FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webová aplikace pro generování kombinací testovacích dat
Student:	Bc. Václav Rechtberger
Vedoucí:	Ing. Miroslav Bureš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Navrhněte a implementujte webovou aplikaci umožňující generování kombinací testovacích dat ze zadané definice úlohy.
V aplikaci bude možné vytvořit projekt obsahující několik definicí úlohy. Pro každou definici úlohy bude systém umožňovat generování sady testovacích situací s různými úrovněmi testovacího pokrytí.
K sadě testovacích situací bude možné přistupovat pomocí specializovaného API a tuto sadu bude možné exportovat ve formátu CSV.
Pro generování testovacích situací můžete využít již dostupný publikovaný algoritmus nebo volně dostupný zdrojový kód. Při generování testovacích scénářů reflektujte sadu omezujících podmínek, která je součástí definice úlohy.
Systém implementujte pomocí technologie J2EE a jeho funkčnost ověřte sadou vhodných testů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. září 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Webová aplikace pro generování kombinací testovacích dat

Bc. Václav Rechtberger

FIT, Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Bureš, Ph.D.

10. ledna 2019

Poděkování

Tímto bych rád poděkoval vedoucímu mé diplomové práce Ing. Miroslavu Burešovi, Ph.D. za jeho přínosné rady, připomínky a vedení v průběhu psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. ledna 2019 .....

Odkaz na tuto práci

Rechtberger, Václav. *Webová aplikace pro generování kombinací testovacích dat*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Častým problémem při testování softwaru je volba vhodných testovacích dat. Hlavní aspektem při této volbě je efektivita testů, z praktického pohledu rozumný kompromis mezi testovacím pokrytím a množstvím kombinací k otestování, tedy kompromis mezi množstvím odhalených defektů a přijatelnou dobou testování.

Předmětem této práce je návrh a vývoj webového uživatelského rozhraní pro generování takových dat za použití dostupných softwarových knihoven pro generování testovacích kombinací. Platforma má být otevřená a dostupná širokému spektru uživatelů, což v současné době chybí mezi stávajícími řešeními. Cílem práce tedy není rozebírat samotnou problematiku generování testů, neboť se jedná o složitější výzkumné téma.

Cíl práce byl naplňen vyvinutím J2EE webové aplikace s přímočarým ovládaním a možností vzdáleného přístupu k testovacím datům v rámci lepší použitelnosti. Aplikace umožňuje specifikaci konkrétní úlohy a parametrů pro generování testů, uložených jako objekty zvláštního projektu, stejně tak jako generování, prohlížení a export vytvořených testovacích scénářů včetně vzdáleného trvalého přístupu k těmto scénářům. Věřím, že tato aplikace bude přínosným nástrojem ve světě vývoje softwaru a jeho testovaní.

Klíčová slova testování software, testovací data, genrování kombinací dat, webová aplikace, J2EE

Abstract

The choice of a suitable testing data is a common problem within software testing. The main aspect of this choice is the effectivity of tests, which is, from a practical point of view, a reasonable compromise between test coverage and the amount of the testing combinations, therefore a compromise between amount of defects detected and acceptable testing run time.

The subject of this thesis is the design and development of a user web platform for this type of data generating while using accessible software libraries for testing combinations generating. The platform should be open and accessible to a wide spectrum of users, which is something that is missing amongst the current solutions. The goal of this thesis is however not to analyze the problematic of test generating, as that is a more complicated research topic.

The aim of the thesis was fulfilled by development of the J2EE web application with straightforward control and the option of remote access to the testing data which helps its usability. The application enables tasks and parameters specification for generating tests which are saved as objects of a special project, as well as generating, viewing and exporting of created testing scenarios including the remote permanent access to the scenarios. I believe that this application will be a beneficial tool in the world of software development and testing.

Keywords software testing, test data, generate data combination, web application, J2EE

Obsah

Úvod **1**

1 Analýza a návrh **3**
1.1 Požadavky na vyvíjený systém 3
1.2 Aktéři, případy užití a scénáře případů užití 5
1.3 Návrh uživatelského rozhraní 18
1.4 Doménový model 24
1.5 Základní procesy v systému 26

2 Realizace **35**
2.1 Použité technologie 35
2.2 Interní procesy v systému 37
2.3 Model nasazení 40
2.4 Testování 44

Závěr **55**

Literatura **57**

A Seznam použitých zkratek **59**

B Obsah přiloženého CD **61**

Seznam obrázků

1.1 Aktéři v systému 6
1.2 Případy užití 7
1.3 Návrh základního layoutu 20
1.4 Návrh menu aplikace 21
1.5 Návrh obrazovky seznamu projektů 21
1.6 Návrh obrazovky projektu 22
1.7 Návrh obarzovky s konfiguracemi pro pokročilé genrování testovacích scénářů 22
1.8 Návrh obrazovky přehledu testovacích scénářů 23
1.9 Návrh obrazovky testovacích scénářů 23
1.10 Použití speciálního odkazu na přístup k testovacím scénářům . . . 24
1.11 Konceptuální datový model systému 25
1.12 Proces správy projektu 27
1.13 Proces správy množiny parametrů 28
1.14 Proces správy parametru 29
1.15 Proces správy konfigurace 30
1.16 Proces správy množiny interakce 31
1.17 Proces správy testovací množiny 32
1.18 Proces vyřízení požadavku na speciální odkaz pro vzdálený přístup 33

2.1 Interní proces systému při vytváření projektu 38
2.2 Interní proces systému při volbě pracovního projektu 38
2.3 Interní proces systému při inicializaci prezentovaných dat 39
2.4 Interní proces systému při vytváření množiny prametrů 40
2.5 Interní proces systému při vytváření parametru 41
2.6 Interní proces systému při vkládání hodnoty do parametru 41
2.7 Sekvence systému při generování testovací množiny 42
2.8 Model nasazení systému 43

Úvod

Mnoho defektů v softwaru je vyvoláno nastavením konkrétní hodnoty některé z proměnných či parametrů systému. Jsou ale i defekty, které jsou více sofistikované a projeví se až kombinací dvou hodnot. Jak lze tušit, dvojkombinace nemusí být strop. V programu se může nacházet celá řada rafinovaných defektů, které se projeví až troj a více kombinací konkrétních hodnot. Tyto defekty jsou sice už méně časté, avšak existuje celá řada kritických systémů, které si nemohou dovolit žádnou chybu [1]. Nabízí se možnost testování všech možných kombinací (takzvané vytěžující testy). Tato možnost je však vzhledem k množství počtu parametrů v reálných systémech a tedy množství možných kombinací prakticky nevyužitelná a používá se jen u obzvláště kritických systémů. Výsledný počet kombinací a tedy množství testovacích scénářů je potřeba vhodným způsobem redukovat.

Pro testování kombinací o síle dvou interagujících parametrů je zaveden název párové či dvoucestné testovaní (anglicky pairwise testing nebo 2-way testing). Mezinárodní organizace pro certifikování softwarových testerů (International Software Testing Qualifications Board, ISQTB) definuje toto dvoucestné testovaní, v ISQTB terminologii All-Pairs Testing či Pairwise Testing, jako techniku black-box testování, kdy kontrolujeme všechny kombinace hodnot všech možných párů parametrů.

Jak ukazují různé studie, například [2, 3, 4, 5, 6], pro odhalení většiny chyb je dvoucestné testování dostačující. Nicméně pro kritičtější systémy nemusí síla párových kombinací dostačovat a tak lze techniku rozšířit na kombinace trojic (3-way testing) [6]. V tomto případě musí výsledné kombinace obsahovat všechny kombinace hodnot pro všechny trojice parametrů. Se vzrůstající kritičnostní systému nemusí dostačovat ani síla trojkombinací a tak je příhodné techniku zobecnit na pokrytí kombinací n-tic (n-way testing), kdy analogicky k předešlým příkladům obsahuje výsledná množina kombinací kombinace všech hodnot všech n-tic parametrů.

V realném světě se nejčastěji používá n do stupně 5, avšak není to neprolomitená bariéra a používá se i stupeň vyšší 5. U těchto testů už samozřejmě

není výsledná množina kombinací tak redukována jako v případě nižších (a tak je dobré zvážit potřebu této síly pokrytí), avšak stále se jedná oproti všem možným kombinacím (n rovnající se počtu parametrů) velmi redukovaný počet kombinací.

Za zmínku stojí, že tato testovací techninka není vhodná pouze pro obvyklé IT systémy, ale je aplikovatelná i ve světě Internet of Things (IoT) systémů, kde je taktéž často třeba řešit problém s obrovským množstvím možných kombinací [7].

Pro generování testovacích kombinací v současnosti existuje řada knihoven, které prakticky implementují řadu algoritmů vyvinutých pro řešení této úlohy, například [8, 9, 10, 11, 12, 13, 14], ale obvykle k nim neexistuje vhodné, uživatelsky přívětivé rozhraní, nebo existuje, ale není webové a volně dostupné.

Proto jsme se takové rozhraní rozhodli vytvořit v této práci.

Kapitola 1

Analýza a návrh

V analytické části budou nejprve určeny základní požadavky a vlastnosti systému. Posléze bude uveden návrh řešení obsahující i doménový model problému. Dále budou uvedeny návrhy obrazovek webové aplikace. A nakonec budou uvedeny postupy řešení - diagramy procesů systému, respektive interakcí uživatele a systému.

1.1 Požadavky na vyvíjený systém

V této kapitole je uvedeno, jaké požadavky jsou kladeny na vyvíjený systém. Tyto požadavky jsou identifikovány na základě analýzy zadání.

1.1.1 Funkcionální požadavky

Funkcionální požadavky popisují chování systému z pohledu uživatele, které musí systém poskytovat či vykonávat.

1. **Správa projektů**

   a) **Možnost vytvořit a spravovat více projektů najednou** Systém umožňuje vytvořit víc projektů. Mezi vytvořenými projekty je možné přepínat.

   b) **Možnost spravovat víc definicí úlohy.** V každém projektu je možno spravovat více specifikací úlohy.

   c) **Kompletní editace projektu** Systém umožňuje provést CRUD operace pro projekt a datové objekty, ze kterých se projekt skládá.

2. **Definice specifikace úlohy** Systémové funkcionality umožňují zadat data odpovídající specifikaci

a) **Množina parametrů** Skládá se z jednotlivých parametrů. Tvoří kořen každého zadání úlohy. Součástí množiny parametrů je i množina konfigurací, dle kterých se generují kombinatorické testovací scénáře (viz dále).

b) **Parametr** Množina hodnot kterých může daný parametr nabývat. Každému prametru je přiřazen datový typ.

c) **Hodnota** Tvoří základní stavební prvek celé specifikace, tedy list stromu.

d) **Datový typ** Určuje obor hodnot parametru. Použito bude datových typů: celé číslo, boolean a výčet.

e) **Konfigurace** Pomocná struktura pro pokročilé genorovaní kombinatorických testovacích scénářů skládající se z množin interakce.

f) **Množina interakcí** Každá množina interakcí udáva její sílu (interaction strength) a množinu parametrů, které jsou podmnožinou dané množiny parametrů, pro které bude síla použita. Množiny interakcí se mouhou v rámci jedné konfigurace překrývat.

3. **Generování kombinatorických testovacích scénářů** Ke generování je možné využít cizí dostupné algoritmy či knihovny s volnou licencí.

   a) **Systém umožňuje generování kombinatorických testovacích scénářů pro definované specifikace úlohy.**

   b) **Podpora různých úrovní testovacího pokrytí pro uniformní sílu interakce**. Pro generování testovacích dat je možné volit různé úrovně pokrytí (sílu interakce mezi parametry z definice úlohy). Systém podporuje generování testovacích scénářů pro sílu interakce 1 až 6.

   c) **Podpora testovacího pokrytí typu mixed interaction strength**. V systému je možné tvořit podmnožiny s různou úrovní interakce (mixed interaction strength). Testovací scénáře bude možné generovat i pro tento typ testovacího pokrytí. Pokud v rámci jedné konfigurace existují množiny interakcí s neprázdným průnikem parametrů, parametry z tohoto průniku budou mít integration strengt vyššího stupně.

   d) **Generovat na pozadí** Genrování scénářů může být déle trvající úkol, je proto nutné, aby generování probíhalo na pozadí.

4. **Přístup k testovacím datům pomocí speciálního API** Pro efektivnější přístup k testovacím datům a především jejich snadnější čitelnost jinými systémy budou generovaná data přístupná skrze speciální API.

   a) **Generovaná data jsou přístupná skrze speciální odkaz.** Tento odkaz slouží pro vzdálený přístup, nevyžaduje přihlášení a lze jej aktivovat a deaktivovat.

b) **API vrátí CSV soubor.** S čárkou použitou jako oddělovačem.

5. **Prohlížení generovaných sad kombinatorických testovacích scénářů**
   a) **Poskytnout metadata o vstupní úloze**
   b) **Poskytnout metadata o výstupním scénáři**
6. **Zabezpečení systému a správa uživatelských účtů**
   a) **Uživatelský účet** Je tvořen emailovou adresou, heslem a rolí.
   b) **Heslo** Heslo nesmí být v databázi drženo v čitelné podobě.
   c) **Role** Systém rozeznává roli uživatele a správce. Jen správce má možnost přidávat a mazat uživatelské účty.
   d) **poskytující přístup k testovacím scénářum nepodléha zabezpečení**

1.1.2 Nefunkcionání požadavky

Nefunkcionální požadavky rozšiřují či upřesňují požadavky funkcionální. Mohou například určovat použité technologie, datové formáty, zabezpečení, dostupnost a jiné vlastnosti systému.

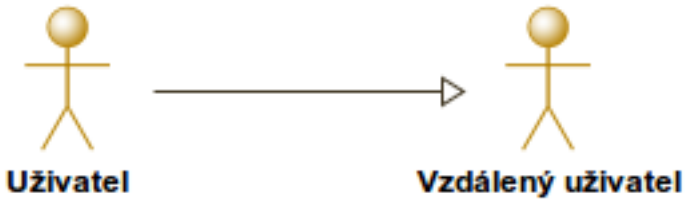
1. **Použití technologie J2EE** [15]
   a) Grafické rozhraní je implementováno pomocí frameworku Java Server Faces (JSF).
   b) Pro inicializaci, kompozici a přístup k objektům je využito technologie Contexts and Dependency Injection (CDI).
   c) Datový model je persistován za využití specifikace Java Persistence API (JPA) a její implementace Hibernate.
   d) Celá aplikace běží na serveru Wildfly 12 [16], který poskytuje J2EE implementaci.
2. **Kvalita uživatelského rozhraní a spolehlivost aplikace**
   a) Ověření návrhu grafického rozhraní pomocí heuristické analýzy Jakoba Nielsena
   b) Manuální testování dle navržených testovacích scénářů ověřujících splnění funkčních požadavků

1.2 Aktéři, případy užití a scénáře případů užití

V této kapitole v detailu popíšeme aktéry interagující se systémem, případy užití a detail případů užití uvedený jako scénáře případů užití.

1.2.1 Aktéři v systému

Aktéři jsou uživatelé systému, uživatelem může být člověk, ale i jiný systém či aplikace. Systém používá dva základní aktéry systému, uživatele a vzdáleného uživatele (viz obrázek 1.1).



Obrázek 1.1: Aktéři v systému

Uživatel je uživatel který do systému vkládá zadání úloh, ke kterým následně generuje testovací scénáře. Dále zpřístupňuje vygenerované scénáře skrze speciální odkazy pro vzdálený přistup k testovacím množinám.

Vzdálený uživatel využívá systém pouze pomocí speciálních odkazů umožňující vzdálený přístup k testovacím množinám, který zpřístupnil jíný uživatel s aktivním přístupem do systému. Vzdáleným uživatelem může být jiný systém či aplikace data využívající.

1.2.2 Případy užití

Případy užití jsou seznamem aktivit, které může uživatel se systémem vykonávat. Jejich scénáře zachycují případnou interakci mezi uživateli a systémem, případně mezi uživateli samotnými. Případy užití by se neměly snažit popsat navigaci v systému, ale pouze popsat aplikační logiku vykonávaných procesů.

Přehled případů užití je uveden na obrázku 1.2.

V dalších podsekcích popisuji detailní scénáře případů užití.

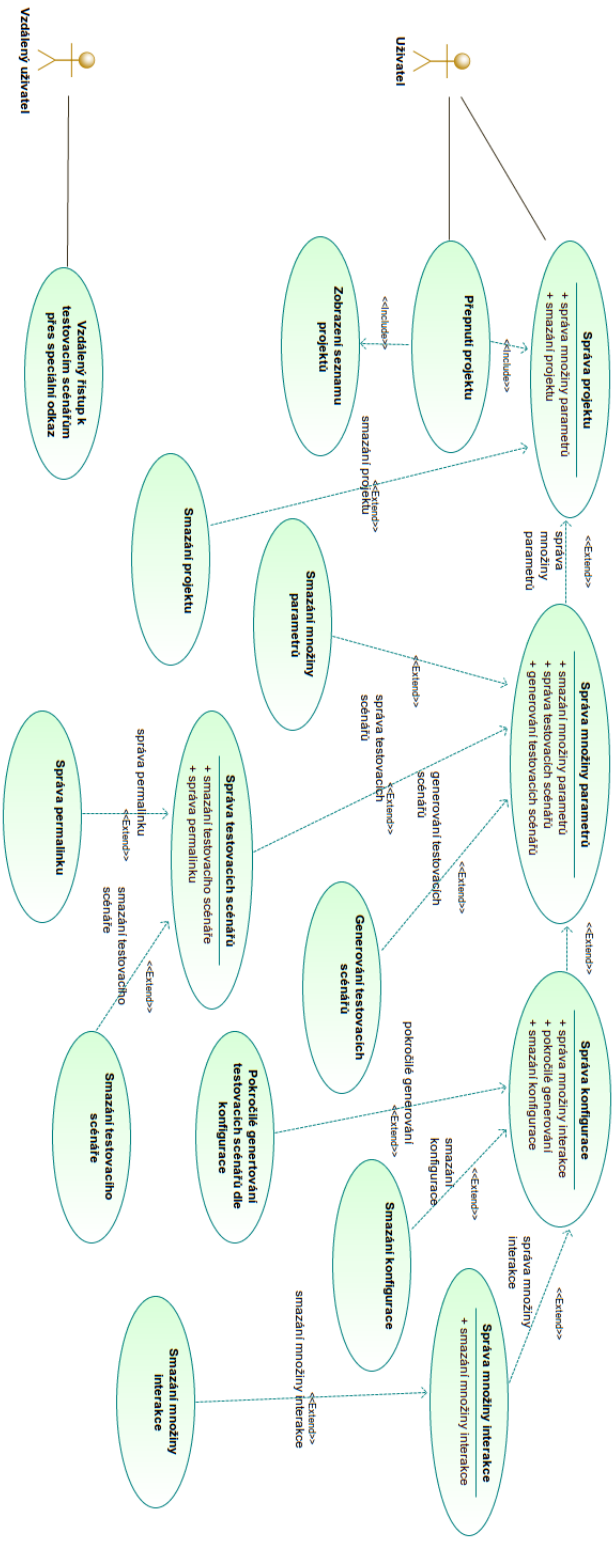
1.2.2.1 Scénáře případů užití

Správa projektu
ID: 1
Výchozí stav:

- UŽIVATEL je přihlášen do systému.

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o zobrazení projektu.



Obrázek 1.2: Případy užití

2. SYSTÉM zobrazí projekt. Jeho název, množiny parametrů a jim náležící parametry s hodnotami.
(správa PS)(smazání projektu)

Koncový stav:

- -

Alternativy
1a: Uživatel vytváří nový projekt

1a1. UŽIVATEL požádá SYSTÉM o vytvoření nového projektu.

1a2. SYSTÉM vyzve uživatele k zadání názvu nového projektu.

1a3. UŽIVATEL zadá název pro nový projekt.

1a4. SYSTÉM vytvoří nový projekt a uloží jej.

Alternativní koncový stav:

- V systému je vytvořen nový projekt.

2a: Uživatel edituje jméno projektu

2a1. SYSTÉM zobrazí projekt. Jeho název, množiny parametrů a jim náležící parametry s hodnotami.

2a2. UŽIVATEL zadá nový název pro právě spravovaný projektu.

2a3. SYSTÉM uloží nový název projektu.

Alternativní koncový stav:

- Projekt je přejmenován.

Smazání projektu
ID: 2
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa systému, ID: 1).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o smazání projektu.
2. SYSTÉM se uživatele dotáže, zda chce opravdu projekt smazat.
3. UŽIVATEL potvrdí smazání projektu

4. SYSTÉM smaže projekt

Koncový stav:

- Projekt je smazán.

Alternativy
3a: Uživatel zamítá smazání projektu

3a1. UŽIVATEL zamítne smazání projektu.

3a2. SYSTÉM neprovede smazání projektu.

Alternativní koncový stav:

- Projekt není smazán.

Správa množiny parametrů
ID: 3
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa systému, ID: 1).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o zobrazení množiny parametrů.
2. SYSTÉM zobrazí množinu parametrů.
   (smazání setu parametrů)

Koncový stav:

- -

Alternativy: 1a: Uživatel vytváří novou množinu parametrů

1a1. UŽIVATEL požádá SYSTÉM o vytvoření nové množiny parametrů.

1a2. SYSTÉM vyzve uživatele k zadání názvu nové množiny parametrů.

1a3. UŽIVATEL zadá název.

1a4. SYSTÉM vytvoří novou množinu parametrů.
(smazání množiny parametrů)
(správa testovacích množin)
(generování testovacích množin)

Alternativní koncový stav:

- -

2a: Uživatel vytváří novou množinu parametrů

2a1. SYSTÉM zobrazí množinu parametrů.

2a2. UŽIVATEL přejmenuje množinu parametrů.

2a3. SYSTÉM uloží nové jméno množiny parametrů.

Alternativní koncový stav:

- -

Smazání množiny parametrů
ID: 4
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa množiny parametrů, ID: 3).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o smazání množiny parametrů.
2. SYSTÉM se UŽIVATELE dotáže, zda chce opravdu množinu parametrů smazat.
3. UŽIVATEL potvrdí smazání množiny parametrů.
4. SYSTÉM smaže množinu parametrů.

Koncový stav:

- Množina parametrů je smazána.

Alternativy
3a: Uživatel zamítá smazání množiny parametrů

3a1. SYSTÉM neprovede smazání množiny parametrů.

Alternativní koncový stav:

- Množina parametrů není smazána.

Generování testovacího scénáře
ID: 5
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa množiny parametrů, ID: 4).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o vygenerování nové množiny parametrů dle zvolené množiny parametrů.
2. SYSTÉM se uživatele dotáže, s jakou sílou interakce chce uživatel scénáře generovat.
3. UŽIVATEL uživatel zvolí sílu.
4. SYSTÉM vygeneruje testovací scénář se zvolenou sílou interakce.
5. SYSTÉM informuje uživatele o vygenerování testovacího setu

Koncový stav:

- Je vygenerován nový testovací scénář. Tento scénář je zařazen k setu parametrů, ze kterého byl vygenerován.

Alternativy
3a: Uživatel zruší generování

3a1. SYSTÉM neprovede generování scénářů.

Alternativní koncový stav:

- Testovací scénáře nejsou vygenerovány.

4a: Při generování scénáře vyvstanou chyby

4a1. SYSTÉM při generování scénářů zaznamená chybu.

4a2. SYSTÉM oznámí toto selhání uživateli.

Alternativní koncový stav:

- Testovací scénáře nejsou vygenerovány.

Správa testovacích scénářů
ID: 6
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa množiny parametrů, ID: 4).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o zobrazení testovacího scénáře.
2. SYSTÉM zobrazí testovací scénář.
   (smazání testovacího)
   (správa odkazů pro vzdálený přístup)

Koncový stav:

- -

Alternativy
2a: Uživatel přejmenovává testovací scénář

2a1. SYSTÉM zobrazí testovací scénář.

2a2. UŽIVATEL zadá nový název testovacího scénáře.

2a3. SYSTÉM uloží nový název testovacího scénáře.

Alternativní koncový stav:

- Testovací scénář je přejmenován.

Smazání tesovacího scénáře
ID: 7
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa testovacího scénáře, ID: 6).*

Hlavní scénář:

1. UŽIVATEL zažádá SYSTÉM o smazání scénáře.
2. SYSTÉM se UŽIVATELE dotáže, zda chce opravdu množinu parametrů smazat.
3. UŽIVATEL potvrdí smazání scénáře.
4. SYSTÉM smaže příslušný testovací scénář.

Koncový stav:

- Testovací scénář je smazán.

Alternativy
3a: Uživatel zamítá smazání testovacího scénáře

3a1. UŽIVATEL zamítá smazání scénáře.

3a2. SYSTÉM neprovede smazání testovacího scénáře.

Alternativní koncový stav:

- Množina parametrů není smazána.

Správa speciálního odkazu pro vzdálený přístup
ID: 8
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa testovacího scénáře, ID: 6).*

Hlavní scénář:

1. UŽIVATEL zažádá systém o zobrazení stavu aktivace odkazu.
2. SYSTÉM zobrazí stav aktivace odkazu.
3. UŽIVATEL změní stav aktivace odkazu.
4. SYSTÉM uloží změnu.

Koncový stav:

- Stav aktivace odkazu je změněn

Alternativy
2a: Uživatel nemění stav odkazu, pouze se informuje o tomto stavu

2a1. SYSTÉM zobrazí stav aktivace odkazu.

2a2. UŽIVATEL prohlídne stav.

Alternativní koncový stav:

- Stav aktivace odkazu není změněn.

Správa konfigurace
ID: 9
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa množiny parametrů, ID: 3).*

Hlavní scénář:

1. UŽIVATEL zažádá systém o zobrazení konfigurace.

2. SYSTÉM zobrazí konfiguraci.
   (pokročilé generování)
   (smazání konfigurace)
   (správa množiny interakce)

Koncový stav:

- -

Alternativy
2a: Uživatel přejmenovává konfiguraci

2a1. SYSTÉM zobrazí konfiguraci.

2a2. UŽIVATEL zadá nové jméno konfigurace.

2a3. SYSTÉM uloží nové jméno konfigurace

Alternativní koncový stav:

- Konfigurace je přejmenována.

Generování testovacích množin
ID: 10
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa konfigurace, ID: 9).*

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o vygenerování nové množiny parametrů dle zvolené množiny parametrů v kontextu dané konfigurace.
2. SYSTÉM vygeneruje testovací scénář v daném kontextu.
3. SYSTÉM informuje uživatele o vygenerování testovacího scénáře.

Koncový stav:

- Je vygenerován nový testovací scénář. Tento scénář je zařazen k setu parametrů, ze kterého byl vygenerován.

Alternativy
2a: Při generování scénáře vyvstanou chyby

2a1. SYSTÉM při generování scénářů zaznamená chybu.

2a2. SYSTÉM oznámí toto selhání uživateli.

Alternativní koncový stav:

- Testovací scénáře nejsou vygenerovány.

Správa množiny interakce
ID: 11
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa konfigurace, ID: 9).*

Hlavní scénář:

1. UŽIVATEL zažádá systém o zobrazení množiny interakce.
2. SYSTÉM zobrazí množinu interace.
   (smazání množiny interakce)

Koncový stav:

- -

Alternativy
2a: Uživatel přejmenovává množinu inetrakce

2a1. SYSTÉM zobrazí množinu interace.

2a2. UŽIVATEL zadá nové jméno setu interakce.

2a3. SYSTÉM uloží nové jméno množiny interakce

Alternativní koncový stav:

- Konfigurace je přejmenována.

Alternativy
2b: Uživatel mení sílu interakce dané množiny

2b1. SYSTÉM zobrazí množinu interace.

2b2. UŽIVATEL zadá novou hodnotu síly interakce pro danou množinu.

2b3. SYSTÉM uloží novou sílu interakce.

Alternativní koncový stav:

- Síla interakce je změněna.

Smazání množiny interakce
ID: 12
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa konfigurace, ID: 11).*

Hlavní scénář:

1. UŽIVATEL zažádá systém o smazání množiny interakce.
2. SYSTÉM se dotáže UŽIVATELE, zda che množinu opravdu smazat
3. UŽIVATEL potvrdí smazání množiny interakce.
4. SYSTÉM smaže množinu interakce.

Koncový stav:

- Množina interakce je smazána, ne však odkazované parametry!

Alternativy
3a: Uživatel zamítá smazání množiny interakce

3a1. UŽIVATEL zamítne smazání množiny interakce.

3a2. SYSTÉM ponechá množinu interakce v systému.

Alternativní koncový stav:

- Množina interakce není smazána.

Smazání konfigurace
ID: 13
Výchozí stav:

- *Dědí se vstupní podmínky rozšiřovaného případu užití (Správa konfigurace, ID: 9).*

Hlavní scénář:

1. UŽIVATEL zažádá SYSTÉM o smazání konfigurace.
2. SYSTÉM se UŽIVATELE dotáže, zda chce opravdu konfiguraci smazat.
3. UŽIVATEL potvrdí smazání konfigurace.
4. SYSTÉM smaže příslušnou konfiguraci.

Koncový stav:

- Konfigurace je smazána i se všemi množinami interakcí. Odkazované parametry skrze množiny interakcí zůstávají v systému.

Alternativy
3a: Uživatel zamítá smazání konfigurace

3a1. UŽIVATEL zamítá smazání konfigurace.

3a2. SYSTÉM neprovede smazání konfigurace.

Alternativní koncový stav:

- Konfigurace není smazána.

Přístup k testovacímu scénáři skrze speciální odkaz
ID: 14
Výchozí stav:

- -

Hlavní scénář:

1. UŽIVATEL provede HTTP GET požadavek na adresu odkazu.
2. SYSTÉM vrátí v těle odpovědi CSV data scénáře.

Koncový stav:

- -

Alternativy
2a: Odkaz není aktivní či neexistuje zdroj, na který odkazuje

2a1. SYSTÉM vratí chybovou hlašku v případě neaktivního či neexistujícího zdroje.

Alternativní koncový stav:

- -

Přepnutí projektu
ID: 15
Výchozí stav:

- UŽIVATEL je přihlášen do systému.

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o přepnutí projektu.

2. -> Zobrazení seznamu projektů (ID: 16)
3. UŽIVATEL zvolí jiný projekt.
4. -> Správa projektu (ID: 1)

Koncový stav:

- Systém používá jiný projekt jako aktuální.

Alternativy
3a: Uživatel nevybere jiný projekt

3a1. UŽIVATEL nezvolí jiný projekt. (Zůstává u stávajícího.)

Alternativní koncový stav:

- Systém používá stejný projekt jako aktuální.

Zobrazení seznamu projektů
ID: 16
Výchozí stav:

- UŽIVATEL je přihlášen do systému.

Hlavní scénář:

1. UŽIVATEL požádá SYSTÉM o zobrazení projektu.
2. SYSTÉM vylistuje UŽIVATELI projekty a zobrazí je.

Koncový stav:

- -

1.2.2.2 Mapování případů užití na požadavky

Abychom měli jistotu, že vytvořené případy užití skutečně všechny pokrývají požadavky kladené na systém, provedli jsme kontrolu pokrytí požadavků jednotlivými případy užití. Výsledek kontroly je uveden v tabulce 1.1. Všechny požadavky kladené na funkcionalitu systému jsou pokryté definovanými uživatelskými scénáři.

1.3 Návrh uživatelského rozhraní

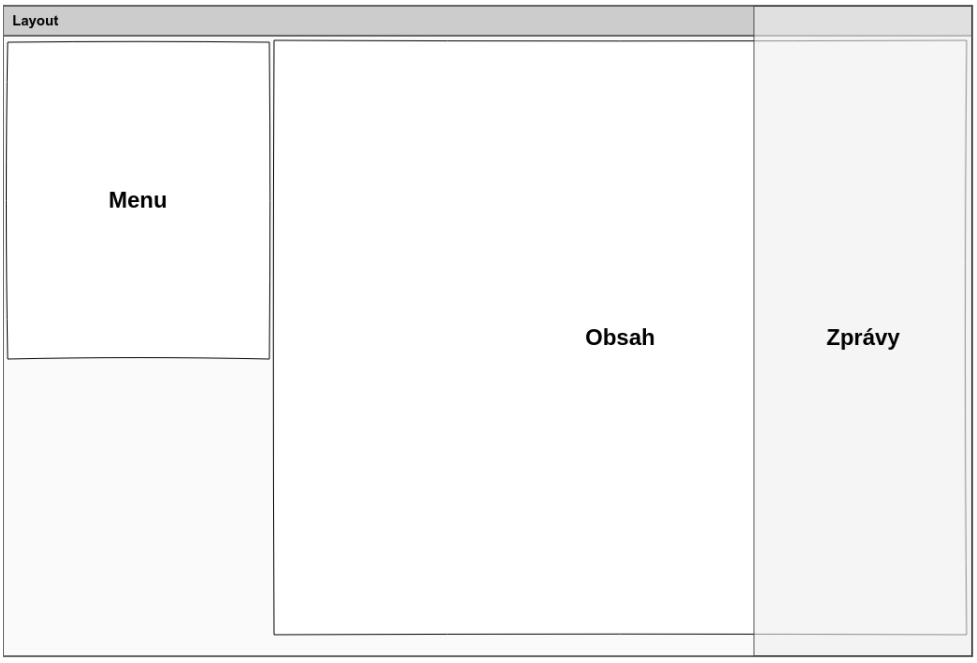
Zde se čtenář seznámí s návrhem grafického rozhraní, který bude předlohou pro výsledný systém. Nejprve navrhneme základní layout, do kterého následně zasadíme specifické komponenty jednotlivých obrazovek.

Případy užití \ Požadavky	Generování testovacích kombinací	Přístup k testovacím datům pomocí speciálního API	Projekt je možné editovat	Projekt obsahuje více definic úlohy	Různé úrovně pokrytí	Vytvoření projektu
Generování testovacích množin	x				x	
Pokročilé generování test. množin...	x			x	x	
Přepnutí projektu			x			x
Přístup k testovací množině přes API		x				
Správa konfigurace			x	x	x	
Správa speciálního odkazu		x	x			
Správa projektu	x		x			
Správa množin interakce	x	x	x		x	
Správa množin parametrů			x			
Správa testovacích množin		x	x			
Zobrazení seznamu projektů			x			

Tabulka 1.1: Mapování případů užití na požadavky

1.3.1 Základní layout

Layoutem se rozumí základní rozvržení hlavních (společných pro téměř všechny obazovky) komponent na obrazovce systému. Těmito komponentami je menu (navigace), pánev pro zobrazení hlavního (aktuálního) obsahu a místo pro notifikaci uživatele. Tato notifikační plocha částečně překrývá pánev pro hlavní obsah, jelikož tyto notifikace v čase mizí a tato oblast je tak využívána jen příležitostně. Překrytí hlavního obsahu je v zásadě i žádoucí, jelikož tak spíše neunikne pozornosti uživatele. Layout vyvíjeného systému je zobrazen na obrázku 1.3.



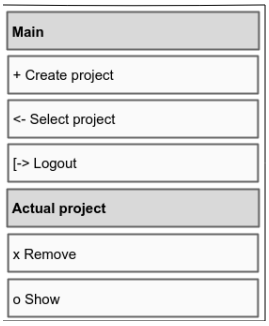
Obrázek 1.3: Návrh základního layoutu

1.3.2 Menu

Menu nabízí hlavní či časté ovládací prvky aplikace. Jednotlivé obrazovky zde mohou přidávat akce pro ně exkluzivní. Obsah se může měnit i na stavu systému obecně. Grafický návrh je k vidění na obrázku 1.4.

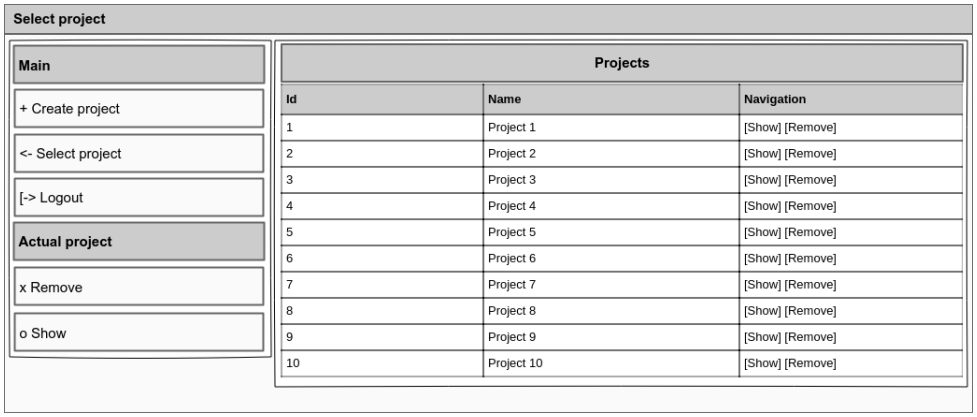
1.3.3 Seznam projektů

Obrazovka slouží k nastavení aktuálního pracovního projektu a k přepínaní mezi nimi. Z tohoto seznamu je možné i přímé smazání projektu. Nastavení aktuálního projektu mění stav systému. Pokud je nastaven aktuální projekt,



Obrázek 1.4: Návrh menu aplikace

jsou v menu zobrazeny akce pro tento projekt (viz sekci Menu 1.3.2). Tato obrazovka je zobrazena na obrázku 1.5.



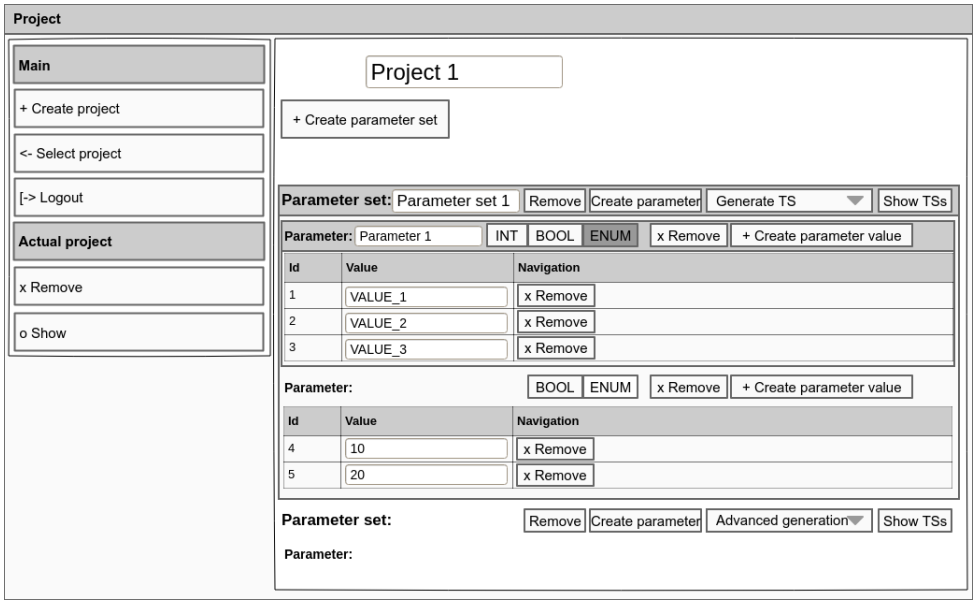
Obrázek 1.5: Návrh obrazovky seznamu projektů

1.3.4 Projekt

Obrazovka informuje uživatele o stavu projektu. Umožňuje mu přejmenovat projekt a spravovat obsažené množiny parametrů (zadání úlohy) a tyto parametry i s jejich hodnotami. Na téže stránce lze vygenerovat základní typ testovacího scénáře. Stránka projektu je vyobrazena na obrázku 1.6

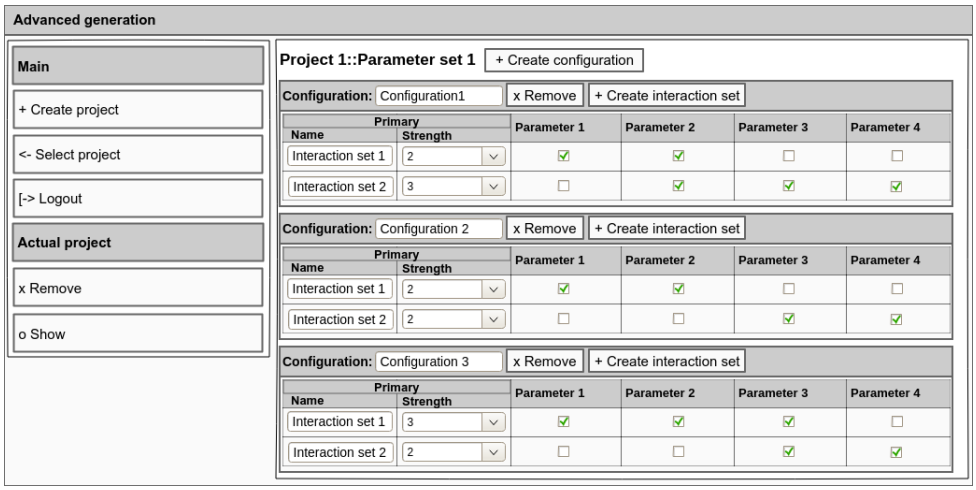
1.3.5 Konfigurace pro pokročilé generování testovacích scénářů

Z této stránky lze vygenerovat scénáře se sofistikovanější mírou pokrytí. Na obrazovce pokročilého generování lze spravovat dodatečné specifikace zadání úlohy, které slouží právě pro toto generování. Těmito parametry jsou konfi-



Obrázek 1.6: Návrh obrazovky projektu

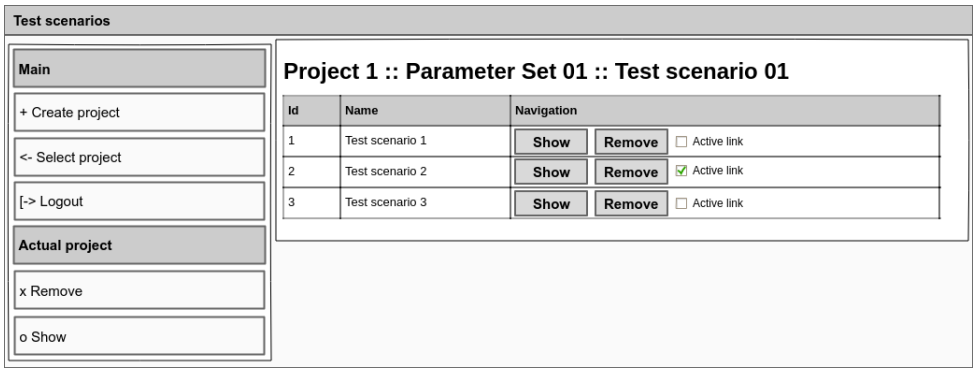
gurace spravující tzv. množiny interakcí. Návrh obrazovky je na obrázku 1.7.



Obrázek 1.7: Návrh obarzovky s konfiguracemi pro pokročilé genrování testovacích scénářů

1.3.6 Přehled testovacích scénářů

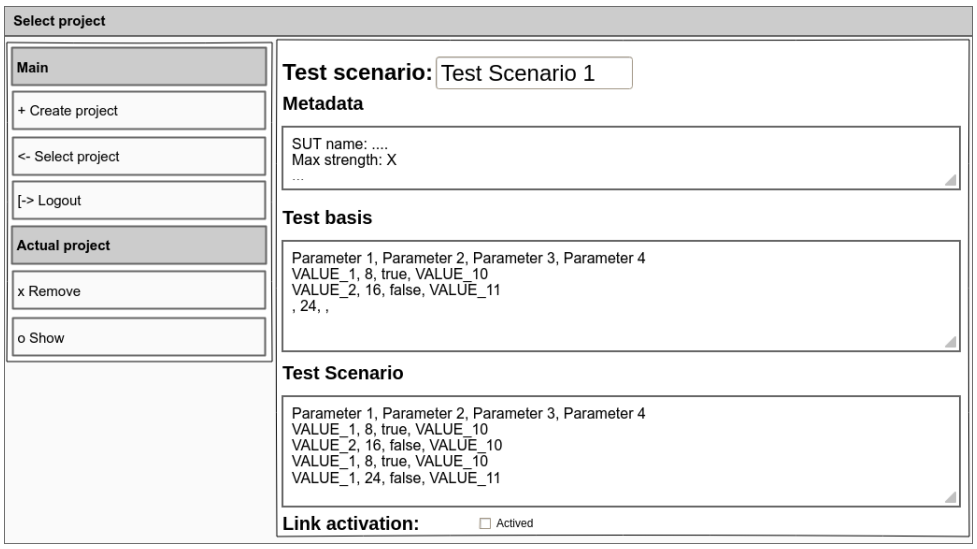
Tato obrazovka slouží k přehledu vygenerovaných scénářů pro danou množinu parametrů. Lze z ní aktivovat či deaktivovat speciální odkaz pro vzdálený přístup k tomuto scénáři. Stránka je vizualizována na obrázku 1.8.



Obrázek 1.8: Návrh obrazovky přehledu testovacích scénářů

1.3.7 Testovací scénář

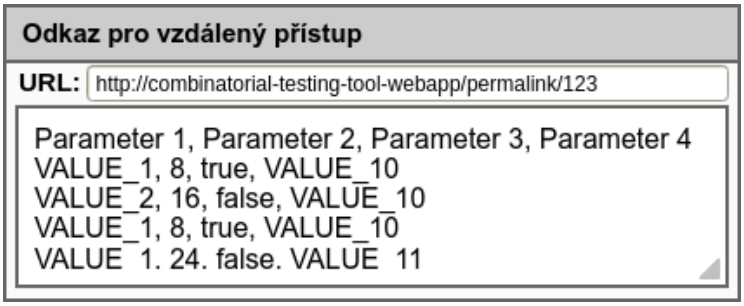
Zobrazuje informace týkající se vygenerovaného scénáře. Obsahuje vygenerovaný scénář, ale i původní množinu, ze které byl scénář vygenerován. Dále obsahuje metadata vztahující se k těmto prvkům. Stránka testovacích scénářů je na vyobrazena a obrázku 1.9.



Obrázek 1.9: Návrh obrazovky testovacích scénářů

1.3.8 Speciální odkaz na přístup k testovacím scénářům

Speciální odkaz na přístup k testovacím scénářům zobrazí testovací kombinace pro množinu scénářů, která má povolen vzdálený přístup. Nejedná se přímo o grafický návrh, ale pro úplnost je přítomen. Vyobrazen je na obrázku 1.10.



Obrázek 1.10: Použití speciálního odkazu na přístup k testovacím scénářům

1.4 Doménový model

Doménový model znázorňuje strukturu dat, se kterými daný systém pracuje. Definuje entity s jejich atributy a vztahy mezi těmito entitami a někdy i metody těchto objektů. Entity reprezentují objekty reálného světa a atributy vlastnosti těchto entit. Speciálním případem může být vztahová entita. Takové entity vznikají pokud chceme vztahu mezi entitami přiřadit nejaký atribut.

Doménový model se dělí na tři typy - návrhový (konceptuální). technický (logický) a fyzický (implementační). Poslední dva typy modelu mohou být zachyceny jedním diagramem.

Návrhový model je blíže entitám rálného světa. Popisuje tyto entity s jejich vlastnostmi a vztahy mezi nimi. Mapuje problematiku řešené domény a nebere v potaz, zda bude výsledkem softwareový produkt nebo úřední formulář. Dovoluje nejvyšší míru abstrakce.

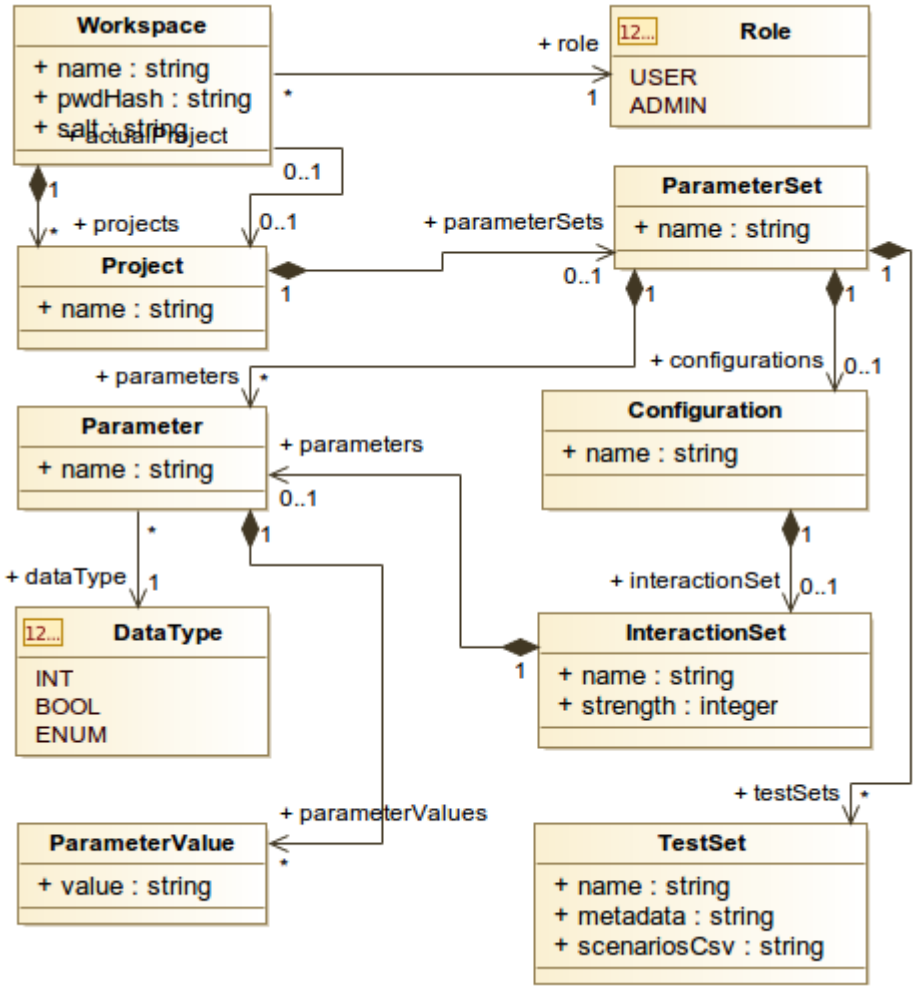
Technický model již bere v potaz obor, v kterém bude problém řešen, ale neřeší jak konkrétně. V našem případě to například znamená, že víme, že řešením bude orientované programování (OOP), ale neřešíme konkrétní jazyk. Je to tedy platformně nezávislé řešení.

Fyzický model již bere v potaz konkrétní platformu. Nedovoluje již žádnou abstrakci, jelikož se jedná o přímou předlohu pro psaní výsledného kódu.

Obecným trendem je, že se snižující se mírou abstrakce model roste či zůstane stejný, nebo by to člověk alespoň čekal. Avšak může docházet i k opačnému trendu. Návrhový model se totiž snaží co nejlépe seznámit s problematickou doménou a tedy může entity mnohem více dekomponovat v rámci lepšího

pochopení problému. Při implementaci či technickém návrhu však může vyjít, že taková granularita není potřebná a pouze navyšuje počet řádků, respektive počet tříd.

V této sekci si uvedeme první typ modelu a sice model návrhový (viz obrázek 1.11). Technický pohled bude vynechán, jelikož pro naše potřeby vystačíme s modelem fyzickým.



Obrázek 1.11: Konceptuální datový model systému

Workspace, nebo-li pracovní plocha, je třída reprezentující uživatelský účet. Obsahuje kredentiály pro přihlášení do systému a uživatelem spravované projekty. V rámci uživatelského komfortu je shraňován aktuální projekt, který může být automaticky otevírán po opětovném přihlášení do sytému.

Role je výčet možných uživatelských rolí. Role určuje operace, ke kterým je uživatel autorizován.

Project je třída, v rámci které jsou spravovány jednotlivé množiny parametrů reprezentované třídou *ParameterSet*.

ParameterSet je třída shraňující jednotlivá zadání úloh. Zadání úlohy je v základu definováno jednotlivými parametry(třída *Parameter*), ale může být rozšířeno o vztahy mezi těmito parametry, takzvanými konfiguracemi (třída *Configuration*). Těchto konfigurací může být více.

Parameter obsahuje množinu hodnot, ze kterých budou tvořeny výsledné kombinatorické scénáře. Každý parametr má určen datový typ (*DataType*).

DataType je výčet datových typů, které můžou parametry(instane třídy *Parameter*) reprezentovat.

ParameterValue je jedna z možných hodnot, kterých může daný parametr (třída *Parameter*) nabývat.

Configuration definuje vztahy mezi jednotlivými parametry z dané úlohy (třída *ParameterSet*).

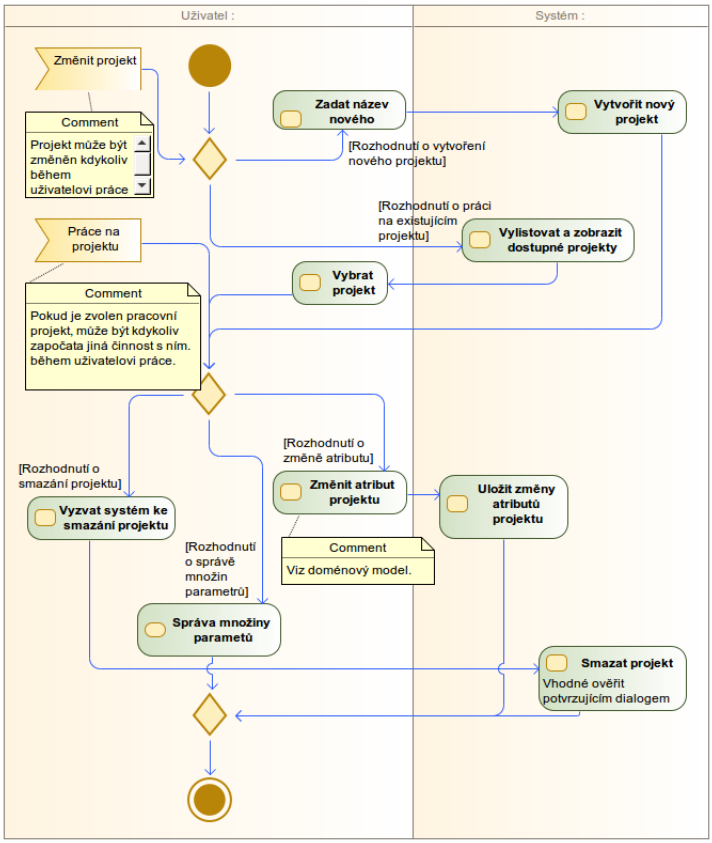
IntreractionSet je členská třída konfigurace (třída *Configuration*), určuje submnožinu parametrů (třída *Parameter*), pro které definuje sílu interakce (viz Pokročilé generování testovacích kombinací).

1.5 Základní procesy v systému

Základní procesy v systému jsou modelovány pomocí diagramů aktivit. Diagram aktivit patří do rodiny UML diagramů a slouží k popisuju business procesů systému pomocí orientovaného grafu. Zachycuje interakce uživatele a systému, popřípadě interakce uživatelů mezi sebou. Diagram aktivit definuje jednotlivé úkony v procesu, které se dělí na aktivity a akce, kdy aktivita je činnost, kterou lze dále škálovat a rozpadne se tak na jednotlivé atomické akce. Atomicitu je v tomto ohledu dobré chápat spíše jako míru detailu, v jakém se autor snaží systém zachytit, nežli další reálnou nedělitelnost. Posloupnost kroků je dána hranami mezi těmito činnostmi a cesta v tomto grafu reprezentuje jedno z možných (viz dále) usktutečnění celého procesu. Každá cesta začíná počátečním vrcholem nebo přijetím signálu a končí vrcholem regulárního konce, předčasného ukončení nebo vysláním signálu. Bylo naznačeno, že cest může být více, diagram aktivit totiž může obsahovat větvící body, kde se dle podmínky může chod ubíhat různými směry, případně se zase spojovat. Zvláštním případem větvení je paralelní region, kde probíhají všechny cesty

současně. Na konci paralelního regionu je bariéra, kde se čeká na doběhnutí všech průchodů. Pro identifikaci, se aktivity (či akce) umisťují do specialních bloků (tzv. swimline).

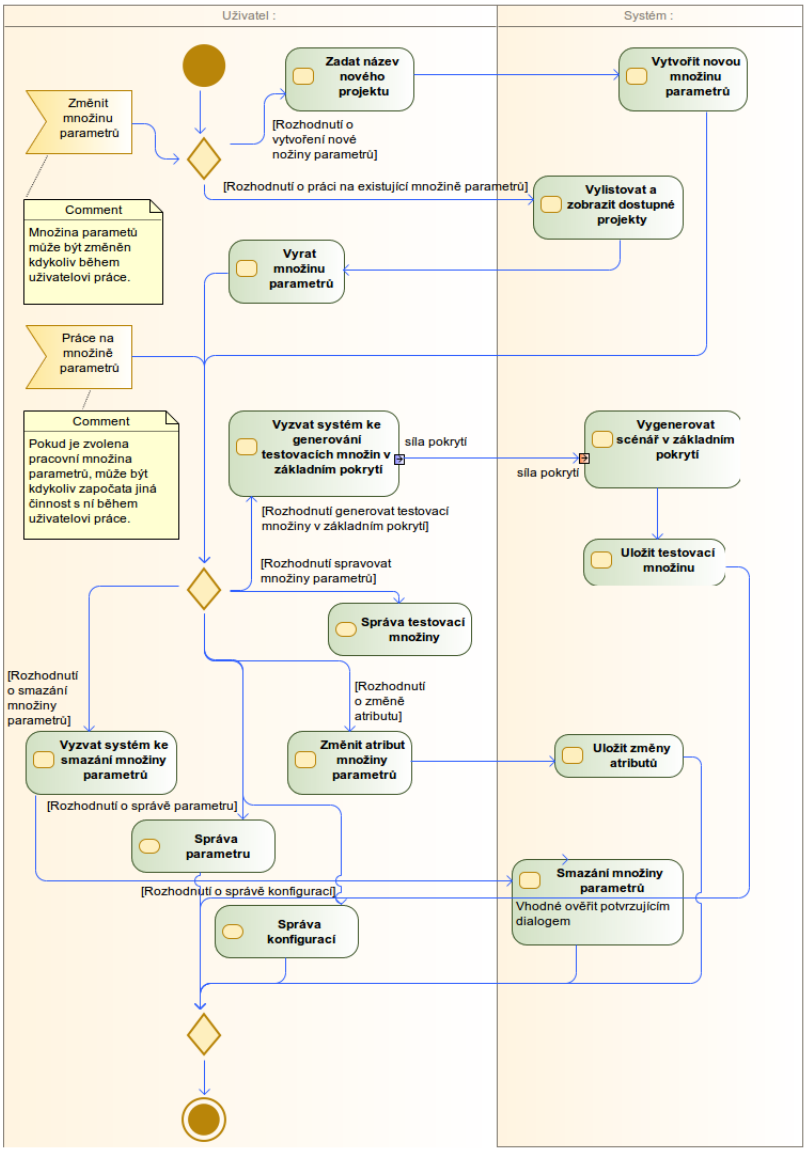
Proces správy projektu zobrazuje hlavní průchod systémem. Specifikuje proces získání projektu, který lze získat buď vytvořením nového projektu nebo načtením některého z rozpracovaných. Krom dalších upřesnění, které lze snadno vyčíst, také odkazuje na další subprocesy, které upřesňují jednotlivé aktivity. Tento diagram odkazuje konkrétně na *proces správy množiny parametrů* který je zachycen na obrázku 1.13.



Obrázek 1.12: Proces správy projektu

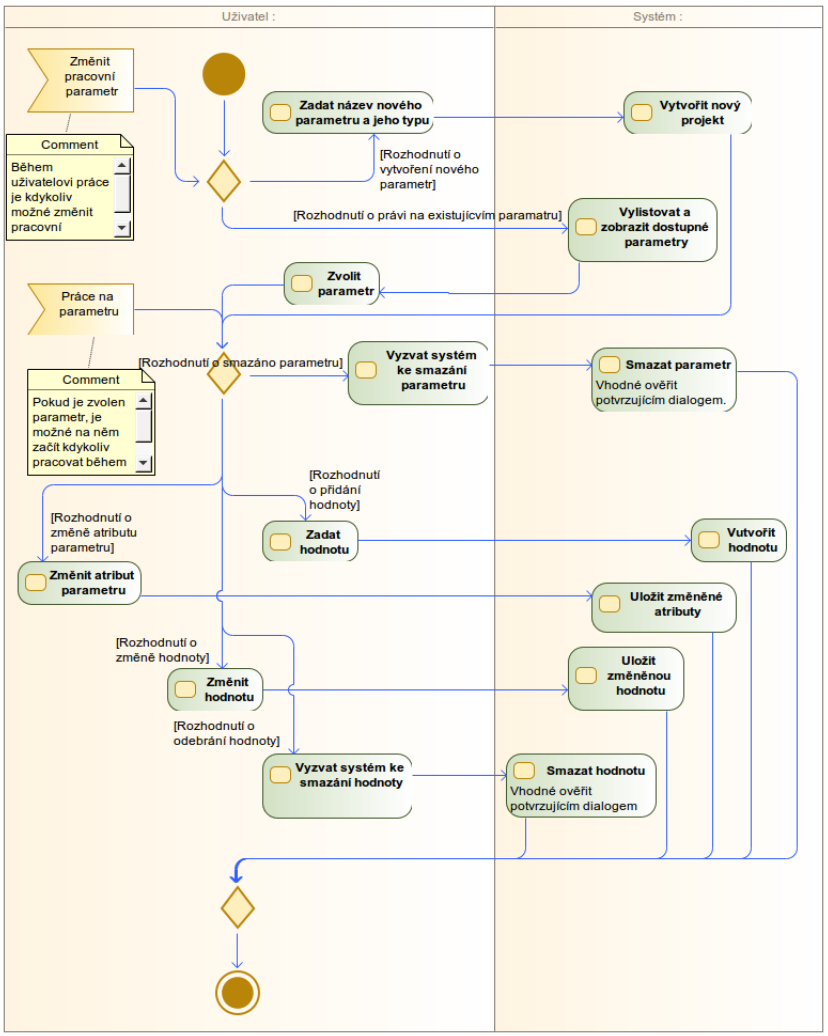
Proces správy množiny parametrů specifikuje aktivitu odkazovanou z *procesu správy projektu* (obrázek 1.12). Vysvětluje v jakém postupu se přidávají jednotlivé parametry, jak se generují výsledné testovací množiny, přímo pro základní verzi či nepřímo skrze delogovaný *proces správy konfigurací* (obrázek 1.15) a dále jak se (skrze *proces správy testovacích množin*) (obrázek 1.17) s těmito testovacími množinami pracuje.

Proces správy parametru ukazuje aktivity, které je možné provádět s



Obrázek 1.13: Proces správy množiny parametrů

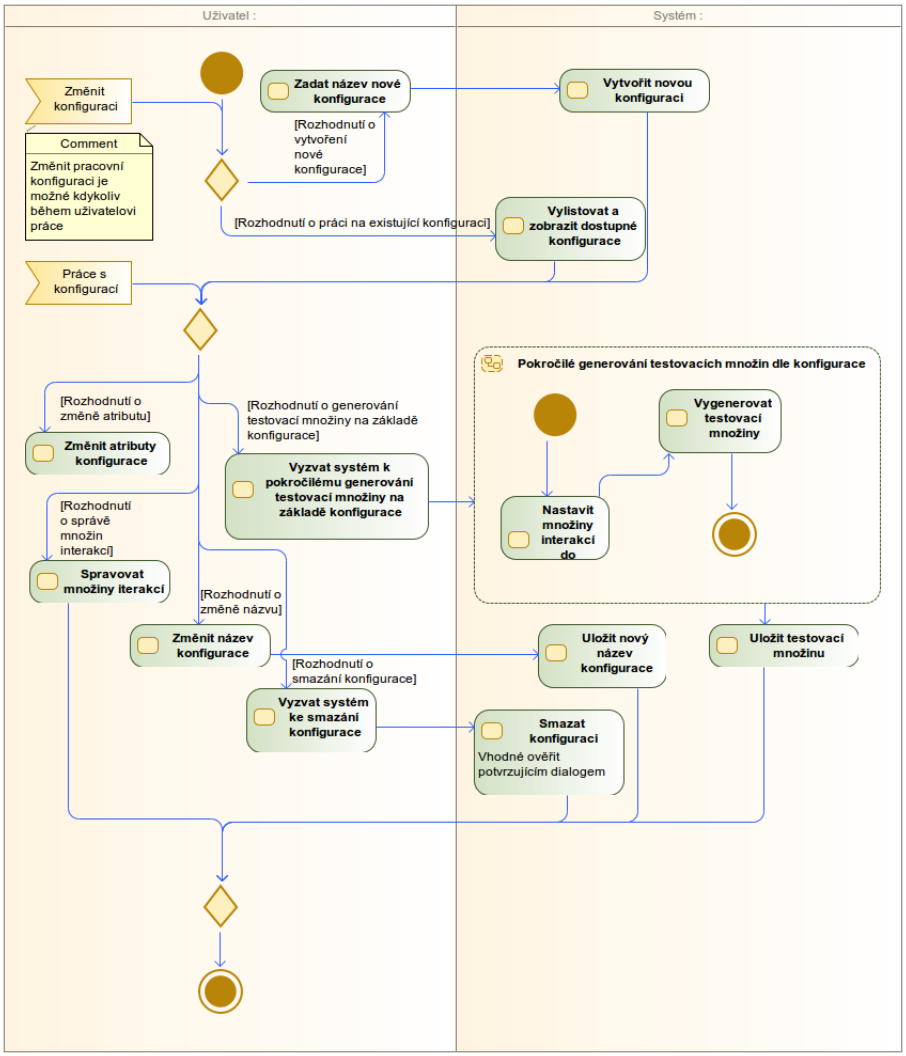
jednotlivými parametry a jejich hodnotami. Tento diagram obsahuje pouze atomické akce a nikam tedy nedeleguje.



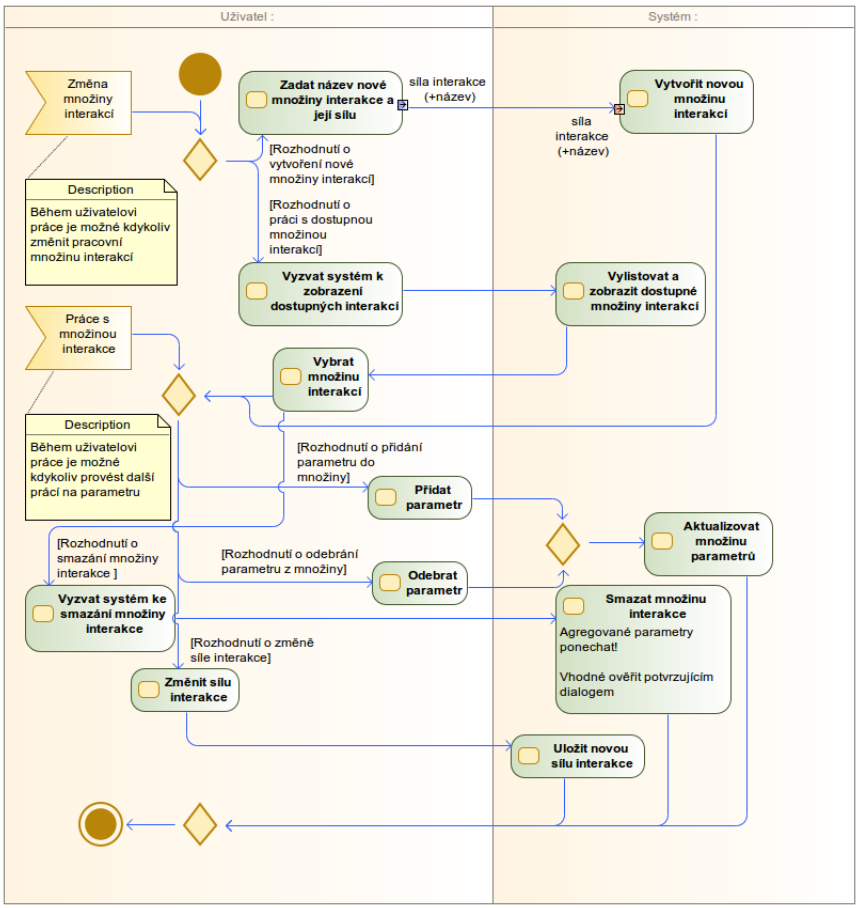
Obrázek 1.14: Proces správy parametru

Proces správy konfigurace je odkazován z *Procesu správy množiny parametrů* (obrázek 1.13) a ukazuje způsob rozšířeného zadání úlohy, které je dále upřesněno v *procesu správy množin interakcí* (obrázek 1.16). Také umožňuje *proces pokročilého genrování scénářů*, jenž odkazuje na subproces, který je v této situaci pro svou jednoduchost uveden přímo.

Proces správy interakce rozšiřuje *proces správy konfigurace* (obrázek 1.15) a upřesňuje tvorbu množin interakcí, které jsou stěžejními prvky celé konfigurace. Upřesňuje proces mazání množiny interakce, kde je velmi důležité, aby odkazované parametry nebyly v žádném případě mazány rekurzivně.



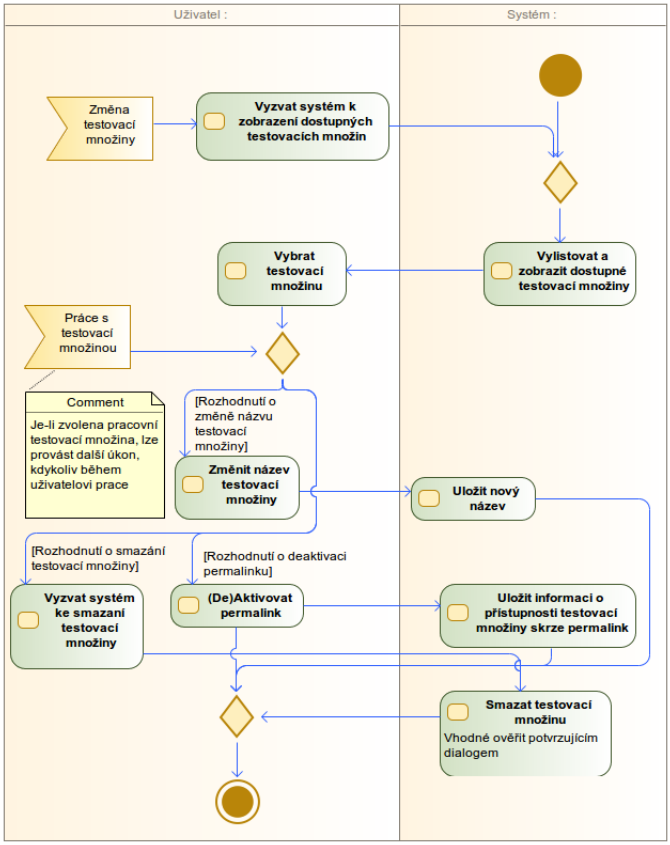
Obrázek 1.15: Proces správy konfigurace



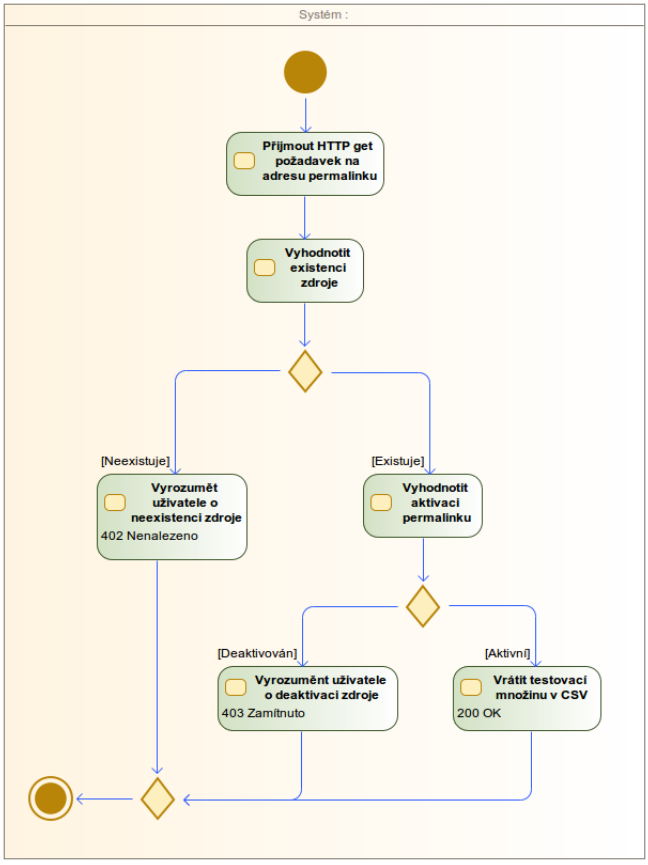
Obrázek 1.16: Proces správy množiny interakce

Proces správy testovací množiny je odkazován z *procesu správy množiny parametrů* (obr. 1.13) a byť je tvořen atomickými akcemi, tak nepřímo souvisí s *procesem vyřízení požadavku na speciální odkaz* (obrázek 1.18), jehož průchod ovlivňuje a to (de)aktivací speciálního odkazu.

Proces vyřízení požadavku na speciální odkaz (obrázek 1.18) ukazuje zpracování tohoto požadavku a nastiňuje jak má systém reagovat. Jeho rozhodnutí souvisí s *procesem správy testovací množiny* (obrázek 1.17).



Obrázek 1.17: Proces správy testovací množiny



Obrázek 1.18: Proces vyřízení požadavku na speciální odkaz pro vzdálený přístup

Kapitola 2

Realizace

V této kapitole jsou uvedeny technické detaily z implementace systému - použité technologie, popis technické realizace interních procesů v systému a model nasazení systému, zároveň vysvětlující i architekturu základních komponent.

2.1 Použité technologie

V této kapitole budou shrnuty použité technologie. Čtenář se seznámí s použitými technologiemi, dozví se, co tyto technologie nabízí a důvod jejich použití v projektu.

2.1.1 JavaServer Faces

JavaServlet Faces (JSF) je java-based framework usnadňující vývoj grafického rozhraní webových aplikací. Nejedná se však o ryze grafický framework, jeho koncept zahrnuje celou MVC architekturu. Konkrétně se jedná komponentově orientovanou MVC architekturu. JSF je postaven na základě tvořeným Servlet API.

2.1.1.1 Facelets

Facelety jsou Extensible Hypertext Markup Language (XHTML) soubory, tedy soubory které kombinují čiste HTML spolu s dalšími tagy, které jsou na pozadí párovány s javovským kódem. Od verze JSF 2.0 nahrazuje technologii JavaServer Pages (JSP).

Tyto rozšiřující tagy jsou brány z knihoven jmenovaných v následující přehledové tabulce 2.1.

2.1.1.2 Primefaces

Framework PrimeFaces[17] je rozšířením základní Facelets knihovny tagů. Jeho úkolem je ještě více ulehčit tvorbu uživatelského rozhraní(UI). Jeho vy-

Knihovna	URI	Zavedený prefix	Popis třídy tagů
JSF Facelets Tag Library	http://java.sun.com/jsf/facelets	ui:	Tvorba šablon
JSF HTML Tag Library	http://java.sun.com/jsf/html	h:	Komponenty uživatelského rozhraní
JSF Core Tag Library	http://java.sun.com/jsf/core	f:	Funkcionality nezávislé na RenderKitu
JSTL Core Tag Library	http://java.sun.com/jstl/core	c:	Jádro JSTL – cykly, podmínky atd.
JSTL Functions Tag Library	http://java.sun.com/jstl/functions	fn:	JSTL funkce – např. toLowerCase atd.

Tabulka 2.1: Přehled knihoven tagů pro JSF

užití je velice snadné díky řade vzorových příkladů dostupných na webu vydavatele. Výsadou frameworku není jen tvorba komponent a layoutu, ale i stylizace rozhraní pomocí řady vizuelních šablon.

2.1.2 Context and Dependency Injection

Context and Dependency Injection (CDI) je technologie, jak již angličtiny znalý čtenář tuší, pro správu kontextu a vkládání závislostí. Spravuje životní cyklus kontextuálních objektů, takzvaných bean. Tyto beany zachycují buď stav aplikace anebo provádějí aplikační logiku. Při jejich inicializaci vzniká tzv. strom závislostí, kde závislostmi jsou další beany. V kostce řečeno, dle hierarchie stromu jsou závislosti vkládány do závislých objektů. Tyto závislosti jsou buď iniciovány nově, nebo vloženy z kontejneru, pokud již iniciovány byly.

2.1.2.1 Expression language

Expression language (EL) je mechanismus umožňující přístup k beanám z prezenční vrstvy, tedy webových stránek. Je tedy využíván Facelety. EL má taky definovanou řadu operátorů, například aritmetických či logických.

2.1.3 JPA

Java Persistence API (JPA) je specifikace objektově relačního zobrazení (ORM), lze je využít jak na J2EE platformě, tak i v SE (standar editon) aplikaci. Objekty spravované pomocí JPA se nazývají entity.

2.1.3.1 Objektově relační mapování

ORM je technika propojující dva světy. Svět relačních databází a svět objektového programování. Zatímco při objektovém modelování je hlavním cílem co nejpřesněji namodelovat realitu, databázoví inženýři se zaměřují na efektivitu ukládání dat. ORM má tedy za úkol nechat programátory zabývat se pouze svou prací.

2.1.3.2 Entita

Entita je plain old java object (POJO). Jedná se tedy z pravidla o jednoduchou datovou strukturu opatřenou příslušnými gettery a settery. Takový

objekt je označen anotací @Entity. Další anotace určují vztahy mezi jednotlivými entitami a další potřebné informace pro ORM. Těmito anotacemi jsou například *@OneToMany* určující kvantifikátor relace, *@Transient* označující nepersistovaný atribut nebo například *@Column* definující název sloupce v databázi.

2.1.3.3 Hibernate

Jedná se o framework umožňující ORM mapování. Poskytuje jak své proprietární rozhraní, tak i implementaci JPA. V projektu bude využito rozhraní JPA. [18]

2.2 Interní procesy v systému

Sekvenční diagramy popisují dynamické děje systému. Narozdíl od diagramů aktivit, které popisují spíše obecné business procesy, sekvenční diagramy popisují tyto děje z pohledu již reálných systémových komponent.

Následující diagramy popisují, až na dva (přepínaní projektů a inicializace dat pro view), akce vytváření konkrétních instancí entit datového modelu. Aby nedocházelo ke zbytečnému opakování stejné či velmi podobné skutečnosti. Uvedu tyto zde na začátku.

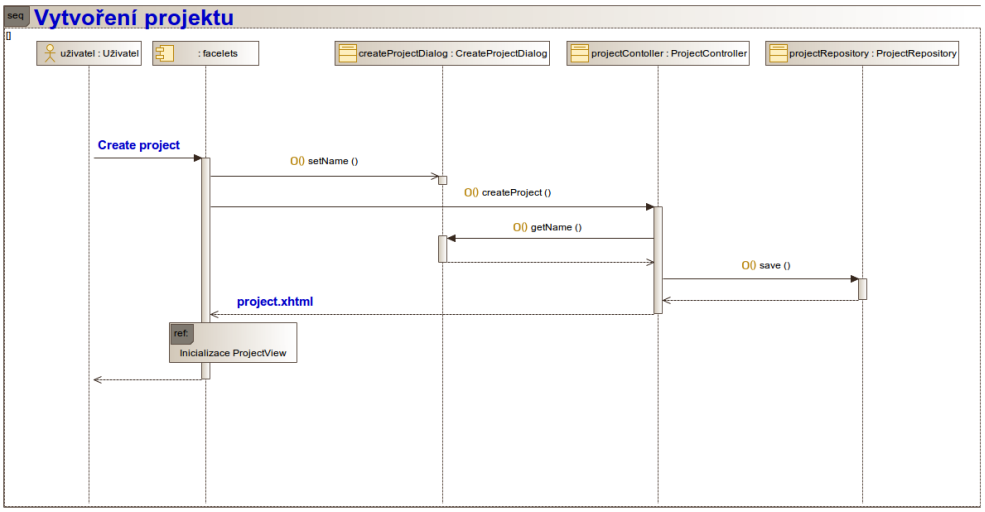
V diagramech jsou uživatelem iniciovány tvořící sekvence (*Create .../Generate test set*), Tyto akce mají za následek zobrazení dialogu s formulářem obsahujícím ruzná pole, která jsou mapována na příslušný datový objekt (*Create...Dialog/GeneratorDialog*). V průběhu vyplňování formuláře jsou hodnoty pomocí asynchroních operací nahrávany (pomocí setterů) do zmíněného objektu, odkud jsou následně při výsledném vytváření modelové entity vyzvedávány (pomocí getterů).

2.2.1 Vytvoření projektu

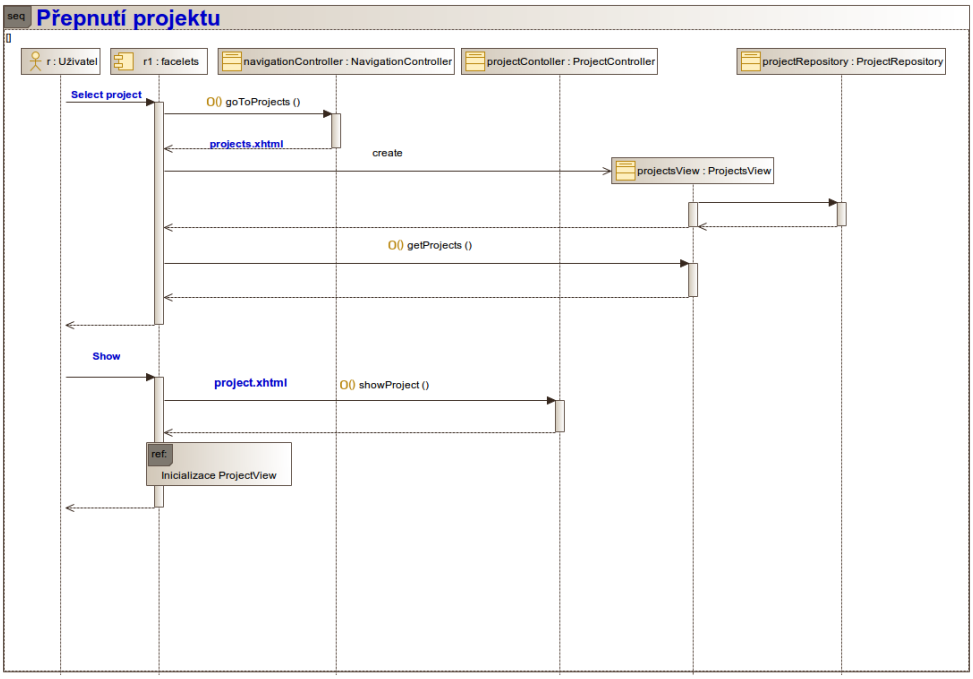
Následující diagram popisuje vytvoření projektu uživatelem. Proces začíná požadavkem na vytvoření projektu, kdy systém zobrazí dialog s příslušným formulářem. Po vyplnění a potvrzení požadavku je objekt persistován v příslušném repozitáři. Celý proces je zobrazen na obrázku 2.1.

2.2.2 Přepnutí projektu

Diagram popisuje přepnutí projektu, případně volbu projektu, není-li vybrán jako aktuální žádný. Uživatel nejprve zažádá systém o vylistování všech uživatelových projektů. Z tohoto seznamu posléze uživatel vybere jeden, s kterým chce pracovat. Systém tento projekt nastaví jako aktuální a přesměruje uživatele na příslušnou obrazovku projektu. Tento proces popisuje obrázek 2.2.



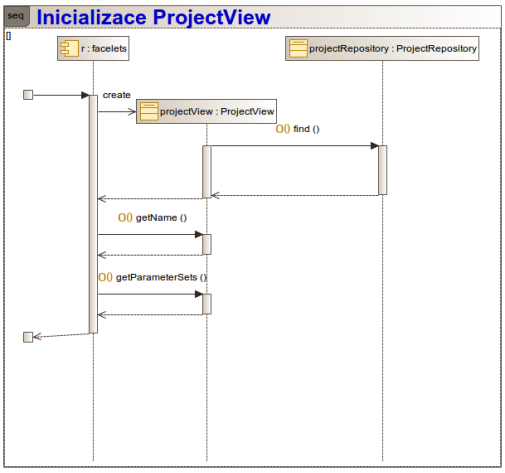
Obrázek 2.1: Interní proces systému při vytváření projektu



Obrázek 2.2: Interní proces systému při volbě pracovního projektu

2.2.3 Inicializace ProjectView

Diagram popisuje inicializaci datového objektu (třída ProjectView) obalujícího data zobrazená na projektové obrazovce. Tato akce je iniciována JSF frameworkem kdykoliv je uživatel nasměrován na obrazovku (*project.xhtml*) tento objekt (beanu) využívající. Interní proces systému je zobrazen na obrázku 2.3



Obrázek 2.3: Interní proces systému při inicializaci prezentovaných dat

2.2.4 Vytvoření množiny parametrů

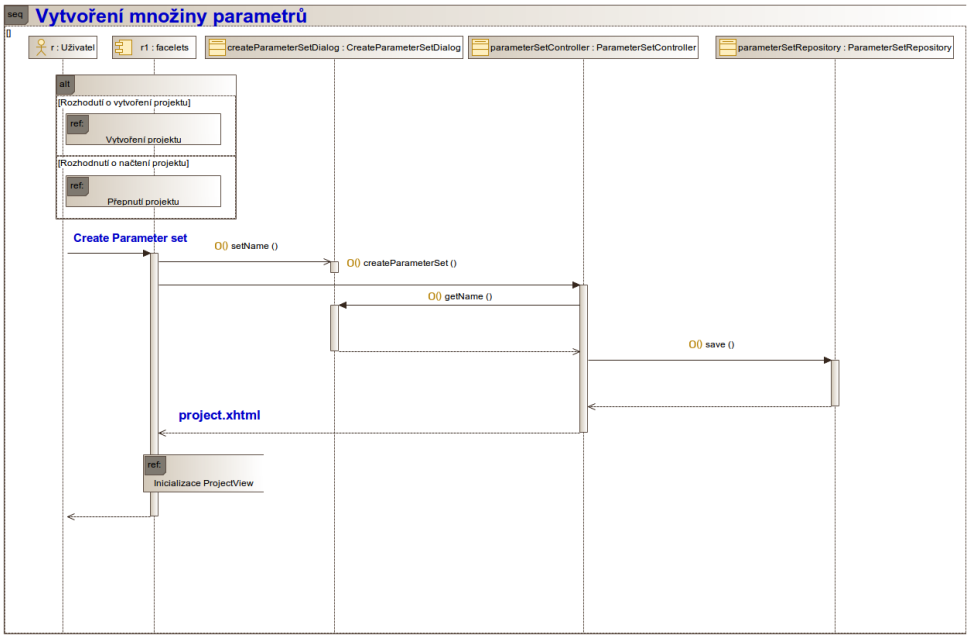
Pro vytvoření množiny parametrů musí být nastaven aktuální projekt, tak je učiněno při jeho přepnutí nebo po vytvoření nového. Uživateli je opět zobrazen příslušný dialog s formulářem. Vytvořený objekt je persistován do příslušného repozitáře. Po vytvoření množiny je reinicializována obrazovka s novými daty. Tuto sekvenci naleznete na obrázku 2.4.

2.2.5 Vytvoření parametru

Pro vytvoření parametru musí být přítomna alespoň jedna množina parametrů, do které může být tento parametr vložen. Tvoření opět probíhá skrze dialog s formulářem. Tuto sekvenci lze nahlédnout na obrázku 2.5

2.2.6 Vložení hodnoty parametru

Existuje-li v sytému parametr, je možné mu přiřadit hodnotu, které může nabývat. Tento proces je zachycen následujícím diagramem 2.6.



Obrázek 2.4: Interní proces systému při vytváření množiny prametrů

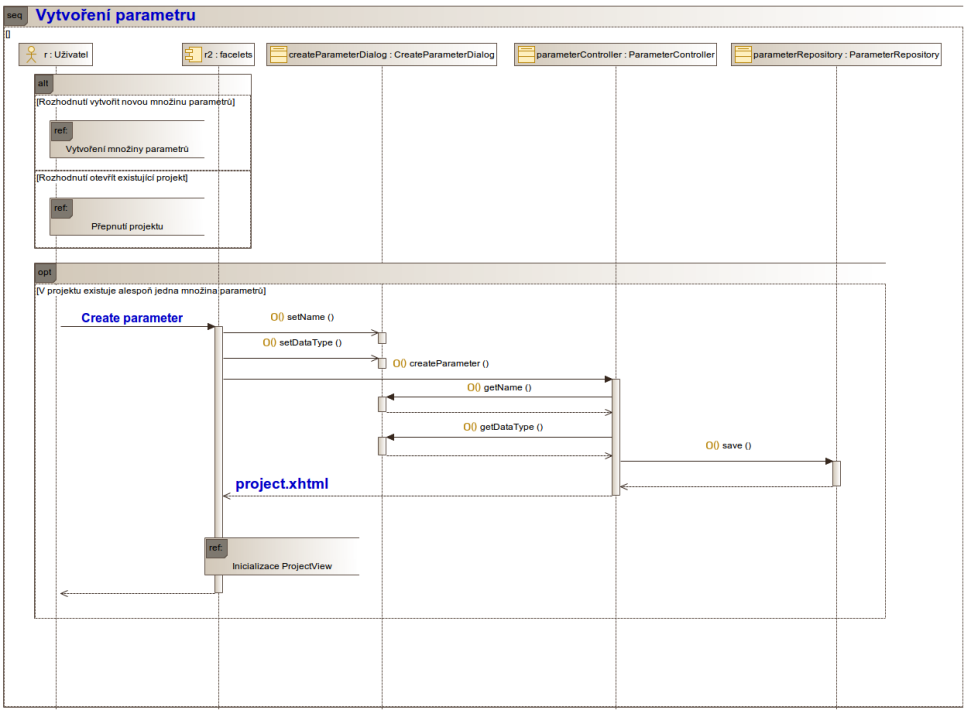
2.2.7 Generování testovacích množin v základním pokrytí se zvolenou silou interakce

Obsahuje-li množina parametrů validní parametry (parametry s neprázdnou množinou hodnot a hodnotami odpovídajícími datovému typu), může být z této množiny generována testovací množina. Jelikož tento proces může trvat delší dobu, probíhá proto na pozadí a o jeho dokončení je uživatel notifikován oznamující zprávou. Celý proces je znázorněn na obrázku 2.7.

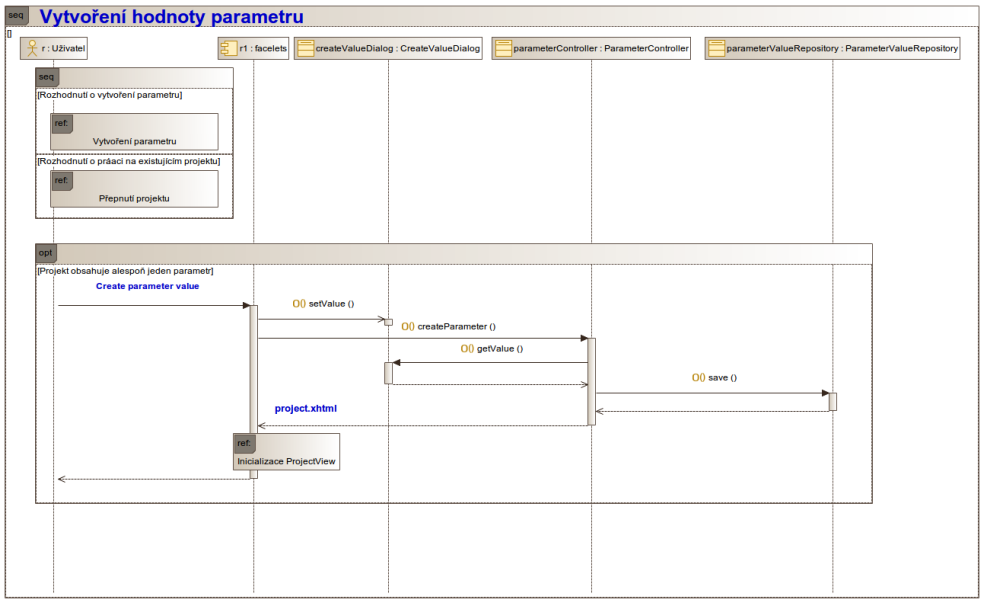
2.3 Model nasazení

Konfiguraci sytému při nasazení jsem znázornil diagramem nasazení, což je grafické zobrazení rozvržení hardwarových a softwarových komponent systému. Kromě samotného rozmístění dále vystihuje komunikační infrastukturu a závislosti mezi jednotlivými komponentami systému. Nasazení vyvinutého systému je zobrazeno na obrázku 2.8.

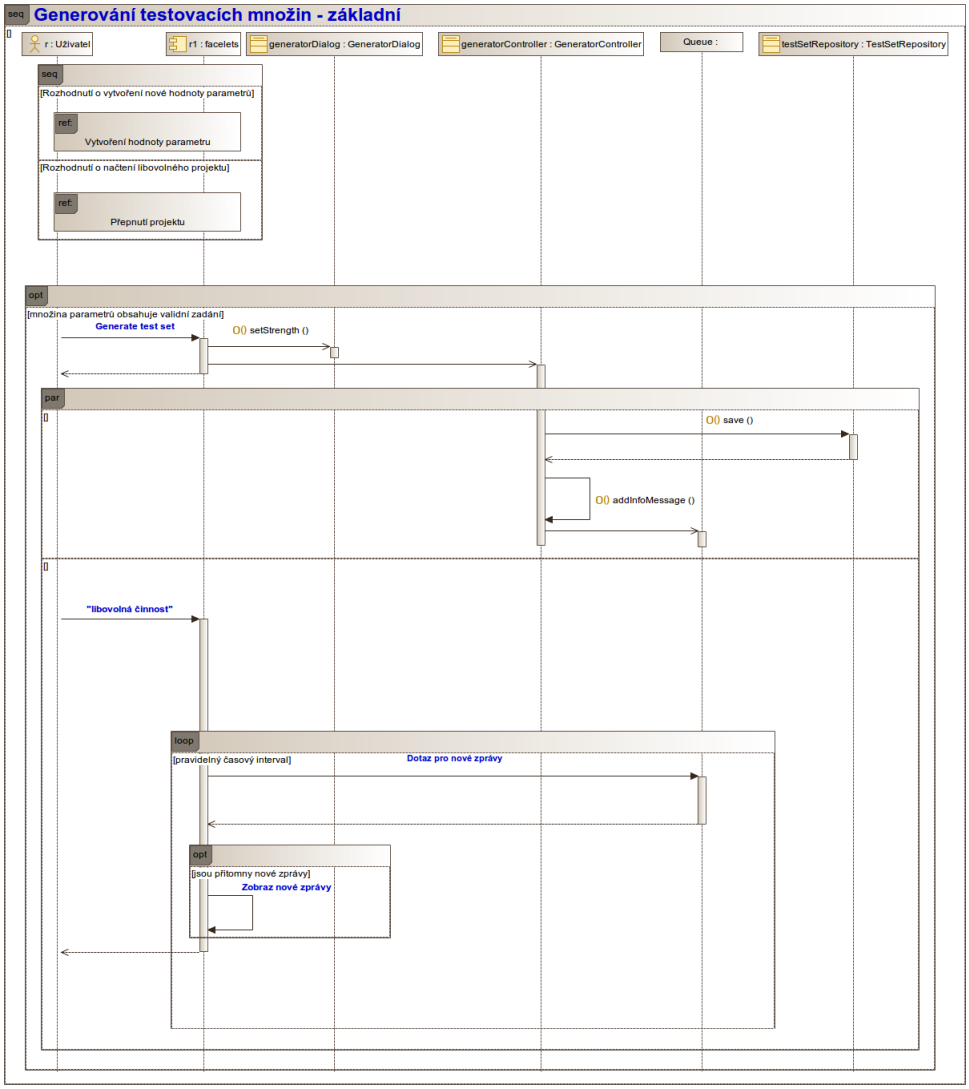
V následujících podsekcích popisuji jednotlivé použité komponenty.



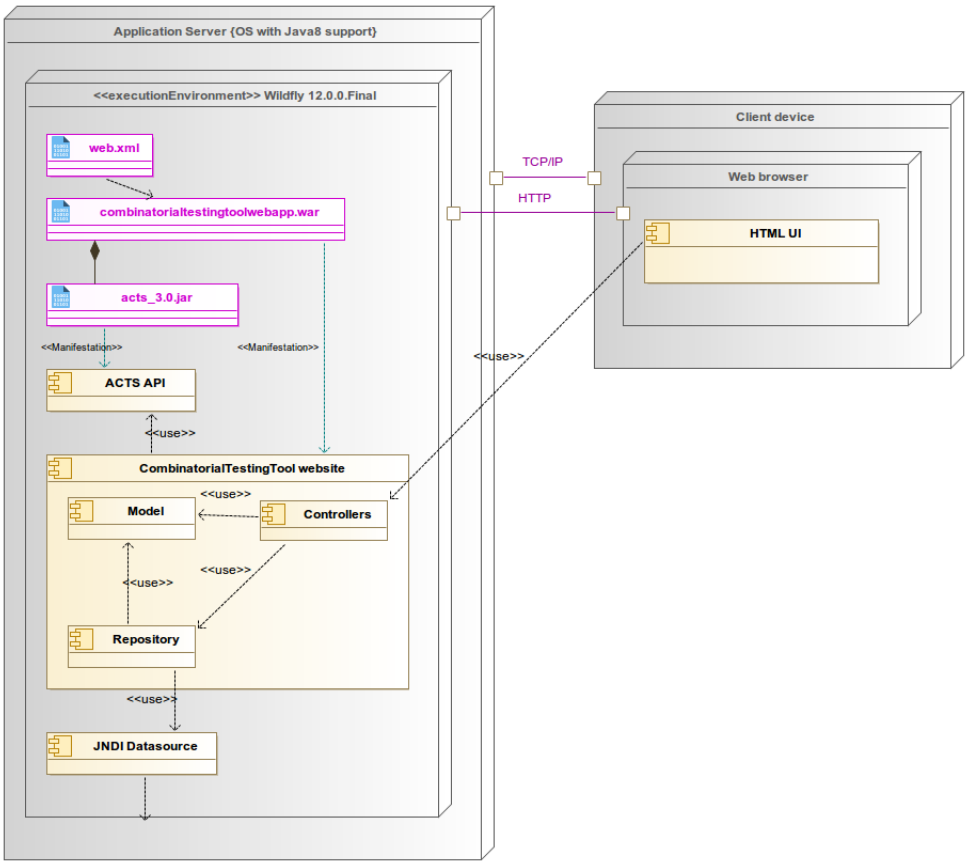
Obrázek 2.5: Interní proces systému při vytváření parametru



Obrázek 2.6: Interní proces systému při vkládání hodnoty do parametru



Obrázek 2.7: Sekvence systému při generování testovací množiny



Obrázek 2.8: Model nasazení systému

2.3.1 Aplikační server

Systém může být spuštěn na libovolném stroji s architekturou a OS podporujícím Javu 8. Systém je vyvíjen v běhovém prostředí aplikačního serveru **Wildfly 12**, který poskytuje J2EE implementaci. S tímto serverem je tedy počítáno i v reálném použití.

Systém je na server nasazen pomocí jediného webového aplikačního archivu (WAR). Ten obshuje potřebné zdrojové kódy a konfigurační soubory. Pro persistenci dat systém využívá Java Naming and Directory Interface (JNDI) datový zdroj. Ten je nutno na serveru nakonfigurovat.

Součástí WAR souboru je i Java Archive (JAR) soubor třetí strany (NIST - National Institute of Standards and Technology) jehož API je využíváno při řešení zadaných úloh.

2.3.2 Klient

K systému lze přistoupit pomocí jakéhokoli stroje, který je shopen síťové komunikace, popřípadě jen emulované (lokální), běží-li systém na tomtýž stroji. Nutností je webový prohlížeč podporující JavaScript, což je ovšem dnes již standardem, až na některé historické prohlížeče.

2.4 Testování

Součástí vývoje každého produktu by mělo být testování. Není li testováno již během vývoje, mělo by tak být učiněno minimálně před uvedením do provozu. Softwarové produkty nejsou v tomto ohledu žádnou vyjímkou, proto jsem navržený a vyvinutý systém otestoval sadou manuálních funkcionálních testů a heuristickou analýzou vytvořeného uživatelského rozhraní.

2.4.1 Manuální testování funkcionalit systému

Manuální testování aplikace spočívá v provádění předem připravených testovacích scénářů, které provádí potencionální uživatel systému. Tyto testy je levné vytvořit, avšak jejich provádění se může prodražit, obzvláště je-li potřeba testy vícekrát opakovat. Tyto aspekty jsou v zásadě opačné k automatickým testům. Jelikož je testován hotový produkt, jsou tyto testy prováděny na konci vývoje softwaru. Jsou to takzvané akceptační testy, protože je na jejich základě akceptován výsledný produkt zadavatelem. Z tohoto důvodu bývaji vytvořeny již na začátku vývoje a jsou součástí případné smlouvy mezi zadavatelem a zhotovitelem produktu.

Navržený a implementovaný systém jsem testoval řadou testů, z nichž nejdůležitější testy jako ilustraci uvádím v této podkapitole. V průběhu odevzdání práce všechny testy prošly úspěšně.

Vytvoření projektu
ID: T1
Výchozí stav:

- Uživatel je přihlášen do systému

Scénář:

1. Klikni na tlačítko *Create project*
2. Zadej název projektu *nový projekt*
3. Klikni na tlačítko *Create*

Požadovaný výsledek:

- Scénář šel provést až do konce

- Tester se nachází na stránce projektu
- Název projektu je *nový projekt*

Přepnutí projektu
ID: T2
Výchozí stav:

- Uživatel je přihlášen do systému
- V systému existuje projekt "projekt k přepnutí"a neni nastaven jako aktuální

Scénář:

1. Klikni na tlačítko *Select project*
2. Klikni na tlačítko *Show* na řádku projektu *projekt k přepnutí*
3. Klikni na talčítko *Create*

Požadovaný výsledek:

- Scénář šel provést až do konce.
- Tester se nachází na stránce projektu.
- Název projektu je *projekt k přepnutí*.

Vytvoření množiny parametrů
ID: T3
Výchozí stav:

- Uživatel je přihlášen do systému

Scénář:

1. Proveď testovací scénář *T1*
2. Klikna na tlačítko *Create parameter set*
3. Zadej název *množia parametrů 1*
4. Klikni na tlačítko *Create*

Požadovaný výsledek:

- Scénář šel provést až do konce
- Tester se nachází na stránce projektu

- Název projektu je *nový projekt*
- V rámci projektu existuje množina parametrů s názvem *množia parametrů 1*

Vytvoření parametru
ID: T4
Výchozí stav:

- Uživatel je přihlášen do systému

Scénář:

1. Proveď testovací scénář *T3*
2. Klikna na tlačítko *Create parameter* u množiny parametrů *množina parametrů 1*
3. Zadej název *parametr 1*
4. Zvol typ parametru *ENUM*
5. Klikni na tlačítko *Create*

Požadovaný výsledek:

- Scénář šel provést až do konce
- Tester se nachází na stránce projektu
- Název projektu je *nový projekt*
- V rámci projektu existuje množina parametrů s názvem *množia parametrů 1*
- V rámci množiny parametů *množia parametrů 1* existuje parametr s názvem *parametr 1* datového typu *ENUM*

Vložení hodnoty
ID: T5
Výchozí stav:

- Uživatel je přihlášen do systému

Scénář:

1. Proveď testovací scénář *T4*

2. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 1*
3. Zadej název *VALUE1*
4. Klikni na tlačítko *Create*

Požadovaný výsledek:

- Scénář šel provést až do konce
- Tester se nachází na stránce projektu
- Název projektu je *nový projekt*
- V rámci projektu existuje množina parametrů s názvem *množia parametrů 1*
- V rámci množiny parametů *množia parametrů 1* existuje parametr s názvem *parametr 1* datového typu *ENUM*
- V rámci parametru *parametr 1* existuje hodnota *VALUE1*

Generování testovacího scénáře
ID: T5
Výchozí stav:

- Uživatel je přihlášen do systému

Scénář:

1. Proveď testovací scénář *T3*
2. Klikna na tlačítko *Create parameter* u množiny parametrů *množina parametrů 1*
3. Zadej název *parametr 1*
4. Zvol typ parametru *ENUM*
5. Klikni na tlačítko *Create*
6. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 1*
7. Zadej název *VALUE1*
8. Klikni na tlačítko *Create*
9. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 1*

10. Zadej název *VALUE2*
11. Klikni na tlačítko *Create*
12. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 1*
13. Zadej název *VALUE3*
14. Klikni na tlačítko *Create*
15. Klikna na tlačítko *Create parameter* u množiny parametrů *množina parametrů 1*
16. Zadej název *parametr 2*
17. Zvol typ parametru *ENUM*
18. Klikni na tlačítko *Create*
19. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 2*
20. Zadej název *VALUE4*
21. Klikni na tlačítko *Create*
22. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 2*
23. Zadej název *VALUE5*
24. Klikni na tlačítko *Create*
25. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 2*
26. Zadej název *VALUE6*
27. Klikni na tlačítko *Create*
28. Klikna na tlačítko *Create parameter* u množiny parametrů *množina parametrů 1*
29. Zadej název *parametr 3*
30. Zvol typ parametru *ENUM*
31. Klikni na tlačítko *Create*
32. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 3*

33. Zadej název *VALUE7*
34. Klikni na tlačítko *Create*
35. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 3*
36. Zadej název *VALUE8*
37. Klikni na tlačítko *Create*
38. Klikna na tlačítko *Create parameter value* u množiny parametrů *množina parametrů 1*, parametru *parametr 3*
39. Zadej název *VALUE9*
40. Klikni na tlačítko *Create*
41. Klikni na tlačítko *Generate TS*
42. Zvol sílu pokrytí *2*
43. Klikni na tlačítko *Generate*
44. Zobrazí se zpráva o vygenerováni testovacích množin (množina parametrů je malá, vygenrováno bude okamžitě)
45. Klikni na tlačítko *Show test sets*

Požadovaný výsledek:

- Scénář šel provést až do konce
- Tester se nachází na stránce testovacích množin
- Je přítomna právě jeda testovací množina
- Tato množina má neaktivní link pro vzdálený přístup

2.4.2 Heuristická analýza uživatelského rozhraní

Heuristická analýza slouží k vyhodnocení kvality či prostě stavu zkoumaného objektu. Heuristika je v podstatě otázka či tvrzení, o které lze říci zda platí či neplatí, případně je lze ohodnotit i na nějaké stupnici. To platí především v případech, kdy je tato otázka nejednoznačná a spočívá spíše v subjektivním názoru, nežli na čistě vědeckém vyhodnocení nějakého parametru. Na základě vyhodnocení odpovědí na tyto otázky lze poté říci, zda zkoumaný objekt vyhovuje či nevyhovuje nějakému zkoumanému aspektu a jeho kvalitě.

V testech uživatelského rozhraní vyvinutého systému bude použita heuristická analýza Jakoba Nielsena, která přichází již s definovanými heuristikami.

V té je krom ohodnocení zkoumané otázky nutné uvést i důvod svého hodnocení. To je dobré jak z důvodu subjektivity hodnocení, tak především z důvodu možnosti odstranění takové chyby.

V případě zde provedené analýzy budeme hodnotit pouze uvedeným slovním hodnocením. Každý z následujících bodů vyhodnotíme pro jednotlivé hlavní oblasti vyvinuté aplikace:

1. **Viditelnost stavu systému** Uživatel ví, co systém dělá a v jakém stavu se nachází.

2. **Spojení mezi systémem a reálným světem** Komunikace mezi systémem a uživatel by měla být pro uživatele příjemná, srozumitelná až intuitivní.

3. **Uživatelská kontrola a svoboda** Lidé dělají chyby, uživatel by měl mít možnost opravit své chyby, případně by měl i předem vědět, že jeho kroky nejsou nezvratné.

4. **Konzistence a standardizace** Systém používá slova v jejich přesném sémantickém významu, uživatel by si neměl lámat hlavu s interpretací.

5. **Prevence chyb** Lepší, než se umět s chybami vypořádat, je chyby nedělat.

6. **Rozpoznání místo vzpomínání** Uživatel se má soustředit na konkrétní činnost, né na to jak tuto činnost vykonat v systému. V případě dialogového průvodce je zas dobré, aby uživatel nemusel vzpomínat, co zadal v minulém kroku.

7. **Flexibilní a efektivní použití** Zkušený uživatel nepotřebuje asistenci, práci by mu neměly ztěžovat instrukce a kroky určené začátečníkům.

8. **Estetický a minimalistický design** Systém zobrazuje jen právě potřebné informace a nabízí jen právě potřebné a použitelné funkce.

9. **Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb** Chybová hlášení by měla být jasná, srozumitelná a uživatel by měl vědět, jak dále postupovat.

10. **Nápověda a návody** Uživatel by neměl dojít do situace, kdy neví jak pokračovat, případně kde tuto informaci získat.

Seznam projektů: 1. Viditelnost stavu systému Menu, odražka aktuálního projektu by mohlo zobrazovat název (nebude opakováno u všech obrazovek).

2. Spojení mezi systémem a reálným světem OK

3. **Uživatelská kontrola a svoboda** Tlačítko smazání projektu sice po kliknutí zobrazí potvrzující dialog, avšak tento fakt není implicitně zřejmý, možná by se hodil popis v bublinkové nápovědě (tooltipu).
4. **Konzistence a standardizace** Tlačítko pro smazání projektu by mohlo mít spíše ikonu koše. U tlačítka pro zobrazení projektu bych volil jinou ikonu, nějakou ikonu ruky například.
5. **Prevence chyb** OK
6. **Rozpoznání místo vzpomínání** Název projektu v tabulce nemusí být dostačující, nějaký popis či náhled dat v projektu by uživateli
7. **Flexibilní a efektivní použití** Pro operace z menu by se hodily klávesové zkratky (nebude opakováno u všech obrazovek).
8. **Estetický a minimalistický design** Identifikátor projektu nemusí být pro uživatele důležitý údaj.
9. **Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb** Bez výhrad
10. **Nápověda a návody** OK

Projekt:
1. **Viditelnost stavu systému** Při déletrvajícím genrování scénářů chybí náhled o stavu generování. Zobrazení typu parametru skrze složené tlačítko není nejvhodnější, uživatel se musí zaměřit na podbarvení tlačítka, aby zjistil které je nastaveno.
2. **Spojení mezi systémem a reálným světem** OK
3. **Uživatelská kontrola a svoboda** Uživatel provádí editující operace, ale není přítomna historie úprav. Dlouhotrvající generování nelze zrušit.
4. **Konzistence a standardizace** OK
5. **Prevence chyb** OK
6. **Rozpoznání místo vzpomínání** Uživateli nemusí být jasno, že za odkazem na správu konfigurací se skrývá možnost pokročilého generování testovacích množin.
7. **Flexibilní a efektivní použití** Pro generování test. množin, přidání množin parametrů a parametrů by se hodily klávesové zkratky.
8. **Estetický a minimalistický design** Viz *Viditelnost stavu systému*. Způsob zobrazení/volby datového typu parametru bych volil spíše kombobox.
9. **Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb** OK

10. Nápověda a návody Po najetí na ovládací tlačítka by bylo vhodné podat uživateli informace v bublinové nápovědě (tooltip).

Pokročilé genrování testovacích scénářů: 1. Viditelnost stavu systému Při déletrvajícím genrování scénářů chybí náhled o stavu generování.

2. Spojení mezi systémem a reálným světem OK

3. Uživatelská kontrola a svoboda Uživatel provádí editující operace, ale není přítomna historie úprav.

4. Konzistence a standardizace OK

5. Prevence chyb OK

6. Rozpoznání místo vzpomínání OK

7. Flexibilní a efektivní použití Pro generování test. množin, přidání konfigurací a přidání množin interakcí by se hodily klávesové zkratky.

8. Estetický a minimalistický design Parametry by bylo vhodnější zobrazovat v řádcích a množiny interakcí naopak ve sloupcích.

9. Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb OK

10. Nápověda a návody Po najetí na ovládací tlačítka by bylo vhodné podat uživateli informace v bublinové nápovědě (tooltip). Uživatel neví, zda je vhodné vkládat parametry do více množin interakcí.

Seznam testovacích množin: 1. Viditelnost stavu systému OK

2. Spojení mezi systémem a reálným světem OK

3. Uživatelská kontrola a svoboda

4. Konzistence a standardizace OK

5. Prevence chyb OK

6. Rozpoznání místo vzpomínání OK

7. Flexibilní a efektivní použití

8. Estetický a minimalistický design Identifikátor testovacího scénáře není pro uživatele potřebný.

9. Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb Tlačítko smazání testovací množiny sice po kliknutí zobrazí potvrzující dialog, avšak tento fakt není implicitně zřejmý, možná popis v bublinkové nápovědě (tooltipu).

10. Nápověda a návody

Testovací množina: 1. Viditelnost stavu systému OK

2. Spojení mezi systémem a reálným světem OK

3. Uživatelská kontrola a svoboda Testovací množina by mohla jít smazat i z této obrazovky.

4. Konzistence a standardizace OK

5. Prevence chyb OK

6. Rozpoznání místo vzpomínání OK

7. Flexibilní a efektivní použití OK

8. Estetický a minimalistický design Data jsou v textových polích, šlo by spíše zobrazit pomocí vhodných grafických komponent.

9. Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb OK

10. Nápověda a návody OK

V průběhu provedené heuristické analýzy jsem našel několik možných potenciálních vylepšení vytvořeného uživatelského rozhraní, které jsou uvedené v detailech analýzy výše. Po konzultaci se školitelem jsme se rozhodli tato vylepšení zařadit do plánu dalšího rozvoje vyvinutého systému.

Závěr

V práci jsem navrhl a implementoval webovou aplikaci poskytující uživatelské rozhraní pro generování testovacích scénářů založených na kombinacích vstupních dat. Aplikace pro výpočet scénářů používá knihovnu "Automated Combinatorial Testing for Software"(ACTS) poskytovanou Národním institutem standardů a technologie (NIST) Spojených států Amerických [19].

Systém uživateli poskytuje platformu pro generování kombinací dat dle zadaných parametrů a jejich hodnot použitelných při testování softwaru. Toto vše probíhá skrze jednoduché a přimočaré rozhraní s možností správy vícero projektů. Generovaná data jsou poskytována i vzdáleně pomocí speciálního rozhraní, kde jsou poskytována v CSV formátu.

Systém byl vyvinut v rámci technologie J2EE. Infrastruktura systémových komponent je postavena dle principů Model-View-Controller (MVC) architektury. Pro implementaci grafického rozhraní (view) je použito JSF XHTML šablon (tzv. facelety). Kontrolery využívají technologie CDI a pro jejich přístupnost z uživatelského rozhraní je použito technologie EL. Datový model systému je tvořen POJO třídami, respektive instancemi jejich tříd, pro persistenci využívající technologii JPA, konkrétně její implementaci Hibernate. Pro přístup k funkcionalitám JPA systém používá JNDI.

Při vývoji aplikace nenastaly žádné závažné komplikace, jistý čas navíc však zkonzumovalo mapování entitních tříd a řešení konfliktů v knihovnách, které systém využívá (tzv. závislosti). Další problémy vyvstaly při psaní teoretické části diplomové práce, kdy mi dělalo potíže upravit text do čitelné formy. Čitelnou formou je míněna konkrétné návaznost jednotlivých myšlenkových celků a sdělení.

Jelikož je systém vyvýjen pro reálné použití v praxi, bude pravděpodobně vhodné provést několik vylepšení. Krom očekávané zpětné vazby uživatelů, která může vést k redesignu uživatelského rozhraní, tak z pohledu funkcionalit by do budoucna bylo vhodné rozšířit zadání úlohy o různá integritní omezení v podobě nežádoucích kombinací dat (například OS:Windows, Prohlížeč:Safari). Další rozšíření je do jisté míry související s předešlou funkcio-

nalitou, v tomto případě by naopak byla cíleně generována data, která by dle specifikace měla způsobit defekt. Z pohledu použitelnosti, by bylo vhodné rozšířit datovou reprezentaci generovaných dat o další datové formáty a umožnit uživateli zpětnou vazbu v podobě ohlášení testovacími množinami nalezené defekty. V neposlední řadě by bylo vhodné provést refaktoring v rámci spravovatelnosti a udržitelnosti systému.

Literatura

[1] Kuhn, D. R.; Kacker, R. N.; Lei, Y.: *Introduction to combinatorial testing.* CRC press, 2013.

[2] Kuhn, D. R.; Wallace, D. R.; Gallo, A. M.: Software fault interactions and implications for software testing. *IEEE transactions on software engineering*, ročník 30, č. 6, 2004: s. 418–421.

[3] Wallace, D. R.; Kuhn, D. R.: Failure modes in medical device software: an analysis of 15 years of recall data. *International Journal of Reliability, Quality and Safety Engineering*, ročník 8, č. 04, 2001: s. 351–371.

[4] Schroeder, P. J.; Bolaki, P.; Gopu, V.: Comparing the fault detection effectiveness of n-way and random test suites. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, IEEE, 2004, s. 49–59.

[5] Burr, K.; Young, W.: Combinatorial test techniques: Table-based automation, test generation and code coverage. In *Proc. of the Intl. Conf. on Software Testing Analysis & Review*, Citeseer, 1998.

[6] Bures, M.; Ahmed, B. S.: On the effectiveness of combinatorial interaction testing: A case study. In *Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on*, IEEE, 2017, s. 69–76.

[7] Bures, M.; Cerny, T.; Ahmed, B. S.: Internet of Things: Current Challenges in the Quality Assurance and Testing Methods. *arXiv preprint arXiv:1805.01241*, 2018.

[8] Ahmed, B. S.; Zamli, K. Z.; Afzal, W.; aj.: Constrained interaction testing: A systematic literature study. *IEEE Access*, ročník 5, 2017: s. 25706–25730.

[9] Zamli, K. Z.; Din, F.; Ahmed, B. S.; aj.: A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem. *PloS one*, ročník 13, č. 5, 2018: str. e0195675.

[10] Calvagna, A.; Gargantini, A.: IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays. In *Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on*, IEEE, 2009, s. 10–18.

[11] Alsewari, A. R. A.; Zamli, K. Z.: Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. *Information and Software Technology*, ročník 54, č. 6, 2012: s. 553–568.

[12] Fraser, G.; Arcuri, A.: A large-scale evaluation of automated unit test generation using evosuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ročník 24, č. 2, 2014: str. 8.

[13] Kuliamin, V. V.; Petukhov, A.: A survey of methods for constructing covering arrays. *Programming and Computer Software*, ročník 37, č. 3, 2011: str. 121.

[14] Chen, X.; Gu, Q.; Li, A.; aj.: Variable strength interaction testing with an ant colony system approach. In *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific*, IEEE, 2009, s. 160–167.

[15] Java™ EE 7 Technologies. `https://www.oracle.com/technetwork/java/javaee/tech/index-jsp-142185.html`, accessed: 2018-12-01.

[16] WildFly Documentation. `http://docs.wildfly.org/12/`, accessed: 2018-12-01.

[17] PrimeFaces User guide. `https://www.primefaces.org/docs/guide/primefaces_user_guide_6_2.pdf`, accessed: 2018-12-01.

[18] HIBERNATE documentation. `http://hibernate.org/orm/documentation/5.2/`, accessed: 2018-12-01.

[19] ACTS library. `https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software/ACTS-Library`, accessed: 2018-12-01.

Příloha **A**

Seznam použitých zkratek

ACTS Automated Combinatorial Testing for Software

API Application Programming Interface

CDI Contexts and Dependency Injection

GUI Graphical user interface

HTML Hypertext Markup Language

J2EE Java Platform, Enterprise Edition

JSF JavaServer Faces

JSP JavaServer Pages

JSTL JSP Standard Tag Library

NIST Národní institut standardů a technologie

ORM Objektově relační zobrazení

XHTML Extensible Hypertext Markup Language

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

`source.zip` ..... zdrojový kód aplikace + návod na spuštění
`thesis.pdf` ..... text práce ve formátu PDF