveň žádný další scénář neodhalil selhání v procesu přihlášení.

Závěrem analýzy tohoto reportu je, že během experimentu došlo k chybě v prostředí testu a je třeba tento test opakovat. Při opakování testu se tento klon zařadil do kategorie „Bez detekovaného selhání“. Sada akceptačních testů neodhalila následující chybové chování: Přihlášenému studentovi se po kliknutí na odkaz *Overview* nezobrazí stránka přehledu, ale *Other Available Exam Dates for My Subjects*. Příčinou, proč nebyl tento problém odhalen

akceptační sadou, je, že služby SupportLibrary nevyužívají při spuštění akce odkazy v menu, ale přímo přistupují na známé URL adresy. Tento problém odhalují funkcionální testy. Z pohledu akceptačních testů je vhodné zvážit, zda pozměnit chování SL, nebo zařadit do sady akceptačních testů kontrolu ovládání aplikace pomocí menu.

#### 23-C1.H0.M0.L0_T_S_03

Posledním poruchovým klonem, u kterého je vhodné podrobně diskutovat výsledek, je *23-C1.H0.M0.L0_T_S_03*. Tento poruchový klon obsahuje následující defekt: Stisknutí tlačítka „*X*“ na učitelské stránce „My Exam Dates“ smaže záznam z obrazovky, ale neprovede se smazání v databázi aplikace. Chyba v procesu UC.12 – *Zrušení zkouškového termínu* je správně odhalena. Zároveň selhaly testy pro UC.13 – *Zobrazení všech účastníků zkoušky*. Výstup testů odhalil chybu v řazení seznamu účastníků zkouškového termínu. Tato odhalená chyba nebyla popsaná v oficiálním popisu chyby zveřejněném na stránkách projektu [4], testy iniciovaly vylepšení dokumentace poruchového klonu.

### 6.4.3 Kategorie: Zásadní selhání

Ověření falešně pozitivních výsledků je třeba provést i v situacích, kdy se zdá být jasné, že v klonu UIS byla odhalena selhání. Je třeba ověřit, zda nedošlo k selhání z jiné příčiny, například chybě serveru nebo databáze. Do této skupiny spadá posledních 12 poruchových klonů UIS.

- 03-C0.H0.M0.L1_S_S_02 – odhaleno chybné zobrazení počtu kreditů
- 12-C0.H0.M1.L0_T_S_08 – odhaleno chybné chování filtru tabulky pro změnu hodnocení
- 13-C0.H1.M0.L0_G_T_D_01 – odhaleno chybné naplnění seznamu možných hodnocení
- 14-C0.H1.M0.L0_D_U_01 – odhalena modifikace zadaného data vytvářené zkoušky
- 15-C0.H1.M0.L0_D_U_02 – odhalena modifikace zadaného data vytvářené zkoušky
- 22-C1.H0.M0.L0_S_S_11 – odhaleno chybné uložení dat do databáze
- 24-C1.H0.M0.L0_T_S_06 – odhaleno chybné uložení dat do databáze

### 06-C0.H0.M0.L1_T_S_05

Popis reportu poruchového klonu *06* je důležitý ve vztahu k poruchovému klonu *18*. V obou případech jde o určitou modifikaci zobrazované tabulky. Chybový klon *18* odstranil sloupec s informací o učitelích, kteří vyučují předmět. V tomto poruchovém klonu *06* jsou zobrazeny všechny informace, ale sloupce jsou v jiném pořadí. Je jasné, že test, který ověřuje základní informaci (seznam vyučovaných předmětů), selže. Za předpokladu, že chyba s chybějícím sloupcem je označena jako primární cíl odhalení funkčními testy, je třeba konstatovat, že tento poruchový neobsahuje zásadní poruchu funkčnosti a akceptační testy vykazují falešně negativní výsledek.

### 21-C0.H1.M0.L0_T_S_04

U tohoto poruchového klonu opět můžeme pozorovat ve specifikaci nepopsanou chybu v řazení účastníků zkoušky. Stejné chování jsme pozorovali u poruchového klonu *23*. Z tohoto důvodu patří do této skupiny poruchových klonů. Poruchový klon charakterem výsledků zapadá lépe do druhé kategorie. Obsahuje selhání v právě jednom případu užití.

### 25-C1.H0.M0.L0_S_S_01 a 26-C1.H0.M0.L0_T_S_01

Během analýzy těchto poruchových klonů pozorujeme dříve popsaný efekt provázání chyb zobrazení se zdánlivě nesouvisejícími případy užití vykonávajícími akce. Chyba v zobrazení stránky způsobí, že není možné ověřit výsledek provedené akce v rámci jiného případu užití. Oba tyto poruchové klony nesou obdobnou zobrazovací chybu prázdné tabulky zkouškových termínů (*25* pro studenta a *26* pro učitele). V rámci ověření výsledku akce smazání nebo vytvoření nového zkouškového termínu tak dojde k chybě.

### 28-C2.H2.M1.L0_M_CR

Zcela dle očekávání tento poruchový klon, který obsahuje poruchy ve všech konfigurovatelných modulech (ve smyslu konfigurace poruch viz [9]), v akceptačních testech neobstál. Není překvapením, že funkční části aplikace jsou přihlášení, obnova databáze (ze souboru, resetem) a změna údajů uživatele. Nárůst počtu *Failed* TC souvisí s faktem, že poruchový klon obsahuje celkem 5 různých chyb z toho dvě závažné.

## 6.5 Zhodnocení experimentu

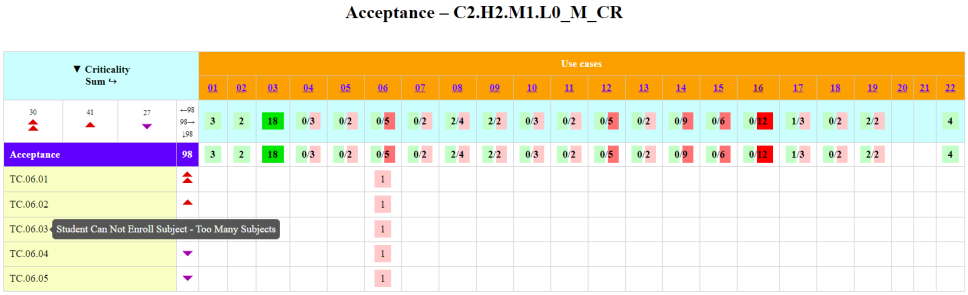
Experiment ověřil, že sada akceptačních testů obstála při ověřování, zda aplikace UIS a její poruchové klony splňují své specifikace, či nikoliv. Byla odhalena všechna selhání spojená s prováděním akcí v systému. Akcemi v systému je myšleno přihlašování na předměty a zkouškové termíny nebo vytváření nových zkouškových termínů. Akceptační testy jsou vhodné pro testování hraničních případů a dodržování omezení systému (maximální počet předmětů, pokusů, ...).

Připravená akceptační sada neodhalí defekty ve statických částech stránek UIS. Toto chování, které je odpovědností funkcionálních testů, je možné doplnit a rozšířit tak sadu AT. Nepříjemným faktem je, že během procesu rozšíření akcí SupportLibrary nebyla odhalena absence kontrol seznamu účastníků zkouškového seznamu z pohledu studenta (na úrovni služeb SL viz popis poruchových klonů *07* a *08*). Proto testy selhávají pouze v případě, kdy není možné provést akci přihlášení pro posledního studenta, protože je v seznamu student navíc. Aktuální sada akceptačních testů by detekovala selhání, pokud by v seznamu studentů byl nebo nebyl aktuální student. Služby SL by bylo vhodné rozšířit o tyto kontrolní operace dynamického obsahu a přidat odpovídající testovací případy do sady akceptačních testů.

Kromě ověření, že akceptační testy detekují očekávaná selhání, bylo díky nim zjištěno několik problémů v UIS, dokumentaci UIS a SL. Všechny tyto problémy byly odstraněny, buď v rámci této práce nebo vedoucím práce.

## 6.6 Zveřejnění výsledků

Výsledky experimentu byly zveřejněny na stránkách projektu TbUIS [4]. Forma zveřejnění je poskytnutí výstupních reportů akceptačních testů pro jednotlivé poruchové klony. Zároveň jsou na stránkách k dispozici tabulky mapující nalezené chyby na případy užití aplikace. Autorem tabulek je vedoucí práce.




Obrázek 6.2: Tabulka provázání případů užití a testovacích případů

Na Obrázku 6.2 je zobrazena tabulka mapování pro poruchový klon *28*. Díky rozšiřujícím Javascriptovým operacím může uživatel snadno změnit zobrazení podle testovacích případů, které selhaly. Tabulka se dá dále filtrovat podle závažnosti (criticality) jednotlivých testovacích scénářů. Tabulky všech poruchových klonů jsou k dispozici na přiloženém DVD ve složce s výsledky experimentu.

Všechny tyto informace nese také Robot Framework report (viz Obrázek 6.3). Výstupy Robot Frameworku nesou výrazně podrobnější informace o průběhu testů, jako jsou časy běhu a parametry jednotlivých klíčových slov. Přínos zjednodušené tabulky spočívá v redukci nadbytečné informace a přidání vazby na UC a závažnost testovacích scénářů.

Poslední částí zveřejnění výsledků je popis experimentu v kapitole 3 na stránkách projektu TbUIS `https://projects.kiv.zcu.cz/tbuis/web/page/testing`

# Acceptance Report

Generated
20200512 12:23:21 UTC+02:00
6 days 3 hours ago

## Summary Information

| | |
|---|---|
| **Status:** | 86 critical tests failed |
| **Start Time:** | 20200512 11:55:05.624 |
| **End Time:** | 20200512 12:23:11.861 |
| **Elapsed Time:** | 00:28:06.237 |
| **Log File:** | log.html |

## Test Statistics

| Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| Critical Tests | 122 | 36 | 86 | 00:24:10 | |
| All Tests | 122 | 36 | 86 | 00:24:10 | |

| Statistics by Tag | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| CRITICAL | 35 | 10 | 25 | 00:06:13 | |
| HappyDay | 25 | 2 | 23 | 00:06:03 | |
| MAJOR | 49 | 14 | 35 | 00:10:39 | |
| MINOR | 38 | 12 | 26 | 00:07:18 | |
| Scenario | 24 | 2 | 22 | 00:07:55 | |
| Self | 8 | 6 | 2 | 00:01:16 | |
| Smoke | 23 | 12 | 11 | 00:03:26 | |
| UC.01 | 3 | 3 | 0 | 00:00:33 | |
| UC.02 | 2 | 2 | 0 | 00:00:10 | |
| UC.03 | 18 | 18 | 0 | 00:02:57 | |
| UC.04 | 3 | 0 | 3 | 00:00:32 | |
| UC.05 | 2 | 0 | 2 | 00:01:15 | |
| UC.06 | 5 | 0 | 5 | 00:00:43 | |
| UC.07 | 2 | 0 | 2 | 00:00:31 | |
| UC.08 | 6 | 2 | 4 | 00:01:03 | |
| UC.09 | 4 | 2 | 2 | 00:01:59 | |
| UC.10 | 3 | 0 | 3 | 00:00:16 | |
| UC.11 | 2 | 0 | 2 | 00:00:16 | |
| UC.12 | 5 | 0 | 5 | 00:00:41 | |
| UC.13 | 2 | 0 | 2 | 00:00:31 | |
| UC.14 | 9 | 0 | 9 | 00:00:45 | |
| UC.15 | 6 | 0 | 6 | 00:00:39 | |
| UC.16 | 12 | 0 | 12 | 00:01:10 | |
| UC.17 | 4 | 1 | 3 | 00:00:43 | |
| UC.18 | 2 | 0 | 2 | 00:00:17 | |
| UC.19 | 4 | 2 | 2 | 00:00:38 | |
| UC.22 | 4 | 4 | 0 | 00:00:38 | |

Obrázek 6.3: Tabulka tagů reportu Robot Frameworku.

# 7 Závěr

V rámci této diplomové práce byla provedena rešerše dostupných nástrojů pro automatické akceptační testování. Na základě této rešerše byl vybrán Robot Framework, jako nástroj vhodný pro přípravu akceptačních testů aplikace UIS.

S využitím prostředků Robot Frameworku a existující SupportLibrary bylo navrženo řešení realizující akceptační testy aplikace UIS. Součástí tohoto řešení je návrh struktury projektu, technické řešení integrace SupportLibrary a příprava jednotlivých testovacích scénářů. Součástí řešení integrace SL je vytvoření utility automatického generátoru obalových tříd SupportLibrary (wrapperů).

Díky připraveným „byznysovým“ klíčovým slovům je příprava jednoho konkrétního scénáře otázkou několika desítek minut včetně ověření žádaného chování. Existující klíčová slova umožňují připravovat scénáře téměř v přirozeném jazyce. Navržené oddělení datových řad a scénářů umožňuje efektivně kontrolovat změny provedené ve akceptační sadě testů.

Vytvořená sada akceptačních testů odhaluje většinu chyb dynamického obsahu aplikace v jednotlivých poruchových klonech UIS. Dynamickým obsahem aplikace rozumíme přihlašování (předměty, zkouškové termíny) či vytváření nových objektů (zkouškové termíny) v aplikaci. Sada testů neodhalila právě dvě chyby dynamického obsahu. Akceptační sada testů se nezaměřuje na statický obsah stránky, který je pokryt funkcionálními testy.

V rámci vývoje sady akceptačních testů jsem rozšířil SupportLibrary o několik nových operací, které byly potřeba pro nové způsoby strojového ovládání aplikace UIS a kontrolu některých částí.

Z důvodu potřeby existence bezporuchového klonu UIS byla nad rámec zadání této práce provedena oprava 28 reportovaných chyb aplikace UIS.

Dalším krokem vylepšení projektu akceptačních testů by mohlo být doplnění automatického generování obalových tříd v prostředí Robot Frameworku. Jinou, z pohledu ověřování kvality software zajímavější, oblastí je z připravené podrobné specifikace generovat scénáře testů RF automatizovaným způsobem. Výstupem automatického generování scénářů by byla definice posloupností operací pro testovací případy ze specifikace.

Výsledky experimentu byly publikovány na webových stránkách projektu TbUIS [4].
Domnívám se, že jsem zadání diplomové práce splnil v celém rozsahu.

# Seznam použitých zkratek

- AT – akceptační testování
- AAT – automatické akceptační testování
- BBD – behavior-driven development
- CI – continous integration
- DB – databáze
- DevOps – developer operations
- DoD – definition of done
- FIT – Framework for Integrated Testing
- GUI – Graphical User Interface – grafické uživatelské rozhraní
- IDE – integrated development environment – komplexní prostředí pro vývoj programů, např. MS Visual Studio, IntelliJ Idea
- UAT – user acceptance tests – uživatelské akceptační testování
- UC – use case – případ užití – popis kroků k dosažení určitého cíle0
- UIS – university information system – univerzitní informační systém
- TbUIS – test bed UIS
- TC – test case – testovací případ

# Seznam obrázků

# Seznam tabulek

# Literatura


[1] Beck, K. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[2] Børge Haugset, T. S. Automated Acceptance Testing as an Agile Requirements Engineering Practice. 2012.

[3] Haugset, B. – Hanssen, G. K. Automated acceptance testing: A literature review and an industrial case study. In *Agile 2008 Conference*, s. 27–38. IEEE, 2008.

[4] Herout, P. *Projekt TbUIS* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, v Plzni, 2020. [cit. 10.5.2020]. Dostupné z: `https://projects.kiv.zcu.cz/tbuis`.

[5] Herout, P. *Specifikace systému UIS* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, v Plzni, 2020. [cit. 10.5.2020]. součást projektu TbUIS. Dostupné z: `https://projects.kiv.zcu.cz/tbuis/web/page/uis`.

[6] Johannes Weiss, I. R. Literature Review of Empirical Research Studies within the Domain of Acceptance Testing. In *42th Euromicro Conference on Software Engineering and Advanced Applications*, s. 181–188. IEEE, 2016.

[7] Leotta, M. et al. An acceptance testing approach for Internet of Things systems. *IEEET Software*. 2018, 12, 5, s. 430–436.

[8] Matyáš, J. Aplikace s možností injekce chyb pro ověřování kvality testu. *Diplomová práce*. 2018.

[9] Šmaus, J. Rozhraní pro administraci aplikace s možností injekce chyb. *Diplomová práce*. 2019.

[10] Miller, R. – Collins, C. T. Acceptance testing. *Proc. XPUniverse*. 2001, vol. 238.

[11] Obiltschnig, D. C++ for Safety-Critical Systems. 04 2012.

[12] *Robot Framework User Guide* [online]. Robot Framework Foundation, 2020. [cit. 8.3.2020]. Dostupné z: `https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html`.

# Seznam příloh

# Příloha A – Obsah DVD

- **bin/** – binární soubory aplikací
  - defect/ – složka se standardními poruchovými klony UIS
  - third-party/install – složka s instalačními soubory SW
  - third-party/maven – složka s Maven závislostmi projektů UIS, SupportLibrary, TbUIS-RobotFramework, WrapperGenerator
  - **RobotFramework.jar** – přeložená verze obalových tříd SL
  - SupportLibrary-java8.jar – verze SupportLibrary pro Javu 8
  - UIS.war – bezporuchový klon aplikace UIS v1.7.2
  - **WrapperGenerator.jar** – utilita pro generování obalových tříd
- **poster/** – složka se soubory posteru
  - **Vais_Radek_2020.pdf** – pdf soubor s posterem práce
  - **Vais_Radek_2020.pub** – zdrojový soubor posteru práce
- **runner/**
  - **results/** – složka s výsledky experimentů
  - **test-runner.cmd** – dávkový soubor k řízení experimentu
  - **open-reports.cmd** – dávkový soubor pro otevření všech reportů zvoleného experimentu
- TbUIS-UIS/ – složka s projektem UIS
- TbUIS-SupportLibrary/ – složka s projektem SupportLibrary
- **TbUIS-RobotFramework/** – složka projektu akceptačních testů
  - **configuration/** – složka s připravenými konfiguracemi SL
  - **src/** – složka zdrojových kódů aplikace
  - **WrapperGenerator/** – složka s projektem WrapperGenerator
    - **src/** – složka zdrojových kódů utility
    - **packages.config** – konfigurační soubor utility
- **tex** – zdrojové soubory textu DP
- **Vais_Radek_Diplomova_Prace.pdf** – pdf soubor této práce

Tučně jsou označeny výstupy této práce.

# Příloha B – Spuštění automatického experimentu

Následující manuál popisuje využití připraveného skriptu pro řízení experimentu včetně přípravy prostředí pro jeho spuštění.

## Prerekvizity

Z příkazové řádky musí být dostupné následující nástroje:

- **git** – nástroj pro správu verzí
- **mvn** – nástroj Maven pro správu projektu

Nástroj Git je použitý k nastavení žádané verze závislostí pro daný běh experimentu. Nástroj nevyžaduje doplňující konfiguraci.

Nástroj Maven je použitý k řízení sestavení aplikace a nasazení UIS na lokální Tomcat server. Při prvním spuštění je třeba přístup na internet, aby nástroj mohl stáhnout z veřejných repozitory závislosti projektu (případně lze použít soubory z DVD `bin/third-parties/maven`). Z důvodů problémů s kompatibilitou pluginu pro spuštění Robot frameworku s Javou verze 11 je potřeba nastavit Maven tak, aby používal Javu verze 8. Toho lze docílit nastavením proměnné prostředí systému `JAVA_HOME` například doplněním příkazu `set JAVA_HOME=<path/to/java8>` na začátek řídícího skriptu. Druhou potřebnou změnou konfigurace Mavenu je nastavení hesla pro přístup k serveru Tomcat (vytvoření přístupu viz dále). Nastavení provedeme modifikací konfiguračního souboru `~/.m2/settings.xml`(`~` značí domovský adresář aktuálního uživatele), do kterého je třeba doplnit následující sekci definice serveru (`<server>`) do kolekce serverů (`<servers>`):

```
<settings>
  <servers>
    <server>
      <id>TomcatServer</id>
      <username>maven</username>
      <password>maven</password>
    </server>
  </servers>
</settings>
```

Je třeba dodržet obsah elementu `<id>`, který je použit pro identifikaci přístupových údajů v definici projektu UIS (`pom.xml`). Uživatelské jméno a heslo musí odpovídat zřízenému přístupu na server Tomcat.

Dále je potřeba mít na lokálním počítači nainstalovanou a spuštěnou instanci MySql databáze s vytvořenou databází dle následující konfigurace:

- Název: `uis-web-db`
- Kódování: `UTF8_general_ci`
- Uživatel: `uis-web`
- Heslo: `uis`

Poslední prerekvizitou je lokální instance Tomcat serveru. Pro potřeby automatického nasazení na server je třeba před jeho spuštěním modifikovat konfiguraci serveru a umožnit tak automatické nasazení aplikace. Je třeba modifikovat soubor `conf/tomcat-users.xml` ve složce serveru. Do souboru je třeba doplnit následující konfiguraci, která zajistí uživateli maven přístupová práva pro nasazení aplikace:

```
<tomcat-users>
  <role rolename="manager-script"/>
  <user username="maven"
        password="maven"
        roles="manager-script"/>
</tomcat-users>
```

## Konfigurace experimentu

Ve výchozím stavu řídící skript experimentu předpokládá, že následující projekty jsou umístěné ve stejné složce:

- **runner/** – složka s řídícím skriptem
- **TbUIS-UIS/** – složka s projektem UIS
- **TbUIS-SupportLibrary/** – složka s projektem SupportLibrary
- **TbUIS-RobotFramework/** – složka s projektem akceptačních testů

Nastavení cesty k jednotlivým projektům lze změnit konfigurací skriptu. V případě změny je nezbytné použít absolutní adresy u všech projektů (např. `C:/xxx/runner`).

Dále script obsahuje nastavení verzí označení systému git (*branch*, *tag* nebo *commit hash*) a proměnnou určující, zda bude nastavení verzí provedeno. Pokud je nastavení verzí vypnuté, budou použity aktuální verze jednotlivých projektů. Toto chování je vhodné využít při vývoji jedné ze závislých komponent (UIS, SL).

Dalším parametrem je výchozí název výstupní složky experimentu. Ve výchozím stavu script používá jako název složky datum ve formátu `DDMMYYYY`. Pokud již složka dle daného dne existuje, script rozšíří název o text `-\d` (např. `11052020-1`). Script obsahuje bezpečnostní omezení pro maximální počet vytvořených složek pro jeden den, jehož výchozí hodnota je 8.

Posledním parametrem konfigurace je složka s definicemi poruchových verzí, které mají být testovány. Složka musí obsahovat pouze validní soubory `seed.xml` aplikace UIS. Zároveň musí být složka umístěna v projektu TbUIS-UIS. Jako výchozí je použita složka `version_seeds`, která obsahuje definice 28 poruchových klonů a jednoho klonu bezporuchového.

## Průběh experimentu

Nejprve je připraveno prostředí pro experiment prostřednictvím nástroje Git a Maven:

1. příprava nové podsložky pro výsledky
2. nastavení verze SupportLibrary
3. kompilace a instalace SupportLibrary
4. nastavení verze UIS
5. kompilace a instalace TbUIS-RobotFramework – obalové třídy SupportLibrary

Po přípravě prostředí jsou v cyklu řízeném seznamem poruchových verzí (viz projekt TbUIS[1]) prováděny následující činnosti:

1. příprava poruchového klonu dle konfigurace
2. nasazení na server (localhost)
3. odstranění dočasných souborů akceptačních testů
4. spuštění sady akceptačních testů

[1]Viz kapitola 3 nebo webové stránky projektu: `https://projects.kiv.zcu.cz/tbuis/web/page/download`

5. uložení výsledků

Výstupem řídícího scriptu není jen soubor výsledků sady akceptačních testů, ale také výstup jednotlivých kroků přípravy (výstup programu Maven), aby bylo možné ověřit, zda nedošlo k chybě při přípravě experimentu.

## Složka results

Výstupem jednoho běhu řídícího scriptu je nová podsložka složky `results`. Ve výchozím stavu je název složky odvozen o data spuštění experimentu. Tato složka obsahuje záznamy kompilací a jednotlivých spuštěných testů. V případě, že experiment obsahoval otestování právě jednoho klonu UIS, jehož verze byla dána souborem MySeed.xml, bude struktura souborů následující:

- **MySeed.xml/** – složka s výsledky klonu
  - **robotframework-reports/** – složka s reportem Robot frameworku
    - **log.html** – záznamy průběhu testů v RF
    - **output.xml** – log průběhu testů RF
    - **report.html** – report z provedených testů RF
    - **TEST-acceptance.xml** – seznam testovacích případů provedené sady testů RF
  - **RF-mvn.log** – výstup nástroje Maven pro průběh testů
  - **test-config-log.txt** – log SupportLibrary načítání konfigurace
  - **test-results-log.txt** – log SupportLibrary běhu knihovny
  - **UIS-mvn.log** – výstup nástroje Maven pro překlad a nasazení klonu UIS
- **SL-compile-mvn.log** – výstup nástroje Maven pro kompilaci SupportLibrary
- **RF-compile-mvn.log** – výstup nástroje Maven pro kompilaci obalových tříd SupportLibrary

V případě experimentu s více klony UIS bude pro každý klon UIS vytvořena nová podsložka dle názvu *xml* souboru.

# Příloha C – Tabulka testovacích scénářů

| Případ užití | Název testovacího případu |
|---|---|
| UC.01 | TC.01.01 CorrectLogin<br>TC.01.02 IncorrectLogin<br>TC.01.03 Login Error – Unknown user |
| UC.02 | TC.02.01 CorrectLogout<br>TC.02.02 Logout Success – teacher |
| UC.03 | TC.03.01 Change Students Emails<br>TC.03.02 Change Students Emails Smoke<br>TC.03.03 Change Students Email Should Fail – constraints<br>TC.03.04 Change Students Firstname<br>TC.03.05 Change Students Firstname Smoke<br>TC.03.06 Change Students Firstname Fails – Constraints<br>TC.03.07 Change Students Lastname<br>TC.03.08 Change Students Lastname Smoke<br>TC.03.09 Change Students Lastname Fails – Constraints<br>TC.03.10 Change Teachers Email<br>TC.03.11 Change Teachers Email – Smoke<br>TC.03.12 Change Teachers Emails Fail – Constraints<br>TC.03.13 Change Teachers Firstname<br>TC.03.14 Change Teachers Firstname – Smoke<br>TC.03.15 Change Teachers Firstname Fails – Constraints<br>TC.03.16 Change Teachers Lastname<br>TC.03.17 Change Teachers Lastname Smoke<br>TC.03.18 Change Teachers Lastname Fails – Constraints |
| UC.04 | TC.04.01 Student Cancel Registration Of Subject<br>TC.04.02 Student Cancel Registration Of Subject Change Select Box<br>TC.04.03 Student Cancel Registration Of Subject Fails – Unregistered Subject |
| UC.05 | TC.05.01 Display Subject<br>TC.05.02 Display Subject – Smoke |
| UC.06 | TC.06.01 Student Enroll Subject |

| Případ užití | Název testovacího případu |
|---|---|
| | TC.06.02 Student Can Not Enroll Subject – No Teacher<br>TC.06.03 Student Can Not Enroll Subject – Too Many Subjects<br>TC.06.04 Student Can Not Enroll Subject – Already Study Subject<br>TC.06.05 Student Display Other Subjects Page |
| UC.07 | TC.07.01 Student Unregister Exam Date<br>TC.07.02 Student Unregister Exam Date Failed |
| UC.08 | TC.08.01 Student Enroll Exam Date<br>TC.08.02 Student Enroll Exam Date – Smoke<br>TC.08.03 Student Can Not Enroll – One Exam Date Twice<br>TC.08.04 Student Can Not Enroll Exam Date – Already Enrolled<br>TC.08.05 Student Can Not Enroll Exam Date – Not Registered subject<br>TC.08.06 Student Display Other Exams |
| UC.09 | TC.09.01 Display And Check Exam<br>TC.09.02 Display And Check Exam – Smoke<br>TC.09.03 Display And Check Other Exams<br>TC.09.04 Display And Check Other Exams – Smoke |
| UC.10 | TC.10.01 Teacher Cancel Teaching Of Subject<br>TC.10.02 Teacher Cancel Teaching Of Subject Fails – Some Students Enrolled<br>TC.10.03 Teacher Cancel Teaching Of Subject Fails – Teacher does not teach this subject |
| UC.11 | TC.11.01 Teacher Display Subject<br>TC.11.02 Teacher Display Subject – Smoke |
| UC.12 | TC.12.01 Teacher Cancel Exam – No students enrolled<br>TC.12.02 Teacher Cancel Exam – Some students enrolled<br>TC.12.03 Teacher Cancel Exam Failed – Subject with exam not found<br>TC.12.04 Teacher Cancel Exam Date Cancelation<br>TC.12.05 Teacher Cancel Exam Date Check Participants |
| UC.13 | TC.13.01 Teacher Display Exam<br>TC.13.02 Teacher Display Exam – Smoke |
| UC.14 | TC.14.01 Teacher Create Exam<br>TC.14.02 Teacher Create Exam – Custom Date |

| Případ užití | Název testovacího případu |
|---|---|
| | TC.14.03 Teacher Create Exam – Custom participants number<br>TC.14.04 Teacher Create Exam Failed – Not Teaching Subject<br>TC.14.05 Teacher Create Exam Failed – Wrong Date<br>TC.14.06 Teacher Create Exam Failed – Number Format Out Of Range<br>TC.14.07 Teacher Create Exam Failed – Wrong Number Format<br>TC.14.08 Teacher Create Exam Failed – Date In History<br>TC.14.09 Teacher Create Exam Failed – Wrong date order |
| UC.15 | TC.15.01 Teacher Evaluates A Exam using Form<br>TC.15.02 Teacher Evaluates B Exam using Form<br>TC.15.03 Teacher Evaluates C Exam using Form<br>TC.15.04 Teacher Evaluates D Exam using Form<br>TC.15.05 Teacher Evaluates E Exam using Form<br>TC.15.06 Teacher Evaluates F Exam using Form |
| UC.16 | TC.16.01 Teacher Evaluates A Exam using Form<br>TC.16.02 Teacher Evaluates F Exam using Form<br>TC.16.03 Teacher Evaluates B Exam using Form<br>TC.16.04 Teacher Evaluates C Exam using Form<br>TC.16.05 Teacher Evaluates D Exam using Form<br>TC.16.06 Teacher Evaluates E Exam using Form<br>TC.16.07 Teacher Updates Evaluation to A Using Table<br>TC.16.08 Teacher Updates Evaluation to F Using Table<br>TC.16.09 Teacher Updates Evaluation to B Using Table<br>TC.16.10 Teacher Updates Evaluation to C Using Table<br>TC.16.11 Teacher Updates Evaluation to D Using Table<br>TC.16.12 Teacher Updates Evaluation to E Using Table |
| UC.17 | TC.17.01 Teacher Start Teaching<br>TC.17.02 Teacher Start Teaching Too Many Subjects<br>TC.17.03 Teacher Start Teaching Aleready Full<br>TC.17.04 Teacher Start Teaching – Smoke |
| UC.18 | TC.18.01 Teacher Display Teachers<br>TC.18.02 Teacher Display Teachers |
| UC.19 | TC.19.01 Test Restore DB On Suite Start<br>TC.19.02 Test Restore DB On Test Start<br>TC.19.03 Test Restore DB On Suite Start – Check Content |

| Případ užití | Název testovacího případu |
|---|---|
| | TC.19.04 Test Restore DB On Test Start – Check Content |
| UC.22 | TC.22.01 Test Restore Custom DB On Suite Start<br>TC.22.02 Test Restore Custom DB On Test Start<br>TC.22.03 Test Restore Custom DB On Suite Start – Check Content<br>TC.22.04 Test Restore Custom DB On Test Start – Check Content |

Tabulka 7.1: Základní testovací scénáře dle případů užití