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

Diplomová práce

Rozšiřující moduly validačního serveru

Plzeň 2014 Hung Duong Manh

Prohlášení

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

V Plzni dne 10. května 2014

Hung Duong Manh

Abstrakt

Tato práce popisuje rozšíření funkčnosti validačního serveru pomocí modulů. Validační server je používán na Západočeské univerzitě v Plzni. V první kapitole je stručně vysvětlen validační server, jeho webové rozhraní a nástroje pro řízení jeho vývoje. Druhá kapitola analyzuje náměty na vylepšení validátoru. V třetí kapitole se na základě provedených analýz vytváří návrh řešení a popis implementace řešení. Nakonec jsou popsány funkční testy modulů.

Abstract

This thesis describes how the validation server used at University of West Bohemia was enhanced with the help of extension plugins. The first chapter explains the validation server, its web interface and tools used for managing the development of validation server. The second chapter analyzes suggestions for improving the validation server. In the third chapter I propose a solution based on each analysis and describe the implementation of the solution. The last chapter describes functional tests for each extension plugin.

Poděkování

Rád bych poděkoval doc. Ing. Pavlu Heroutovi, Ph.D. za umožnění vzniku této práce a Ing. Štěpánovi Caisovi, vedoucímu práce, za směrování mé činnosti.

Obsah

1 Úvod

V rámci předmětů vyučovaných na vysokých školách studenti běžně odevzdávají své práce pro vyhodnocení vyučujícími či cvičícími. Charakter těchto prací je někdy takový, že by je šlo velmi jednoduše vyhodnotit automatizovaně. Automatizované vyhodnocování je nejen rychlejší, takže ušetří čas, ale i spravedlivější v tom smyslu, že vyhodnocuje vždy stejně.

Pro tyto účely byl na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni vytvořen validační server, který má sloužit k automatizovanému vyhodnocování studentských prací. Do validačního serveru lze přidat novou funkčnost pouhým doprogramováním rozšiřujícího modulu. Tyto moduly pak umožňují nové způsoby validace a dohromady dokážou vytvořit komplexní test.

V minulých letech vznikalo postupně malé množství rozšiřujících modulů, které navíc byly vytvořeny různými autory. Štábní kultura každého modulu se proto značně lišila. S rostoucím počtem modulů se ukázalo, co vše by se dalo zlepšit, zjednodušit a sjednotit.

Navíc se také objevila nutnost lepší správy a organizace vývoje modulů. Doposud totiž zdrojové kódy udržoval každý autor zvlášť, což pak činilo problém pro jiného autora, který měl na předchozí práci navázat. O validátor už projevují zájem i jiné předměty z ostatních kateder. Bylo by proto vhodné mít sepsané návody pro celý vývojářský cyklus.

Cílem této práce je sjednotit programový kód stávajících modulů, zjednodušit pro programátory vytváření nových modulů, zdokumentování postupů pro vývoj na validátoru a rozšíření funkčnosti validátoru, ať už pomocí nových modulů nebo úpravou kódu samotného validačního serveru.

2 Teoretická část

2.1 Validační server

2.1.1 Co je validační server a k čemu slouží

Validační server, neboli validátor, je webová aplikace napsaná v programovacím jazyce Java. Aplikace běží na vlastním serveru s Apache Tomcat pod Katedrou informatiky a výpočetní techniky Západočeské univerzity v Plzni a slouží k automatické kontrole elektronicky odevzdávaných souborů. Apache Tomcat je open source webový server, do kterého se nasazují webové aplikace využívající technologii Java Servlet a JSP.

Zřejmé výhody automatické kontroly jsou:

- snížení doby reakce na odevzdanou úlohu,
- objektivnost kontroly,
- úspora času pro studenta,
- masivní úspora času vyučujícího,
- jednotné odevzdávací místo s přístupem 24/7.

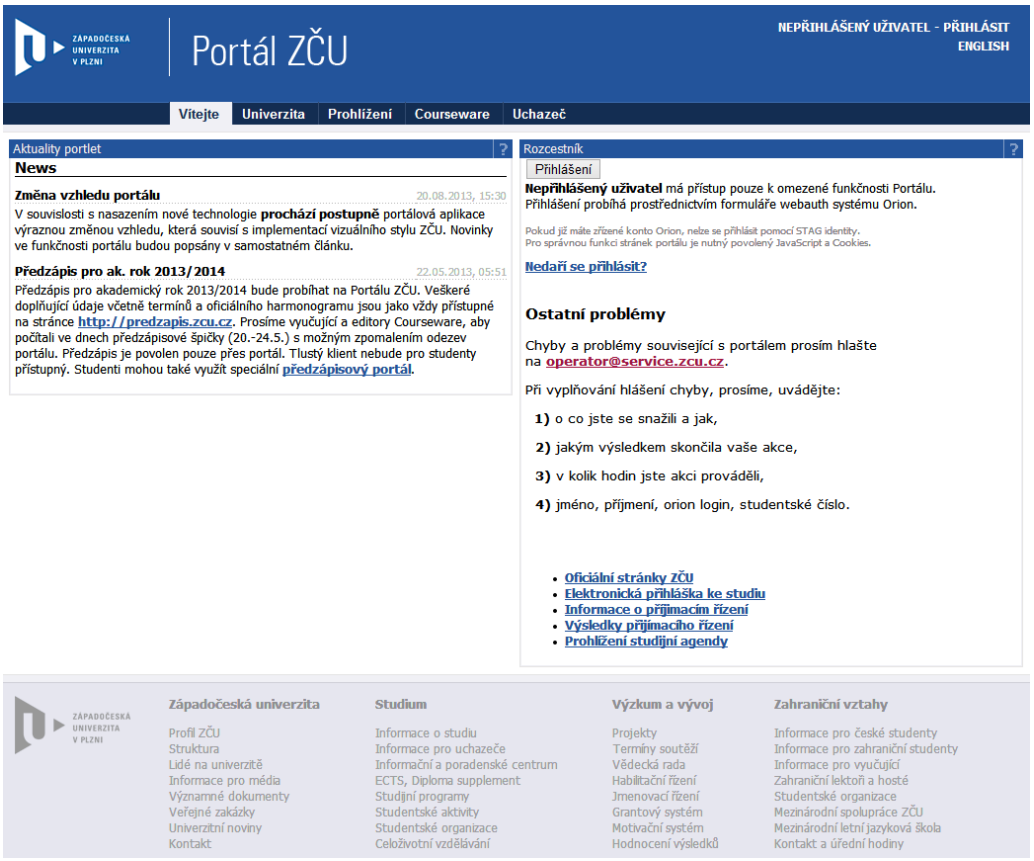
Nevýhodou však je, že výstupem automatické kontroly validátoru je pouze informace, zda práce vyhovuje zadaným kritériím nebo ne. Validátor tedy neumí odevzdávanou práci ohodnotit známkou. Stále však lze validátor použít jako filtr pro splnění formálních požadavků na práci. Příkladem může být esej, která musí splňovat minimální počet stránek, aby mohla být vůbec přijata pro hodnocení.

Typ kontroly, který může být validátorem proveden, je pouze limitován možnostmi jazyka Java. Zjednodušeně by se dalo tedy říci, že pokud si programátor dokáže danou kontrolu představit algoritmizovaně, pak ji lze na validátoru provést [9].

2.1.2 Validační server a Portál ZČU

Západočeská univerzita v Plzni provozuje pro své studenty a zaměstnance Portál ZČU (obr. 2.1), což je webové místo, které se snaží poskytnout všechny relevantní informace, týkající se působení přihlášeného uživatele na ZČU, na jednom místě. Pro přístup na Portál je třeba se přihlásit Orion kontem.

Orion konto je elektronickou identitou studenta či zaměstnance ZČU a umožňuje svému majiteli používat elektronické informační zdroje ZČU.



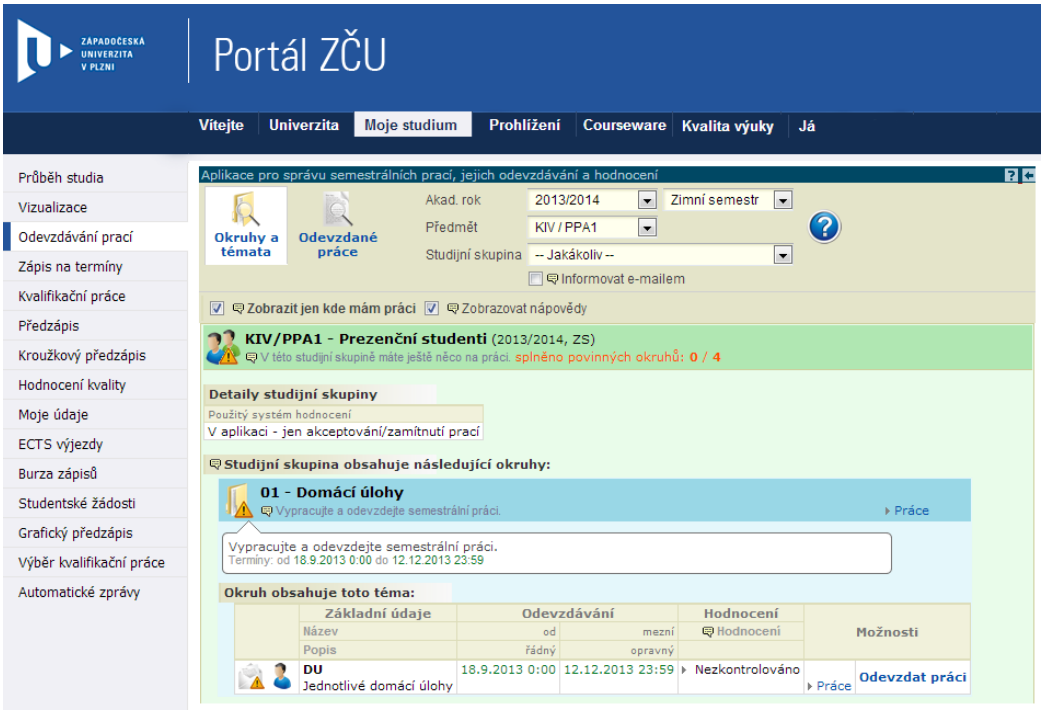
Obrázek 2.1: Portál ZČU

Každý uživatel v Portálu má svoji nastavenou roli, která mu poskytuje jiná uživatelská práva. Uživatel, který má například roli student, nevidí a nemůže dělat to samé, co uživatel s rolí vyučující.

Na Portálu si studenti mohou mimo jiné prohlížet svůj rozvrh, průběh svého studia nebo provádět předzápis. Studenti mají možnost na Portálu

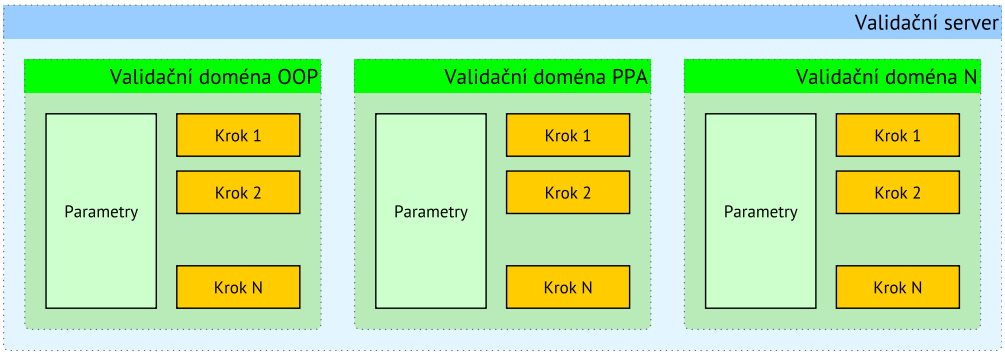
odevzdávat i své studenské práce přes odevzdávací portlet (obr. 2.2).

Validační server je plně integrován do odevzdávacího portletu Portálu ZČU. V praxi to znamená, že pokud se vyučující rozhodne pro využití validačního serveru, pak všechny práce odevzdané studenty na dané téma prochází validační doménou nastavenou pro toto téma.



Obrázek 2.2: Odevzdávací portlet v Portálu ZČU

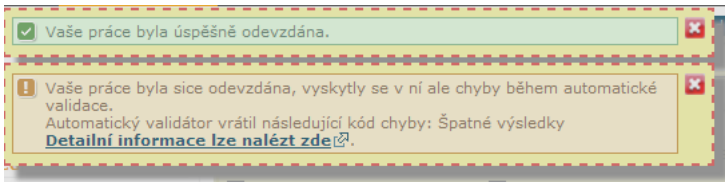
Validační doména je nejvyšší uživatelsky manipulovatelná jednotka validátoru a určuje jako taková celý proces validace. Každá doména sestává ze sekvence kroků, které se mají provést (počet kroků není nijak omezen). Krok představuje jednu vykonatelnou činnost validace (např. zkompilování adresáře). V rámci kroku lze využít rozšiřitelný modul neboli vlastní akci (oba pojmy jsou zaměnitelné). Domény mají také své nastavitelné parametry jako například typ souboru, který může vstupovat do validace, nebo maximální velikost vstupního souboru. V příloze A jsou uvedeny všechny používané parametry domén. Na obrázku 2.3 je zobrazen vztah mezi validačním serverem, doménou a kroky.



Obrázek 2.3: Schéma vztahů mezi validátorem, doménou a kroky

2.1.3 Jak validace probíhá

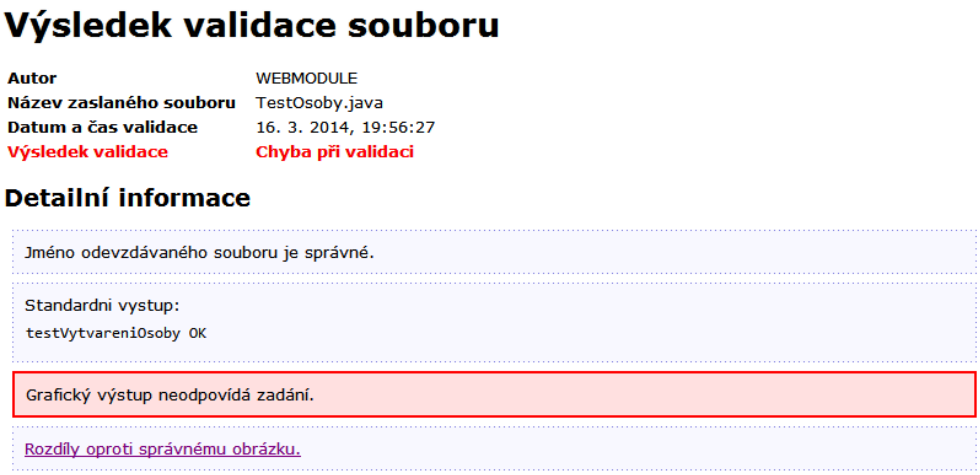
Student nejprve odevzdá práci přes webový prohlížeč na Portálu ZČU. Jakmile je práce nahrána na Portál, odešle se v zápětí validačnímu serveru ke kontrole. Validační server nyní začne aplikovat jednotlivé kroky validace nastavené ve validační doméně. Každá konkrétní instance validace má svůj výstupní výsledek, který je přítomný ve všech krocích a do kterého mohou být v rámci kroku generovány 3 typy zpráv: chyby, varovaní nebo zprávy informativního charakteru. Pokud práce projde všemi kroky bez chyb, je přijata. V opačném případě je práce odmítnuta a studentovi se v okně Portálu zobrazí zpráva o chybě (obr. 2.4). V okně je také odkaz na detailnější popis chyby, který je vygenerován transformací výstupního výsledku validace do HTML stránky (obr. 2.5).



Obrázek 2.4: Špatné výsledky při odevzdaní práce

2.1.4 Přístup na souborový systém validátoru

Přístup na souborový systém je důležitý, protože jenom tak lze restartovat validační server, nahrávat nové verze rozšiřujících modulů či nasazovat novou verzi samotného validátoru. Nehledě na to, že občas je to jediný způsob jak ladit vlastní akce či chybu ve validátoru.



Obrázek 2.5: Podrobnější výpis výsledku validace při chybě

Přihlášení na validační server přes SSH

1. nejprve je třeba se připojit na `eryx.zcu.cz` pomocí nástroje putty,
2. po navázaní SSH spojení nás server vyzve, abychom se přihlásili – k tomu použijeme Orion konto,
3. nyní je třeba se připojit na samotný validační server. K tomu slouží příkaz:

```
> ssh -K valid@validator.zcu.cz
```

   Aby připojení proběhlo úspěšně, je třeba mít povolen přístup od správce validátoru.

4. úspěšné připojení poznáme podle indikace uživatele:

```
valid@valid:~$
```

2.1.5 Adresářová struktura

Validátor má vždy 3 složky: `valid`, `valid-common` a `validrun`. Z pohledu vývoje rozšiřujících modulů jsou důležité pouze `valid` a `valid-common`. Složky začínající názvem `validt` jsou složky testovacího validátoru, viz kapitola 2.2.3 na straně 12.

```
/opt
├── valid
├── valid-common
├── validrun
├── validt
├── validt-common
└── validtrun
```

Adresářová struktura validátoru je celkem rozsáhlá a bylo by zbytečné popisovat každý adresář. Následuje popis podstatných adresářů pro vývoj rozšiřujících modulů.

valid

Do složky `logs` se generují výpisy konzolového výstupu Tomcatu a validačního serveru. Výstup validačního serveru se vypisuje konkrétně do souboru `ZCU.log` – vypisují se do něj jak ladící vypisy, tak neošetřené vyjímky. Složka `lib` obsahuje všechny knihovny, které validační server potřebuje ke svému fungování. Nachází se zde samotný validační server `VS.jar`, rozšiřující moduly a ostatní potřebné knihovny. V `auth` se nachází `.jsp` stránky, které využívá webové rozhraní validačního serveru. Složky `css` a `js` obsahují CSS (Cascading Style Sheets) a javascriptové zdroje pro tyto stránky.

```
/opt/valid
├── logs
└── webapps
    └── vs
        ├── WEB-INF
        │   └── lib
        ├── auth
        ├── css
        └── js
```

valid-common

Ve složce `domains` se nachází všechny domény. Každá doména má svůj podadresář, ve kterém se nachází soubor pro nastavení parametrů domény – `domain.xml`, soubor pro nastavení uživatelských práv pro danou doménu – `run.policy` a samotná definice validace pro doménu – `webmodule.xml` (viz ukázka 2.2)

Do složky `workdir` se generují pracovní adresáře validací. Pracovní adresář je dočasný adresář, který je generován pro každou novou instanci validace. V rámci daného pracovního adresáře lze pak pracovat se soubory podle uživatelských práv definovaných v souboru `run.policy`. V nastavení domény je přepínač, kterým lze nastavit, zda se má pracovní adresář po validaci smazat. Nesmazání pracovního adresáře se hodí, pokud ladíme domény.

Složka `plugins` obsahuje knihovny rozšiřujících modulů. Vztah mezi touto složkou a složkou `valid/webapps/vs/WEB-INF/lib` je takový, že při každém restartu validačního serveru se knihovny kopírují z první složky do druhé. Je to proto, aby nový nezkušený uživatel nepracoval hned s hlavní složkou, která obsahuje i další knihovny, které by způsobily pád aplikace v případě smazání.

```
/opt/valid-common
├── data
│   ├── domains
│   └── workdir
└── lib
    └── plugins
```

2.2 Webové rozhraní validačního serveru

2.2.1 Stav před vytvořením webového rozhraní

Všechna nastavení domén a jejich kroků byly v původní verzi validátoru prováděny editací xml souboru a jako takové byly nesrozumitelné pro někoho, kdo validátor viděl poprvé. Pro editaci musel mít navíc uživatel přístup přímo na souborový systém validačního serveru. Z tohoto pohledu byla práce s validátorem uživatelsky velmi nepřivětivá a z bezpečnostního hlediska nebezpečná.

```
<?xml version="1.0" ?>
<!DOCTYPE validation-process PUBLIC "-//Validation-Process //" "process.dtd">

<validation-process>

    <!-- volani podprocesu, ktery zkontroluje, zda se jedna o Java zdrojak -->
    <call process="/common/checkjavasource.xml">
        <param name="file">inputFile</param>
    </call>

    <!-- Vse uvnitr tohoto elementu ma casovy limit provedeni 5000 ms -->
    <time-limited time="5000" exceed-message="ppa1.prekrocil">

        <!-- prelozim zaslany soubor -->
        <call process="/common/compilejava.xml">
            <param name="file">inputFile</param>
        </call>

    </time-limited>

    <!-- Pokud uz je ve ValidationResult nejaka chyba, koncim -->
    <quit-if-error/>

    <!-- Script -->
    <script>
        var testOuput = new java.io.File(workDir,'test.out');
        var vzorOuput = new java.io.File(domainDir,'vzor.out');
    </script>

    <time-limited time="3000" exceed-message="ppa1.prekrocil">

      <!-- zjistim vystup testovane tridy -->
      <call process="/common/invokejava.xml">
         <param name="mainClass">className</param>
         <param name="classPath">workDir</param>
         <param name="stdIn">null</param>
         <param name="stdOut">testOutput</param>
         <param name="stdErr">null</param>
         <param name="args">null</param>
      </call>

    </time-limited>

    <!-- porovnam co vylezlo a co melo vylezt -->
    <call process="/common/comparefiles.xml">
        <param name="file1">vzorOutput</param>
        <param name="file2">testOutput</param>
    </call>

</validation-process>
```

Ukázka 2.1: Původní způsob nastavení validační domény

Validační doména z ukázky 2.1 provádí čtyři jednoduché činnosti:

1. ověření, že odevzdaný soubor má koncovku `.java`,
2. kompilace zdrojového souboru s časovým limitem pět sekund,

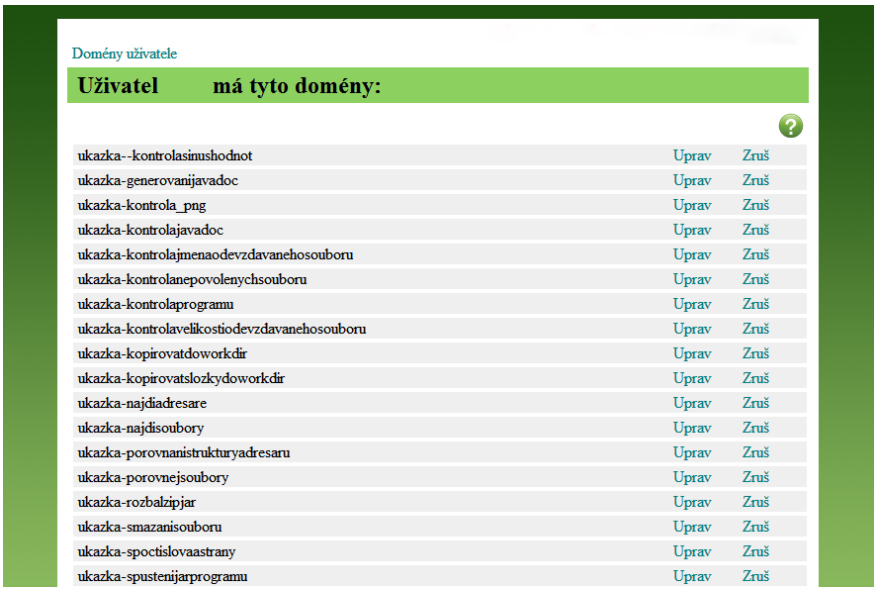
3. spuštění přeloženého souboru,
4. porovnání výsledků.

Z ukázky je vidět, že tento způsob nebyl moc srozumitelný pro uživatele bez technického pozadí a že nastavit čtyři jednoduché činnosti bylo celkem náročné.

2.2.2 Vznik webového rozhraní

Výše zmíněné důvody vedly k myšlence vytvořit k validátoru jednoduché uživatelské rozhraní, kde by si uživatel mohl spravovat své domény a přihlašování by probíhalo přes jednotné Orion konto, které používá celá ZČU.

Samotná realizace této myšlenky přišla až v roce 2010, kdy Veronika Dudová vytvořila v rámci své bakalářské práce webové rozhraní pro validační server [3] (obr. 2.6 – ukázkové domény na obrázku jsou výsledkem pozdějších prací).



Obrázek 2.6: Úvodní obrazovka webového rozhraní validátoru

Webové rozhraní výrazně zjednodušilo práci s válidátorem a xml struktura pro definici domény byla zcela přepracována – zavedly se kroky validace (ukázka 2.2). Protože se každá činnost formalizovala do kroku, bylo možné

závest rozšiřující moduly jako externí činnost, kterou lze jednoduše přidat a přiřadit ke kroku. Kroky se provádí tak, jak jdou v souboru za sebou.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domena>
    <userName>testUser</userName>
    <popis>Domena ukazka-spoctislovaastrany</popis>
    <krok nazev="spocti_slova_a_strany">
        <popis>Spocte pocet slov a pocet stran v pdf nebo doc souboru</popis>
        <podminka>
            <typ>vzdy</typ>
        </podminka>
        <akce>
            <typ>vlastniakce</typ>
            <param key="nazevakce">Spocti_slova_a_strany</param>
            <param key="Soubor"></param>
            <param key="Pocet slov">pocetSlov</param>
            <param key="Pocet stran">pocetStran</param>
        </akce>
    </krok>
    <krok nazev="vypis_pocet_slov">
        <popis>Vypise obsah promenne, ve ktere je ulozen pocet slov</popis>
        <podminka>
            <typ>vzdy</typ>
        </podminka>
        <akce>
            <typ>vloz</typ>
            <param key="co">info</param>
            <param key="text"></param>
            <param key="skript">"Pocet slov:" +pocetSlov</param>
        </akce>
    </krok>
    <krok nazev="vypis_pocet_stran">
        <popis>Vypise obsah promenne, ve ktere je ulozen pocet stran</popis>
        <podminka>
            <typ>vzdy</typ>
        </podminka>
        <akce>
            <typ>vloz</typ>
            <param key="co"></param>
            <param key="text"></param>
            <param key="skript">"Pocet stran: " +pocetStran</param>
        </akce>
    </krok>
</domena>
```

Ukázka 2.2: Současná podoba souboru defininující doménu

V rámci jednoho kroku lze provést tyto akce:

- nic,
- skok na jiný krok,
- vložení textu do výstupu validace,

- ukončení validace,
- spuštění vlastní akce – rozšiřujícího modulu.

Jednotlivé kroky mohou být také nastaveny, aby se provedly podmíněně. K tomuto účelu slouží javascriptové proměnné a přednastavené podmínky (obr. 2.7). Kombinace javascriptových proměnných v podmínkach a možných akcí poskytuje velmi silný nástroj, jak řídit průběh validace. Lze tak například vytvořit smyčku kroků.

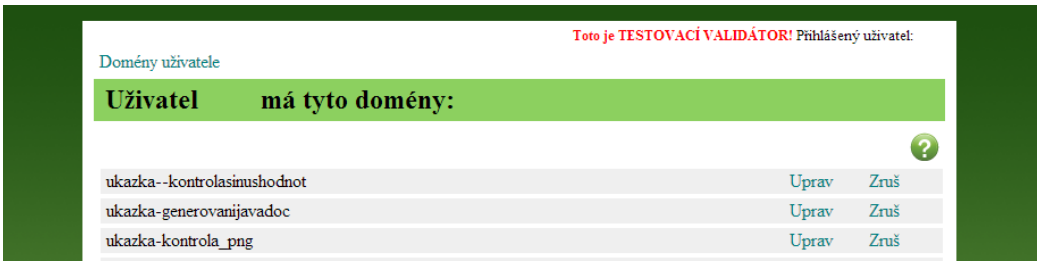
Obrázek 2.7: Podmínky provedení kroku ve validační doméně

2.2.3 Testovací validační server

Validátor je používán téměř po celý akademický rok a bylo by problematické, kdyby studenti nemohli odevzdat svojí práci, protože úprava ve zdrojových kódech validátoru rozbila celou aplikaci. Pro účely vývoje a testování byla proto spuštěna druhá instance validátoru – testovací validátor. Na tomto testovacím validátoru je nasazena zpravidla poslední vývojová verze. Nová verze validátoru se nasazuje až po důkladném otestování všech nových funkčností na testovacím validátoru. Proces nasazení nové verze probíhá v součinnosti se správcem serveru ostrého validátoru (CIV).

Vyučující, kteří mají zájem o využití validačního serveru a chtějí si vyzkoušet jeho možnosti, dostávají přístup pouze na testovací validační server. Totéž platí pro programátory, kteří se podílí na vývoji validátoru.

Webové rozhraní ostrého a testovacího validátoru vypadá identicky. Aby bylo jednoduše rozpoznatelné, kterou instanci uživatel upravuje, je v pravém horním rohu upozornění (obr. 2.8).



Obrázek 2.8: Upozornění na testovací validátor v pravém horním rohu

Webové rozhraní pro ostrý validátor běží na adrese:

```
https://validator.kiv.zcu.cz/
```

Webové rozhraní pro testovací validátor běží na adrese:

```
https://validator-test.kiv.zcu.cz/
```

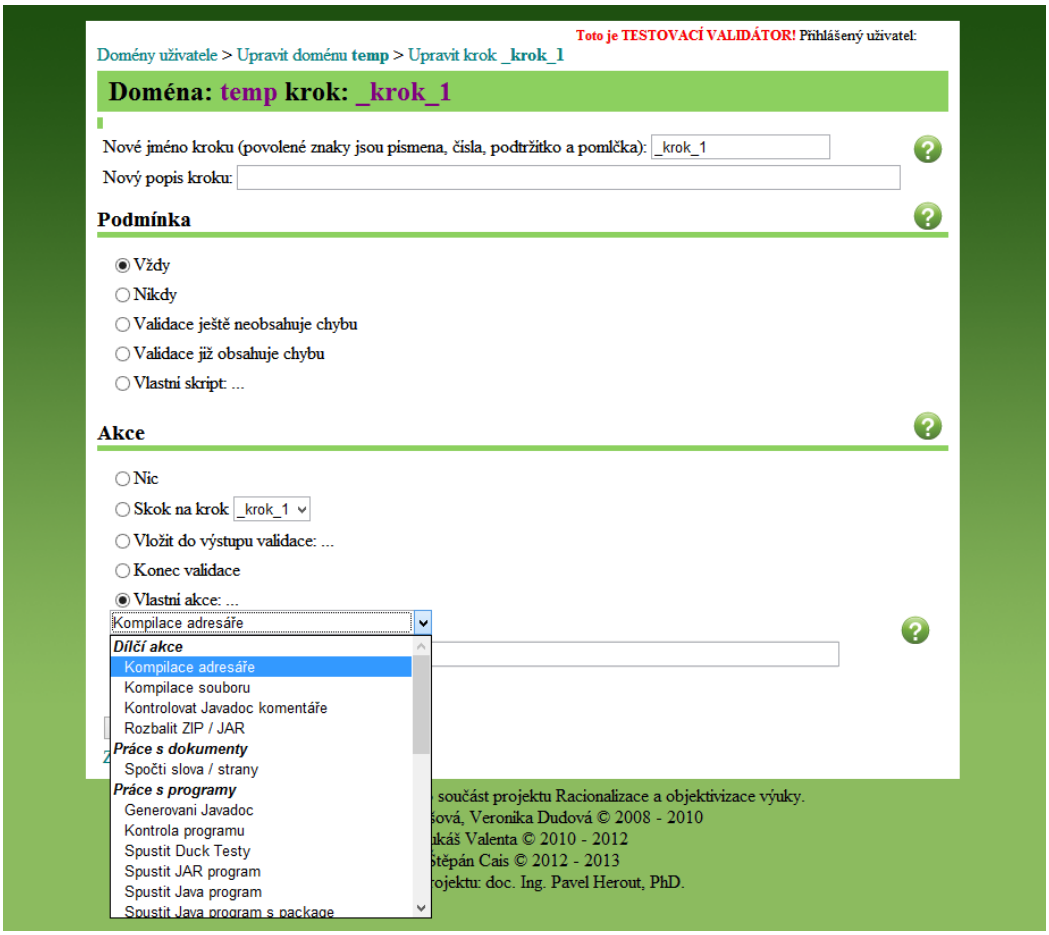
Přístup na souborový systém testovacího validátoru je pak

```
> ssh -K validt@validator.zcu.cz
```

2.3 Rozšiřující moduly validačního serveru

Jak bylo naznačeno v kapitole 2.2.2, rozšiřující moduly jsou akce, které mohou být použity v rámci kroku validace (obr. 2.9).

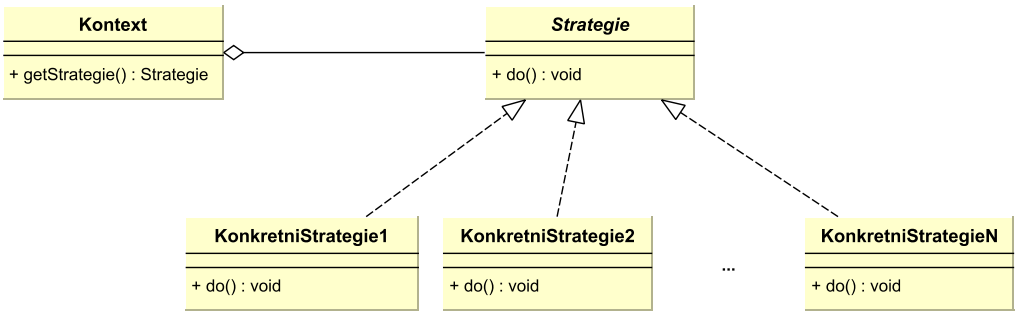
Rozšiřující modul je z implementačního hlediska Java třída, která musí splňovat konkrétní rozhraní. Tím, že existuje jednotné rozhraní, pak mohou být nové rozšiřující moduly jednoduše přidány do validátoru, aniž by se musel jakýmkoliv způsobem upravovat programový kód samotného validátoru či



Obrázek 2.9: Výběr vlastní akce – rozšiřujícího modulu

webového rozhraní. V OOP (objektově orientovaném programování) se tento způsob řešení vyskytuje tak často, že vznikl samotný návrhový vzor - strategy pattern [4].

Návrhový vzor (ang. design pattern), je ověřené obecné řesení nějakého návrhového problému v objektově orientovaném programování. V návrhových vzorech se využívají vlastnosti OOP jako je dědičnost a polymorfismus, aby se zjednodušil celkový návrh aplikace. Pokud například architekt aplikace očekává, že se bude provádět stále stejná činnost, ale bude se měnit její konkrétní implementace, použije strategy pattern. Existují různé katalogy návrhových vzorů, kde si programátoři mohou najít, zda podobný problém s návrhem aplikace neřešil už někdo před nimi. Každý návrhový vzor má své pro a proti – tím, že něco zjednoduší, zároveň i zavede komplexitu do jiné části kódu [4].



Obrázek 2.10: UML diagram tříd pro strategy pattern

Na obrázku 2.10 je znázorněn UML diagram tříd pro strategy pattern. `Kontext` je třída, která pracuje pouze s rozhraním `Strategie`. Třídy `KonkretniStrategie` pak implementují `Strategie` a v metodě `do()` mají kód, který vykonává požadovanou funkčnost. `Kontext` nepotřebuje vědět, co přesně která `KonkretniStrategie` provádí, pouze stačí, že může volat metodu `do()`. Detaily si pak zajistí sama `KonkretniStrategie`.

2.3.1 Vytvoření rozšiřujícího modulu – před prací

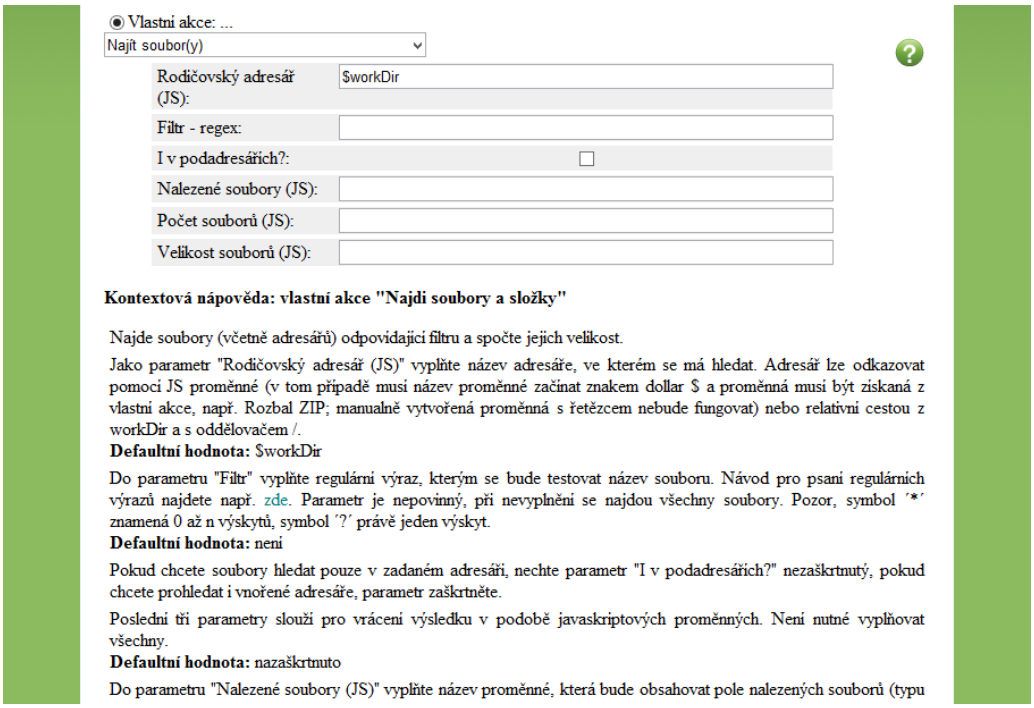
Pro vytvoření vlastní akce bylo třeba implementovat rozhraní `VlastniAkce` spolu se sedmi metodami.

1. **public String getHelp()**
   Metoda vrací řetězec, který obsahuje nápovědu pro vlastní akci (obr. 2.11). Řetězec bylo třeba skládat programově, tzn. bylo třeba ručně psát tagy HTML elementů a zalamovat text řetězce, což nebylo moc přehledné ani příjemné.

2. **public String getId()**
   Metoda vrací název pro akci. Tento název se ukládá do XML souboru obsahující nastavení domény (viz ukázka 2.2).

3. **public String getKategorii()**
   Metoda vrací název kategorie, přičemž tato kategorie musela být nadefinována v rozhraní `VlastniAkce`.

4. **public String getNazev()**
   Metoda vrací název akce pro zobrazení ve webovém rozhraní. Nemusí být totožné s `getId()`.



Obrázek 2.11: Nápověda vlastní akce ve webovém rozhraní

5. **public String getPopis()**
   Metoda vrací popis akce.

6. **public List<ParametrAkce> getParametry()**
   Na obrázku 2.11 jsou vidět parametry akce. Momentálně existují pouze 3 druhy parametrů: řetězec, checkbox a odkaz na nahrání souboru.

7. **public void execute()**
   V této metodě se provádí hlavní kód validace. Metoda nemá žádnou návratovou hodnotu jako takovou, ale modifikuje objekt, který je parametrem metody. Vstupními parametry jsou:

   - `ValidationInfo info`,
   - `FullValidationResult result`,
   - `Scriptable scriptable`,
   - `Collection<Parametr> parameters`.

   `ValidationInfo` je objekt, který poskytuje informace a data o konkrétní instanci validace popř. obecné informace o validačním serveru.

Většinou se tento objekt používá k získání nahraného souboru studentem a získání pracovního adresáře validace.

`FullValidationResult` představuje výsledek validace. Objekt je předáván do všech kroků validace a ty do něho mají možnost generovat tři druhy zpráv. Metodou `result.addInfo()` se přidává řetězec informativního charakteru. Informativní zpráva se zobrazuje jako modrý pruh (obr. 2.12) a nemá žadný vliv na validaci jako takovou, pouze slouží k vypsání nějaké informace pro uživatele.

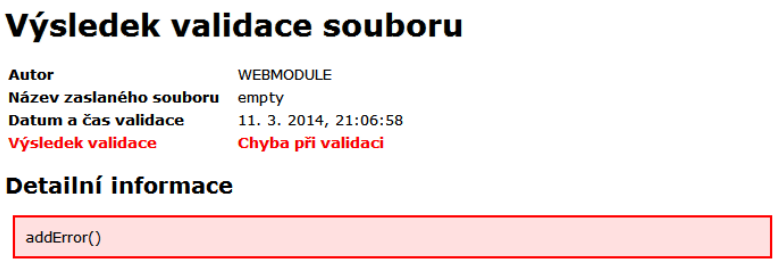
Obrázek 2.12: Informační zpráva ve výsledku validace

Metodou `result.addWarning()` se přidává varovná zpráva. Varovná zpráva také nemá žádný vliv na samotnou validaci. Zobrazuje se jako oranžový pruh (obr. 2.13) a většinou se používá pro případy, když se objeví nějaký problém, který není kritický, ale student by ho měl opravit.

Obrázek 2.13: Varovná zpráva ve výsledku validace

Poslední typ zprávy, chyba, se přidává metodou `result.addError()`. Chybová zpráva narozdíl od dvou předchozích už má vliv na validaci. Pokud po vykonání všech kroků validace obsahuje výsledek validace chybu, validace bude vyhodnocena negativně a studentova práce bude odmítnuta. Validace s chybou ve výsledku pokračuje až na poslední krok. Někdy je však třeba validaci ukončit už dříve, protože další akce

jsou zpravidla závislé na předchozích a nebudou tak mít správný vstup. Pro tento případ se využije podmínka „Validace již obsahuje chybu“ (obr. 2.7).



Obrázek 2.14: Chyba ve výsledku validace

`Scriptable` poskytuje přístup k javascriptovým proměnným, které existují v rozsahu instance validace. Lze tak získávat hodnoty proměnných, ale i přidávat další proměnné. Přidané proměnné budou pak viditelné v dalším kroku, kde je lze použít pro podmínku nebo parametr akce.

`Context` představuje kontext běhového prostředí javascriptu a obsahuje informace o právě prováděném skriptu jako je například zásobník volání funkcí.

`Collection<Parametr> parameters` je kolekce, která obsahuje vstupní parametry akce. Jedná se o dvojice klíč–hodnota. Implementačně je použito rozhraní `Collection` a ne `Map`, tudíž nelze hodnotu získat jedním voláním, ale je třeba projít celou kolekci a explicitně se ptát, zda se jedná o deklarovaný parametr. Parametry se deklarovaly jako konstantní řetězce vlastní akce.

Parametrem akce může být buď řetězec, checkbox nebo speciální nahrávací tlačítko. V případě checkboxového parametru je chování takové, že pokud je checkbox uživatelem zaškrtnut, pak je tento klíč v kolekci a pokud není zaškrnutý, tak se v kolekci vůbec nevyskytuje. Naopak řetězcový parametr se v kolekci vyskytuje vždy, pouze se jedná o prázdný řetězec, pokud uživatel nic nevyplní.

Nahrávací parametr slouží pro nahrání archivu do validační domény. Archiv se pak vždy rozbalí do složky `validation_data`, kde můžou být soubory čteny ve vlastní akci. Tento parametr už se tolik nevyužívá, protože při nahrání dojde k přemazání dosavadního obsahu složky `validation_data`.

2.3.2 Apache Maven

Apache Maven je nástroj pro správu Java projektů, který využívá projektově-objektový model, sadu standardů, životní cyklus projektu, systém pro správu závislostí a logiku pro vykonávání cíle pluginů v definovaných fázích životního cyklu projektu [2].

Rozšiřující moduly se na server nasazují jako `jar` archiv, který obsahuje zkompilované třídy vytvořené vlastní akce. Tento `jar` archiv musí splňovat jedinou podmínku, aby validátor byl schopný modul načíst – musí obsahovat soubor `MANIFEST.MF`, ve kterém je řádka:

```
AKCE: cely.nazev.tridy.vcetne.Baliku
```

Pro zjednodušení a automatizaci vytváření `jar` archivů se všechny vlastní akce vytváří jako Maven projekty.

Všechny vlastní akce mají svůj adresář s následující strukturou:

```
Adresář vlastní akce
├── src
│   └── main
│       ├── java
│       └── resources
├── target
└── pom.xml
```

Složka `java` obsahuje zdrojové soubory. Složka `resources` obsahuje ostatní vedlejší zdroje jako například konfigurační textové soubory. Do složky `target` se generuje tzv. artifakt, neboli výsledek Maven sestavení. Nejdůležitější pro celý Maven projekt je soubor `pom.xml`. V tomto souboru se specifikuje projekt, jaké závislosti se v projektu používají a jakým způsobem má být projekt sestaven. V ukázce 2.3 je příklad velmi prostého `pom.xml`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven
      -4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>cz.zcu.validator.actions</groupId>
  <artifactId>ukazka</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>
  <name>ukazka</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>cz.zcu.validator</groupId>
      <artifactId>VS-SDK</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>org.mozilla</groupId>
      <artifactId>rhino</artifactId>
      <version>1.7R4</version>
    </dependency>
  </dependencies>
</project>
```

Ukázka 2.3: Struktura `pom.xml`

Maven provádí svoji činnost pomocí pluginů. Ty se starají o kompilaci zdrojových kódů, vytváření balíčků, nasazování na servery a o ostatní činnosti, které jsou nutné pro sestavení projektu. V ukázce 2.3 je plugin, který se stará o kompilaci zdrojových souborů.

Do elementu `dependecies` se píší jednotlivé závislosti projektu. Závislosti jsou jiné knihovny, které využívá náš projekt, aby mohl fungovat. Pokud například v naší vlastní akci budeme chtít převést binární data do kódování base64, tak můžeme využít knihovnu Apache Commons Codec, která poskytuje statickou metodu `Base64.encodeBase64String()`. V ele-

mentu `dependencies` by pak přibyl záznam pro přidání závislosti na knihovně Apache Commons Codec (ukázka 2.4).

```
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.9</version>
</dependency>
```

Ukázka 2.4: Závislost na knihovně Apache Commons Codec

Závislosti se přidávají z uložiště závislostí – repository. Repository se dělí na dva typy: lokální a vzdálené. Maven se nejdříve snaží závislost najít v lokálním repository a pokud ji tam nenajde, pokusí se ji stáhnout ze vzdáleného.

Šablonový `pom.xml` pro vlastní akce je možné stáhnout v souborech na Redmine provozované KIV.

Vývojář vlastní akce pak pouze zadá příkaz pro sestavení projektu a Maven už sám pro vlastní akci vygeneruje nasaditelný `jar` soubor. Tento soubor stačí překopírovat do složky `/valid-common/libs/plugins` na validačním serveru a restartovat server. Po restartu by nově přidaná akce měla být vidět v seznamu vlastních akcí.

2.4 Řízení vývoje validačního serveru

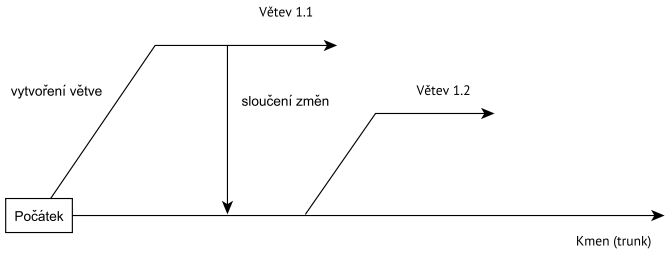
2.4.1 Systém pro správu verzí

Systémy pro správu verzí slouží pro správu změn dokumentů, počítačových programů a jiných druhů informací. Systém udržuje v uložišti (repository) seznam souborů, které spadají pod jeho správu (jsou verzovány). Určitá konfigurace těchto souborů tvoří dohromady revizi, která je charakterizována číslem či písmenem. Pokud dojde ke změně verzovaného souboru, revize se navýší či změní. Revize mohou být porovnávány, obnovovány či slučovány [7].

Při vývoji software je takový systém nepostradatelný, protože výrazně zjednodušuje udržování verzí a vývoj kódu, zejména pokud se na vývoji podílí více programátorů. Verzovací systémy umožňují vytváření vývojových větví

(branch) a vývojových štítků (tag).

Hlavní vývojové větvi se říká kmen (trunk). Z kmene se pak větví další verze, když je třeba vyvinout paralelně novou funkcionalitu nebo když hrozí, že prováděné změny ponechají kmen delší dobu v nefunkčním stavu. Po dokončení vývoje ve vývojové větvi se nová funkcionalita sloučí opět do kmene. Na obrázku 2.15 je schéma větvení a slučování změn v průběhu času.



Obrázek 2.15: Větvení a slučování změn ve verzovacím systému v průběhu času

Štítkování slouží pro snáze zapamatovatelnější označení specifických revizí, například když se jedná o verzi, která bude uvolněna pro veřejnost. K oštítkované verzi se lze kdykoliv vrátit. Systémy pro správu verzí fungují i jako zálohy pro případ, že je třeba získat určitou konfiguraci souborů z minulosti.

Nejdůležitějšími operacemi verzovacího systému jsou commit/check in a update/check out. Operace commit slouží k nahrání lokálně upravených souborů do repository. Operace update je přesným opakem commit. Update provádí aktualizaci lokálních souborů porovnáním se soubory v repository.

Vývojář na samotném začátku vývoje udělá prvotní update, čímž vytvoří lokální kopie souborů. S těmito lokálními nadále pracuje a provádí v nich změny. Po dokončení vývoje nebo na konci dne, pokud vývoj bude trvat déle, provede vývojář commit, čímž se nahrajou provedené změny do repository. Z repository pak získávají ostatní vývojáři operací update už upravené soubory. Update provádí každý programátor na začátku každého vývojového kola, aby měl aktuální verzi souborů. Pokud to neudělá nebo pokud se soubor v repository během lokálního vývoje změní, může nastat konflikt.

Konflikt je situace, kdy vývojář chce provést commit a lokální verze souboru se liší od verze v repository. Řešení konfliktu jsou následující [10]:

- zahození lokálních změn, přijmutí verze z repository a provedení změn opětovně,
- přemázání verze v repository lokální verzí,
- sloučení obou verzí.

Před touto prací se verzovaly pouze zdrojové kódy samotného validačního serveru. Uložiště bylo spravováno CIV a používal se verzovací systém SVN. Vlastní akce nebyly verzovány a jejich kód si udržoval každý programátor validátoru individuálně.

Na začátku této práce se začali společně s validačním serverem verzovat i vlastní akce. Pro verzování se stále používá systém SVN, tentokrát je však systém uložen na serveru spravovaného Katedrou informatiky a výpočetní techniky ZČU. Nový programátor validačního serveru či vlastních akcí si může kdykoliv pomocí operace update stáhnout aktualní verzi zdrojových souborů z repository a začít pracovat. Adresa repository pro tento projekt je: `https://forge.kiv.zcu.cz/svn-validator`.

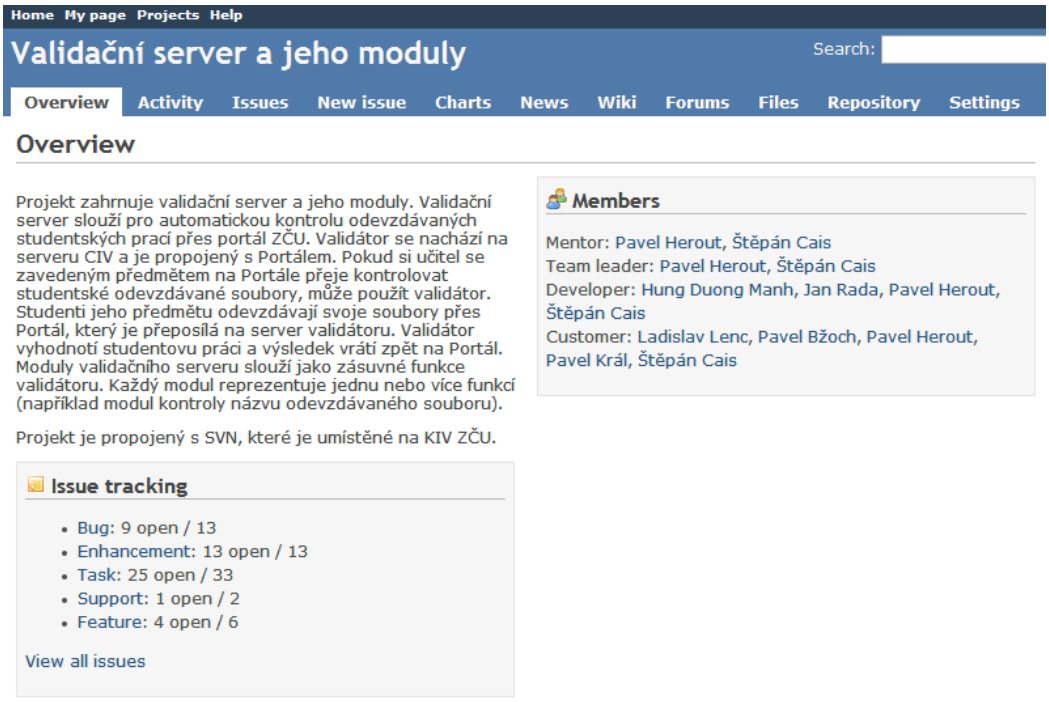
2.4.2 Systém pro správu požadavků

Systém pro sledování požadavků je nástroj v softwarovém inženýrství, který slouží pro sběr požadavků, jejich správu a sledování jejich vývoje [5]. Požadavkem se rozumí jakýkoliv požadavek na změnu vyvíjeného software.

Pro projekt validačního serveru a jeho vlastních akcí se využívá systém Redmine. Redmine je bezplatný open source nástroj na řízení projektu, který zahrnuje i systém pro správu požadavků. Redmine vzniknul v roce 2006 a je naprogramovaný pomocí frameworku Ruby on Rails. Předností Redmine je jednoduchost, přehlednost a integrace s verzovacím systémem [8]. Na obrázku 2.16 je úvodní obrazovka pro tento projekt v Redmine.

Použití Redmine vede k tomu, že v našem projektu lze:

- vyhledávat v celé historii úkolů,
- dohledat jak byly úkoly vyřešeny,
- organizovat plánování do budoucna,



Obrázek 2.16: Úvodní obrazovka v Redmine pro projekt „Validační server a jeho moduly“

- zjistit, které úkoly se mají momentálně řešit,
- vytvářet a delegovat úkoly na konkrétní řešitele,
- mít jednotné místo s přístupem k informacím projektu,
- měřit efektivitu.

Tato práce byla v rámci všech prací na validátoru první, u které byl pro řízení vývoje použit systém pro správu požadavků. Při řízení vývoje projektu byl Redmine použit pro správu požadavků s integrací na SVN a jako informační báze. Adresa KIV Redmine je: `http://students.kiv.zcu.cz:3000`.

Systém pro správu požadavků v Redmine

Všechny úkoly (v Redmine nazvané jako issue) související s vývojem validačního serveru a jeho modulů jsou nově vedeny v Redmine. Na obrázku 2.17 je obrazovka Redmine s výpisem úkolů.

Úkoly v projektu „Validační server a jeho moduly“ mohou být typu:

- bug,
- enhancement,
- task,
- support,
- feature.

Obrázek 2.17: Obrazovka Redmine s výpisem úkolů

Rozdíly mezi jednotlivými typy jsou minimální. Typy slouží pro kategorizaci. Každý úkol v projektu musí mít povinně typ, předmět, popis, stav a prioritu.

Pro nově vytvořené úkoly se používá stav New. Pro úkoly, které mají už nekoho přiřazené pro vyřešení, se používá stav Assigned. Pokud je úkol vyřešen, ale je třeba ho ještě ověřit, použije se stav Resolved. Stav Verified

označuje úkol, které jsou vyřešeny a ověřeny. Posledním stavem je Closed. Úkoly v tomto stavu už by se neměly měnit. Možné použitelné stavy jsou nastavitelné podle rolí v systému. Výše uvedené stavy se vztahují na použití stavů v tomto projektu.

Redmine umožňuje integraci s verzovacím systémem SVN. Při commitu lze do zprávy commitu psát klíčová slova pomocí kterých lze pracovat s úkoly. Obsahuje-li například zpráva commitu text:

```
references #1
```

bude u úkolu s číslem 1 uveden odkaz na tento commit v sekci Související revize. Dalším klíčovým slovem je například `closes`, které automaticky úkolu nastaví stav Closed.

Celý životní cyklus jednoho úkolu v projektu „Validační server a jeho moduly“ vypadá zhruba takto. Uživatel vytvoří úkol, kde popíše námět na vylepšení, nastaví prioritu a ohodnotí časovou náročnost popřípadě ještě nastaví konkrétního řešitele (pokud ví, komu je tento úkol určen). Až se řešitel dostane k tomuto úkolu a ověří, že má smysl se zabývat tímto úkolem, nastaví stav úkolu na Assigned. V průběhu vývoje/řešení úkolu řešitel průběžně aktualizuje procentuální stav dokončení a pokud je to třeba, píše komentáře k úkolu. Pokud je s úkolem spojena nějaká změna ve zdrojových kódech, napíše řešitel do zprávy commitu odkaz na číslo úkolu. Při dosažení 100 % dokončení nastaví řešitel stav na Resolved a vyčká na kontrolu zadavatelem. Pokud je zadavatel spokojen, změní stav na Resolved, v opačném případě resetuje % dokončení a nastaví stav na Assigned, dokud není spokojen. Stav Closed nastaví zadavatel obvykle po ověření funkčnosti s novou verzí validačního serveru.

Informační báze v Redmine

Redmine poskytuje k projektu také wiki. Wiki je webová aplikace, která umožňuje kolaborativní úpravu obsahu. Wiki může sloužit jako encyklopedie pojmů pro projekt či místo pro sepisování návodů. Obecně je to místo, které shromažďuje všechny informace spojené s nějakým tématem.

Kolaborativní přístup umožňuje jakémukoliv členovi projektu upravovat obsah ostatních nebo také vytvářet nový obsah. Pokud například bude v

návodu chyba či nějaká nejasnost, stačí stránku s návodem poupravit.

Během práce na tomto projektu bylo ve wiki sepsáno několik návodů týkajících se validačního serveru a jeho modulů. Návody pokrývají postup od přípravy lokálního vývojového prostředí pro validační server až po nasazení vlastních akcí a validačního serveru.

Na obrázku 2.18 je úvodní obrazovka pro wiki v Redmine. Redmine umožňuje fulltextové vyhledávání v celém projektu. Tudíž celý projekt společně s wiki a úkoly tvoří jednu velkou informační bázi.

Obrázek 2.18: Obrazovka wiki v Redmine pro projekt „Validační server a jeho moduly“

3 Analýza námětů na vylepšení a plán řešení

Na začátku této práce v létě 2013 bylo v Redmine sepsáno 17 úkolů týkajících se validačního serveru a jeho modulů. Během práce počet úkolů v Redmine rostl. Celkem jsem stihnul dokončit 34 úkolů, přičemž 17 úkolů bylo přidáno během řešení. Tyto přidané úkoly nebyly z počátku v plánu a představují tak práci navíc z mé strany. V tabulce 3.1 jsou přidané úkoly označeny hvězdičkou. Úkol 16 „Rozmyslet pořadí plnění úkolů“ je obsažen v kapitole 3.2.

Tabulka 3.1: Seznam úkolů v pořadí přidání

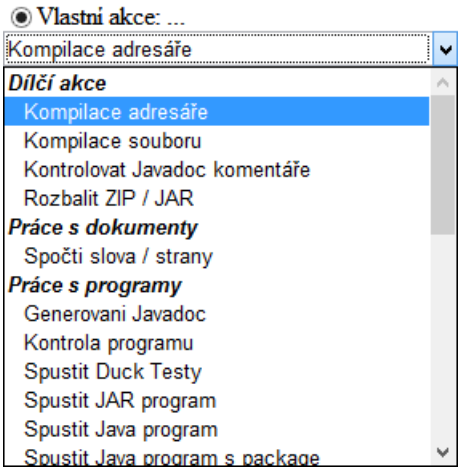
Pořadí přidání	Název úkolu
1	Parametrizace odkazem u vlastních akcí
2	Refaktor akce Spuštění JUnit
3	Vyzkoušení a popsání akce na kontrolu počtu stránek a slov
4	Refaktorování akce SpustitPMD a SpustitUMLTestovani
5	Úprava souborové struktury vlastních akcí
6	Úprava vlastních akcí na použití checkboxů
7	Změnit použití nápovědy u vlastních akcí
8	Vytvoření akce Vymaž soubory podle vzoru
9	Předdefinované hodnoty parametrů
10	Kategorizace vlastních akcí
11	Doplnění serveru / util class o standardně používané akce
12	Úprava pom.xml souboru
13	Vyřešit umístění pomocných souborů pro jednotlivé ukázky
14	Domény otestovatelné bez ukázkových souborů
15	Doplnění Wiki o procesu práce s validátorem
16	Rozmyslet pořadí plnění úkolů
17	Nápověda na vytváření akcí do Wiki
18*	Dynamické přidávání dalších kategorií
19*	ukazka-kontrolanepovolenychsouboru
20*	ukazka-kopirovatslozkydoworkdir
21*	ukazka-najdisoubory
22*	ukazka-porovnanistrukturyadresaru
23*	Přidání nových položek do rozhraní ValidationResult
24*	Přejmenovat vlastní akci spoctislova na spoctislovaastrany
25*	Vytvoření nové třídy AbstraktniVlastniAkce
26*	String index out of range: -86
27*	Nejde vlastní akce Velikost souboru
28*	Jak nastavit počáteční adresář pro Javadoc?
29*	OK hlášení JUnit testu metody
30*	Vlastní akce Kopírování souborů
31*	Nová vlastní akce generovanijavadoc
32*	Nefungujici domena a prapodivne chovani validatoru
33*	Chyba při rozbalování JAR
34*	Chybné vyhodnocení výsledku duck-testu

3.1 Analýza námětů na vylepšení

3.1.1 Kategorizace vlastních akcí

Cílem úkolu bylo zkontrolovat správné přiřazení všech existujících vlastních akcí do předdefinovaných kategorií.

Každá vlastní akce spadá kvůli přehlednosti do určité kategorie. Na obrázku 3.1 je zobrazení kategorií ve webovém rozhraní validačního serveru.



Obrázek 3.1: Zobrazení kategorií ve webovém rozhraní validátoru

Kategorie se u vlastní akce nastavuje v metodě `getKategorii()`, která vrací řetězec s názvem kategorie. Tato metoda však může vracet pouze předdefinové řetězcové konstanty, které jsou nadefinované v třídě `VlastniAkce` (obr. 3.1).

```
public final String KAT_DOKUMENT = "Prace s dokumenty";
public final String KAT_PROGRAM = "Prace s programy";
public final String KAT_SOUBOR = "Prace se soubory";
public final String KAT_DILCI = "Dilci akce";
```

Ukázka 3.1: Předdefinové kategorie vlastních akcí

Tyto řetězcové konstanty jsou použity při generování seznamu vlastních akcí na obrazovce „Upravit krok“.

3.1.2 Změnit použití nápovědy u vlastních akcí

Cílem úkolu bylo přesunout text nápovědy vlastních akcí do samostatného souboru, aby nebylo třeba modifikovat zdrojový kód akce v případě úpravy nápovědy. Při implementanci mělo být zváženo použití knihovny z projektu Apache Commons.

Apache Commons je projekt, který se zaměřuje na vytváření znovupoužitelných knihoven v Javě [1]. Jednou takovou knihovnou je Apache Commons Configuration, která je určená pro načítání konfiguračních souborů.

Knihovna poskytuje obecné rozhraní, přes které lze načítat parametry uložené v různých zdrojích. Zdrojem může být properties soubor, XML dokument či Windows INI soubor. V ukázce 3.2 je předveden způsob práce s knihovnou.

```
Configuration config = new XMLConfiguration(new File("config.xml"));
String parametrA = config.getString("parametrA");
```

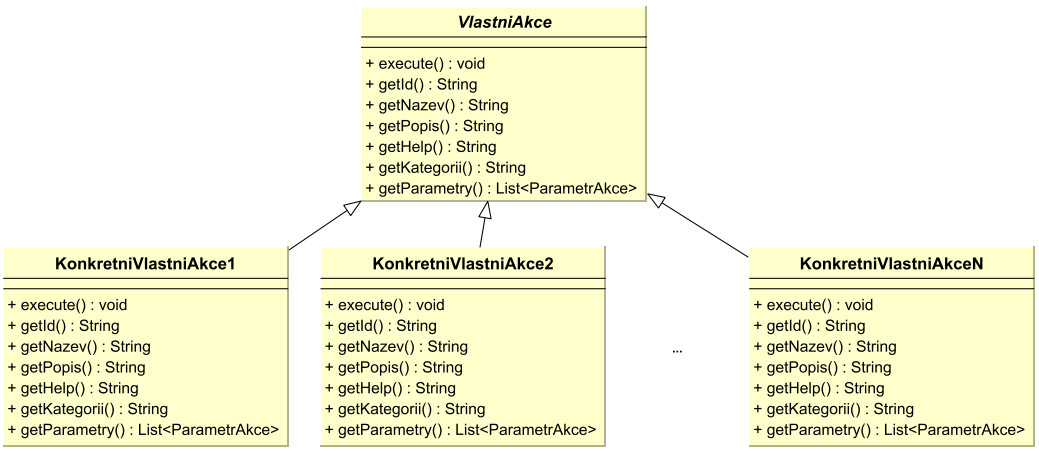
Ukázka 3.2: Získání parametru z XML dokumentu

Nápověda k vlastním akcím se získává jako návratová hodnota metody `getHelp()`, jejíž hlavička je v rozhraní `VlastniAkce`. Metoda vrací řetězec, ve kterém je celá nápověda včetně HTML formátování.

3.1.3 Vytvoření nové třídy AbstraktniVlastniAkce

Cílem úkolu bylo vytvořit novou abstraktní třídu `AbstraktniVlastniAkce`, která by umožnila definovat společné chování pro všechny vlastní akce, a implementovat do této abstraktní třídy načítání nápovědy a řetězcových konstant ze souboru.

Všechny vlastní akce používají rozhraní `VlastniAkce`, aby je validační server byl schopen rozpoznat jako vlastní akce a použít v rámci kroku validace. Rozhraní však umožňuje pouze definovat hlavičky metod, ne jejich implemenentace. Na obrázku 3.2 je znázorněn diagram tříd pro rozhraní `VlastniAkce`.



Obrázek 3.2: UML diagram tříd pro rozhraní `VlastniAkce`

3.1.4 Refaktor akce „Spuštění JUnit“

Cílem úkolu bylo přepsat vlastní akci `SpusteniVsechJunitTestu` tak, aby prováděla pouze samotné spuštění JUnit testů.

Tato vlastní akce byla původně vytvořena na míru pro předmětu KIV/-OOP a prováděla několik činností:

1. kopírování `JUnit` adresáře z adresáře validační domény do pracovního adresáře,
2. kopírování `Framework` adresáře z adresáře validační domény do pracovního adresáře,
3. kompilaci zdrojových souborů v pracovním adresáři,
4. spuštění JUnit testů v souborech, které začínaly slovem „Test“ a končily příponou class.

Pro činnosti 1, 2, 3 existují už samostatné vlastní akce - „Kopírovat adresáře do workdir“ a „Kompilace adresáře“.

3.1.5 Nápověda na vytváření akcí do Wiki

Cílem úkolu bylo sepsat do Redmine wiki nápovědu, jak vytvářet nové vlastní akce využívající Maven. Redmine wiki je dostupná na adrese `http://students.kiv.zcu.cz:3000/projects/validator/wiki`.

U tohoto úkolu se vybízelo, aby byl proveden dřív. Uznal jsem ale za vhodnější napsat nápovědu až po větším obeznámení se s celkovým pracovním postupem.

3.1.6 Úprava pom.xml souboru

Cílem bylo poupravit u všech vlastních akcí soubor `pom.xml`, aby se do generovaného, nasaditelného jar souboru nedostaly nepotřebné věci.

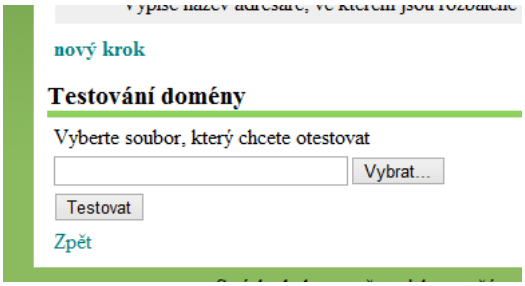
V úkolu zmínil zadavatel pouze, že se do generovaného jar souboru přidávají soubory ze složky `resources`. Prvotní analýza neodhalila v generovaných souborech žádné nepotřebné soubory navíc. Moje hypotéza je taková, že problém byl způsoben nevhodným pojmenováním složky `resources` pro různé pomocné soubory, a proto se jakýkoliv obsah složky `resources` přidával do generovaného jar souboru. Po implementaci úkolu „Vytvoření nové třídy AbstraktniVlastniAkce“ se však tato složka začala využívat korektním způsobem pro přibalování konfiguračního souboru a nápovědy, a ostatní soubory kromě těchto dvou byly přesunuty do jiných složek. Ve výsledku se tak tento úkol vyřešil společně s úkolem „Vytvoření nové třídy AbstraktniVlastniAkce“.

3.1.7 Domény otestovatelné bez ukázkových souborů

Cílem úkolu bylo umožnit testování domén bez posílání souboru na server.

Validační domény lze testovat ve webovém rozhraní. K otestování musí uživatel vždy nahrát soubor, který chce otestovat, což odpovídá reálnému využití – validátor se používá při odevzdávání práce studentem. Na obrázku 3.3 je tlačítko pro otestování domény ve webovém rozhraní.

U některých ukázkových domén by byla užitečná možnost otestovat doménu bez vybrání souboru, protože dané domény žádný vstupní soubor nepotřebují a pracují pouze s předpřipravenými soubory.



Obrázek 3.3: Testování domény ve webovém rozhraní

3.1.8 Vytvoření akce Vymaž soubory podle vzoru

Cílem úkolu bylo vytvořit novou vlastní akci, která by uměla smazat soubory v konkrétním adresáři podle vzorového adresáře.

V předmetu KIV/OOP studenti často odevzdávají archivy, které obsahují jak jejich vlastní zdrojové kódy, tak jiné zdrojové kódy, které potřebují v rámci celé odevzdávané práce. Studenti mají za úkol například vytvořit jednotkové testy k poskytnutým třídám. Na validačním serveru se pak kompilují a testují všechny zdrojové soubory obsažené v odevzdaném archivu. Může však nastat situace, že by si student upravil zdrojové kódy poskytnutých tříd tak, aby testy prošly. Z tohoto důvodu by se hodila vlastní akce, která by smazala pomocné soubory, které pak mohou být nahrazeny soubory, o kterých máme jistotu, že nebyly upraveny.

3.1.9 Vyzkoušení akce na kontrolu počtu stránek a slov

Cílem úkolu bylo vyzkoušet funkčnost vlastní akce „Spočti slova“ a napsat nápovědu k této vlastní akci.

Vlastní akce „Spočti slova“ slouží ke zjištění počtu slov a stran v dokumentech typu doc a pdf. Akce má tři parametry.

První parametrem – „Soubor (JS)“ – je soubor, který se má analyzovat. Pokud parametr není zadán, použije se jako hodnota odevzdávaný soubor. V opačném případě musí být v parametru uvedena javascriptová proměnná, která v sobě obsahuje objekt typu `java.io.File`. Druhým parametrem – „Počet slov (JS)“ – je výstupní javascriptová proměnná, do které se má uložit počet slov nalezených v dokumentu. Výstupní proměnné se vytvoří v

rámci běhu akce. Třetím parametrem – „Počet stran (JS)“ – je výstupní javascriptová proměnná, do které se má uložit počet stran nalezených v dokumentu. Pro správné fungování vlastní akce musí být uveden alespoň jeden z parametrů „Počet stran (JS)“ nebo „Počet slov (JS)“.

3.1.10 Úprava souborové struktury vlastních akcí

Cílem úkolu bylo, aby všechny existující vlastní akce měly v SVN uložené i texty a soubory z ukázkových domén.

Soubory, které se využívají v ukázkových doménách, a popisy kroků ukázkových domén nebyly nijak verzovány. Přístup k ukázkovým doménám dostávají zpravidla všichni uživatelé testovacího validátoru. Kdyby se stalo, že by někdo ukázkovou doménu smazal, byly by nenávratně smazány i tyto soubory a texty, a celá ukázková doména by se musela tvořit znovu. Proto by u sebe měly mít vlastní akce i tyto ostatní zdroje související s ukázkovou doménou, která se k nim váže.

3.1.11 Refaktorování akce SpustitPMD a SpustitUMLTestovani

Cílem úkolu bylo refaktorovat akce „SpustitPMD“ a „SpustitUMLTestovani“ do Maven formátu.

Tyto dvě vlastní akce jako jediné ještě nebyly převedeny do Maven formátu. Netýkal se jich proto ani úkol „Vytvoření abstraktní třídy AbstraktniVlastniAkce“, kdy se text nápovědy a konfigurace vlastní akce přesunuly do souborů v adresáři `resources` (bez Maven formátu by se do nasaditelného jar souboru nepřídávaly automaticky soubory z `resources` a akce by nefungovala správně, protože by se nenačetl konfigurační soubor).

3.1.12 Doplnění serveru / util class o standardně používané akce

Cílem úkolu bylo vytvořit na straně serveru novou pomocnou třídu pro často používáné činnosti ve vlastních akcích.

Často používanou činností je například získání objektu typu `java.io.File` z řetězce, který uživatel zadal v parametru vlastní akce či převod regulárního výrazu zapsaného v DOS tvaru do tvaru, se kterým může pracovat standardní Java metoda `Pattern.matches()`.

3.1.13 Doplnění Wiki o procesu práce s validátorem

Cílem úkolu bylo napsat návody do wiki, které popisují kompletně celý proces práce s validátorem.

Do wiki bylo třeba dopsat ještě tyto návody:

- jak spustit validátor na lokálním PC,
- jak provést build validátoru na lokálním PC,
- jak nasadit na server novou verzi validátoru.

3.1.14 Vyřešit umístění pomocných souborů pro jednotlivé ukázky

Cílem úkolu bylo provést analýzu, kam by bylo nejlepší uložit soubory používáné v ukázkových doménách pro veřejné stažení.

Ukázkové soubory byly umístěny ke stažení na adrese home.zcu.cz/∼scais, což je univerzitou poskytnutý domovský adresář pro Orion uživatele scais.

3.1.15 Úprava vlastních akcí na použití checkboxů

Cílem úkolu byla úprava parametrů typu ano/ne na checkboxové parametry ve všech vlastních akcích a přepsat odpovídajícím způsobem nápovědu k těmto parametrům.

Doposud byly ve všech vlastních akcích parametry typu ano/ne implementovány tak, že uživatel musel do textového pole zadávat konkrétního hodnotu – ano/true či ne/false. Takto zadané hodnoty uživatelem se následně ještě ve

zdrojovém kódu validovaly a až poté bylo možné nastavit hodnotu boolevské proměnné. Textové zadávání pro parametry typu ano/ne je uživatelsky neintuitivní a technicky nevhodné.

3.1.16 Předdefinované hodnoty parametrů

Cílem úkolu bylo umožnit u všech parametrů vlastních akcí nastavit předdefinové hodnoty.

Při výběru vlastní akce ve webovém rozhraní se textové pole či checkboxy vygenerují prázdné. U některých parametrů není vždy zřejmé, co může uživatel zadat bez toho, aniž by se podíval do nápovědy, nebo se ve většině případů používá stejná hodnota, takže uživatel je nucen ji opakovaně zadávat. Předdefinované hodnoty parametrů by toto vyřešily.

3.1.17 Parametrizace odkazem u vlastních akcí

Cílem úkolu bylo umožnit u vybraných parametrů vlastních akcí referencovat hodnoty javaskriptových proměnných, přičemž referencovat by bylo možné pouze javaskriptové proměnné, které v sobě mají uložen objekt typu `java.io.File`. Příkladem takové akce je „Najdi soubory“, která slouží pro vyhledání souborů v samotném pracovním adresáři či jeho podadresáři.

Některé vlastní akce potřebují v parametru uvést cestu ke konkrétnímu souboru. Cestu ale nelze uvést napevno, protože název pracovní složky se mění s každou novou validací. Existují vlastní akce, jejichž výstupem je javaskritpová proměnná, která obsahuje objekt typu `java.io.File` – například „Rozbal ZIP / JAR“ či „Najdi soubory“.

3.1.18 Dynamické přidávání dalších kategorií

Cílem úkolu byla možnost dynamicky definovat kategorie vlastních akcí. Tento úkol navazuje na úkol „Kategorizace vlastních akcí“.

3.1.19 ukazka-kontrolanepovolenychsouboru

Cílem úkolu bylo přidat do ukázkového archivu pro doménu „ukazka-kontrola-nepovolenychsouboru“ soubor `nazdar.txt`, který má ověřit, že akce rozlišuje velikost písmen.

Vlastní akce „Kontrolovat nepovolené soubory“ funguje tak, že prohledává v zadaném adresáři výskyt nepovolených typů souboru, které jsou zadány parametrem. Pokud akce nalezne nepovolený typ, přidá do výsledku validace chybu. Prohledávání souborů rozlišuje velikost písmen.

Ukázková doména má v parametru uvedeno `TXT`. Při běhu ukázkové domény by proto soubor `nazdar.txt` měl být ignorován.

3.1.20 ukazka-kopirovatslozkydoworkdir

Cílem úkolu bylo upravit text, který se vypisuje při úspěšném zkopírování složek akcí „Kopírovat složky do workdir“.

Akce při úspěšném zkopírování složek vypisovala text: „Adresáře byly úspěšně zkopírovány“. Správně by však měla vypisovat: „Složky byly úspěšně zkopírovány“.

3.1.21 ukazka-najdisoubory

Při použití akce „Najdi soubory“ se do JS proměnné ukladá plná cesta souboru. Při použití akce „Najdi adresáře“ se do JS proměnné ukládá pouze název adresáře. Cílem úkolu bylo rozhodnout, který z přístupů je vhodnější a upravit podle rozhodnutí druhou akci.

3.1.22 ukazka-porovnanistrukturyadresaru

Cílem úkolu bylo přidat do ukazkové domény „ukazka-porovnanistruktury-adresaru“ soubor navíc, aby bylo možné otestovat funkčnost přepínače „Jsou extra soubory chyba?“.

Doména „ukazka-porovnanistrukturyadresaru“ má dva kroky. V prvním kroku se z adresáře domény zkopírují do pracovního adresáře dva podadresáře: `vzorovyAdresar` a `porovnavanyAdresar`. V druhém kroku je použita vlastní akce „Porovnat strukturu adresářů“. Akce má tři parametry: „Vzorový adresář (JS)“, „Porovnávaný adresář“ a „Jsou extra soubory chyba?“. Parametry „Vzorový adresář (JS)“ a „Porovnávaný adresář“ mají hodnotu `vzorovyAdresar` a `porovnavanyAdresar`.

Pokud bude v adresáři `porovnavanyAdresar` soubor navíc, měla by vlastní akce „Porovnat struktury adresářů“ při zaškrtnutí parametru „Jsou extra soubory chyba?“ vypsat do výsledku validace chybu.

3.1.23 Přidání nových položek do rozhraní ValidationResult

Cílem úkolu bylo přidání další položky `VR_TEST_NOT_PASSED` do seznamu stavů, kterými může validace skončit.

Validace může končit různými stavy. Implicitně se používá obecný stav `VR_OK`, který je určen pro validaci bez chyb, a `VR_BAD_RESULTS`, který nastává když se ve výsledku validace vyskytne chyba. Nový stav `VR_TEST_NOT_PASSED` by měl sloužit pro indikaci situace, kdy odevzdaný soubor neprošel automatickými testy.

3.1.24 Přejmenovat vlastní akci spoctislova na spoctislovaastrany

Cílem úkolu bylo přejmenovat vlastní akci „spoctislova“ na „spoctislovaastrany“, aby více odrážela její funkcionalitu.

3.1.25 String index out of range: -86

Cílem úkolu bylo vyřešit bug, který nastával při testování domény.

Bug se projevoval v prohlížečích Chrome 28 a Opera 15. V prohlížeči Firefox 22 se tento bug neprojevoval. Výpis chyby indikoval použití špatného

indexu, který byl mimo velikost zpracovávaného řetězce, a odkazoval do souboru `FileInput.java` na řádku 104. Na této řádce se prováděl kus kódu metody `getParameter()`, který je uveden v ukázce 3.3.

```
int end = content.indexOf("----------", pos);
```

Ukázka 3.3: Bug v metodě `getParameter()`

Metoda `getParameter()` slouží k získání parametru při poslání formuláře z klienta na server a funguje tak, že v HTTP požadavku najde index výskytu řetězce `Content-disposition: form-data; name="nazevParametru"`, od tohoto indexu hledá oddělovač dalšího parametru a nakonec vrátí hodnotu, která je o jednu řádku nad oddělovačem.

Ukázalo se, že vykreslovací jádro používané prohlížečem Chrome – WebKit (nyní už Blink), používá pro oddělování parametrů menší počet pomlček než ostatní jádra. Zatímco jádra ostatních prohlížečů používájí 10 pomlček, Webkit používá pouze 6.

3.1.26 Nejde vlastní akce Velikost souboru

Cílem úkolu bylo vyřešit bug, který se vyskytnul v kroku 3 domény „oop03“.

Popis bugu byl takový, že při editování kroku 3 se nezobrazily žádné parametry akce. Při pokusu o uložení kroku nastala vyjímka `java.lang.NullPointerException`.

Když jsem se snažil bug reprodukovat editováním kroku 3 v doméně „oop03“, nenarazil jsem na žádné problémy. Uložení také proběhlo v pořádku. Bug se následně již neprojevil. Po konzultaci s vedoucím práce byl bug uzavřen s tím, že problémy byly zřejmě způsobené souběhem několika událostí (restartování serveru, spouštění ukázkové domény).

3.1.27 Jak nastavit počáteční adresář pro Javadoc?

Cílem úkolu bylo poskytnout pomoc při nastavování vlastní akce „Kontrolovat Javadoc komentáře“. Při testování domény „oop03“ vyskočila zadavateli vyjímka `org.mozilla.javascript.EvaluatorException`.

U tohoto úkolu jsem opět nedokázal chybu reprodukovat. V logu jsem našel, že k vyjímce došlo, ale při spuštění stejné domény proběhlo vše v pořádku. Tento úkol i úkol „Nejde vlastní akce Velikost souboru“ byly v logu zaznamenány krátce po sobě, z čehož usuzuji, že vyjímky při testování byly následkem nesprávného běhu celého validátoru a po restartu už vše fungovalo jak mělo.

3.1.28 OK hlášení JUnit testu metody

Cílem úkolu bylo změnit výstup akcí „Spustit konkrétní JUnit test“ a „Spustit všechny JUnit testy“ tak, aby při úspěšném průběhu testů vypsaly hlášku „OK“ a „OK xy tests“.

Výstupem obou akcí jsou 2 soubory – jeden soubor pro chybový výstup a jeden soubor pro standardní výstup. Do standardního výstupu se při úspěšném průběhu všech testů nevypisoval žádný text.

3.1.29 Vlastní akce Kopírování souborů

Cílem úkolu bylo přidat do vlastní akce „Kopírovat soubory do workdir“ možnost kopírovat soubory z podsložky domény a možnost úvest soubory, které se mají zkopírovat, regulárním výrazem.

Vlastní akce „Kopírovat soubory do workdir“ měla pouze jeden parametr „Seznam souborů pro zkopírování“. Do tohoto parametru se uváděly konkrétní soubory, které se měly zkopírovat ze složky domény.

3.1.30 Nová vlastní akce generovanijavadoc

Cílem úkolu bylo dokončit vlastní akci „Generování Javadoc“ tak, aby fungovala.

Vlastní akce „Generování Javadoc“ měla sloužit k vygenerování Javadoc dokumentace zdrojových souborů na straně serveru a měla pouze jediný parametr. Tento parametr měl sloužit jako řetězec, který se přesně tak, jak je zapsán, předá příkazu javadoc, aby vygeneroval dokumentaci.

Akce nebyla dokončena a generování dokumentace nefungovalo při zadání parametru `-d / /*.java` – příkaz javadoc vypsal chybu, že soubor nelze najít. Analýzou chyby jsem došel k závěru, že příkaz javadoc se snažil najít soubor `*` místo aby interpretoval hvězdičku jako všechny soubory. Zkoumáním Linuxovského programu shell jsem zjistil, že při použití hvězdičky program shell automaticky dosazuje za hvězdčiku cestu ke všem souborům v daném adresáři.

3.1.31 Nefungující doména a prapodivné chování validátoru

Cílem úkolu bylo zjistit, co bylo příčinou divného chování validátoru, když s ním zadavatel pracoval.

Zadavatel popisuje, že při práci s validátorem najednou zmizela doména „oop01“. Zmizení nastalo po tom, co se zadavatel snažil přidat další krok do domény.

Při analýze jsem zjišťoval, jestli doména zmizela i z adresářové struktury serveru. Adresář pro doménu „oop01“ stále existoval, pouze se nezobrazoval. Jediný soubor, který se během úpravy domény mohl změnit a který mohl mít vliv na zobrazení domény ve webovém rozhraní, je soubor `webmodule.xml`.

Nevěděl jsem, kde konkrétně chyba byla, takže jsem si udělal zálohu souboru `webmodule.xml` a zkoušel upravovat soubor, abych lokalizoval chybu. Chyba byla v tom, že v jednom z kroků zadavatel zvolil možnost „vložení do výstupu validace“ a „vlastním skriptem“ a do vlastního skriptu zadal nevalidní hodnotu. Tato nevalidní hodnota byla statický text bez uvozovek.

3.1.32 Chyba při rozbalování JAR

Cílem úkolu bylo při rozbalování archivu akcí „Rozbalit ZIP / JAR“ odchytit vyjímku, která nastane při pokusu o rozbalení archivu s akcentovanými znaky.

Pokud byl vlastní akci předán archiv obsahují soubor nebo podsložku s akcentovanými znaky, byl tento soubor či složka při rozbalování ignorován. Archiv se rozbalil, ale defacto špatně a uživatel se to neměl možnost dozvědět,

dokud se nepodíval do souborové struktury validátoru.

Vlastní akce „Rozbalit ZIP / JAR“ využívá pro rozbalování pomocnou statickou metodu `ZipExtractor.extractZipArchive()` na straně serveru. Tato pomocná metoda už měla `catch` sekci, ktera odchytávala veškeré chyby během rozbalování a pouze vypsala chybu na chybový výstup.

3.1.33 Chybné vyhodnocení výsledku duck-testu

Cílem úkolu bylo upravit akci „Spustit Duck Testy“ tak, aby mohly `main()` metody testů vypisovat do standardního výstupu jiný text.

Akce „Spustit Duck Testy“ funguje následujícím způsobem:

1. rozbalení odevzdaného archivu,
2. spuštění main metod u všech souborů začínající slovem `TestDuck`,
3. vyhodnocení výstupů po spuštení main metod.

Třídy začínající `TestDuck` jsou JUnit testy napsány tak, že mají `main()` metodu, ve které provádí samotné testování. Při validaci je validátor pouze volá. V kroku číslo 3 pak vyhodnocování probíhá tak, že se kontroluje počet řádek výstupního souboru. Pokud se tedy v rámci `main()` metody vypíše do standardního výstupu nějaký test další text, zvýší se počet řádek a validace vrátí špatný výsledek. Aby mohly testy vypisovat do standardního výstupu další text, bude třeba změnit způsob vyhodnocování výstupního souboru.

3.2 Plán řešení

Rozmyslet pořadí plnění úkolů byl požadavek ze zadání diplomové práce a zároveň i úkol v Redmine. Proto tento úkol není uveden samostatně při analýze námětů na vylepšení v kapitole 3.1, ale je mu věnována samostatná kapitola.

Některé úkoly spolu souvisely nebo na sobě závisely a bylo nutné dopředu určit pořadí vykonání, aby se vzájemně negativně neovlivňovaly a aby nebylo třeba dělat nějaké činnosti vícekrát. Plán musel také brát v potaz vyšší

prioritu úkolů souvisejících s předmětem KIV/OOP, protože tento předmět je vyučován během zimního semestru a využívá pro kontrolu samostatných prací studentů validační server. Výsledkem analýzy všech počátečních úkolů byla tabulka 3.2.

Tabulka 3.2: Počáteční plán řešení

Pořadí vykonání	Název úkolu
1	Rozmyslet pořadí plnění úkolů
2	Kategorizace vlastních akcí
3	Změnit použití nápovědy u vlastních akcí
4	Refaktor akce Spuštění JUnit
5	Nápověda na vytváření akcí do Wiki
6	Úprava pom.xml souboru
7	Domény otestovatelné bez ukázkových souborů
8	Vytvoření akce Vymaž soubory podle vzoru
9	Vyzkoušení a popsání akce na kontrolu počtu stránek a slov
10	Úprava souborové struktury vlastních akcí
11	Refaktorování akce SpustitPMD a SpustitUMLTestovani
12	Doplnění serveru / util class o standardně používané akce
13	Doplnění Wiki o procesu práce s validátorem
14	Vyřešit umístění pomocných souborů pro jednotlivé ukázky
15	Úprava vlastních akcí na použití checkboxů
16	Předdefinované hodnoty parametrů
17	Parametrizace odkazem u vlastních akcí

Úkoly během práce přibývaly. Nové úkoly jsem se snažil zařadit do současné práce tak, jak bylo nejvhodnější. Konečné pořadí vykonání úkolů je v tabulce 3.3

Tabulka 3.3: Konečný seznam úkolů v pořadí vykonání

Pořadí vykonání	Název úkolu
1	Rozmyslet pořadí plnění úkolů
2	Kategorizace vlastních akcí
3	Změnit použití nápovědy u vlastních akcí
4	Vytvoření nové třídy AbstraktniVlastniAkce
5	Refaktor akce Spuštění JUnit
6	Nápověda na vytváření akcí do Wiki
7	Úprava pom.xml souboru
8	Domény otestovatelné bez ukázkových souborů
9	Vytvoření akce Vymaž soubory podle vzoru
10	Vyzkoušení a popsání akce na kontrolu počtu stránek a slov
11	Úprava souborové struktury vlastních akcí
12	Refaktorování akce SpustitPMD a SpustitUMLTestovani
13	Doplnění serveru / util class o standardně používané akce
14	Doplnění Wiki o procesu práce s validátorem
15	Vyřešit umístění pomocných souborů pro jednotlivé ukázky
16	Úprava vlastních akcí na použití checkboxů
17	Předdefinované hodnoty parametrů
18	Parametrizace odkazem u vlastních akcí
19	Dynamické přidávání dalších kategorií
20	ukazka-kontrolanepovolenychsouboru
21	ukazka-kopirovatslozkydoworkdir
22	ukazka-najdisoubory
23	ukazka-porovnanistrukturyadresaru
24	Přidání nových položek do rozhraní ValidationResult
25	Přejmenovat vlastní akci spoctislova na spoctislovaastrany
26	String index out of range: -86
27	Nejde vlastní akce Velikost souboru
28	Jak nastavit počáteční adresář pro Javadoc?
29	OK hlášení JUnit testu metody
30	Vlastní akce Kopírování souborů
31	Nová vlastní akce generovanijavadoc
32	Nefungující doména a prapodivné chování validátoru
33	Chyba při rozbalování JAR
34	Chybné vyhodnocení výsledku duck-testu

4 Návrh řešení a implementace námětů na vylepšení

V této kapitole navrhu pro každý úkol řešení na základě analýzy a popíšu implementaci řešení.

Sekci příklady použití mají pouze úkoly, které mění stávající funkčnost nebo zavádí novou. U úkolů typu oprava chyby nebo změna vypsané zprávy by tato sekce nedávala smysl.

Dokumentace k jednotlivým řešením je v případě vlastních akcí obsažena v jejich HTML nápovědě a v ostatních případech v úkolech na Redmine.

4.1 Kategorizace vlastních akcí

4.1.1 Návrh řešení

Pro správnou kategorizaci vlastních akcí je potřeba projít současné existující vlastní akce a zamyslet se nad jejich zařazením do konkrétní kategorie. Pokud vlastní akce nezapadá do dané kategorie, akci je nutno přeřadit.

Nejlepší způsob, jak toto provést, je otevřít si seznam vlastních akcí ve webovém rozhraní a zkontrolovat, zda je vlastní akce zařazena do správné kategorie. Špatně zařazené vlastní akce se sepíšou a poté se upraví jejich kategorie v metodě `getKategorii()`.

4.1.2 Implementace řešení

U všech vlastních akcí jsem zkontroloval, jestli patří do odpovídajících kategorií. Pokud byly ve špatné kategorii, nastavil jsem jim správnou. Nové kategorie jsem nepřidával, protože bych musel provádět změnu kódu na dvou místech a věděl jsem, že v dalším úkolu budu implementovat dynamické přidávání kategorií, viz úkol „Dynamické přidávání dalších kategorií“. V tabulce 4.1 je seznam akcí, které byly překategorizovány.

Tabulka 4.1: Přehled překategorizovaných vlastních akcí

Název vlastní akce	Špatná kategorie	Správná kategorie
`RozbalZipJar`	Práce se soubory	Dílčí akce
`SpusteniPmd`	Dílčí akce	Práce s programy
`SpusteniUmlTestovani`	Dílčí akce	Práce s programy
`SpustitDuckTesty`	Dílčí akce	Práce s programy

4.2 Změnit použití nápovědy u vlastních akcí

4.2.1 Návrh řešení

Pokud bychom v metodě `getHelp()` použili mechanismus načtení parametrů z knihovny Apache Commons Configuration, bude možné text nápovědy přepsat do souvislého bloku textu a přesunout jej do souboru.

Apache Commons Configuration pracuje primárně s parametry jako dvojicí klíč–hodnota. Protože text nápovědy může být libovolný dlouhý text s HTML značkami, je vhodným řešením použití XML dokumentu s tím, že text nápovědy bude zasazen do CDATA bloku, aby se nemusely používat HTML entity. Properties soubory nejsou uzpůsobené pro dlouhý souvislý text v hodnotách.

Nejprve zkusím načítání nápovědy ze souboru implementovat a odladit v jedné vlastní akci. Pokud se bude nápověda u zvolené vlastní akce zobrazovat v pořádku, implementuji načítání ze souboru i u ostatních vlastních akcích.

4.2.2 Implementace řešení

Vybral jsem si jednu konkrétní akci – `RozbalZipJar`, abych mohl otestovat použití knihovny Apache Commons Configuration. Do metody `getHelp()` této akce jsem implementoval načítání konfiguračního souboru s nápovědou přes knihovnu Apache Commons Configuration. Nápovědu jsem se rozhodnul uložit souboru do `config.xml` a tento soubor umístit do složky `src/main-/resources`.

Akce `RozbalZipJar` správně načítala konfigurační soubor s nápovědou. Narazilo se však na dvě komplikace. První komplikace byla, že část kódu pro

načítání konfiguračního souboru by se musela opakovat beze změny ve všech ostatních vlastních akcích. Mnohem vhodnější by bylo nadefinovat tuto část společně pro všechny akce. Do společného rozhraní `VlastniAkce` konkrétní implementaci ale přidat nešlo. Aby bylo možné nadefinovat standardní chování společné pro všechny vlastní akce, musel být proveden refaktoring stávající hiearchie vlastních akcí.

Druhou komplikací bylo, že pro text nápovědy knihovna Apache Commons Configuration vlastně vůbec nebyla třeba. Text nápovědy mohl být v prostém textovém souboru a načítán běžně jako řetězec. Avšak s ohledem na možnosti knihovny Apache Commons Configuration a potřebou pro definování standardního společného chování pro všechny vlastní akce vyplynulo na povrch jiné použití pro tuto knihovnu. Konfigurační soubory šlo využít pro definování řetězců, které se používájí v metodách `getId()`, `getKategorii()`, `getNazev()`, `getPopis()` a `getParametry()`.

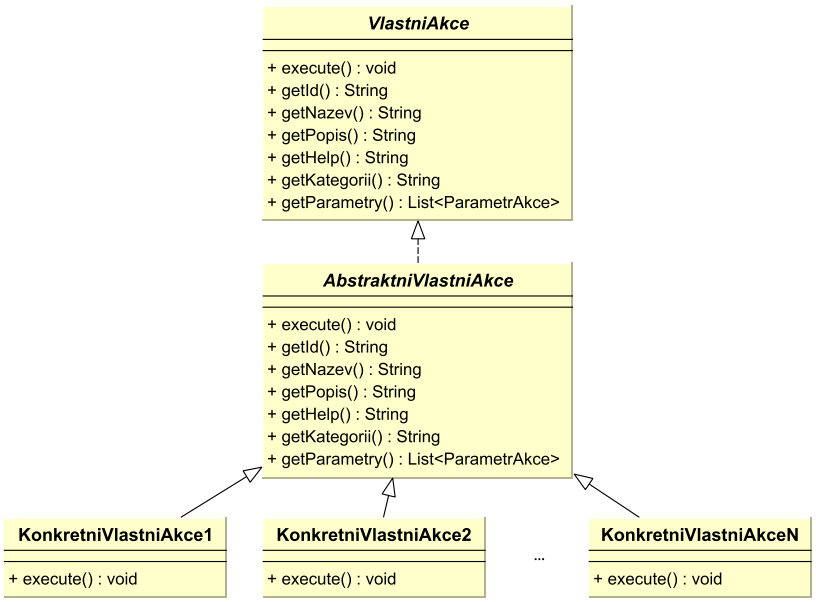
Popsané dvě komplikace vedly k tomu, že tento úkol byl nakonec uzavřen a místo něj vytvořen úkol „Vytvoření nové třídy AbstraktniVlastniAkce“, kde se snažím tyto dvě komplikace vyřešit.

4.3 Vytvoření nové třídy AbstraktniVlastniAkce

4.3.1 Návrh řešení

Prvním cílem je mít třídu `AbstraktniVlastniAkce`, která bude implementovat společné metody pro všechny vlastní akce a bude kompatibilní s používaným rozhraním. Tato třída bude proto dědit od rozhraní `VlastniAkce`. Naopak samotné vlastní akce už nebudou používat rozhraní `VlastniAkce` a místo toho budou dědit od abstraktní třídy `AbstraktniVlastniAkce`. Zdrojový kód vlastních akcí se po tomto refaktoringu výrazně zpřehlední, protože vlastní akce budou muset implementovat pouze metodu `execute()`. Na obrázku 4.1 je znázorněn diagram tříd po refaktoringu a použití `AbstraktniVlastniAkce`.

Druhým cílem je implementovat načítání nápovědy a hodnot ze souboru v metodách `getHelp()`, `getId()`, `getKategorii()`, `getNazev()`, `getPopis()` a `getParametry()`. Načítání obou souborů může být provedeno už v konstruktoru `AbstraktniVlastniAkce` a v metodách `getHelp()`, `getId()`, `getKate-`



Obrázek 4.1: UML diagram tříd pro abstraktní třídu `AbstraktniVlastniAkce`

`gorii()`, `getNazev()`, `getPopis()` a `getParametry()` se bude pouze vracet načtený řetězec či seznam parametrů.

Celá nápověda vlastní akce je psána pomocí HTML. Tudíž stačí nápovědu přepsat do prostého souvislého textu, odstranit Java escape znaky a tento text uložit do souboru. Do konfiguračního souboru lze bez potíží přesunout id, název, popis a kategorii akce.

Zpracování parametrů bude o něco komplikovanější. Metoda `getParametry()` vrací `List<ParametrAkce>`, což je seznam všech parametrů vlastní akce. Každý parametr se do seznamu vkládá jako nový objekt třídy `ParametrAkce` v jehož konstruktoru se musí uvést id, název, popis a typ parametru. V ukázce 4.1 je příklad metody `getParametry()` s dvěma parametry.

```
@Override
public List<ParametrAkce> getParametry() {

    ArrayList<ParametrAkce> list = new ArrayList<ParametrAkce>();
    list.add(new ParametrAkce(PARAM_ZIP, "ZIP/JAR archiv (JS)", "ZIP/JAR archiv zadany JS
        promennou typu java.io.File, defaultne odevzdavany soubor.", ""));
    list.add(new ParametrAkce(PARAM_DEST, "Cilovy adresar (JS)", "Promenna, kam se ulozi adresar s
        extrahovanymi soubory.", ""));

    return list;
}
```

Ukázka 4.1: Metoda `getParametry()` s dvěma parametry

Každý parametr musí mít v konfiguračním souboru jiný klíč. Aby bylo možné při naplňování seznamu parametrů akce načíst texty pro tyto parametry, je nutné dopředu znát, pod jakými klíči je hledat. V konfiguračním souboru tedy musí být před samotnými texty parametrů ještě uvedena deklarace všech parametrů. Až poté mohou být uvedeny hodnoty pro jednotlivé parametry vlastních akcí.

V ukázce 4.2 je příklad celého konfiguračního souboru pro vlastní akci `Rozbal_ZIP`. Čárky musí být zapsány s escape lomítkem, jinak by se čárka interpretovala jako oddělovač v seznamu hodnot, viz parametr `parameters` a parametr `param_zip.description` v ukázce.

```
id = Rozbal_ZIP
name = Rozbalit ZIP / JAR
description = Rozbalení ZIP / JAR archivu
category = Dílčí akce

parameters = param_zip, param_dest

param_zip.id = ZIP archiv
param_zip.name = ZIP / JAR archiv (JS)
param_zip.description = ZIP / JAR archiv zadaný JS proměnnou typu java.io.File\, defaultně
    odevzdávaný soubor.
param_zip.type =

param_dest.id = Cílový adresář
param_dest.name = Cílový adresář (JS)
param_dest.description = Proměnná\, kam se uloží adresáře s extrahovanými soubory.
param_dest.type =
```

Ukázka 4.2: Příklad konfiguračního souboru

4.3.2 Implementance řešení

Vytvořil jsem abstraktní třídu `AbstraktniVlastniAkce` v balíčku `cz.zcu.validationserver.webmodule.actions`, která implementuje rozhraní `VlastniAkce`. V této třídě jsem vytvořil konstruktor, který volá private metody `nactiNapovedu` a `nactiKonfiguraci`. V metodě `nactiNapovedu` načítám nápovědu uloženou v souboru `src/main/resources/NazevVlastniAkce-help.html`. V metodě `nactiKonfiguraci` načítám celý konfigurační soubor uložený v souboru `src/main/resources/NazevVlastniAkce-text.properties`.

Poté jsem implementoval metody `getId()`, `getKategorii()`, `getNazev()`, `getPopis()` a `getParametry()`.

V ukázce 4.3 je samotná implementace načítání nápovědy.

```
private void nactiNapovedu(String helpFilename) {

    InputStream helpResource = getClass().getResourceAsStream("/" + helpFilename);

    try {
        help = IOUtils.toString(helpResource, "UTF−8");
    } catch (IOException e) {
        throw new ValidationError("Chyba pri zpracovani souboru s napovedou vlastni akce " +
            getClass().getSimpleName(), ValidationResult.VR_SERVER_ERROR, e);
    }
}
```

Ukázka 4.3: Metoda pro načítání nápovědy

V metodě `nactiNapovedu()` využívám statickou metodu `IOUtils.toString()` z knihovny Apache Commons IO pro získání obsahu nápovědy jako řetězce. Vrácení nápovědy v metodě `getHelp()` se pak zkrátí na jednu řádku.

V ukázce 4.4 je implementace metody `nactiKonfiguraci()`.

```
private void nactiKonfiguraci(String configFilename) {

    InputStream configResource = getClass().getResourceAsStream("/" + configFilename);

    try {

        config = new PropertiesConfiguration();
        ((PropertiesConfiguration) config).setEncoding("UTF−8");
        ((PropertiesConfiguration) config).load(configResource)

    } catch (ConfigurationException e) {
        throw new ValidationError("Chyba pri zpracovani souboru s nastavenim vlastni akce " +
            getClass().getSimpleName(), ValidationResult.VR_SERVER_ERROR, e);
    }

}
```

Ukázka 4.4: Metoda pro načítání konfiguračního souboru

V metodě `nactiKonfiguraci()` nejprve vytvořím instanci třídy `PropertiesConfiguration()`, kterou přiřadím do instační proměnné `Configuration config`. Poté je nutné explicitně nastavit kódování načítaného properties souboru, protože standardně je nastaveno kódování ISO-8859-1. Standardní kódování by vyžadovalo, aby české znaky byly uvedeny ve formátu \uXXXX, což by bylo uživatelsky nepřívětivé, a proto bylo zvoleno kódování UTF-8,

které je univerzálnější. Posledním příkazem metody `nactiKonfiguraci()` je pak samotné načtení konfiguračního souboru.

Metody `getId()`, `getKategorii()`, `getNazev()`, `getPopis()` pak vrací hodnotu klíče v konfiguračním souboru.

Metoda `getParametry()` nejprve získá seznam hodnot v klíči `parameters` konfiguračního souboru a po té vytváří pro každý parametr novou instanci třídy `ParametrAkce`, viz ukázka 4.5

```
@Override
public List<ParametrAkce> getParametry() {

    List<Object> list = config.getList("parameters");
    ArrayList<ParametrAkce> parametry = new ArrayList<ParametrAkce>();

    for (Object o : list) {
        String paramName = o.toString();
        parametry.add(
                new ParametrAkce(config.getString(paramName + ".id"),
                        config.getString(paramName + ".name"),
                        config.getString(paramName + ".description"),
                        config.getString(paramName + ".type", ""))
        );
    }

    return parametry;
}
```

Ukázka 4.5: Metoda `getParametry()` po implementaci načítání konfiguračního souboru

Po vytvoření třídy `AbstraktniVlastniAkce` jsem ve všech vlastních akcích vyextrahoval nápovědu a konstantní řetězce do textových souborů, změnil reference na vyextrahované řetězce a změnil hlavičku tříd.

Tato změna byla rozsáhlá, výsledkem však byla mnohem přehlednější nápověda a zdrojové soubory vlastních akcí, což má zásadní vliv na jednoduchost při vytváření a udržování vlastních akcí. V tabulce 4.2 je seznam všech vlastních akcí, které jsem pozměnil.

4.3.3 Příklady použití

Pro vytvoření nové vlastní akce je třeba udělat následující kroky:

Tabulka 4.2: Přehled refaktorovaných vlastních akcí v rámci úkolu „Vytvoření nové třídy AbstraktniVlastniAkce“

Název vlastní akce	Název vlastní akce
`KompilaceAdresare`	`PorovnejSoubory`
`KompilaceSouboru`	`PorovnejStrukturuAdresaru`
`KontrolaJavaDocDokumentace`	`RozbalZipJar`
`KontrolaNazvuOdevzdavanehoSouboru`	`SmazatSouboryAdresare`
`KontrolaNepovolenychTypuSouboru`	`SpoctiSlova`
`KontrolaProgramu`	`SpusteniJARProgramu`
`KontrolaVelikostiOdevzdavanehoSouboru`	`SpusteniJavaProgramu`
`KopirovatDoWorkdir`	`SpusteniKonkretnihoJUnitTestu`
`KopirovatSlozkydoWorkdir`	`SpusteniVsechJUnitTestu`
`NajdiAdresare`	`SpustitDuckTesty`
`NajdiSoubory`	`VlozitScreenshot`
`PorovnatPNG`	

1. vytvořit nový Maven projekt s poupravenou pom.xml šablonou,
2. vytvořit třídu, která dědí od `AbstraktniVlastniAkce` a implementovat metodu `execute()`,
3. vytvořit konfigurační soubor a soubor s nápovědou.

4.4 Refaktor akce „Spuštění JUnit“

4.4.1 Návrh řešení

Akce bude provádět pouze samotné spuštění JUnit testů a pouze pro jeden konkrétní .class soubor, zbytek kódu bude odstraněn. Akce bude mít pouze jediný parametr: složku, která obsahuje testy. Domény, které využívají tuto akci, budou muset být přenastaveny tak, aby před krokem s akcí „Spuštění všech JUnit testů“ měly dodatečné tři kroky s funkcionalitou bodů 1 až 3 v analýze úkolu.

4.4.2 Implementace

Z třídy `SpusteniVsechJUnitTestu` jsem odstranil nadbytečný kód a akci refaktoroval tak, aby využívala nový parametr. Akce nyní provádí pouze samotné spuštění JUnit testů. V ukázce 4.6 je kus kódu zajišťující spuštění testů.

```
File testOutput = new java.io.File(info.getWorkDir(), outDirName + File.separator + fileName + ".
    out");
File errOutput = new java.io.File(info.getWorkDir(), outDirName + File.separator + fileName + ".err"
    );

ValidatorJavaInvoker invoker = new ValidatorJavaInvoker(info, result, null, testOutput, errOutput);
invoker.invokeClass(info.getWorkDir(), "cz.zcu.validationserver.junit.JUnit4Invoker", fileName, null);

int exitValue = invoker.getExitValue();

if (exitValue != 0) {
    vypisChybu(result, testOutput, errOutput);
}
else {
    result.addInfo("\nStandardni vystup: <pre wrap>" + FileUtils.getFileContentsAsString(
        testOutput) + "</pre>");
}
```

Ukázka 4.6: Spuštění testů v akci „Spustit všechny JUnit testy“

Nejprve se vytvoří soubory, do kterých se bude generovat standardní výstup a chybový výstup. Akce pak spustí třídu `cz.zcu.validationserver.junit.JUnit4Invoker`, která je na straně serveru a které předá název souboru, který obsahuje testy. Akce pak čeká na návratovou hodnotu. Pokud proběhnou testy v pořádku, vloží se jako informační zpráva do výsledku validace obsah souboru se standardním výstupem. Pokud testy neprojdou, akce vypíše chybu.

4.4.3 Příklady použití

Akce má pouze jediný parametr – název zkompilovaného souboru s testy. V tomto parametru musí být uveden přesný název souboru i s příponou, například `TestProgramu.class`. Class soubor musí v době spuštění kroku s touto akcí již existovat.

V rámci domény lze akci využít například tak, že:

- student odevzdá jediný zdrojový soubor, například `Auto.java`,
- v prvním kroku se z adresáře validace zkopíruje soubor s testy, například `TestAuto.java`,
- v druhém kroku se zkompilují všechny zdrojové soubory v pracovním adresáři,

- ve třetí kroku se použije akce „Spuštění všech JUnit testů“, která spustí testy ve zkompilovaném souboru. `TestAuto.class`.

4.5 Nápověda na vytváření akcí do Wiki

4.5.1 Návrh řešení

Do Redmine wiki napíšu návod, který bude popisovat vytvoření nové akce zcela od začátku, protože došlo k zásadní změně ve struktuře akcí.

4.5.2 Implementace

Když jsem se pouštěl do psaní návodu, všimnul jsem si, že ve wiki už je návod pojmenovaný „Vytvoření nového modulu a jeho commit na server“. V tomto návodu už byl popsaný celý postup. Protože se ale struktura vlastních akcí výrazným způsobem změnila, bylo třeba provést úpravy. Postupoval jsem podle návodu a aktualizoval jsem části, které se změnily.

4.6 Domény otestovatelné bez ukázkových souborů

4.6.1 Návrh řešení

Aby bylo možné otestování domény bez nahrání souboru, bude třeba změnit dvě věci – javaskriptovou část, která neumožňuje stisknout tlačítko bez vybrání lokálního souboru, a serverovou část, která se stará o samotné zpracování nahraného souboru. Dále bude třeba přidat další parametr do nastavené domény, který bude specifikovat, zda má být doména otestovatelná bez nahrání souboru či nikoliv.

4.6.2 Implementace

V souboru `domenaUprav.jsp` jsem upravil kontrolu prázdnosti nahrávacího pole. Ukázka 4.7 obsahuje kus kódu, který prováděl zmiňovanou kontrolu. Pouhé upozornění, že pole nemůže být prázdné, jsem nahradil potvrzením, zda uživatel chce opravdu testovat doménu bez souboru.

```
var x=document.forms["myForm"]["soubor"].value
if (x==null || x=="")
{
    // stary kod
    // alert("Musíte vybrat soubor k testování!");
    // return false;

    var r=confirm("Opravdu chcete testovat bez souboru?");
    return r;
}
```

Ukázka 4.7: Kontrola prázdnosti nahrávaného souboru při testování domény

Protože proces validace vždy pracuje se vstupním souborem, musel být na straně serveru vytvořen prázdný soubor. Tento soubor slouží pouze jako náhrada za nahraný soubor a má nulovou velikost. Tím se nerozbije funkčnost akcí, které volají `info.getInputFile()`, pokud by se použily v doméně, která nevyžaduje vstupní soubor.

V další fázi jsem parametrizoval, zda má být doména otestovatelná bez vstupního souboru či nikoliv. Do souboru `domain.xml`, který obsahuje nastavení domény, jsem pro všechny vlastní akce přidal nový klíč `domain.disable_test_file` s hodnotou 0. Tento klíč jsem přidal také do šablony pro nové vlastní akce, abych zajistil, že nový parametr bude i u nově vytvořených domén. Podle parametru `domain.disable_test_file` se určuje, zda se na webové stránce má zobrazit pole pro výběr souboru či nikoliv. V ukázce 4.8 je předvedeno využití nového parametru.

```
<form
    name="myForm"
    ENCTYPE="multipart/form-data"
    action="<%=request.getContextPath()%>/auth/domena/testuj/"
    method="post"
    <% if (!isTestFileDisabled) { %>
    onsubmit="return validateForm();"
    <% } %>
>

<% if (!isTestFileDisabled) { %>
```

```
<p>Vyberte soubor, který chcete otestovat</p>
<% } %>
<p>
    <input type="hidden" name="akce" value="testujDomenu" />
    <input type="hidden" name="domena" value="<%= domain.getName()%>" />
    <input type="file" name="soubor" size="30" <% if (isTestFileDisabled) { %> class="skryj" <% }
        %> />
<p>
<input type="submit" name="Odeslat2" value="Testovat" />
</p>

</form>
```

Ukázka 4.8: Použití parametru `domain.disable_test_file`

4.6.3 Příklady použití

Na stránce detailní nastavení domény je nyní nový klíč `domain.disable_test_file`. Klíč může mít hodnotu 0 a 1. Při hodnotě 0 musí uživatel vybrat soubor pro nahrání, pokud chce testovat doménu. Při hodnotě 1 lze doménu testovat pouhým stisknutím tlačítka „Testovat“. Hodnota 1 znamená, že při zvolení hodnoty 1 bude vstupním souborem prázdný soubor.

4.7 Vytvoření akce Vymaž soubory podle vzoru

4.7.1 Návrh řešení

Nová akce bude mít 2 parametry: vzorový adresář a cílový adresář. Ze vzorového adresáře se bude číst struktura, zatímco v cílovém adresáři se bude mazat. V cílovém adresáři se budou mazat pouze soubory a podadresáře, které se zároveň vyskytují i ve vzorovém adresáři. Soubory navíc v cílovém adresáři se budou ignorovat.

4.7.2 Implementace

Vytvořil jsem dvě třídy: `VymazSouboryPodleVzoru` a `CustomFileVisitor`. Třída `VymazSouboryPodleVzoru` dědí od třídy `AbstraktniVlastniAkce` a

tudíž vykonává samotnou validaci. Uvnitř metody `execute()` se zpracovávají parametry akce a prochází adresářová struktura vzorového adresáře. V ukázce 4.9 je uveden způsob procházení adresáře. Pro procházení se používá nová metoda `Files.walkFileTree()`, která je dostupná až v Java 7.

```
CustomFileVisitor cfv = new CustomFileVisitor(model.getAbsolutePath(), target.getAbsolutePath(),
     result);
Files.walkFileTree(model.toPath(), cfv);
```

Ukázka 4.9: Procházení vzorového adresáře v akci `VymazSouboryPodleVzoru`

Metoda `visitFile()` v třídě `CustomFileValidator` definuje činnost, která se má provést pro každý procházený soubor ve zdrojovém adresáři. V této metodě provádím to, že dělám kontrolu, zda právě zpracovávaný soubor ve vzorovém adresáři existuje také v cílovém adresáři. Pokud ano, smažu jej a vypíšu info zprávu.

4.7.3 Příklady použití

Student odevzdává archiv, ve kterém jsou tři soubory: `A.txt`, `B.txt` a `foo/C.txt`. V predešlém kroku validace jsme provedli rozbalení odevzdávaného archivu do adresáře `cil` v pracovním adresáři. Nyní chceme smazat soubor `foo/C.txt`.

Použijeme akci „Vymaž soubory podle vzoru“, kde jako cílový adresář nastavíme `cil` a jako vzorový adresář `vzor`. Složku `vzor` jsme překopírovali do pracovního adresáře například z adresáře domény a uvnitř adresáře je pouze prázdný soubor `foo/C.txt`. Po vykonání akce bude stav takový, že adresář `cil/foo/` bude prázdný.

4.8 Vyzkoušení akce na kontrolu počtu stránek a slov

4.8.1 Návrh řešení

Vytvořím si novou ukázkovou doménu, ve které v prvním kroku použiji vlastní akci „Spočti slova“. V dalších krocích vypíšu obsah výstupních parametrů akce „Spočti slova“. Do ukázkové domény následně zkusím odevzdat doc a pdf dokument a porovnám reálný počet stran a slov s obsahem vypsaných parametrů akce. Nápovědu k akci dopíši po otestování funkčnosti.

4.8.2 Implementace řešení

Pro otestování této akce jsem si vytvořil jednoduchý pětistránkový dokument v programu Microsoft Word 2007. Tento dokument jsem následně vyexportoval do pdf. Ve webovém rozhraní validátoru jsem vytvořil ukázkovou doménu „ukazka–spoctislova“ se třemi kroky.

V prvním kroku jsem použil samotnou vlastní akci „Spočti slova“ a nastavil ji parametr „Počet slov (JS)“ na hodnotu `pocetSlov` a parametr „Počet stran (JS)“ na hodnotu `pocetStran`. V druhém kroku vypisuji obsah proměnné `pocetSlov`. V třetím kroku vypisuji obsah proměnné `pocetStran`. Ukázkovou doménu jsem následně vyzkoušel s doc i pdf verzí vytvořeného dokumentu.

Nápovědu jsem sepsal do souboru `SpoctiSlova-help.properties`.

```
<h4>Kontextová nápověda: vlastní akce "Spočti slova"</h4>

<p>Akce slouží ke zjištění počtu slov a stran v dokumentech doc a pdf.</p>

<p>Jako parametr "Soubor (JS)" vyplňte název doc / pdf souboru. Pokud necháte pole prázdné, bude zpracován odevzdávaný
       soubor.
Soubor lze odkazovat pomocí JS proměnné.

<p>Parametr "Počet slov (JS)" slouží pro vrácení počtu slov v podobě javaskriptové proměnné.

<p>Parametr "Počet stran (JS)" slouží pro vrácení počtu stran v podobě javaskriptové proměnné.

<p><b>Příklad 1</b></p>
<p><b>hodnota Soubor (JS): </b>
<p><b>hodnota Počet slov (JS): </b>pocetSlov</p>
<p><b>hodnota Počet stran (JS): </b>pocetStran</p>
<p>U odevzdávaného souboru (parametr "Soubor (JS)" není zadán) spočte počet slov a počet stran. V proměnné <b>
      pocetSlov</b> bude uložen
počet slov (číslo) a v proměnné <b>pocetStran</b> bude uložen počet stran (číslo).</p>
```

Ukázka 4.10: Nápověda pro akci `SpoctiSlova`

4.8.3 Příklady použití

Viz nápověda v ukázce 4.10.

4.9 Úprava souborové struktury vlastních akcí

4.9.1 Návrh řešení

U všech vlastních akcí vytvořím nový adresář, do kterého umístím text z ukázkových domén. Každý krok ukázkové domény bude mít v textovém souboru vlastní odstavec.

4.9.2 Implementace

V každé vlastní akci jsem vytvořil adresář `example-domain-resources` a v tomto adresáři soubor `example-test.txt`. Do souboru `example-test.txt` jsem poté přesunul popisy kroků z odpovídající ukázkové domény.

4.10 Refaktorování akce SpustitPMD a SpustitUMLTestovani

4.10.1 Návrh řešení

Obě akce převedu do Maven formátu a zároveň akce refaktoruji tak, aby využívaly třídu `AbstraktniVlastniAkce`.

4.10.2 Implementace

U obou akcí jsem upravil adresářovou strukturu tak, aby odpovídala Maven projektu. Do nejvyššího adresáře obou akcí jsem zkopíroval šablonový

pom.xml, který jsem upravil, aby odpovídal spravnému projektu. Třídy jsem refaktoroval, aby dědily od třídy `AbstraktniVlastniAkce` a následně jsem vytvořil konfigurační soubor a soubor s nápovědou.

4.11 Doplnění serveru / util class o standardně užívané akce

4.11.1 Návrh řešení

Na straně serveru vytvořím novou třídu, která bude mít statickou metodu pro získání objektu typu `java.io.File` z řetězce a statickou metodu pro převod regulárního výrazu v DOS tvaru do Java tvaru.

4.11.2 Implementace

V balíčku `cz.zcu.validationserver.utils` jsem vytvořil třídu `VlastniAkceUtils`. V této tříde jsem vytvořil tři metody: `convertToFile()`, `convertToFileFromJavaScript` a `convertFromDOSRegex`.

Metoda `convertTofile()` funguje tak, že se snaží ze vstupního řetězce vytvořit objekt typu `java.io.File`. Pokud je vstupní řetězec prázdný, vratí metoda vstupní soubor validace neboli studentem odevzdaný soubor. Pokud začína vstupní řetězec znakem $, pokusí se metoda získat soubor či adresář z javaskriptového kontextu. V ostatních případech metoda předpokládá, že uživatel zadává název souboru či adresáře relativní cestou vůči pracovnímu adresáři.

Metoda `convertToFileFromJavaScript()` je využívána metodou `convertToFile()` a provádí extrahování objektu `java.io.File` z javaskriptového kontextu.

Metoda `convertFromDOSRegex()` nahrazuje ve vstupním řetězci znaky „`*`“, „`?`“ a „`.`“ za znaky „`\\S*`“, „`\\`“ a „`\\.`“.

Celý zdrojový soubor s touto třídou je v příloze B.

4.11.3 Příklady použití

Pokud má vlastní akce parametr, kde se zadává konkrétní soubor či adresář v pracovním adresáři, je vhodné použít metodu `VlastniAkceUtils.convertToFile()`. Zajistí se tím jednotné fungování parametrů, ve kterých lze použít javaskriptové proměnné. Metoda se použije tak, že se jí předá vstupní řetězec obsahující hledaný soubor/adresář, dále řetězec definující název skriptu při extrahování z javaskriptové proměnné, objekt typu `ValidationInfo`, objekt typu `Context` a objekt typu `Scriptable`. Poslední tři uvedené objekty se předají beze změny z parametrů metody `execute()` vlastní akce.

Metoda `VlastniAkceUtils.convertFromDOSRegex()` se využije v případě, kdy uživatel domény může zadat parametr pomocí regulárního výrazu. Příkladem může být akce „Najdi soubory“, která využívá regularní výraz pro určení hledaných souborů adresářů. Metoda se použije tak, že se jí předá parametr akce obsahující regulární výraz v DOS tvaru. Výsledek metody lze pak použít ve standardních Java metodách, které umí pracovat s regulárními výrazy například `Pattern.matches()`.

4.12 Doplnění Wiki o procesu práce s validátorem

4.12.1 Návrh řešení

Do Redmine wiki dopíšu návody, které chybí.

4.12.2 Implementace

Při psaní návodů jsem vycházel ze starých materialů, které mi byly poskytnuty vedoucím práce. Staré návody se ukázaly být nepostradatelné, protože mnoho věcí nebylo přirozeně odvoditelných. Výsledkem byly návody: „Příprava prostředí pro vývoj validačního serveru“, ve kterém popisuji, jak nastavit lokální verzi validačního serveru, a návod „Build serverové části validátoru a jeho nasazení na validační server“.

4.13 Vyřešit umístění pomocných souborů pro jednotlivé ukázky

4.13.1 Návrh řešení

Ukázkové soubory by měly být uloženy na obecnější adrese spojené s validačním serverem a modifikace souborů by měla být umožněna více uživatelům. Požadované chování je takové, aby do nového umístění mohl uživatel nahrát soubor a tento soubor byl ihned dostupný veřejnosti z namapované webové adresy. Jako webovou adresu jsem navrhnul `validator-test.zcu.cz/examples/`. Na tuto webovou adresu byl namapován adresář `validt-common/www/examples`.

4.13.2 Implementace

Samotnou implementaci řešení měl na starosti Ing. Lukáš Valenta, který je správcem CIV a má tak potřebná práva pro namapování adresáře na webovou adresu.

4.13.3 Příklady použití

Adresář `validt-common/www/examples` funguje nyní jako uložiště pro soubory, které chceme poskytnout uživatelům ke stažení. Stačí požadovaný soubor nakopírovat do tohoto adresáře a následně přistoupit na adresu `validator-test.zcu.cz/examples/`, odkud lze zkopírovat odkaz.

4.14 Úprava vlastních akcí na použití checkboxů

4.14.1 Návrh řešení

Všechny vlastní akce, které mají parametr typu ano/ne, přepracuji tak, aby využívaly checkboxy a odpovídajícím způsobem upravím nápovědu k akcím.

4.14.2 Implementace

Pro zobrazení parametru jako checkboxu je třeba pouze uvést, že parametr je typu checkbox v souboru `NazevVlastniAkce-text.parameters`, kde jsou definovány použité parametry. V ukázce 4.11 je příklad checkboxového parametru.

```
param_name.type=checkbox
```

Ukázka 4.11: Definování parametru jako checkboxu

Seznam vlastních akcí, kterých se převedení ano/ne parametrů na checkboxy týkala, je uveden v tabulce 4.3

Tabulka 4.3: Seznam vlastních akcí, ve kterých se měnily ano/ne parametry na checkboxy

Název vlastní akce
`KontrolaJavaDocDokumentace`
`KontrolaNepovolenychTypuSouboru`
`KontrolaProgramu`
`NajdiAdresare`
`NajdiSoubory`
`PorovnejStrukturuAdresaru`

4.15 Předdefinové hodnoty parametrů

4.15.1 Návrh řešení

Jako první je třeba stanovit, kam se budou ukládat předdefinované hodnoty. Předdefinové hodnoty jsou vlastnostní každého parametru, tudíž je přirozené, že se budou psát do konfiguračního souboru k definici parametru.

Pole pro zadávání hodnoty parametrů na webové stránce se generují jako klasické HTML `input` prvky. Input značka ma atribut `value`, do kterého lze při generování stránky vypsat předvyplněnou hodnotu. Při generování

stránky tedy bude nutné přistoupit k předdefinované hodnotě daného parametru a vypsat ji do atributu `value`.

4.15.2 Implementace

V souboru `krokUprav.jsp` se generuje seznam vlastních akcí, ze kterých může uživatel následně vybírat vlastní akce, které se mají použít v daném kroku. Pokud však uživatel edituje krok, se kterým už je nějaká vlastní akce spjatá, zobrazí se mu v parametrech hodnoty, které má u ní nastavené. Pro vyplnění hodnoty parametrů jsem tedy při generování seznamu vlastních akcí přidal logiku, aby se do atributu `value` použila předdefinovaná hodnota, pokud se nejedná o vlastní akci, která je momentálně nastavená na daném kroku.

Při generování seznamu vlastních akcí se pracuje s objekty typu `ParametrAkce`. Seznam těchto objektů vrací každá vlastní akce pomocí metody `getParametry()` v třídě `AbstraktniVlasniAkce`. Třídě `ParametrAkce` jsem proto přidal novou proměnnou `vychoziHodnota` a tuto proměnnou nastavuji při načítání parametrů z konfiguračního souboru v metodě `getParametry()`. Jako klíč pro výchozí hodnotu jsem použil `param_nazev-parametru.default`.

Protože se jedná o nový klíč, který se doposud nevyskytoval u žádné akce, a protože na něj lze zapomenout při vytváření konfiguračního souboru, musí načítání počítat s tím, že tento klíč v konfiguračním souboru nebude. To je zajištěno přetíženou metodou `getString()` objektu typu `Configuration`, která má jako druhý parametr hodnotu, která se má použít v případě nenalezení klíče v konfiguračním souboru. Tuto hodnotu jsem nastavil jako prázdný řetězec.

Do konfiguračního souboru všech vlastních akcí jsem přidal tento nový klíč a u parametrů, kde to mělo smysl, jsem definoval i jeho hodnotu. V tabulce 4.4 je uveden seznam vlastních akcí, které jsem upravil.

4.15.3 Příklady použití

V konfiguračním souboru vlastní akce lze v u každého parametru specifikovat hodnotu klíče `param_name.default`. Pokud bude hodnota klíče prázdná, nebude mít parametr žádnou předdefinovanou hodnotu.

Tabulka 4.4: Seznam vlastních akcí, které byly upraveny v rámci úkolu „Předdefinované hodnoty parametrů“

Název vlastní akce	Název vlastní akce
KompilaceAdresare	PorovnejSoubory
KompilaceSouboru	PorovnejStrukturuAdresaru
KontrolaJavaDocDokumentace	RozbalZipJar
KontrolaNazvuOdevzdavanehoSouboru	SmazatSouboryAdresare
KontrolaNepovolenychTypuSouboru	SpoctiSlova
KontrolaProgramu	SpusteniJARProgramu
KontrolaVelikostiOdevzdavanehoSouboru	SpusteniJavaProgramu
KopirovatDoWorkdir	SpusteniKonkretnihoJUnitTestu
KopirovatSlozkydoWorkdir	SpusteniVsechJUnitTestu
NajdiAdresare	SpustitDuckTesty
NajdiSoubory	VlozitScreenshot
PorovnatPNG	

Pokud bude hodnota klíče vyplněna, použije se tato hodnota při generování seznamu vlastních akcí a jejich parametrů. Pro nastavení předdefinované hodnoty checkboxů se používá hodnota `checked`.

4.16 Parametrizace odkazem u vlastních akcí

4.16.1 Návrh řešení

Ve třídě `VlastniAkceUtils` je k dispozici statická metoda `convertToFile()`, která provádí požadovanou funkčnost, tj. převod řetězce, který obsahuje znak dolaru, na objekt typu `java.io.File`. Bude tedy třeba pouze vybrat parametry, u kterých má smysl použít tento způsob odkazování, a použít metodu `convertToFile()`.

4.16.2 Implementace

U vlastních akcí, které jsou uvedené v tabulce 4.5, jsem použil metodu `convertToFile()` pro možnost odkazování se na javaskriptovou proměnnou obsahující `File` pomocí dolaru. Jednalo se o parametry, kde uživatel specifikuje, jaký soubor či adresář chce použít.

U uvedených akcí jsem také upravil nápovědu, aby odrážela možnost odkazování se na javaskritpové proměnné.

Tabulka 4.5: Seznam vlastních akcí, které byly upraveny v rámci úkolu „Parametrizace odkazem u vlastních akcí“

Název vlastní akce	Název vlastní akce
GenerovaniJavadoc	SmazatSouboryAdresare
KompilaceAdresar	SpoctiSlova
KompilaceSouboru	SpusteniJARProgramu
KontrolaJavaDocDokumentace	SpusteniJavaProgramu
KontrolaNepovolenychTypuSouboru	SpusteniKonkretnihoJUnitTestu
KontrolaProgramu	SpusteniPMD
KopirovatDoWorkdir	SpusteniUMLTestovani
NajdiAdresare	SpusteniVsechJUnitTestu
NajdiSoubory	SpustitDuckTesty
PorovnatPNG	VymazSouboryPodleVzoru
PorovnejStrukturuAdresaru	
RozbalZipJar	

4.17 Dynamické přidávání dalších kategorií

4.17.1 Návrh řešení

Ideální by bylo, kdyby si autor akce mohl sám zvolit jakoukoliv kategorii a nebýt omezen předdefinovanými kategoriemi. V konfiguračním souboru autor jednoduše uvede kategorii a ta se pak použije.

4.17.2 Implementace

Mechanismus načítání kategorie ve třídě `AbtraktniVlastniAkce` je vyhovující, pouze jsem určil výchozí kategorii pro případ, že vlastní akci bude chybět klíč pro kategorii. Jako výchozí kategorii jsem definoval `Bez kategorie`.

V souboru `krokUprav.jsp` jsem pak upravil generování seznamu akcí v prvku `select`. Místo pevného pole jsem použil dynamické struktury. Protože kategorie mohou být jakékoliv a není dopředu známé, do jaké kategorie dané akce patří, musím nejprve projít všechny akce a vytvořit si seznam kategorií a akcí do nich spadajících. Poté lze generovat prvek `select`. V ukázce 4.12 je uveden kus kódu zajišťující generování selectboxu.

```
Map<String, List<VlastniAkce>> categories = new TreeMap<String, List<VlastniAkce>>();
List<VlastniAkce> categoryActions = null;
String categoryName = null;

for (VlastniAkce a : akce) {
    categoryName = a.getKategorii();
```

```
    categoryActions = categories.get( categoryName );

    if ( categoryActions == null) {
        categoryActions = new ArrayList<VlastniAkce>();
        categories.put(a. getKategorii (), categoryActions );
    }

    categoryActions .add(a);
}

for (Map.Entry<String, List<VlastniAkce>> entry : categories.entrySet()) { %>
    <optgroup label="<%=entry.getKey()%>">
    <%
    for ( VlastniAkce a : entry.getValue()) { %>
        <option value="<%= a.getId() %>" title="<%= a.getPopis() %>"<%= (selected.equals(a.getId()) ? " selected
            =\"selected\"" : "") %>><%= a.getNazev() %></option>
    <%
    }
    %> </optgroup>
}
```

Ukázka 4.12: Dynamické generování kategorií ve webovém rozhraní

4.17.3 Příklady použití

Vlastní akce bude mít jakoukoliv kategorii, která bude uvedena v klíči `category` v konfiguračním souboru. Autor akce by se měl omezit pouze na alfanumerické znaky. Pokud bude v názvu kategorie překlep, bude vlastní akce zařazena do nové kategorie.

4.18 ukazka-kontrolanepovolenychsouboru

4.18.1 Návrh řešení

Do ukázkového archivu přidám soubor `nazdar.txt`.

4.18.2 Implementace

Do archivu `ukazka-kontrolanepovolenychsouboru` jsem přidal prázdný soubor `nazdar.txt`. Tento archiv jsem pak nahrál do adresáře pro ukázkové soubory na validačním serveru.

4.19 ukazka-kopirovatslozkydoworkdir

4.19.1 Návrh řešení

Poupravím řetězec, který se vypisuje při úspěšném zkopírování v akci „Kopírovat složky do workdir“.

4.19.2 Implementace

Ve třídě `KopirovatSlozkyDoWorkdir` jsem poupravil všechny řetězce, které zmiňovaly složky na adresáře.

4.20 ukazka-porovnanistrukturyadresaru

4.20.1 Návrh řešení

Do adresáře `porovnavanyAdresar` přidám navíc adresář.

4.20.2 Implementace

Do podadresáře `porovnavanyAdresar` v adresáři domény `ukazka-porovnani-strukturyadresaru` jsem přidal prázdný adresář `c`. Poté jsem ověřil, že prvek navíc v porovnávaném adresáři při povolení parametru „Jsou extra soubory chyba?“ vypíše chybu.

4.21 Přidání nových položek do rozhraní ValidationResult

4.21.1 Návrh řešení

Do rozhraní `ValidationResult` přidám nový stav `VR_TEST_NOT_PASSED`.

4.21.2 Implementace

Do rozhraní `ValidationResult` jsem přidal novou konstantu `VR_TESTS_NOT_PASSED` s hodnotou 1001.

4.21.3 Příklady použití

Metoda `addError()` třídy `FullValidationResult` má přetíženou variantu, kde jako druhý parametr lze specifikovat konečný stav. Pokud jsou v rámci vlastní akce spouštěny testy, lze v případě neúspěchu vložit chybu tímto způsobem: `result.addError("Testy neprosly", ValidationInfo.VR_TESTS_NOT_PASSED)`.

4.22 Přejmenovat vlastní akci spoctislova na spoctislovaastrany

4.22.1 Návrh řešení

Vlastní akci `spoctislova` přejmenuji na `spoctislovaastrany` a odpovídajícím způsobem upravím pomocné soubory a nápovědu.

4.22.2 Implementace

Název třídy `SpoctiSlova` jsem změnil na `SpoctiSlovaAStrany`. Dále jsem upravil nápovědu a v konfiguračním souboru jsem změnil id akce na `Spocti_slova_a_strany` a název akce na `Spočti slova / strany`.

4.23 String index out of range: -86

4.23.1 Návrh řešení

Upravím metodu `getParameter()` tak, aby vyhledávala pouze šest pomlček.

4.23.2 Implementace

Hledání oddělovače v metodě `getParameter()` jsem upravil tak, aby metoda `indexOf()` vyhledávala pouze šest pomlček. Metodu `indexOf()` není třeba měnit, protože nalezne jak šest pomlček tak i standardních deset.

4.24 Ok hlášení JUnit testu metody

4.24.1 Návrh řešení

Samotné testování a generování výstupního souboru se standardním výstupem zajišťuje třída `SingleJUnit4Invoker` pro spuštění konkrétního JUnit testu a třída `JUnit4Invoker` pro spuštění všech testů. Obě tyto třídy upravím tak, aby při splnění všech testů vypsaly do výstupního souboru požadované hlášky.

4.24.2 Implementace

Do `main()` metody třídy `SingleJUnit4Invoker` jsem přidal vypsání OK hlášení: `System.out.println(methodName + "OK")`. Do `main()` metody třídy

`JUnit4Invoker` jsem přídal vypsání OK hlášení: `System.out.println("OK "+ passedCount + "tests")`.

4.25 Vlastní akce Kopírování souborů

4.25.1 Návrh řešení

Do akce přidám dva nové parametry - parametr pro určení podadresáře a parametr pro regulární výraz pro specifikování souborů ke zkopírování. Parametr pro seznam souborů ke zkopírování a parametr pro regulární výraz si oponují, přičemž jeden z nich musí být vyplněn. Regulární výrazy jsou silnější prostředek než seznam souborů, takže bude mít vyšší prioritu tento parametr.

4.25.2 Implementace

Do vlastní akce `KopirovatDoWorkdir` jsem přidal parametry `param_podadresar` a `param_regex`. V případě použití regulárního výrazu využívám metodu `listFiles()` třídy `File`. Této metodě lze předat objekt typu `FileFilter`, který vyfiltruje soubory, které odpovídají konkrétním podmínkám. Jako vyhovující podmínku jsem zvolil, aby název souboru vyhovoval zadanému regulárnímu výrazu.

4.25.3 Příklady použití

Parametr „Z podadresáře“ slouží pro uvedení z jakého podadresáře validační domény se mají soubory kopírovat. Při uvedení hodnoty `test` se budou soubory kopírovat z podadresáře `test`.

Do parametru „Nebo regulární výraz“ se zadává regulární výraz, kterému mají odpovídat názvy souborů pro zkopírování. Hodnota `*.jar` v tomto parametru by akce vybrala všechny soubory s koncovkou jar.

4.26 Nová vlastní akce generovanijavadoc

4.26.1 Návrh řešení

Akce funguje tak, že spouští přímo příkaz javadoc a tomu přepošle zadaný parametr z vlastní akce. Pro správné fungování je ale třeba, aby se rozbalil znak hvězdičky na všechny soubory. Toho lze dosáhnout dvěma způsoby. Buď při výskytu rozbalíme hvězdičku manuálně, což by bylo pracné a navíc by bylo třeba analyzovat zadávaný řetězec, nebo využijeme linuxovský shell.

Využití linuxovského shellu je vhodnější řešení, protože zadávaný parametr vlastní akce pak bude fungovat jako kdybychom ho zadali přímo v terminálu a navíc nebude třeba provádět nesouvisející kód s generováním javadocu.

4.26.2 Implementace

Místo původního spuštění javadocu metodou `exec(String)` jsem použil přetíženou metodu `exec(String[])`. Pole pro přetíženou variantu musí obsahovat alespoň jeden prvek, což je název spouštěného programu. Další prvky už jsou individuální parametry pro spouštěný program. V ukázce 4.13 je kus kódu provádějící spuštění.

```
// Puvodni verze
Process s = java.lang.Runtime.getRuntime().exec("javadoc" + fullNamesCommand, null, info.getWorkDir()
    );

// Nova verze
Process s = java.lang.Runtime.getRuntime().exec(new String[] {"/bin/sh", "-c", "javadoc" +
    fullNamesCommand});
```

Ukázka 4.13: Spouštění `javadoc` příkazu programátorsky

Po výše uvedené úpravě už generování javadoc dokumentace fungovalo, jenže jsem poté narazil na jiný problém. V testovací doméně pro tuto akci hlásil následující krok, který pracoval s vygenerovanou dokumentací, že adresář s dokumentací neexistuje. Přitom když jsem ověřoval, zda se dokumentace vygenerovala, tak adresář s dokumentací byl v pracovním adresáři.

Následně jsem zjistil, že další krok v testovací doméně se provedl ještě dřív, než generování dokumentace skončilo. Při spuštění shellu kód vlastní akce pokračuje dál a na nic nečeká. Pro správné fungování akce jsem tedy přidal akci další parametr, který určuje, jakou dobu má akce počkat, než skončí vykonávání.

4.26.3 Příklady použití

Vlastní akce „Generovaní Javadoc“ má nyní dva parametry. Prvním parametrem je celý řetězec, který se má předat javadoc příkazu. Řetězec má tvar jako kdybychom ho zadávali na lokálním počítači, například `-d /doc /*.java` (vygeneruje javadoc dokumentaci pro všechny java soubory do složky `doc` v momentálním adresáři, kde momentální adresář je pracovní adresář validace).

Druhým parametrem je doba čekání na dokončení generování v milisekundách. Výchozí hodnota je 2000ms. Často je však třeba s hodnotami experimentovat, protože generování ovlivňuje například momentální zátěž serveru. Hodnota by měla být dostatečně velká, aby validace neskončila chybou z důvodu nenalezení vygenerované dokumentace, ale neměla by být příliš velká, protože pak bude validace trvat delší dobu.

4.27 Chyba při rozbalování JAR

4.27.1 Návrh řešení

Protože je cílem odchytávat vyjímku až ve vlastní akci, bude nutno nechat vyjímku probublat až do vlastní akce.

4.27.2 Implementace

Metodu `ZipExtractor.extractZipArchive()` jsem upravil, aby nezachytávala vyjímku, ale aby ji přeposlala dále. Tato metoda byla použita i jinde v kódu validační serveru než pouze ve vlastní akci „Rozbal ZIP / JAR“, takže jsem tuto vyjímku musel odchytit na příslušných místech. Ve vlastní akci

„Rozbal ZIP / JAR“ jsem vyjímku odchytil a do výsledku validace přidal chybu s popisem, že rozbalovaný archiv obsahuje diakritiku.

4.28 Chybné vyhodnocení výsledku duck-testu

4.28.1 Návrh řešení

Místo kontroly počtu řádek se bude kontrolovat výsledek duck testů jiným způsobem. Na třetí řádek výstupního souboru se generuje výsledek duck testů. Pokud projdou všechny testy, vyskytuje se na této řádce `OK`. Pokud nějaký test neprojde, vypíše se `E` jako error. Výsledku duck testů lze tedy kontrolovat na výskyt písmena E.

4.28.2 Implementace

Ve třídě `SpustitDuckTesty` jsem upravil testovací podmínku na výskyt písmena E na třetí řádce výstupního souboru z duck testů.

5 Otestování implementovaných modulů

Žádný počítačový software není bez chyb. Pro chyby v oblasti informatiky se používá termín „bug“, což přeloženo z angličtiny znamená brouk. Tento termín vzniknul tak, že v počátcích informatiky zalezl brouk do útrob velkého sálového počítače a způsobil tak nefunkčnost celého systému [6].

Softwarový bug může nastat, když je splněna alespoň jedna z následujících podmínek [6]:

- software nedělá to, co je v jeho specifikaci,
- software dělá něco, co specifikace říka, že by neměl dělat,
- software dělá něco, co specifikace nezmiňuje,
- software neděla nic, co by specifikace nezmiňovala, ale měl by to dělat,
- software je těžko pochopitelný, špatně použitelný nebo pomalý.

Vlastní akce lze testovat dvěma způsoby. Lze je jednak testovat na implementační úrovni pomocí jednotkových testů a jednak na uživatelské úrovni pomocí funkčních testů. Jednotkové testy se při vývoji vlastních akcí nepoužívají, protože metoda `execute()` nemá žádnou návratovou hodnotu a mění stavy vstupních parametrů. Napodobení stavu vstupních parametrů by dalo větší práci než samotný test. Implementované moduly jsem testoval na uživatelské úrovni tak, že jsem vytvořil ukázkovou doménu (testovací sadu) a spustil validaci. Jednalo se tedy o manuální testování. Jako správnou funkčnost jsem vyhodnotil situaci, kdy byly splněny následující podmínky: a) validace proběhla všemi kroky validace, b) validace skončila bez chyby nebo s chybou podle toho, jestli se testoval bezchybný průběh nebo chybový stav, c) krok s testovanou akcí vygeneroval výstup jaký měl, pokud akce nějaký výstup měla. Díky použitému postupu testování pomocí ukázkových domén jsem navíc značně ulehčil práci uživateli validátoru. Uživatel se dokáže lépe zorientovat v případech použití vlastních akcí a jejich funkčnosti.

Všechny vstupní soubory pro ukázkové domény (pokud jej mají) jsou dostupné na adrese `https://validator-test.zcu.cz/examples/`.

5.1 generovanijavadoc

Vlastní akce slouží pro vygenerování Javadoc dokumentace ze zdrojových souborů.

Pro modul byla vytvořena ukázková doména „ukazka-generovanijavadoc“. Ukázková doména provádí čtyři kroky. V prvním kroku se zkopíruje konkrétní soubor `SkeletonMethod.java` z adresáře domény do pracovního adresáře. V druhém kroku se použije vlastní akce `generovanijavadoc` pro vygenerování Javadoc dokumentace souboru `SkeletonMethod.java`. Ve třetím kroku se do javaskriptové proměnné `foundFiles` uloží všechny soubory v pracovním adresáři validace. Poslední krok vypíše obsah proměnné `foundFiles`.

Správně fungující doména bude mít na konci validace výpis proměnné `foundFiles`, který bude zahrnovat soubory vygenerované `javadoc` nástrojem. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.2 porovnatpng

Vlastní akce slouží pro porovnání dvou png obrázku na identičnost.

Pro modul byla vytvořena ukázková doména „ukazka-kontrola_png“. Ukázková doména provádí tři kroky. V prvním kroku se rozbalí vstupní soubor. Pro tuto ukázkovou doménu je to archiv `odevzdavany_png.zip` pro nesouhlasící obrázky a `vzorovy_png.zip` pro totožné obrázky. V druhém kroku se z adresáře domény zkopíruje vzorový png obrázek, vůči kterému se bude odevzdávaný obrázek porovnávat. V posledním kroku se použije vlastní akce `porovnatpng`.

Správně fungující doména bude mít na konci validace v případě odevzdání `odevzdavany_png.zip` chybu, protože oba obrázky nejsou totožné, a v případě odevzdání `vzorovy_png.zip` výsledek OK, protože oba obrázky jsou totožné. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.3 kontrolajavadoc

Vlastní akce slouží pro kontrolu úplnosti vygenerované Javadoc dokumentace podle vybraných parametrů (např. zda se mají kontrolovat private metody, public metody, atd.).

Pro modul byla vytvořena ukázková doména „ukazka-kontrolajavadoc“. Ukázková doména provádí dva kroky. V prvním kroku se rozbalí vstupní soubor `ukazka-kontrolajavadoc.zip`, který obsahuje vygenerovanou Javadoc dokumentaci. V druhém kroku se použije vlastní akce `kontrolaJavaDoc`, která slouží pro kontrolu Javadoc komentářů.

Správně fungující doména bude mít na konci validace stav, jehož výsledek je dán volbou parametrů u akce `kontrolajavadoc`. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.4 kontrolajmenaodevzdavanehosouboru

Vlastní akce slouží pro kontrolu názvu odevzdávaného souboru. Soubor musí splňovat tvar, který je u akce zadán pomocí regulárního výrazu.

Pro modul byla vytvořena ukázková doména „ukazka-kontrolajmenaodevzdavanehosouboru“. Ukázková doména provádí jediný krok, ve kterém se používá vlastní akce `kontrolajmenaodevzdavanehosouboru`. Vlastní akce je nastavena tak, aby přijámala pouze vstupní soubor v regulárním tvaru `?h?j.*`.

Správně fungující doména bude mít na konci validace výsledek OK, pokud se odevzdá soubor s názvem, který odpovídá regulárnímu výrazu (např. `ahoj.txt`), nebo chybu, pokud se odevzdá soubor s jiným názvem. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.5 kontrolanepovolenychsouboru

Vlastní akce slouží pro prohledání adresáře na výskyt nepovolených typů souborů. Přípony typů souboru se zadávají parametrem akce.

Pro modul byla vytvořena ukázková doména „ukazka-kontrolanepovolenych-

souboru“. Ukázková doména provádí dva kroky. V prvním kroku se rozbalí vstupní soubor. Pro tuto ukázkovou doménu je to archiv `ukazka-kontrola-typusouboru.zip`, který obsahuje bmp obrázek a podadresář `podadresar`, který obsahuje soubory `ahoj.TXT` a `nazdar.txt`. V druhém kroku se použije vlastní akce `kontrolanepovolenychsouboru`, která má zapnuté prohledávání podadresářů a jako nepovolené typy nastavené TXT a bmp soubory. Akce je case-sensitive, tudíž rozlišuje malá a velká písmena.

Správně fungující doména bude mít na konci validace chybu, protože vstupní soubor obsahuje nepovolené typu. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.6 kontrolavelikostiodevzdavanehosouboru

Vlastní akce slouží pro zkontrolování velikosti vstupního souboru. Požadovanou povolenou velikost si uživatel určí parametrem.

Pro modul byla vytvořena ukázková doména „ukazka-kontrolavelikostiodevzdavanehosouboru“. Ukázková doména provádí jediný krok, ve kterém se používá vlastní akce `kontrolavelikostiodevzdavanehosouboru`.

Tato akce je nastavena tak, aby přijímala vstupní soubor o velikosti v rozmezí 5 až 10 bytech.

Správně fungující doména bude mít na konci validace chybu, pokud se odevzdá soubor, který je mimo uvedené rozmezí, a výsledek OK, pokud bude mít odevzdaný soubor velikost spadající do uvedeného intervalu. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.7 kopirovatdoworkdir

Vlastní akce slouží pro zkopírování souborů do pracovního adresáře z adresáře domény. Soubory ke zkopírování se zadávají seznamem nebo regulárním výrazem.

Pro modul byla vytvořena ukázková doména „ukazka-kopirovatdoworkdir“. Ukázková doména provádí tři kroky. V prvním kroku se použije vlastní akce

`kopirovatdoworkdir`, která zkopíruje z adresáře domény do pracovního adresáře soubor `validation_data/vypisovac.jar`. Ve druhém kroku se prohledá obsah pracovního adresáře akcí `najdisoubory` a všechny nalezené soubory se uloží do proměnné `nalezeneSoubory`. V posledním kroku se vypíše obsah proměnné.

Správně fungující doména bude mít na konci validace ve výpisu nalezených souborů soubor `vypisovac.jar`, který byl v rámci druhého kroku zkopírován z adresáře validace do pracovního adresáře. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.8 `kopirovatslozkydoworkdir`

Vlastní akce slouží pro zkopírování adresářů do pracovního adresáře z adresáře domény. Adresáře ke zkopírování se zadávají seznamem nebo regulárním výrazem.

Pro modul byla vytvořena ukázková doména „ukazka-kopirovatslozkydoworkdir“. Ukázková doména provádí jediný krok, ve kterém se používá vlastní akce `kopirovatslozkydoworkdir`. Tato akce je nastavena tak, aby kopírovala adresáře `kopirovat1` a `kopirovat2` z adresáře domény do pracovního adresáře.

Správně fungující doména bude mít na konci validace zprávu o tom, že se úspěšně zkopírovaly adresáře `kopirovat1` a `kopirovat2`. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.9 `najdiadresare`

Vlastní akce slouží pro vyhledání adresářů v pracovním adresáři a uložení nalezených adresářů do proměnné. Adresáře lze vyhledat seznamem či regulárním výrazem.

Pro modul byla vytvořena ukázková doména „ukazka-najdiadresare“. Ukázková doména provádí čtyři kroky. V prvním kroku se rozbalí vstupní soubor. Vstupní archiv musí obsahovat dva prázdné adresáře `ahoj` a `ahoj2`. Ve druhém kroku se použije vlastní akce `najdiadresare`, která je nastavena

tak, aby použila regulární výraz `ahoj*`, nalezené adreáře uložila do proměnné `nalezeneAdresare` a počet nalezených adresářů do proměnné `pocetAdresaru`. Ve třetím kroku se testuje, zda akce nalezla právě dva adresáře – pokud ne, validace skončí chybou. V posledním kroku se vypíše obsah proměnné `nalezeneAdresare`.

Správně fungující doména vypíše na konci validace nalezené adresáře. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.10 najdisoubory

Vlastní akce slouží pro vyhledání souborů v pracovním adresáři a uložení nalezených souborů do proměnné. Soubory lze vyhledat seznamem či regulárním výrazem.

Pro modul byla vytvořena ukázková doména „ukazka-najdisoubory“. Ukázková doména provádí čtyři kroky. V prvním kroku se rozbalí vstupní soubor. Vstupní archiv obsahuje dva prázdné soubory `ahoj.txt` a `ahoj2.txt`. Ve druhém kroku se použije vlastní akce `najdisoubory`, která je nastavena tak, aby použila regulární výraz `ahoj*.txt`, nalezené soubory uložila do proměnné `nalezeneSoubory` a počet nalezených souborů do proměnné `pocetSoubory`. Ve třetím kroku se testuje, zda akce nalezla právě dva soubory – pokud ne, validace skončí chybou. V posledním kroku se vypíše obsah proměnné `nalezeneSoubory`.

Správně fungující doména vypíše na konci validace nalezené soubory. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.11 porovnejstrukturyadresaru

Vlastní akce slouží pro porovnání struktury dvou adresářů. Adresáře musí být totožné nebo mohou být v porovnávaném adresáři soubory navíc, pokud je zaškrtnut parametr „Jsou extra soubory chyba?“. V opačném případě vloží akce do výsledku chybu.

Pro modul byla vytvořena ukázková doména „porovnejstrukturyadresaru“. Ukázková doména provádí dva kroky. V prvním kroku se z adresáře validace zkopírují do pracovního adresáře adresáře `vzorovyAdresar` a `porovnavanyAdresar`. Ve druhém kroku se použije akce `porovnejstrukturyadresaru`, která porovná zmíněné dva adresáře.

Správně fungující doména vypíše na konci validace výsledek OK, pokud není v druhém kroku zaškrtnuta možnost „Jsou extra soubory chyba?“, nebo chybu, pokud tato možnost je zaškrtnuta. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.12 `porovnejsoubory`

Vlastní akce slouží pro porovnání textového obsahu dvou souborů. Soubory musí být totožné, jinak akce vloží do výsledku chybu.

Pro modul byla vytvořena ukázková doména „ukazka-porovnejsoubory“. Ukázková doména provádí tři kroky. V prvním kroku se z adresáře validace zkopíruje soubor `vzorovy.txt`. Ve druhém kroku se tento soubor uloží do proměnné, aby tento soubor šlo použít jako parametr v jiné akci. V posledním kroku se použije akce `porovnejSoubory`, která porovná obsah souboru `vzorovy.txt` a `porovnavany.txt`, který je součástí ukázkového archivu `ukazka-porovnej-soubory.zip`.

Správně fungující doména vypíše na konci validace výsledek OK, protože jsou oba soubory totožné. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.13 `rozbalzipjar`

Vlastní akce slouží pro rozbalení vstupního souboru do vybraného adresáře.

Pro modul byla vytvořena ukázková doména „ukazka-rozbalzipjar“. Ukázková doména provádí dva kroky. V prvním kroku se rozbalí vstupní soubor do pracovního adresáře. Ve druhém kroku se vypíše obsah pracovního adresáře.

Správně fungující doména vypíše na konci validace seznam souborů, které

se nacházely v odevzdaném archivu. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.14 `smazatsouboryadresare`

Vlastní akce slouží pro vymazání vyjmenovaných souborů či adresářů v zadaném adresáři.

Pro modul byla vytvořena ukázková doména „ukazka-smazanisouboru“. Ukázková doména má tři kroky. V prvním kroku se zkopíruje vzorový adresář `ahoj` z adresáře domény do pracovního adresáře. Ve druhém kroku se vyhledá a uloží do proměnné soubor `vysledky.txt`, který je součástí adresáře `ahoj`. V posledním kroku se použije akce `smazatsouboryadresare` pro smazání souborů uvedených v parametru. V parametru jsou uvedeny soubory jednak pomocí odkazu na proměnnou a jednak prostým vyjmenováním.

Správně fungující doména bude mít na konci validace v pracovním adresáři adresář `ahoj`, ve kterém budou chybět soubory, které byly uvedeny v parametru třetího kroku. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.15 `spoctislovaastrany`

Vlastní akce slouží pro spočtení počtu slov a stran v doc / pdf dokumentu.

Pro modul byla vytvořena ukázková doména „ukazka-spoctislovaastrany“. Ukázková doména má tři kroky. V prvním kroku se použije akce `spoctislovaastrany` pro spočtení počtu slov a stran odevzdaného pdf / doc dokumentu, a výsledky uloží do proměnných `pocetSlov` a `pocetStran`. V druhém kroku se vypíše počet slov. V posledním kroku se vypíše počet stránek.

Správně fungující doména bude mít na konci validace ve výsledku takový počet slov a stran, jaký má odevzdaný dokument. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.16 spustenijarprogramu

Vlastní akce slouží pro spuštění jar programu. Jar programu lze předat parametry běžným způsobem jako z příkazové řádky.

Pro modul byla vytvořena ukázková doména „ukazka-spustenijarprogramu“. Ukázková doména má tři kroky. V prvním kroku se do pracovního adresáře z adresáře validace zkopíruje spustitelný jar program a textový soubor. Jar program provádí to, že vypisuje parametry předané při spuštení a pokud parametr začíná přepínačem `-f`, tak z parametru vytvoří objekt typu `java.io.File` a vypíše, zda soubor existuje. Ve druhém kroku se do proměnné `nalezeneSoubory` uloží zkopírovaný textový soubor. V posledním kroku se použije akce `spustenijarprogramu`, která spustí spustelný jar program a předá mu v parametru odkaz na textový soubor.

Správně fungující doména bude mít na konci validace ve výsledku standardní a chybový výstup spouštěného programu, což je v tomto případě vypsání vstupních parametrů a zpráva, že textový soubor existuje. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.17 spustenijavaprogramu

Vlastní akce slouží pro spuštění binárního java programu. Programu lze předat parametry běžným způsobem jako z příkazové řádky.

Pro modul byla vytvořena ukázková doména „ukazka-spustenijavaprogramu“. Ukázková doména má tři kroky. V prvním kroku se do pracovního adresáře z adresáře validace zkopíruje binární soubor programu a textový soubor. Program provádí to, že vypisuje parametry předané při spuštení a pokud parametr začíná přepínačem `-f`, tak z parametru vytvoří objekt typu `java.io.File` a vypíše, zda soubor existuje. Ve druhém kroku se do proměnné uloží zkopírovaný textový soubor. V posledním kroku se použije akce `spustenijavaprogramu`, která spustí spustelný binární soubor a předá mu v parametru odkaz na textový soubor.

Správně fungující doména bude mít na konci validace ve výsledku standardní a chybový výstup spouštěného programu, což je v tomto případě vypsání vstupních parametrů a zpráva, že textový soubor existuje. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.18 spustenikonkretnihojunittestu

Vlastní akce slouží pro spuštění jednoho JUnit testu v přeloženém binárním java souboru.

Pro modul byla vytvořena ukázková doména „ukazka-spustenikonkretnihojunittestu“. Ukázková doména má tři kroky. V prvím kroku se do pracovního adresáře zkopírují zdrojové soubory z adresáře domény. Ve druhém kroku se zdrojové kroky zkompilují. V posledním kroku se spustí test `testObsah` v souboru s testy `TestPraceSeCtvercem`. Test `testObsah` pouze vypíše zprávu, že se spustil test `testObsah`.

Správně fungující doména bude mít na konci validace výpis, že test prošel a že se sputil test `testObsah`. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.19 spustenipmd

Vlastní akce slouží pro spuštění programu PMD nad předaným java zdrojovým souborem. PMD je program, který analyzuje zdrojový kód a upozorňuje na přestupky vůči nadefinovaným pravidlům.

Pro modul byla vytvořena ukázková doména „ukazka-spustenipmd“. Ukázková doména má dva kroky. V prvním kroku se do pracovního adresáře zkopíruje z adresáře domény soubor pravidel, používaný programem PMD. Ve druhém kroku se použije akce `spustenipmd`, která spustí program PMD nad vstupním zdrojovým souborem, který se musí jmenovat `Ctverec.java`.

Správně fungující doména bude mít na konci validace výpis porušení pravidel nebo výsledek OK, pokud nebudou žádná pravidla porušena. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.20 spusteniumltestovani

Vlastní akce slouží pro testování UML diagramů vytvořených v programu UMLet. Jedná se o starší akci, která v sobě kombinuje více funkcí. Akce fun-

guje tak, že pro každý soubor v pracovním adresáři s koncovkou uxf spouští speciální jar program, který je předán v parametru akce a následně vyhodnotí výsledek programu. Jedná se o specializovanější verzi akce `spustenijarprogramu` pro potřeby předmetu KIV/OOP.

Pro modul byla vytvořena ukázková doména „ukazka-spusteniumltestovani“. Ukázková doména má tři kroky. V prvním kroku se do pracovního adresáře zkopíruje z adresáře domény jar soubor, který provádí testování. Ve druhém kroku se do proměnné uloží uxf soubor, který se odevzdává. V posledním kroku se použije akce `spusteniumltestovani`, která spustí spustitelný jar soubor a předá mu uxf soubor.

Správně fungující doména bude mít na konci validace zprávu, že testy proběhly v pořádku. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.21 `spustenivsechjunittestu`

Vlastní akce slouží pro spuštění všech JUnit testů v přeloženém binárním java souboru.

Pro modul byla vytvořena ukázková doména „ukazka-spustenivsechjunittestu“. Ukázková doména má tři kroky. V prvím kroku se do pracovního adresáře zkopírují zdrojové soubory z adresáře domény. Ve druhém kroku se zdrojové kroky zkompilují. V posledním kroku se spustí všechny testy v souboru `TestPraceSeCtvercem`.

Správně fungující doména bude mít na konci validace zprávu, že všechny testy prošly. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.22 `vlozitscreenshot`

Vlastní akce slouží pro vložení png obrázku v pracovním adresáři do výstupu validace.

Pro modul byla vytvořena ukázková doména „ukazka-vlozitscreenshot“.

Ukázková doména má dva kroky. V prvním kroku se do pracovního adresáře zkopíruje z adresáře domény adresář `ahoj`, který obsahuje png obrázek – screenshot. V druhém kroku se použije akce `vlozitscreenshot`, která přidá do výstupu validace veřejně přístupný odkaz na daný obrázek.

Správně fungující doména bude mít na konci validace funkční odkaz na obrázek. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

5.23 `vymazsouborypodlevzoru`

Vlastní akce slouží pro vymazání souborů v cílovém adresáři podle vzorového adresáře.

Pro modul byla vytvořena ukázková doména „ukazka-vymazsouborypodlevzoru“. Ukázková doména má tři kroky. V prvním kroku se rozbalí vstupní soubor, což je pro tuto doménu archiv `ukazka-vymazsouborypodlevzoru.zip`. Archiv obsahuje jediný adresář, který má vnořenou další strukturu. V druhém kroku se zkopíruje do pracovního adresáře z adresáře domény vzorový adresář. V posledním kroku se použije akce `vymazsouborypodlevzoru`, která vymaže obsah adresáře z rozbaleného archivu podle vzorové složky.

Správně fungující doména bude mít na konci validace zprávu, že byly smazány soubory, které se vyskytovaly jak ve vzorovém tak cílovém adresáři. Otestování ukázkovou doménou při mnou prováděných testech proběhlo úspěšně.

6 Závěr

Validační server se používá na Západočeské univerzitě v Plzni pro automatizované testování studentských prací odevzdaných přes Portál ZČU u některých předmětů na Katedře informatiky a výpočetní techniky.. Pro správu vývoje validačního serveru a vlastních akcí se nyní používá systém Redmine a SVN. Do Redmine se píší veškeré náměty na vylepšení či nové moduly.

Existující náměty na vylepšení funkčnosti validačního serveru byly v Redmine vedeny jako 17 úkolů. Pro 17 těchto úkolů jsem vytvořil analýzu a plán řešení. V Redmine během práce přibývaly další úkoly. V rámci této práce jsem vyřešil celkem 34 úkolů v Redmine, přičemž 17 z nich jsem udělal navíc oproti původnímu plánu – pro tyto úkoly navíc jsem také vytvořil analýzu.

Na základě analýz jednotlivých úkolů jsem udělal návrh řešení, provedl implementaci navrhnutého řešení a popsal příklad použití. Vyjímkou byly úkoly, u kterých se jednalo o hlášení bugů a které jsem nebyl schopný zreprodukovat. Dokumentace byla vytvořena jednak napsáním nápovědy pro vlastní akce a jednak historií commitů do SVN a propojením těchto commitů s úkoly v Redmine.

Všechny momentálně existující vlastní akce mají ukázkovou doménu, která zároveň slouží i jako uživatelský test. Ukázkové domény jsou funkční a prokazují tak funkčnost implementace.

Všechny body zadání jsem splnil. Největší změny, které se mi podařilo implementovat, jsou:

- nový způsob definování vlastních akcí pomocí konfiguračního souboru a nápovědy,
- možnost definovat výchozí hodnotu pro parametry vlastních akcí,
- možnost odkazovat javaskriptové proměnné pomocí dolaru v parametrech vlastních akcí.

Validační server je aktivně používán a náměty na jeho vylepšení se stále objevují, takže je možné dále pokračovat v jeho úpravách. Další rozšíření do budoucna by mohlo být například odstranění Java skriptletů z jsp souborů a přidání dynamického chování do webových stránek pomocí javaskriptu.

Seznam zkratek

CIV	Centrum informatizace a výpočetní techniky - stará se o provoz informační technologie na ZČU
JAR	Java Archive - stromová reprezentace syntaktické struktury zdrojového kódu
JSP	JavaServer Pages - technologie pro vytváření dynamických webových stránek
HTML	HyperText Markup Language - značkovací jazyk pro vytváření webových stránek
OOP	Object-oriented Programming - objěktově orientované programování
SVN	Subversion - systém pro správu verzí

Literatura

[1] *Apache Commons* [online]. 17.3.2014 [cit. 2014-03-29], dostupné z: http://commons.apache.org/.

[2] *Maven : the definitive guide*. Sebastopol, Calif: Oreilly, 2008, ISBN 978-0596517335.

[3] DUDOVÁ, V.: *Webová konfigurace validačního serveru*. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2010.

[4] Freeman, E.; Freeman, E.; Sierra, K.; aj.: *Head First design patterns*. Sebastopol, CA: O'Reilly, 2004, ISBN 0-596-00712-4.

[5] Janák, J.: *Issue Tracking Systems*. Diplomová práce, Masaryova Univerzita, Fakulta informatiky, 2009.

[6] Patton, R.: *Software testing*. Indianapolis, IN: Sams Pub, 2006, ISBN 978-0-672-32798-8.

[7] Pilato, C.: *Version control with Subversion*. Sebastopol, CA: O'Reilly Media, 2008, ISBN 978-0-596-51033-6.

[8] Redmine: *Overview - Redmine* [online]. 28.3.2014 [cit. 2014-03-28], dostupné z: http://www.redmine.org/.

[9] Valenta, L.: *Validační server* [online]. Publikováno v září 2007 [cit. 2014-03-15, dostupné z: http://vs.kiv.zcu.cz/doc/index.html.

[10] ariejan de vroom: *How to resolve Subversion Conflicts* [online]. 4.2.2014 [cit. 2014-03-27], dostupné z: http://ariejan.net/2007/07/04/how-to-resolve-subversion-conflicts/.

A Parametry validační domény

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties version="1.0">

    <comment>Defaultní nastavení domény</comment>

    <!-- zda je domena aktivni -->
    <entry key="domain.enabled">1</entry>

    <!-- ucuje, zda je zapnut 'debug' rezim, tj. zda mj. prochazi kompletni stacktrace do vystupnich
         html -->
    <entry key="core.debug">0</entry>

    <!-- zda po sobe uklizet docasne soubory - pro ladeni. 1/0 -->
    <entry key="core.do_cleanup">0</entry>

    <!-- pokud je core.concurrency_disabled = 1, pak je zakazan jakykoliv paralelismus,
     tj. ulohy validace se spousteji za sebou -->
    <entry key="core.concurrency_disabled">0</entry>

    <!-- maximalni velikost odevzdaneho souboru v KB -->
    <entry key="max_file_size">50</entry>

    <!-- seznam pripon souboru, ktere se akceptuji. Pokud neni uvedeno nebo obsahuje znak '*',
         akceptuji se vsechny soubory -->
    <entry key="accept_extensions">*</entry>

    <!-- maxilani doba celeho procesu validace v teto domene sekundach.
     Default hodnota je 30 vterin, neni-li uvedena -->
    <entry key="max_validation_time">30</entry>

    <!-- nazev souboru s lokalizacnimi hlaskami pro tuto domenu. Cesta je relativne
    vuci adresari domeny! -->
    <entry key="resource_bundle">nl.texts</entry>

    <!-- nazev souboru s java policy nastavenim. Vyuzije se v pripade, kdy se spousti Java tridy.
    Pote je tento soubor predan nove spustenemu JVM jako parametr a z nej se nactou bezpecnostni
         opatreni.
     Cesta je relativne k adreasri domeny.-->
    <entry key="java_policy_file">run.policy</entry>

    <!-- Jake vsechny vystupy validace generovat -->
    <entry key="custom.output.classes">cz.zcu.validationserver.output.HTMLResultOutput</entry>

    <!-- jake generovat typy vystupu: moznosti: 'file', 'string' ci oboji oddelene carkou -->
    <entry key="html_output_type">file</entry>

    <!-- zde musi byt cesta k html adresari, kam se ukladaji vysledne html soubory. Tato
     cesta se nebere relativne vuci data adresari! -->
    <entry key="html_output_dir">data/html</entry>

    <!-- zacatek URL k http serveru, pres ktery budou vysledky pristupne -->
    <entry key="html_output_url">http://localhost/vs/</entry>
```

```
    <!-- url na css styl pro exportovane html soubory -->
    <entry key="html_css_url">/validator/vs.css</entry>

     <!-- odkaz na pouzite knihovny, ktere pro domenu pouzit -->
    <entry key="domain.classpath">iText-2.0.8.jar</entry>

    <entry key="compiler.java.options">-encoding UTF-8</entry>

    <entry key="core.default.charset">UTF-8</entry>

    <!-- urcuje, zda vypnout vyzadovani vstupniho souboru pri testovani domeny -->
    <entry key="domain.disable_test_file">0</entry>

</properties>
```

B Třída `VlastniAkceUtils`

```
package cz.zcu.validationserver.utils;

import java.io.File;

import org.apache.log4j.Logger;
import org.mozilla.javascript.Context;
import org.mozilla.javascript.Script;
import org.mozilla.javascript.Scriptable;

import cz.zcu.validationserver.validation.ValidationInfo;


/**
 * Ruzne pomocne metody pro pouziti ve vlastnich akcich
 *
 * Created: 30. 7. 2013 - 19:49:21
 * @author Hung Duong Manh (zer0@students.zcu.cz), e-mail: zer0@students.zcu.cz
 **/
public class VlastniAkceUtils {

    private static final Logger logger = Logger.getLogger(VlastniAkceUtils.class);
    private static final boolean isDebugEnabled = logger.isDebugEnabled();


    /**
     * Converts given user input into file accordingly to type of input.
     *
     * @param patternFileRough
     *            User String input.
     * @param sourceName
     *            A string describing the source, such as the name of action
     * @param info
     *            Reference to info.
     * @param scope
     *            Reference to scope.
     * @param jsContext
     *            Reference to context.
     *
     * @return File object which represent user input.
     */
    public static File convertToFile(String patternFileRough, String sourceName, ValidationInfo info,
          Context jsContext, Scriptable scope) {

        if (patternFileRough.isEmpty()) {

            return info.getInputFile();

        }
        else if (patternFileRough.startsWith("$")) {

            return convertToFileFromJavaScript(patternFileRough.substring(1), sourceName, jsContext,
                  scope);

        }
        else {
```

```
            // return new file with given name in workdir
            return new File(info.getWorkDir() + File.separator + patternFileRough.replaceAll("/",
                File.separator));
        }
    }

    /**
     * Converts given String containing name of an JavaScript variable into File
     * object.
     *
     * @param patternFileRough
     *            Name of JavaScript variable.
     *
     * @param jsContext
     *            Reference to context.
     *
     * @param scope
     *            Reference to scope.
     *
     * @return Returns new instance of an File object.
     */
    public static File convertToFileFromJavaScript(String patternFileRough, String sourceName,
         Context jsContext, Scriptable scope) {

        // zjištění zadaného adresáře
        Script souborScript = jsContext.compileString(patternFileRough + ";", sourceName, 0, null);
        Object vysledek = souborScript.exec(jsContext, scope);

        return (File) Context.jsToJava(vysledek, File.class);
    }

    /**
     * Prevede regularni vyraz zadany v DOS tvaru (znak '?' znamena prave jeden
     * vyskyt, znak '*' znamena 0 az n vyskytu) na regularni vyraz stejneho
     * vyznamu, ale pouzitelneho pro volani Pattern.matches(). Pokud regularni
     * vyraz zustal v puvodnim tvaru, zpracovani regularniho vyrazu by
     * nefungovalo spravne.
     *
     * @param regex
     *            Puvodni regularni vyraz zapsany v DOS tvaru.
     * @return Novy regularni vyraz ve tvaru vyhovujicimu Pattern.matches().
     */
    public static String convertFromDOSRegex(String regex) {

        regex = regex.replace("*", "\\S*");

        regex = regex.replace("?", "\\S");

        // protoze . znamena "Matches any character"
        regex = regex.replace(".", "\\.");

        return regex;
    }
}
```