

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

Diplomová práce

Nástroje pro podporu testování

Originál zadání diplomové práce.

Poděkování

Děkuji vedoucímu diplomové práce Doc. Ing. Pavlu Heroutovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost během konzultací.

Prohlášení

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

V Plzni dne 15. května 2014

Jan Rada

Abstract

Supporting tools for testing

This diploma thesis deals with the supporting tools for testing, especially the *Selenium*, *JMeter* and *Redmine* tools. The main objective of the thesis was to propose and prepare several small student projects for the needs of the “KIV/OKS” (Software Quality Assurance) subject instructed at the University of West Bohemia. With regard to the requirement of automated validation of the submitted assignments I selected the combination of the *Selenium* and *JUnit* technologies as the most suitable for this purpose. Two simple web applications containing intentionally entered errors were created as a basis for testing. Finally, applications in the *Java* programming language were created, serving for validation of each of the couple of designed projects. The applications are used for local inspection of projects performed by students before the projects are submitted and are also used as a basis for automated validation as soon as the projects are submitted to the server.

Abstrakt

Nástroje pro podporu testování

Tato diplomová práce se zabývá nástroji pro podporu testování, zejména nástroji *Selenium*, *JMeter* a *Redmine*. Hlavním cílem práce bylo navrhnout a připravit několik malých studentských projektů pro potřeby předmětu „KIV/OKS“ (Ověřování kvality software) vyučovaného na Západočeské univerzitě v Plzni. S ohledem na požadavek automatické validace těchto úloh při jejich odevzdávání jsem pro tento účel vybral jako nejvhodnější použití kombinace technologií *Selenium* a *JUnit*. Jako podklad pro testování byly vytvořeny dvě jednoduché webové aplikace obsahující záměrně zanesené chyby. Na závěr byly vytvořeny aplikace v programovacím jazyku *Java*, které slouží k validaci každého z dvojice navržených projektů. Tyto aplikace jsou využívány k lokální kontrole projektu prováděné studentem před jeho odevzdáním a jsou také použity jako základ automatické validace při odevzdávání projektů na server.

Obsah

1	Úvod	1
2	Testování softwaru	2
2.1	Vymezení pojmu	2
2.2	Důvody testování	2
2.3	Kategorie testů	3
2.4	Ideální počet a typ testů	4
2.5	Automatizované testování	5
3	Nástroje pro podporu testování	6
3.1	Vymezení pojmu	6
3.2	JUnit	6
3.3	TestNG	7
3.4	Selenium	8
3.4.1	Selenium IDE	8
3.4.2	Selenium RC	11
3.4.3	Selenium WebDriver	12
3.4.4	Selenium Grid	14
3.4.5	Selenium 2 + JUnit	14
3.4.6	Selenium 2 + TestNG	16
3.4.7	Selenium a návrhové vzory	16
3.5	JMeter	17
3.5.1	Základní elementy testového plánu	19
3.6	Redmine	20
4	Validátor	22
4.1	Validační doména	22
4.2	Princip validace	23
5	Výběr nástrojů pro realizaci projektů	25
5.1	Selenium	25
5.2	JMeter	26
5.3	Redmine	27
5.4	Shrnutí	27

6	Webová aplikace pro testování	28
6.1	Převodník	29
6.1.1	Úmyslné chyby	30
6.2	Diskusní fórum	31
6.2.1	Úmyslné chyby	32
7	Projekt „oks-09“	33
7.1	Cíl projektu	33
7.2	Analýza	33
7.3	Řešení	34
7.3.1	Ověření možností Validátoru	34
7.3.2	Výsledný způsob kontroly	35
7.4	Kontrolní program	36
7.4.1	Popis tříd programu	37
7.5	Validační doména	39
7.6	Zadání projektu	40
7.7	Zhodnocení	40
8	Projekt „oks-10“	41
8.1	Cíl projektu	41
8.2	Analýza	41
8.2.1	JUnit vs. TestNG	41
8.2.2	Vytváření instance WebDriverů	42
8.2.3	Použití konstant pro lokátory	43
8.2.4	Volání privátních metod	43
8.3	Řešení	44
8.4	Kontrolní program	45
8.5	Validační doména	46
8.6	Zadání projektu	46
8.7	Zhodnocení	46
9	Závěr	47
	Slovník použitých termínů a zkratk	48
	Použité zdroje a literatura	50
	Přílohy	52
	Příloha A – Specifikace webové aplikace Převodník	53
	Příloha B – Specifikace webové aplikace Diskusní fórum	56

Příloha C – Obsah přiloženého CD	61
Příloha D – Uživatelská příručka	63

1 Úvod

Od začátku letního semestru akademického roku 2013/2014 je na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni vyučován předmět „KIV/OKS“ (Ověřování kvality software). Náplní tohoto předmětu je seznámit studenty se základními principy zajištění kvality a testování software včetně získání praktických zkušeností se softwarovými nástroji užívanými k podpoře těchto činností [1]. Pro potřeby předmětu bylo nezbytné navrhnout několik menších studentských projektů sloužících k doplnění výuky a praktickému seznámení s některými nástroji a technologiemi.

Tato diplomová práce se zaměřuje zejména na nástroje *Selenium*, *JMeter* a *Redmine*. Cílem práce je prozkoumat možnosti výše uvedených nástrojů, na základě získaných poznatků vybrat některé z nich pro přípravu studentských úloh a tyto úlohy kompletně připravit. Důležitým požadavkem je možnost automatické validace úloh při jejich odevzdávání.

V rámci přípravy studentských úloh je zapotřebí formulovat zadání s ohledem na skutečnost, že výstup by mělo být možné automaticky validovat s použitím prostředků již existující aplikace validátoru. Dále je nutné vyhotovit podklady pro řešení navržených úloh (testovanou aplikaci obsahující úmyslně zanesené chyby) a vzorová řešení. Rovněž je vyžadováno pro jednotlivé navržené úlohy vytvořit kontrolní programy. Tyto programy budou mít k dispozici studenti a budou sloužit k lokální kontrole zpracovaných úloh před jejich odevzdáním. Na závěr by měla být připravena automatická kontrola *Validátorem*, prováděná při odevzdávání úloh na server. Realizace tohoto bodu spočívá v sestavení tzv. *validační domény*, která je tvořena posloupností kroků vedoucích k odhalení nedostatků v odevzdávaných úlohách. Stěžejní částí této kontroly bude spuštění totožného kontrolního programu, který budou mít k dispozici studenti.

2 Testování softwaru

Následující text obsahuje vysvětlení a ujasnění některých pojmů z oblasti testování. V této části práce bylo čerpáno zejména z [2], [3] a [4].

2.1 Vymezení pojmu

Testování softwaru je proces, jehož cílem je odhalit chyby v testované aplikaci. Chybou obvykle rozumíme rozpor mezi očekávaným a skutečným chováním, ale za chybu může být například považována i špatná uživatelská přívětivost. Jako chybu obecně označujeme nesoulad skutečného stavu s požadavky na aplikaci. Požadavky jsou definovány specifikací produktu. Ta může mít podobu pouze ústní, myšlenou (zpravidla u projektů velmi malého rozsahu), nebo se může jednat o psaný dokument splňující určité náležitosti. Podle literatury [2] hovoříme o chybě (přesněji softwarové chybě) v následujících případech:

- „Software nedělá něco, co by podle specifikace produktu dělat měl.
- Software dělá něco, co by podle specifikace produktu dělat neměl.
- Software dělá něco, o čem se produktová specifikace nezmiňuje.
- Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo – podle názoru testera software – jej koncový uživatel nebude považovat za správný.“

2.2 Důvody testování

Důvodem pro testování softwaru je odhalení chyb v aplikaci ještě před tím, než mohou nějakým způsobem uškodit. Cílem je chyby zachytit v co možná nejranější fázi vývoje produktu. S rostoucí dobou odhalení chyby totiž výrazně rostou náklady (časové, popř. finanční) spojené s její opravou [5]. Dalším důvodem může být snaha ujistit se, že je produkt takový, jaký byl zamýšlen, a zda vyhovuje specifikaci.

Úspěšné vykonání testů ovšem není možné považovat za důkaz správnosti programu. K získání důkazu správnosti by bylo zapotřebí otestovat množinu všech možných vstupů programu (což je většinou vzhledem k její velikosti nereálné), popř. provést matematický důkaz. Testováním lze ale bezpečně odhalit, že je produkt špatně a díky tomu včas zajistit jeho opravu. Vhodné použití testování jednoznačně přispívá ke zvýšení kvality výsledného softwarového produktu.

2.3 Kategorie testů

Testy lze členit do kategorií podle množství kritérií. V následující části textu zmíním pouze nejčastěji používaná dělení.

Jednou z možností je rozdělení testů na dynamické a statické. Klíčový je v tomto případě fakt, zda se v rámci testu provádí (dynamický test) nebo neprovádí (statický test) spuštění testované aplikace.

Dalším často používaným kritériem je úroveň znalosti testovaného kódu. Zde se používá označení testování černé skříňky (black box) nebo bílé skříňky (white box). V některých případech se zavádí i pojem testování šedé skříňky (gray box). Testování černé skříňky znamená, že tester nemá žádné podrobné informace a znalosti o testované aplikaci z hlediska její implementace a testuje pouze na základě známých vstupů a očekávaných výstupů. Tento způsob testování nejlépe simuluje reálné použití aplikace běžným uživatelem. Naopak při testování bílé skříňky jsou testerovi známy veškeré detaily implementace. Díky tomu je možné testy více zacílit na některá problematická místa implementace a více se soustředit na určitou část množiny vstupů. Testování bílé skříňky ovšem přináší riziko, že tester ovlivněný detailní znalostí implementace testy příliš přizpůsobí na míru programu. Testy poté nemusí být účinné. Testování šedé skříňky je kombinací předchozích případů, kdy tester má nějaké informace o implementaci (například zná použitý algoritmus), ale přesná implementace je mu skryta.

Testy je možné rozdělit podle fáze životního cyklu softwarového produktu, v níž se testování provádí. Skupiny testů jsou pak následující:

- **jednotkové testy** – zejména v rané fázi vývoje, testuje se správná funkčnost malých částí systému (jednotek),
- **integrační testy** – testuje se správná funkčnost různých částí systému (jednotek) zapojených do celku,
- **systémové testy** – obsáhlé testy napříč celým systémem,
- **akceptační testy** – kontrola splnění testovacích scénářů, kterými je podmíněno převzetí produktu zákazníkem.

Posledním z nejčastěji používaných způsobů členění testů je členění podle tzv. dimenze kvality na testy:

- **funkčnosti** – testuje se správné chování systému,
- **použitelnosti** – testuje se uživatelská přívětivost,
- **spolehlivosti** – testuje se chování při nedostatku nebo nedostupnosti zdrojů, zotavení po chybě,
- **podpory** – testuje se náročnost instalace, funkčnost na požadovaných SW a HW konfiguracích,
- **výkonu** – testují se požadované odezvy systému při různých úkonech,
- **a další** (lokalizovatelnosti, kompatibility, bezpečnosti, atd.).

2.4 Ideální počet a typ testů

Kvůli rozmanitosti vytvářených softwarových produktů není možné ideální počet a typ testů jednoznačně definovat. Obojí se odvíjí od nároků kladených na danou aplikaci. Zcela odlišné nároky budou kladeny na jednoduchou utilitu určenou pro vlastní potřebu, na informační systém, nebo na software provádějící řízení nějakého kritického procesu. Stanovení optimální hladiny testování je tedy záležitostí kvalifikovaného odhadu, jehož úspěšnost je ovlivněna zkušeností testera. Při tomto

rozhodování mohou být nápomocné některé metriky poskytované softwarovými nástroji – např. pokrytí kódu testy. Typ testů bývá v některých případech omezen samotným produktem. Pokud například nemáme k dispozici jeho zdrojové kódy ani základní informace o implementaci, jsme odkázáni jen na testy černé skříňky. Vyloučeny jsou v tomto případě také statické nebo jednotkové testy. Zaměření produktu souvisí s volbou testů podle dimenze kvality. Například u bankovního systému budou zcela jistě kladeny zvýšené nároky na oblast bezpečnosti. Výsledná množina testů se označuje jako tzv. *testovací mix*.

2.5 Automatizované testování

Základní myšlenkou každého testu je testovací scénář. Ten má podobu několika logicky navazujících kroků a může být předáván ústně nebo může mít například následující podobu:

1. Prostřednictvím příkazové řádky spusťte program *faktorial*.
2. Jako vstup zadejte pomocí klávesnice číslici 5 a potvrďte stiskem klávesy *Enter*.
3. Zkontrolujte, zda textový výstup zapsaný do souboru *vystup.txt* v adresáři *vystup* odpovídá faktoriálu čísla 5, tedy hodnotě 120.

Díky testovacímu scénáři je možné stejný test vykonat opakovaně. U jednoduchých programů malého rozsahu se lze spokojit s manuálním vykonáváním testovacích scénářů přímo testerem. Pokud bychom ale potřebovali otestovat rozsáhlejší aplikaci, byl by tento způsob z hlediska časové náročnosti nevyhovující. S rostoucím počtem prováděných testů by se navíc zvyšovala i pravděpodobnost, že se chyby dopustí samotný tester. Výše uvedený testovací scénář by přitom bylo možné přepsat do podoby jednoduchého skriptu. Tester by pak prováděl pouze jeho spuštění a vyhodnocení výsledku.

Z důvodu usnadnění a zefektivnění procesu testování existuje řada testovacích frameworků. Ty zpravidla přinášejí možnost testovací scénář zapsat ve formě zdrojového kódu, který pak může být opakovaně spouštěn. S použitím některých nástrojů lze automatizovat i spuštění testů (například vždy před buildem produktu). Náplň práce testera pak tvoří psaní nových testů a hlášení chyb, které byly testy odhaleny. Tester by měl dále dohlížet na provedení oprav reportovaných chyb.

3 Nástroje pro podporu testování

3.1 Vymezení pojmu

Označení „nástroje pro podporu testování“ je poměrně široký pojem. Jako zástupce této množiny lze chápat různé testovací frameworky, které jsou přínosné hlavně z hlediska opakovatelnosti testů. Dále do této skupiny můžeme zařadit nástroje sloužící k řízení automatizovaného spouštění testů, jejich vyhodnocování a ukládání výsledků. Také si pod tímto pojmem můžeme představit softwarové produkty zaměřené na reportování chyb a na řízení procesu jejich odstranění. Obecně se tedy dá říci, že mezi nástroje pro podporu testování řadíme jakýkoli softwarový produkt, který s testováním nějakým způsobem souvisí.

3.2 JUnit

V této podkapitole jsem čerpal z [6]. *JUnit* [7] je nejznámější testovací framework zaměřený na tvorbu jednotkových testů v programovacím jazyce *Java*. V důsledku rozsáhlé uživatelské základny je nástroj podporován většinou vývojových prostředí pro tuto platformu a má široké možnosti integrace. Framework přináší oddělení produkčního kódu od testovacího a jde s jeho pomocí úspěšně nahradit i primitivní formy testování typu kontrolních výpisů na standardní výstup. Použití může mít pozitivní vliv také na strukturu zdrojového kódu aplikace, jelikož nutí programátora psát poměrně krátké jednotky kódu (metody). Podporovány jsou dnes již hojně používané anotace a od několika posledních verzí i parametrizované testy nebo registrace vlastních posluchačů. Tvůrci *Kent Beck* a *Erich Gammy* propagují cestu nezávislých testů, u kterých nezáleží na pořadí jejich vykonávání. Podoba jednoduché *JUnit* testovací třídy (tzv. *test case*) je znázorněna v ukázce kódu 3.1.

```
1 public class UkazkovyTest {
2     @Test
3     public void test () {
4         assertTrue ( TestovanaTrida . vratTrue ( ) );
5     }
6 }
```

Ukázka kódu 3.1: Jednoduchý *JUnit* test

V ukázce kódu 3.1 je použita jediná testovací metoda s názvem *test*, ve které je prováděno ověření návratové hodnoty statické metody *vratTrue*. Testovací třídy lze s využitím prostředků *JUnit* seskupovat do větších celků (nazývaných *test suite*) reprezentovaných samostatnou třídou a testy v nich obsažené pak spouštět naráz.

3.3 TestNG

Framework *TestNG* („Test Next Generation“) [8] je inspirován frameworkem *JUnit* a z hlediska použití je tomuto frameworku také velmi podobný. Autory jsou *Cédric Beust* a *Hani Suleiman*. Jedná se o druhý nejpoužívanější testovací framework pro jednotkové testy, takže jeho podpora ve vývojových prostředích je velmi dobrá. Lze jej využívat téměř totožně jako *JUnit*, ale navíc dovoluje i naprosto odlišný přístup a pohled na testování aplikací jednotkovými testy. Nabízí totiž například členění testů (testových metod) v rámci testové třídy do skupin a definici závislostí mezi jednotlivými testy nebo skupinami. Tím umožňuje ovlivnit pořadí vykonání testů. Rozšířené možnosti *TestNG* lze s výhodou využít zejména při integraci s jinými frameworky. Princip použití závislostí mezi testy, které jsou vytvořeny s využitím anotací, je patrný z ukázky kódu 3.2.

```
1 public class UkazkovyTest {
2     @Test
3     public void test1() {
4         assertTrue(TestovanaTrida.vratTrue());
5     }
6
7     @Test(dependsOnMethods = { "test1" })
8     public void test2() {
9         assertFalse(TestovanaTrida.vratFalse());
10    }
11 }
```

Ukázka kódu 3.2: Jednoduchý *TestNG* test

V ukázce kódu 3.2 je definovanou závislostí (řádek 7) zajištěno, že testovací metoda *test2* bude při každém spuštění testů vykonána pouze po úspěšném vykonání metody *test1*. Třídy s *TestNG* testy se dají obdobně jako u *JUnit* sloučit do větších celků, ale zde je každý *test suite* tvořen konfiguračním souborem ve formátu *XML*.

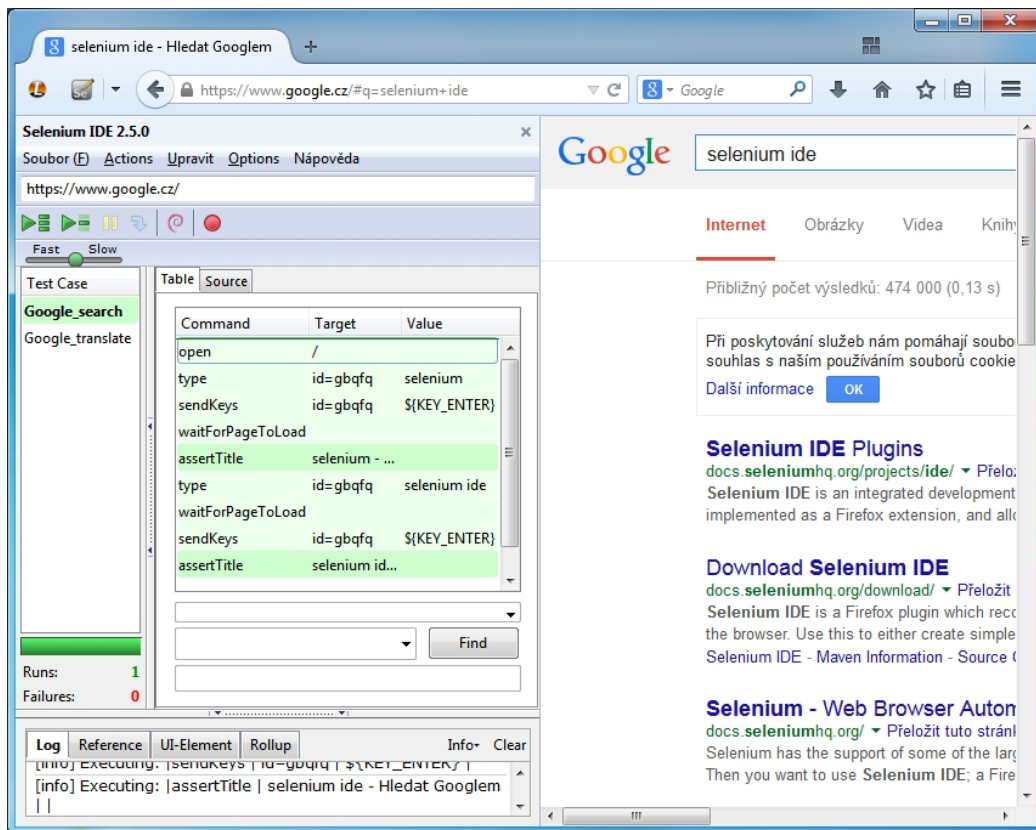
3.4 Selenium

Selenium [9] je sada nástrojů určených primárně k automatizovanému testování webových aplikací. Dovoluje však i jiné způsoby užití – například k automatizaci rutinních úkonů prováděných prostřednictvím webového rozhraní nějakého systému. Pomocí *Selenium* lze dokonale simulovat chování uživatele, jeho interakci s webovou aplikací a testovat tak její funkcionalitu. Jelikož se jedná o testování již vytvořené a spuštěné webové aplikace prostřednictvím *GUI*, spadají *Selenium* testy do kategorie systémových testů a jejich uplatnění je tedy až v pokročilejších fázích vývoje softwarového produktu. Za vývojem projektu *Selenium* v dnešní době stojí mnoho osobností, původním tvůrcem je ale *Jason Huggins*. *Selenium* je v současnosti nejrozšířenějším a nejpoužívanějším produktem tohoto druhu.

3.4.1 Selenium IDE

Selenium IDE je jedním z nástrojů sady *Selenium*. Je distribuováno ve formě doplňku do internetového prohlížeče *Firefox* (viz obr. 3.1). Nástroj je cílen zejména na skupinu uživatelů bez programátorských znalostí a zkušeností, kterým i tak dovoluje vytvářet poměrně komplexní testy. Jeho předností je grafické uživatelské rozhraní umožňující vytvářet testy prostřednictvím záznamu uživatelem prováděných akcí. Takto zaznamenané posloupnosti akcí (testové případy), tvořené dílčími kroky, lze dále editovat nebo rovnou beze změn spouštět. Pro potřeby testování je ale nezbytné doplnit přinejmenším kroky provádějící ověřovací příkazy.

Testy je samozřejmě možné vytvářet i bez využití záznamu a kroky testů přidávat manuálně. Každý krok testu se skládá z trojice: příkaz (command), cíl (target) a hodnota (value). Příkaz tvoří řetězec reprezentující nějakou akci – např. *open* provádí přechod na *URL* adresu. Řetězec zadávaný do pole „cíl“ může mít různou interpretaci. Ve většině případů se jedná o lokátor elementu webové stránky, nad kterým má být vykonána požadovaná akce. U výše uvedeného příkazu *open* je ale v poli „cíl“ očekávána *URL* adresa, na kterou má být proveden přechod. Do posledního z polí („hodnota“) se zadává řetězec, který je v rámci akce použitý jako vstup (například do textového pole formuláře). Během vyplňování polí může být nápomocna on-line nápověda zobrazovaná na kartě „Reference“. Při manuálním vytváření testových případů nebo při editaci testových případů vytvořených záznamem je také velmi užitečná kontextová nabídka zobrazovaná po stisku pravého tlačítka myši na ně-



Obrázek 3.1: *Selenium IDE* – grafické uživatelské rozhraní

kterém z elementů stránky. Nabídka obsahuje návrhy příkazů dostupných pro daný element. Krok testového případu lze tedy jednoduše vytvořit výběrem požadované akce z kontextové nabídky.

Pokud jsou ve vytvořeném testovém případě zadány veškeré přechody mezi stránkami jako relativní adresy (což je doporučováno), je pak snadné test spustit na jiné adrese bez nutnosti jeho velkých úprav. Relativní adresy se vztahují k tzv. *base URL* zobrazené v horní části okna doplňku, kterou stačí přepsat. Toto je výhodné v situacích, kdy shodný test požadujeme spouštět například na testovací i produkční verzi webové aplikace.

Doplněk podporuje vytvoření a uložení sady testů (*test suite*) složené z více testových případů. Jedná se o jedinou možnost, jak po spuštění doplňku načíst více testových případů naráz. Výchozí formát ukládaných sad testů i testových případů je *HTML*.

Jak jsem již zmínil, *Selenium IDE* cílí spíše na uživatele bez programátorských znalostí. Velmi dobře ale poslouží i zkušenějším uživatelům při seznamování s pro-

duktem *Selenium*. Tito uživatelé ale budou postupně narážet na různá omezení. Omezením je například absence řídicích struktur použitelných v testu. Limitující je také spuštění testů výhradně v internetovém prohlížeči *Firefox*. Pokročilejší způsoby použití *Selenium* testů jsou popsány v následujících podkapitolách.

Selenium IDE poskytuje export vytvořených testů do podoby zdrojového kódu v různých programovacích jazycích. Z pohledu této práce je zajímavý zejména export do zdrojového kódu *Javy* využívajícího *JUnit* testy.

Základní příkazy

Základní příkazy pro tvorbu testů jsou uvedeny v tabulce 3.1.

Příkaz	Cíl	Hodnota	Popis
open	<adresa>	—	přechod na požadovanou <i>URL</i> adresu, adresa je prvním parametrem
sendKeys	<lokátor>	<řetězec>	postupný stisk kláves odpovídajících znakům řetězce, který tvoří druhý parametr příkazu, prvním parametrem je lokátor elementu
clickAndWait	<lokátor>	—	událost stisku levého tlačítka myši, parametrem je lokátor elementu
assertTitle	<vzor>	—	porovnání titulku stránky se vzorem
assertText	<lokátor>	<vzor>	porovnání textu elementu (například nadpisu) se vzorem
assertValue	<lokátor>	<vzor>	porovnání obsahu elementu (například obsahu vstupního pole formuláře) se vzorem

Tabulka 3.1: *Selenium IDE* – základní příkazy

Ke každému z dostupných *assert* příkazů (např. *assertTitle*) existuje jeho *verify* varianta (např. *verifyTitle*). Rozdíl mezi *assert* a *verify* příkazy spočívá v odlišném chování při výskytu chyby ověření. U *assert* příkazů je po chybě zbytek testového případu přeskočen. Naopak při selhání *verify* příkazu je zbytek testu vykonán (i tak je ale označen jako chybný).

Lokátory

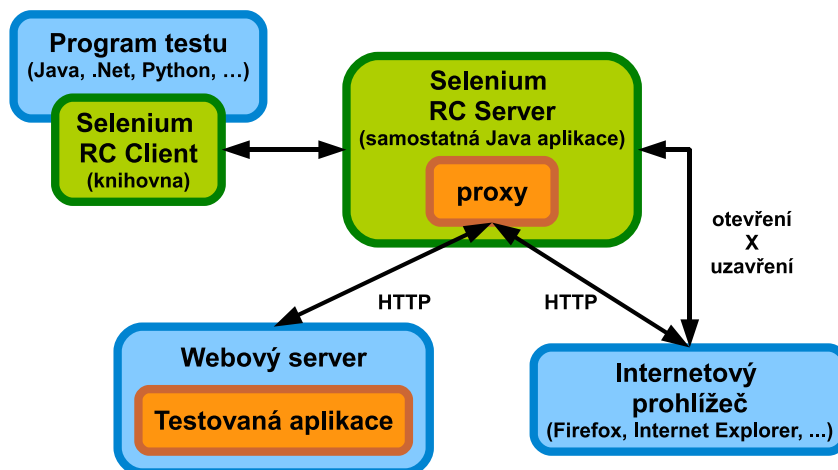
Typy lokátorů (řetězců vyplňovaných do polí „target“ sloužících k výběru elementů webové stránky) použitelných při vytváření testů v *Selenium IDE* jsou následující:

- **identifer** – vybírá elementy webové stránky, jejichž atributy *id* nebo *name* vyhovují zadanému řetězci,
- **id** – vybírá elementy webové stránky s daným *id*,
- **name** – vybírá elementy webové stránky s daným *name*,
- **XPath** – vybírá elementy webové stránky podle *XPath* výrazu,
- **link** – vybírá hypertextové odkazy tvořené daným textem,
- **DOM** – vybírá elementy webové stránky s využitím *DOM* (elementy vyhovující danému *JavaScriptovému* popisu elementů v objektovém modelu stránky),
- **CSS** – vybírá elementy webové stránky vyhovující danému lokátoru kaskádových stylů.

Poznámka: Preference lokátorů používaných při záznamu testového scénáře lze upravovat v nastavení *Selenium IDE*.

3.4.2 Selenium RC

Nástroj *Selenium RC* (někdy též označován jako *Selenium 1*) je oproti *Selenium IDE* určen pokročilejším uživatelům a jeho použití vyžaduje alespoň elementární programátorské znalosti. Základem je tzv. *Remote Control Server*. Jedná se o program napsaný v programovacím jazyku *Java*, který musí být spuštěn před vykonáváním testů. *RC Server* zajišťuje vytvoření okna prohlížeče, interpretuje a spouští *Selenium* příkazy (tzv. „Selenese“) a figuruje jako prostředník mezi internetovým prohlížečem a testovanou aplikací. Druhou část *Selenium RC* tvoří *RC Client*, což je knihovna poskytující rozhraní mezi použitým programovacím jazykem (ve kterém jsou s použitím knihovny testy napsány) a *RC Serverem*. Zjednodušené schéma je na obrázku 3.2.



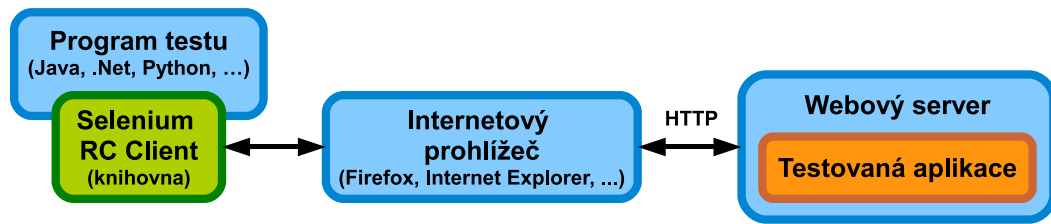
Obrázek 3.2: *Selenium RC* – zjednodušené schéma

Hlavní výhodou pro zkušené programátory je možnost psát scénáře testů přímo v oblíbeném programovacím jazyce (což je obvykle rychlejší než záznam a editace kroků v *Selenium IDE*). Díky tomu lze testy obohatit o libovolnou logiku realizovanou standardními konstrukcemi daného programovacího jazyka včetně řídicích struktur nebo o použití dalších knihoven.

Selenium RC je na oficiálních stránkách označeno jako „deprecated“ (zastaralé) a ačkoli je do jisté míry stále podporováno, tak není příliš vhodné jej používat. Důvodem je zejména horší kompatibilita s novějšími verzemi internetových prohlížečů a nestabilní chování testů.

3.4.3 Selenium WebDriver

Selenium WebDriver je plnohodnotnou náhradou za starší *Selenium RC*. Nástroj je díky jednotnému rozhraní *WebDriver*, podporovaného internetovými prohlížeči, možné používat bez spuštění jakékoli aplikace serveru (viz obrázek 3.3). Využití pokročilejších funkcí (například spuštění testů na jiném stroji), ale běžící instanci serveru nadále vyžaduje. Použití *Selenium WebDriver* je na úrovni zdrojového kódu velmi podobné *Selenium RC*. Přináší ale rozšíření některých možností a celkově je uživatelsky přívětivější.



Obrázek 3.3: *Selenium WebDriver* – zjednodušené schéma

V některých zdrojích je jako ekvivalentní označení pro *Selenium WebDriver* uváděno *Selenium 2*. Přesněji je ale *Selenium 2* souhrnný název balíku, který zahrnuje starší *Selenium RC* (*Selenium 1*) i novější *Selenium WebDriver*. Takto budou pojmy chápány také ve zbytku práce.

Při psaní testů je nejprve nutné vytvořit instanci třídy implementující rozhraní *WebDriver*. Použitá třída určuje internetový prohlížeč, který bude použitý k vykonání testů. Dostupné jsou následující implementace:

- `FirefoxDriver`,
- `HtmlUnitDriver`,
- `InternetExplorerDriver`,
- `ChromeDriver`,
- `OperaDriver`,
- `SafariDriver`,
- `PhantomJSDriver`,
- `AndroidDriver`,
- `iOSDriver`.

Velmi zajímavá je například implementace *HtmlUnitDriver*, která je založena na prohlížeči *HtmlUnit*. Tento internetový prohlížeč je naprogramovaný v jazyce *Java* a nemá grafické uživatelské rozhraní. To umožňuje spuštění testů i na strojích bez grafického rozhraní. Výhodou *HtmlUnitDriver* je také rychlost vykonání testů. Naopak nedostatkem je horší podpora *JavaScriptu*.

Lokátory

Možnosti lokalizace elementů webové stránky při použití *Selenium WebDriver* jsou tyto:

- **id** – vybírá elementy webové stránky s daným *id*,
- **className** – vybírá elementy webové stránky s daným názvem *class*,
- **name** – vybírá elementy webové stránky s daným *name*,
- **tagName** – vybírá elementy webové stránky s daným označením elementu v objektovém modelu stránky,
- **cssSelector** – vybírá elementy webové stránky vyhovující danému lokátoru kaskádových stylů,
- **xpath** – vybírá elementy webové stránky vyhovující danému *XPath* výrazu,
- **linkText** – vybírá hypertextové odkazy tvořené daným textem,
- **partialLinkText** – vybírá hypertextové odkazy, jejichž část textu odpovídá danému řetězci,

Poznámka: Lokátory použitelné v rámci *Selenium WebDriver* částečně odpovídají lokátorům použitelným v *Selenium IDE*.

3.4.4 Selenium Grid

Selenium Grid je posledním zástupcem z množiny *Selenium* nástrojů. Používá se k paralelnímu běhu testů. Umožňuje také současné testování na několika různých platformách. Díky distribuci testů mezi více strojů se zkracuje doba potřebná k jejich vykonání.

3.4.5 Selenium 2 + JUnit

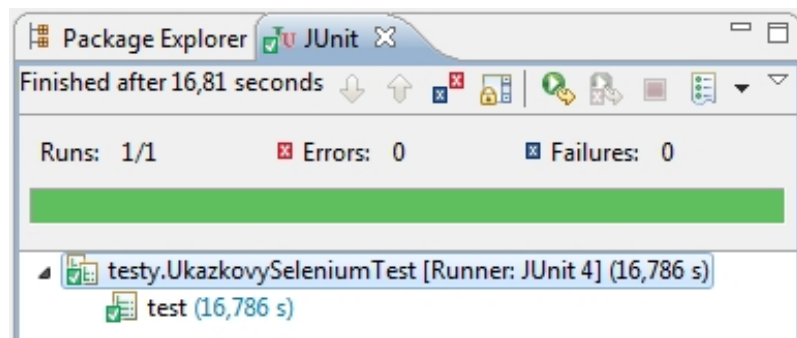
K použití *Selenium* testů v praxi je velmi výhodná integrace s dalšími testovacími frameworky. Na platformě *Java* je vhodným frameworkem například *JUnit*. Podoba výsledných testů (viz ukázka kódu 3.3) se od klasických *JUnit* testů příliš neliší.

V testových metodách se využívají standardní *assert* metody z knihovny *JUnit*, ale ve zbytku těla testu jsou používány metody a objekty *Selenium* knihovny doplněné o libovolný *Java* kód. V testech je rovněž možné využívat funkcionalitu poskytovanou dalšími knihovnami.

```
1 public class UkazkovySeleniumTest {
2     /**
3     * Test vyhledani slova "selenium" na Google.
4     */
5     @Test
6     public void test() {
7         WebDriver driver = new FirefoxDriver();
8         driver.get("https://www.google.cz/");
9
10        // zadani vstupu
11        WebElement vyhledavaciPole = driver.findElement(By
12            .name("q"));
13        vyhledavaciPole.sendKeys("selenium");
14
15        // vyckani na dynamicke prekresleni stranky
16        WebDriverWait cekani = new WebDriverWait(driver, 10);
17        cekani.until(ExpectedConditions
18            .titleContains("selenium"));
19
20        // ziskani titulku okna
21        String titulek = driver.getTitle();
22
23        driver.quit();
24
25        // porovnani titulku s ocekavanyms výsledkem
26        assertEquals("selenium – Hledat Googlem", titulek);
27    }
28 }
```

Ukázka kódu 3.3: Jednoduchý *Selenium* test

Spuštění testů je díky frameworku *JUnit* velmi jednoduché. Provádí se například přímo v použitém vývojovém prostředí (viz obrázek 3.4) nebo lze testy spouštět nástroji *Maven* [10] či *Ant* [11].



Obrázek 3.4: Vývojové prostředí *Eclipse* – spuštění *JUnit* testů

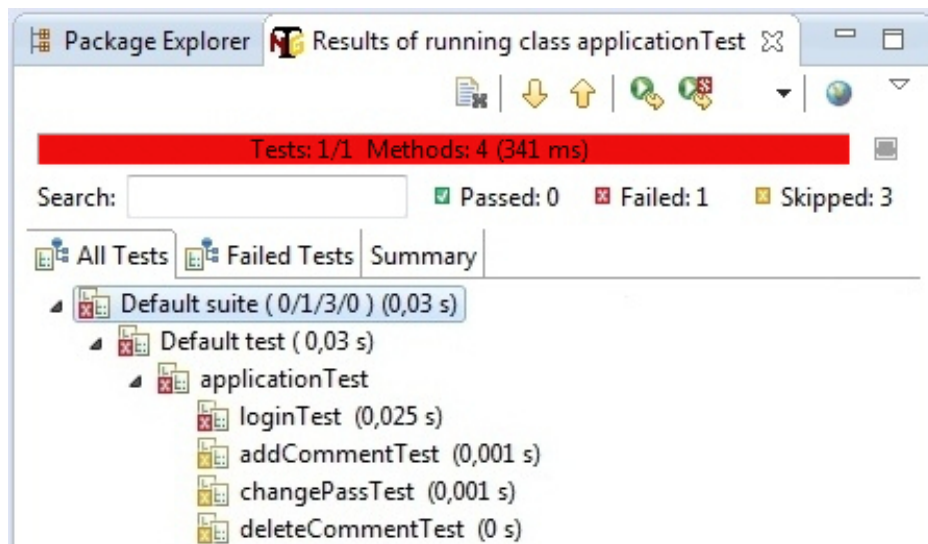
3.4.6 Selenium 2 + TestNG

Integrace s testovacím frameworkem *TestNG* má stejné přínosy jako integrace s *JUnit*. Nabízí ale navíc definice závislostí mezi testy. To je výhodné zejména při testování webových aplikací většího rozsahu. Pomocí závislostí lze eliminovat duplicitní kód testů a usnadnit lokalizaci chyb na základě výsledků testů.

Typickým příkladem je testování webové aplikace, která má určitou funkcionální přístupnou pouze pro registrované uživatele. Před každým testem této „skryté“ funkcionality je tedy nutné vykonat přihlášení uživatele a po skončení testu zase odhlášení. Samotné přihlášení uživatele je navíc kontrolováno samostatným testem. Duplicitní kód provádějící přihlášení lze eliminovat vytvořením opakovaně volané metody. Pokud je ale v aplikaci chyba v přihlášení, dojde k selhání velkého počtu testů a identifikace chyby je komplikovaná. S použitím závislostí je možné jednoduše zajistit, že všechny testy funkcionality dostupné přihlášenému uživateli proběhnou až po testu přihlášení (tzn. v okamžiku, kdy je již uživatel přihlášen). V případě selhání testu přihlášení jsou všechny závislé testy přeskočeny (ve výstupu označeny jako „skipped“) a z výstupu spuštění testů je na první pohled identifikovatelná chyba v aplikaci (viz obrázek 3.5).

3.4.7 Selenium a návrhové vzory

Přirozenou součástí životního cyklu každé webové aplikace je její postupný vývoj. Dochází k opravám chyb, rozšíření funkcionality a k dalším změnám. *Selenium* testy jsou však velmi závislé na uživatelském rozhraní aplikace. Každá menší změna tak může významně ovlivnit funkčnost testů. Aby byly vytvořené testy udržitelné,



Obrázek 3.5: Vývojové prostředí *Eclipse* – spuštění *TestNG* testů

je při jejich psaní nezbytné dodržovat některé konvence a případně se inspirovat *návrhovými vzory*.

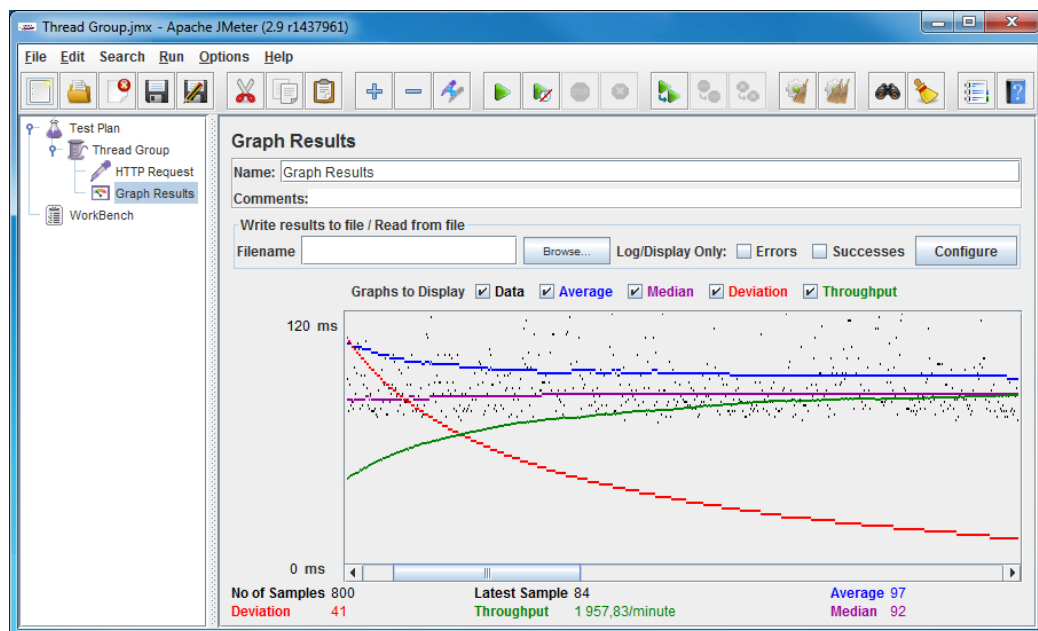
Zajímavý a často používaný *návrhový vzor* má název *PageObjects*. Tento *návrhový vzor* vychází ze skutečnosti, že mnohem častěji dochází ke změně uživatelského rozhraní aniž by se výrazně změnila aplikací poskytovaná funkcionální. Slovní formulace scénářů testů se v těchto případech nemění (provádí se stejné akce), ale kód *Selenium* testů je přesto potřeba upravit (úprava lokalizace elementů stránky). Myšlenka *návrhového vzoru* proto spočívá v oddělení vzhledu stránky (lokalizace elementů) od její funkcionality. Implementace se provádí vytvořením tříd reprezentujících jednotlivé stránky webové aplikace. Každá z těchto tříd obsahuje metody odpovídající funkcím dané stránky. Lokalizace elementů stránky se tedy vyskytují pouze v těchto metodách a zde také probíhá jejich úprava v případě změny testované aplikace. Kód testů je realizován vytvářením instancí tříd ekvivalentních stránkám aplikace (*PageObjects*) a voláním jejich metod. Při změně uživatelského rozhraní aplikace zůstává kód testů neměnný.

3.5 JMeter

JMeter [12] je dalším nástrojem (vytvořeným v programovacím jazyce *Java*) zaměřeným na testování webových aplikací. Tentokrát se ale primárně jedná o jejich zátěžové testy. Během několikaletého vývoje (přibližně od roku 2001) získal

JMeter řadu dalších funkcí. Jedná se tak o poměrně komplexní nástroj využitelný k mnoha účelům. Kromě testů webových aplikací dokáže ověřovat webové služby, síťovou infrastrukturu, úložiště, emailové servery, databáze a další. Mimo sledování výkonnostních parametrů zvládá i kontrolu správnosti odpovědí. Typickým použitím je u webové aplikace předem vyzkoušet chování při očekávaném zatížení nebo naopak simulovat její extrémní zatížení (tzv. *stresové testy*). Program sice zastupuje běžné uživatele a s webovým serverem komunikuje jako internetový prohlížeč, odpovědi serveru se ale nesnaží interpretovat. Neprovádí se tedy vykonání *JavaScriptu*.

K vytváření testů je vhodné využít grafické uživatelské rozhraní (viz obrázek 3.6). Zajímavou funkcí je sestavení testů prostřednictvím záznamu. K tomu je nutné program nakonfigurovat jako *proxy server* a nastavit jej v internetovém prohlížeči. Následně se dotazy na servery prováděné prohlížečem ukládají jako součást testového plánu. Ten lze ale vytvořit i zcela manuálně. Testy se sestavují přidáváním předdefinovaných elementů v levé části okna a jejich případnou konfigurací. Výsledný test je možné ihned spustit a sledovat průběžné výsledky v okně aplikace. Testy se ukládají ve formátu *XML*. Ke spouštění uložených testů není nezbytné využívat grafické rozhraní aplikace. Podporováno je spouštění prostřednictvím příkazové řádky bez *GUI*.



Obrázek 3.6: *JMeter* – grafické uživatelské rozhraní

Pro simulaci větší zátěže může být limitující výkon stroje, na kterém je *JMeter* spuštěn. Větší zátěž lze zajistit použitím více strojů, které spolu komunikují.

3.5.1 Základní elementy testového plánu

V této podkapitole jsou stručně popsány základní elementy testového plánu použité v ukázce na obrázku 3.6.

ThreadGroup

Tento element je základem každého testového plánu. Ostatní vkládané elementy tvoří jeho potomky. Základní konfigurační parametry jsou:

- **Number of threads** – počet spouštěných vláken (počet uživatelů),
- **Ramp-up period** – určuje prodlevu spouštění vláken,
- **Loop count** – počet opakování každého vlákna.

HTTP Request

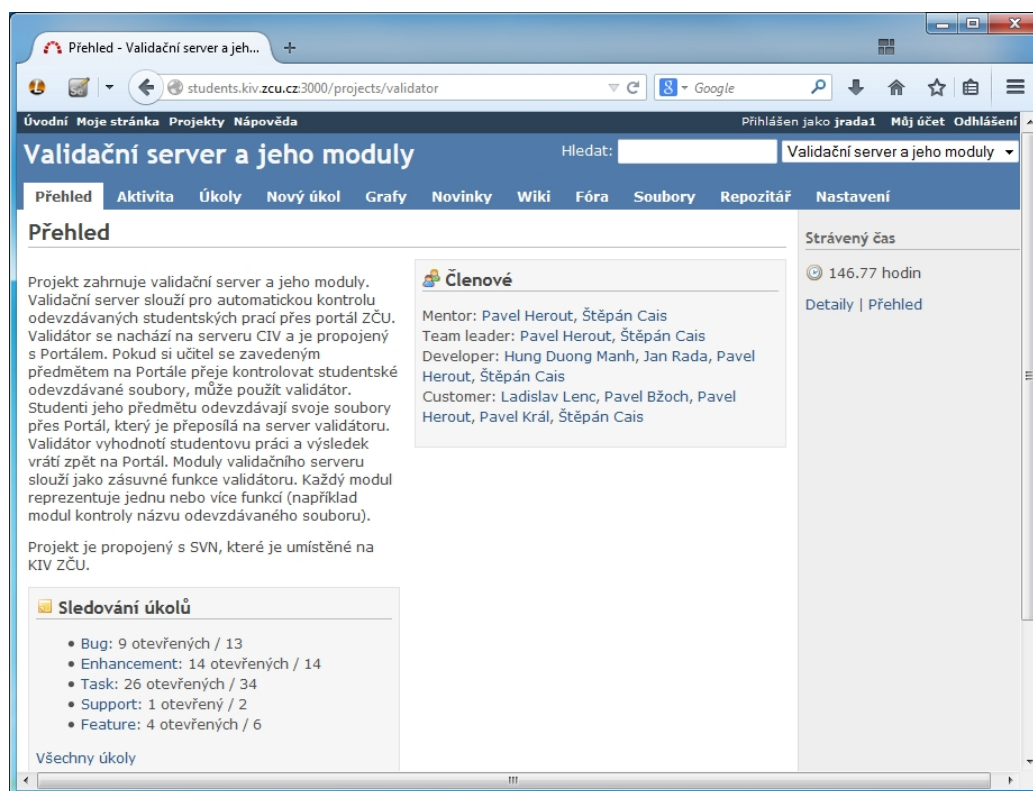
HTTP Request je element spadající do kategorie *Samplers*. Elementy této kategorie zajišťují odesílání požadavků. V konfigurační části *HTTP Request* se nastavuje adresa serveru, port, použitá metoda (*GET/POST*) a další parametry spojené s *HTTP* dotazem.

Graph Result

Jedná se o element z kategorie *Listeners* (posluchači). Posluchači provádějí měření a zobrazují nebo vypisují výsledky testu. Element *Graph Result* v podobě grafu vykresluje doby odezvy, propustnost, ...

3.6 Redmine

Webová aplikace *Redmine* [13] slouží ke kompletní správě a řízení softwarových projektů. Ukázka webového rozhraní je na obrázku 3.7. Základní funkcí je vytváření projektů, přiřazování pracovníků k projektům a definice jejich rolí (projektový manažer, vývojář, tester, ...). V rámci projektu se provádí vytváření „úkolů“ (tzv. *issues*), které se postupně zpřesňují a zařazují do plánů. Každý úkol je spjat se sadou metadat (název, popis, časový odhad, stav, priorita, mezní termín dokončení, ...). Úkoly mohou představovat novou funkcionalitu, ale slouží i k hlášení chyb nalezených v aplikaci. Správa úkolů přináší cenné informace o pracovním vytížení pracovníků a díky různým statistikám zpětnou vazbu pro plánování. Úkoly je také možné prostřednictvím doplňků spravovat přímo ve vývojovém prostředí – programátor tak má neustále k dispozici rozpis přiřazené práce a úkoly může označovat jako rozpracované nebo dokončené bez nutnosti otevírat internetový prohlížeč.



Obrázek 3.7: *Redmine* – webové rozhraní

Redmine dále poskytuje *wiki*, což je prostředek k vytváření hypertextových dokumentů. Obsah se tvoří pomocí jednoduchého značkovacího jazyka přímo v okně

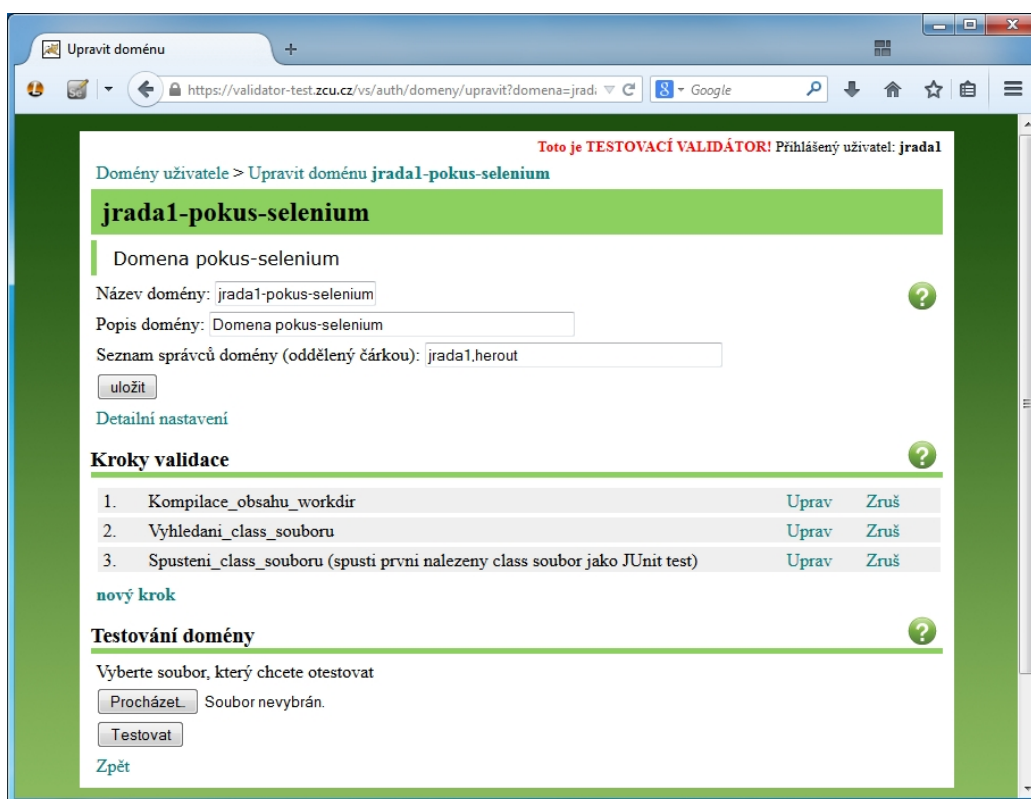
prohlížeče. Jedná se o rychlý způsob sdílení informací v rámci vývojového týmu nebo i směrem k uživatelům. Ke komunikaci lze také využít diskusní fórum. V neposlední řadě *Redmine* nabízí sdílení souborů nebo prostředí k procházení obsahu *repozitáře*.

4 Validátor

Validátor je aplikace vyvíjená na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Slouží jako prostředek pro automatickou validaci studenty odevzdávaných úloh (zejména programů) z různých předmětů. Odevzdávání úloh probíhá přes univerzitní *Portál* [14] (přesněji přes k tomu určený *portlet* zobrazovaný na stránce daného předmětu), který úlohu předá *Validátoru* a následně zobrazí výsledky validace.

4.1 Validáční doména

Příprava automatické validace spočívá ve vytvoření tzv. *validační domény*. Ta se skládá z posloupností kroků, jejichž cílem je ověřit funkčnost odevzdaného programu. Ukázka jednoduché validační domény složené z trojice kroků je na obrázku 4.1. Každý krok obsahuje podmínku (za jakých okolností má být vykonán) a prováděnou akci.



Obrázek 4.1: Testovací *Validátor* – webové rozhraní

Dostupné podmínky jsou tyto:

- **vždy**,
- **nikdy**,
- **validace ještě neobsahuje chybu**,
- **validace již obsahuje chybu**,
- **vlastní skript**.

Volba „vlastní skript“ umožňuje napsat vlastní *JavaScriptový* kód. V rámci kódu je možné používat *JavaScriptové* proměnné definované v rámci akce jiných kroků stejné *validační domény*. Akce mohou být následující:

- **nic** – žádná akce není provedena,
- **skok na krok** – dovoluje výběr jiného kroku domény, na který má být proveden přechod,
- **vložit do výstupu validace** – používá se k textovému výpisu do výstupu validace,
- **konec validace** – ukončí validaci,
- **vlastní akce** – výběr z předdefinovaných akcí tvořených moduly *Validátoru*,
- **vlastní skript** – použití vlastního kódu v *JavaScriptu*.

Mezi akcemi dostupnými při volbě „vlastní akce“ jsou akce realizující základní operace se soubory nebo archivy (vyhledání, odstranění, rozbalení archivu, ...). Dále jsou k dispozici akce pro práci s programy (překlad programu, spuštění programu, spuštění *JUnit testů*, ...). Existuje zde ale i množství dalších pokročilých akcí (např. porovnání dvojice obrázků). Nabídku akcí je možné dále rozšiřovat vytvářením modulů *Validátoru*.

4.2 Princip validace

Při předání souboru k validaci je na serveru vytvořen dočasný pracovní adresář (tzv. *workdir*) s unikátním názvem, do kterého je uložen odevzdaný soubor. V tomto

adresáři probíhají veškeré operace (např. rozbalení archivu, kontrola názvů, ...) vyvolané akcemi jednotlivých validačních kroků. Po skončení validace je tento adresář v závislosti na nastavení domény odstraněn.

Na serveru *Validátoru* se také pro každou *validační doménu* vytvoří adresář, který obsahuje základní konfigurační soubory domény. Do tohoto adresáře je možné předem nahrát další soubory (zpravidla programy nebo knihovny), které lze poté v průběhu validace používat.

5 Výběr nástrojů pro realizaci projektů

Na základě průzkumu zadaných nástrojů pro podporu testování bylo zapotřebí stanovit, které z nich jsou vhodné pro přípravu studentských projektů. Hlavním kritériem výběru byly možnosti uložení výstupu, který by dokládal použití nástroje předem definovaným způsobem. U nástrojů sloužících přímo k testování byl důležitým parametrem také způsob spuštění vytvořených testů za účelem ověření jejich funkčnosti. Pro přípravu úloh by byly ideální nástroje, jejichž výstup by bylo možné ověřit aplikací *Validátor* pouze s využitím aktuálně podporovaných validačních akcí (tzn. bez nutnosti vytvářet rozšiřující moduly). Dalším kritériem bylo množství a dostupnost prostředků (zejména hardwarových nebo softwarových), které by bylo nezbytné zajistit, jako podklad pro plnění konkrétní úlohy. Při rozhodování byla v úvahu vzata také míra budoucí využitelnosti znalostí a zkušeností získaných během realizace projektu spočívajícího v praktickém použití daného nástroje. Zhodnocení vlastností jednotlivých zkoumaných nástrojů s ohledem na přípravu projektů je uvedeno v navazujících podkapitolách.

5.1 Selenium

Z rodiny nástrojů *Selenium* má smysl se zabývat pouze nástroji *Selenium IDE* a *Selenium WebDriver*. Používání *Selenium RC* je na ústupu (nahrazuje jej *Selenium WebDriver*) a *Selenium Grid* představuje pokročilejší způsob *Selenium* testování, který není z hlediska výuky podstatný.

Doplňek internetového prohlížeče *Firefox*, *Selenium IDE*, je pro výukové účely velmi vhodný. Jeho instalace navíc zabere jen několik málo minut. Během chvíle tak lze skrze intuitivní grafické uživatelské rozhraní vytvářet první jednoduché testy a sledovat jejich průběh. Studentský projekt zaměřený na seznámení se s tímto produktem by vyžadoval nalezení vhodné (zdrojový kód stránek umožňující použití pouze základních typů lokátorů, výskyt chyb, ...) testované webové aplikace nebo její naprogramování. Validace výstupu úlohy, tedy testů vytvořených podle definovaného zadání, by ale v tomto případě byla komplikovanější. *Validátor* běží na serveru bez grafického prostředí a navíc nedisponuje funkcemi, které by spuštění *Selenium* testů ve formátu *HTML* umožnily. Řešením by mohlo být využití exportu do podoby *JUnit* testů využívajících *Selenium WebDriver*. Tato funkce je v *Selenium IDE* snadno

dostupná prostřednictvím menu. Spuštění *JUnit* testů v prostředí *Validátoru* je navíc plně podporováno. Problém s absencí grafického prostředí na serveru by bylo možné vyřešit náhradou výchozí použité implementace *WebDriver* v exportovaných testech (*FirefoxDriver*) za *HtmlUnitDriver*. Kontrolu odevzdaných testů (založenou nejspíše na výskytu konkrétních lokátorů v jejich zdrojovém kódu a na předem známém a očekávaném počtu testů končících chybou) by bylo možné provádět samostatným *Java* programem. Ten by se spouštěl jako jeden z kroků příslušné *validační domény* na *Validátoru* a podle jeho výstupu by bylo rozhodnuto o celkovém výsledku validace.

Další projekt, tentokrát využívající *Selenium WebDriver*, by mohl být přínosný zejména po praktickém seznámení s nástrojem *Selenium IDE*. Poskytoval by srovnání tvorby testů psaných přímo v programovacím jazyce (pravděpodobně *Java*) s jejich tvorbou prostřednictvím záznamu a editace kroků v *Selenium IDE*. Vhodné by bylo použití *Selenium WebDriver* společně s frameworkem *JUnit*, což by usnadnilo spuštění testů na *Validátoru*. Jako podklad pro testování by opět byla nezbytná vhodná webová aplikace. Psaní testů přímo v podobě zdrojového kódu dává jejich tvůrci větší volnost a nelze tak spoléhat na určitou strukturu kódu jako v případě testů exportovaných ze *Selenium IDE*. To by mohla být z hlediska automatické validace komplikace. Při přípravě úlohy by proto bylo nutné v jejím zadání striktně definovat požadavky na zdrojový kód testů nebo důkladně analyzovat problémové situace (například použití konstant pro uložení hodnot lokátorů) a vhodným způsobem je ošetřit. Způsob validace by mohl být shodný s validací u projektu zaměřeného na použití *Selenium IDE* (tzn. založený na spuštění samostatného kontrolního programu).

5.2 JMeter

Nutnou podmínkou realizace projektu založeného na použití programu *JMeter* by bylo zajištění serveru, který by byl určen výhradně pro potřeby testování a jehož zvýšené zatěžování by neohrozilo dostupnost žádných důležitých služeb. Použití některého z veřejně dostupných webových serverů je zcela vyloučeno, jelikož provádění zátěžových testů by mohlo být považováno za hackerský útok. Je potřeba zohlednit, že i test generující obvyklý počet požadavků, který je spuštěný několika studenty naráz, může ve výsledku znamenat enormní zatížení cílového serveru. V důsledku kolísajícího zatížení serveru (daného počtem současně testujících studentů) mohou být také při opakovaném spuštění testu jeho výsledky velmi odlišné.

XML formát uložených testů *JMeteru* by mohl být vhodný pro jejich statickou kontrolu. V tomto případě by ale byla problematická formulace zadání projektu. Pokud by měl vytvořený test odpovídat určitému vzoru použitému při validaci, muselo by být zadání velmi podrobné (podobné spíše návodu) a úloha by tak neměla velký smysl.

I když *JMeter* podporuje spuštění bez *GUI* a vykonání odevzdaných testů na *Validátoru* by díky tomu bylo reálné, tak dynamický způsob kontroly není v tomto případě vhodný. Kvůli okolnímu provozu se mohou výsledky totožného testu velmi lišit v závislosti na času jeho provedení. Výsledky tedy nelze porovnat s očekávanými hodnotami a jediným výstupem může být informace o tom, zda je test spustitelný. Limitující je v tomto případě i omezená doba, během které musí být validace provedena (zátěžové testy přitom běžně probíhají několik minut).

5.3 Redmine

Projekt sloužící k osvojení nástroje *Redmine* je při dodržení podmínky automatické validace téměř nerealizovatelný. I u velmi jednoduchého zadání – např. vložení chyb odhalených v rámci jiného studentského projektu do *Redmine*, nelze splnění úlohy kontrolovat strojově. Validaci by sice bylo možné založit na ověření počtu vložených chyb konkrétním uživatelem, ale důležitý je v tomto případě i vhodný popis nalezených chyb. Ten samozřejmě není jednoznačný a automatická kontrola je proto nevhodná. Výstupem *Redmine* navíc není žádný validovatelný soubor. Jediným proveditelným usnadněním kontroly by mohlo být použití vhodného rozšiřujícího modulu – např. modulu poskytujícího export *issues* z *Redmine* do podoby tabulky ve formátu *XLS* [15]. Z důvodu nejednoznačného popisu reportovaných chyb by ale i v tomto případě musela být finální kontrola projektu prováděna vyučujícím.

5.4 Shrnutí

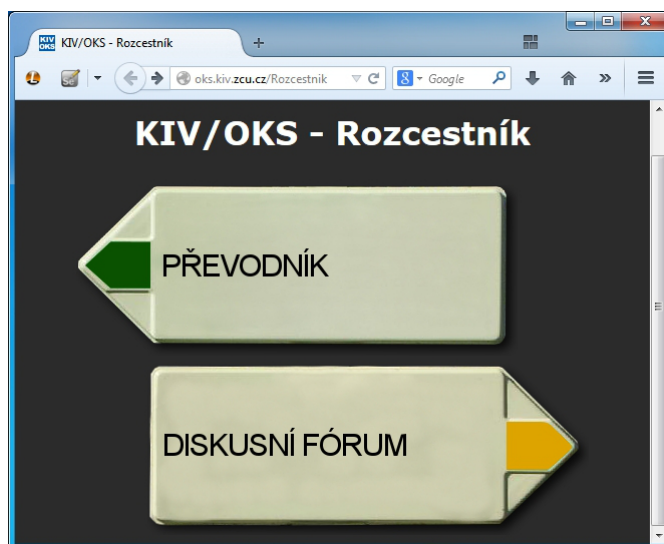
Na základě výše uvedených poznatků byl jako nejvhodnější produkt pro přípravu projektů vybrán framework *Selenium*. Snahou bylo vytvoření obou navrhovaných projektů (*Selenium IDE* i *Selenium WebDriver*). Zvoleným způsobem zajištění webové aplikace sloužící jako podklad pro testování bylo její naprogramování.

6 Webová aplikace pro testování

Při výběru prostředků k vytvoření testované webové aplikace jsem se rozhodoval podle osobních preferencí a nakonec zvolil platformu *Java EE*. Verze použitých technologií (*Servlety*, *JSP*) byly ovlivněny cílovým produkčním prostředím s webovým kontejnerem *Winstone*.

Požadavkem vyučujícího předmětu „KIV/OKS“ bylo vytvoření dvojice webových aplikací, z nichž jedna by sloužila k demonstraci některých postupů v rámci přednášek předmětu a druhá by byla podkladem pro plnění samostatných úloh. Cílem bylo, aby obě vytvářené aplikace byly jednoduché (rychlé pochopení, snadné pokrytí testy), ale zároveň umožnily ukázkou hlavních přínosů *Selenium* testování.

Aplikace byly původně vyvíjeny odděleně, ale později bylo kvůli nutnosti jejich současného nasazení na webový server nevyhnutelné sloučení do jednoho projektu. Sloučení spočívalo ve změně struktury zdrojových projektů, úpravě některých referencí a vytvoření hlavní stránky [19] (*Rozcestník* - viz obrázek 6.1) s odkazy na jednotlivé aplikace. Hlavním přínosem této změny je pohodlnější *deploy* obou aplikací naráz prostřednictvím jediného *WAR* souboru.



Obrázek 6.1: *Rozcestník* – stránka s odkazy na testované aplikace

Obě aplikace jsou vytvořeny v souladu s architekturou *MVC*, která zajišťuje oddělení aplikační logiky od prezentační vrstvy. Veškerá aplikační logika je proto řešena výhradně *servlety* a *JSP* soubory tvoří pouze prezentační vrstvu. Aplikace byly nejprve

vytvářeny bez úmyslně zanesených chyb a důkladně testovány s použitím *Selenium* testů (*Selenium WebDriver* + *JUnit*). Při testování byl využíván webový kontejner *Apache Tomcat v7.0* [16]. Několik úmyslných chyb bylo do aplikací přidáno až po jejich dokončení. Popis aplikací je uveden v následujících podkapitolách. Generovaný *HTML* kód (*HTML 5*) i vytvořené kaskádové styly (*CSS 3*) jsou validní podle odpovídajících specifikací definovaných konsorciem *W3C*. (Validita je ověřována pomocí oficiálních validátorů [17] a [18] v rámci zhotovených testů.)

Ačkoli jsou webové aplikace částečně tvořeny statickým obsahem (stránky *Úvod* a *Nápověda*), nebylo kvůli omezením kontejneru *Winstone* možné (v konfiguračním souboru *web.xml*) provést mapování příslušných adres přímo na odpovídající *JSP* soubory. Řešení si vyžádalo vytvoření jednoduchých *servletů*, na které jsou tyto adresy mapovány, a které provádějí pouze přechod na danou *JSP* stránku.

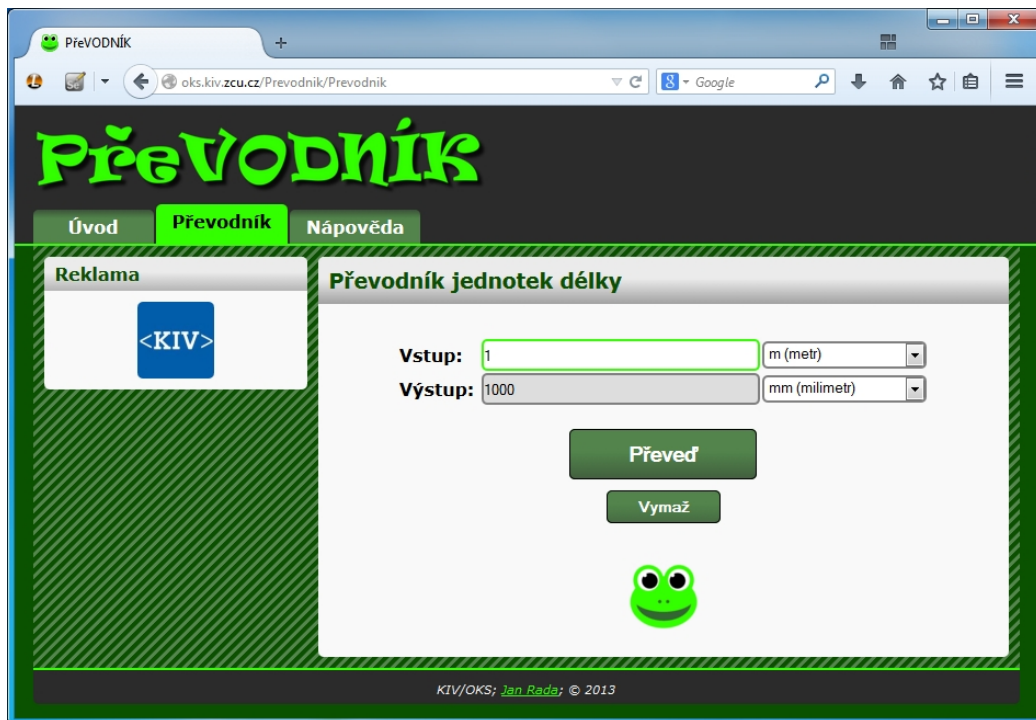
Kvůli jednoznačné identifikovatelnosti chyb byly k aplikacím zpětně vytvořeny dokumenty specifikace (viz příloha A a příloha B), které jednoznačně definují na ně kladené požadavky.

6.1 Převodník

Převodník [20] (viz obrázek 6.2) je aplikace používaná k ukázkám *Selenium* testování na přednáškách předmětu „KIV/OKS“. Jak již název napovídá, tak její funkcí jsou převody. Konkrétně se jedná o převody délkových jednotek.

Formulář sloužící k převodům je umístěn na stránce *Převodník*. Obsahuje jediné vstupní pole pro zadání číselné hodnoty. Dále jsou k dispozici dvě výběrová tlačítka (tzv. *comboboxy*) určující jednotku vstupu a požadovanou jednotku převedeného výstupu. Výsledek převodu je zobrazen v needitovatelném poli s popiskem „Výstup“ po stisku tlačítka „Převod“. K vymazání polí formuláře a nastavení výchozích jednotek je určeno tlačítko „Vymaž“.

Převodní jednotky dostupné v aplikaci je možné měnit prostřednictvím textového konfiguračního souboru. V tomto souboru jsou uvedeny zobrazované názvy jednotek a jim odpovídající převodní koeficienty. Při změně jednotek tak není nutné provádět překlad zdrojových kódů aplikace.



Obrázek 6.2: *Převodník* – webová aplikace pro nácvik testování

6.1.1 Úmyslné chyby

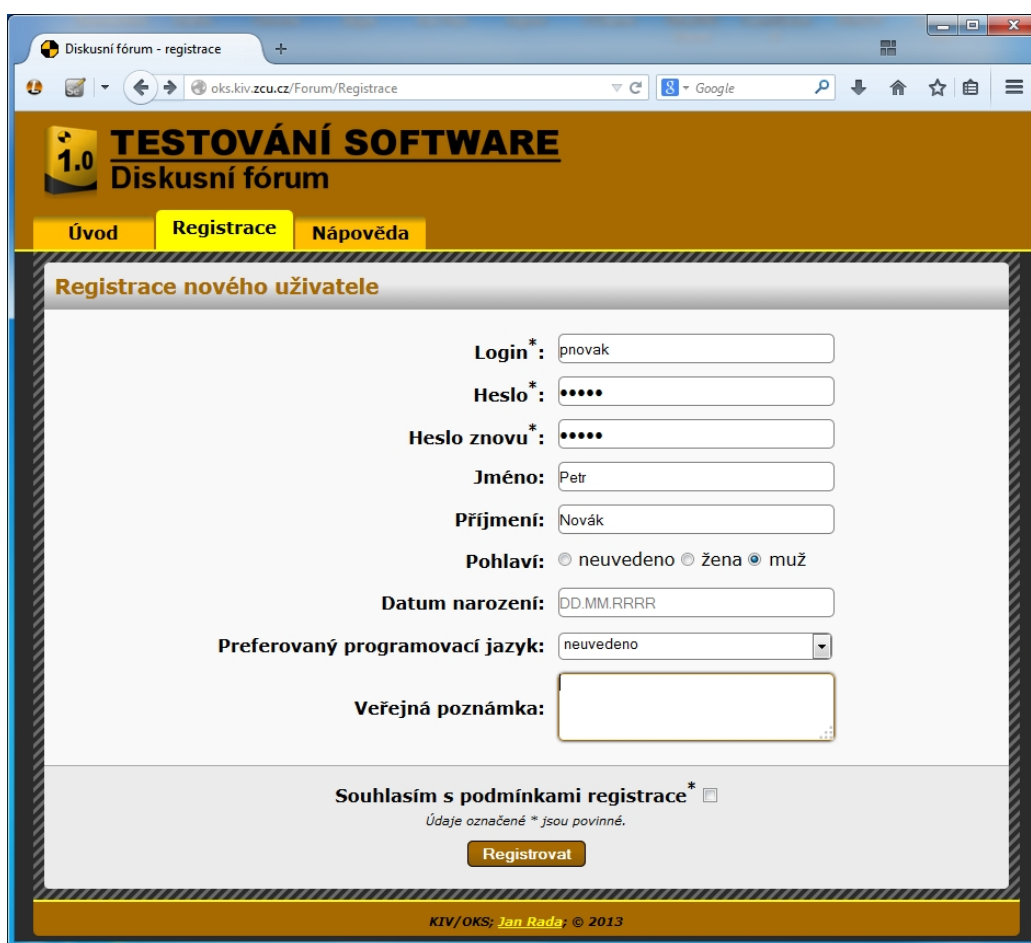
Chyby, které byly do aplikace *Převodník* zaneseny pro potřeby testování, jsou uvedeny v následujícím výčtu:

- odkaz na webové stránky Západočeské univerzity v Plzni (na stránce *Úvod*) směřuje na stránky Katedry informatiky a výpočetní techniky,
- tlačítko „Vymaž“ (na stránce *Převodník*) provádí pouze vymazání obsahu pole „Vstup“ (obsah pole „Výstup“ i zvolené jednotky zůstávají původní),
- chybný výsledek převodu v případě použití jednotky decimetr jako jednotky vstupu nebo výstupu (použije se převodní koeficient pro jednotku palec),
- při zadání záporné hodnoty do pole „Vstup“ a potvrzení převodu tlačítkem „Převeď“ je místo výpisu chybového hlášení převod vykonán a v poli „Výstup“ je zobrazena záporná hodnota.

Poznámka: Specifikace aplikace se nachází v příloze A.

6.2 Diskusní fórum

Aplikace *Diskusní fórum* [21] představuje smyšlené diskusní fórum, jehož jedinou funkcionalitou je „registrace“ nového uživatele. K tomu slouží registrační formulář na stránce *Registrace*. Vyplněné údaje o novém uživateli nejsou po odeslání formuláře nikam ukládány, ale dochází pouze k jejich zobrazení. Některé vstupy jsou během registračního procesu použity k získání jiných údajů, kterými jsou pak zastoupeny na stránce zobrazené po registraci (ze zadaného hesla je vytvořen *MD5* hash a zadané datum narození slouží k výpočtu věku uživatele).



The screenshot shows a web browser window with the URL `oks.kiv.zcu.cz/Forum/Registrace`. The page title is "TESTOVÁNÍ SOFTWARE 1.0 Diskusní fórum". The navigation menu includes "Úvod", "Registrace" (highlighted), and "Nápověda". The main content area is titled "Registrace nového uživatele" and contains a registration form with the following fields:

- Login ***:
- Heslo ***:
- Heslo znovu ***:
- Jméno**:
- Příjmení**:
- Pohlaví**: neuvedeno žena muž
- Datum narození**:
- Preferovaný programovací jazyk**:
- Veřejná poznámka**:

At the bottom of the form, there is a checkbox for "Souhlasím s podmínkami registrace*" and a "Registrovat" button. A small note below the checkbox states "Údaje označené * jsou povinné." The footer of the page reads "KIV/OKS; Jan Rada; © 2013".

Obrázek 6.3: *Diskusní fórum* – webová aplikace pro nácvik testování

6.2.1 Úmyslné chyby

Úmyslně vytvořené chyby v aplikaci *Diskusní fórum* jsou tyto:

- odkaz na popis předmětu „KIV/OKS“ (na stránce *Úvod*) směřuje na stránky Katedry informatiky a výpočetní techniky,
- při neúspěšném odeslání registračního formuláře na stránce *Registrace* (například z důvodu nevyplnění všech povinných polí) nedojde k předvyplnění vstupního pole s popiskem „Jméno“ původně zadanou hodnotou (obsah pole je vymazán),
- MD5 hash zobrazovaný po úspěšném odeslání formuláře (stránka *Registrace*) je chybný (na náhodné pozici obsahuje chybný znak),
- vypočítaný věk uživatele zobrazovaný (pouze při vyplnění pole „Datum narození“) po úspěšném odeslání registračního formuláře (stránka *Registrace*) je vypočítán jako rozdíl aktuálního a zadaného roku, tudíž může mít v některých případech chybnou hodnotu (např. uživateli narozenému 1. 10. 2000 je k 1. 5. 2014 13 let, ale vypočítaný věk je 14)
- obsah pole „Veřejná poznámka“ (stránka *Registrace*) není po úspěšném odeslání formuláře „uložen“ (tzn. není vypsán v odpovídajícím poli na stránce zobrazené po odeslání formuláře)

Poznámka: Specifikace aplikace se nachází v příloze B.

7 Projekt „oks-09“

Projekt s označením „oks-09“ (devátý projekt v rámci předmětu „KIV/OKS“) bude zaměřen na použití nástroje *Selenium IDE*. Jako testovaný produkt bude použita webová aplikace s názvem *Diskusní fórum*.

7.1 Cíl projektu

Cílem projektu je seznámit jeho řešitele (studenta) se základy použití nástroje *Selenium IDE*. Jedná se zejména o vytvoření testu s využitím záznamu prováděných akcí do podoby kroků testu, editaci těchto kroků, manuální vytváření kroků testu, nebo vytvoření *test suite*. Student by se měl také naučit používat lokátory elementů stránky a osvojit si základní *Selenium* příkazy.

7.2 Analýza

Předpokládaný způsob validace zpracovaného projektu je s využitím *Validátoru*. Z tohoto důvodu musí být výsledné odevzdávané testy v podobě, kterou lze na *Validátoru* snadno kontrolovat. Prvním krokem tedy bude ověřit spuštění jednoduchého (manuálně vytvořeného) *Selenium* testu využívajícího *Selenium WebDriver* (konkrétně implementaci *HtmlUnitDriver*, která nevyžaduje grafické prostředí) a framework *JUnit*. V případě úspěchu bude vhodné pokus opakovat i pro test získaný exportem z nástroje *Selenium IDE*. Na základě výsledku pokusu bude možné stanovit, zda kontrolu bude reálně provádět spuštěním testů nebo zda bude proveditelná pouze statická kontrola (případně kombinace obou způsobů).

Jelikož zadání projektu bude přesně formulováno, bude předem známo, které elementy webové stránky musí být ve vytvořených testech použity (například uložení hodnoty vstupního pole „Jméno“ není možné testovat bez zadání vstupu do tohoto pole, stisku tlačítka „Registrovat“ a přečtení uložené hodnoty z odpovídajícího pole stránky zobrazené po registraci). Každý element stránky lze lokalizovat několika typy lokátorů. Za předpokladu, že by všechny použitelné lokátory byly jednoznačné, by bylo možné prostřednictvím statické kontroly kódu testů ověřit, zda se všechny očekávané elementy stránky v testech opravdu používají – tzn. provedla

by se statická kontrola pokrytí testy. Jediným omezením je v tomto případě skutečnost, že všechny typy lokátorů nejsou jednoznačné (stejný element stránky lze např. použitím *XPath* výrazu adresovat několika způsoby). Kvůli tomu by musela být množina použitelných typů lokátoru omezena v zadání projektu pouze na typy, u kterých je zaručena jednoznačnost lokátoru. Také by muselo být zajištěno, že prostřednictvím této podmnožiny typů lokátorů bude možné adresovat všechny důležité elementy stránky.

Pokud bude pokus ověřující spuštění *Selenium* testů na *Validátoru* úspěšný, bude možné spuštění testů použít i při kontrole. Počet chyb v aplikaci je předem známý a při vhodném rozdělení testů do metod (v *Selenium IDE* do testových případů) půjde ověřit počet neúspěšných testů. Při použití tohoto způsobu kontroly ovšem hrozí, že počet neúspěšných testů bude sice souhlasit, ale jejich obsah nebude odpovídat testům, které měly chybou opravdu skončit (cíleně vytvořené testy končící chybou). Dynamickou kontrolu by tedy bylo vhodné doplnit statickou kontrolou, která by po spuštění testů ověřila, zda se v testech končících chybou opravdu používají očekávané elementy stránky, které s chybou v aplikaci souvisí.

Při odevzdávání zpracovaného projektu je potřeba zajistit, aby zpětná vazba informující o výsledku kontroly byla dostatečně rychlá. I v případě spuštění *Selenium* testů prostřednictvím *HtmlUnitDriveru* by při jejich větším počtu byla odezva příliš pomalá. Hrozilo by tak, že testy nebudou vykonány v limitu definovaném v nastavení validace. Z tohoto pohledu by bylo nejlepším řešením provádět kontrolu odevzdaných testů jen staticky a dynamickou kontrolu spustit jen u testů končících chybou (tzn. maximálně 5 testů). Automatické rozpoznání chybových testů na základě statické kontroly by ale nebylo příliš spolehlivé, takže oddělení těchto testů od ostatních by bylo nutné zajistit v zadání (například požadavkem na umístění chybových testů do jiného adresáře).

7.3 Řešení

7.3.1 Ověření možností Validátoru

Kvůli ověření možnosti spuštění *Selenium* testů v prostředí *Validátoru* bylo nejprve nutné vytvořit jednoduchou *validační doménu*, která obsahuje kroky provádějící překlad a spuštění *JUnit* testu. Aby byl *JUnit* test využívající knihovnu *Selenium* pře-

ložitelný a spustitelný v prostředí *Validátoru*, bylo zapotřebí do příslušného adresáře *Validátoru* tuto knihovnu nahrát a vhodně upravit konfiguraci (nastavení oprávnění, úprava *classpath*). Během prvních pokusů spuštění testu končilo chybovým výpisem. Z jeho obsahu byl identifikován problém v oblasti nastavení práv a později s pomocí vývojářů *Validátoru* identifikována a opravena chyba v aplikaci. Následně již bylo spuštění *Selenium* testů funkční, což umožnilo uvažovat spuštění testů jako jednu z možností kontroly.

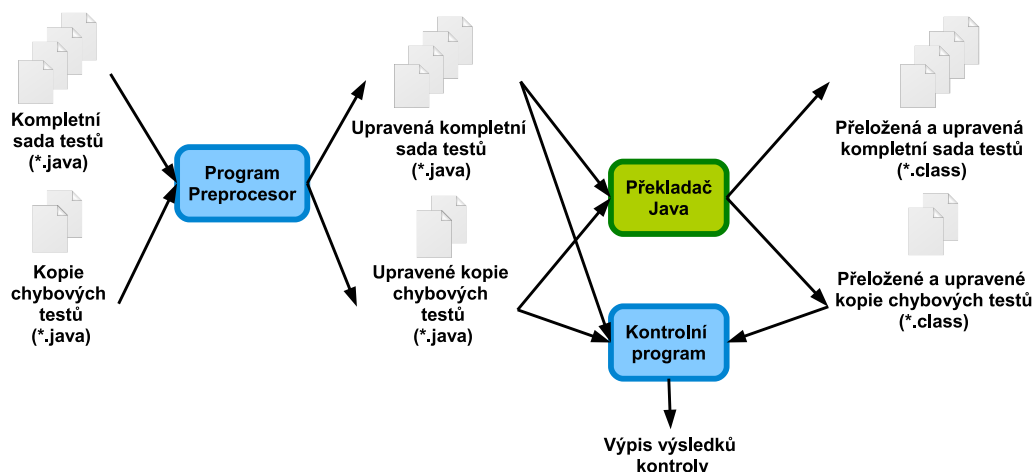
Kód testů získaný exportem z nástroje *Selenium IDE* byl na *Validátoru* přeložitelný a spustitelný pouze po provedení několika úprav. Tyto úpravy spočívaly zejména v náhradě použité implementace *WebDriver* (náhrada *FirefoxDriver* za *HtmlUnitDriver*), odpovídající úpravě importů a změně balíku (*package*). V případě použití dynamické kontroly, by tedy bylo zapotřebí zajistit provedení těchto úprav. Provedení úprav by mohlo být požadováno v zadání projektu, ale kvůli usnadnění a eliminaci chyb způsobených manuální úpravou je vhodnějším způsobem programová realizace úprav až v prostředí *Validátoru*.

7.3.2 Výsledný způsob kontroly

Po ověření funkčnosti spuštění *Selenium* testů na *Validátoru* jsem sestavil finální postup kontroly. Podle něj byl následně vytvořen kontrolní program. Kontrola vychází z odevzdání archivu, který bude v jednom adresáři obsahovat soubory se zdrojovým kódem všech testů (soubory získané exportem ze *Selenium IDE*) a soubor *test suite*, v dalším adresáři budou umístěny kopie pouze těch souborů s testy, jejichž spuštění končí chybou. Chybové testy budou při kontrole spouštěny a je tedy zapotřebí provést úpravy jejich zdrojového kódu popsané v předchozí podkapitole. Ostatní testy spouštěny nebudou, ale jako doplněk statické kontroly jejich obsahu je vhodné provést alespoň jejich překlad. Aby mohl být překlad proveden, je nutné i v souborech těchto testů provést minimálně změnu *package*. Celý postup kontroly (znázorněný na obrázku 7.1) bude následující:

1. V souborech odevzdaných testů provést nezbytné úpravy zdrojového kódu (zajištěno programem).
2. Provést překlad všech (v bodě 1 upravených) testů (tzn. kontrola, zda obsah souborů odpovídá *Java* kódu).

3. Provést statický test pokrytí testy spočívající v průchodu obsahu souborů z kompletní sady odevzdaných testů a ověření výskytu všech očekávaných lokátorů elementů stránky.
4. Provést dynamický test chybových testů spočívající ve spuštění testů a porovnání počtu testů končících chybou s očekávaným počtem a dále v porovnání lokátorů v testech končících chybou s očekávanými lokátory v těchto testech.



Obrázek 7.1: „oks-09“ – schéma kontroly

7.4 Kontrolní program

Při kontrole je využíván program, který se skládá ze dvou částí (viz obrázek 7.1). První část (*Preprocessor*) provádí úpravy zdrojového kódu odevzdaných testů. Tyto úpravy musí být provedeny před překladem testů (překlad testů bez úprav nelze provést kvůli neodpovídajícímu *package*). Teprve druhá část programu vykonává samotnou kontrolu. Jako vstup používá upravené zdrojové kódy vytvořené *Preprocesorem* a zároveň spouští již přeložené chybové testy.

Očekávané elementy webové stránky, resp. jejich lokátory, jsou v programu zapísány staticky. Z hlediska návrhu programu by zajisté bylo vhodnější očekávané lokátory načítat dynamicky – například z textového souboru. Tento způsob ale není za daných okolností vhodný. Kontrolní program bude používán i k lokální kontrole výsledků zpracovaného projektu studenty (bude jim dodán ve zkompilevané podobě spolu se zadáním projektu) a není proto žádoucí, aby očekávané lokátory byly na první pohled zřejmé (viditelné v textovém konfiguračním souboru).

Názvy a cestu k adresářům s odevzdanými testy je možné zadat jako vstupní argumenty při spuštění programu. Pokud nejsou žádné argumenty zadány, program použije výchozí hodnoty.

7.4.1 Popis tříd programu

TypLokatoru.java

Tato třída je výčtovým typem, který reprezentuje jednotlivé použitelné typy lokátorů, s jejichž pomocí lze adresovat elementy webové stránky. Dostupné jsou tyto typy:

- ID,
- NAME,
- LINK_TEXT,
- OTHER.

Jedná se pouze o lokátory, u kterých je zaručena jejich jednoznačnost. Jedinou výjimkou je typ „OTHER“, který je možné použít jako náhradu za ostatní (samostatně nedefinované) typy. V současné podobě kontrolního programu není typ „OTHER“ využíván. Jeho použití vyžaduje zvýšenou pozornost, jelikož pro úspěch kontroly musí být zaručeno, že hodnota lokátoru tohoto typu bude jednoznačná.

ElementStranky.java

ElementStranky.java je předpisem k vytvoření instance odpovídající konkrétnímu elementu webové stránky. Atributem je název popisující tento element a také mapa, jejíž klíče tvoří typy lokátorů použitelné k lokalizaci daného elementu (datový typ „TypLokatoru“) a hodnotami mapy jsou řetězce lokátorů odpovídající typu klíče. Ve třídě jsou také připraveny statické metody, které vytvářejí instance (datového typu „ElementStranky“) klíčových elementů webové aplikace *Diskusní fórum*.

Metoda.java

Tato třída reprezentuje testovou metodu načtenou ze zdrojového kódu testové třídy. Atributy jsou: název třídy ve které je metoda umístěna, název balíku této třídy, název metody a kód těla metody. Instance třídy *Metoda* se během kontroly používají při ověřování výskytu očekávaných lokátorů v kódu chybových testů.

TestovyPripad.java

TestovyPripad je objekt reprezentující určitou funkcionalitu aplikace, jejíž testování je požadováno. Je tvořen popisem a mapou. Klíče mapy tvoří elementy stránky (datový typ „ElementStranky“), které musí být k testu dané funkcionality použity a hodnotami jsou počty povinných výskytů těchto elementů (přesněji jejich lokátorů) v těle metody odpovídajícího testu. Posledním atributem je množina, jejíž prvky odpovídají testovým metodám (datový typ „Metoda“), které danému testovému případu vyhovují. Tento atribut je využíván až v okamžiku provádění kontroly, při které jsou do množiny reference na vyhovující metody vkládány.

Pomocne.java

Třída *Pomocne.java* obsahuje pomocné statické metody sloužící k čtení a zápisu do souborů. Tyto metody jsou opakovaně volány z různých tříd programu.

Preprocesor.java

V této třídě je obsažen kód provádějící úpravy zdrojového kódu odevzdaných testů do podoby přeložitelné *Java* překladačem. U kompletní sady odevzdaných testů (které jsou jen překládány a nejsou spuštěny) se provádí pouze změna balíku (*package*). U chybových testů (při kontrole se provádí i jejich spuštění) dochází k dalším úpravám kódu – například záměna použité implementace *WebDriveru* (náhrada *FirefoxDriver* za *HtmlUnitDriver*).

Validator.java

Validator.java je hlavní třída kontrolní části programu, která obsahuje kód spouštějící kontrolu odevzdaných testů a zajišťuje výpis výsledků kontroly. Při kontrole se využívají třídy *ValidatorChyby.java* a *ValidatorVse.java*, jejichž popis je uveden níže.

ValidatorVse.java

Tato třída zajišťuje statickou kontrolu kompletní sady odevzdaných testů. Názvy kontrolovaných tříd testů jsou načteny ze souboru *test suite*, jehož pojmenování je pevně stanoveno (zadává se jako argument při spuštění programu). Pokud některá z odevzdaných testových tříd není součástí *test suite*, není použita ani při statické kontrole. Tímto je zajištěna kontrola požadavku zadání, že veškeré testy musí být součástí *test suite*. Při kontrole je ověřováno, zda se v kompletní sadě testů vyskytuje test všech požadovaných částí aplikace (tzn. v testech se prostřednictvím lokátoru přistupuje ke všem očekávaným elementům stránky).

ValidatorChyby.java

V této třídě probíhá dynamická kontrola chybových testů. Spuštěním testů je získán počet testů, které skončily chybou a názvy metod odpovídajících testům. Následně jsou z nepřeložených zdrojových kódů testů načteny metody (jejich kód) s těmito názvy a je provedeno ověření, zda testové metody končící chybou testují funkcionality, která chybu skutečně obsahuje (tzn. provede se ověření, zda se v metodách prostřednictvím lokátoru přistupuje ke všem očekávaným elementům stránky).

7.5 Validační doména

Validační doména byla připravována na testovací instanci *Validátoru*. Počáteční kroky *validační domény* provádějí kontrolu názvu odevzdaného archivu, jeho rozbalení, kontrolu názvu obsažených adresářů a kontrolu existence testových souborů. Pokud některý z těchto kroků skončí chybou, je validace ukončena. Tímto je zajištěno, že vstupy kontrolního programu (tzn. soubory testů) existují.

Původní verze části kontrolního programu s názvem *Preprocesor* ukládala soubory s upraveným kódem odevzdaných testů do nových adresářů, které byly *Preprocesorem* rovněž vytvořeny. Z bezpečnostních důvodů jsou veškeré programy spuštěné na *Validátoru* spuštěny pod speciálním uživatelem. Vlastníkem *Preprocesorem* vytvořených adresářů se tedy stal tento uživatel. Tato vlastnost *Validátoru* zapříčinila, že v následujícím kroku *validační domény* nebylo kvůli nedostatečným přístupovým právům možné vykonat překlad kódu upravených testů. Program tak bylo nezbytné upravit do podoby, ve které nejsou adresáře vytvářeny přímo programem, ale již se předpokládá jejich existence. Zároveň byl do *validační domény* přidán krok, který vytvoření adresářů zajistí (provede kopii adresářů z adresáře domény).

Po odstranění výše popisovaného problému s přístupovými právy byla již validace funkční. Testování validace a kontrolního programu jsem prováděl manuálně odevzdáváním testů, které obsahovaly různé nedostatky. Tím jsem ověřil, že validace na veškeré očekávané nedostatky v odevzdávaných testech reaguje adekvátním způsobem.

7.6 Zadání projektu

Formulaci výsledného zadání projektu provedl vyučující předmětu „KIV/OKS“. Podle finálního zadání jsem upravil kontrolní program tak, aby jím ověřované výskyty lokátorů v testech a počty chybových testů odpovídaly zadání. Program jsem poté zkompiloval a připravil k distribuci spolu se zadáním projektu.

7.7 Zhodnocení

Výsledné řešení je funkční. Kontrolní program je odolný proti základním způsobům oklamání – např. odevzdání chybových testů, které sice dle očekávání končí chybou, ale neprovádí testování chybné části testované aplikace. Ošetření pokročilejších technik překonání kontrolního programu nebylo vyžadováno, jelikož náročnost jejich provedení by zpravidla byla vyšší, než zpracování úlohy dle zadání. Funkčnost kontroly je závislá na struktuře zdrojového kódu testů exportovaných z nástroje *Selenium IDE*. Pokud by při některé z aktualizací *Selenium IDE* došlo k úpravě exportu, bylo by pravděpodobně nutné provést drobné úpravy kontrolního programu.

8 Projekt „oks-10“

Projekt „oks-10“ bude spočívat ve vytvoření sady testů v programovacím jazyce *Java* s využitím frameworku *JUnit* a knihovny *Selenium WebDriver*. Jako testovaná webová aplikace opět poslouží *Diskusní fórum*.

8.1 Cíl projektu

Cílem projektu je seznámit řešitele s tvorbou *Selenium* testů psaných ve formě zdrojového kódu *Java* a zprostředkovat tak srovnání s vytvářením testů v grafickém uživatelském prostředí nástroje *Selenium IDE*. Během zpracování projektu si řešitel prakticky vyzkouší použití některých základních datových typů a metod poskytovaných knihovnou *Selenium*.

8.2 Analýza

Při návrhu projektu „oks-10“ je možné vycházet z poznatků získaných během přípravy projektu „oks-09“. Vstupem kontrolního programu bude i v tomto případě kód testů v jazyce *Java*, který využívá framework *JUnit* a knihovnu *Selenium WebDriver*. Kontrolu tedy bude vhodné založit na stejném principu – tzn. spuštění testů a následná statická kontrola jejich zdrojového kódu.

8.2.1 JUnit vs. TestNG

Místo testovacího frameworku *JUnit* by bylo možné použití alternativního frameworku *TestNG*. Při testování jednoduché webové aplikace jakou je *Diskusní fórum* by ale rozšířené možnosti *TestNG* oproti *JUnit* nebyly využity. Framework *JUnit* je navíc obecně známější, doplňky k jeho použití častěji bývají součástí základních distribucí vývojových prostředí a jeho použití v prostředí *Validátoru* bylo již vyzkoušeno během přípravy projektu „oks-09“.

8.2.2 Vytváření instance WebDriveru

Jak již bylo uvedeno v části 5.1, psaní testů přímo ve formě zdrojového kódu dává jejich tvůrci větší volnost. Například vytváření a ukončování instance *WebDriver*, která je pro jakýkoli *Selenium* test nezbytná, lze provést několika způsoby.

Nejzákladnější možností je vytvoření instance na začátku každé testové metody a obdobně ukončení na konci každé testové metody. Při použití tohoto způsobu ale metody obsahují zbytečný duplicitní kód. V důsledku toho je i poměrně komplikovaná případná změna použité implementace *WebDriver*.

O něco lepším způsobem je v testovací třídě deklarovat statickou referenční proměnnou pro *WebDriver* a více využít možnosti frameworku *JUnit*. Kód umístěný v samostatných metodách s anotacemi „@Before“ a „@After“ je automaticky proveden před (before) a po (after) každé z testových metod. Vytvoření a ukončení instance *WebDriver* lze tedy umístit do těchto dvou metod a tím eliminovat duplicitu v kódu. Řešení ale stále není ideální. Vytvoření instance *WebDriver* je poměrně časově náročné a jeho opakované provádění před každou testovou metodou značně prodlužuje celkovou dobu vykonání testů. Opakované vytváření instance *WebDriver* je navíc zcela zbytečné. Jedinou vytvořenou instancí by bylo možné použít napříč všemi testovými metodami.

Opakované využívání jediné instance lze zařídit náhradou použitých anotací „@Before“ a „@After“ za „@BeforeClass“ a „@AfterClass“. Instance *WebDriver* je pak vytvořena pouze jedenkrát a to před vykonáním testových metod dané třídy. Následně jsou vykonány jednotlivé testové metody a až poté je instance ukončena. Pokud je ve třídě větší počet testových metod, je vykonání testů znatelně rychlejší.

Asi nejlepším přístupem je vytvoření třídy předka všech ostatních testových tříd, v této třídě předka deklarovat statický atribut pro *WebDriver* a také zde definovat metody s anotacemi „@BeforeClass“ a „@AfterClass“ zajišťující vytvoření požadované instance *WebDriver*. Ve třídách potomků je pak možné k instanci přistupovat prostřednictvím statického atributu předka. Při potřebě změnit použitou implementaci *WebDriver* stačí úpravu provést pouze ve třídě předka. Ze stejného důvodu je výhodné ve třídě předka definovat také statický atribut sloužící k uložení tzv. *base URL*. Testy je pak možné snadno spustit i na jiné webové adrese.

Z pohledu validace jsou některé výše uvedené varianty problematické. Při opakovaném vytváření instance *WebDriver* před každým testem je kvůli omezenému času validace velmi limitován počet vykonatelných testů. Pokud by navíc v zadání projektu nebylo striktně požadováno použití implementace *HtmlUnitDriver*, byla by programová úprava odevzdaného zdrojového kódu do podoby využívající *HtmlUnitDriver* komplikovaná. Projekt je proto vhodné připravit tak, aby připouštěl použití pouze jediné z výše uvedených variant (pravděpodobně poslední zmiňované).

Poznámka: V případě použití frameworku *TestNG* lze provádět vytváření a ukončování instance *WebDriver* ještě úsporněji. Díky metodám s anotacemi „@BeforeSuite“ a „@AfterSuite“ je možné instanci vytvořit pouze jedenkrát před vykonáním všech testů *test suite* a až po jejich vykonání ji ukončit (tzn. jedna instance *WebDriver* je používána napříč testovými třídami).

8.2.3 Použití konstant pro lokátory

Pokud jsou testy psány ručně, nelze jako v případě testů exportovaných z nástroje *Selenium IDE* předpokládat, že řetězce lokátorů jsou umístěny přímo v kódu metod. Lokátory mohou být definovány například jako konstanty a v tělech testových metod pouze používány. Statická kontrola spočívající v ověření výskytu očekávaných lokátorů by proto musela fungovat korektně i při použití konstant. Další možností, jak použití konstant z kódu zpětně odstranit a vyhnout se tak složitější kontrole, je využití *dekompilace*. Pokud je zdrojový kód *zkompilován* a následně se provede jeho *dekompilace*, je v dekompilemém výstupu každé použití konstanty nahrazeno přímo její hodnotou. *Dekompilaci* lze provést například *Java* programem *jd-cli* [22].

8.2.4 Volání privátních metod

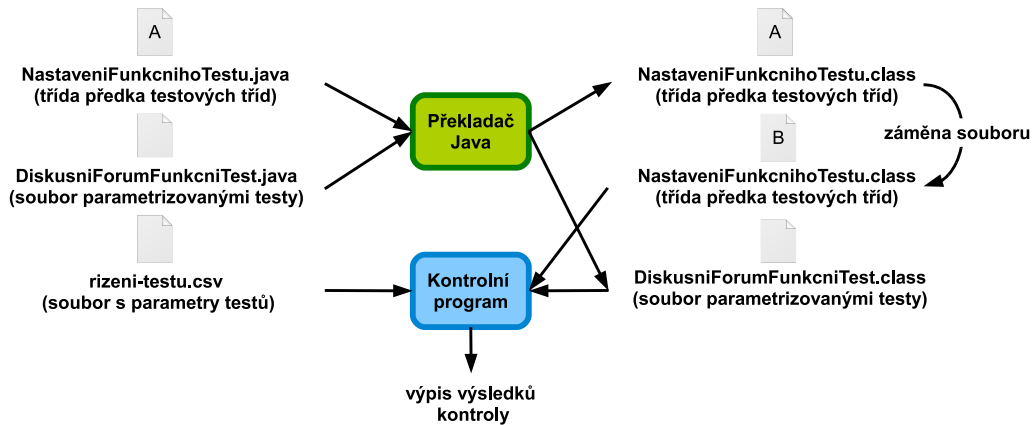
Další možnou komplikací kontroly ručně psaných testů je použití privátních metod volaných z testových metod. Během statické kontroly testů v projektu „oks-09“ jsou použité lokátory vyhledávány pouze v kódu testových metod. Kontrolu u projektu „oks-10“ by ale bylo nutné provádět tak, aby vyhledávání lokátorů probíhalo i v kódu metod, které jsou z testových metod volány.

8.3 Řešení

Komplikace spojené s různými způsoby vytvoření instance *WebDriveru* jsem vyřešil definicí třídy, která je použitelná jako předek všech testových tříd a která vytváření instance provádí. Třída bude distribuována spolu se zadáním projektu a její použití bude jedním z požadavků plynoucích ze zadání. Toto řešení rovněž umožní snadnou změnu použité implementace *WebDriveru* za *HtmlUnitDriver* před spuštěním testů na *Validátoru*. Změna bude proveditelná záměnou celé třídy předka za ekvivalentní třídu, která již použití implementace *HtmlUnitDriver* obsahuje.

Pokusy s programem *jd-cli* provádějícím dekompilaci přeložených *Java* programů jsem ověřil možnost jeho využití jako prostředku k odstranění konstant ze zdrojového kódu testů. Volání privátních metod z testových metod by ale bylo vhodné zakázat v zadání projektu nebo zohlednit při programování kontrolní aplikace. Obě tyto problematické situace byly nakonec vyřešeny specifickým návrhem zadání, které vytvořil vyučující předmětu „KIV/OKS“. Zadání bylo sestaveno tak, aby při jeho plnění byly použity parametrizované *JUnit* testy. Jedním z přesně definovaných parametrů testů je lokátor elementu stránky. Parametry jsou ve finální podobě testů načítány z *CSV* souboru, čímž je vyloučeno uložení hodnot lokátorů jako konstant a značně usnadněna kontrola. Součástí zadání je navíc kostra testů, díky které je eliminováno vytváření privátních metod a jejich volání z testových metod.

Navržené zadání projektu a kostra testů byla později upravena tak, aby v odevzdaných testech bylo možné provést změnu použité implementace *WebDriveru* pouze náhradou jediné třídy (viz první odstavec této podkapitoly). Dále byla provedena úprava, která umožňuje náhradou této třídy změnit i zdroj parametrů testů. Ověření funkčnosti vytvořených testů bude při kontrole probíhat jejich spuštěním pouze s omezenou množinou parametrů. Tím se doba kontroly výrazně zkrátí. Ověření požadovaného pokrytí testy bude prováděno statickou kontrolou *CSV* souboru s parametry testů. Na základě finální podoby zadání projektu jsem vytvořil kontrolní program. Schéma kontroly je znázorněno na obrázku 8.1.



Obrázek 8.1: „oks-10“ – schéma kontroly

Postup kontroly je následující:

1. Překlad odevzdaných testů.
2. Náhrada přeložené třídy předka testů ekvivalentní třídou, která používá *HtmlUnitDriver* a jiný zdroj parametrů testů.
3. Spuštění kontroly:
 - (a) Spuštění parametrických *JUnit* testů, ověření počtu vykonaných testů a testů končících chybou.
 - (b) Statická kontrola *CSV* souboru s parametry testů ověřující požadované pokrytí (realizováno ověřením výskytu očekávaných parametrů v *CSV* souboru).

8.4 Kontrolní program

Kontrolní program k projektu „oks-10“ se skládá z jediné třídy *Validator.java*. Program nejprve provádí spuštění odevzdaných testů. Zde se předpokládá, že byla již předem provedena záměna přeložené třídy předka testů (*NastaveniFunkcnihoTest.class*) za její upravenou verzi. Upravená verze zajišťuje použití *HtmlUnitDriveru* a obsahuje také odlišnou implementaci metody sloužící k načtení parametrů testů. Odlišnost implementace metody spočívá v použití pevně zadaných parametrů místo parametrů načtených z odevzdaného *CSV* souboru. Tímto je omezen počet vykonávaných testů. Množina pevně zadaných parametrů je však dostatečně reprezentativní, aby při jejím použití byla funkčnost odevzdaných parametrizovaných

testů dostatečně ověřena. Výsledky spuštění testů jsou porovnány s očekávaným výstupem.

Další část programu kontroluje pokrytí webové aplikace testy. Dostatečné pokrytí testy se ověří načtením parametrů testů z odevzdaného *CSV* souboru a jejich porovnáním s očekávanými parametry. Pokud některá z částí kontroly odhalí nedostatky v odevzdaných testech, je do konzole vypsána hláška informující o nalezených problémech.

8.5 Validační doména

Validační doména projektu „oks-10“ vychází z *validační domény* projektu „oks-09“. V doméně byla přizpůsobena kontrola struktury odevzdaného balíku jinému zadání a byly odstraněny kroky související se spuštěním *Preprocesoru*. Po provedení překladu odevzdaných testů je přepsána přeložená třída předka testů její modifikovanou podobou a ihned poté se spustí kontrolní program.

8.6 Zadání projektu

Zadání projektu sestavil vyučující předmětu „KIV/OKS“. Zadání spočívá ve vytvoření sady funkčních testů využívajících parametrizované *JUnit* testy a nástroj *Selenium WebDriver*. Díky vhodné formulaci zadání doplněného o závaznou kostru testů bylo vytvoření kontrolního programu poměrně jednoduché.

8.7 Zhodnocení

Testování kontrolního programu jsem prováděl manuálně prostřednictvím vytvořené *validační domény* na *Validátoru*. Nalezené drobné nedostatky byly ihned opraveny a výsledný kontrolní program je plně funkční. Použití z hlediska kontroly problematických konstrukcí ve zdrojovém kódu testů bylo eliminováno zadáním projektu. Kontrolní program je tak znatelně jednodušší, než jsem původně předpokládal.

9 Závěr

Cílem mojí diplomové práce bylo navrhnout a připravit několik malých studentských projektů pro potřeby předmětu „KIV/OKS“. Součástí zadání bylo vytvořit ukázkové řešení, realizovat automatickou validaci těchto úloh (prováděnou při jejich odevzdávání na server) a důkladně otestovat funkčnost.

Z důvodu nemožnosti jednoznačného automatického vyhodnocení úloh souvisejících s *JMeter* a *Redmine* nebyly tyto nástroje pro přípravu úloh použity. V rámci práce byly vytvořeny dvě úlohy zaměřené na testování webových aplikací s využitím frameworku *Selenium*. Jako podklad pro tyto úlohy jsem vytvořil jednoduchou webovou aplikaci, která obsahuje záměrně zanesené chyby. Webová aplikace se skládá ze dvou částí, z nichž jedna je určena pouze k demonstraci některých postupů v rámci přednášek předmětu „KIV/OKS“ [20] a druhá je využívána právě při zpracování samostatných úloh [21]. První z úloh slouží k seznámení s nástrojem *Selenium IDE*, který existuje jako doplněk do internetového prohlížeče *Firefox*. Druhá úloha se zaměřuje na použití *Selenium WebDriver* v kombinaci s testovacím frameworkem *JUnit*. Pro obě úlohy jsem naprogramoval aplikace, které provádí jejich kontrolu a tvoří klíčovou část automatické validace.

Vytvořené úlohy napomáhají k získání praktických zkušeností s některými částmi testovacího frameworku *Selenium*. Vzhledem k rozsáhlosti frameworku *Selenium* nebylo pochopitelně možné v rámci omezeného počtu úloh obsáhnout veškeré možnosti tohoto nástroje. Do značné míry byla limitující i podmínka automatické validace úloh. Zhotovené kontrolní programy bylo možné vytvořit jen díky uvedení některých požadavků v zadání úloh (viz příloha A a příloha B). Funkčnost kontrolních programů a celkově automatické validace byla již úspěšně ověřena i při skutečném řešení úloh studenty v rámci výuky předmětu „KIV/OKS“ v letním semestru akademického roku 2013/2014.

Slovník použitých termínů a zkratek

Ant – Nástroj pro správu a řízení softwarových projektů včetně automatizace sestavení aplikací.

Build – Sestavení softwarového produktu do spustitelné podoby.

Classpath – Řetězec, který určuje virtuálnímu stroji *Javy* nebo kompilátoru, kde se nacházejí třídy programu nebo balíky knihoven.

CSS – Cascading Style Sheets neboli kaskádové styly jsou prostředkem k popisu způsobu zobrazení webových stránek.

CSV – Formát textového souboru, ve kterém jsou hodnoty odděleny čárkami.

DOM – Document Object Model neboli objektový model dokumentu je způsob reprezentace *XML* dokumentu, který umožňuje přístup k jeho jednotlivým částem.

Framework – Softwarová struktura, která slouží jako podpora při programování a vývoji softwarových produktů.

GUI – Graphical User Interface neboli grafické uživatelské rozhraní.

HTML – HyperText Markup Language neboli značkovací jazyk pro hypertext je jazyk primárně určený k tvorbě stránek publikovaných v prostředí internetu.

HTTP – Hypertext Transfer Protocol je protokol sloužící k výměně hypertextových dokumentů.

JavaScript – Programovací jazyk.

JSP – Java Server Pages je jazyk pro vytváření dynamických *HTML* stránek založených na programovacím jazyce *Java*.

Maven – Nástroj pro správu a řízení softwarových projektů včetně automatizace sestavení aplikací.

MD5 – Označení kryptografické funkce, která zadanému vstupu jednoznačně přiřazuje řetězec fixní délky.

MVC – Model View Controller je softwarová architektura.

Návrhový vzor – Obecné řešení problému, které lze opakovaně aplikovat při řešení některých úloh.

Portlet – Komponenta, která je základním stavebním prvkem webových portálů.

Proxy server – Prostředník mezi klientem a cílovým serverem. Z pohledu klienta se jeví jako server, z pohledu cílového serveru se jedná o klienta.

Repozitář – V oblasti vývoje softwaru se jedná o sdílené uložení obsahující zdrojové kódy nebo instalační balíčky nějakého softwarového produktu.

Servlet – Program v programovacím jazyce *Java*, který tvoří část webové aplikace.

URL – Uniform Resource Locator lze přeložit jako jednotný lokátor zdrojů. Jedná se o řetězec s přesně definovanou strukturou, který určuje umístění nějakého zdroje v prostředí internetu.

Utilita – Pomocný program.

WAR – Web application ARchive neboli archiv pro webové aplikace je formát obsahující zdrojové kódy a další soubory webové aplikace.

XLS – Výchozí formát souborů tabulkového procesoru *Microsoft Excel*.

XML – Extensible Markup Language neboli rozšiřitelný značkovací jazyk.

XPath – Jazyk s jehož pomocí je možné vybírat elementy *XML* a pracovat s jejich hodnotami a atributy.

Použité zdroje a literatura

- [1] KIV/OKS - popis předmětu. *Portál ZČU* [online]. [cit. 2014-05-03]. Dostupné z: <<https://portal.zcu.cz/stag?urlid=prohlizeni-predmet-sylabus&predmetZkrPrac=KIV&predmetZkrPred=OKS&predmetRok=2013&predmetSemestr=XX&formRozvrhZobrazeni=20>>.
- [2] PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
- [3] MCCONNELL, Steve. *Code complete*. 2nd ed. Washington: Microsoft Press, 2004. ISBN 07-356-1967-0.
- [4] HEROUT, Pavel. *Texty k přednáškám z předmětu KIV/OKS*. Plzeň, 2014.
- [5] CAPERS, Jones. *A short history of the cost per defect metric* [online]. 2009, s. 23 [cit. 2014-05-03]. Dostupné z: <semat.org/wp-content/uploads/2012/03/a_short_history_of_the_cost_per_defect_metric.doc>.
- [6] KONTRHONZ, Otakar. *Porovnání testovacích nástrojů pro jazyk Java a platformu Spring* [online]. Praha, 2012 [cit. 2014-05-03]. Dostupné z: <https://dip.felk.cvut.cz/browse/pdfcache/kotrhot_a_2012dipl.pdf>. Diplomová práce. České vysoké učení technické v Praze.
- [7] *JUnit* [online]. [cit. 2014-05-03]. Dostupné z: <<http://junit.org/>>.
- [8] *TestNG* [online]. [cit. 2014-05-03]. Dostupné z: <<http://testng.org/>>.
- [9] *SeleniumHQ* [online]. [cit. 2014-05-03]. Dostupné z: <<http://docs.seleniumhq.org/>>.
- [10] *Apache Maven* [online]. [cit. 2014-05-03]. Dostupné z: <<http://maven.apache.org/>>.
- [11] *Apache Ant* [online]. [cit. 2014-05-03]. Dostupné z: <<http://ant.apache.org/>>.
- [12] *Apache JMeter* [online]. [cit. 2014-05-03]. Dostupné z: <<http://jmeter.apache.org/>>.

- [13] *Redmine* [online]. [cit. 2014-05-03]. Dostupné z: <<http://www.redmine.org/>>.
- [14] *Portál ZČU* [online]. [cit. 2014-05-03]. Dostupné z: <<https://portal.zcu.cz/>>.
- [15] XLS Export. *Redmine* [online]. [cit. 2014-05-03]. Dostupné z: <http://www.redmine.org/plugins/redmine_xls_export>.
- [16] *Apache Tomcat* [online]. [cit. 2014-05-03]. Dostupné z: <http://tomcat.apache.org/>.
- [17] Markup Validation Service. *W3C* [online]. [cit. 2014-05-03]. Dostupné z: <<http://validator.w3.org/>>.
- [18] CSS Validation Service. *W3C* [online]. [cit. 2014-05-03]. Dostupné z: <<http://www.css-validator.org/>>.
- [19] *KIV/OKS* [online]. [cit. 2014-05-03]. Dostupné z: <<http://oks.kiv.zcu.cz/>>.
- [20] Převodník. *KIV/OKS* [online]. [cit. 2014-05-03]. Dostupné z: <<http://oks.kiv.zcu.cz/Prevodnik>>.
- [21] Diskusní fórum. *KIV/OKS* [online]. [cit. 2014-05-03]. Dostupné z: <<http://oks.kiv.zcu.cz/Forum>>.
- [22] Command line Java Decompiler. *GitHub* [online]. [cit. 2014-05-03]. Dostupné z: <<https://github.com/kwart/jd-cmd>>.

Přílohy

Příloha A

Specifikace webové aplikace Převodník

Verze dokumentu: 1.0 15. 03. 2014

1.1 02. 04. 2014

Aplikace se skládá ze tří stránek: „Úvod“, „Převodník“ a „Nápověda“. Navigace je řešena horizontálním menu umístěným v hlavičce. Aplikace slouží k převodům vybraných délkových jednotek.

Stránka „Úvod“

Stránka „Úvod“ obsahuje stručné informace o aplikaci a odkazy na webovou stránku s informacemi o předmětu „KIV/OKS“ (na univerzitním *Portálu*), na web *KIV* a na web *ZČU*.

Stránka „Převodník“

Stránka „Převodník“ obsahuje následující ovládací prvky:

Pole pro zadání hodnoty k převodu

- Kladné celé číslo (např. 5).
- Kladné desetinné číslo ve standardním formátu (např. 5.1), ve kterém je jako oddělovač desetinné části použit znak . (tečka) nebo , (čárka).
- Kladné desetinné číslo ve vědeckém formátu (např. 0.51E1), ve kterém je jako oddělovač desetinné části použit znak . (tečka) nebo , (čárka).
- Bílé znaky na začátku a na konci zadané hodnoty jsou ignorovány a po odeslání formuláře odstraněny.
- Pole musí být vyplněno.

Pole s výběrem jednotky vstupu

- Výběr z nabízených možností.

Pole pro výpis převedené hodnoty

- Needitovatelné pole.

Pole s výběrem jednotky výstupu

- Výběr z nabízených možností.

Tlačítko k odeslání formuláře

Tlačítko k vymazání formuláře

V případě, že některý ze zadaných vstupů nesplňuje výše uvedené požadavky, je vypsána hláška „Nelze převést“ doplněná o podrobnější popis chyby. Při neúspěšném odeslání formuláře zůstává zachován obsah pole s hodnotou k převodu i zvolená jednotka vstupu a výstupu. Pole sloužící k výpisu převedené hodnoty zůstává v tomto případě prázdné.

Po úspěšném odeslání formuláře tlačítkem „Převést“ je v příslušném poli zobrazena převedená hodnota ve standardním formátu. Převedená hodnota je vždy vypsána bez nevýznamových nul.

Tlačítko „Vymaž“ provádí vymazání obsahu formuláře do výchozího stavu (tzn. vymazání obsahu pole pro vstup i výstup a zvolení výchozí jednotky vstupu i výstupu).

Stránka „Nápověda“

Stránka „Nápověda“ obsahuje stručnou nápovědu k použití aplikace.

Příloha B

Specifikace webové aplikace Diskusní fórum

Verze dokumentu: 1.0 10. 03. 2014
1.1 15. 03. 2014
1.2 02. 04. 2014

Aplikace se skládá ze tří stránek: „Úvod“, „Registrace“ a „Nápověda“. Navigace je řešena horizontálním menu umístěným v hlavičce. Aplikace představuje smyšlené diskusní fórum, jehož jedinou funkcionalitou je registrace nového uživatele. Po úspěšné registraci jsou zobrazeny údaje o registrovaném uživateli, které se sestavují na základě vstupů zadaných do registračního formuláře.

Stránka „Úvod“

Stránka „Úvod“ obsahuje stručné informace o aplikaci a odkazy na webovou stránku s informacemi o předmětu „KIV/OKS“ (na univerzitním *Portálu*), na web *KIV* a na web *ZČU*.

Stránka „Registrace“

Stránka „Registrace“ obsahuje registrační formulář s následujícími vstupy:

Login

- Řetězec o délce 5 až 15 znaků (včetně krajních hodnot).
- Řetězec může být složen pouze z písmen bez diakritiky a číslic, přičemž číslice nesmí být prvním znakem.
- Pole musí být vyplněno.

Heslo

- Řetězec o délce 5 až 15 znaků (včetně krajních hodnot).
- Řetězec může být složen pouze z písmen, číslic a znaků (tzn. vše mimo bílých znaků).
- Pole musí být vyplněno.

Heslo znovu

- Řetězec o délce 5 až 15 znaků (včetně krajních hodnot).
- Řetězec může být složen pouze z písmen, číslic a znaků (tzn. vše mimo bílých znaků).
- Obsah pole musí být totožný s obsahem pole *Heslo* (shoda s obsahem pole *Heslo* je ověřována pouze v případě vyplnění pole *Heslo* i *Heslo znovu*).
- Pole musí být vyplněno.

Jméno

- Řetězec o délce 2 až 15 znaků (včetně krajních hodnot).
- Řetězec může být složen pouze z písmen.
- Pole musí být vyplněno.

Příjmení

- Řetězec o délce 2 až 15 znaků (včetně krajních hodnot).
- Řetězec může být složen pouze z písmen.
- Pole musí být vyplněno.

Pohlaví

- Výběr z nabízených možností.

Datum narození

- Řetězec složený pouze z číslic a znaku . (tečka).
- Datum musí být ve formátu DD.MM.RRRR.
- Datum musí existovat (30.2.1965 je nevalidní).
- Datum nesmí být v budoucnosti (1.1.2030 je nevalidní).
- Pole nemusí být vyplněno.

Preferovaný programovací jazyk

- Výběr z nabízených možností.

Veřejná poznámka

- Řetězec s maximální délkou 140 znaků.
- Řetězec může být složen z písmen číslic a libovolných znaků (včetně bílých znaků).
- Pole nemusí být vyplněno.

U všech vstupních polí jsou ignorovány bílé znaky na začátku a na konci zadaných řetězců. Tyto bílé znaky jsou po odeslání formuláře ze zadaných řetězců odstraněny.

Odeslání formuláře je prováděno stiskem tlačítka „Registrovat“. V případě, že některý ze zadaných vstupů nesplňuje výše uvedené požadavky, je vypsána hláška „Chybné vstupy - opravte prosím zvýrazněná pole“ a u vstupních polí s nevalidní hodnotou je vypsán podrobnější popis chyby. Při neúspěšném odeslání formuláře zůstávají veškeré zadané vstupy mimo „Heslo“, „Heslo znovu“ a „Souhlasím s podmínkami registrace“ vyplněny původně zadanou hodnotou.

Pokud je formulář odeslán bez zatrženého políčka „Souhlasím s podmínkami registrace“, je vypsána hláška „Musíte souhlasit s podmínkami registrace“.

Po úspěšné registraci je vypsána hláška „Nový uživatel byl úspěšně vytvořen“ a následující needitovatelná pole:

Login

- Zadaný login uživatele.

MD5 hash hesla

- MD5 hash zadaného hesla.

Jméno

- Zadané jméno.

Příjmení

- Zadané příjmení.

Pohlaví

- Zadané pohlaví.

Věk

- Věk vypočítaný ze zadaného data narození s přesností na dny.

Preferovaný programovací jazyk

- Zadaný preferovaný programovací jazyk.

Veřejná poznámka

- Text zadané veřejné poznámky.

Poznámka: Na stránce zobrazené po úspěšné registraci je rovněž zobrazen odkaz na úvodní stranu.

Stránka „Nápověda“

Stránka „Nápověda“ obsahuje stručnou nápovědu k použití aplikace.

Příloha C

Obsah příloženého CD

- **DP** – text diplomové práce v *PDF*,
- **Oks-09**
 - **Kontrola** – spustitelná ukázka kontroly,
 - **Projekt_Eclipse** – zdrojový projekt kontrolního programu,
 - **Validator_domena** – *XML* soubor s kroky validační domény,
 - **Zadani** – zadání projektu,
- **Oks-10**
 - **Kontrola** – spustitelná ukázka kontroly,
 - **Projekt_Eclipse** – zdrojový projekt kontrolního programu,
 - **Validator_domena** – *XML* soubor s kroky validační domény,
 - **Zadani** – zadání projektu,
- **Webova_aplikace**
 - **Projekt_Eclipse** – zdrojový projekt webových aplikací (*Převodník, Diskusní fórum*),
 - **Specifikace** – dokumenty specifikace webových aplikací (*Převodník, Diskusní fórum*),
 - **War** – vytvořený *WAR* soubor sloužící k nasazení aplikací na server.

Poznámka: Autorem zadání projektů a vzorových řešení použitých v ukázkách kontroly je Pavel Herout.

Příloha D

Uživatelská příručka

Spuštění ukázky kontroly

Úspěšné spuštění ukázky kontroly vyžaduje *JRE* (*Java Runtime Environment*), doporučovaná minimální verze je 1.7.0_03.

„oks-09“

Obsah adresáře kontroly projektu „oks-09“ je:

- **export_chyby** – odevzdané testy končící chybou,
- **export_vse** – kompletní sada odevzdaných testů,
- **lib** – knihovny využívané kontrolním programem,
- **validace** – soubory kontrolního programu.

Kontrolu je možné spustit pomocí souboru *kontrola.bat* nebo zadáním následujících příkazů do příkazového řádku *Windows*:

```
java validace.Preprocesor export_vse export_chyby oks09All oks09Err
```

```
javac -encoding UTF-8 -cp lib\selenium-server-standalone-2.41.0.jar oks09All\*.java
```

```
javac -encoding UTF-8 -cp lib\selenium-server-standalone-2.41.0.jar oks09Err\*.java
```

```
java -cp lib\selenium-server-standalone-2.41.0.jar;. validace.Validator oks09All SuiteForum.java oks09Err
```

„oks-10“

Obsah adresáře kontroly projektu „oks-10“ je:

- **lib** – knihovny využívané kontrolním programem,
- **oks10** – odevzdané soubory testů,
- **validace** – soubory kontrolního programu.

Kontrolu je možné spustit pomocí souboru *kontrola.bat* nebo zadáním následujících příkazů do příkazového řádku *Windows*:

```
javac -encoding UTF-8 -cp lib\selenium-server-standalone-2.41.0.jar oks10\  
*.java
```

```
copy /Y validace\NastaveniFunkcnihoTestu.class oks10\  

```

```
java -cp lib\selenium-server-standalone-2.41.0.jar;. validace.Validator ok  
s10\rizeni-testu.csv
```