Západočeská univerzita
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Analýzy automaticky vyhodnocovaných studentských projektů

Plzeň 2016 Jitka Fürbacherová

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2016

Jitka Fürbacherová

Abstract

The main goal of this thesis is to interpret data stored on the validation server which were obtained during the validation of student's projects. The result of this thesis is to be used by lecturers for a detailed analysis of the semester and of the attitude of students. The solution are two Java desktop applications which use library for the generating of graphs. This thesis describes used technologies and tools that needed to be studied for the implementation of this thesis. These are the function principle of the validation server and the UML-test tool. In the next part the analysis of the data stored on the validation server is done and the of graphs for the graphic representation is designed. After that there is the description of applications which implement the designed representation.

Abstrakt

Účelem bakalářské práce je přehledně interpretovat data uložená na validátoru, získaná při automatické kontrole studenských prací. Výsledek práce má sloužit přednášejícímu k podrobnější analýze, jak průběhu semestru, tak i přístupů jednotlivých studentů. Řešením jsou dvě Java aplikace, které využívají knihovny pro generování grafů. Tento dokument nejprve popisuje jednotlivé technologie, které byly využity, a nástroje, které bylo potřeba pro realizaci bakalářské práce nastudovat. Jedná se o princip fungování validačního serveru a nástroje UML–test. Poté je provedena analýza dat uložených na validátoru a navržena sada grafů pro jejich grafickou reprezentaci. Následně jsou popsány aplikace, které návrh realizují.

Poděkování

Ráda bych poděkovala vedoucímu práce doc. Ing. Pavlu Heroutovi, Ph.D. za cenné rady, věcné připomínky a ochotu vždy pomoci s jakýmkoliv problémem.

Obsah

1 Úvod

V předmětech KIV/PPA1, KIV/OOP, KIV/OKS a dalších odevzdávají studenti na Portál řadu semestrálních projektů. Tyto projekty jsou automaticky validovány. Tento postup probíhá již řadu let, a tak jsou k dispozici cenné údaje, které mohou pomoci při zvyšování kvality výuky zmíněných předmětů v příštích letech.

Cílem této bakalářské práce je provést analýzu dat, která jsou na validátoru uložena. Následně navrhnout a vytvořit nástroj pro snadné získání relevantních dat a jejich přehledného zpracování nejlépe formou agregovaných grafů. Jelikož se na validačním serveru vyskytuje velké množství domén, které jsou používány při validaci několika předmětů v průběhu let, bude nutné konkrétní data nejdříve vyfiltrovat.

Speciální pozornost bude kladena na vyhodnocování domácích úloh týkajících se UML diagramů. V roce 2015 byl totiž v rámci jiného studentského projektu vytvořen rozsáhlý systém, který umožňuje velmi detailní testování odevzdaných UML diagramů. Pomocí těchto testů lze získat například časovou řadu chyb jednotlivých úloh.

Výstupem práce bude sada agregovaných a personálních grafů, které budou znázorňovat vizualizaci průběhu celého semestru a jednotlivých odevzdání všech domácích úloh konkrétních studentů.

2 Použité technologie

Pro realizaci této bakalářské práce budou použity níže popsané technologie. Dále je také samozřejmě předpokládána znalost objektově orientovaného programovacího jazyka *Java*. Více informací o tomto jazyku je možné získat v [1, 2]. Pro práci s časem bude využíváno API *DateTime* z balíčku *java.time*, které přináší *Java 8*. Tato knihovna bude také použita při testování aplikace pro zjištění doby jejího běhu.

2.1 Parsování HTML

Úkolem této práce je vizualizovat data uložená na validačním serveru ve formátu HTML (viz 3.2 na straně 10). Tyto soubory bude nutné zpracovávat a načíst z nich potřebné informace. Protože se bude jednat pouze o jednoduché získání několika řetězců, bude stačit HTML soubor parsovat nějakým běžně dostupným parserem.

2.1.1 JSoup

V současné době je pro parsování HTML souborů v jazyce *Java* nejvíce využívaná knihovna *JSoup*. Zároveň je tento nástroj také nejznámější.

Tato knihovna slouží k usnadnění vyhledávání, čtení a zpracování HTML elementů. Vyhledávání a extrakce dat probíhá s využitím CSS selekce a průchodu DOM. Oblíbenost JSoup je dána mimo jiné tím, že umožňuje načíst HTML kód z řetězce, souboru i z webové stránky. Podle toho, odkud je žádoucí data ve formátu HTML načítat, je potřeba zvolit vhodné parametry metody `Jsoup.parse()`, která, jak již název napovídá, slouží pro parsování. Její dva nejpoužívanější tvary jsou `Jsoup.parse(String html)` a `Jsoup.parse(String html, String baseUri)`. Parametr *html* obou těchto metod představuje text, na který je vznesen požadavek parsování. Druhý příklad se používá při načítání HTML kódu z webu. Výstupem těchto metod je instance třídy *Document*, se kterou je možno dále pracovat. Pokud je pouze část HTML kódu validní (zbytek může být poškozen), používá se metoda `Jsoup.parseBodyFragment(String html)`. Výstup je rovněž instance třídy *Document*. Pro komplexnější načtení HTML kódu ze souboru je možné pou-

žití `Jsoup.parse(File input, String encoding, String baseUri)`. Pomocí parametru *encoding* se nastaví kódování, *baseUri* slouží pro pokročilé načítání a pro většinu případů postačí předat metodě prázdný řetězec.

Pro používání knihovny je nutné stáhnout příslušný *.jar* soubor a připojit ho k projektu. *JSoup* je možno libovolně využívat i modifikovat. Šířena je pod MIT licencí. Další doplňující informace a návod jsou v [3].

Při předběžných experimentech bylo zjištěno, že pro danou úlohu tato knihovna naprosto vyhovuje.

2.1.2 Další podobné knihovny

Kromě výše zmíněné knihovny nabízí *Java* pro parsování HTML souborů také následující knihovny:

- **Jericho HTML Parser** šířený pod EPL (Eclipse public licence) lze využívat jak pro soukromé, tak komerční účely. Hlavní výhodou je, že umí zpracovávat i nevalidní, tedy poškozené HTML soubory. Celá knihovna má volně dostupnou online dokumentaci ve formě *Javadoc*.
- **HTML Cleaner** je opět bezplatně použitelný framework. Vyvíjen je pouze jednotlivcem, takže neposkytuje tolik funkcí jako předešlé knihovny.
- **TagSoup** „open source“ knihovna disponuje pouze základními funkcemi.
- **JTidy** je knihovna, která primárně slouží pro kontrolu syntaxe HTML. Využít ji lze ale i pro parsování.

Žádná z uvedených knihoven nepřináší takovou funkcionalitu, kterou by bylo možné v této práci využít a zároveň by ji *JSoup* neposkytovala.

2.2 Tvorba grafů

Stejně jako pro parsování HTML existují v *Javě* knihovny i pro vytváření a zobrazování grafů.

2.2.1 JFreeChart

Za nejznámější knihovnu pro grafickou reprezentaci dat se v současné době považuje JFreeChart. Framework disponuje širokým uplatněním a má velmi dobrou dokumentaci včetně široké škály ukázkových příkladů. Dostupný je také celý kód. *Javadoc* lze nalézt v [4]. Tento nástroj je hojně využíván již řadu let, šířen je pod licencí *GNU Lesser General Public Licence*, lze jej tedy bezplatně využívat.

JFreeChart umožňuje vytvářet velké množství druhů 2D i 3D grafů: sloupcové, koláčové, spojnicové, bodové atd. Pro tuto práci byly vybrány sloupcové grafy, ukázka sloupcového grafu vytvořeného touto knihovnou je uvedena na obrázku 2.1.

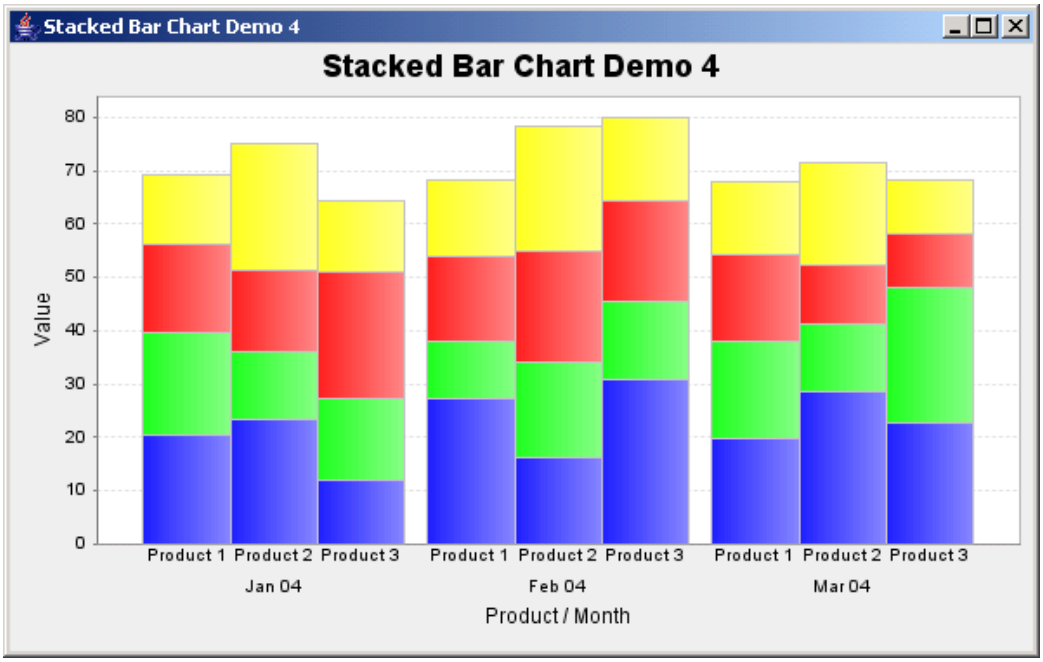
Tvorba libovolného grafu probíhá obvykle v následujících krocích v uvedeném pořadí:

1. Vytvoření datové množiny (tzv. *dataset*) ze vstupních dat, na která je vznesen požadavek vizualizace. V *datasetu* jsou uloženy informace ve tvaru: klíč – hodnota.
2. Vytvoření objektu JFreeChart z datové množiny dle definovaného způsobu zobrazení. Použita je třída *ChartFactory*, která obsahuje statické metody pro tvorbu velkého množství druhů grafů. V této části je již v paměti definována velká část konečné podoby grafu.
3. Vykreslení požadovaného grafu vytvořeného v předchozím kroku, které probíhá zejména pomocí metody *draw(Graphics2D g, Rectangle2D r)* implementované ve třídě *JFreeChart*.

Jednou z hlavních charakteristických výhod této knihovny je oddělení dat potřebných k vytvoření grafu a prostředků pro jejich reprezentaci. Další kladnou vlastností je podpora přímého exportování výstupu do PNG a JPEG.

2.2.2 Knihovna základních grafických prvků

Rudolf Pecinovský vyvinul knihovnu základních grafických prvků pro ukázku návrhových vzorů a výuku objektově orientovaného programování [6]. Tato knihovna poskytuje velmi dobře propracované API pro základní geometrické



Obrázek 2.1: Ukázka možností knihovny JFreeChart [5]

tvary a autorka této práce se s ní detailně seznámila v předmětu KIV/OOP. Při předběžných pokusech se ukázalo, že pro jeden konkrétní typ požadovaných grafů bude jednodušší použít raději tuto základní knihovnu, než se pokoušet nakonfigurovat odpovídající graf z výše zmíněného JFreeChart.

Základním kamenem knihovny je virtuální plátno, na které jsou objekty vykreslovány s využitím třídy *SpravcePlatno*, která slouží jako manažer. Dohlíží na to, aby po změně zobrazení libovolného prvku se všechny ostatní správně překreslily. Pro správné fungování existují následující možné přístupy:

1. Překreslování plátna v pravidelných intervalech bez ohledu, zda se na něm vyskytla nějaká změna.

2. Překreslování plátna pouze na výslovnou žádost.

Rudolf Pecinovský zvolil druhou variantu, jejíž výhodou je samozřejmě úspora spotřebovaného výkonu počítače, pokud není plátno zrovna překreslováno. Nevýhoda spočívá v tom, že kreslené objekty musí každou svou změnu ohlásit manažeru.

Připraveno bylo několik základních prvků, jako je obdélník, elipsa, trojúhelník a text. Knihovna byla vytvořena důsledně objektově, proto tyto obrazce definují společné rozhraní *ITvar*. Vykreslovaný prvek se nejprve přihlásí o instanci třídy *SpravcePlatna*, ta jej zařadí mezi spravované objekty.

Třída *SpravcePlatna* také disponuje metodou *ulozJakoObrazek(File soubor)*, která slouží pro uložení výstupu ve formátu PNG.

2.2.3 Další podobné knihovny

Java nabízí pro tento účel také následující knihovny:

- **Openchart2** – k vytváření sloupcových, koláčových a dalších 2D grafů, využívat ji lze zdarma pod licencí *GNU Lesser GPL*,
- **Plot4j** – pro tvorbu 2D grafů, vytvořené soubory je rovněž možno snadno exportovat do PNG i JPEG,
- **GRAL Graphing Library** – umožňuje zpracovat(importovat) data z textových souborů, audia i videa, umožňuje různé druhy zpracování dat a poskytuje rozhraní ve *Swingu*.

Informace uvedené v této části byly čerpány převážně z [7], kde je možné nalézt značné množství užitečných poznatků o knihovnách pro vytváření grafů a další jejich příklady.

2.3 JUnit

Jednotkové testy budou v této práci používány jako samozřejmá součást vývoje programu, čímž napomáhají k vyšší kvalitě výsledného programu. Nejsou ale smyslem této práce.

Jednotkové testy slouží pro ověření funkčnosti nejmenších částí systému. Nejčastěji se používají pro testování entitních tříd a jejich metod. *JUnit* framework je podporován řadou vývojových prostředí, je tedy možné využít automatické generování kostry jednotkových testů, které zejména v rozsáhlejších projektech ušetří množství rutinní práce. Také pro spuštění je ve většině

případů integrována podpora. Práce je vytvářena ve vývojovém prostředí *Eclipse*, které samozřejmě JUnit podporuje včetně automatického spouštění.

V praxi slouží *JUnit* testy pro programátora jako prvotní ověření funkčnosti jeho kódu. Jednotkové testy tedy většinou vyvíjí přímo programátor. Veškeré informace uvedené v této sekci jsou převzaté z [8, 9].

2.3.1 Implementace

Pro každou třídu, kterou chceme otestovat, je potřeba vytvořit samostatnou testovací třídu, která bude mít, dle používané konvence, stejný název doplněný o řetězec „Test“. Otestovat je možné libovolnou nesoukromou metodu. Aby byl test účinný, je nutné testovat všechny možné výstupy. Třída *org.junit.Assert* poskytuje množství *assert* metod (převzato z [9]):

- **testy rovnosti** – rovnají-li se hodnoty, test prošel,

```
assertEquals(datovy_typ ocekavanaHodnota,
             datovy_typ skutecnaHodnota)
```

- **testy logické shody**,

```
assertTrue(boolean skutecnaHodnota)
assertFalse(boolean skutecnaHodnota)
```

- **testy obsahu referenčních proměnných**,

```
assertNull(Object object)
assertNotNull(Object object)
```

- **testy shodnosti referenčních proměnných**.

```
assertSame(Object ocekavany, Object skutecny)
assertNotSame(Object ocekavany, Object skutecny)
```

2.3.2 Vyhodnocení testů

Obecně platí, že pokud selže libovolná část testu, je celý test považován za chybný. Při vyhodnocování chybných testů je nutné rozlišovat hlášení *Error*

a *Failure*. *Failure* nastane v případě, že metoda *assert* odhalila chybu, tudíž problém se vyskytuje v programu. *Error* ukazuje na problém v implementaci samotného testu. Test je považován za úspěšný až v momentě, kdy se nevyskytují ani *Error* ani *Failure*.

3 Validační server

V současné době odevzdávají studenti mnoha předmětů své semestrální práce na Portál. U některých předmětů je možný i další krok, který spočívá v automatické validaci odevzdaných úloh. Pomocí speciálního portletu je po odevzdání úkolu navázáno spojení s validačním serverem, který jej otestuje a výsledek zašle zpět na Portál [10].

Portál si lze představit jako webové místo, kde jsou uživatelům k dispozici všechny potřebné informace v personalizované podobě. Rozlišeny jsou různé role uživatelských účtů (běžný, privilegovaný,...). Na Portálu v Courseware je portlet, ve kterém učitelé zadávají semestrální práce svých předmětů.

Podrobnější popis Coursewaru, Portálu a portletů, včetně návodu, jak s nimi správně pracovat, je uveden v [11].

3.1 Princip validačního serveru

Na studentské projekty je typicky vznesena řada požadavků, které je při kontrole potřeba ověřit. Dle [12]: „Obecně lze říct, že jakákoliv kontrola, u které si odborník dokáže představit, jak by ji naprogramoval, je proveditelná validačním serverem.“

Validační server nejprve přijme úlohu k otestování spolu s definicí konkrétních pravidel – validační doménou. Studentova práce je postupně testována jednotlivými požadavky zmíněné validační domény. Nejprve jsou zkontrolovány obecné vlastnosti, jako je například název a velikost souboru. Poté je připraveno vše potřebné pro samotnou validaci (pracovní adresář, vlákno pro validaci atd.). Následuje spuštění validačního procesu. Výsledek validace je uložen jako HTML soubor na validační server a obratem je též poskytnut studentovi. Tyto HTML soubory budou v bakalářské práci zpracovávány a popisuje je následující kapitola.

Další informace o validačním serveru a podrobnější vysvětlení jeho fungování je možno získat v [12].

3.2 Výstupní soubory validačního serveru

Validační server je umístěn na dedikovaném počítači, který je na adrese `validator.zcu.cz`, jehož souborový systém bude dále popisován.

3.2.1 Umístění výstupních souborů

V adresáři `/opt/valid-common/www/results/` je složka pro výsledky z jednotlivých domén (například `oop_UML08`) [13]. V případě odevzdání alespoň jedné úlohy v daný den, je v této složce vytvořen další adresář, jehož jméno tvoří aktuální datum (například `2015-11-29`). Do tohoto adresáře je ukládán zmíněný HTML soubor obsahující informace o průběhu validace. Celková cesta k němu může tedy vypadat následovně:

```
/opt/valid-common/www/results/oop_uml08/2015-11-29/23-56-18-08_uml.jar.html
```

V názvu souboru je obsažen aktuální čas (zde `23:56:18`) a název odevzdaného souboru (zde `08_uml.jar`). V těchto složkách pojmenovaných pomocí data se tedy nachází záznam o vyhodnocení všech úloh ve formátu HTML, které byly v daný den odevzdány. A to bez ohledu na výsledek validace a studenta, který je odevzdal.

V adresáři `/opt/valid-common/` se dále nachází složka `/data/store/`, ve které se vyskytují podadresáře pro jednotlivé domény (například `oop_uml08`). V nich je opět složka pro každý den, ve kterém byla odevzdána alespoň jedna úloha. V tomto adresáři je vytvořen další podadresář pojmenovaný osobním číslem pro každého studenta, který v tento den odevzdal nějakou úlohu. Všechny odevzdané soubory jedním konkrétním studentem v jedné doméně v daný den jsou tedy uloženy na jednom místě a rozlišeny časem odevzdání. Například: `17-33-22-08_uml.jar`. Tyto soubory nejsou pro vyhodnocování výsledků podstatné a byly by zpracovány pouze v případě nekonzistence výsledků v adresáři `results`.

3.2.2 Struktura výstupních souborů

V každém HTML výstupním souboru validace je obsaženo studijní číslo studenta, jehož řešení úlohy je validováno. Dále se zde nachází název zaslaného souboru, datum a čas validace a výsledek validace. Pozor na skutečnost, že

pokud mají vteřiny hodnotu *00*, tak tato hodnota není zobrazována. Například čas *17:12:00* je zobrazován jako *17:12*. Čas jedna vteřina po páté hodině odpolední je již dle očekávání vypsán jako *17:00:01*.

Dále následuje nadpis *Detailní výsledek validace*, za kterým je individuální výpis průběhu validace. Nejprve jsou reportovány výsledky kontrol obecných vlastností, jako je například správné pojmenování a formát odevzdávaného souboru, či přítomnost nepovolených adresářů. Při úspěšném splnění těchto kontrol následuje výpis průběhu testování samotného odevzdávaného projektu. V případě, že je během validace nalezena chyba, je zde uveden nějaký typ chybového výpisu, například: `Chyba při validaci`. Pokud se jedná o UML domácí úlohu a nástroj *UML test*, který bude popisován v následující kapitole, odhalil chybu, následuje její výpis. Například:

```
t0301012_hasCorrectRelationsCountBetween__IZvyrazneny_Par: UML:
Mezi elementy 'IZvyrazneny' a 'Par' chybi vazba
```

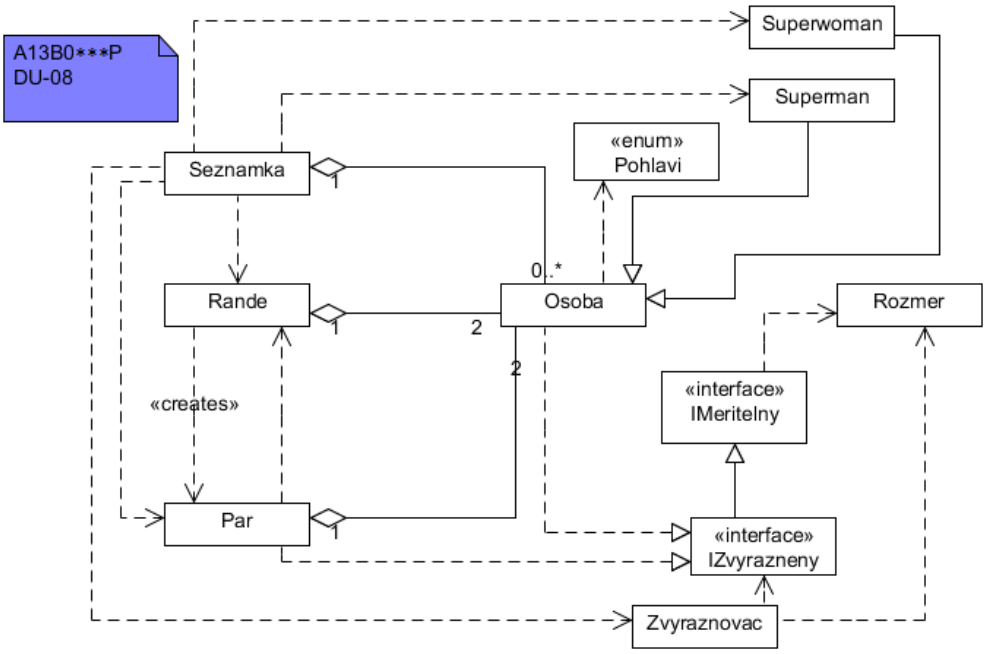
Význam částí tohoto chybového výpisu bude vysvětlen v kapitole 4.2.2 na straně 14.

4 Vyhodnocování UML diagramů

4.1 UML diagram

UML je modelovací jazyk navržený pro grafickou vizualizaci, specifikaci a návrh softwaru. Svými výrazovými prostředky pokrývá celý životní cyklus systému. Hlavním důvodem rozšířenosti tohoto jazyka je, že umožňuje popsat systém tak, aby mu rozuměli programátoři, analytici i zákazníci. V současné době existuje několik druhů UML diagramů, například: diagram tříd, balíčků, aktivit, nasazení atd.

V předmětu KIV/OOP vytvářejí studenti UML diagramy tříd ve čtyřech projektech v nástroji UMLet. Následně je odevzdávají na Portál ve formátu PNG a UXF. Validován je pouze UXF soubor. Na obrázku 4.1 je ukázán jeden reálný UML diagram od studenta, který byl použit jako vstup pro validaci.



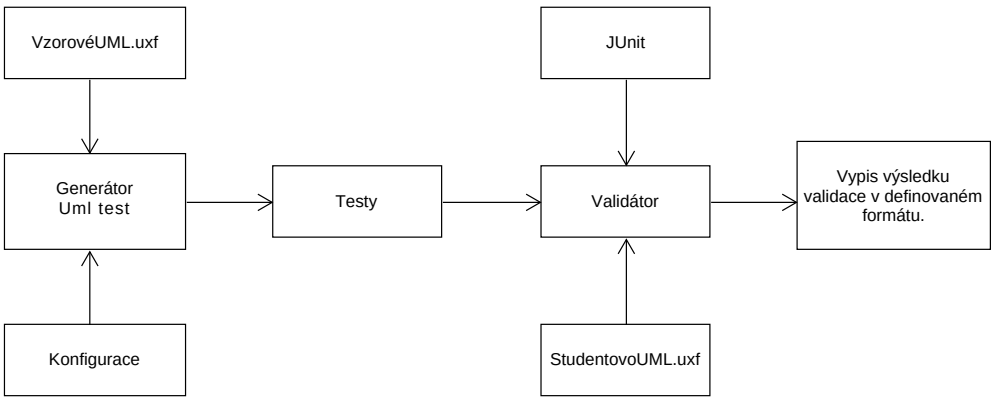
Obrázek 4.1: UML diagram od studenta

4.2 UML test

V rámci jiného studenského projektu byl navržen a implementován systém pro automatické testování UML diagramů [14]. Tento systém umožňuje ze zadaného vzorového UML diagramu ve formátu UXF extrahovat pravidla, která jsou dále použita pro automatické vygenerování *JUnit* testů. Pomocí těchto vytvořených testů se validují diagramy odevzdané studenty.

4.2.1 Způsob konstrukce a vyhodnocování testů

Způsob konstrukce a použití testů je znázorněn na obrázku 4.2. Na vstupu program obdrží vzorový UXF soubor, který obsahuje vzorovou množinu entit s definovanými vlastnostmi a konfigurační soubor. Ten obsahuje množinu kontrolovaných aspektů, ve kterých se vzorové a validované diagramy musí shodovat. Poté je vytvořen seznam pravidel, která musí odevzdané soubory splňovat v podobě velkého množství (řádově 100) testů. Validátor těmito testy, které si lze představit jako klasické *Java* třídy, s využitím *JUnit* otestuje studenské diagramy. Výstupem je jednoznačně a přehledně definovaná chyba, jejíž formát bude vysvětlen v následující sekci.



Obrázek 4.2: Konstrukce a použití automatických testů

Důležité je, že některé testovací metody jsou označeny anotací `@important`, ostatní testy jsou pak na nich závislé. V případě, že taková metoda selže, je testování přerušeno. Tím se zabrání například zbytečnému testování vlastností neexistujícího objektu.

4.2.2 Testovací metody

Každá testovací metoda má své jméno ve specifikovaném formátu. Nejprve je uvedeno *ID* testu ve formátu `t[kategorie][podkategorie][pořadové číslo testu v rámci kategorie]` následuje znak „_“ a poté textový název testu.

Přehled kategorií a podkategorií testů(viz [1]):

- 1 – Testy poznámek
  - 01 – Testy existence jedné poznámky
  - 02 – Testy správného obsahu poznámek
  - 03 – Testy shodnosti obsahu poznámky s názvem UXF souboru
- 2 – Testy tříd
  - 01 – Testy unikátnosti jmen a správného pojmenování elementů
  - 02 – Testy správnosti typu elementů
  - 03 – Testy správného počtu elementů
  - 04 – Testy (ne)obarvení elementu
  - 05 – Testy fontu názvu elementu
  - 06 – Testy, zda elementy nemají dodatečný text
  - 07 – Testy překrývání elementů
- 3 – Testy relací
  - 01 – Testy počtu relací mezi elementy
  - 02 – Testy orientace relací
  - 03 – Testy typu relací mezi elementy
  - 04 – Testy obsahu slepých a nevyužitých relací
  - 05 – Testy počtu relací v diagramu
  - 06 – Testy grafické správnosti
  - 07 – Není využita, číslo zůstává z předchozí verze
  - 08 – Testy, zda diagram neobsahuje nenapojené vazby
  - 09 – Test křížení vazeb

Pokud metoda odhalí chybu, je do výstupu validace vypsán její název, podle kterého je možno metodu zpětně zařadit do jednotlivých kategorií a podkategorií. Například:

```
t0301012_hasCorrectRelationsCountBetween__IZvyrazneny_Par: UML:
Mezi elementy 'IZvyrazneny' a 'Par' chybi vazba
```

Uvedený název znamená, že chybu odhalily testy třetí kategorie (*Testy relací*), konkrétně první podkategorie (*Testy správného počtu relací mezi elementy*). Hodnota `012` udává, že validovaný soubor neprošel 12. testem v kategorii relačních testů. Zbytek názvu tvoří jméno testu a bližší identifikace elementů u kterých byla daná chyba zjištěna. Dále je vypsán typ chyby, což je nejčastěji UML, ale může být i například ERR při chybě ve vstupním souboru. Na závěr je uveden popis chyby.

Identifikátor metody (např.: t0301012), která odhalila chybu v odevzdané úloze, bude dále v této práci použit jako identifikátor UML chyb.

5 Analýza a zpracování výstupu validací

Tím, že jsou na validátoru uloženy postupně všechny, ať již platné, či neplatné studentské pokusy, má vyučující možnost udělat si určitou představu o studentově aktivitě v průběhu celého semestru. To je samozřejmě pouze doplňující informace, protože základní informací je to, zda student úspěšně odevzdal všechny požadované úkoly ve stanovených mezních termínech. Ovšem i tato doplňující informace může být pro vyučujícího důležitá, neboť z ní může usuzovat na to, zda student plní své povinnosti včas, či naopak je nechává na poslední chvíli. Vypovídající hodnotu mohou mít i v případě existence nějakého podezření z plagiátorství. Všechny tyto informace se dají z dat uložených na validátoru získat. Ovšem uvážíme-li situaci, že běžný student odevzdá průměrně 30 prací, je zřejmé, že bude potřeba data nějak přehledně zpracovat a výsledek vizualizovat. Jelikož je důraz kladen na celkový přehled, bude vhodné data reprezentovat pomocí grafů.

Dále je třeba si uvědomit, že pro vyučujícího jsou zajímavé nejen informace o jednotlivých studentech, ale i agregované grafy, které dokáží podat informaci o průběhu zpracování semestrálních projektů v konkrétním akademickém roce. Z těchto informací je možno vyčíst například to, že zadání projektů byla příliš obtížná nebo že studenti nemají pro zpracování konkrétního projektu dostatečné množství informací, času či další podobné skutečnosti.

5.1 Návrh reprezentace dat

Jednotlivé výsledky validací (popsané v kapitole 3.2.2 na straně 10) poskytují značné množství informací o dané úloze. Proto bude vhodné získané poznatky reprezentovat prostřednictvím většího počtu oddělených grafů. Jednotlivé grafy budou zaměřeny na určitou specifickou charakteristiku, kterou budou vizualizovat.

Nejdříve je potřeba definovat, jaké informace z dostupných dat mohou vyučujícího zajímat. Tím se dostáváme k základnímu dělení grafů – z hlediska celého předmětu (agregované grafy) a z hlediska konkrétního studenta (personální grafy).

5.1.1 Návrh agregovaných grafů

V této části je kladen důraz na přehlednou vizualizaci průběhu odevzdávání prací po celý semestr. Nejdříve je nutno si stanovit, které informace jsou zajímavé a proč.

- Stav odevzdávání domácích úloh, ze kterého bude jasně patrné, kolik jednotlivých domácích úkolů již bylo úspěšně validováno a kolik proběhlo pokusů o jejich odevzdání. Tento graf by mohl posloužit jako zpětný ukazatel obtížnosti jednotlivých úloh a studenské aktivity během celého semestru. Během výuky předmětu by také mohl poskytnout vyučujícímu informaci o problémech s odevzdáním nějaké konkrétní úlohy.
- Stav úspěšných odevzdání domácích úloh, který by byl shodný s předešlým grafem, ale neobsahoval by neúspěšné pokusy. Jelikož by byl tento graf zaměřen pouze na úspěšné validace, poskytoval by například rychlý přehled o poměru včasných a opožděných úloh.
- Přehled validovaných úloh, který bude znázorňovat poměr včasných a opožděných validací. Tento graf je velice podobný předešlému, nezohledňuje ovšem neúspěšné validace. Slouží tedy k rychlé orientaci dochvilnosti studentů. Pomocí toho grafu bude možné například vysledovat, zda se mezi domácími úlohami nevyskytuje některá, na kterou by studenti potřebovali více času.
- Hodina odevzdávání domácích úloh. Tento graf bude poskytovat přehled o aktivitě studentů vzhledem k denní době. Výstupem je tedy informace, ve kterých hodinách jsou studenti nejaktivnější. Využití tohoto grafu nemá příliš praktický význam, protože služby validátoru jsou garantovány v režimu 24/7 (kromě profylaxe, která ale neprobíhá v průběhu semestru). Graf spíše ilustruje rozložení studentských aktivit během dne.
- Doba mezi zadáním úlohy a úspěšným odevzdáním úlohy, ze které bude jasně patrné, po kolika dnech studenti na zadané úloze začínají pracovat a za jak dlouho se jim ji podaří zvalidovat. V grafu budou také barevně odlišeny validace provedené v řádném a náhradním (mezním) termínu.
- Stejný graf jako předchozí, ovšem včetně neúspěšných pokusů.

5.1.2 Návrh vizualizace aktivity studenta

Neméně důležitým požadavkem je vizualizovat získaná data o konkrétním studentovi. Nebylo jednoduché tento graf navrhnout, protože má v sobě obsahovat značné množství informací. Mělo by z něj být jasně patrné, kdy a jaké úlohy student odevzdával. Potřeba je také od sebe odlišit úspěšné validace odevzdané v řádném i náhradním termínu a neúspěšné pokusy a to u všech konkrétních projektů. Jelikož se vizualizace vztahuje ke konkrétnímu studentovi, bude vhodné identifikační číslo, které je uvedené přímo v HTML souboru s výstupem validace, doplnit o jméno studenta.

Tento graf bude například poskytovat vyučujícímu při zkoušce rychlý přehled o aktivitě studenta. Podobný informační charakter by mohl mít pro samotného studenta a poskytnout mu zpětnou vazbu na jeho práci během semestru. Mohl by pomoci studentovi vysledovat období, ve kterém má sklony polevovat při plnění školních povinností.

Bylo navrženo několik verzí tohoto grafu a po konzultacích s vedoucím práce byla vybrána jedna z nich, která byla dále iterativně vylepšována.

5.2 Zpracování vstupních souborů

Vstupem požadované aplikace jsou výstupní HTML soubory validačního serveru, které je nutné nejprve nějak zpracovat. Pro parsování byla vybrána knihovna *JSoup* (důvody výběru a stručný popis knihovny se nachází v kapitole 2.1.1 na straně 2). Veškerá požadovaná data, jako je osobní číslo, datum a čas validace, výsledek validace a název odevzdávaného souboru, jsou uložena do paměti. Pro určení, zda byla úloha odevzdána včas, je zpracován také soubor se začátky řešení úloh a řádnými termíny jejich odevzdání. Mezi parametry při spuštění se také nachází CSV soubor se jmény studentů. Detailnější způsob implementace bude popisován v kapitole 6.2 na straně 20.

Program musí mít informaci o tom, kdy byly jednotlivé domácí úlohy zadány a do kdy musí být studenty splněny, a to jak v řádném tak v náhradním termínu. Je třeba vzít též v úvahu, že projektů je v jednom předmětu větší množství (10 až 15). Takže tato informace o termínech odevzdání nepředstavuje „jenom dvě data“, ale jedná se o množství strukturovaných dat.

Z tohoto důvodu byla navržena následující hierarchie tohoto konfigurač-

ního souboru: na jednom řádku identifikační číslo úlohy a na následujících řádcích se vyskytuje datum začátku řešení a řádný termín odevzdání pro jednotlivé roky. Obě data jsou oddělena středníkem. Mezi jednotlivými úlohami je vložen prázdný řádek. Ukázka části toho souboru:

```
DU-01
2012-09-26;2012-10-03
2013-09-25;2013-10-02
2014-09-24;2014-10-01
2015-09-23;2015-09-30

DU-02
2012-10-03;2012-10-10
2013-10-02;2013-10-09
2014-10-01;2014-10-08
2015-09-30;2015-10-07

DU-03
2012-10-10;2012-10-17
2013-10-09;2013-10-16
2014-10-08;2014-10-15
2015-10-07;2015-10-15
```

Protože ve zpracovávaných HTML souborech je použit jako identifikátor studenta jeho osobní číslo, bude potřeba zjistit i jméno studenta a s osobním číslem jej spárovat. Spárování osobního čísla se jménem studenta je důležité pro vytváření personálních grafů, kde je jméno studenta uvedeno. Samozřejmě je dbáno na ochranu soukromí studentů a tyto grafy slouží pouze pro potřeby vyučujícího (případně mohou být poskytnuty studentovi, k němuž se vztahují) a nebudou v této podobě nikde zveřejňovány.

Ke zmíněnému spárování slouží vstupní soubor aplikace, který získá vyučující ze STAGu. Jsou v něm uvedena jména studentů a jejich osobní čísla ve formátu *.csv*. Soubor je pojmenován vždy rokem, ke kterému se vztahuje. V prvním sloupečku jsou uvedena osobní čísla jednotlivých studentů, druhý sloupeček obsahuje jména a třetí sloupeček příjmení studentů. Dále následují další sloupce, které nebudou mít pro tuto práci žádný význam a nebudou nijak zpracovávány.

6 Realizace a funkčnost navržené aplikace

V této kapitole budou popisovány konkrétní způsoby implementace celé aplikace. UML diagram vytvořených tříd je uveden v příloze A.

Během vývoje byly jednotlivé třídy průběžně testovány jednotkovými testy, viz podrobněji kapitola 8 na straně 29.

6.1 Kořenový balík *bp*

V hlavním balíku programu je samozřejmě umístěna třída *Hlavni*, ve které se vyskytuje metoda *main* aplikace. Nejprve je ověřena správnost argumentů předaných na příkazové řádce včetně existence vstupních souborů. Význam a formát parametrů je uveden v příloze *Uživatelská příručka*. Následně je vyvolán konstruktor třídy *PripravaDat*, ve které jsou načtena všechna dostupná data včetně konfiguračních souborů. Poté jsou dle požadavků na vstupu volány příslušné třídy pro tvorbu grafů.

Dále je zde umístěna třída *Konstanty*, která slouží pro uchování všech globálních konstant. Vyskytují se zde především konstanty pro tvorbu celkového grafu jednotlivých odevzdání, textové popisky a názvy všech grafů. Také obsahuje regulární výraz odpovídající všem osobním číslům studentů a názvy i identifikační čísla jednotlivých domácích úloh. Právě prostřednictvím této třídy může poučený uživatel konfigurovat výstup aplikace.

Poslední komponentou hlavního balíku je výčtový typ *MozneVysledky-Validace*, který obsahuje, jak již název napovídá, všechny možné výsledky validací.

6.2 Balík pro čtení dat

V této části budou popisovány třídy z balíku *bp.nactenidat*, které, jak již název napovídá, slouží pro načtení dat ze vstupních souborů do paměti.

6.2.1 VysledekValidaceSouboru

Entitní třída *VysledekValidaceSouboru* reprezentuje veškeré informace o konkrétní validaci. Jedná se o údaje načtené z HTML souborů (osobní číslo, název úlohy, výsledek validace a datum a čas validace) rozšířené o informace získané z konfiguračních souborů (jméno studenta, začátek řešení úlohy a termín odevzdání úlohy). Rovněž obsahuje identifikační číslo domácí úlohy, které je určeno ve třídě *PripravaDat*. Dle převodní tabulky uvedené ve třídě *Konstanty* je podle názvu odevzdané domácí úlohy nastaveno příslušné identifikační číslo.

6.2.2 PripravaDat

PripravaDat nejprve vytvoří mapy se začátky a řádnými termíny odevzdání domácích úloh a se jmény studentů pro jednotlivé roky. Poté je rekurzivně procházen vstupní adresář s HTML soubory, které jsou parsovány pomocí knihovny *JSoup* (viz: kapitola 2.1.1 na straně 2). V případě, že získané osobní číslo odpovídá regulárnímu výrazu uvedenému v *Konstanty*, je z vytvořených map získáno jméno studenta, který soubor odevzdal, a datum začátku řešení i termín odevzdání domácích úloh. Dle převodní tabulky uvedené ve třídě *Konstanty* je podle názvu odevzdané domácí úlohy nastaveno příslušné identifikační číslo. Poté je zavolán konstruktor třídy *VysledekValidaceSouboru*. Nejdůležitějším prvkem této třídy je mapa:

```
TreeMap<Integer, ArrayList<VysledekValidaceSouboru>> vysledky =
 new TreeMap<Integer, ArrayList<VysledekValidaceSouboru>>();
```

Klíčem zmíněné mapy je kalendářní rok, ve kterém studenti úlohy odevzdávali, a hodnotou je *ArrayList* s instancemi třídy *VysledekValidaceSouboru*. Prvky tohoto *ArrayListu* tedy tvoří veškeré dostupné informace o jednotlivém odevzdání úlohy.

6.3 Balík pro agregované grafy

V kapitole 5.1.1 na straně 17 bylo zdůvodněno, proč bude přínosné vytvořit více druhů grafů vždy pro dané požadavky. Toto opatření je nutné, neboť

chceme zohlednit množství informací, jako je identifikační číslo úlohy, často také den, ve kterém byla odevzdána, počet pokusů atd.

Pro tvorbu agregovaných grafů byla zvolena knihovna *JFreeChart*. Popis této knihovny se nachází v sekci 2.2.1 na straně 4. Nyní budou popisovány jednotlivé třídy pro tvorbu požadovaného výstupu umístěné v balíku *bp.agregovanegrafy*.

6.3.1 AktivitaG

Parametry konstruktoru této třídy jsou: požadovaný titulek grafu, mapa *vysledky* s výsledky validací všech studentů v jednom roce, tento rok, výstupní adresář, do kterého mají být grafy tvořeny, a identifikační číslo úlohy, které udává, jaké domácí úlohy budou nyní vizualizovány. Pokud by měl být vizualizován celý průběh semestru všech domácích úloh předmětu, je tento parametr nastaven na hodnotu *null*. Nejprve je vytvořena mapa:

```
TreeMap<Integer, Integer> hodinaOdevzdani =
 new TreeMap<Integer, Integer>();
```

Klíč tvoří 24 hodnot, které představují konkrétní hodinu, a hodnotou je počet všech pokusů o odevzdání právě v tuto hodinu. Nejprve je procházena celá vstupní mapa *vysledky*, u jednotlivých jejích hodnot je zjištěna hodina validace a následně připočtena jednička na příslušné místo v mapě *hodinaOdevzdani*. Dle hodnot v *hodinaOdevzdani* je poté vykreslen sloupcový graf metodou `draw(Graphics2D g, Rectangle2D r)` knihovny *JFreeChart*. Na závěr je výsledek uložen ve formátu PNG do výstupního adresáře, pod zpracovávaný rok do složky *denni-doba-pokusu*. V případě neexistence této cesty jsou potřebné adresáře vytvořeny.

6.3.2 DnuOdZadaniG

Parametry konstruktoru této třídy opět tvoří titulek grafu, rok, výsledky validací v tento rok, *vysledky*, výstupní adresář a navíc ještě přepínač *jenUspesne*. Tato třída je totiž používána ke tvorbě dvou druhů grafů. Pokud je hodnota přepínače nastavena na *true*, jsou zpracovávány jen úspěšné validace. V opačném případě vizualizujeme i neúspěšné pokusy.

Data uložená ve *vysledky* je opět nutné přetransformovat. K tomuto účelu slouží následující mapy:

```
TreeMap<Integer, Integer> denOdZadaniKO =
 new TreeMap<Integer, Integer>();

TreeMap<Integer, Integer> denOdZadaniOKvcas =
 new TreeMap<Integer, Integer>();

TreeMap<Integer, Integer> denOdZadaniOKpozde =
 new TreeMap<Integer, Integer>();
```

Klíčem je vždy počet dní, který uplynul od zadání dané domácí úlohy. Hodnotou je pak počet úspěšných včasných/opožděných odevzdání či neúspěšných pokusů o validaci po těchto dnech.

Vytvořené mapy jsou poté s barevnou odlišností vizualizovány pomocí knihovny *JFreeChart* do grafu. Výstup je uložen do výstupního adresáře pod příslušný rok, do složky `den-od-zadani-ok` (přepínač *jenUspesne* byl nastaven na hodnotu *true*) nebo do `den-od-zadani-vse` (v opačném případě).

6.3.3 StavOdevzdaniG

Na stejném principu jako dvě předešlé funguje i třída *StavOdevzdaniG*. Parametry konstruktoru tvoří požadovaný titulek grafu, rok, výsledky validací v tento rok, přepínač *jenUspesne* a výstupní adresář. Zmíněný přepínač napovídá, že i tato třída produkuje grafy dvou typů. Výstupem je přehled odevzdání všech pokusů o odevzdání a přehled úspěšných validací. Nejprve jsou vytvořeny následující mapy:

```
TreeMap<String, Integer> neuspesnePokusy =
 new TreeMap<String, Integer>();

TreeMap<String, Integer> uspesneVcasne =
 new TreeMap<String, Integer>();

TreeMap<String, Integer> uspesneOpozdene =
 new TreeMap<String, Integer>();
```

Klíčem je řetězec odpovídající identifikačnímu číslu úlohy a hodnotou je počet úspěšných včasných/opožděných odevzdání či neúspěšných pokusů o validaci této úlohy. Mapy jsou opět pomocí *JFreeChart* přeneseny do grafu a odlišeny barevně. Výsledek je uložen do výstupního adresáře pod příslušný rok do složky `stav-odevzdani`.

6.4 Balík pro tvorbu personálních grafů

Vzhledem k velkému množství vizualizovaných informací a jejich rozmanitosti bylo rozhodnuto vytvořit vlastní typ grafu přesně pro tento účel. Pro samotné vykreslování je vytvořena speciální třída, která používá knihovnu základních grafických prvků (viz 2.2.2 na straně 4).

V této části budou popisovány třídy pro tvorbu personálních grafů z balíku *bp.celkoveodevzdani*.

6.4.1 DataStudentuVRoce

Na rozdíl od předešlého případu, kdy jsou veškerá data zpracovávána z globálního pohledu celého semestru, tento graf je zaměřen vždy na konkrétního studenta. Proto bylo nutné načtená data poněkud transformovat. K tomuto účelu slouží právě tato třída.

Nejprve je mapa, předaná v konstruktoru (*vysledky*) obsahující výsledky validací všech studentů v daném roce, transformována do následující mapy (*vysledkyStudenta*):

```
TreeMap<String, ArrayList<VysledekValidaceSouboru>> vysledkyStudenta =
 new TreeMap<String, ArrayList<VysledekValidaceSouboru>>();
```

Klíč tvoří osobní číslo studenta a hodnotou jsou všechny výsledky validací tohoto studenta.

Dále je pro každého studenta vytvořena mapa *dnyAodevzdavani*, do které je zaznamenáno, kolik dní se pokoušel jednotlivé úlohy odevzdávat. V neposlední řadě také mapy:

```
TreeMap<String, TreeMap<LocalDate, Integer>> ulohaDenPocet=
 new TreeMap<String, TreeMap<LocalDate, Integer>>();

TreeMap<String,TreeMap<LocalDate,Int>>ulohDenPocSprav=
 new TreeMap<String, TreeMap<LocalDate, Int>>();

TreeMap<String, TreeMap<LocalDate,Integer>> ulohaDenPocetSpravne=
 new TreeMap<String, TreeMap<LocalDate, Integer>>();
```

Klíčem je samozřejmě identifikační číslo úlohy a hodnotou počet úspěšných/neúspěšných pokusů. Aby nebylo nutné vytvářet speciální mapu pro včasné a opožděné validace, jsou počty úspěšných opožděných odevzdání vynásobené konstantou *-1*. Toto opatření lze provést, neboť v jeden konkrétní den nemůže být konkrétní úloha zároveň odevzdána včas a zároveň pozdě.

6.4.2 VykresliGraf

V této třídě je vykresleno pozadí: bílé plátno (rozměry udány ve třídě *Konstanty*), osy x a y. Podle mapy *dnyAOdevzdavani* jsou vykresleny šedivé obdélníky, které budou tvořit pozadí každé druhé úlohy, a to tak, aby velikost bílých i šedivých částí odpovídala počtu dní, kdy byla daná úloha odevzdávána. Smyslem je zvýšit přehlednost tvořeného grafu. Popisky osy y určuje proměnná *maxPocetOdevzdaniZaDen*, ve které je uložen maximální počet odevzdání jedné úlohy, která student provedl v jeden den.

Následně jsou dle mapy *ulohaDenPocet* vykresleny žluté obdélníky reprezentující celkový počet neúspěšných odevzdání a na osu x připsán datum těchto pokusů. Klíč zmíněné mapy tedy určuje souřadnici y – datum odevzdání a hodnota udává počet odevzdání v tento den. Na závěr jsou vykresleny obdélníky dle absolutní hodnoty mapy *ulohDenPocSprav*, které představují úspěšné validace. Zeleně jsou označeny práce odevzdané v řádném termínu, červeně pak v termínu náhradním. Klíč mapy opět určuje datum validace a hodnota mapy udává počet validací v tento den. Šířka vykreslovaných obdélníků je udaná ve třídě *Konstanty*. Výška je spočítaná pomocí hodnot počtu odevzdaných pokusů vynásobených proměnnou *vyskaJednotkovehoPokusu*. Ta je spočítána jako podíl výšky bílého pozadí grafu a proměnné *maxPocetOdevzdaniZaDen*.

Výsledný graf je uložen do výstupního adresáře, pod příslušný rok do složky `celkovy-graf`.

7 Analýza a zpracování dat získaných nástrojem *UML test*

Nyní se zaměříme na data získaná nástrojem *UML test* popsaná v kapitole 4 na straně 12.

7.1 Návrh interpretace dat

Nejdříve bude provedena analýza, které informace z dat jsou zajímavé, a následně bude navržena realizace, ovšem jen u těch případů, u kterých je to možné a nebo to má smysl.

- Agregované chyby přes jednotlivé domácí úlohy by bylo vhodné reprezentovat pomocí sloupcového grafu. Na ose x by se vyskytovaly identifikátory UML chyb a na ose y počet výskytů těchto chyb. Vyučující předmětu by tak mohl jednoduše získat přehled o výskytu jednotlivých chyb v domácí úloze. V případě enormního výskytu konkrétní chyby by pak mohl učinit opatření k zlepšení výkonu studentů. S realizací tohoto grafu by neměl být žádný problém.

- Velice nápomocným ukazatelem by mohl být graf výskytu jednotlivých chyb a jejich skupin v čase. Poskytoval by informaci o tom, jak se mění počet výskytů konkrétních (skupin) chyb v průběhu semestru. Očekávaným výsledkem by bylo postupné ubývání chyb v průběhu odevzdávání jednotlivých projektů, což by značilo, že se studenti postupně zlepšují a učí vytvářet správné UML diagramy. V případě stagnace výskytu nějaké chyby by se mohl vyučující na daný problém zaměřit a pokusit se znovu vysvětlit studentům správný postup.

  Z analýzy dat získaných nástrojem *UML test* ale bohužel vyplývá, že tento graf není realizovatelný. Totožné identifikátory chyb mají v různých úlohách různý význam. Drobnější odlišnosti se ojediněle vyskytují dokonce i v rámci jedné domácí úlohy. Například bylo odhaleno toto nejednoznačné použití identifikátorů:

  ```
  t0301006_hasCorrectRelationsCountBetween_IZvyrazneny_Osoba
  t0301006_hasNoRelationsBetween_IZvyrazneny_Osoba
  ```

7.2 Zpracování výstupu validací

Vstupem požadované aplikace jsou podobně jako v předchozí aplikaci HTML výstupní soubory validačního serveru. Parsovat se bude obdobně jako v předchozím případě pomocí knihovny *JSoup*. Do paměti je opět potřebné načíst údaje z HTML souborů, tentokrát navíc rozšířené o popis HTML chyb. Jelikož budou vizualizovány pouze četnosti jednotlivých chyb, není potřeba zpracovávat výsledky úspěšných validací a výsledky s jinou než HTML chybou. Rovněž nebudou potřeba žádné konfigurační soubory s časem odevzdání nebo se jmény studentů.

7.3 Realizace navržené aplikace

Celý projekt je kvůli přehlednosti a budoucí rozšiřitelnosti koncipován velice podobně jako předchozí aplikace. UML diagram vytvořených tříd projektu se nachází v příloze A.

V současné době by nebylo nutné z HTML souborů například načítat datum a čas validace, do budoucna je však možné, že se objeví další požadavky na vizualizaci. Z tohoto důvodu je také v hlavním projektu aplikace implementován výčtový typ *MozneVysledkyValidace*, třebaže v aktuální verzi bude zpracováván pouze výsledek *Špatné výsledky*.

V hlavním balíku se také nachází třída *Konstanty* a *Hlavni* s *main* aplikace. Ve stejném duchu jako v předchozím programu je v hlavní třídě nejprve ověřen vstup od uživatele a poté zavolán konstruktor třídy *PripravaDat*, ve které je vytvořena mapa:

```
TreeMap<Int , ArrayList<VysledekValidaceUML>>chybneVysledkyUML =
 new TreeMap<Int , ArrayList<VysledekValidaceUML>>();
```

Klíčem je rok odevzdávání a hodnotou je seznam výsledků validací s UML chybou. Získaná mapa je poté předaná jako parametr instanci třídy *ChybyUmlHistogram*

7.3.1 Načtení dat

Stejně jako v předchozím případě jsou nejprve v zadaném adresáři rekurzivně nalezeny HTML soubory a z nich pomocí *JSoup* extrahovány potřebné informace. Pokud osobní číslo odevzdávajícího odpovídá regulárnímu výrazu, který reprezentuje osobní čísla studentů, a byla nalezena UML chyba, je vyvolán konstruktor třídy *VysledekValidaceUML* s parametry: osobní číslo odevzdávajícího, název odevzdaného souboru, identifikační číslo úlohy (které je jako v předchozí aplikaci nastaveno pomocí převodní tabulky uvedené v *Konstanty*), čas a datum validace, řetězec s výsledkem validace, přepínač *obecnaChyba* (nyní je vždy nastaven na *false* a signalizuje, že se v úloze nevyskytuje jiná než UML chyba) a řetězec s UML chybou. Instance této třídy je uložena jako hodnota do mapy *chybneVysledkyUML* (popsané v kapitole 7.3 na straně 27) pod klíčem, který reprezentuje rok odevzdání.

7.3.2 Tvorba grafu

V balíku *uml.tvorbaGrafu* se nachází třída *ChybyUmlHistogramUml*, která slouží pro vykreslení navrženého grafu. V konstruktoru přijme jako své parametry požadovaný titulek, identifikační číslo úlohy, pro kterou má být výsledek zpracováván, výstupní adresář a mapu *vysledkyUML* s výsledky validací. Nejprve je vytvořena mapa:

```
TreeMap<String, Integer> umlChyby =
 new TreeMap<String, Integer>();
```

Klíčem je identifikátor UML chyby a hodnotou počet všech výskytů této konkrétní chyby během odevzdávání dané domácí úlohy.

8 Testování aplikací

Navržené a implementované aplikace je samozřejmě nutné otestovat a ověřit tak jejich funkčnost. Entitní třídy obou programů byly otestovány pomocí JUnit testů a správnost výstupu byla ověřena funkčním testováním. Všechny implementované testy se nacházejí v adresáři *testy* příslušného projektu.

Dále byly výstupy programu porovnávány s daty uloženými na Portálu. Takto byl například kontrolován počet pokusů o odevzdání konkrétní úlohy v jeden den či data jednotlivých odevzdávání. Veškeré nalezené rozdíly mezi produkty aplikací a dat uložených na Portálu se podařilo logicky vysvětlit. Jednalo se například o rozdílné zpracování vícenásobného úspěšného odevzdání jedné úlohy.

8.1 Testování pomocí JUnit

Ke třídám *VysledekValidaceSouboru* a *VysledekValidaceUML* byly implementovány jednotkové testy. Ověřena je existence instance těchto tříd i všech jejich atributů. Realizovány byly též testy rovnosti, logické shody a obsahu referenčních proměnných.

8.2 Funkční testování

Protože je první realizovaná aplikace poměrně rozsáhlá a čerpá ze značného množství vstupních dat, ověřovala by se správnost zpracování reálných dat poměrně složitě. Pro tento účel byla navržena třída *TestCelkovehoOdevzdani*, ve které jsou nejprve inicializovány všechny proměnné programu, které by byly načtené ze vstupních souborů, na konkrétní hodnoty. Zároveň je k těmto konkrétním hodnotám nakreslen požadovaný výsledek mimo tento program (například v grafickém editoru). Na závěr je zkontrolován výstup programu s nakresleným požadovaným výsledkem. Jelikož se oba obrázky přesně shodují, lze předpokládat, že program pracuje správně.

8.3 Scénáře funkčního testování

Podobně jako v předchozí sekci, byla pro tento účel připravena speciální vstupní data, u nichž lze snadno určit správný výstup programu. Předchozí případ testoval pouze správné vytvoření a vykreslení personálního grafu. Otestovat je nutné také samotné načítání a parsování HTML souborů i tvorbu ostatních grafů.

Pro tento účel byly vytvořeny adresáře obsahující pouze menší množství HTML dat, aby bylo možné výsledek jednoduše ověřit. Zároveň je žádoucí, aby tyto soubory byly co nejvíce rozmanité. To znamená, že musí obsahovat všechny možné výsledky validací a zahrnovat i ty, které nebyly vytvořeny studentskými validacemi ale například testovacím nástrojem *webmodule*.

Program se vždy musí chovat korektně a poskytovat logicky správné výsledky. Otestován byl také případ, kdy chybí informace o jméně studenta, datu zadání a řádném i mezním odevzdání domácího úkolu. V takovém případě jsou všechny úspěšné validace považovány za včasné.

Jelikož se jedná o nástroj, kterým si může funkčnost programu snadno ověřit i koncový uživatel, bude způsob spouštění těchto testů popisován v příloze *Uživatelská příručka* (B.6).

9 Dosažené výsledky

Při generování grafů ze všech dostupných dat bylo v první aplikaci zpracováno 12 918 a v druhé 3 406 HTML souborů.

9.1 Přehled vytvořených grafů

V této části budou postupně předvedeny a popsány všechny druhy vytvořených grafů.

9.1.1 Stav úspěšného odevzdávání

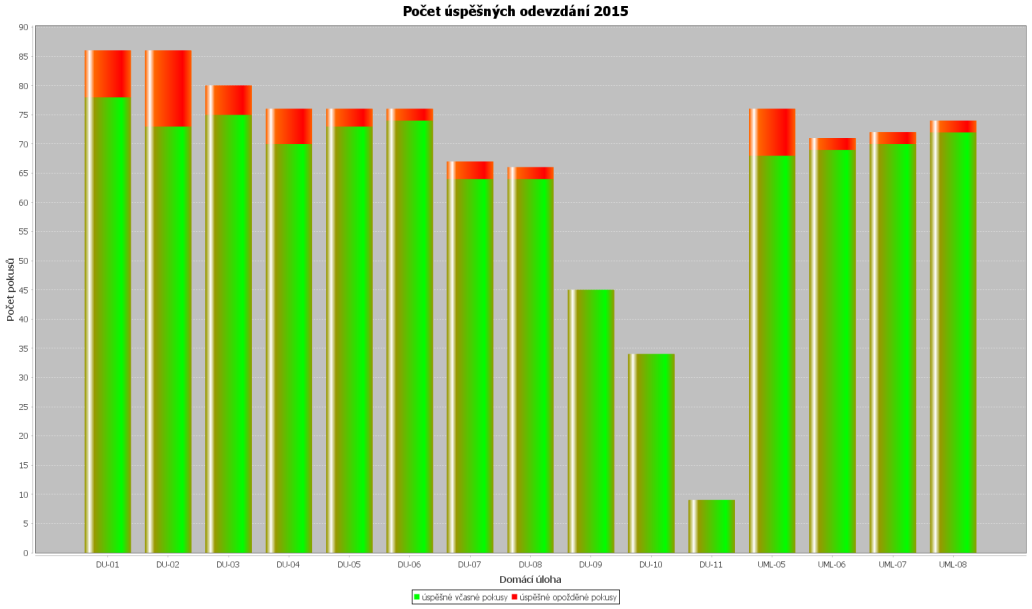
Na obrázku 9.1 je uveden graf, který znázorňuje stav úspěšného odevzdání domácích úloh. Na ose x se vyskytují identifikátory jednotlivých domácích úkolů, osa y značí počet úspěšných odevzdání. V grafu jsou barevně odlišeny včasné a opožděné validace. Zelená představuje validace provedené v řádném termínu a červená ty zpožděné, odevzdané až v termínu náhradním. Graf se vždy vztahuje k jednomu konkrétnímu roku, který je uveden v názvu.

Graf je uložen ve formátu PNG pod názvem: `uspesne-odevzdane-DU_` doplněným o rok, ke kterému se graf vztahuje.

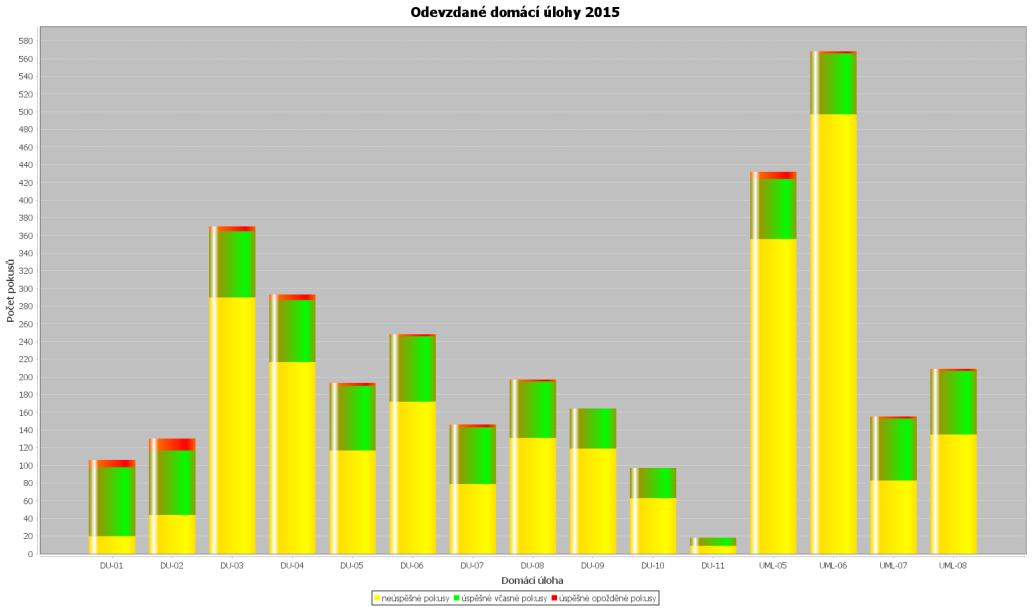
9.1.2 Stav odevzdávání

Obrázek 9.2 představuje velice podobný graf předešlému. Rozdíl spočívá v tom, že osa y značí všechny odevzdání studenských projektů. Zahrnuje tedy kromě úspěšných validací také neúspěšné pokusy o odevzdání domácích úloh, které jsou vyznačeny žlutě.

Vytvořený graf je uložen pod názvem: `odevzdane-DU_` doplněným o rok, ke kterému se vztahuje, stejně jako v předchozím případě do PNG souboru.



Obrázek 9.1: Graf stav úspěšného odevzdání



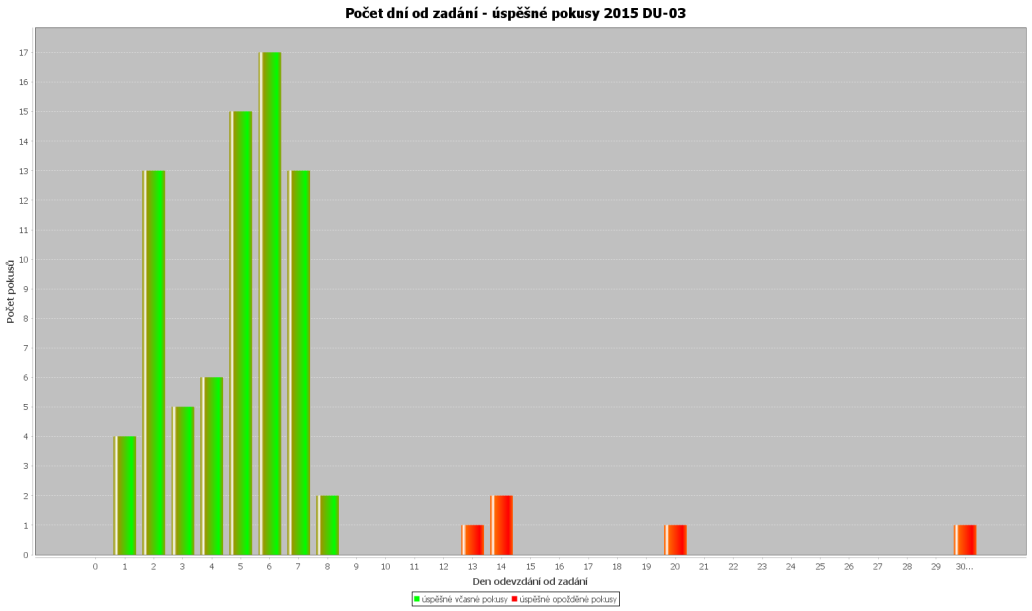
Obrázek 9.2: Graf stav odevzdávání

9.1.3 Počet dní od zadání – úspěšné pokusy

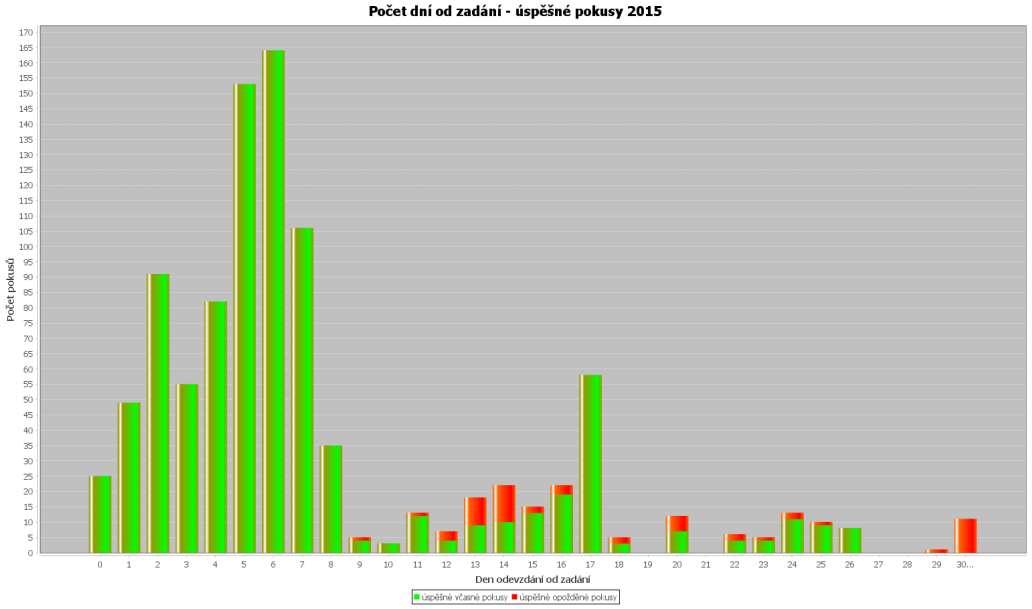
Ukázka tohoto grafu pro jednu konkrétní úlohu je uvedena na obrázku 9.3 a pro celý semestr na obrázku 9.4. Osa x představuje kolikátý den od zadání byla úloha zvalidována. Nula značí, že byla úloha odevzdána v den zadání. V případě dosažení a překročení hraniční hodnoty *30* je odevzdání zahrnuto pod hodnotu *„30..“*. Osa y představuje počet těchto validací. Barevně jsou od sebe odlišeny včasné (zeleně) a opožděné (červeně) validace.

Je zřejmé, že v případě grafu pro jednotlivé domácí úlohy se nemohou střídat červené a zelené sloupečky, nebot' od určitého dne (překročení řádného termínu odevzdání) se jedná již jen o opožděné validace. Toto však neplatí pro druhý z grafů (9.4), nebot' znázorňuje odevzdávání všech domácích úloh a počet dní od zadání úkolu do mezního termínu jeho splnění se může pro jednotlivé domácí úlohy lišit. Ostatně z uvedeného grafu je patrné, že tomu tak skutečně je.

Výstup je uložen pod identifikujícím názvem složeném z řetězce „vse“, roku, ke kterému se vztahuje a identifikačního čísla úlohy, pokud se jedná o jednotlivé projekty. V případě, že graf reprezentuje odevzdávání v celém semestru, je identifikátor nahrazen řetězcem „celkovy“.



Obrázek 9.3: Graf počet dní od zadání – úspěšné pokusy – jedna konkrétní úloha



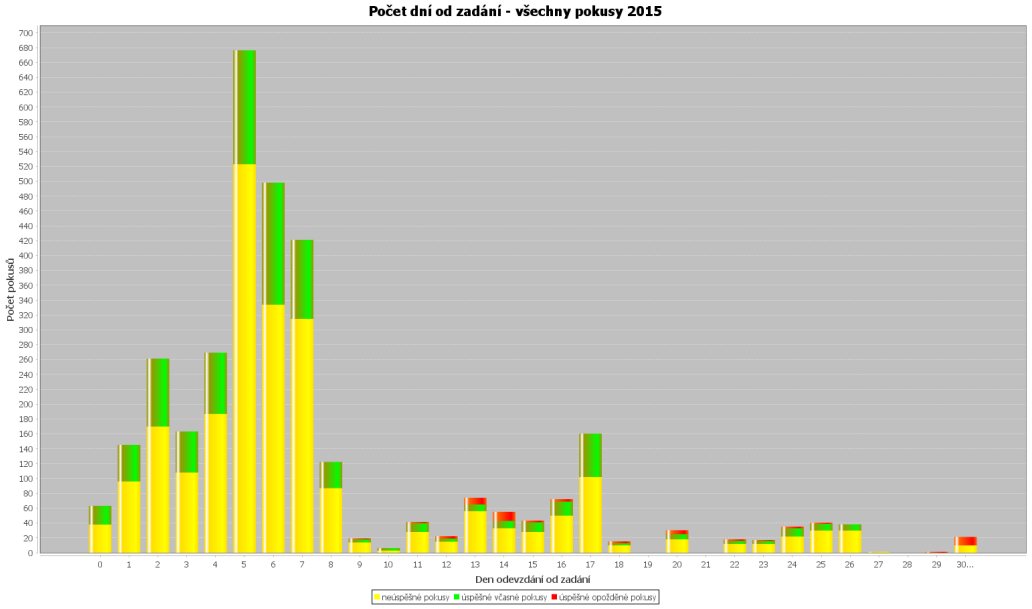
Obrázek 9.4: Graf počet dní od zadání – úspěšné pokusy – celý semestr

9.1.4 Počet dní od zadání – všechny pokusy

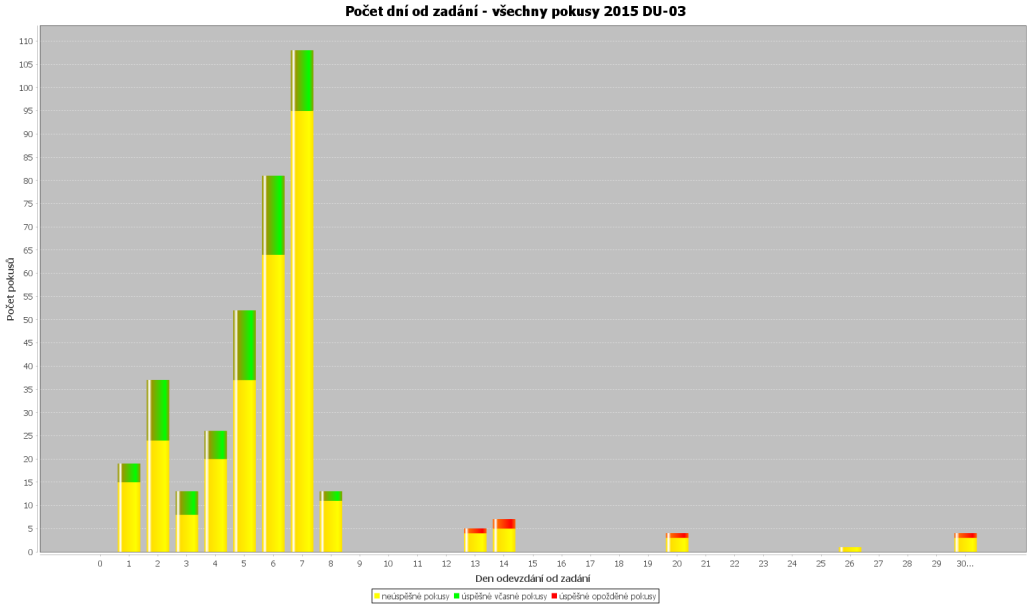
Podobně jako v předešlém případě jsou zde i pro tento druh grafu uvedeny dva ukázkové případy. Pro konkrétní domácí úlohu obrázek 9.5, pro celý semestr obrázek 9.6. Tyto grafy jsou podobné grafům z předchozí sekce, navíc však obsahují neúspěšné validace. Osa y tedy znázorňuje počet všech pokusů o odevzdání. Neúspěšné validace jsou v grafech znázorněny žlutě.

Oba grafy 9.5 i 9.6 se vztahují ke třetí domácí úloze v roce 2015, proto by jejich červené a zelené sloupečky měly být shodné, pouze jeden z nich je doplněn o žluté sloupečky. Vizuální kontrolou grafů lze jednoduše ověřit, že tomu tak skutečně je. Totéž samozřejmě platí pro obrázky 9.3 a 9.4.

Vytvořený graf je uložen stejně jako v předešlém případě, pouze řetězec „ok“ nahradí text „vse“.



Obrázek 9.5: Graf počet dní od zadání – všechny pokusy – jedna konkrétní úloha



Obrázek 9.6: Graf počet dní od zadání – všechny pokusy – celý semestr

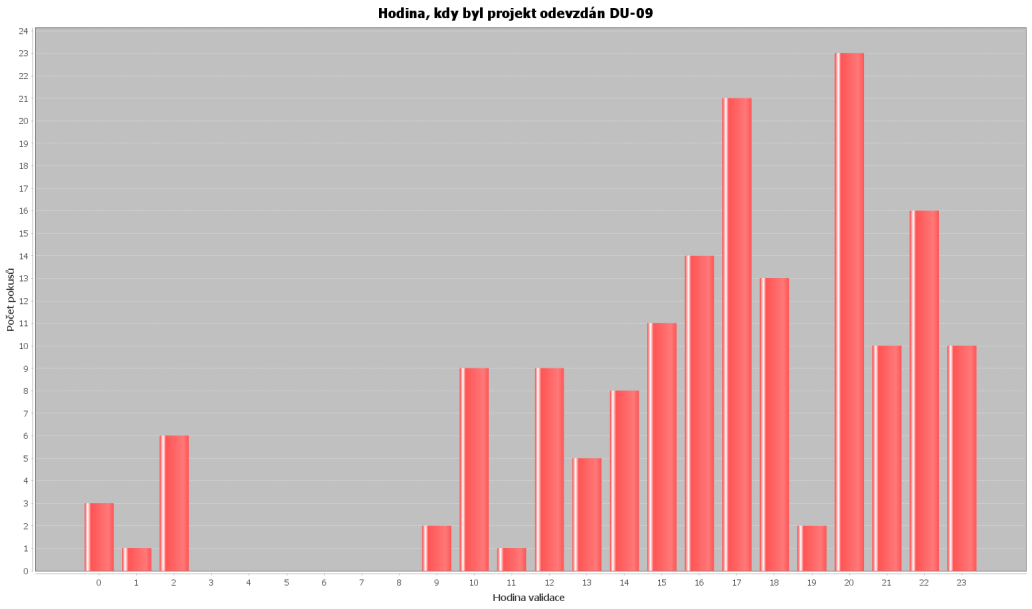
9.1.5 Denní doba pokusu

Přehled aktivity studentů během dne lze získat z grafu uvedeného na obrázku 9.7. Osa x značí jednotlivé hodiny, na ose y se vyskytuje počet odevzdání

jednotlivých domácích úloh pro konkrétní projekt. Tento graf je rovněž vyhotoven i společně pro všechny domácí úlohy.

Zajímavé je, že pro jednotlivé domácí úlohy se rozvržení aktivity během dne poměrně liší. Naopak celkové grafy aktivity v jednotlivých letech jsou už velice podobné a je z nich jasně patrné, že studenti odevzdávají domácí úkoly nejvíce ve večerních hodinách.

Graf je uložen do PNG souboru pojmenovaným rokem, ke kterému se vztahuje, a identifikačním číslem konkrétní úlohy nebo opět řetězcem „celkovy“ pro případ, že se jedná o souhrn všech projektů.

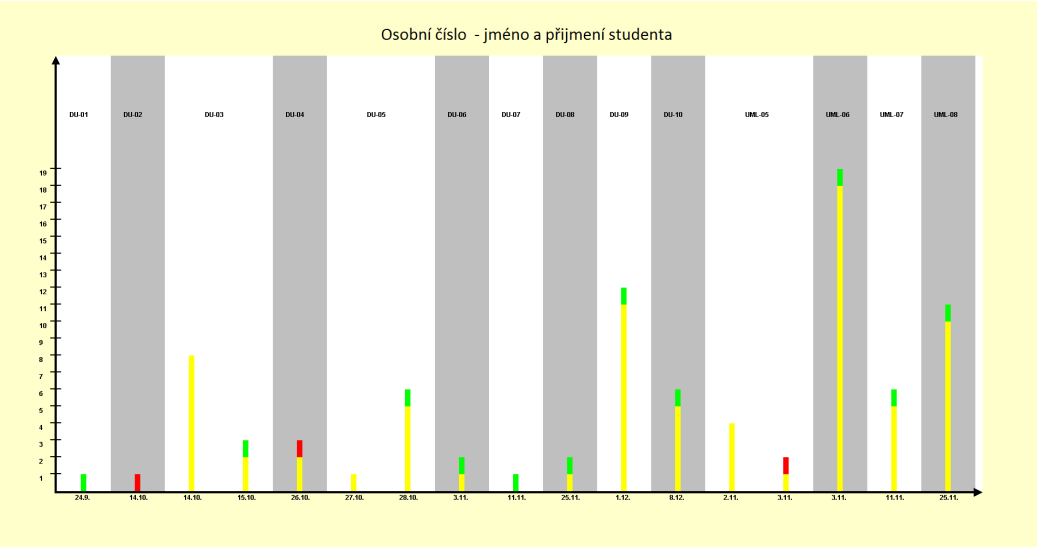


Obrázek 9.7: Graf denní doba pokusu

9.1.6 Personální grafy

Příklad celkového grafu jednotlivých odevzdávání je uveden na obrázku 9.8. Kvůli ochraně soukromí studentů je osobní číslo a jméno studenta nahrazeno řetězcem „Osobní číslo – jméno studenta“. Na ose x je znázorněn datum odevzdávání, v horní části grafu se také vyskytují popisky představující identifikátory jednotlivých úloh. Na ose y jsou vyznačeny jednotlivé počty odevzdání. V grafu jsou barevně odlišena včasná/opožděná úspěšná odevzdání

a neúspěšné pokusy. Zelená značí validace provedené v řádném termínu, červená v termínu náhradním a žlutá představuje neúspěšné pokusy o validaci. Měřítko obou os je vypočítáváno relativně vzhledem k počtu dnů, ve kterých student úlohy odevzdával, a maximálnímu počtu odevzdání v jeden konkrétní den.



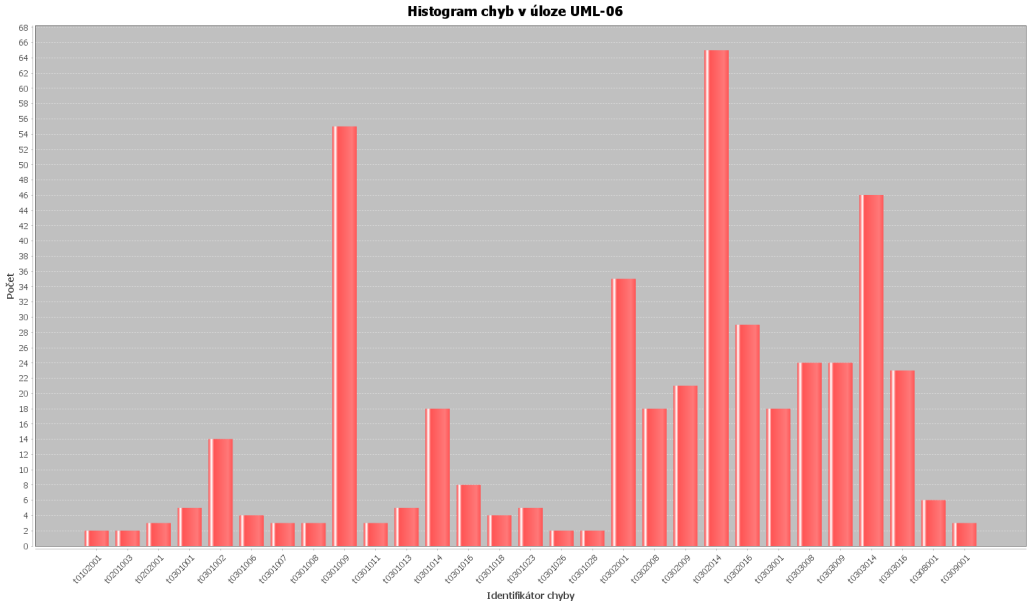
Obrázek 9.8: Celkový graf jednotlivých odevzdávání

Výstup je uložen ve formátu PNG pod názvem skládajícím se z osobního čísla studenta a celkového počtu pokusů o odevzdání všech úloh tohoto studenta. Například *A13B0123P-20.png*. Tento poněkud nezvyklý název dovoluje vyučujícímu už jen z názvu souboru vybrat grafy aktivity studentů, které jsou zvláštní buď malým, nebo naopak velkým počtem pokusů.

9.1.7 Histogram výskytu UML chyb

Obrázek 9.9 ilustruje histogram výskytu UML chyb, který je na rozdíl od předešlých grafů produktem druhé aplikace. Na ose x jsou identifikátory, které představují jednotlivé UML chyby (vysvětlené v kapitole 4.2 na straně 13). Osa y udává počet výskytů těchto chyb.

Pokud by v projektu bylo příliš velké množství druhů UML chyb, je využita konstanta minimálního počtu výskytů chyb (v současné době je nastavena na hodnotu 1). Vykreslovány jsou pak pouze chyby s větším počtem



Obrázek 9.9: Ukázka histogramu výskytu UML chyb

výskytů než je uvedená konstanta. Toto opatření nezmenší informační charakter grafu, neboť vyučujícího zejména zajímají často se opakující problémy při řešení domácích úloh. Vyskytuje-li se nějaká chyba jen ojediněle, není potřeba věnovat jí zvýšenou pozornost.

9.2 Výkonnostní požadavky

Veškeré testování probíhalo na domácím notebooku s operačním systémem Windows 10, procesorem AMD FX-7500 Radeon R7 a 8 GB RAM. Pokud by byl stroj výkonnější, program by samozřejmě běžel rychleji. Časových hodnot bylo dosaženo pětinásobným změřením doby běhu, odstraněním dvou nejvyšších hodnot (aby se omezil vliv náhodných událostí operačního systému) a ze zbylých třech hodnot byl vypočítán průměr. Program byl pro testování spuštěn bez informačních výpisů.

Přehled doby běhu první aplikace je znázorněn v následující tabulce (9.1). První dva řádky a levý sloupeček udávají konfiguraci programu při spuštění. Na prvním řádku je druh grafu, který je na druhém řádku ještě rozdělen na sloupeček s dobou běhu programu a počtem vytvořených grafů. Sloupečky udávají léta, ke kterým se vizualizace grafické informace vztahuje.

	Agregované grafy		**Personální grafy**		**Oba druhy**	
	čas [s]	**počet**	**čas [s]**	**počet**	**čas [s]**	**počet**
2012	29	50	42	67	59	117
2013	31	50	49	89	71	139
2014	20	50	54	105	66	155
2015	30	50	49	83	63	133
vše	60	200	123	344	174	544

Tabulka 9.1: Doba běhu první aplikace

Z tabulky je patrné, že doba běhu programu kromě celkového počtu vygenerovaných grafů závisí na dalších aspektech. Čas potřebný k dosažení výsledků mohou ovlivňovat různé události operačního systému, ale také rozmanitost vstupních dat. Počet vygenerovaných agregovaných grafů je dán počtem domácích projektů, množství personálních grafů zas záleží na počtu studentů předmětu, kteří odevzdali alespoň jeden domácí úkol. Pokud by tedy například větší množství studentů odevzdalo pouze jednu domácí úlohu, bude se pracovat s méně daty, než v případě pilných žáků, kteří odevzdají (téměř) vše. Podobně se také může stát, že v některém konkrétním roce bude většina semestrálních projektů odevzdána na větší množství pokusů.

Na první pohled je také patrné, že čas potřebný k vytvoření agregovaných i personálních grafů najednou je výrazně nižší, než kdyby byly grafy tvořeny postupně. Tento rozdíl je způsoben zejména tím, že vstupní data je potřeba načíst vždy všechna bez rozdílu, ať už je tvořen jakýkoliv druh grafu.

Doba běhu druhé aplikace, tedy čas potřebný k vytvoření histogramů UML chyb pro všechny projekty, ve kterých studenti odevzdávají UML diagramy, se pohybuje kolem tří minut.

10 Závěr

V práci byly řešeny dva podobné případy spočívající v analýze a zobrazování dat o výsledcích prací odevzdávaných na validační server. Při řešení úkolu bylo nutné provést detailní analýzu vstupních dat, která jsou velmi rozsáhlá. Zahrnují několik desítek tisíc souborů, které vznikly validací studentských projektů v průběhu několika let ze dvou předmětů vyučovaných vedoucím práce. Situace byla o to složitější, že bylo nutné získaná data vyfiltrovat, protože občas obsahovala neplatné soubory starších verzí domén validátoru atp. Podařilo se navrhnout vyhovující podobu výsledné grafické reprezentace dat, což byl iterační proces. Výsledkem práce jsou tedy dva programy, na výslovné přání vedoucího práce, spouštěné v konzoli. Umožňují získat požadované informace v jejich přehledné grafické reprezentaci. Obě aplikace jsou navržené modulárně, byly při vývoji průběžně testovány a jsou standardním způsobem dokumentovány, takže případné další vylepšování by nemělo být problém. Programy byly také důsledně otestovány funkčními testy, jejichž podoba je vysvětlena v příloze B – *Uživatelská příručka*. Data pro jejich provedení se nachází na přiloženém CD.

Při testování byla ověřena funkčnost vytvořených aplikací a také zjištěna doba běhu programů. Vzhledem k tomu, že se předpokládá generování grafů většinou pouze z dat jednoho semestru, je tento čas potřebný k vygenerování požadovaných grafů považován za dostačující. Nejvíce času (řádově minutu) zabere vytvoření celkového grafu jednotlivých odevzdávání. Tento personální graf bude nejspíše vygenerován pouze jednou – před zkouškou, proto nebyla doba běhu řešena. V průběhu semestru nesou cenné informace agregované grafy a histogram výskytu UML chyb, častěji tak budou zřejmě generovány právě tyto grafy. Jejich vytvoření je časově mnohem méně náročné.

Celkově bylo vytvořeno 19 Java tříd, které obsahují přes 3000 řádek. Z toho 10 tříd je implementováno pro realizaci první aplikace a 6 pro druhou aplikaci. Zbytek slouží pro testování.

Případné rozšíření práce se nyní příliš neuvažuje, protože v současné době zřejmě splňuje všechny požadavky vedoucího práce. Program je díky způsobu návrhu (třída *Konstanty*) ve značné míře konfigurovatelný, což by pravděpodobně pokrylo všechny drobné požadavky na úpravy.

Literatura

[1] PECINOVSKÝ, *Rudolf. Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Vyd. 1. Brno: Computer Press, 2007. 527 s. ISBN 978-80-251-1582-4.

[2] HEROUT, Pavel. *Učebnice jazyka Java. 5.*, rozš. vyd. České Budějovice: Kopp, 2010. 386 s. ISBN 978-80-7232-398-2.

[3] HUSTON, Pete. *Instant jsoup How-to*. Google Commerce Ltd, 2013. 38 stran. ISBN 978-1782167990.

[4] *JFree* [online]. c2016, [cit. 8.3.2016]. Dostupné z <http://www.jfree.org/jfreechart/api/javadoc/index.html>.

[5] *Demo Source and Support* [online]. [cit. 8.3.2016]. Dostupné z <http://www.java2s.com/Code/Java/Chart/JFreeChartStackedBarChartDemo4.htm>.

[6] PECINOVSKÝ, Rudolf. *Java 8: úvod do objektové architektury pro mírně pokročilé*. První vydání. Praha: Grada Publishing, 2014. 655 stran. Myslíme v. Knihovna programátora. ISBN 978-80-247-4638-8.

[7] SACHIN. *FromDev* [online]. c2016, [cit. 28.3 2016]. Dostupné z <http://www.fromdev.com/2012/09/Free-Open-Source-Java-Charting-Library.html>.

[8] ROON, Patton, *Software Testing*. Vyd 2. Sams Publishing. 2005. 408 s. ISBN:0-672-32798-8.

[9] HEROUT, Pavel. *Záznam přednášek z OKS* [online]. c2016, [cit. 18.4.2016]. Dostupné z <http://www.kiv.zcu.cz/~herout/vyuka/oks/prednasky/oks-1a4.pdf>.

[10] DUDOVÁ, Veronika. *Webová konfigurace validačního serveru.* Plzeň, 2010. 48 s. Bakalářská práce na Fakultě aplikovaných věd Západočeské univerzity. Vedoucí bakalářské práce Pavel Herout.

[11] Kolektiv pracovníků CIV. *Referenční příručka* [online]. c2014, [cit. 20.4. 2016]. Dostupné z <https://is-stag.zcu.cz/PortalNapovedy/build/pdf-zcu/Courseware.pdf>.

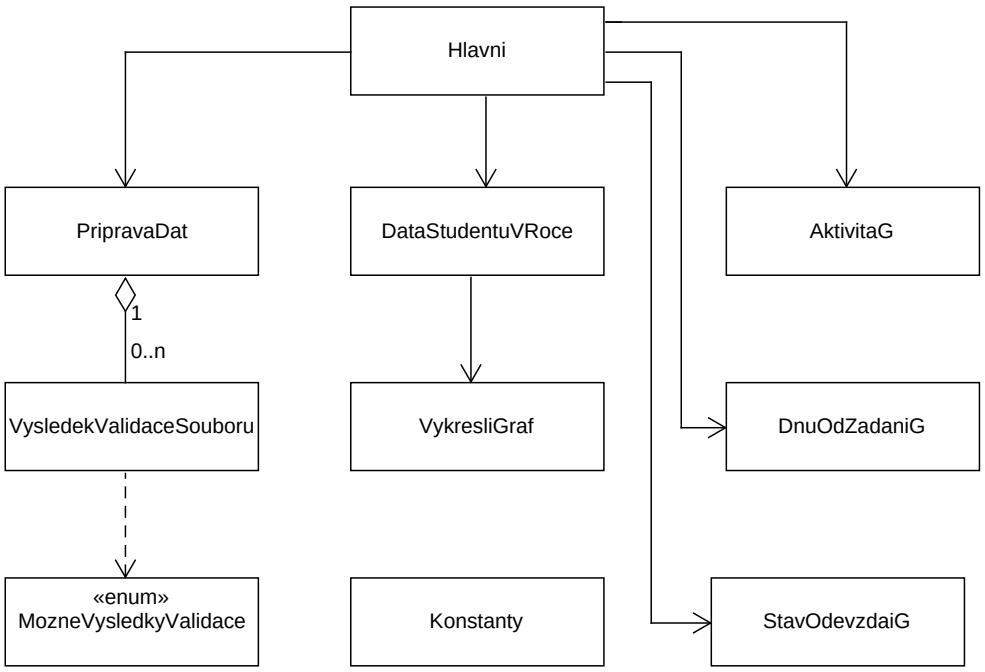
[12] VALENTA, Lukáš. Validační server [online]. c2007, [cit. 4.2.2016]. Dostupné z <http://vs.kiv.zcu.cz/doc/>.

[13] DUONG MANH, Hung. *Rozšiřující moduly validačního serveru.* Plzeň, 2014. 87 s. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity. Vedoucí diplomové práce Štěpán Cais.

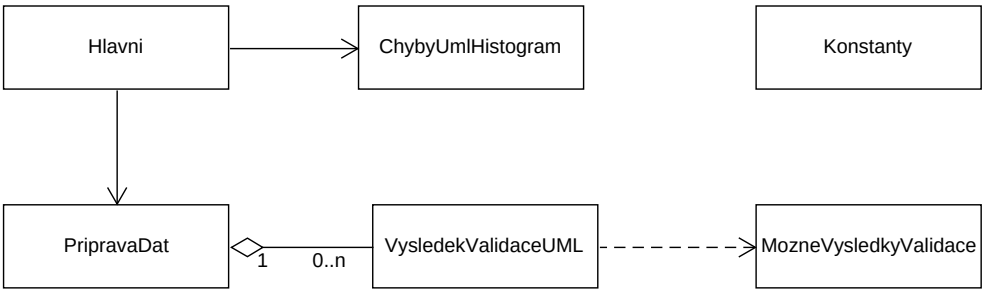
[14] MIKOLÁŠOVÁ, Zuzana. WITZ, Lukáš. ZÍBAR, Karel. *Dokumentace k projektu pro Vyhodnocování UML diagramů.* Plzeň, 2015. 19 s. Semestrální práce na Fakultě aplikovaných věd Západočeské univerzity. Vedoucí semestrální práce Pavel Herout.

A UML diagramy

Součástí této práce jsou UML diagramy vytvořených programů. Na obrázku A.1 se nachází UML diagram tříd první aplikace. UML diagram tříd druhé aplikace je uveden na obrázku A.2.



Obrázek A.1: UML diagram první aplikace



Obrázek A.2: UML diagram druhé aplikace

B Uživatelská příručka

Nyní bude popisována první (Vizualizátor chyb) i druhá aplikace (Vizualizátor UML chyb) z uživatelského hlediska. Oba programy se spouští z konzole, proto je v obou případech vhodné nejprve v příslušném adresáři (ve kterém se nachází *jar* soubor) spustit příkazový řádek.

B.1 Vizualizátor chyb

B.1.1 Spuštění aplikace

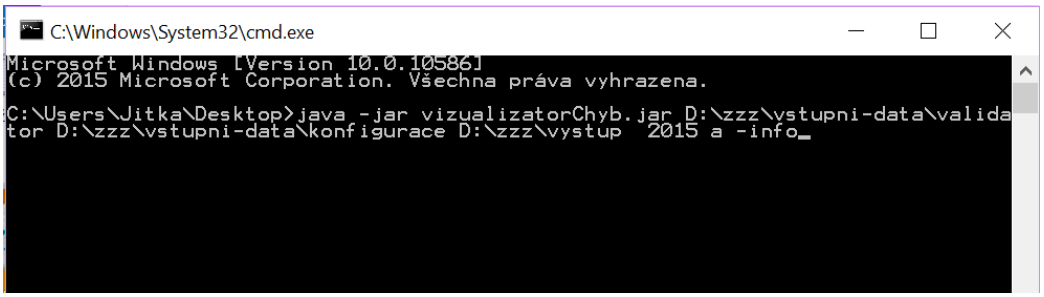
Program se spouští příkazem:

```
java -jar vizualizatorChyb.jar vstupniData konfiguracniData
vystupniAdresar rok prepinac -info
```

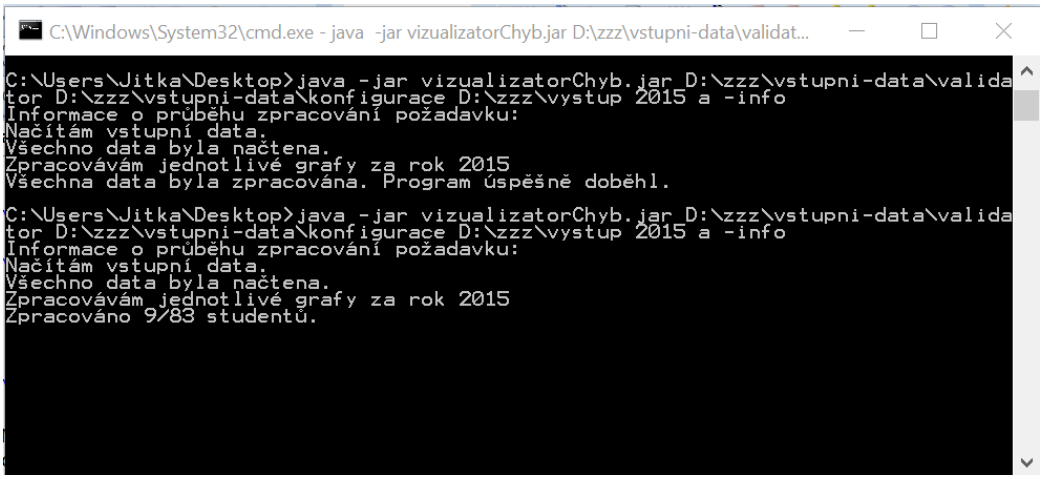
Nejprve je uveden název *jar* souboru, zde `vizualizatorChyb.jar`. Poté následují parametry programu, přičemž první tři jsou povinné, bez nich aplikace nepůjde spustit. Nejdříve je potřeba uvést adresář vstupních dat, které bude program vizualizovat. Následují konfigurační soubor a výstupní adresář. Pro zachování správného kódování všech znaků je nutné, aby byl konfigurační soubor v kódování UTF-8. Zbylé parametry jsou nepovinné, dle potřeby je možno uvést pouze některé. Důležité ale je, že musí být zachováno jejich pořadí. Parametr `rok` pochopitelně značí rok, který si uživatel přeje zpracovávat, `prepinac` zastupuje znaky *a* – pro případ tvoření pouze agregovaných grafů a *j* – pro případ tvoření pouze grafů jednotlivých odevzdání.

Ukázka možného spuštění programu je uvedena na obrázku B.1. V tomto konkrétním případě by program zpracoval data z roku 2015 a vytvořil by z nich pouze agregované grafy. Při běhu programu by byly vypisovány informační výpisy. Ukázka konzole s informačními výpisy je uvedena na obrázku B.2. Při pozorném prozkoumání příkazové řádky je vidět, že program byl spuštěn dvakrát. Proces vzniklý prvním spuštěním již doběhl. Druhý příklad znázorňuje informační výpisy v průběhu programu.

Nepovinné parametry tedy zpřesňují uživatelův požadavek, program pak generuje jen specifikovanou množinu grafů. Od velikosti této množiny se sa-



Obrázek B.1: Ukázka spuštění vizualizátoru chyb



Obrázek B.2: Ukázka informačních výpisů

mozřejmě také odvíjí doba běhu programu. V případě neuvedení požadovaného roku, jsou zpracována všechna dostupná vstupní data. Při absenci přepínače program vytvoří agregované i personální grafy. Vynechání parametru `-info` způsobí vypnutí informačních výpisů. Pokud budou zadány jen povinné parametry, program zpracuje všechna dostupná data, vytvoří všechny druhy grafů a nebude vypisovat aktuální stav běhu programu.

Pokud uživatel zadá neplatný vstupní soubor, program skončí s chybovým hlášením, ve kterém je uvedeno, jaký soubor se nepodařilo nalézt. V případě opomenutí některého ze vstupních parametrů je vypsána nápověda a uživatel je vyzván k opravě.

B.2 Konfigurace aplikace

Ke konfiguraci aplikace slouží třída *Konstanty*. Nejprve jsou uvedeny názvy souborů odevzdávaných domácích úloh a jejich identifikační čísla. Ve třídě se také vyskytují názvy vytvářených grafů a jejich popisky. Dále jsou zde konstanty pro samotnou tvorbu grafů, jako je například šířka sloupečku v personálních grafech. Všechny tyto hodnoty lze dle požadavků konfigurovat.

B.3 Testování aplikace

Pro uživatelské testování aplikace byla připravena data, která jsou umístěna na přiloženém CD v kořenovém adresáři ve složce *testovaci_data/prvni*. V jednotlivých testovacích adresářích se vždy nachází několik vybraných HTML souborů s výsledky validací, tak aby šlo jednoduše určit správný výsledek grafů. Zároveň je dbáno, aby byly pokryty všechny možné případy, jako jsou například všechny různé výsledky validací, odevzdání v řádném a náhradním termínu či neplatný autor (v tomto případě je za neplatného autora považován každý, kdo není student). Jednotlivé podadresáře s testovacími daty obsahují soubor *info.txt*, ve kterém je uveden popis těchto dat.

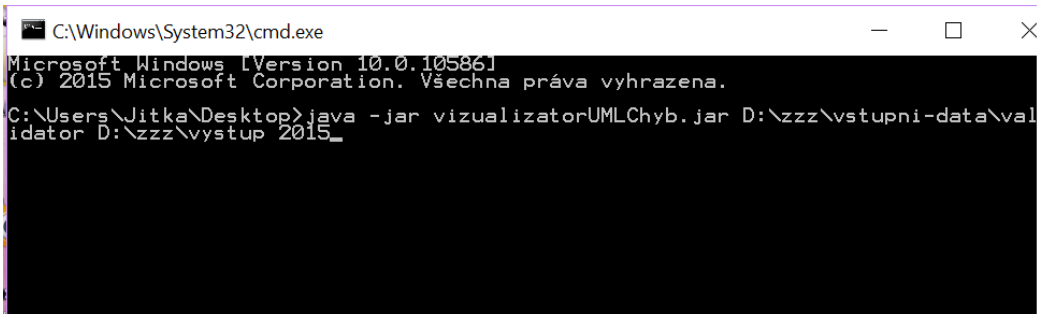
B.4 Vizualizátor UML chyb

B.4.1 Spuštění aplikace

Program se spouští příkazem:

```
java -jar vizualizatorUMLChyb.jar vstupniData vystupniAdresar
rok
```

Nejprve je uveden název *jar* souboru, zde `vizualizatorUMLchyb.jar`. Poté následují parametry programu, přičemž první dva jsou povinné, bez nich aplikace nepůjde spustit. Nejprve je potřeba uvést adresář vstupních dat, které bude program vizualizovat, pak následuje výstupní adresář. Ukázka možného spuštění programu je uvedena na obrázku B.3.



Obrázek B.3: Ukázka spuštění vizualizátoru UML chyb

Pokud by nebyl uveden požadovaný rok, byla by zpracována všechna dostupná vstupní data.

B.5 Konfigurace aplikace

Ke konfigurování aplikace slouží třída *Konstanty*. V této třídě jsou uvedeny, a je možno je měnit, názvy souborů, které mají studenti odevzdávat a identifikační čísla domácích úloh. Lze zde také nastavit minimální počet výskytu chyb, který musí být splněn, aby byla daná chyba vizualizována v histogramu výskytů UML chyb.

B.6 Testování aplikace

Pro uživatelské testování aplikace byla připravena data, která jsou umístěna na přiloženém CD v kořenovém adresáři ve složce *testovaci_data/druha*. V jednotlivých testovacích adresářích se vždy nachází několik vybraných HTML souborů s výsledky validací, tak aby šlo jednoduše předvídat výsledek grafů. Zároveň je dbáno, aby byly pokryty všechny možné případy, jako jsou například odevzdání v řádném a náhradním termínu, neplatný autor a výskyt značného množství druhů chyb v UML diagramu. Jednotlivé podadresáře s testovacími daty obsahují soubor *info.txt*, ve kterém je uveden popis těchto dat.