ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Automatické testování procesů v Camunda BPM

Zuzana Pavlatová
Softwarové inženýrství a technologie

Srpen 2021
Vedoucí práce: Ing. Lukáš Zoubek

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pavlatová** Jméno: **Zuzana** Osobní číslo: **466924**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Automatické testování procesů v Camunda BPM

Název bakalářské práce anglicky:

Automatic testing of processes in Camunda BPM

Pokyny pro vypracování:

1) Popište možnosti automatického testování aplikací
2) Představte technologii Camunda BPM a její využití na FEL ČVUT
3) Navrhněte způsob automatického testování elektronizovaných procesů studijního oddělení FEL ČVUT
4) Vytvořte automatické testy vybraného procesu SO
5) Formulujte doporučení pro implementaci a testování dalších procesů

Seznam doporučené literatury:

PATTON, Ron. Testování softwaru. Vyd. 1. Praha: Computer Press, c2002. 313 s. Programování. ISBN 8072266365.
ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Zoubek, katedra softwarového inženýrství FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Lukáš Zoubek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání	Podpis studentky

Poděkování / Prohlášení

Děkuji vedoucímu práce, Ing. Lukáši Zoubkovi, za jeho trpělivost a cenné rady. Dále Bc. Danielu Groschupovi za vše, co pro mě během studia udělal a za veškerou pomoc s touto prací. Také děkuji všem kolegům z CZM, kteří mi jakkoliv pomohli objasnit záludnosti Camundy. Velké díky patří rodině a kamarádům a především mojí úžasné mamince, která si z mého studia snad odnesla alespoň znalost rozdílu mezi hardware a software.

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 13. 8. 2021

....

Abstrakt / Abstract

Tato práce analyzuje možnosti automatizovaného testování procesů v aplikaci eProcesy/Camunda fungující na platformě Camunda BPM. Práce zahrnuje informace o testování software a automatizaci testů. Vysvětluje, co je Camunda BPM a přibližuje, jak se využívá na ČVUT FEL. Dále je zvolen proces, který je podrobně popsán a jsou pro něj navrženy vhodné procesní testy a nástroje, které budou pro testování využity. Tyto testy jsou implementovány a jsou sepsána doporučení pro další postup při automatizaci testů pro další procesy eProcesů/Camundy.

Klíčová slova: ČVUT, FEL, proces, Camunda, BPM, testování, software

This work analyzes the possibilities of automated process testing of eProcesy/Camunda application operating on the Camunda BPM platform. The work includes information about software testing and test automation. It explains what Camunda BPM is and how it is used at CTU FEE. Furthermore, a process is selected and it is described. Suitable process tests are designed and tools for test automation are selected. These tests are implemented and recommendations for further steps in automating tests for other eProcesy/Camunda processes are written.

Keywords: CTU, FEE, process, Camunda, BPM, testing, sofware

Obsah /

1 Úvod ... 1
1.1 Motivace ... 1
1.2 Cíle práce ... 1
2 Testování software ... 3
2.1 Teorie ... 3
2.2 Rozdělení způsobů testování ... 3
2.2.1 Černá a bílá skříňka ... 3
2.2.2 Statické a dynamické testování ... 4
2.2.3 Testy splněním a selháním ... 4
2.2.4 Manuální a automatické testování ... 4
2.3 Úrovně testování ... 4
2.3.1 Unit testy ... 5
2.3.2 Integrační testy ... 5
2.3.3 Systémové testy ... 5
2.3.4 Akceptační testy ... 5
2.4 Procesní testování ... 6
3 Automatizované testování ... 7
3.1 Popis ... 7
3.2 Vlastnosti ... 7
3.3 Porovnání automatizovaného a manuálního testování ... 8
3.3.1 Základní popis ... 8
3.3.2 Rychlost ... 8
3.3.3 Lidské zdroje ... 8
3.3.4 Spolehlivost ... 8
3.3.5 Programovatelnost ... 8
3.4 Techniky ... 8
3.5 Proces automatizovaného testování ... 9
3.5.1 Výběr testovacího nástroje ... 9
3.5.2 Definice rozsahu automatizace ... 9
3.5.3 Plánování, návrh a vývoj testů ... 9
3.5.4 Spuštění testů ... 9
3.5.5 Údržba testů ... 9
3.6 Vybrané nástroje pro automatizaci ... 9
3.6.1 JUnit ... 10
3.6.2 TestNG ... 10
3.6.3 Mockito ... 10
3.6.4 Selenium ... 11
4 Elektronizace podnikových procesů ... 12
4.1 Procesní řízení ... 12
4.1.1 Podnikový proces ... 12
4.1.2 Modelování podnikových procesů ... 12
4.2 Notace BPMN ... 13
4.2.1 Tokové objekty ... 13
4.2.2 Spojovací objekty ... 14
4.2.3 Plavecké dráhy ... 14
4.2.4 Artefakty ... 14
4.2.5 Data ... 14
4.3 Notace DMN ... 14
4.4 Camunda BPM ... 15
4.4.1 Využití Camunda BPM na ČVUT FEL ... 16
5 Vybraný proces aplikace eProcesy/Camunda ... 17
5.1 Současný stav ... 17
5.2 Popis vybraného procesu ... 17
5.2.1 Popis workflow procesu ... 17
5.2.2 Aktéři ... 17
5.2.3 Prvky procesu ... 18
5.3 Další vlastnosti procesu ... 19
6 Příprava na implementaci ... 20
6.1 Výběr testovacího nástroje ... 20
6.2 Definice rozsahu automatizace ... 20
6.3 Návrh testů ... 21
6.3.1 Pokrytí cest procesu ... 21
7 Implementace procesních testů ... 23
7.1 Implementace testů ... 23
7.1.1 Příprava prostředí ... 23
7.1.2 Příprava testovací třídy ... 24
7.1.3 Důležité části testů ... 26
7.1.4 Popis jednotlivých testů ... 28
7.2 Spuštění testů ... 30
7.3 Údržba testů ... 30
7.4 Vyhodnocení testů a doporučení pro další implementaci ... 30
8 Závěr ... 32
Literatura ... 33
A Procesní diagram žádosti o individuální studijní plán ... 35
B Happy path procesu Žádost o ISP ... 36

C Diagram bodů a cest procesu 37
D Přehled všech prvků procesu 38
E 42
F Ukázka rozhodovací DMN tabulky pro výběr referentek 43
G Definice proměnných testovací třídy 44
H Přehled vytvořených testů 45
I Vyhodnocení testu *shouldExecuteHappyPath()* 46
J Seznam použitých zkratek a pojmů 47

/ Obrázky

2.1. V model ... 4
2.2. Princip unit testu ... 5
3.1. Selenium - komponenty ... 11
4.1. Příklad značení procesu v notaci BPMN 2.0 ... 13
4.2. Vztah BPMN a DMN ... 15
4.3. Architektura platformy Camunda ... 16
6.1. ... 22
7.1. Struktura projektu ... 23
7.2. Struktura a soubory pro testování ... 24
7.3. Dependencies v pom.xml ... 24
7.4. Třída ApplicationProcessTest ... 25
7.5. Definice procesního enginu ... 25
7.6. Využití *@Mock* ... 25
7.7. Registrace mocků ... 25
7.8. Nastavení výchozího chování úkolů ... 26
7.9. Spouštění testovaného scénáře ... 26
7.10. Metody pro ověřování průchodu procesem ... 27
7.11. ... 27
7.12. ... 28
7.13. Test pro ověření DMN tabulky ... 29
7.14. Test pro ověření spuštění přidružené časové události ... 29
7.15. Vyhodnocení testů ... 30
7.16. Vyhodnocení pokrytí procesu testy ... 30

Kapitola 1
Úvod

V roce 2019 byla do pilotního provozu nasazena aplikace pro elektronizaci žádostí studentů Fakulty elektrotechnické ČVUT v Praze. Tato aplikace funguje na platformě **Camunda BPM** pro automatizaci procesů. Umožňuje studentům generovat formuláře, které poté odevzdají na studijní oddělení. Několik formulářů je již plně elektronizovaných a dají se tedy vyřídit bez nutnosti návštěvy studijního oddělení. Studenti vidí v aplikaci **eProcesy** stav žádosti, mohou jí editovat a stáhnout si vygenerovaný formulář, případně rozhodnutí. Studijní referentky mohou v aplikaci **Camunda** vybrané žádosti kontrolovat, podstupovat je dále proděkanovi ke schválení a následně generovat rozhodnutí o studentově žádosti. Mimoto systém automaticky zasílá notifikace studentovi při změně stavu jeho žádosti a referentce při obdržení nové žádosti. Až doposud byla aplikace testována převážně manuálně a je potřeba testy urychlit a zefektivnit.

Tato práce se tedy bude zabývat analýzou automatizace testů ověřujících správné fungování konkrétního procesu. Nejprve obecně rozebereme problematiku testování software a poté se již zaměříme přímo na automatizované testy. Vysvětlíme si, co je Camunda BPM a přiblížíme si její fungování a využití na fakultě a popíšeme si procesy, které zachycuje. Poté už navrhneme konkrétní druhy a nástroje automatického testování vybraného procesu. Testy budou následně implementovány a vyhodnoceny.

1.1 Motivace

Automatizované testy by měly zefektivnit a usnadnit testování softwaru a tím přinést i časovou a finanční úsporu. Jejich efektivita spočívá především v tom, že umožňují opakovaně spouštět testy a testovat funkcionality bez vynaložení dalšího úsilí až do doby, než se funkcionalita změní - pak musí být uzpůsobeny i testy na ní závislé.

Fakultní aplikace pro elektronizaci procesů studijního oddělení, na kterou se v této práci zaměříme, byla dosud testována manuálně a tedy musely být při každé změně v aplikaci využity lidské zdroje, které jí znovu otestovaly, což bylo velice neefektivní. Z toho důvodu je potřeba aplikaci pokrýt automatizovanými testy, které sníží čas a zdroje použité k manuálnímu testování.

1.2 Cíle práce

Hlavním cílem této bakalářské práce je analyzovat možnosti automatizovaného testování procesů v aplikaci eProcesy/Camunda. Na základě analýzy poté, pokud to bude možné, vytvořit a vyhodnotit testy pro vybraný proces. Tento cíl rozpadneme na několik dílčích cílů, které zároveň povedou ke splnění hlavního cíle. Dílčí cíle jsou následující:

- Obecně popsat automatické testování a možnosti, jak testovat.
- Představit technologii Camunda BPM a její využití na Fakultě elektrotechnické.
- Zanalyzovat možnosti automatického testování elektronizovaných procesů studijního oddělení a navrhnout způsoby testování.

- Vhodně implementovat testy na konkrétní proces.
- Doporučit další postup v testování a implementaci procesů.

Kapitola 2
Testování software

V této kapitole popíšeme, co je testování software, jaké existují druhy a úrovně testů a k čemu se využívají.

2.1 Teorie

Existují různé definice testování softwaru. V této práci se budeme řídit následující definicí: *Jedná se o proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů.*[1] Kromě toho to zahrnuje i přípravu na testování, tedy statickou analýzu zkoumající například požadavky nebo kód, návrh testů a po otestování i jejich vyhodnocení.

Pro testování softwaru existují obecně známé **axiomy**, které definuje Patton: [2]

- *Žádný netriviální program není možné otestovat úplně.* Je prakticky nemožné software zcela otestovat a odhalit všechny defekty. Hodnot k otestování je příliš mnoho a není možné je použít všechny.
- *Testování je věcí odhadu rizika.* Z prvního axiomu je zřejmé, že nemůžeme otestovat vše a proto vybíráme, co a jak otestovat dle vlastního uvážení, které by mělo minimalizovat riziko přehlédnutí defektů.
- *Testování může prokázat jen přítomnost defektů, nikoli jejich absenci.* Opět vycházíme z prvního axiomu, jelikož pokud software neotestujeme úplně, nikdy si nemůžeme být jistí, že jsme nějaký defekt nevynechali.
- *Čím více defektů je nalezeno, tím více jich v produktu je.* Obvykle se defekty vyskytují ve skupinách, jelikož programátor dělá často stejné chyby. Pokud tedy nalezneme vice defektů v jedné části systému, je velice pravděpodobné, že tam budou další.

2.2 Rozdělení způsobů testování

Testovat můžeme dle různých parametrů, jako je kód nebo způsob vyhodnocení testů. V následujících podkapitolách si několik takových druhů testů představíme.

2.2.1 Černá a bílá skříňka

Jedná se o **rozdělení podle znalosti vnitřní struktury**. Dle definice Rona Pattona v případě **černé skříňky** známe způsob fungování softwaru a víme tedy, co bude na vstupu a co očekáváme na výstupu jednotlivých funkcí. Neznáme ale způsob, jakým k tomu dojít. Může se potom stát, že zbytečně důkladně testujeme velice jednoduchou funkcionalitu a naopak málo otestujeme mnohem složitější část kódu.[2]

Princip **bílé skřínky** je opačný - máme přístup ke zdrojovému kódu a můžeme ho neomezeně zkoumat a testovat. Testy se tak dají více přizpůsobit a je menší šance, že opomeneme otestovat důležitou funkcionalitu. Kombinací předchozích dvou principů je **šedá skříňka**, při které je k dipozici specifikace, dokumentace vývojářů a v omezené míře i zdrojový kód, díky kterému sestavíme efektivnější testy.[1]

2.2.2 Statické a dynamické testování

Při statickém testování testujeme něco, co neběží - zkoumaný objekt pouze prohlížíme a kontrolujeme, u dynamického software spustíme a pracujeme s ním.[2]

Těmito způsoby můžeme testovat bílou i černou skříňku. **Statické testování černé skříňky** je vlastně testováním specifikace, kdežto **statické testování bílé skříňky** se zabývá revizí návrhu a architektury kódu. **Dynamické testování černé skříňky** znamená, že testujeme běžící program a pracujeme s ním jako cílový zákazník - testujeme tedy chování softwaru. V případě **dynamického testování bílé skříňky** podle kódu určujeme, co (ne)testovat a jak.[2]

2.2.3 Testy splněním a selháním

Při **testování splněním** kontrolujeme funkčnost softwaru a snažíme se aplikovat co nejjednodušší testové případy. Přesně naopak je to při **testování selháním** - snažíme se chyby záměrně vyvolat a nalézt je pomocí co nejvíce hraničních hodnot.[1]

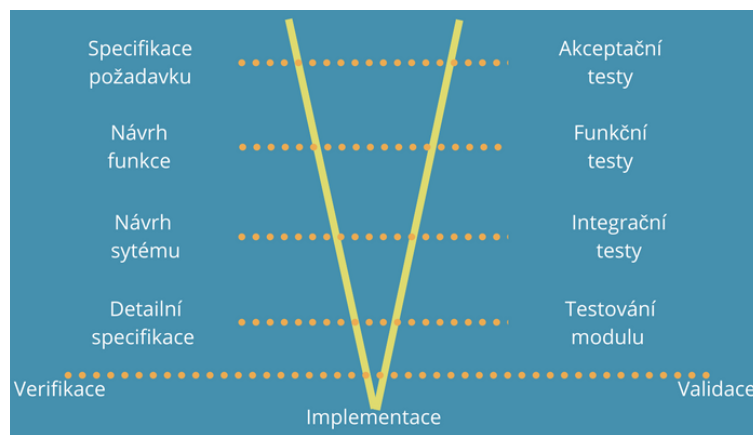
2.2.4 Manuální a automatické testování

Jedná se o testování podle způsobu vyhodnocování jednotlivých testů. Tento způsob rozdělení bude pro tuto práci klíčový, jelikož se v dalších kapitolách zaměříme právě na automatické testování softwaru.

U **manuálního testování** je potřeba osoba, která kontroluje výstupy testů, **automatický test** to zjistí sám a pouze oznámí, zda prošel, či nikoli. Ideální je **kombinovat** oba způsoby testování a manuálními testy se zaměřit především na proměnlivé části aplikace, u kterých by se automatické testy musely příliš často přepisovat. Při manuálních testech se také mnohem jednodušeji zjišťuje postup, kterým došlo k nějaké chybě a lépe se tak odhaluje místo, na kterém vznikla. Jednoznačnou nevýhodou je však zdlouhavost testování a působení lidského faktoru, který nepřináší stoprocentně přesné výsledky.[1] Na automatické testování se více zaměříme v Kapitole 3.

2.3 Úrovně testování

Vývoj je prováděn na různých úrovních a těm pak odpovídá i jejich testování. Existují čtyři základní úrovně, na které lze vývoj, respektive testování, rozdělit. Tyto úrovně jsou znázorněny na Obrázku 2.1 a podrobněji je popíšeme v následujících podkapitolách.



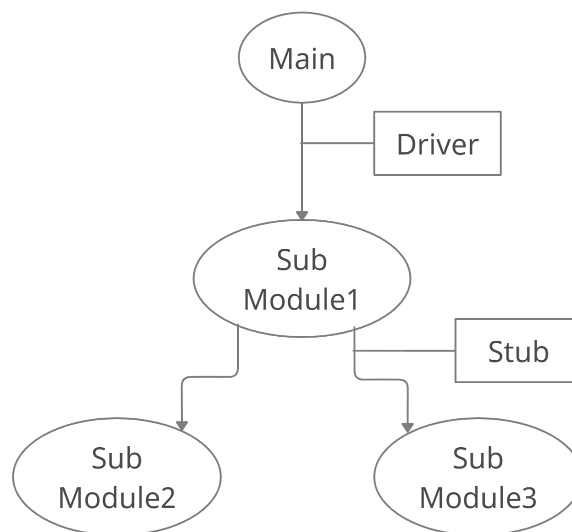
Obrázek 2.1. V model [3]

V levé části **V modelu** jsou fáze vývoje aplikace, v pravé jsou právě jednotlivé úrovně testů ověřující různé fáze vývoje. Principem je testování od malých částí až k větším funkčním celkům, jejich propojení až po testování celé aplikace. Nelze přejít do další úrovně testů, dokud není úspěšně otestována ta předchozí.[3]

2.3.1 Unit testy

Unit testování, někdy také nazýváno jednotkové testování či testování modulu, testuje **nejmenší celky programu**. Jsou jimi například třídy, procedury, funkce nebo metody. Jsou testovány **nezávisle** na sobě a v izolaci. Pro testování tedy nelze využívat ani databázi a testovací data musíme **simulovat**. Takovým simulovaným objektům se říká stubs či mocks (zástupci). Dále jsou využívány drivery, neboli řídící programy, které simulují nadřazené moduly volající moduly testované.[1–2]

Samotný test pak funguje tak, že ověří vstupy, získá výstupy a **porovná je s očekávanými, předem nadefinovanými hodnotami**. Výsledek pak může nabývat dvou hodnot - **prošel či neprošel**.[1]



Obrázek 2.2. Princip unit testu

2.3.2 Integrační testy

Pro integrační testy **seskupujeme moduly** do samostatných systémů, respektive podsystémů. Do daného systému se postupně zařazují správně fungující jednotky, které **nesmí funkčnost systému narušit**.[1]

2.3.3 Systémové testy

Jedná se o poslední fázi testování, ve které je možné odchytit chyby před odevzdáním produktu zákazníkovi. Systémové testy zahrnují například **funkční testy**, které ověřují, zda produkt splňuje specifikované požadavky nebo **testy použitelnosti**, které ověřují přehlednost a komfortnost používání aplikace. Dále třeba **výkonnostní testy**, **testy spolehlivosti** nebo **testy robustnosti**, které ověřují, jak se systém umí vypořádat s chybami.[1]

2.3.4 Akceptační testy

Tyto testy ověřují, že byla splněna zákazníkem předem definovaná **akceptační kritéria**. Ta jsou definována jako měřitelné a ověřitelné podmínky pro přijetí produktu.[1]

2.4 Procesní testování

Pro účely této práce je potřeba věnovat pozornost kategorii procesních testů, jelikož se právě jim budeme v praktické části této práce věnovat.

Procesní testování není testováním návrhu, ale **struktury**. Z toho důvodu je těžké ho zařadit do jedné z úrovní testů, kombinuje totiž vlastnosti více z nich. Předpokládáme znalost strukturovaných informací o chování systému ve formě **cest a rozhodovacích bodů**, které mohou být reprezentovány diagramem. **Očekávaným výsledkem procesních testů je, že je testovací případ spustitelný**. Tím je zkontrolováno, že jsou prováděny jednotlivé akce definované v rámci procesu. Oproti jiným testovacím technikám nemusí být kontrolován výsledek testů, jelikož od procesu kromě jeho spustitelnosti nic neočekáváme.[4]

Kapitola 3
Automatizované testování

V této kapitole už se zaměříme konkrétně na automatizované testy. Popíšeme si jejich princip, výhody a vlastnosti. Dále si představíme různé techniky pro automatizaci testů a postup, jak se na automatizaci připravit. Nakonec popíšeme některé konkrétní nástroje využívané pro automatizované testování v jazyce Java.

3.1 Popis

Při vývoji softwaru je obvykle potřeba víc než jedna iterace testování. Může se stát, že budou do softwaru zavlečeny nové chyby při opravování těch starých, případně jsou dříve nahlášené chyby opraveny nesprávně. Testy se tedy musí opakovat a mohou se stát příliš rutinními a jednotvárnými a tím spíše může docházet k chybnému vyhodnocení testů. Proto se stále populárnějším řešením stává **automatizace testů**. Nemusí a neměly by plně nahradit manuální testování, ovšem mohou dobu testování **výrazně zkrátit a proces zefektivnit a zkvalitnit výstupy**. Na rozdíl od manuálního testování, které musí prováděno osobou u počítače, která ručně provádí všechny kroky testovacích scénářů, automatizované testy k tomu využívají speciální nástroje.[1–2]

Automatizují se především právě opakující se, neboli tzv. regresní testy. Mohou to ale být i testy výkonnostní ověřující fungování při vysoké zátěži systému, nebo funkční, kde se může automatizovat například generování testovacích případů. Velice často se automatizují také jednotkové a integrační testy popsány v Kapitole 2. Nelze automatizovat například testy použitelnosi či podporovatelnosti.[2]

Obecně by se dalo říct, že automatizované testování je jakékoliv testování, k jehož provedení je využíván software. Mezi největší výhody využití softwaru k testování řadíme **snížení nákladů**, **vysoké pokrytí regresními testy**, **vyšší účinnost testů** nebo třeba **přepoužitelnost** některých testů. Tyto výhody shrnují vlastnosti popsané v následující podkapitole.[1]

3.2 Vlastnosti

Patton definuje 4 nejdůležitější vlastnosti automatizace testů:

- **Rychlost.** Oproti manuálnímu testování je zde nepochybně značný rozdíl. Automatizované testy mohou spustit i několik testů v jedné vteřině, což by bylo manuálně prakticky nemožné.
- **Efektivita.** Díky zrychlení celého procesu testování zbývá více času na plánování testů.
- **Správnost a přesnost.** Při ručním zadávání mohou vznikat překlepy a chyby z nepozornosti. To se při automatizovaném testování stát nemůže a výsledky jsou vždy přesné.
- **Neúnavnost.** Testy můžeme spouštět do nekonečna a pokaždé budou dodržovat všechny výhody uvedené výše.[2]

3.3 Porovnání automatizovaného a manuálního testování

3.3.1 Základní popis

Manuální testy jsou vykonávány bez použití jakýchkoliv nástrojů.

Automatizované testy využívají pro vykonání testů software.

3.3.2 Rychlost

Manuální testování je časové náročné, jelikož musí v každém kroku využívat lidské zdroje pro vykonávání testů.

Automatizované testy jsou ze své podstaty mnohem rychlejší.

3.3.3 Lidské zdroje

Manuální testy podléhají závislosti na lidech, kteří musí testy vykonávat.

Automatizované testy jsou tvořeny lidmi, ovšem spouštět a vyhodnocovat je pak může software a lidí je tedy potřeba znatelně méně.

3.3.4 Spolehlivost

Manuální testy jsou vykonávány lidmi, je zde tudíž šance, že člověk udělá chybu a test vyhodnotí nesprávně.

Automatizované testy jsou tzv. neunavitelné, mohou být tedy spouštěny kdykoliv, v jakémkoliv množství a s jakýmkoliv počtem iterací a budou probíhat stále stejně.

3.3.5 Programovatelnost

Manuální testy nelze naprogramovat a využít tak všeho, co dnes nabízí nejrůznější knihovny a nástroje.

Automatizované testy mohou naplno využít všech možností, které knihovny nabízi.

3.4 Techniky

Existují různé techniky pro automatizované testování, které jsou v praxi obvykle kombinovány a není tedy použita pouze jedna technika.

- **Zachycení a přehrávání aktivity uživatele.** Testovací nástroj zachycuje aktivity uživatele provádějícího v rozhraní aplikace jednotlivé testovací případy a následně umožňuje jejich přehrávání. Úspěšnost testu se zjišťuje porovnáním určité proměnné a její očekávané hodnoty.
- **Modifikace vygenerovaných skriptů.** Jedná se opět o zachycení aktivity, ovšem vygenerovaný skript můžeme následně upravovat a dosáhnout tak lepší udržitelnosti a znovupoužitelnosti.
- **Testování řízené daty.** Využívá se v situacích, kdy je potřeba opakovat stejné testy s různými vstupy a výstupy. Vstupní a výstupní data jsou uložena v samostatných souborech a postupně se z nich čtou.
- **Testy řízené klíčovými slovy.** V souborech jsou kromě vstupních a výstupních dat uloženy také příkazy, neboli klíčová slova, která tvoří testovací skript. Ta jsou pak za běhu načítána a prováděna a tím se dynamicky vytváří logika testu.[1]

3.5 Proces automatizovaného testování

Celý proces automatizovaného testování se skládá z 5 kroků:

- Výběr testovacího nástroje.
- Definice rozsahu automatizace.
- Plánování, návrh a vývoj testů.
- Spuštění testů.
- Údržba testů.[5]

3.5.1 Výběr testovacího nástroje

Odvíjí se od toho, jaký software testujeme a zda je s vybraným nástrojem kompatibilní. Nástroj musí samozřejmě vyhovovat především osobě nebo osobám, které ho budou využívat a proto záleží také na skriptovacím jazyce a způsobu testování, který daný nástroj umožňuje.[1]

3.5.2 Definice rozsahu automatizace

Jedná se o výběr oblasti, kterou chceme automatizovat. Existují určitá kritéria, která by se při výběru měla dodržovat:

- **Stabilita.** Mělo by se jednat o testy, které se příliš často nemění.
- **Četnost provádění.** Čím častěji se test provádí, tím větší smysl má jeho automatizace, jelikož ho nemusíme manuálně opakovat stále dokola.
- **Důležitost.** Vybíráme testy s ohledem na to, zda se jedná o kritickou funkcionalitu.
- **Obtížnost provádění.** Je vhodné automatizovat testy s velkým množstvím kroků a operací, velkým objemem dat atd.
- **Časová náročnost.** Čím náročnější test, tím je výhodnější ho automatizovat.
- **Složitost automatizace.**[1]

Některé oblasti naopak nejsou k testování vhodné:

- Testovací případy, které jsou nové a nebyly spuštěny alespoň jednou manuálně.
- Případy, na které se často mění požadavky a jsou na jejich základě často měněny.
- Testy, které jsou prováděny ad-hoc, tedy nejsou strukturovány a plánovány.[5]

3.5.3 Plánování, návrh a vývoj testů

V této fázi dochází k výběru nástrojů pro automatizaci a vhodného frameworku, jsou navrženy testovací scénáře a je určen rozsah automatizace. Je také stanoven plán spouštění testů.

3.5.4 Spuštění testů

Automatizované testy jsou spouštěny a vyhodnocovány.

3.5.5 Údržba testů

S každou změnou software a přidáním nových funkcionalit musí být aktualizovány i testy, aby byly stále efektivní.

3.6 Vybrané nástroje pro automatizaci

Představíme si několik konkrétních nástrojů, které se dají pro automatizované testování využít. Některé z těchto nástrojů budou následně využity v praktické části této práce.

3.6.1 JUnit

Jedná se o open source framework, který je využíván pro psaní **unit testů**. Framework umožňuje **porovnávání očekávaných a reálně získaných výstupů** a není tedy potřeba do testování manuálně zasahovat.[6]

Příkladem může být jednoduchý test ověřující výsledek kalkulačky při sčítání dvou čísel. Kalkulačce řekneme, aby sečetla čísla 3 a 1 a poté porovnáme očekávaný výsledek s výsledkem z kalkulačky. Metoda *assertEquals* přijímá jako první parametr očekávaný výstup a jako druhý parametr reálný výstup testovaného programu. Pokud se hodnoty rovnají, test je vyhodnocen jako úspěšný, v opačném případě se zobrazí chyba *AssertionError*.

```
@Test
void sumTwoNumbers() {
    Calculator calculator = new Calculator();
    calculator.sum(3, 1);
    assertEquals(4, calculator.getResult());
}
```

3.6.2 TestNG

Framework TestNG je velice podobný výše popsanému jUnit. Lehce se liší například v anotaci, jelikož TestNG nabízí její širší využití. Podporuje navíc například *@BeforeTest* a *@AfterTest*, což určí, že metoda s touto anotací se spustí před každým testem s anotací *@Test*. TestNG navíc také umožňuje rozdělovat testovací metody do skupin a testy se mohou spouštět v samostatných a umožní tak paralelní testování, což jUnit nepodporuje.[7]

3.6.3 Mockito

Jedná se o další framework využívány pro psaní unit testů. Principem je vytváření tzv. **mocků**, které jsou **náhradou za reálné objekty**. Při psaní testů totiž často potřebujeme **simulovat** chování reálných objektů, aniž bychom znali jejích implementaci.

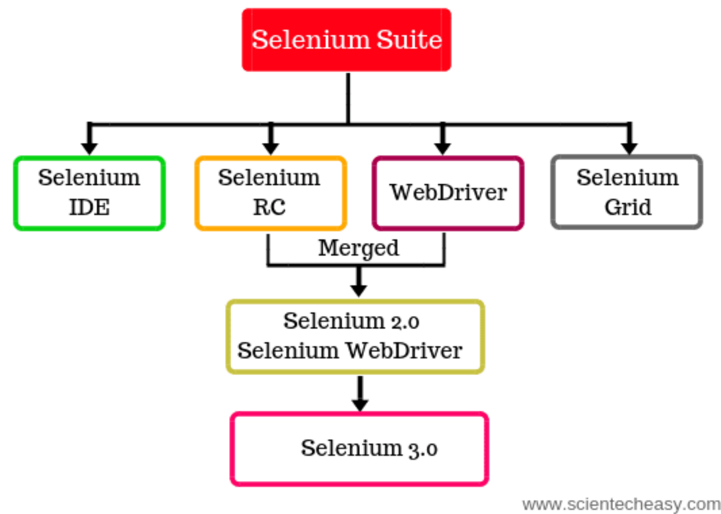
Každý mock je po vytvoření prázdný a můžeme libovolně definovat jeho chování. Pro každý testovací případ se obvykle vytváří nový mock, aby bylo možné pokaždé definovat jiné chování. Mockito se obvykle kombinuje se zmíněným JUnit nebo TestNG.[8]

Znovu použijeme příklad se sčítáním dvou čísel pomocí kalkulačky. Vytvoříme mock pro kalkulačku pomocí anotace *@Mock*. V metodě *sumTwoNumbers()* nejprve řekneme mocku, jak se má chovat při zavolání metody *sum(3, 1)* a poté porovnáme, zda metoda vrací správnou hodnotu.

```
@RunWith(MockitoJUnitRunner.class)
public class CalculatorTest {
    @Mock
    private Calculator calculatorMock;

    @Test
    void sumTwoNumbers() {
        Mockito.when(calculatorMock.sum(eq(3), eq(1))).thenReturn(4);
        assertEquals(4, calculatorMock.sum(3, 1));
    }
}
```

3.6.4 Selenium

Selenium je volně dostupný testovací nástroj pro automatické testování **uživatelského rozhraní webových aplikací**. Jedná se o JavaScriptový framework využívaný k black box testování popsanému v kapitole 2.2.1. Umožňuje vytvářet a automatizovat **testovací skripty pro funkční testy**. Skripty mohou být napsané v různých programovacích jazycích a mohou být spouštěny v různých prohlížečích a na různých platformách. Nástroj se skládá ze čtyř komponent znázorněných na následujícím obrázku.



Obrázek 3.1. Selenium - komponenty [9]

Selenium IDE je pluginem do internetového prohlížeče Mozilla Firefox. Jednotlivé příkazy pro testování se píší ve formátu HTML. Selenium RC umožňuje uživateli výběr programovacího jazyka k definování příkazů a jde použít i na jiných prohlížečích. WebDriver opět umožňuje výběr programovacího jazyka a navíc komunikuje přímo s prohlížečem a pro spuštění testů tedy není potřeba využívat Selenium server. Selenium Grid umožňuje v kombinaci se Selenium RC spouštět testy paralelně napříč různými stroji i prohlížeči.[7]

Kapitola 4
Elektronizace podnikových procesů

Cílem této práce je vytvořit testy pro platformu Camunda BPM, která funguje na principu spouštění procesů. Proto tuto kapitolu zahájíme úvodem do procesního řízení a poté vysvětlíme, co je Camunda, jak funguje a jak s procesy pracuje. Také si přiblížíme, jak se využívá přímo na ČVUT FEL. ■

4.1 Procesní řízení

Procesní řízení, neboli Business Process Management (BPM), je *soubor činností týkajících se plánování, sledování výkonnosti a realizace firemních procesů.*[10]

4.1.1 Podnikový proces

Proces je obecně **soubor činností, které v posloupnosti společně dosahují předem stanoveného cíle**. Podnikový proces je dle Řepy definován jako *souhrn činností, transformujících (pomocí lidí a nástrojů) souhrn vstupů do souhrnu výstupů (zboží nebo služeb), přičemž tyto výstupy jsou určeny pro jiné lidi nebo procesy.*[11]

Skládá se z různých **událostí a aktivit**. Událost nemá žádnou dobu trvání a může jí být například doručení zboží. Ta pak vyvolá aktivitu nebo sérii aktivit, které na událost reagují, v tomto případě třeba kontrola zboží a roztřídění na skladě. Kontrola a třídění jsou aktivity, které trvají určitý čas. Pokud je aktivita jednoduchá činnost, která už nelze rozpadnout na menší logické celky, nazýváme jí úkol (Task).

Součástí typického procesu jsou i **rozhodovací body** (Decision points). Jsou to určité body v čase, ve kterých dochází k rozhodování mezi více možnostmi, kde každá možnost ovlivňuje další postup procesu. Například pokud by po kontrole zboží bylo vyhodnoceno zboží jako závadné, bylo by vráceno zpět dodavateli, v opačném případě je převezeno na sklad k roztřídění.

Dále proces zahrnuje **aktéry**, například osoby, organizace, software jednající jménem osoby nebo organizace. Ti vykonávají jednotlivé činnosti. V neposlední řadě jsou součástí procesu **fyzické i nemateriální objekty**, jako například dokumenty, zboží nebo elektronické nahrávky.

Vykonání procesu vede ke **splnění předem daného cíle**, tedy k určitému výsledku. Ten může být pozitivní a přináší aktérům hodnotu - například zaplacení za dodané zboží a jeho využití. O negativní výsledek se jedná v případě, kdy nepřináší aktérům žádnou hodnotu a v našem příkladu jím může být například vrácení zboží z důvodu dodání špatného zboží nebo nedodání zboží.[12]

4.1.2 Modelování podnikových procesů

Modelování podnikových procesů slouží k **zachycení a optimalizaci existujících procesů**. Nejčastěji se k tomu využívá grafická notace znázorňující jednotlivé procesní diagramy, které obsahují jednotlivé úkoly, přechody mezi nimi, rozhodovací body, aktéry a mnoho dalšího.

Nejzákladnějším způsobem grafického znázornění procesů jsou tzv. **vývojové diagramy** neboli flowcharts. Nejrozšířenější je však v dnešní době notace **BPMN**, konkrétně poslední verze BPMN 2.0, kterou využívá i platforma Camunda a na ní postavená školní aplikace, o které pojednává tato práce.[12]

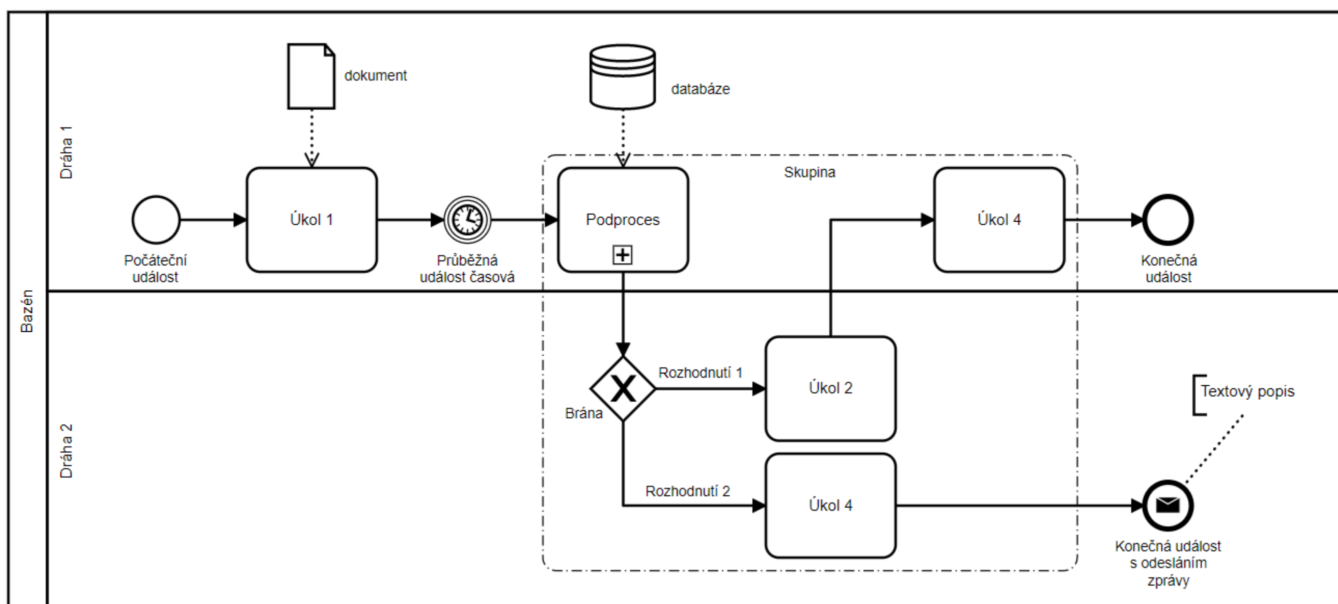
4.2 Notace BPMN

Za vznikem notace BPMN stojí konsorcium Object Management Group (OMG), které udává standardy pro různé technologie. Účelem bylo vyvinout srozumitelnou a přehlednou notaci pro modelování podnikových procesů, se kterou mohou pracovat různé skupiny uživatelů od analytiků až po vývojáře. Od roku 2011 se používá norma BPMN 2.0.

Prvky notace BPMN se dělí do 5 kategorií:

- **Tokové objekty (Flow Objects).**
- **Spojovací objekty (Connecting Objects).**
- **Plavecké dráhy (Swimlanes).**
- **Artefakty (Artifacts).**
- **Data.**

Pro vysvětlení značení byl vytvořen proces na Obrázku 4.1 s popisem jednotlivých prvků, které budou následně vysvětleny. Kromě diagramu lze se soubory formátu *.bpmn pracovat jako s XML dokumenty, čehož využijeme v implementační části této práce.



Obrázek 4.1. Příklad značení procesu v notaci BPMN 2.0

4.2.1 Tokové objekty

Tokové objekty jsou základní grafické prvky - **události** (Events), **aktivity** (Activities) a **brány** (Gateways).

Události nastávají během procesu jako **výsledek nějaké aktivity nebo jsou samy její příčinou**. Jsou znázorněny běžným kolečkem pro počáteční událost, tučným pro kon-

covou událost a dvojitým pro průběžnou událost. Kromě toho mohou být uvnitř kolečka různé symboly, které doplňují událost o její další vlastnosti, jako například doručení/odesílání zprávy nebo čekání určitý časový úsek.

Aktivity jsou **všechny činnosti prováděné během procesu** a dělí se na **atomické úkoly** (Task) nebo složitější **podprocesy** (Subprocess). Jejich grafickou reprezentací je obdélník s textem uvnitř. Podproces je doplněn o symbol plus, který naznačuje, že lze aktivitu rozbalit. Úkol může být doplněn symboly v levém horním rohu, které určují jeho podstatu. Symbol osoby ukazuje, že jde o tzv. uživatelský úkol (User Task), který vykonává osoba manuálně, případně za využití software.

Brány **rozdělují, nebo naopak slučují tok procesů**. Proces tak může například pokračovat jen jednou z několika cest, nebo mu naopak brána dovolí paralelní běh více toků. Jsou značeny kosočtvercem s různými symboly uvnitř. Symbol X označuje, že je brána tzv. exkluzivní a lze tedy vybrat pouze jeden tok procesu, kterým se z brány bude pokračovat.

4.2.2 Spojovací objekty

Slouží k **vzájemnému propojení tokových objektů** a případně dalších objektů. Pomocí šipek také určují pořadí, v jakém na sebe objekty navazují.

4.2.3 Plavecké dráhy

Seskupují prvky do tzv. bazénů (Pools) a drah (Lanes). Bazén reprezentuje jednotlivé **účastníky procesu**, v případě nutnosti rozdělení skupin účastníku na podskupiny se v rámci bazénu zavádí také dráhy.

4.2.4 Artefakty

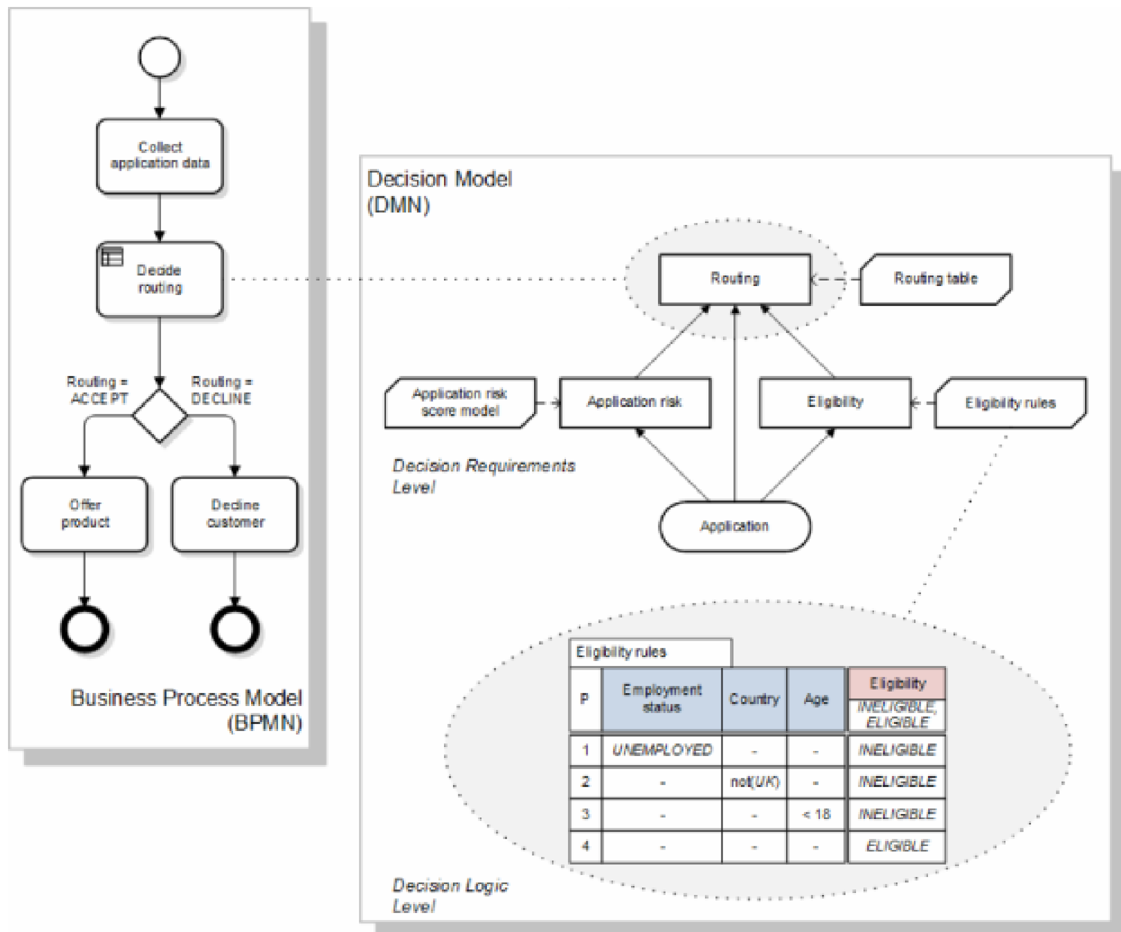
Poskytují dodatečné informace o procesu, typicky pro potřeby doplnění poznámek k procesu.

4.2.5 Data

Jedná se o jakékoliv dodatečné informace, které hrají v procesu důležitou roli. Typicky jde o fyzické objekty, které v procesu figurují nebo další klíčové informace. Příkladem může být znázornění nějakého dokumentu.[13]

4.3 Notace DMN

Stejně jako BPMN, i notace DMN byla vyvinuta v Object Management Group. Jejím účelem je **modelovat rozhodnutí do diagramů a případně je i automatizovat**. Rozhodovací modely lze provázat s modely notace BPMN a zjednodušují rozhodovací logiku v BPMN notaci znázorněnou tzv. branami. Pokud v BPMN přiřadíme úkolu, ve kterém k rozhodnutí dochází, vlastnost **Business Rule Task**, pak zde může být využita právě rozhodovací DMN tabulka. Definice rozhodnutí znázorňuje Decision Requirements Diagram (DRD). Ten určuje, která rozhodnutí mají být učiněna, vztahy mezi nimi a požadavky k jejich uskutečnění. Samotná rozhodnutí jsou učiněna za pomoci rozhodovací logiky, která také umožňuje rozhodnutí automatizovat.[14] Vztah mezi BPMN a DMN je zobrazen v Obrázku 4.2. Je zde znázorněn i zmiňovaný Business Rule Task, ve kterém dochází k využití DMN modelu.

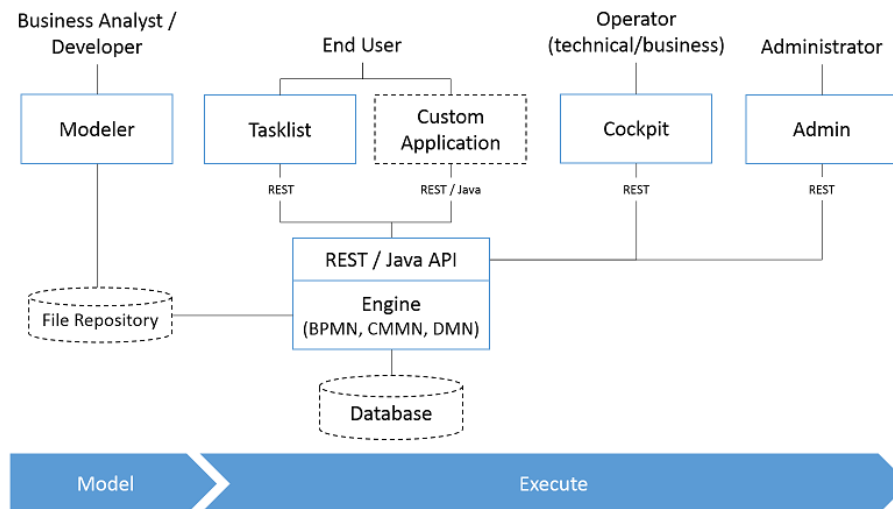


Obrázek 4.2. Vztah BPMN a DMN [14]

4.4 Camunda BPM

Camunda BPM je open-source platforma převážně v jazyce Java. Pomocí desktopové aplikace Camunda Modeler umožňuje namodelování procesů v notaci BPMN 2.0, CMMN 1.1 a DMN 1.1. Modely jsou ukládány do relační databáze jako procesní definice do souborů formátu XML a **při spuštění procesu je z definice vytvořena procesní instance pro daný běh procesu**. S definicemi a instancemi pracuje součást Camundy zvaná Engine. Součástí Camundy jsou i webové aplikace Tasklist, Cockpit a Admin a rozhraní Java API a REST API umožňující využití procesního enginu ve vlastních aplikacích.[15]

Tasklist slouží účastníkům procesu k práci na jim přidělených úkolech, vyplňování s nimi souvisejících formulářů a posouvání procesu dále, typicky tlačítky Odeslat/Zrušit. Další webová aplikace Cockpit umožňuje monitorování a správu nasazených procesních definic a jejich instancí, sledování jejich stavu a jejich případné opravy. V aplikaci Admin se spravují uživatelé, skupiny uživatelů a jejich oprávnění přístupů a zásahů do procesů.[15]



Obrázek 4.3. Architektura platformy Camunda [15]

4.4.1 Využití Camunda BPM na ČVUT FEL

Platforma Camunda BPM je na fakultě využívána k **elektronizaci procesů pro podávání studentských žádostí na studijní oddělení a jejich řízení**. Dříve se tyto žádosti podávaly pouze v papírové podobě a student musel fyzicky zajistit potřebné dokumenty a podpisy od příslušných osob. Automatizace těchto postupů vedla ke vzniku webové aplikace **eProcesy**, skrze kterou studenti žádosti podávají a kterou si fakulta sama implementovala za využití dříve zmíněného REST API. Studenti zde spouštějí nové instance procesů, vyplňují zde formuláře a vidí historii a stav podaných žádostí. V odpovědi na podané žádosti je studijní oddělení a případně další osoby zpracovávají přes webovou aplikaci v rámci fakulty nazývanou jako **Camunda**, která zahrnuje aplikace Tasklist, Cockpit a Admin. Procesní engine Camundy v současnosti zpracovává 11 různých procesů odpovídajících 11 různým žádostem, které mohou studenti podávat. Některé z nich jsou již plně elektronizované a žádosti se tedy dají vyřídit kompletně online, jiné procesy umožňují alespoň generování předvyplněných dokumentů, které se dále zpracovávají již mimo aplikace eProcesy a Camunda.[16–17]

Kapitola 5
Vybraný proces aplikace eProcesy/Camunda

Čtvrtý cíl této práce říká, že má být vybrán jeden proces, který budeme testovat, jelikož u ostatních procesů bude již postup obdobný a na jednom procesu snadno ověříme, zda má automatizace testů smysl. V této kapitole bude tedy vybrán a podrobně popsán zvolený proces.

5.1 Současný stav

Procesy aplikace byly až doposud testována pouze manuálně a jelikož je spuštěna teprve v pilotním provozu, provádějí se v procesech stále změny a ty je potřeba často testovat. Automatizované testy by mohly výrazně pomoci se zkrácením celého testování a pomohly by odhalit další chyby.

Prozatím jsou **plně elektronizovány** 3 procesy, u kterých se vyplatí přemýšlet o automatizovaných testech, ostatní procesy zahrnují jen generování formulářů a obsahují malé množství kroků, nevyplatí se pro ně tedy testy prozatím automatizovat. Jelikož jsou si všechny tři procesy velice podobné, tak zvolíme jeden z nich, provedeme analýzu a následně implementujeme automatické testy. Pro ostatní procesy tak v budoucnu nebude problém přepoužít testy implementované v rámci této práce. Vybraný proces se jmenuje **Žádost o individuální studijní plán** a podrobněji si ho popíšeme v následující podkapitole.[17]

5.2 Popis vybraného procesu

5.2.1 Popis workflow procesu

Proces zobrazený v Příloze A popisuje postup při podávání žádosti o individuální studijní plán (ISP). Student nejprve v aplikaci eProcesy vytvoří žádost se všemi povinnými náležitostmi jako je jméno, bydliště, případné přílohy atd. Následně žádost odešle a ta se přiřadí příslušné referentce dle studentova studijního oboru. Referentka ve webové aplikaci Camunda Tasklist žádost projde a pokud v ní nenajde chyby, vyplní své vyjádření k žádosti a odešle jí dále ke zpracování proděkanovi. V opačném případě ji vrátí studentovi k opravě. Proděkan si opět v této aplikaci žádost zobrazí a buď vydá finální rozhodnutí, nebo ji pošle zpět studentovi nebo referentce k opravě. Po vydání rozhodnutí se žádost naposledy vrací k referentce, která musí již mimo aplikaci Camunda vytvořit rozhodnutí a vytisknout žádost, proděkan vše následně podepíše a referentka odešle studentovi. Poté může celou žádost uzavřít a proces se tak ukončí.[17]

5.2.2 Aktéři

Proces je tvořen jedním bazénem se třemi drahami. Ty reprezentují aktéry procesu:

- **Student.** Iniciuje spuštění procesu, podává žádost na studijní oddělení.

- **Referentka.** Provádí kontrolu a vyhodnocení žádosti, uzavírá celý proces zpracováním žádosti.
- **Proděkan.** Vydává oficiální vyjádření k žádosti.

5.2.3 Prvky procesu

Kromě plaveckých drah a spojovacích objektů proces obsahuje velké množství tokových objektů. Jsou jimi události, aktivity v podobě úkolů a brány.

Události:

- *Požadavek na vytvoření žádosti o Individuální studijní plán (ISP).* Počáteční událost iniciovaná studentem, který žádá o individuální studijní plán.
- *Žádost o ISP zrušena.* Koncová událost, která nastane, když se student rozhodne žádost zrušit.
- *Formulář žádosti o ISP vytištěn na SO a dál bude zpracován v papírové podobě.* Koncová událost, která nastane, když proces proběhne v pořádku a žádost je buď schválena, nebo zamítnuta.

Úkoly:

- *Vyplnění žádosti o ISP a naskenování případné přílohy.* Student vyplní v aplikaci eProcesy formulář žádosti a případně vloží přílohy v povolených formátech. Žádost odešle a v aplikaci může sledovat její stav.
- *Odeslání emailu studentovi.* Obrázek v levém horním roku úkolu znázorňuje, že se jedná o tzv. úkol služby (Service Task). To znamená, že úkol využívá nějaký druh služby, jako je webová služba nebo automatizovaná aplikace. V tomto případě systém automaticky odešle studentovi potvrzovací email o úspěšném doručení žádosti ve chvíli, kdy přijme jeho žádost (tedy ve chvíli, kdy student žádost odešle).
- *Vybrání referentek.* Úkol využívající business engine, který provede jeho vyhodnocení. V tomto případě engine určí, kterou z referentek přiřadit k řešení žádosti na základě studentova studovaného programu a typu studia na fakultě.
- *Kontrola SO.* Vybraná referentka zkontroluje, zda je žádost formálně správně a doplní k ní své vyjádření.
- *Oprava nedostatků.* Student upraví formulář žádosti dle pokynů referentky.
- *Kontrola žádosti a případně rozhodnutí.* Proděkan zkontroluje žádost a vyjádření referenky a na jeho základě učiní rozhodnutí, případně žádost vrátí k přepracování.
- *Úprava vyjádření SO.* Referentka upraví své vyjádření k žádosti.
- *Vygenerování formuláře žádosti.* Systém vygeneruje dokument s oficiální žádostí.
- *Tisk dokumentů, zajištění podpisu a odeslání.* Referentka stáhne vygenerovaný dokument, vytiskne ho, zajistí podpis pověřené osoby a na jeho základě vytvoří rozhodnutí. Tyto dokumenty potom založí do studentovy složky a také je odešle studentovi dopisem. Pokud splní všechny tyto kroky, v aplikaci Camunda žádost uzavře.
- *Odeslání odkazu na rozhodnutí emailem.* Studentovi je po uzavření žádosti systémem automaticky odesláno rozhodnutí o jeho žádosti.

Brány:

- *Podat?* Studentovo rozhodnutí, zda chce žádost odeslat studijnímu oddělení, nebo zrušit.
- *Je žádost formálně správně?* Kontrola studijní referentky, zda je žádost formálně správně podaná. Pokud ano, postupuje ji dále proděkanovi k vyjádření, pokud ne, vrací ji studentovi k opravě.

- *Chce podat i nadále?* Studentovi je žádost vrácena k opravě a může se rozhodnout, zda ji chce i nadále podat, nebo proces zrušit.
- *Je žádost a vyjádření formálně správně?* Zde je žádost kontrolována proděkanem. Může jí vyhodnotit jako správnou a připojit k ní své rozhodnutí. V opačném případě ji vrací k přepracování studentovi nebo referentce.

5.3 Další vlastnosti procesu

Proces je znázorněn v notaci BPMN 2.0, kterou jsme popsali v Kapitole 4.2. Obsahuje i Business Rule Task, u kterého jsme zmínili, že může spolupracovat s DMN diagramem pro rozhodování. Přesně to proces dělá a využívá rozhodovací tabulku pro přiřazení správné referentky nebo skupiny referentek dle studentova studijního programu a formy studia (prezenční vs. dálková). Ukázka z této tabulky se nachází v Příloze F.

Kapitola 6
Příprava na implementaci

V této kapitole popíšeme, jaké způsoby testování budou aplikování a za využití jakých nástrojů proběhnou. Testy zde budou také navrženy tak, aby mohly být na základě této přípravy implementovány. Tato kapitole tedy pokryje první tři kroky testovacího procesu popsané v Kapitole 3.5.

6.1 Výběr testovacího nástroje

Testování procesů jsme již krátce popsali v Kapitole 2.4 a stejný způsob bude využit i v této práci, jelikož se jedná o běžný postup při testování procesů.

Procesy pro aplikace eProcesy a Camunda jsou postaveny jako samostatné **Maven projekty** v jazyce Java. Pro testy tedy využijeme nástroj **JUnit** z Kapitoly 3.6.1, konkrétně verzi JUnit 4. Kromě technologie pro samotné testy využijeme i **Mockito** pro mockování testovaných objektů a **h2 databázi** pro jejich ukládání. Pro testování procesů v Camundě existuje několik knihoven a nástrojů, které budou v praktické části této práce využity:

- **JUnit @Rule**. Anotace @Rule zavádí pravidlo pro testovací třídu. Pro Camundu se využívá k definování procesního enginu ProcessEngineRule, který inicializuje samotný proces.
- **TestCoverageProcessEngineRule.** Pomocí @Rule v kombinaci s @ClassRule definuje procesní engine, ten ale v tomto případě kromě spouštění procesů a přidružených služeb sleduje také pokrytí procesu testy.
- **@Deployment.** Pro nasazení a zrušení nasazení definice testovaného procesu, více procesů či rozhodovacích tabulek. Pro každou testovací metodu je proces při jejím volání nasazen a při dokončení opět smazán a je tedy zaručeno, že pracujeme vždy s původní, nezměněnou definicí procesu či tabulky.
- **Assert.** Knihovna pro porovnávání očekávání od testu s jeho skutečným výsledkem.
- **camunda-bpm-mockito.** Knihovna využívající framework Mockito pro mockování služeb volaných z procesu a mockování procesů samotných.[18–23]

6.2 Definice rozsahu automatizace

Ze zadaných cílů pro tuto práci vyplývá, že má být testován konkrétní proces. V kapitole Kapitole 5 je popsán vybraný proces i důvody pro jeho výběr. Zároveň musíme dodržet **kritéria pro automatizaci** z Kapitoly 3.5.2. U procesu se **nepředpokládá, že by mělo být výrazně zasahováno do jeho toku**, tudíž je to vhodný kandidát pro automatizované testy. Správný průchod procesem je samozřejmě také důležitou součástí aplikace. Vzhledem k velkému počtu kroků, rozhodovacích bodů a účastníků procesu **je i z hlediska obtížnosti vhodný k automatizaci**. Manuální průchod všemi variantami kombinací rozhodnutí je **časově náročný**, tedy proces splňuje i toto kritérium. Z toho vyplývá, že průchod procesem a správné vyhodnocení přidružených úkolů je oblast, na kterou by se automatizované testy měly zaměřit.

6.3 Návrh testů

Při psaní procesních testů pro zvolený proces se zaměříme na 3 oblasti s ohledem na to, že známe strukturu testovaného procesu:

- **Průchody procesem.** Procesy obvykle obsahují body, ve kterých se větví do více cest. Tyto rozhodovací body jsou notaci BPMN značeny jako brány, které přijímají parametr, na základě kterého cestu zvolí. Kombinacemi těchto parametrů pro všechny brány obsažené v procesu chceme dosáhnout co nejvíce kombinací. Kromě toho může mít proces několik různých konců a opět chceme testovat všechny způsoby a kombinace, jak každého konce dosáhnout.
- **Komponenty související s procesem.** Typicky přidružené služby volané z definice procesu. Proces vyvolává například odeslání zpráv nebo vyhodnocení formuláře. Testy kontrolují, zda byly tyto akce volány, ovšem netestují jejich funkčnost, jelikož nepatří přímo do definice procesu a jsou tedy používány jen ve formě mocků.
- **Pokrytí procesu testy.** Tato část souvisí s průchodem procesem. Chceme, aby byl každý prvek navštíven alespoň jednou. Toho samozřejmě nejde docílit při jednom průchodu procesem, pokrytí se tedy počítá z celkového pokrytí celou třídou s testy.

6.3.1 Pokrytí cest procesu

Při definici rozsahu automatizace jsme určili, že je vhodné testovat průchod procesem. Je tedy nutné navrhnout **testovací scénáře**, které budou obsahovat různé průchody procesem a na jejich základě určit hodnoty pro body, ve kterých dochází k rozvětvení procesu dle parametrů.

Existují různé úrovně pokrytí cest v aplikaci, od malého až po velice intenzivní pokrytí testy. Ty jsou určeny **hloubkou pokrytí** (test depth level) značenou N, která nabývá hodnot N = 1, 2, 3 atd. Čím vyšší číslo, tím vyšší hloubka pokrytí. Pro účely této práce byla zvolena hloubka N=3 pro vysokou intenzitu testů. Znamená to tedy, že pro každý bod (reprezentující prvky procesu) je testována kombinace **vstup-výstup-výstup**.[4]

V procesu **identifikujeme body a cesty** a vytvoříme zjednodušený diagram procesu zobrazený v Příloze C. Každý bod označený písmenem reprezentuje prvek procesu, kterým je úkol, nebo brána. Dvojitě označený bod určuje počáteční či koncovou událost. Čísla reprezentují cesty mezi body a šipky určují pořadí bodů. Pro větší přehlednost vznikla tabulka v Příloze D vysvětlující označení každého z prvků, aby později při psaní testů nedošlo k záměně některých prvků. V tabulce jsou také přiřazeny identifikátory ze souboru definice procesu, opět pro lepší orientaci.

Z diagramu vytvoříme **přehled vstupů a výstupů cest z každého bodu**. Tento přehled potom usnadní tvorbu trojic vstup-výstup-výstup. Kombinace do těchto trojic pro pokrytí hloubky N=3 je spolu s přehledem vstupů a výstupů obsažena v Příloze E. Z vytvořených kombinací pak můžeme přistoupit k tvorbě **testovacích scénářů**.

Každý scénář musí začínat **počáteční událostí** a končit **koncovou událostí**. Všechny scénáře tedy vycházejí z bodu A a končí v bodě D, L, nebo R. Cílem je vytvořit **co nejméně co nejjednodušších průchodů**. Tedy tak, aby se využily všechny kombinace hloubky N=3, ale aby se scénář zbytečně nenatahoval, pokud je již kombinace pokrytá jiným scénářem. Tímto způsobem definujeme 6 scénářů TS1 až TS6 a každému přiřazujeme pro identifikaci barvu, která pak bude klíčová při psaní testů, jelikož bude rozlišovat v názvech testů jednotlivé scénáře. Přehled scénářů je patrný z následující tabulky:

Testovací sekvence			
	Test	Sekvence kroků	Sekvence prvků
N=3	1	1–2–3	A–B–C–D
	2	1–2–4–5–6–7–8–10–12	A–B–C–E–F–G–H–I–K–L
	3	1–2–4–5–6–7–9–13–14–17–13–16–10–12	A–B–C–E–F–G–H–J–M–N–J–M–I–K–L
	4	1–2–4–5–6–7–8–10–11–7–8–10–12	A–B–C–E–F–G–H–I–K–G–H–I–K–L
	5	1–2–4–5–6–7–8–10–11–7–9–13–15–18–19–20	A–B–C–E–F–G–H–I–K–G–H–J–M–O–P–Q–R
	6	1–2–4–5–6–7–9–13–16–10–11–7–9–13–14–17–13–14–17–13–15–18–19–20	A–B–C–E–F–G–H–J–M–I–K–G–H–J–M–N–J–M–N–J–M–O–P–Q–R

Obrázek 6.1.

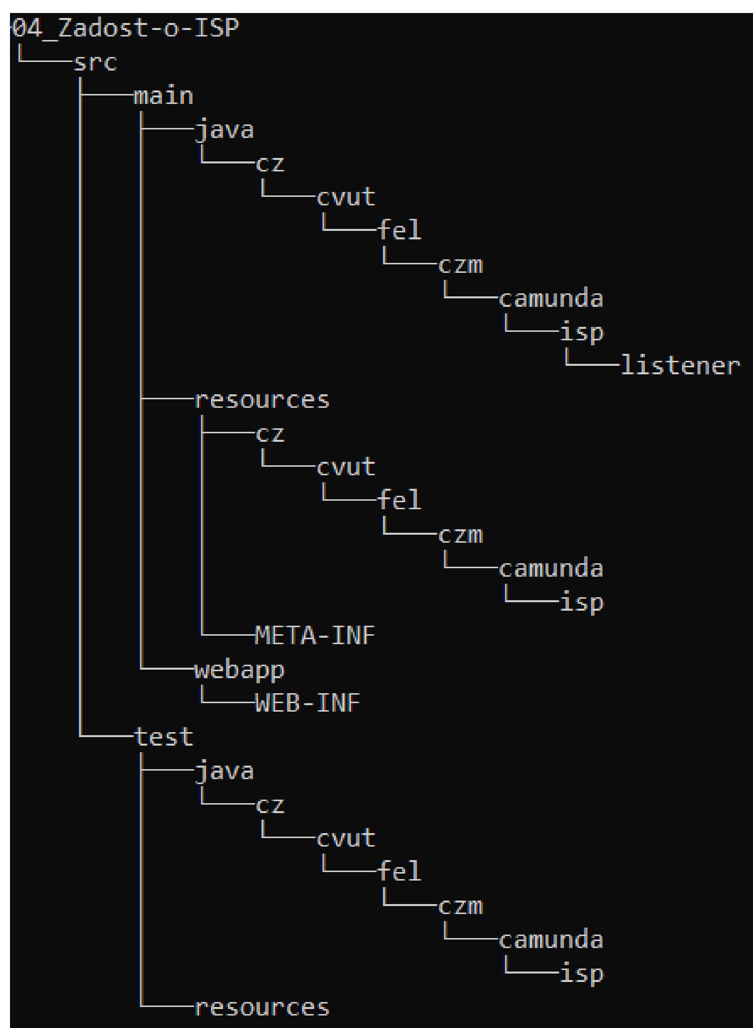
Kapitola 7
Implementace procesních testů

V této kapitole popíšeme postup implementace procesních testů, které jsme definovali v předchozí kapitole.

7.1 Implementace testů

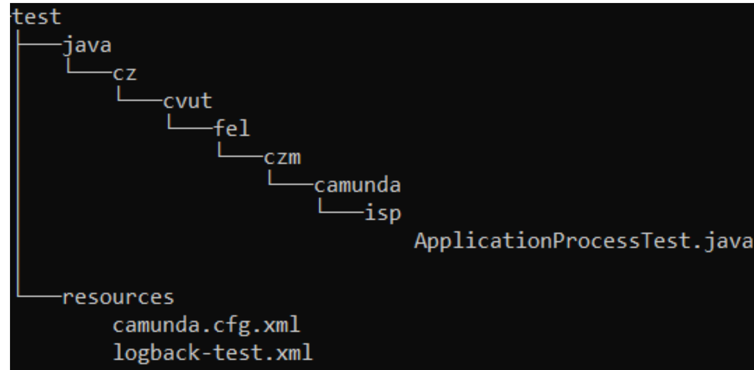
7.1.1 Příprava prostředí

Jak bylo již zmíněno, každému procesu přísluší Maven projekt. Projekt pro proces Žádost o individuální studijní plán má následující strukturu:



Obrázek 7.1. Struktura projektu

Složka *main* a všechny její podložky a soubory jsou již existující soubory, složka test byla vytvořeny pro účely psaní procesních testů. Kromě samotné testovací třídy *ApplicationProcessTest* musí testy pro Camundu obsahovat i **konfigurační soubor** *camunda.cfg.xml*, ve kterém je konfigurace pro procesní engine pracující s procesy.



Obrázek 7.2. Struktura a soubory pro testování

Důležitou součástí každého Maven projektu je soubor *pom.xml*, ve kterém jsou definovány tzv. **dependencies**. Jsou to všechny moduly a konkrétní knihovny využívané v projektu. Pro využití JUnit, Mockita a porovnání výsledků potřebujeme následující:[18–23]

```xml
<dependency>
    <groupId>org.camunda.bpm.extension</groupId>
    <artifactId>camunda-bpm-assert-scenario</artifactId>
    <version>1.0.0</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.camunda.bpm.extension</groupId>
    <artifactId>camunda-bpm-process-test-coverage</artifactId>
    <version>0.3.2</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.camunda.bpm.extension.mockito</groupId>
    <artifactId>camunda-bpm-mockito</artifactId>
    <scope>test</scope>
    <version>4.13.0</version>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Obrázek 7.3. Dependencies v pom.xml

7.1.2 Příprava testovací třídy

Nyní již můžeme přistoupit k přípravě testovací třídy *ApplicationProcessTest.java*. Definici třídy předchází anotace *@Deployment* pro **nasazení testovaného procesu** *SO_Zadost_ISP.bpmn* a rozhodovacích tabulek *referent.dmn* a *videdeanDecision.dmn*.

```
@Deployment(resources = {"SO_Zadost_ISP.bpmn",
        "referent.dmn",
        "vicedeanDecision.dmn"})
@RequiredHistoryLevel("full")
public class ApplicationProcessTest {
```

Obrázek 7.4. Třída ApplicationProcessTest

Spuštění **procesního enginu** a spuštění kontroly pokrytí procesu testy definujeme pomocí *@Rule*. Požadované pokrytí testy nastavíme na 90 %, ovšem jen z důvodu, že knihovna obsahuje chybu, která špatně vyhodnocuje pokrytí a není tak možné dosáhnout stoprocentního pokrytí. Doopravdy ale cílíme na 100%.[22]

```
@Rule
@ClassRule
public static TestCoverageProcessEngineRule rule = TestCoverageProcessEngineRuleBuilder.create()
        .assertClassCoverageAtLeast(0.9)
        .build();
```

Obrázek 7.5. Definice procesního enginu

Vytvoříme přehled všech **proměnných**, které budeme v testech využívat. Jedná se o jednotlivé prvky procesu, ve kterém má každý prvek přiřazené unikátní ID. Pro lepší orientaci v nich si je zde definujeme. Tento přehled je zobrazený v Příloze G.

Anotací *@Mock* namockujeme scénář procesu, který poté budeme pro každou metodu spouštět a dále všechny služby volané z definice procesu a v metodě *setup()* mocky zaregistrujeme.

```
@Mock
private ProcessScenario applicationProcess;

@Mock
private MyBean myBean;

@Mock
private NotificationListener notificationListener;
```

Obrázek 7.6. Využití @Mock

```
MockitoAnnotations.initMocks( testClass: this);
Mocks.register("myBean", myBean);
Mocks.register("username", VAR_STUDENT_ASSIGNEE);
Mocks.register("notificationListener", notificationListener);
```

Obrázek 7.7. Registrace mocků

Metoda *setup()* je označena *@Before*, což v JUnit znamená, že bude volána před spuštěním každého testu a tedy zde nastavíme vše, co budeme v testech využívat. Pro každý úkol z procesu je zde nastaveno jeho **výchozí chování** a pro jednotlivé rozhodovací body jsou zde vytvořené seznamy možností.[18–23]

```
@Before
public void setup() {
    MockitoAnnotations.initMocks( testClass: this);
    Mocks.register("myBean", myBean);
    Mocks.register("username", VAR_STUDENT_ASSIGNEE);
    Mocks.register("notificationListener", notificationListener);

    actions.addAll(Arrays.asList("approved", "disapproved", "backToReferent", "backToStudent"));
    continueOptions.addAll(Arrays.asList("yes", "no"));

    studyInfoVariables = Variables.createVariables()
            .putValue(VAR_PROGRAM_ID,  value: "programmes/BN1/")
            .putValue(VAR_STUDY_FORM,  value: "FULLTIME");

    when(applicationProcess.waitsAtUserTask(USER_TASK_SUBMIT_APPLICATION))
            .thenReturn(task -> {
                assertThat(task).isAssignedTo(VAR_STUDENT_ASSIGNEE);
                task.complete(withVariables(VAR_CONTINUE, continueOptions.get(0)));
            });
    when(applicationProcess.waitsAtServiceTask(SERVICE_TASK_SEND_EMAIL_APPLICATION_RECEIVED))
            .thenReturn(new ServiceTaskAction() {
                @Override
                public void execute(ExternalTaskDelegate externalTask) {
                    externalTask.complete();
                }
            });
    when(applicationProcess.waitsAtBusinessRuleTask(BUSINESS_RULE_TASK_CHOOSE_REFERENT))
            .thenReturn(new BusinessRuleTaskAction() {
                @Override
                public void execute(ExternalTaskDelegate externalTask) {
                    externalTask.complete();
                }
            });
```

Obrázek 7.8. Nastavení výchozího chování úkolů

7.1.3 Důležité části testů

Celkem vzniklo 19 testů, které ověřují průchody procesem dle testovacích scénářů a dále správnou práci s proměnnými v rozhodovacích bodech a s přidruženými událostmi. Přehled testů je zobrazen v Příloze H.

Každý test začíná anotací *@Test* a následuje samotná metoda. V rámci každého testu musí dojít ke **spuštění scénáře**. V metodě *run()* je předán namockovaný scénář. Pomocí *startByKey()* je určen bod, ze kterého má scénář začít, pro všechny testy to bude počáteční událost procesu. Metoda *execute()* pak scénář spustí.

Výchozí chování jednotlivých úkolů je definováno v metodě *setup()* metodou *when()*, ovšem v každém testu můžeme toto chování přepsat tak, aby průchod procesem postupoval požadovanou cestou.

```
Scenario scenario = Scenario.run(applicationProcess)
        .startByKey(PROCESS_KEY, studyInfoVariables)
        .execute();
```

Obrázek 7.9. Spouštění testovaného scénáře

Pro ověřování správného chování procesu a jeho proměnných využíváme metody *assertThat()* a *verify()*. Ověřujeme, zda scénář obsahuje správně definované proměnné z rozhodovacích bodů. Dále sledujeme, zda scénář postupovat správnou cestou a to tak, že zkoumáme, které z úkolů byly splněny, které se nikdy neuskutečnily a kterou koncovou událostí scénář skončil.

```
assertThat(scenario.instance(applicationProcess)).variables()
        .containsEntry(VAR_CONTINUE, continueOptions.get(1))
        .doesNotContainKeys(VAR_FORMALLY_CORRECT, VAR_ACTION);
verify(applicationProcess, never())
        .hasCompleted(SERVICE_TASK_SEND_EMAIL_APPLICATION_RECEIVED);
verify(applicationProcess).hasStarted(START_EVENT);
verify(applicationProcess).hasFinished(END_EVENT_APPLICATION_CANCELLED);
```

Obrázek 7.10. Metody pro ověřování průchodu procesem

Příkladem jednoduchého průchodu procesem je testování tzv. **happy path**. Jedná se o **výchozí průchod procesem**, tedy ideální případ, kdy je proces zahájen, všechny úkoly bez problémů splněny a nejjednodušší cestou se dostane až ke koncové události, která v našem případě zaznamenává úspěšné vyhodnocení žádosti. Takový průchod procesem je zobrazen v Příloze B a odpovídá mu test z Obrázku 7.11. Happy path kopíruje výchozí nastavení úkolů z metody setup() a není tedy potřeba toto chování předefinovávat.

```
@Test
public void shouldExecuteHappyPath() {

    Scenario scenario = Scenario.run(applicationProcess)
            .startByKey(PROCESS_KEY, studyInfoVariables)
            .execute();

    assertThat(scenario.instance(applicationProcess)).variables()
            .containsEntry(VAR_CONTINUE, continueOptions.get(0))
            .containsEntry(VAR_FORMALLY_CORRECT, true)
            .containsEntry(VAR_ACTION, actions.get(0));
    verify(myBean, (times( wantedNumberOfInvocations: 1))).getTaskReferentNameForVicedean();
    verify(applicationProcess, never())
            .hasCompleted(USER_TASK_APPLICATION_REVIEW);
    verify(applicationProcess, never())
            .hasCompleted(USER_TASK_REFERENT_REVIEW);
    verify(applicationProcess).hasFinished(END_EVENT_APPLICATION_COMPLETED);
}
```

Obrázek 7.11. Test ověřující průchod happy path

Ve vyhodnocení testu v Příloze I je zobrazena i posloupnost všech spuštěných úkolů v rámci scénáře a je tedy viditelné, že úkoly odpovídají požadovanému scénáři. Kladné vyhodnocení testu také ukazuje, že ověřování pomocí *assertThat()* a *verify()* bylo úspěšné.

V průchodu červeným scénářem v testu *executeRedScenario()* musí naopak dojít k předefinování chování úkolu. Scénář totiž předpokládá, že student zahájí proces, začne s vyplňováním formuláře, ale rozhodne se žádost neodeslat a zruší ji. Proces tak končí událostí Žádost o ISP zrušena.[18–23]

```
@Test
public void shouldExecuteRedScenario() {
    when(applicationProcess.waitsAtUserTask(USER_TASK_SUBMIT_APPLICATION))
            .thenReturn(task -> {
                task.complete(withVariables(VAR_CONTINUE, continueOptions.get(1)));
            });

    Scenario scenario = Scenario.run(applicationProcess)
            .startByKey(PROCESS_KEY)
            .execute();

    assertThat(scenario.instance(applicationProcess)).variables()
            .containsEntry(VAR_CONTINUE, continueOptions.get(1))
            .doesNotContainKeys(VAR_FORMALLY_CORRECT, VAR_ACTION);
    verify(applicationProcess, never())
            .hasCompleted(SERVICE_TASK_SEND_EMAIL_APPLICATION_RECEIVED);
    verify(applicationProcess).hasFinished(END_EVENT_APPLICATION_CANCELLED);
}
```

Obrázek 7.12.

7.1.4 Popis jednotlivých testů

Pro ověřování průchodů procesem na základě testovacích scénářů kombinací kroků o hloubce N=3 popsaných v přechozí Kapitole 6.3.1 slouží tyto testy:

- **shouldExecuteHappyPath()**. Scénář projde happy path procesu.
- **shouldEndWithCancellation()**. Ověřuje, zda scénář skončí událostí Zrušení žádosti.
- **shouldEndWithCancellationAfterReview()**. Podobně jako předchozí test končí zrušením, ovšem až po kontrole a vrácení k přepracování ze strany referentky.
- **shouldExecuteRedScenario()**. Ověřuje testovací scénář TS1.
- **shouldExecuteBlueScenario()**. Ověřuje testovací scénář TS2.
- **shouldExecuteGreenScenario()**. Ověřuje testovací scénář TS3.
- **shouldExecuteYellowScenario()**. Ověřuje testovací scénář TS4.
- **shouldExecutePinkScenario()**. Ověřuje testovací scénář TS5.
- **shouldExecutePurpleScenario()**. Ověřuje testovací scénář TS6.
- **shouldExecuteVicedeanDecisionTwice()**. Testuje zda úkol pro kontrolu žádosti proděkanem proběhne víckrát na základě opakovaného vracení žádosti k přepracování.

Dále je potřeba otestovat rozhodovací tabulku *referent.dmn*. Ta je využívána v byznys rule úkolu pro výběr referentky. Na základě studovaného programu a formě studia je k řešení žádosti přiřazená konkrétní referentka či skupina referentek. V několika testech ověříme, zda tabulka z Přílohy F) přiřazuje správné výsledky:

- **shouldContainStudyInfo()**. Ověřuje, zda proces přijímá informace o studentovi.
- **shouldGotAssignedReferentGroup()**. Ověřuje, zda je k vyřízení žádosti přidělena správná studijní referentka.
- **shouldGotAssignedDifferentReferentGroup()**. Ověřuje, zda je k vyřízení žádosti přidělena správná studijní referentka.
- **shouldGotAssignedDefaultReferentGroup()**. Jsou zadány takové informace o studentovi, které v tabulce nejsou obsaženy a měla by tak být zvolena výchozí možnost výběru referentky.

```
@Test
public void shouldGotAssignedReferentGroup() {

    Scenario scenario = Scenario.run(applicationProcess)
            .startByKey(PROCESS_KEY, studyInfoVariables)
            .execute();

    assertThat(scenario.instance(applicationProcess)).variables()
            .containsEntry(VAR_REFERENT_GROUP, "13922-all");

    verify(applicationProcess)
            .hasFinished(END_EVENT_APPLICATION_COMPLETED);
}
```

Obrázek 7.13. Test pro ověření DMN tabulky

Samostatně potřebujeme otestovat přidružené časové události u úkolů *Kontrola SO*, *Úprava vyjádření SO*, *Kontrola a případně rozhodnutí* a *Tisk dokumentů, zajištění podpisů a odeslání*. Tyto události zajišťují, že pokud není úkol splněn v nějakém časovém limitu, je vyvolána akce přidružená k této události. Všechny 4 události mají nastavený časový limit 2 dny. pokud do té doby referentka nebo proděkan nesplní úkol, je jim aplikací odeslán email s upozorněním. Testujeme, zda je událost po dvou dnech opravdu vyvolána a spustí tak službu *notificationListener*. Testy provádíme pro všechny události a pro různě dlouhé časové úseky:

- **shouldSendReminderPrintTaskIsLate()**. Ověřuje, zda byla odeslána notifikace o nesplnění úkolu. Časový posun je nastaven na 2 dny a 1 minutu.
- **shouldSendReminderReferentReviewTaskIsVeryLate()**. Úkol má zpoždění 3 dny, měla by tedy být opět odeslána notifikace.
- **shouldNotSendReminderVicedeanDecideTaskIsLate()**. Úkol má zpoždění 1 den, notofikace tedy není odeslána.
- **shouldSendReminderReferentCheckTaskIsVeryLate()**. Ověřuje, zda při zpoždění 4 dny opět odešle notifikaci.

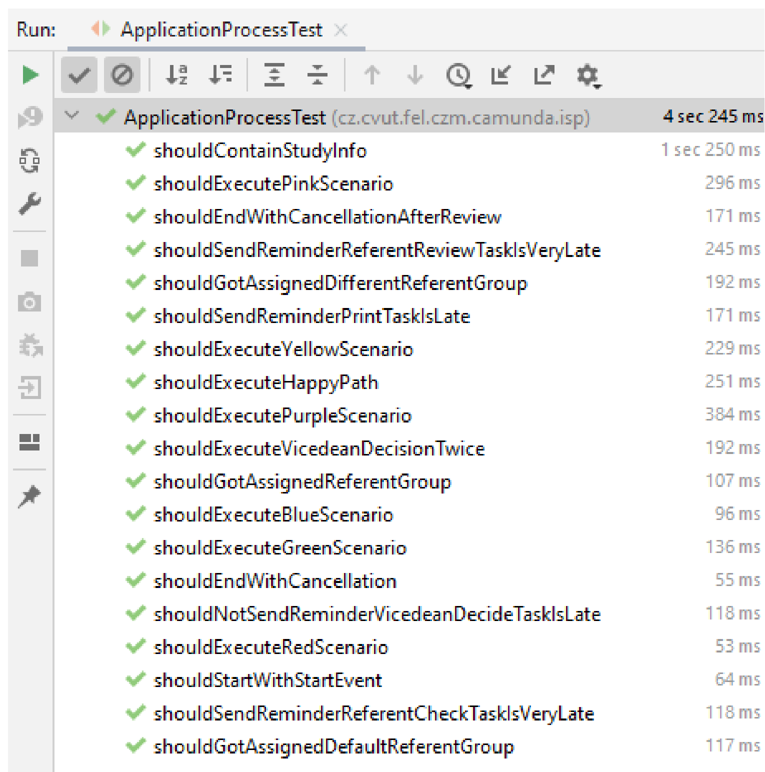
```
@Test
public void shouldSendReminderPrintTaskIsLate() {
    when(applicationProcess.waitsAtUserTask(USER_TASK_PRINT_FORM))
            .thenReturn(task -> {
                task.defer( period: "P2DT1M", task::complete);
            });

    Scenario.run(applicationProcess)
            .startByKey(PROCESS_KEY, studyInfoVariables)
            .execute();

    verify(applicationProcess, times( wantedNumberOfInvocations: 1))
            .hasCompleted(USER_TASK_PRINT_FORM);
    verify(applicationProcess, times( wantedNumberOfInvocations: 1))
            .hasCompleted(BOUNDRARY_EVENT_TIMER_PRINT);
    verify(applicationProcess)
            .hasFinished(END_EVENT_APPLICATION_COMPLETED);
}
```

Obrázek 7.14. Test pro ověření spuštění přidružené časové události

Poslední testovat oblastní je již zmiňované pokrytí procesu testy. Pravidlo *TestCoverageProcessEngineRule* vyhodnotí pokrytí pro každý z testů a na závěr udělá celkové vyhodnocení pokrytí celou testovací třídou.[18–23]

7.2 Spuštění testů

Všechny testy byly spuštěny a kladně vyhodnoceny.



Obrázek 7.15. Vyhodnocení testů

Pokrytí testy bylo procesním enginem také vyhodnoceno jako dostatečné.

```
srp 11, 2021 2:25:31 ODP. org.camunda.bpm.extension.process_test_coverage.junit.rules.TestCoverageProcessEngineRule succeeded
INFO: cz.cvut.fel.czm.camunda.isp.ApplicationProcessTest succeeded.


srp 11, 2021 2:25:31 ODP. org.camunda.bpm.extension.process_test_coverage.junit.rules.TestCoverageProcessEngineRule handleClassCoverage
INFO: cz.cvut.fel.czm.camunda.isp.ApplicationProcessTest test class coverage is: 0.9761904761904762
```

Obrázek 7.16. Vyhodnocení pokrytí procesu testy

7.3 Údržba testů

Poslední krok procesu automatizace testů nebude v rámci této práce uskutečněn.

7.4 Vyhodnocení testů a doporučení pro další implementaci

V rámci práce jsme otestovali proces Žádosti o individuální studijní plán. Zvolili jsme **procesní testy** ověřující možné průchody procesem a namodelovali jsme situace, během

kterých by mohlo dojít k výskytu chyb. Samotné testování nezaručuje, že je proces zcela bezchybný, jelikož pomocí testování je možné pouze ověřit přítomnost chyb, ale ne jejich nepřítomnost. Všech 19 implementovaných testy byly vyhodnoceny jako **úspěšné**, z toho důvodu lze považovat celé testování za úspěšné. Primárním cílem procesních testů je na základě struktury procesu ověřit, zda je proces spustitelný. Jelikož v každém testu kontrolujeme, kterými body a kolikrát scénář prošel a jakých hodnot v klíčových bodech nabýval, můžeme říct, že jsme testování průchodu procesu do jisté míry automatizovali.

Během testování dalších procesů již není nutné procházet procesem automatizace od začátku, jelikož první 2 kroky zůstanou stejné, pokud nebude rozhodnuto, že musí být například využity konkrétní nástroje. Ve třetím kroku, tedy návrhu testů, se však již změnám nevyhneme. Je potřeba opět **zmapovat všechny prvky procesu a zvolit vhodnou hloubku pokrytí** v závislosti na velikosti a složitosti procesu. U složitějších procesů s více možnými konci a více rozhodovacími bodů, je vhodné, stejně jako v této práci, zvolit hloubku pokrytí N=3. U jednodušších procesů s jedním, maximálně dvěma konci a malým množství bran doporučuji zvolit hloubku pokrytí N=2. U velice jednoduchých procesů, kde je jeden až dva rozhodovací body a jeden koncový bod doporučuji využít hloubku pokrytí N=1. Z těchto kombinací pak budou vytvořeny **testovací scénáře**, na jejichž základě budou sestaveny testy ověřující správné průchody procesem.

Musí být **identifikovány a testovány všechny přidružené události**, jako byla v našem případě časová událost pro zasílání připomínek. Je také potřeba zmapovat všechny **služby**, které proces ke svému vykonávání využívá. Otestovány musí být opět i **DMN rozhodovací tabulky**. Toto jsou kritická místa v procesech, ve kterých by mohlo dojít k chybám. Kromě toho je výhodné držet si přehled o procentuálním pokrytí procesu testy.

Nejprve je tedy potřeba proces projít a najít rozhodovací body. Po jejich nalezení a rozdělení procesu na jednotlivé větve rozhodnout, kterou hloubku testování zvolit a vytvořit příslušné testovací scénáře. Následně pomocí testů ověřit průchody pomocí vytvořených scénářů. Následně otestovat jednotlivé rozhodovací body samostatně. Tím by měl být celý proces otestován a riziko vzniku chyb minimalizováno.

Během testování jsme zjistili, že kód implementující volané služby není možné testovat. Nebylo možné ověřit, že metody dělají to, co mají, jelikož **nevrací žádnou návratovou hodnotu** nebo pracují s daty, ke kterým **nemáme přístup**. Nebylo tak možné plně ověřit, že se konkrétní metody ve službách volají se správnými parametry a zda končí úspěchem. Během dalšího vývoje tedy doporučuji doplnit návratové hodnoty a rozhodovací parametry jako argumenty u jednotlivých metod, se kterými procesy pracují. Čistota kódu výrazně komplikovala testovatelnost celého procesu. Kromě toho zcela schází jakákoliv dokumentace a komentáře v kódu.

Kapitola 8
Závěr

Cílem této práce bylo zanalyzovat a implementovat automatizované testy pro ověření fungování vybraného procesu aplikací eProcesy/Camunda funngujících na platformě Camunda BPM.

V Kapitole 2 jsme představili možnosti testování software a popsali úrovně testů. Poprvé zde bylo zmíněno i procesní testování.

V následující Kapitole 3 jsme se zaměřili na automatizaci testů, proces automatizace a vybrané nástroje k automatizovaným testům využívané. Představili jsme zde důvody, proč testy automatizovat a pravidla, kterými bychom se měli při výběru testů k automatizaci řídit.

V Kapitole 4 jsme popsali platformu Camunda BPM. Tomu ale předcházela definice pojmů proces a procesní řízení, s nimiž Camunda pracuje a bez jejichž znalosti bychom se neobešli. Byla zde představena notace BPMN 2.0 pro modelování podnikových procesů a byly popsány jednotlivé prvky, které tato notace využívá. Vedle BPMN jsme si představili i notaci DMN pro řízení rozhodování. Se znalostí procesů a platformy Camunda BPM jsme pak mohli popsat, jak se využívá na Fakultě elektrotechnické v aplikaci eProcesy/Camunda.

Na to přímo navazuje Kapitola 5, která podrobně popisuje výběr procesu k návrhu a implementaci testů a vysvětluje jednotlivé prvky procesu.

Šestá Kapitola 6 se zabývá prvními třemi kroky automatizace testů. V té jsme zvolili způsob, kterým se bude aplikace testovat. Jelikož cílem bylo testovat konkrétní proces, bylo zvoleno procesní testování vycházející ze znalosti struktury procesu. To v našem případě platí, protože máme k dispozici BPMN a DMN modely vybraného procesu Žádost o individuální studijní plán. Na základě kombinací vstupů a výstupů hloubky N=3 v jednotlivých úkolech procesu byly tedy navrženy testovací scénáře na průchody procesem. Kromě samotných průchodů byly identifikovány i další body, které je potřeba testovat. Jsou jimi přidružené časové události zasílající notifikaci pro zpoždění při plnění úkolu, body, ve kterých dochází k větvení procesu na základě určitých parametrů a tzv. Business Rule Tasks (úkoly), ke kterým mohou být přiřazeny rozhodovací DMN tabulky. Pokud se tak stane, je potřeba ověřit, že tabulka pracuje správně a na základě vložených vstupů vrací i odpovídající hodnoty na výstupu. Tím také ověříme společné fungování BPMN modelu s DMN tabulkou.

Následující kapitola pokryla čtvrtý bod automatizace testů, tedy jejich implementaci. Bylo vytvořeno 19 testů, které testují průchody procesem na základě testovacích scénářů a identifikovaná kritická místa, jako je tabulka pro přiřazování referentek, či časová událost odesílající notifikaci. Údržbou těchto testů se tato práce nebude zabývat. Tím jsme tedy dokončili proces automatizace testů a tyto testy byly vyhodnoceny. Na základě zkušeností a znalostí získaných při psaní této práce byl doporučen postup pro testování dalších procesů a pro úpravy kódu aplikace.

Testování ukázalo, že proces funguje správně a korektně také spolupracuje s rozhodovacími tabulkami a službami. Práce by se tak dala považovat za úspěšnou, zároveň totiž splnila stanovené dílčí cíle popsané v úvodu.

Literatura

[1] Petr ROUDENSKÝ a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

[2] Ron PATTON. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80-7226-636-5.

[3] *iQuest blog*.
http://blog.iquest.cz/2017/07/metody-vyvoje-aplikaci-waterfall-v.html. Navštíveno: 8.4.2021.

[4] *TMAP: Process Cycle Test (PCT)*.
https://www.tmap.net/wiki/paths. Navštíveno: 6.8.2021.

[5] *Guru99: Automation Testing Tutorial: What is Automated Testing?*
https://www.guru99.com/automation-testing.html. Navštíveno: 5.7.2021.

[6] *Tutorialspoint*.
https://www.tutorialspoint.com/junit/junit_quick_guide.htm. Navštíveno: 4.4.2021.

[7] *Guru99*.
https://www.guru99.com/junit-tutorial.html/. Navštíveno: 8.4.2021.

[8] *Voho*.
http://voho.eu/wiki/mockito/. Navštíveno: 10.7.2021.

[9] *Scientech Easy*.
https://www.scientecheasy.com/2018/11/selenium-tutorial.html. Navštíveno: 10.5.2021.

[10] *Trask solutions a.s.*
https://www.trask.cz/publikace/nizkonakladove-bpm. Navštíveno: 13.7.2021.

[11] Václav ŘEPA. *Podnikové procesy, Procesní řízení a modelování*. Praha: Grada Publishing a.s., 2007. ISBN 978-80-247-2252-8.

[12] Marlon DUMAS, Marcello La ROSA, Jan MENDLING a Hajo A. REIJERS. *Fundamentals of Business Process Management*. Springer, 2013. ISBN 978-3-642-33142-8.

[13] *OMG Specification of Business Process Model and Notation*.
https://www.omg.org/spec/BPMN/. Navštíveno: 20.7.2021.

[14] *OMG Specification of Decision Model and Notation*.
https://www.omg.org/spec/DMN/. Navštíveno: 20.7.2021.

[15] *Camunda*.
https://docs.camunda.org/manual/7.7/introduction/. Navštíveno: 15.7.2021.

[16] *Aplikace eProcesy*.
https://fel.cvut.cz/eprocesy/.

[17] *Aplikace Camunda*.
https://fel.cvut.cz/camunda/.

[18] *Camunda Specification: Testing Process Definitions*.
https://camunda.com/best-practices/testing-process-definitions/. Navštíveno: 27.7.2021.

[19] *Camunda Blog: Testing Entire Process Paths*.
https://camunda.com/blog/2020/10/testing-entire-process-paths/. Navštíveno: 27.7.2021.

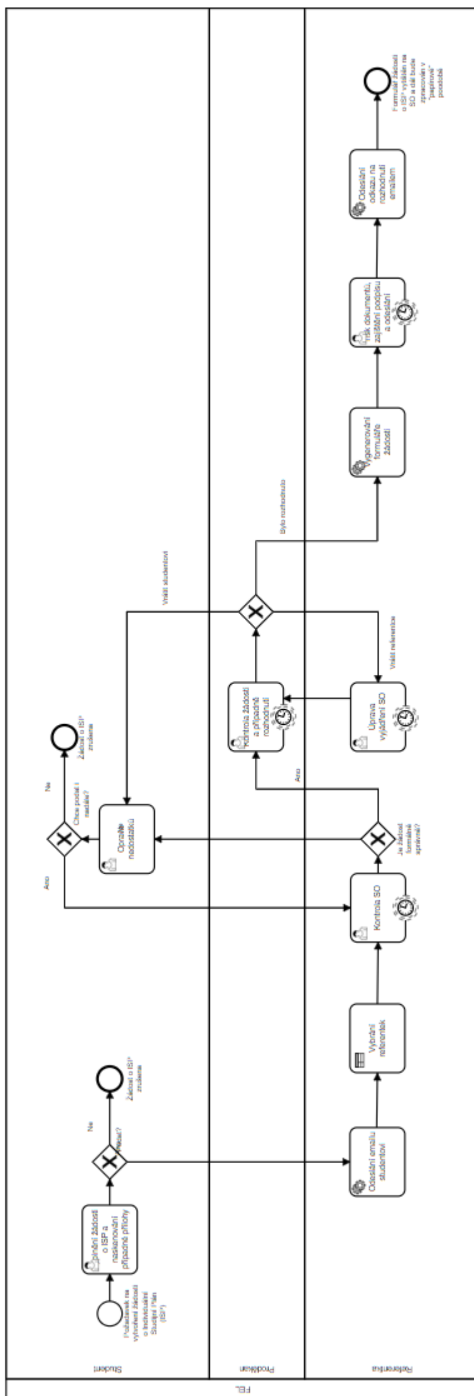
[20] *Camunda Blog: Testing Process Dependencies*.
https://camunda.com/blog/2020/11/testing-process-dependencies/. Navštíveno: 27.7.2021.

[21] *Camunda Blog: Test Your Processes With JUnit 5*.
https://camunda.com/blog/2021/01/test-your-processes-with-junit-5/. Navštíveno: 28.7.2021.

[22] *Camunda Blog: Measure Your Coverage*.
https://camunda.com/blog/2020/12/measure-your-coverage/. Navštíveno: 27.7.2021.

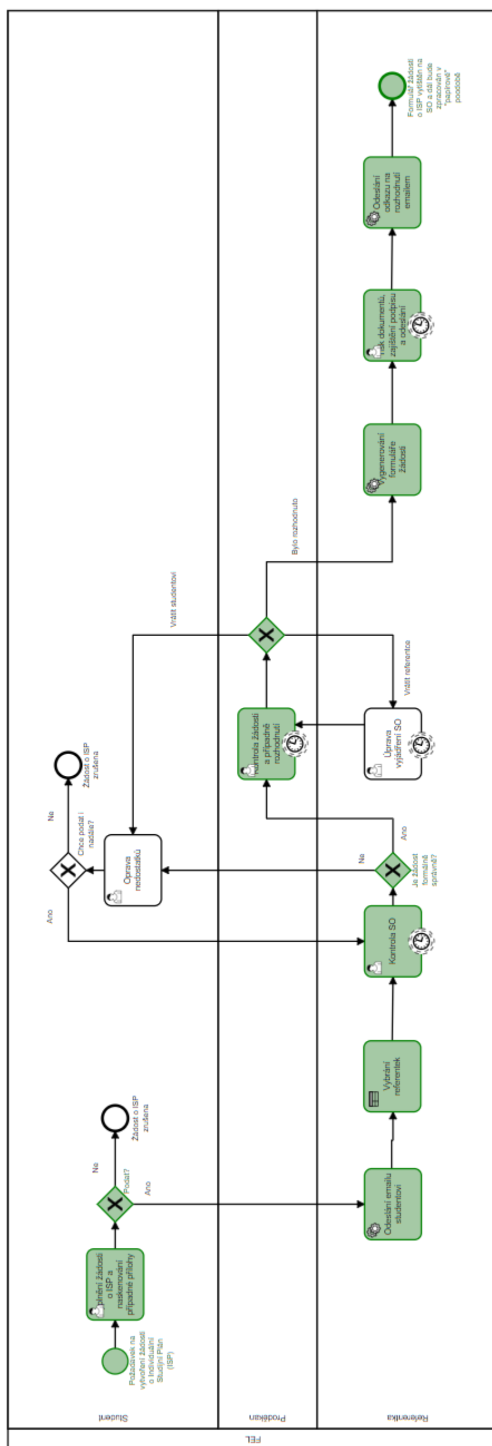
[23] *Camunda Specification: Testing*.
https://docs.camunda.org/manual/7.15/user-guide/testing/. Navštíveno: 2.8.2021.

Příloha A
Procesní diagram žádosti o individuální studijní plán



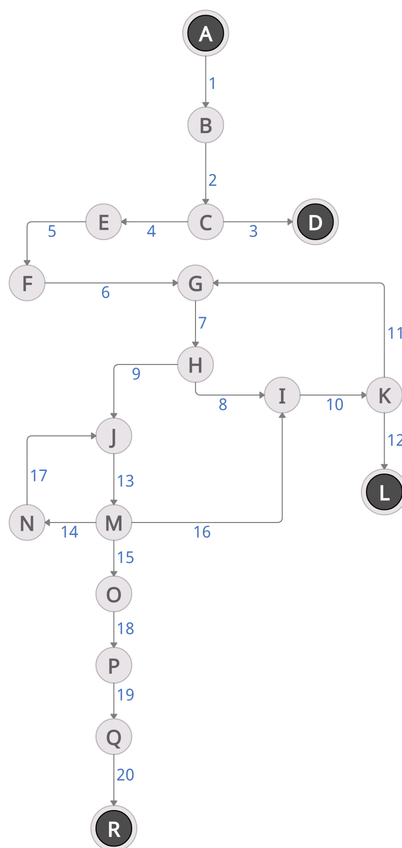
Obrázek A.1. Proces žádosti o individuální studijní plán

Příloha B
Happy path procesu Žádost o ISP



Obrázek B.2. Happy path procesu Žádost o ISP

Příloha C
Diagram bodů a cest procesu



Obrázek C.3. Diagram bodů a cest procesu

Příloha D
Přehled všech prvků procesu

Element/prvek					
Identifikační údaje	ID prvku	XML ID	Název prvku		
	A	StartEvent_0zgpfwh	Požadavek na vytvoření žádosti o Individuální Studijní Plán (ISP)		
Typ prvku	startovací událost				
Aktér	student				
Data	ID	Název	Typ		
	FormField_3chkcvs	basic info	studentBasicInfo		
Následující prvek	ID prvku	XML ID	ID hrany	XML ID hrany	
	B	submission	1	SequenceFlow_1x4h5qu	

Element/prvek					
Identifikační údaje	ID prvku	XML ID	Název prvku		
	B	submission	Vyplnění žádosti o ISP a naskenování případné přílohy		
Typ prvku	uživatelský úkol				
Aktér	student				
Vstupní data	ID	Název	Typ	Povinné?	
	address	Kontaktní adresa	address	ano	
	phone	Kontaktní telefon	string	ano	
	studentDesc	Zdůvodnění studenta	string	ano	
	semestrDefinition	Pro následující semestry	string	ano	
	creditsNumber	Celkem kreditů	long	ano	
	_REF_elsaDecision	Scan vyjádření ELSA	file	ne	
	_REF_medicalReport	Zpráva od lékaře	file	ne	
	_REF_attachment	Další příloha	file	ne	
Výstupní data	ID	Název	Typ	Nabývá hodnot	Vysvětlení
	continue	Podat žádost?	string	yes	student chce nadále podat žádost
				no	student chce žádost zrušit
Následující prvek	ID prvku	XML ID	ID hrany	XML ID hrany	
	C	ExclusiveGateway_09ijjk4	2	SequenceFlow_1xtyjdi	

Element/prvek				
Identifikační údaje	ID prvku	XML ID	Název prvku	
	C	ExclusiveGateway_09ijjk4	Podat?	
Typ prvku	exkluzivní brána			
Aktér	student			
Následující prvek	ID prvku	XML ID	ID hrany	XML ID hrany
	D	EndEvent_10q2b9p	3	SequenceFlow_0lqnlmh
	E	Task_13uwvcx	4	SequenceFlow_0pwkq41

Element/prvek			
Identifikační údaje	ID prvku	XML ID	Název prvku
	D	EndEvent_10q2b9p	Žádost o ISP zrušena
Typ prvku	koncová událost		
Aktér	student		
Následující prvek	-		

Element/prvek				
Identifikační údaje	ID prvku	XML ID	Název prvku	
	E	Task_13uwvcx	Odeslání emailu studentovi	
Typ prvku	úkol služby			
Aktér	referentka			
Následující prvek	ID prvku	XML ID	ID hrany	XML ID hrany
	F	Task_1ah4pyr	5	SequenceFlow_1umb8pj

Obrázek D.4. Přehled všech prvků procesu

Element/prvek						
Identifikační údaje	ID prvku	XML ID			Název prvku	
	F	Task_1ah4pyr			Vybrání referentek	
Typ prvku	byznys pravidlo					
Aktér	referentka					
Následující prvek	ID prvku	XML ID			ID hrany	XML ID hrany
	G	referentCheck			6	SequenceFlow_1g0xaj6

Element/prvek						
Identifikační údaje	ID prvku	XML ID			Název prvku	
	G	referentCheck			Kontrola SO	
Typ prvku	uživatelský úkol					
Aktér	referentka					
Výstupní data	ID	Název	Typ	Nabývá hodnot		Vysvětlení
	formalyCorrect	Je žádost formálně správně?	boolean	true		žádost je formálně správně a může být předána proděkanovi
				false		žádost není správně a student jí musí upravit
Následující prvek	ID prvku	XML ID			ID hrany	XML ID hrany
	H	ExclusiveGateway_0neifxz			7	SequenceFlow_0teis18

Element/prvek						
Identifikační údaje	ID prvku	XML ID			Název prvku	
	H	ExclusiveGateway_0neifxz			Je žádost formálně správně?	
Typ prvku	exkluzivní brána					
Aktér	referentka					
Následující prvek	ID prvku	XML ID			ID hrany	XML ID hrany
	I	Task_1u3qp82			8	SequenceFlow_10p294e
	J	vicedeanDecide			9	SequenceFlow_0oc3dg9

Element/prvek							
Identifikační údaje	ID prvku	XML ID			Název prvku		
	I	Task_1u3qp82			Oprava nedostatků		
Typ prvku	uživatelský úkol						
Aktér	student						
Vstupní data	ID		Název			Typ	Povinné?
	address		Kontaktní adresa			address	ano
	phone		Kontaktní telefon			string	ano
	studentDesc		Zdůvodnění studenta			string	ano
	semestrDefinition		Pro následující semestry			string	ano
	creditsNumber		Celkem kreditů			long	ano
	soDesc		Komentář referentky			string	readonly
	vicedeanDesc		Komentář proděkana			string	readonly
	_REF_elsaDecision		Scan vyjádření ELSA			file	ne
	_REF_medicalReport		Zpráva od lékaře			file	ne
	_REF_attachment		Další příloha			file	ne
Výstupní data	ID	Název	Typ	Nabývá hodnot		Vysvětlení	
	continue	Podat žádost?	string	yes		student chce nadále podat žádost	
				no		student chce žádost zrušit	
Následující prvek	ID prvku	XML ID			ID hrany	XML ID hrany	
	K	ExclusiveGateway_1eier24			10	SequenceFlow_05iufx9	

Element/prvek						
Identifikační údaje	ID prvku	XML ID			Název prvku	
	J	vicedeanDecide			Kontrola žádosti a případně rozhodnutí	
Typ prvku	uživatelský úkol					
Aktér	proděkan					
Výstupní data	ID	Název	Typ	Nabývá hodnot		Vysvětlení
	action	Je žádost formálně správně?	String	approved		žádost schválena
				disapproved		žádost zamítnuta
				backToStudent		žádost vrácena studentovi k úpravě
				backToReferent		žádost vrácena referentce k úpravě
Následující prvek	ID prvku	XML ID			ID hrany	XML ID hrany
	M	ExclusiveGateway_1iwj2yl			13	SequenceFlow_03v9jre

Element/prvek				
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**	
	K	ExclusiveGateway_1eier24	Chce podat i nadále?	
Typ prvku	exkluzivní brána			
Aktér	student			
Následující prvek	**ID prvku**	**XML ID**	**ID hrany**	**XML ID hrany**
	G	referentCheck	11	SequenceFlow_0iw145y
	L	EndEvent_0ckmrl6	12	SequenceFlow_1bd46ti

Element/prvek			
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**
	L	EndEvent_0ckmrl6	Žádost o ISP zrušena
Typ prvku	koncová událost		
Aktér	student		
Následující prvek	-		

Element/prvek				
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**	
	M	ExclusiveGateway_1iwj2yl	Je žádost a vyjádření formálně správně?	
Typ prvku	exkluzivní brána			
Aktér	proděkan			
Následující prvek	**ID prvku**	**XML ID**	**ID hrany**	**XML ID hrany**
	I	Task_1u3qp82	16	SequenceFlow_0fgovj2
	N	UserTask_01s4v3o	14	SequenceFlow_0mqu5ot
	O	Task_1sypl51	15	SequenceFlow_0xy7bgg

Element/prvek					
Identifikační údaje	**ID prvku**	**XML ID**		**Název prvku**	
	N	UserTask_01s4v3o		Úprava vyjádření SO	
Typ prvku	uživatelský úkol				
Aktér	referentka				
Výstupní data	**ID**	**Název**	**Typ**	**Nabývá hodnot**	**Vysvětlení**
	formalyCorrect	Je žádost formálně správně?	boolean	true	žádost je formálně správně a může být předána proděkanovi
				false	žádost není správně a student jí musí upravit
Následující prvek	**ID prvku**	**XML ID**		**ID hrany**	**XML ID hrany**
	J	vicedeanDecide		17	SequenceFlow_14f92sb

Element/prvek				
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**	
	O	Task_1sypl51	Vygenerování formuláře žádosti	
Typ prvku	úkol služby			
Aktér	referentka			
Následující prvek	**ID prvku**	**XML ID**	**ID hrany**	**XML ID hrany**
	P	UserTask_0tyrutt	18	SequenceFlow_05qe808

Element/prvek				
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**	
	P	UserTask_0tyrutt	Tisk dokumentů, zajištění podpisu a odeslání	
Typ prvku	uživatelský úkol			
Aktér	referentka			
Následující prvek	**ID prvku**	**XML ID**	**ID hrany**	**XML ID hrany**
	Q	ServiceTask_0nemsdw	19	SequenceFlow_0t6cuue

Element/prvek				
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**	
	Q	ServiceTask_0nemsdw	Odeslání odkazu na rozhodnutí emailem	
Typ prvku	Úkol služby			
Aktér	referenka			
Následující prvek	**ID prvku**	**XML ID**	**ID hrany**	**XML ID hrany**
	R	EndEvent_0uacui2	20	SequenceFlow_1a8hzl8

Element/prvek			
Identifikační údaje	**ID prvku**	**XML ID**	**Název prvku**
	R	EndEvent_0uacui2	Formulář žádosti o ISP vytištěn na SO a dál bude zpracován v papírové podobě
Typ prvku	koncová událost		
Aktér	referentka		
Následující prvek	-		

Příloha E

Pokrytí cest		
Bod	**Vstupní akce**	**Výstupní akce**
A *(start)*	-	1
B	1	2
C	2	3 4
D *(konec)*	3	-
E	4	5
F	5	6
G	6 11	7
H	7	8 9
I	8 16	10
J	9 17	13
K	10	11 12
L *(konec)*	12	-
M	13	14 15 16
N	14	17
O	15	18
P	18	19
Q	19	20
R *(konec)*	20	-

Hloubka pokrytí N=3	
Bod	**Kombinace**
A *(start)*	–
B	1–2–3 1–2–4
C	2–3 2–4–5
D *(konec)*	–
E	4–5–6
F	5–6–7
G	6–7–8 6–7–9 11–7–8 11–7–9
H	7–8–10 7–9–13
I	8–10–12 8–10–11 16–10–12 16–10–11
J	9–13–14 9–13–15 9–13–16 17–13–14 17–13–15 17–13–16
K	10–12 10–11–7
L *(konec)*	–
M	13–14–17 13–15–18 13–16–10
N	14–17–13
O	15–18–19
P	18–19–20
Q	19–20
R *(konec)*	–

Obrázek E.5.

Příloha F

Ukázka rozhodovací DMN tabulky pro výběr referentek

Choose referent group | **Hit Policy:** First

	When Programme ID string	And Study form string	Then Referent group string	Annotations
1	"programmes/BN1/"	"FULLTIME"	"13922-all"	???Elektrotechnika a informatika, strukturovaný
2	"programmes/BN2/"	"FULLTIME"	"13922-referentky-STM, 13922-all"	Softwarové technologie a management
3	"programmes/BP1/"	"FULLTIME"	"13922-referentky-EEM, 13922-all"	Elektrotechnika, energetika a management
4	"programmes/BP2/"	"FULLTIME"	"13922-referentky-KME,13922-all"	[illegible]unikace, multimédia a elektronika
5	"programmes/BP3/"	"FULLTIME"	"13922-referentky-KYR, 13922-all"	Kybernetika a robotika
6	"programmes/BP4/"	"FULLTIME"	"13922-referentky-OI, 13922-all"	Otevřená informatika
7	"programmes/MP1/"	"FULLTIME"	"13922-referentky-EEM, 13922-all"	Elektrotechnika, energetika a management
8	"programmes/MP2/"	"FULLTIME"	"13922-referentky-KME, 13922-all"	Komunikace, multimédia a elektronika
9	"programmes/MP3/"	"FULLTIME"	"13922-referentky-KYR, 13922-all"	Kybernetika a robotika
10	"programmes/MP4/"	"FULLTIME"	"13922-referentky-OI, 13922-all"	Otevřená informatika

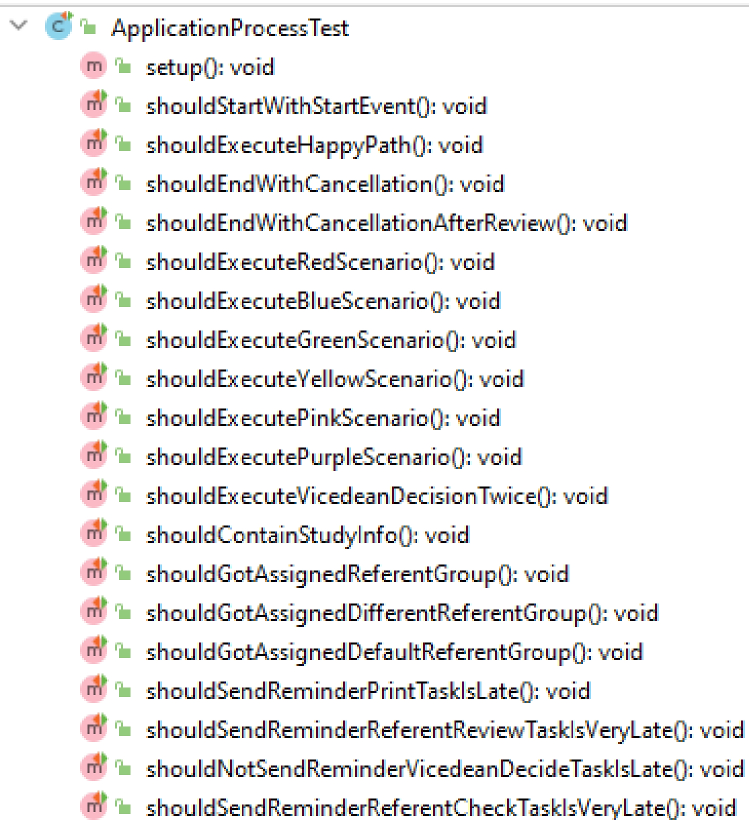
Obrázek F.6. Ukázka rozhodovací DMN tabulky pro výběr referentek

Příloha G
Definice proměnných testovací třídy

```
public static final String PROCESS_KEY = "zadost_isp";
public static final String START_EVENT = "StartEvent_0zgpfwh";
public static final String END_EVENT_APPLICATION_COMPLETED = "EndEvent_0uacui2";
public static final String END_EVENT_APPLICATION_CANCELLED = "EndEvent_10q2b9p";
public static final String END_EVENT_APPLICATION_CANCELLED_AFTER_REVIEW = "EndEvent_0ckmrl6";
public static final String BOUNDRARY_EVENT_TIMER_PRINT = "BoundaryEvent_1yvpsdc";
public static final String BOUNDRARY_EVENT_TIMER_REFERENT_REVIEW = "BoundaryEvent_16r0fuy";
public static final String BOUNDRARY_EVENT_TIMER_VICEDEAN_DECISION = "BoundaryEvent_0899an5";
public static final String BOUNDRARY_EVENT_TIMER_REFERENT_DECISION = "BoundaryEvent_1ep1608";

public static final String USER_TASK_SUBMIT_APPLICATION = "submission";
public static final String USER_TASK_REFERENT_CHECK = "referentCheck";
public static final String USER_TASK_APPLICATION_REVIEW = "Task_1u3qp82";
public static final String USER_TASK_VICEDEAN_CHECK = "vicedeanDecide";
public static final String USER_TASK_REFERENT_REVIEW = "UserTask_01s4v3o";
public static final String USER_TASK_PRINT_FORM = "UserTask_0tyrutt";

public static final String SERVICE_TASK_SEND_EMAIL_APPLICATION_RECEIVED = "Task_13uwvcx";
public static final String SERVICE_TASK_GENERATE_FORM = "Task_1sypl51";
public static final String SERVICE_TASK_SEND_EMAIL_APPLICATION_EVALUATED = "ServiceTask_0nemsdw";

public static final String BUSINESS_RULE_TASK_CHOOSE_REFERENT = "Task_1ah4pyr";

public static final String VAR_FORMALLY_CORRECT = "formalyCorrect";

public static final String VAR_ACTION = "action";
ArrayList<String> actions = new ArrayList<String>();

public static final String VAR_CONTINUE = "continue";
ArrayList<String> continueOptions = new ArrayList<String>();
```

Obrázek G.7. Definice proměnných třídy ApplicationProcessTest

Příloha H
Přehled vytvořených testů



Obrázek H.8. Přehled vytvořených testů

Příloha I
Vyhodnocení testu *shouldExecuteHappyPath()*

```
14:17:44.460 [main] DEBUG org.camunda.bpm.engine.test - annotation @Deployment creates deployment for ApplicationProcessTest.shouldExecuteHappyPath
14:17:44.503 [main] INFO  org.camunda.bpm.scenario - * Starting scenario at 2021-08-11 14:17:44
14:17:44.513 [main] INFO  org.camunda.bpm.scenario - | Completed startEvent        'Požadavek na vytvoření žádosti o Individuální Studijní Plán (ISP)'
 (StartEvent_0zgpfwh @ zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.513 [main] INFO  org.camunda.bpm.scenario - * Acting on userTask          'Vyplnění žádosti o ISP a naskenování případné přílohy' (submission @
 zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.537 [main] INFO  org.camunda.bpm.scenario - | Completed userTask          'Vyplnění žádosti o ISP a naskenování případné přílohy' (submission @
 zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.537 [main] INFO  org.camunda.bpm.scenario - | Completed exclusiveGateway  'Podat?' (ExclusiveGateway_09ijjk4 @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.537 [main] INFO  org.camunda.bpm.scenario - | Completed serviceTask       'Odeslání emailu studentovi' (Task_13uwvcx @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.537 [main] INFO  org.camunda.bpm.scenario - | Completed businessRuleTask  'Vybrání referentek' (Task_1ah4pyr @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.537 [main] INFO  org.camunda.bpm.scenario - * Acting on userTask          'Kontrola SO' (referentCheck @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.556 [main] INFO  org.camunda.bpm.scenario - | Completed userTask          'Kontrola SO' (referentCheck @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.556 [main] INFO  org.camunda.bpm.scenario - | Completed exclusiveGateway  'Je žádost formálně správně?' (ExclusiveGateway_0neifxz @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.556 [main] INFO  org.camunda.bpm.scenario - * Acting on userTask          'Kontrola žádosti a případně rozhodnutí' (vicedeanDecide @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.577 [main] INFO  org.camunda.bpm.scenario - | Completed userTask          'Kontrola žádosti a případně rozhodnutí' (vicedeanDecide @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.577 [main] INFO  org.camunda.bpm.scenario - | Completed exclusiveGateway  (ExclusiveGateway_1iwj2yl @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.577 [main] INFO  org.camunda.bpm.scenario - | Completed serviceTask       'Vygenerování formuláře žádosti' (Task_1sypl51 @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.577 [main] INFO  org.camunda.bpm.scenario - * Acting on userTask          'Tisk dokumentů, zajištění podpisu a odeslání' (UserTask_0tyrutt @
 zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.588 [main] INFO  org.camunda.bpm.scenario - | Completed userTask          'Tisk dokumentů, zajištění podpisu a odeslání' (UserTask_0tyrutt @
 zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.588 [main] INFO  org.camunda.bpm.scenario - | Completed serviceTask       'Odeslání odkazu na rozhodnutí emailem' (ServiceTask_0nemsdw @ zadost_isp #
 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.588 [main] INFO  org.camunda.bpm.scenario - | Completed noneEndEvent      'Formulář žádosti o ISP vytištěn na SO a dál bude zpracován v "papírové"
 podobě' (EndEvent_0uacui2 @ zadost_isp # 21e16bbd-fa9e-11eb-9ad0-00ff1e5da95a)
14:17:44.598 [main] INFO  org.camunda.bpm.scenario - * Finishing scenario at 2021-08-11 14:17:44
srp 11, 2021 2:17:44 ODP. org.camunda.bpm.extension.process_test_coverage.junit.rules.TestCoverageProcessEngineRule succeeded
INFO: shouldExecuteHappyPath(cz.cvut.fel.czm.camunda.isp.ApplicationProcessTest) succeeded.
srp 11, 2021 2:17:44 ODP. org.camunda.bpm.extension.process_test_coverage.junit.rules.TestCoverageProcessEngineRule handleTestMethodCoverage
INFO: shouldExecuteHappyPath test method coverage is 0.5952380952380952
14:17:44.603 [main] DEBUG org.camunda.bpm.engine.test - annotation @Deployment deletes deployment for ApplicationProcessTest.shouldExecuteHappyPath
```

Obrázek I.9. Vyhodnocení testu *shouldExecuteHappyPath()*

Příloha J
Seznam použitých zkratek a pojmů

API	Rozhraní pro programování aplikací
BPMN	Business Process Model and Notation - grafická notace pro modelování podnikových procesů
CMMN	Case Management Model and Notation - grafická notace pro modelování událostí
DMN	Decision Model and Notation - grafická notace pro modelování rozhodnutí
FEL ČVUT	Fakulta elektrotechnická Českého vysokého učení technického v Praze
framework	ucelený soubor tematicky zaměřených knihoven
HTML	Hypertextový značkovací jazyk (Hypertext Markup Language)
open source	otevřený software - zdrojový kód softwaru je volně dostupný
REST	Representational State Transfer
unit test	jednotkový test
workflow	posloupnost činností
XML	Extensible Markup Language neboli rozšiřitelný značkovací jazyk