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

Diplomová práce

Rozšíření editoru procesů vývoje software

Plzeň 2019 Václav Janoch

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 26. června 2019

Václav Janoch

Poděkování

Rád bych poděkoval Ing. Petru Píchovi za vstřícnost, trpělivost, odborné rady a cenné připomínky, které mi pomohly tuto diplomovou práci vypracovat.

Abstract

The aim of this thesis is to extend and improve the possibilities of using a graphic editor of software development processes used as an extension of the SPADe tool. To do this, an analysis of the process editor will be performed to identify possible improvements and shortcomings to the existing implementation. In the next part, this work will deal with the implementation of specific extensions and subsequent application testing. Part of the work is to get acquainted with software development methodologies and various types of patterns and anti-patterns, with SPADe and its meta-data model.

Abstrakt

Cílem této diplomové práce je rozšířit a zlepšit možnosti používání grafického editoru procesů vývoje softwaru, využívaného jako rozšíření nástroje SPADe. Za tímto účelem bude provedena analýza editoru procesů pro identifikaci možných vylepšení a nedostatků stávající implementace. V další části se tato práce bude zabývat implementací konkrétních rozšíření a následným testováním aplikace. Součástí práce je seznámení s metodologiemi vývoje softwaru a různými typy patternů a anti-patternů (procesních vzorů a chyb), s nástrojem SPADe a jeho datovým meta-modelem.

Obsah

1 Úvod

Vývoj softwaru sebou nese i často opakující se chyby v projektovém řízení či chování jednotlivých členů vývojového týmu. Pro snížení rizika výskytu chyb (takzvaných „bad practices“ nebo anti-patternů) existuje několik metodologií a postupů vývoje software. Jednotlivé metodologie pro vývoj softwaru obsahují jasně daná pravidla a postupy, které lze označit za vzory a následně pomocí nich ověřovat nebo vyhledávat anti-patterny v aktuálních projektech a tím zvýšit úspěšnost daného projektu. Za tímto účelem vzniká na Katedře informatiky a výpočetní techniky (KIV), Fakulty aplikovaných věd (FAV), Západočeské univerzity v Plzni (ZČU), nástroj SPADe (Software Process Anti-pattern Detector). Tento nástroj nejprve získá data z Application Lifecycle Management (ALM) nástrojů a uloží je ve sjednoceném formátu do datového skladu. Nad těmito daty je následně provedena analýza pro identifikaci anti-patternů a výsledky analýzy jsou předány vývojovému týmu společně s postupy, jak je vyřešit.

Tato práce se bude zabývat analýzou a rozšířením stávající implementace grafického editoru procesů, který vznikl jako samostatné rozšíření nástroje SPADe v rámci bakalářské práce. Grafický editor slouží k vytvoření jednotlivých anti-patternů případně vzorů vývoje softwaru. Za účelem rozšíření funkčnosti bude provedena analýza architektury aplikace a její možnosti zachycení vzorů a anti-patternů. Budou prozkoumány možnosti napojení aplikace na datový sklad nástroje SPADe a implementována možnost kontroly modelu v rámci datového skladu spolu s možností získání modelu ve formě SQL dotazů. V aplikaci budou provedeny změny za účelem zlepšení použitelnosti a přehlednosti. Pro ověření správné funkčnosti implementace vznikne několik ukázek anti-patternů nebo jejich příznaků.

2 Kontext práce

Tato kapitola definuje a stručně popisuje základní termíny a koncepty zkoumané problematiky, jako je softwarový proces, procesní vzory a chyby (resp. patterny a anti-paterny) vyskytující se v projektu vývoje softwaru. Detekce těchto konceptů je důležitá pro omezení nežádoucích důsledků na dokončení prací při vývoji softwaru, jako jsou například nedodržení termínů, rozsahu projektu a překročení nákladů na vývoj. Kapitola se bude také zabývat metodikami vývoje softwaru.

2.1 Proces vývoje softwaru

Proces vývoje softwaru tvoří sada vzájemně provázaných a časově uspořádaných aktivit, které provádějí členové vývojového týmu společně či jednotlivě. Tyto aktivity mění vstupy vcházející do projektu na příslušné výstupy, čímž napomáhají k rozvoji vytvářeného softwaru prostřednictvím několika etap a milníků. Proces je tvořen pomocí třech základních stavebních prvků [25][35], kterými jsou:

- role,
- úkoly,
- výsledky práce (z angl. work product).

Role je přidělena osobě, vykonávající určitou činnost, např. vývojář, systémový tester, analytik požadavků atd., ze kterých vycházejí určité kompetence a odpovědnosti. Konkrétní osoba může mít v rámci projektu více rolí, stejně tak jedna role může být přiřazena více lidem. Existuje zde tedy vazba N:M mezi rolí a osobou.

Za úkoly lze považovat atomické jednotky práce, vyplývající z pracovního postupu v procesu, např. popsání funkčních požadavků, tvorba nebo vykonání jednotkových (unit) testů, případně týmové schůzky. Úkoly obsahují konkrétní cíle, vstupy a výstupy. Jednotlivé úkoly mohou být přiřazeny jedné nebo více osobám, respektive jedné nebo více rolím.

Pracovní produkty (v softwarově inženýrské praxi též běžně nazývány artefakty) představují vstupy a výstupy jednotlivých úkolů, činností nebo celého procesu. Artefakty mohou mít různou úroveň formality v závislosti jejich typu, povaze a důležitosti. Mohou to být například formální dokumenty, jako je dokument specifikace požadavků, testovací scénáře, zdrojový kód, ručně psané poznámky nebo screenshoty spolu s fotografiemi náčrtků z porad, apod.

Při rostoucí složitosti procesu je potřeba k jeho srozumitelnému popisu využívat další koncepty, jako jsou:

- aktivita,
- dělení procesu na sub-procesy,
- a kategorie,
- pomocné materiály.

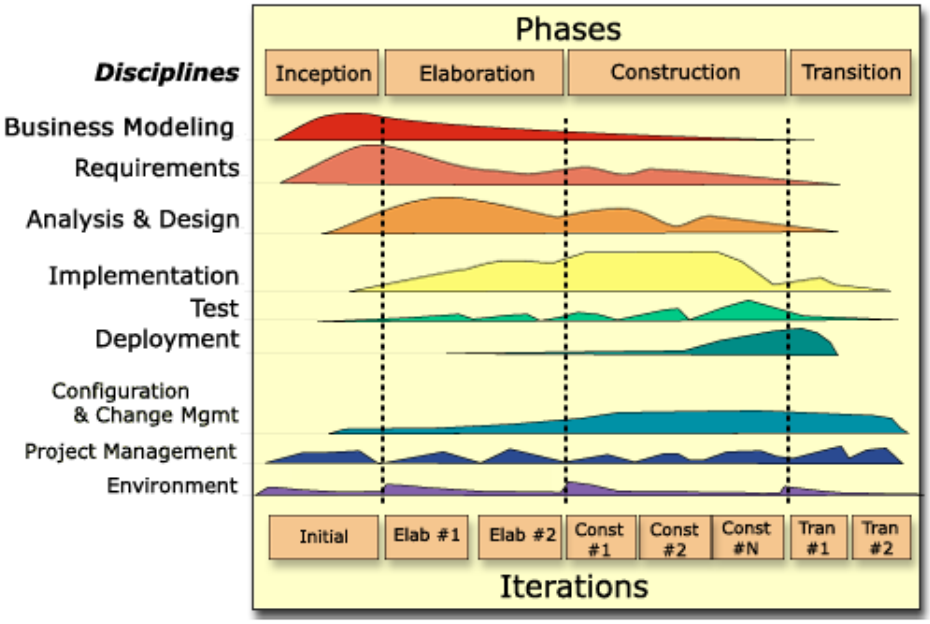
Aktivita představuje sadu souvisejících úkolů se společným specifickým cílem nebo s periodickým výskytem, např. komunikace se zákazníkem.

Proces s rostoucí velikostí a složitostí je vhodné rozložit do menších procesů. Tyto menší procesy umožní věnovat se přesnějším úkolům, které vedou k meziproduktu. Jednotlivé sub-procesy mohou být reprezentovány jako fáze, které se vyskytují v sekvenčních a iteračních metodikách nebo iterace, časově malé opakující se části životního cyklu vývoje softwaru. Může se jednat o týdenní nebo dokonce denní sub-procesy. Koncepty lze rozdělit do kategorií na základě jejich rozdílných aspektů. Proces může být rozdělen na jednotlivé disciplíny např. návrh, implementace, testování, nasazení atd. Každý úkol a artefakt je obvykle spojen s jednou nebo více disciplínami, viz obrázek 2.1.

Pomocné materiály vedou k jisté automatizaci nebo usnadnění procesu. Mohou obsahovat šablony artefaktů, manuály, pokyny a návody pro nástroje.

Metodika

Metodiky vývoje softwaru bývají velmi často tvořeny pomocí procesních modelů. Jedná se o polo-standardizované, dobře popsané a strukturované modely, které představují zobecněnou formu procesu vývoje softwaru vhodnou pro modifikaci vzhledem ke konkrétnějšímu kontextu [13]. Metodika



Obrázek 2.1: Unified Process fáze a disciplíny

dále může kromě samotného modelu, zahrnovat pokyny pro její použití nebo přizpůsobení konkrétnějším potřebám projektu. Metodika může také obsahovat více variant modelu procesu, takzvané workflow. Metodiky by měly napomáhat k úspěšnému dokončení softwarového projektu, k čemu využívají převážně sbírku osvědčených postupů a metod (best practices).

2.2 Projekt

Softwarový projekt lze označit jako instanci vykonání konkrétního softwarového procesu, ze kterého daný projekt vychází. Projekt formuje konkrétní vstupní zdroje na konkrétní výsledky práce v závislosti na konkrétním kontextu projektu. Projekt nemusí procházet celým procesem vývoje softwaru, ale může být pouze instancí některé z volitelných částí procesu. Projekty mohou přizpůsobit své aktivity aktuálním potřebám a to z důvodu rozdílných úrovní detailu použitého procesu a metodiky. V některých případech se může projekt oddělit od daného modelu procesu, nebo přímo vynechat některou ze zahrnutých částí modelu.

2.3 Praktiky

Praktiky jsou částečně standardizované dílčí procesy, respektive způsoby provádění konkrétního úkolu. Jako postup lze brát popis úloh jednotlivých rolí nebo jak pracovat s jednotlivými nástroji či technologiemi. Případně může konkrétní praktika popisovat, kdy vytvořit konkrétní výstup nebo jaké informace by měly být obsaženy v konkrétních dokumentech.

Některé praktiky mohou prostupovat celým procesem a být specifickým aspektem celé skupiny procesů (popř. metodik) odlišujících je od ostatních přístupů. Jako příklad lze použít iterativní vývoj. Praktiky mohou být také specifičtější a mohou udávat, jakou technologii využít pro zachycení konkrétních znalostí, například použití diagramu tříd pomocí Unified Modeling Language (UML).

Jednotlivé praktiky lze vyjmout z konkrétních procesů a použít je v jiných. Proces může být tedy považován za soustavu praktik, na druhé straně praktiky lze považovat za šablony či vzorce sub-procesů. Tyto části mohou být dále specifikovány do různé úrovně detailu nebo upraveny v závislosti na kontextu.

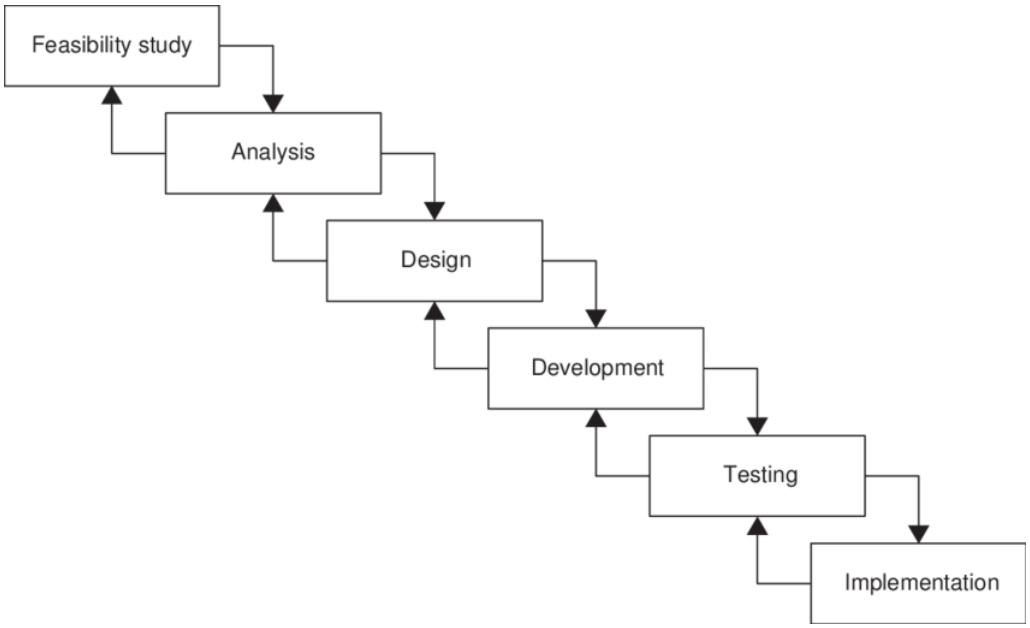
2.3.1 Příklady praktik

V této podkapitole budou představeny některé existující metodiky a praktiky v nich používané.

Vodopád

Metodika vodopádu představuje sekvenční metodiku procesu vývoje softwaru. Metodika obsahuje mimo jiné následující praktiky:

- **Fáze** – Životní cyklus projektu je rozdělen do fází. Jednotlivé fáze se zaměřují na konkrétní aspekty projektu a na jejich konci se vždy nachází jistá sada artefaktů. Pokud proces přejde do následující fáze, jen velmi obtížně se může vrátit do fáze předchozí. Tato vlastnost vychází ze sekvenční podstaty vodopádu, viz obrázek 2.2. Samotné fáze mohou být využity k hledání vzorů nacházejících se v metodice vodopádu. Za vzor může být považován určitý počet fází a jejich zaměření nebo dílčí úkoly a výstupy konkrétní fáze (artefakty). Vzor lze, také hledat pomocí ověření přítomnosti jednotlivých artefaktů a jejich stavu po skončení konkrétní fáze.



Obrázek 2.2: Ukázka Vodopádu [39]

- **Aktivita rolí** – Stejně jako artefakty, by se i určité role přiřazené jednotlivým pracovníkům měly vyskytovat jen v určitých fázích projektu. Například role vývojáře by se neměla vyskytovat ve fázi analýzy. Omezený výskyt rolí v procesu vodopádu může být využit k nalezení dalších potencionálních vzorů.

- **Robustnost artefaktů** – Základem úspěšné fáze vodopádové metodiky a následného přechodu do následující fáze je předpoklad, že artefakty vzniklé v předchozí fázi jsou co možná nejvyšší úrovně detailu. Pro detekci určitých vzorů v modelu vodopádu, může být využito formální stránky artefaktů, konkrétně tedy jejich struktura, obsah a velikost.

2.3.2 (R)UP

Metodika Unifed Process (UP) patří do skupiny iterativních metodik. Dále existují specifičtější formy jako například Enterpise Unifed Process (EUP) [3] nebo IBM Rational Unifed Process (RUP). UP specifikuje devět disciplín (praktik, viz obrázek 2.1). Jednou z využívaných praktik je rozdělení kompetencí mezi 5 rolí:

- **analytik**,

- **architekt**,

- **vývojář**,
- **tester**,
- **projekt manager**.

UP praktikuje dělení do čtyř fází. Jednotlivé fáze končí dosažením určitého milníku. Milník je tvořen souborem kritérií, které musí produkt splňovat a znalostí pracovníků na kterých závisí pokračování projektu.

Další praktikou využívanou v UP je rozdělení celého procesu do časově omezených a opakovatelných dílčích procesů (iterací). Každá iterace obsahuje:

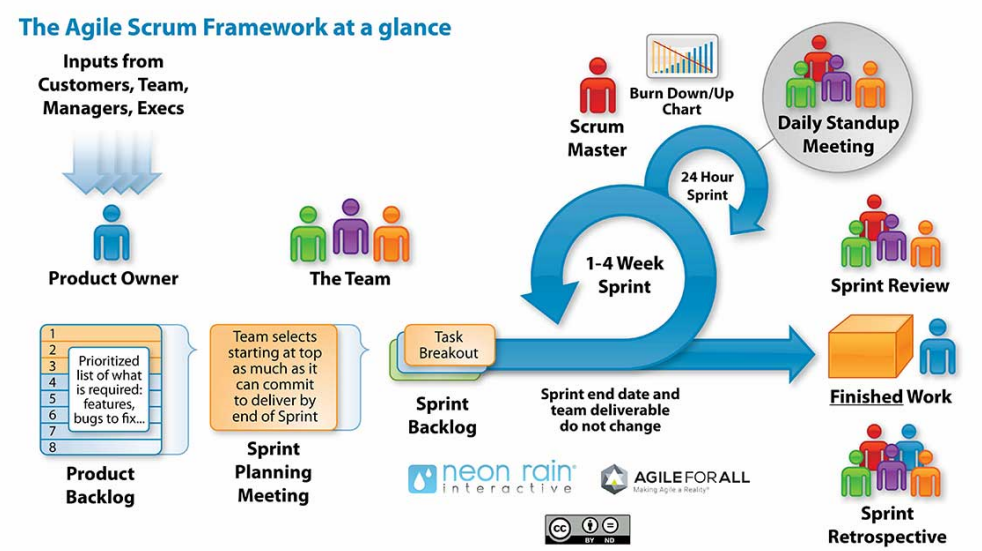
- **plánovací schůzku**,
- **provedení iterace**,
- **zpětná vazba od zákazníka**,
- **zhodnocení iterace**.

Samotná kritéria milníku tvoří jednu z možností pro popsání vzoru představující UP. Pro detekci vzorů mohou také sloužit samotné fáze, jejich obsah a aktivita příslušných rolí podobně jako v předešlém modelu vodopádu. Lze také využít počet iterací, jejich délku, strukturu, případně přítomnost příslušných výstupů a činností v průběhu iterace, pro nalezení vzorů týkajících se iterací.

2.3.3 Scrum

Scrum patří do Agilní metodiky. Principy využívané v agilních metodikách jsou flexibilita při řešení změn, zaměření se na požadavky a pravidelné vytváření přírůstků funkčnosti. Scrum byl představen v roce 1995 Schwaberem a Sutherlandem [38]. Hlavním cílem metodiky Scrum je maximalizovat kvalitu procesu a tím i výsledné výstupy. Procesní model metodiky Scrum je znázorněn na obrázku 2.3[19]. Ve Scrumu jsou praktikovány následující typy činností, rolí a artefaktů.

- **Self-organizing tým** – Členové týmu nejsou nuceni do předem stanovené struktury nebo hierarchie rolí v týmu. Jednotlivé role, si každý tým přerozdělí sám.



Obrázek 2.3: Model procesu Scrum[19]

- **Sprint** - Sprint označuje ve Scrumu iteraci. Jedná se o nepřerušitelný časový interval, nejčastěji po dobu jednoho měsíce, ohraničený setkáním se zákazníkem. Na začátku sprintu bývá z pravidla schůzka označovaná jako Sprint Planning Meeting. Tato schůzka je složena z činností, jako je změna priorit současných nesplněných požadavků nebo změny zdrojů pro zvýšení pokroku ve vývoji.

- **Backlog** – Backlog představuje jistou formu plánu. Obsahuje jednotlivé požadavky, které mají být implementovány. Existují dva typy backlogu. Prvním typem je projektový backlog, který obsahuje spolu s informací o nevykonaných úkolech i jejich různou úroveň zpracování (popisu, analýzy, časového odhadu). Druhým typem je Sprint Backlog obsahující dobře popsané a prioritizované úkoly, plánované pro současnou (resp. nadcházející) iteraci, které jsou potřebné pro správné časové odhady. Detailnost popisu a prioritizace slouží ke snazšímu odhadu náročnosti úkolů a tím i naplánování iterace, která má omezené časové i jiné zdroje.

- **User Stories** – Stručný popis funkčních požadavků z pohledu uživatele. Obvykle jsou umístěny na tzv. Agile Board. Jednotlivým členům týmu je přiřazen některý z požadavků spolu s jeho časovým odhadem.

- **Daily Scrum** - Krátká denní schůzka vývojového týmu pro vzájemné informování o aktuálních problémech a stavu úkolů přidělených jednotlivým členů.

- **Retrospektiva sprintu** – Interní setkání vývojového týmu, k vyhodnocení sprintu na jeho konci. Jsou zde probírány pokroky a stav projektu, problémy v procesu nebo případné potřeby změny procesu a dílčích postupů. Obvyklým výstupem je artefakt představující záznam retrospektivy.

- **Timeboxing** – Scrum je zaměřen na přísné dodržování časových omezení. Timeboxing je často aplikován na sprinty, daily scrum a další schůzky. Důvodem je vytvořit jistou míru časového tlaku, která napomáhá udržet pozornost na nejpodstatnější záležitosti.

- **Specific Roles** – Ve Scrumu se vyskytují dvě specifické role: Product Owner, zastupující většinou zájmy zákazníka a dohlížející na vytvářený produkt a Scrum Master. Prací Scrum Mastera je dohlížení na správné dodržení Scrum procesu, mj. hlídání časových omezení u výše zmíněných schůzek.

2.4 Pattern a Anti-pattern

Pojem design pattern, nebo-li návrhový vzor poprvé definoval architekt Christopher Alexander, který v architektuře definoval vzor jako „opakovaně použitelnou formu řešení designového problému [1]“. Alexander poukazoval na to, že i odlišné budovy mohou být postaveny pomocí souboru jednodušších vzorů. Lze tedy za vzor považovat obecné řešení problému či úkolu, ze kterého může být získáno specifické řešení[2].

V softwarovém inženýrství může být na pattern pohlíženo z více pohledů. Lze pohlížet na pattern, jako na identifikovatelný a opakovatelně použitelný koncept, strukturu dat nebo postupů[30]. Bez ohledu na to, zda pattern něco přináší, nebo spíše škodí.

Druhý pohled pohlíží na vzor, jako na běžně se vyskytující šablonu, která má pozitivní důsledky pro danou situaci. V oblasti procesu vývoje softwaru a projektového managementu mohou být tyto vzory také nazývány jako „dobré“ (good) nebo „nejvíce osvědčené praktiky“ (best practices). Na metodiky se pak dá nahlížet jako na ucelené sady best practices[26] [7].

Anti-pattern je oproti patternu jasně definován a to tak, že se jedná o *„jednotný přístup k řešení opakujících se problémů, které se ukázaly jako neúčinné “*[2]. Anti-patterny mohou být tedy považovány za podmnožinu

patternů z prvního pohledu nebo jeho opakem z druhého pohledu. Anti-patterny mohou být také často označovány za takzvané bad practices.

Patterny a anti-patterny se mohou vyskytovat v mnoha oblastech vývoje softwaru. Může se jednat jak o samotný programový kód, architekturu nebo týmovou spolupráci. Z pohledu nástroje SPADe lze za anti-pattern považovat problém, chybu, špatně aplikované řešení problému, zneužití best practice nebo nedodržení předepsaného procesu.

2.4.1 Příklady Anti-patternů

Mushroom Management

Mushroom management se vztahuje na řízení funkčních požadavků na software. Anti-pattern se objeví tehdy, pokud vývojáři pracují s nedostatečně specifickými funkčními požadavky. Pokud jsou požadavky vytvořeny pouze v počáteční fázi vývoje a vývojář nemá způsob jak získat dodatečné informace, je nucený k pseudoanalýze, bez účasti zákazníka. Při Mushroom managementu mohou projekty vynechat analýzu a přejdou rovnou k designu a kódování[26] [37].

Metric Abuse

Anti-pattern Metric Abuse patří do skupiny manažerských anti-patternů. Vyznačuje se nekompetentním nebo škodlivým využíváním metrik pro jednotlivé části projektu[26]. Metriky mohou být zneužívány úmyslně, případně mohou být měřeny nesprávné nebo žádné oblasti. Tyto případy mohou ukazovat uspokojivý stav projektu, ale plně nevypovídají o skutečném stavu projektu. To může vést ke zvýšení nákladů na celý projekt nebo k nesplnění jeho kvalitativních požadavků.

Fire Drill

Anti-pattern Fire Drill spočívá v nepřiměřeném množství času věnovaného vyjednávání funkčních požadavků mezi managementem a zákazníkem, během něhož vývojový tým nemůže pracovat. Následně je potřeba zrychlit samotný vývoj pro dosažení výsledku v určitý termín. Toto zrychlení může vést ke špatné kvalitě kódu, nedostatečnému otestování aplikace a nekvalitní dokumentaci. Může se zde objevit kontrast v komunikaci management-vývojář, kdy je nejdříve období ticha a po realizaci panika a hluk.[7]

Half Done Is Enough

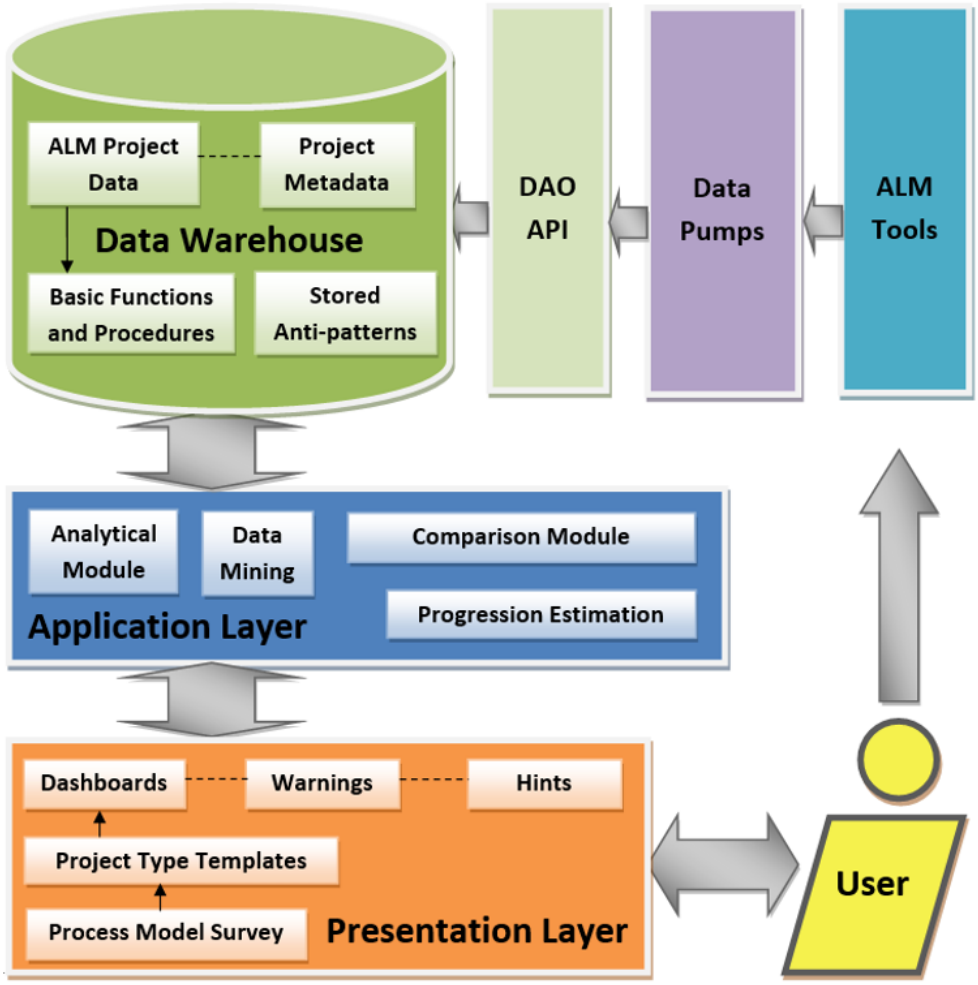
Anti-pattern Half Done Is Enough se vyznačuje tím, že je ze strany managementu kladen příliš velký důraz na růst aplikace, to znamená převážně na implementační část. Ostatní části jako testování, refactoring, dokumentace a údržba jsou na minimální úrovni nebo neexistují vůbec. Tím se u daného projektu zvyšuje tzv. technický dluh, nedostatečná modularita, architektonická eroze a klesá znovupoužitelnost částí programovacího kódu. Důsledkem toho je, že u projektu postupně dochází ke snížení počtu nových funkcí, prodlužování doby oprav a odkládání termínů.[12]

Corncob

Jako Corncob lze označit nějakou osobu, která se vyznačuje agresivním chováním vůči vývojovému týmu nebo celé společnosti. Činnosti vývojového týmu nebo celého projektu se zastaví na určité hranici a není schopný dalšího postupu. Corncob osoba často nesouhlasí k klíčovými rozhodnutími v procesu vývoje software a jsou kvůli ní často měněny. Častými změnami jsou práce zdržovány a projekt může dosáhnout zpozdění. [7].

3 Nástroj SPADe

Software Process Anti-pattern Detector (SPADe) je nástroj vyvíjený na KIV FAV ZČU. Primárním účelem nástroje je dolování dat z nástrojů pro řízení životního cyklu aplikace (Application Lifecycle Management - ALM), které obsahují informace týkající se řízení projektů při vývoji softwaru.



Obrázek 3.1: Schéma nástroje SPADe [36]

3.1 ALM nástroje

„Application Lifecycle Management pokrývá celý životní cyklus od vzniku myšlenky, přes vývoj, testování, nasazení, podpory a nakonec zobchodování systému.“[10].

ALM obsahuje několik různých typů disciplín provázející vývoj softwaru, jako je například projektové řízení, správa požadavků, řízení změn, verzí, evidence práce na projektu, atd (viz obrázek 3.2). ALM nástroje poskytují podporu prostřednictvím softwarových aplikací usnadňujících práci s dříve zmíněnými disciplínami vývoje softwaru. Stejně tak umožňují komunikaci a spolupráci mezi vývojovými týmy i jednotlivými odděleními. Nástroj SPADe byl implementován tak, aby byl schopný pracovat s různými ALM nástroji. SPADe umožňuje ukládat a analyzovat víceméně kompletní historii projektů vývoje software. Data jsou získávána z issue-trackerů, Version Control Systému (VCS), WIKI stránek projektů a v budoucnu z emailové korespondence a dalších potencionálních ALM[32][36]. V současné době SPADe umožňuje získávat data například z těchto nástrojů:

- Atlassian Jira[1],
- Bugzilla[2],
- Git[3],
- GitHub[4],
- Redmine[5].

Nástroj SPADe je navržen pro snadné rozšíření stávající sady ALM nástrojů, ze kterých lze dolovat data.

3.2 Funkce SPADe

Funkčnost nástroje SPADe je zobrazena na obrázku 3.1. Jak bylo řečeno, jednou z funkčností nástroje SPADe je získávání dat z ALM nástrojů. SPADe využívá k dolování dat z ALM nástrojů datové pumpy. Každý ALM nástroj má implementovanou vlastní pumpu, která funguje na principu Extract -

[1]https://www.atlassian.com/software/jira
[2]https://www.bugzilla.org
[3]https://git-scm.com
[4]https://github.com
[5]https://www.redmine.org

Obrázek 3.2: Ukázka služeb ALM

Transform - Load (ETL) a ukládá data transformovaná do formátu jednotného datového modelu, viz obrázek 3.3 do společného datového skladu. K ukládání dat využívá vstupní Data Access Object (DAO) datového skladu SPADe. Data uložená v datovém skladu nástroje SPADe jsou dále zpracována a jejich primární využití slouží primárně k identifikaci anti-patternů (často opakovaných chyb) projektového řízení, viz [32]. Informace o výskytu jednotlivých anti-patternů, jsou dále předány ve srozumitelné formě projektovému manažerovi nebo vedoucímu vývojového týmu a zároveň poskytnuty rady, jak anti-patrerny odstranit či jinak řešit na základě informací získaných analýzou předešlých projektů a znalostí v oboru. SPADe může sloužit mimo jiné také k různým účelům vyhodnocení chování jednotlivých rolí a aspektů v daném projektu [33]. Jednou z možností využití je analýza veškerých dat v projektu a pokus například o přiřazení role (např. architekta) na základě konkrétní činnosti osoby v projektu, pokud není daná role ALM nástrojem nikomu explicitně přiřazena. Další možnost využití nástroje SPADe je analýza činnosti

člena týmu pro ověření adekvátnosti jeho aktivit vzhledem k přiřazené roli. Jedná se tedy o opačnou operaci oproti předchozí zmíněné funkčnosti.

Terminologie využívaná v datovém modelu, potažmo metamodelu, nástroje SPADe je převzata z několika existujících relevantních zdrojů. Jsou jimi metamodely Software & Systems Process Engineering Meta-Model (SPEM) [27] a Open Services for Lifecycle Collaboration (OSLC) [31], výsledky příbuzného výzkumu, jako Software Workflow Language (SEWL) [14], který se zabývá podobně zaměřenou tématikou a metodiky vývoje software, například Rational Unified Proces (RUP) [25] .

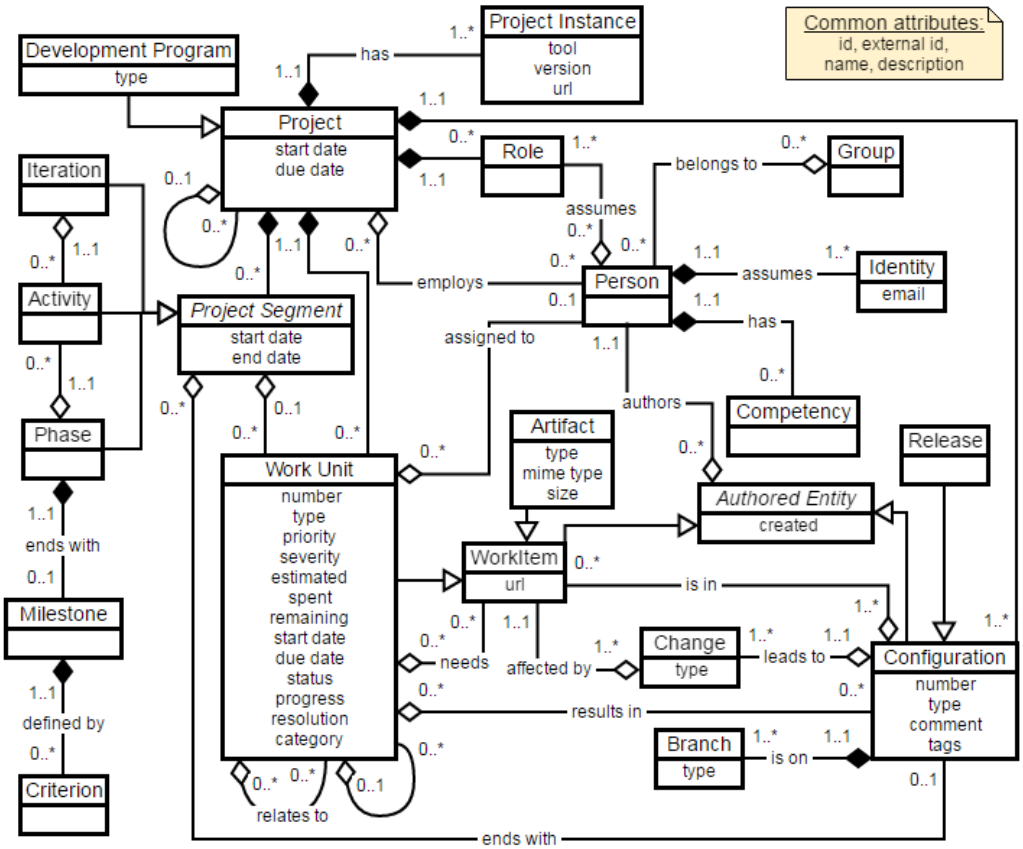
Modely procesu vývoje softwaru využívané k řízení projektů spočívají na třech základních konceptech: úkol (v terminologii SPADe nazývaný **Work Unit**; termín převzatý z [14]), role (**Role**), artefakt (**Artifact**, termín převzatý z terminologie RUP [32]). Pro potřeby nástroje SPADe, aby mohl pojmout všechny metodiky procesu vývoje softwaru, je potřeba modelovat menší celky projektu v závislosti na metodologii. Tyto celky, lze nazývat segmenty a jsou to:

- fáze (**Phase**),
- iterace (**Iteration**),
- aktivita (**Activity**).

Některé další elementy, jako například vývojová větev (**Branch**), **Commit** a další, byly převzaty z terminologie používané v ALM nástrojích a praxi softwarového inženýrství. Jiné, méně obvyklé elementy zahrnují například **Work Item** a **Configuration**. **Configuration** slouží k zachycení stavu vyvíjené aplikace v určitém časovém bodě. Představuje soubor artefaktů, úkolů, obsahu úložiště a dalších aspektů projektu, které byly změněny oproti předchozímu stavu jednou atomickou akcí. **Configuration** může být i samotná revize z VCS nástroje, která může být označená pomocí tagem nebo označovat release. **Work Item** představuje nadřazenou abstraktní třídu pro zachycení společných atributů artefaktů (**Artifact**), úkolů (**Work Unit**) a konfigurací (**Configuration**) a dalších tříd od konfigurace oddědených (např. **Commit**), viz doménový model na obrázku 3.3.

Nástroj SPADe je implementován pomocí programovacího jazyka Java. Aplikace využívá existence rozhraní pro programování aplikací (Application Programming Interface – API) pro dolování dat v Javě a Representational

State Transfer (REST) technologii u všech vybraných ALM nástrojů. Datový sklad je realizován pomocí MySQL databáze a pro komunikaci se systémem řízení báze dat (Database Management System – DBMS) je využíváno objektově relační mapování (Object-Relational Mapping – ORM) [36].



Obrázek 3.3: Datový model nástroje SPADe

V souvislosti s nástrojem SPADe vznikly některé samostatné aplikace rozšiřující jeho funkčnost. První aplikace vznikla za účelem vytvořit grafické uživatelské rozhraní (Graphical User Interface – GUI) pro datový sklad nástroje SPADe, následně také vznikl Grafický editor modelů procesů, popřípadě anti-patternů. Druhou jmenovanou aplikací se zabývá i tato práce. Daná aplikace umožňuje tvorbu modelů procesů, jejich částí nebo anti-patternů vyjádřených s využitím datového modelu SPADe (doménová forma na obrázku 3.3). Modely jsou zpracovány pomocí jazyka eXtensible Markup Language (XML), jehož správné složení je validováno pomocí vytvořeného XML Schema Definition (XSD) schématu[20][24], více informací o daném editoru se nachází v kapitole 4.

4 Analýza výchozího stavu implementace

Aplikace pro editaci procesů vývoje softwaru, vznikla jako samostatný program, rozšiřující již existující nástroj SPADe v rámci bakalářské práce v roce 2017. Cílem této práce bylo vytvořit grafický nástroj pro vytváření modelů procesu vývoje softwaru, patternů a anti-patternů odvozených dle SPADe metamodelu. Aplikace implementovaná v rámci této práce, umožňuje ukládat vytvořený datový model do XML souborů stejně jako jej zpětně načíst. V souvislosti s prací s XML soubory vzniklo XSD schéma[17], podle kterého je daný datový model validován. Dále je využíváno jednotlivých elementů obsažených v XSD schématu tak, že pomocí programu xjc.exe a skriptu utility Ant byly vygenerovány .class soubory, které byly následně spojeny do knihovny `SPADEPAC`.

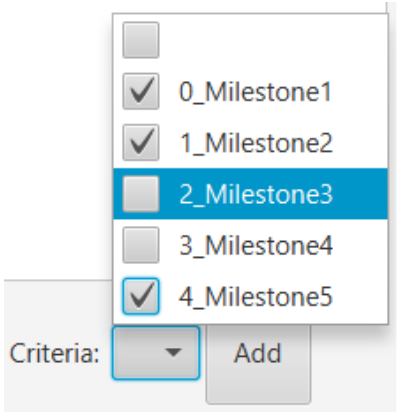
4.1 Architektura a funkčnost

Editor procesů byl v původní verzi implementován pomocí programovacího jazyka Java, konkrétně ve verzi 1.8. Grafické uživatelské rozhraní je vytvořeno pomocí frameworku JavaFX. Framework JavaFX musel být dále doplněn pomocí knihovny `ControlFX` (verze 8.40.11)[15] o potřebné a chybějící grafické prvky, jako je například komponenta `CheckComboBox`. Náhled komponenty je vidět na obrázku 4.1. Jedná se o rozšířenou komponentu `ComboBox`, kdy lze vybrat více než jednu položku ze seznamu. Zdrojové kódy k původní implementaci se nacházejí na veřejně přístupném GitHub úložišti pod tagem SPADEUI_1.0(Bakalarka)[1].

Aplikace je strukturovaná do jednotlivých balíků. Konkrétně jsou to balíky:

- `abstractform` - Tento balík obsahuje třídy se základními grafickými prvky formulářů, od kterých lze následně oddědit okna pro zadávání dat o jednotlivých segmentech a elementech procesu vývoje softwaru.
- `graphics` - V tomto balíku se nacházejí třídy pro jednotlivé grafické prvky.

[1] https://github.com/VenaJanoch/BakalarkaSPADEUI

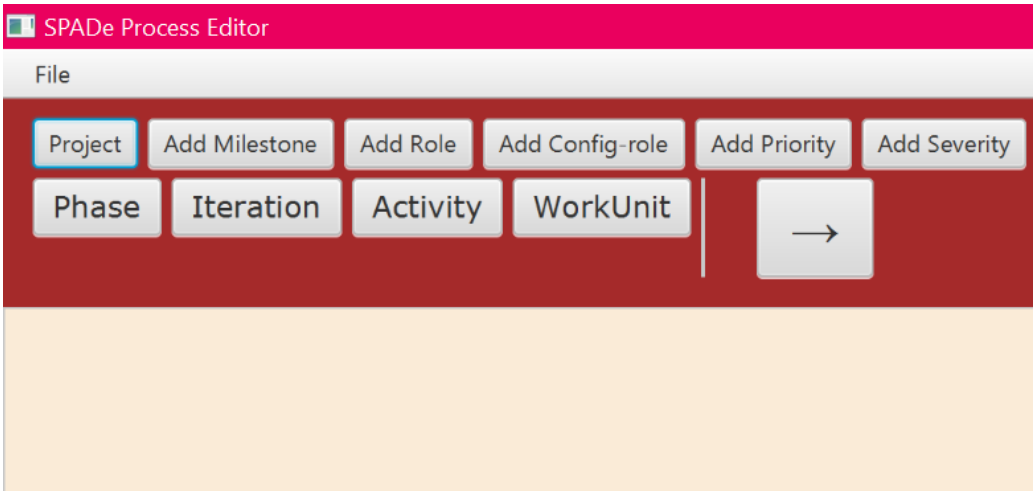


Obrázek 4.1: Ukázka komponenty CheckComboBox

- `forms` - Zde se nacházejí třídy, které dále dědí některou třídu z balíku `abstractform`. Jejich pomocí jsou zadávány veškeré informace do datových struktur aplikace.
- `interface` - Balík s rozhraními, která by měla sjednocovat funkčnost formulářů.
- `run` - Balík obsahující třídu `Run` pro spuštění aplikace.
- `service` - Balík obsahující třídy s obslužným kódem ke GUI, pro práci s datovým modelem, varovnými okny. Nacházejí se zde také výčtové typy.
- `tables` - Obsahuje třídy, představující kontejnery uchovávající data určené do tabulkových grafických komponent.
- `XML` - Zde se nacházejí třídy pro převod datového modelu do formátu XML.

Aplikace je tvořena pomocí jednoho hlavního okna (viz obrázek 4.2), které obsahuje samotné kreslící plátno a v horní části panel s tlačítky. První řada slouží k vyvolání tabulkových formulářů pro Elementy. Elementy jsou vytvářeny ve stejném okně, jako se nachází jejich přehledová tabulka a následně jsou přidány do výběrového seznamu, ze kterého mohou být přiřazeny do konkrétního segmentu nebo elementu pomocí komponenty `ComboBox`, popř. `CheckComboBox` (v případě nutnosti vybrání více položek ze seznamu). Pomocí tlačítek ve spodní řadě lze přidat instance segmentů (**Phase, Iteration a Activity**) a elementu **Work Unit** na kreslící plátno. Prvky určené pro kreslící plátno mohou být přidány dvěma způsoby: klasickým stisknutím tlačítka, nebo přetažením daného tlačítka na kreslící plátno. S jednotlivými

prvky na plátně lze volně pohybovat a v případě instancí elementu **Work Unit** i spojovat. Tato funkčnost je umožněna pomocí tlačítka označeného šipkou v horní části obrazovky. Pro zadání informací o konkrétním prvku umístěném na plátně je potřeba vyvolat nové okno, které obsahuje formulář pro zadání potřebných dat a v případě segmentů i vlastní kreslící plátno pro přidávání elementů **Work Unit** náležících danému segmentu a to dvojklikem na daný prvek na plátně. V těchto formulářích lze také vytvořit vazbu mezi elementy (např. přiřadit autora k artefaktu), popřípadě určit příslušnost prvku k segmentu. Aplikace také využívá systém varovných vyskakujících oken, jako reakci na nepovolenou manipulaci s **Work Unit**, na nepovolený vstup do formuláře, při odstranění prvku nebo pro potvrzení ukončení celého programu. Více informací o aplikaci lze nalézt v textu původní bakalářské práce [20].



Obrázek 4.2: Ukázka hlavní obrazovky nástroje

4.2 Identifikované chyby

Při procházení existující implementace aplikace bylo nalezeno několik oblastí, ve kterých byly zjištěny nedostatky a chyby. Pro označení těchto oblastí je v textu zavedena ID konvence:

- **AD (Architecture Defect)** – Oblast architektury aplikace,
- **DD (Decomposition Defect)** – Oblast dekompozice programového kódu,
- **FD (Functional Defect)** – Oblast funkčních chyb aplikace,

- **DMD (Data Model Defect)** – Oblast špatné manipulace s datovým modelem aplikace,
- **VD (Visual Defect)** – Oblast grafického rozhraní,
- **UFD (User Friendly Defect)** – Oblast uživatelsky přívětivé manipulace s aplikací.

Jak bylo řečeno výše, aplikace je dělena do různých balíků. Bohužel jednotlivé vrstvy aplikace jsou velice promíchané a to zhoršuje jak funkcionalitu aplikace, tak i její následné testování a rozšíření. Například samotný formulář pro editaci některého segmentu nebo elementu obsahuje jak data z datového modelu, tak i logiku aplikace. Samotné formuláře jsou tvořeny pomocí hierarchie dědění, které alespoň v nějaké míře udržují aplikaci přehlednou. Příklad špatné struktury aplikace se nachází ve třídě `Control`, která slouží pro kontrolu vstupů uživatele, práci s určitými soubory, grafické výpočty, konverzi formátu data nebo zobrazování oken. Tuto skutečnost lze označit jako problém s architekturou aplikace, **AD1**.

Některé metody by mohly být lépe dekomponované a tím by se zpřehlednila jak samotná struktura aplikace, tak i některé třídy. Jako příklad lze uvést třídy `FillCopyForms`, `FillForms`, `FillFromsXML`, které obsahují metody pro naplnění datové struktury konkrétního segmentu nebo elementu při kopírování prvku, samotném vytváření nebo načítání modelu z XML souboru. V případě kopírování se jedná o vytvoření nové datové struktury, kdy lze použít určitou část implementace ze třídy `FillFroms` a implementace ve třídě `FillFormsXML`, přičemž se jedná ve značné míře o stejné operace nad datovým modelem. Tento problém může být označen jako problém s dekompozicí programového kódu, **DD1**.

Při následné manipulaci a testování již hotové aplikace bylo odhaleno několik nedostatků jak z pohledu implementace grafického rozložení a obsahu, tak i v práci se samotným datovým modelem a manipulací s aplikací. Funkční chyby se týkají převážně ukládání prvků do XML souboru a načítání z XML souboru a dále se zde objevila nemožnost kopírování a vkládání jednotlivých elementů **Work Unit**.

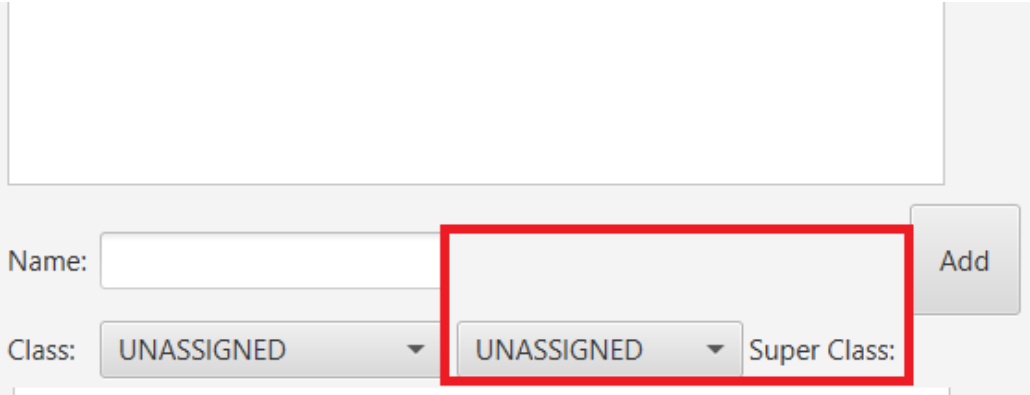
- **(FD1)** – Nefunkční vkládání instancí **Work Unit** z jednoho plátna na druhé (např. z hlavního projektového plátna na plátno fáze či iterace).

- **(FD2)** – Při exportování projektu do XML souboru je v případě elementu **Branch** uložena chybná hodnota u logického atributu *main*. Vždy uložena opačná hodnota oproti zvolené.
- **(FD3)** – Při načtení modelu z XML nejsou původní prvky **Change** a **Artifact** probarveny červenou barvou i v případě, že je zvolena neexistence prvku (lze požadovat, aby data explicitně neobsahovala prvek s určitými atributy; tato volba se právě červenou barvou vizuálně projevuje).
- **(FD4)** – Kontrola vstupů zadávaných do formulářů nefunguje vždy správně a místy je zmatečná.

Veškeré položky obsažené v tabulkových formulářích nejsou editovatelné. Jednou vytvořené položky je možné upravit pouze tak, že jsou smazány a vytvořeny znovu. Pokud jsou položky mazány, nedojde k tomu na všech místech, která mazané položky odkazují. Například, pokud je smazán některý záznam z přehledové tabulky pro element **Criterion**, je smazána konkrétní instance z datového modelu, ale nikoliv odkazy na ni z **Milestone**. Tento nedostatek lze označit jako chybná manipulace s datovým modelem, **DMD1**.

V oblasti grafických chyb se nejčastěji jedná o chyby, respektive nedostatky, převážně v práci s layoutem a chybějících informacích v přehledových tabulkách. Jednotlivé chyby budou očíslovány pro zlepšení přehlednosti a možné využití v následujících kapitolách.

- Layout formulářů **(VD1)** – U některých formulářů (konkrétně u formuláře **Change, Artifact a Work Unit**) se nelze dostat ke spodním ovládacím prvkům bez nutnosti zvětšit samotné okno.
- Chybějící sloupce v tabulkách **(VD2)** – U formuláře **Role** se nevyskytuje sloupec s informacemi o zadaném atributu popisu (*description*). U formuláře **Milestone** se pole pro zadání atributu *description* nenachází jak v zadávacím panelu ve spodní části, tak ani v přehledové tabulce.
- Chybný text ve formulářích **(VD3)** – Ve formulářích **Resolution** a **Status**, které slouží pro vytvoření výčtových typů pro element **Work Unit** se nacházejí špatné názvy v nabídkách jednotlivých typů. Ve formuláři **Role** se u atributu *role type* vyskytuje prohozený popisek a editační pole, viz obrázek 4.3.
- Resize oken **(VD4)** – Některá okna nefungují správně na změnu velikosti a obsah formulářů není zcela viditelný.



Obrázek 4.3: Chybně umístěný popisek pole

- **(VD5)** – Text jména prvku na plátně někdy přetéká mimo ikonu, která ho reprezentuje.

Pokud bychom chtěli pracovat s takto implementovanou aplikací dlouhodobě, mohla by být z pohledu uživatelské přívětivosti nepříjemná nutnost pokaždé vyvolávat okno pro editaci konkrétního segmentu nebo elementu a následně veškerá tato okna zase zavírat. Druhým problémem z pohledu uživatelské přívětivosti by mohl být přehuštěný horní panel s tlačítky pro přidání segmentu, respektive vyvolání přehledových tabulek o elementu s pouhými názvy konkrétních prvků. Jedná se tedy o problém s uživatelskou přívětivostí, **UFD1**.

4.3 Možná rozšíření aplikace

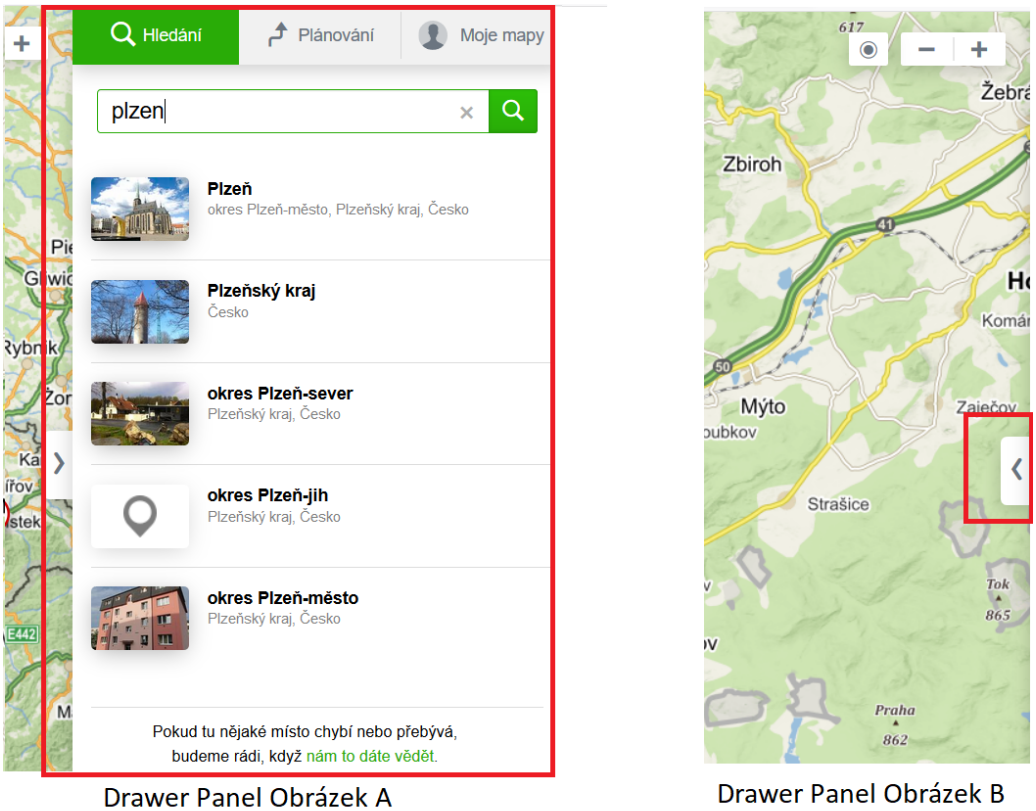
Tato podkapitola popisuje možné úpravy pro zlepšení vzhledu a použitelnosti aplikace. Budou zde navrženy některé možnosti odstranění zjištěných nedostatků z podkapitoly 4.2 a dosažení požadované funkčnosti.

V případě problému s uživatelskou přívětivostí, zmíněném v předešlé podkapitole, se aplikace stává značně nepřehlednou a nepraktickou. Z tohoto důvodu by bylo nejlepším řešením formulářová okna vložit přímo do hlavního okna aplikace pro vytvoření kompaktního vzhledu aplikace.

Jednou z možností, jak zabudovat okna formulářů do hlavního okna aplikace, je zmenšení kreslícího plátna a vložení pevného panelu na levou nebo pravou stranu okna. Při využití pevného panelu bohužel aplikace přijde o značnou část viditelné části plátna pro modelování prvků. Druhým problémem,

který se zde vyskytuje je nutnost editace jednotlivých prvků v editačních tabulkách. Editace by musela být prováděna opět za pomocí nového okna, které by se zobrazilo po vybrání konkrétní řádky v tabulce. Druhou možností editace je další pevný panel, ve kterém by se zobrazovaly konkrétní prvky pro editaci daného elementu. Zvolením druhé možnosti by došlo k ještě většímu omezení prostoru pro modelování procesu na plátně.

Pro eliminaci problému se zmenšením viditelné části plátna při využití pevného panelu lze využít grafickou komponentu `Drawer` panel[9]. `Drawer` panel je ve většině případů využíván jako navigační panel, který poskytuje způsob, jak jednoduchým způsobem pracovat s navigačními prvky v aplikaci. `Drawer` panel je viditelný pouze při určitých činnostech uživatele jako je například kliknutí na tlačítko, respektive reakce na některou uživatelskou činnost viz 4.4. Tato vlastnost umožňuje využívat překrytou plochu k důležitějším věcem, přesto udržovat ovládací prvky dosažitelné jedním kliknutím myši [9].



Obrázek 4.4: Ukázka Drawer panelu

K lepší uživatelské použitelnosti aplikace by také mohlo přispět rozdělení horního panelu s jednotlivými tlačítky pro vyvolání formulářů konkrétních elementů, stejně tak jako změna jednoduchých tlačítek s textem na ikonami konkrétního segmentu či elementu.

Pro zlepšení udržitelnosti a použitelnosti aplikace je potřeba vytvořit rozdělení mezi jednotlivými vrstvami aplikace. A tím eliminovat problém s architekturou aplikace, viz podkapitola 4.2. Při použití některého ze známých vzorů architektury a souvisejících prezentačních vzorů by se jednotlivé vrstvy měly od sebe oddělit a tím pádem zlepšit jak následný vývoje aplikace, tak i její přehlednost, udržitelnost a znovupoužitelnost určitých částí.

4.4 Požadavky na změnu implementace

Tato podkapitola popisuje požadavky na rozšíření funkčnosti (nad rámec dříve identifikovaných chyb) vzniklé analýzou schopnosti výchozí verze aplikace efektivně modelovat jednotlivé anti-patterny:

- **Změna elementů a segmentů určených pro plátno.** – Změnit množinu segmentů a elementů, se kterými lze pracovat na modelovacím plátně. Nově bude možné přidávat pouze tyto elementy: konfigurace (**Configuration**), role (**Role**), artefakty (**Artifact**), **Commit** a **Committed Configuration**.

- **Propojení mezi jednotlivými prvky nově vzniklé množiny prvků určených pro plátno.** – Umožnit vytvoření vazby mezi jednotlivými elementy umístěnými na plátně. Vytvořit mechanismus umožnující provádět kontrolu oprávněnosti vytvoření vazby mezi konkrétními prvky (např. nelze spojit **Artifact** a **Commit**, protože v datovém modelu přímá vazba mezi těmito entitami neexistuje).

- **Umožnění zobrazení počtu instancí prvku na plátně.** – Pro ulehčení práce s modelací procesu vývoje softwaru umožnit zadat počet instancí konkrétního elementu nebo segmentu umístěného na plátně a zobrazit tento údaj u konkrétního prvku. Ten pak reprezentuje skupinu prvků stejného typu se stejnými atributy, bez nutnosti vkládat každý z nich na plátno zvlášť. Dále umožnit zadat porovnávací kritérium pro četnost s operátory $<$, $>$, $=$ ke každému prvku.

- **Soft kopírování** – Umožnit kopírování a následné vložení prvku obsaženého na modelovacím plátně. Nově vzniklá instance objektu převezme

pouze základní údaje o objektu, jako jsou například jméno, počet instancí. Dále umožnit kopírování prvků obsažených v přehledové tabulce, taktéž pouze základní informace.

- **Vygenerování SQL dotazu.** – Vytvořit SQL dotaz založený na struktuře datového skladu SPADe z modelu vytvořeného v aplikaci a následné zobrazení dotazu s možností vykopírování dotazu. Zanechat ale i funkčnost ukládání modelů do XML formy.
- **Vyhledání patternu/anti-patternu v projektu.** – Umožnit výběr projektu z těch v datovém skladu nástroje SPADe a spustit nad ním vytvořené SQL dotazy z modelu. Zobrazit výsledky v přehledné formě (např. tabulkou).
- **Zvýraznění prvků plátna** – zvýraznit vybraný prvek na plátně, pomocí přerušovaného obrysu obdélníku.
- **Více násobný výběr prvků plátna** - Umožnit výběr více prvků umístěných na plátně, jejich společný pohyb a kopírování. Umožnit přidávat jednotlivé prvky do množiny po jednom pomocí klávesnice a myši.

Dále mezi nové funkční požadavky patří nedostatky zjištěné analýzou v podkapitole 4.2 shledané za nutné či vhodné k nápravě:

- **úprava grafické podoby aplikace,**
- **úprava architektury s ohledem na další rozšíření,**
- **odstranění nalezených chyb a nedostatků.**

5 Realizace grafického nástroje

Tato kapitola se zabývá možnostmi implementace požadovaných změn. Dále je zde popsána samotná implementace jednotlivých částí aplikace s některými ukázkami programového kódu.

5.1 Analýza realizovaných změn

V této podkapitole budou popsány změny provedené v aplikaci v závislosti na požadavcích uvedených v podkapitole 4.4.

5.1.1 Změny v GUI

K odstranění špatné uživatelské přívětivosti **UFD1**, viz podkapitola 4.3, bylo pro práci s jednotlivými formuláři, určenými pro zadávání dat do datového modelu aplikace, zvoleno přepracování stávajícího systému vyskakujících oken do systému využívajícího `Drawer` panelů a rozdělení editačních polí a přehledových tabulek do dvou částí. Tím by se měla zlepšit manipulace s aplikací a být odstraněna značná potřeba zavírání a otevírání editačních formulářů. Z pohledu editoru modelů procesů bude `Drawer` panel vyvolán pomocí výběrového seznamu nebo z již existujících tlačítek v horním panelu. `Drawer` panel bude využit i k vyřešení problému s editačním oknem, které bude umístěno na protější stranu od panelu s přehledovými tabulkami a vždy se objeví při výběru prvku v tabulce. Druhou možností umístění editačního panelu může být napojení panelu přímo vedle panelu s přehledovou tabulkou. Značná výhoda využití `Drawer` panelu se nachází v možnosti dané panely jedním kliknutím sbalit do základního stavu a využít celé plochy modelovacího plátna.

Jednou z možností jak získat funkčnost `Drawer` panelu je jeho vlastní implementace. Musel by být implementován animovaný přechod některého z již existujících layoutů v JavaFX pro překrývání kreslícího plátna a obslužná část událostí volaných nad daným panelem.

Druhou z možností je využít některou z již existujících knihoven určených pro usnadnění práce s grafickými prvky v JavaFX. Existuje značné množství

knihoven, které umožňují alespoň základním způsobem pracovat s `Drawer` panelem. Bohužel většina těchto knihoven je určena pro vývoj mobilních aplikací, například knihovna `Charm` [16]. Jednou z existujících knihoven určených pro desktop aplikace je knihovna `JFoenix`[22]. Tato knihovna obsahuje již implementovaný `Drawer` panel a jeho obslužné události. Umožňuje například automatické umístění panelu do základní polohy po kliknutí mimo panel, přizpůsobení velikosti prvků v okolí `Drawer` panelu, spolupráci jednotlivých `Drawer` panelů umístěných v aplikaci. V neposlední řadě lze také daný panel vyvolat jednoduchým vytažením ze základní polohy, případně jej zase jednoduchým tahem skrýt. Tato vlastnost by zlepšila použitelnost aplikace na dotykových přístrojích, jako jsou tablety nebo all-in-one počítače.

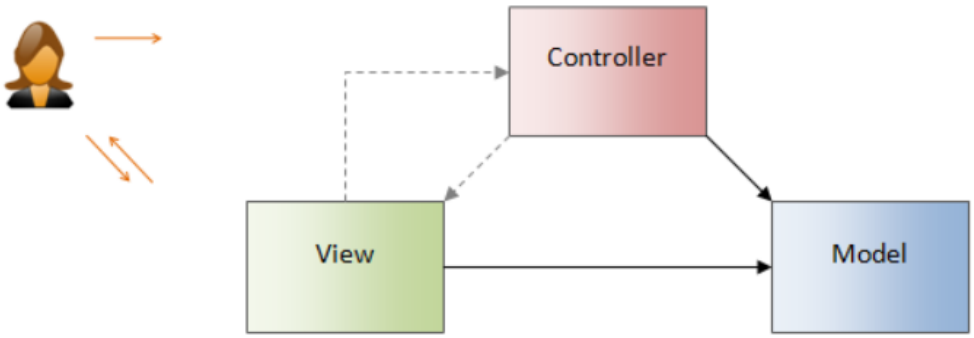
Druhou zásadní změnou v GUI je přepracování hlavního panelu aplikace umístěného v horní části obrazovky. Panel by měl být zpřehledněn odstraněním tlačítek sloužících k vyvolání přehledových tabulek a ponechání pouze tlačítek umožnujících přidání jednotlivých prvků na modelovací plátno. K vyvolání přehledových tabulek umístěných v `Drawer` panelu, lze využít komponenty `ComboBox` s možností výběru jednotlivých elementů a segmentů, nebo vytvořit postranní panel s tlačítky. Pro zpřehlednění vzniknou jednoduché ikony reprezentující jednotlivé typy elementů, které lze přidat na plátno. Vznikne také ikona pro samotný projekt a tlačítko umožnující vytvoření spojení prvků na plátně. Vzniklé ikony spolu se zjednodušením samotného panelu by měly přispět ke snazší manipulaci a větší přehlednosti aplikace.

5.1.2 Architektonické změny

Jedním z hlavních problémů zjištěných v rámci analýzy v podkapitole 4.2 je problém s architekturou aplikace **AD1**. Proto je potřeba přepracovat jednotlivé vrstvy aplikace, jednak pro zlepšení možnosti její testovatelnosti, tak pro její následné rozšíření a vývoj. Jako možnosti pro přepracování architektury mohou být využity následující architektonické vzory.

MVC (Model - View - Controller) je jeden z nejvíce používaných prezentačních vzorů vycházející ze stejnojmenné architektury. Architektura MVC dělí aplikaci na 3 logické celky [23] [28], Model, View a Controller, viz obrázek 5.1. Model prezentuje aplikační logiku a data. View slouží k zobrazení dat uživateli. Controller slouží ke kontrole vstupů od uživatele a modifikuje datový model aplikace. Změny v datovém modelu je potřeba předat dále do všech View, kterých se daná změna týká. K tomuto účelu je nejvýhodnější využít návrhový vzor Observable. View a Controller jsou zaregistrovány do

modelu, který při změnách stavu oznamuje zaregistrovaným komponentám, že je potřeba načíst aktuální data. Velmi často je využíváno hierachického přístupu, kdy je MVC vytvořeno pro každou komponentu.

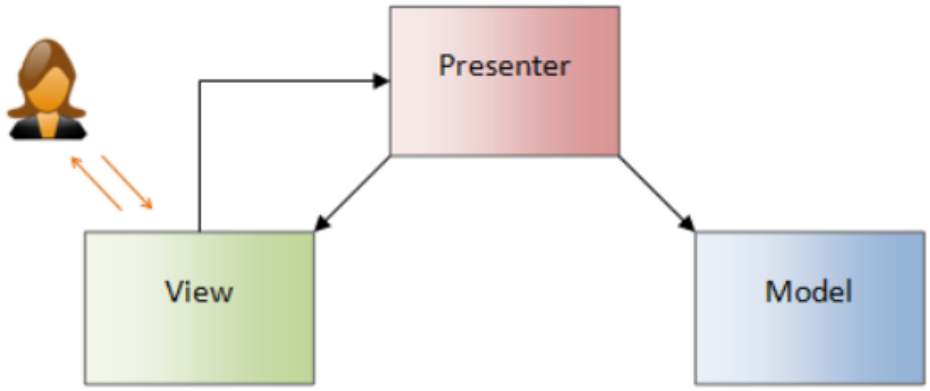


Obrázek 5.1: Ukázka prezentačního vzoru MVC

MVP (Model - View - Presenter) je prezentační vzor z větší části podobající se vzoru MVC. Oproti klasickému MVC byl Controller vyměněn za takzvaný Presenter. Presenter má v tomto případě silnější vazbu na View, než tomu je v případě MVC[28]. Obsahuje aplikační a prezentační logiku, manipuluje s Modelem a pomocí systému notifikací zajistí aktualizaci View, nebo ovlivňuje View přímo. Samotné View má také typicky přímou vazbu na Presenter a je využíváno ke kontrole vstupů a výstupů. Typická činnost View by měla být pouze to, že v reakci na nějakou událost zavolá některou z metod Presenteru. MVP se dále dělí na dvě hlavní variace – Supervising Controller a Passive View – v závislosti na tom jak velkou zodpovědnost má View [4].

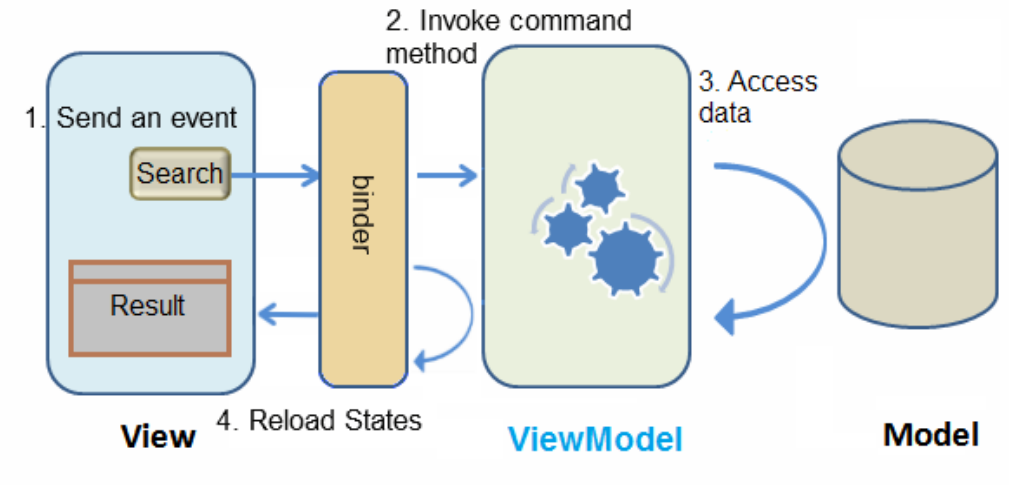
Vzor Supervising Controller Při použití vzoru Supervising Controller má vrstva View zodpovědnost za zobrazování dat z Modelu buď pomocí data bindingu nebo vzoru Observer. Veškerá složitější prezentační logika se nachází v Presenteru, který View libovolně ovlivňuje.

Vzor Passive View zcela odstranil vazbu View na Model, viz obrázek 5.2. Z tohoto důvodu má Presenter mnohem větší zodpovědnost, protože musí správně synchronizovat View a Model a také tvořit prezentační logiku aplikace. Tento vzor vznikl z důvodu usnadnění automatických testů. Samotné View se velmi těžko pokrývá automatickými testy, a proto je veškerá logika přesunuta do Presenteru, který se lépe testuje.



Obrázek 5.2: Ukázka prezentačního vzoru Passive View

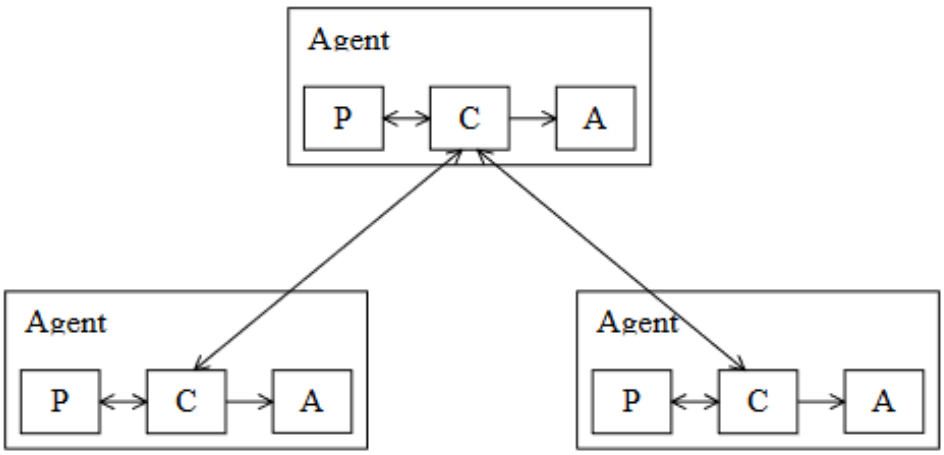
MVVM (Model - View - ViewModel) Vzor MVVM (Model - View - ViewModel), nebo-li vzor Presentation Model (PM) je ilustrován na obrázku 5.3). Největší rozdíl od MVC se nachází v tom, že View již nepřistupuje k Modelu přímo, ale přes mezi vrstvu, kterou tvoří kombinace Modelu a Presenteru. Z tohoto pohledu se vzor PM velice podobá vzoru MVP, je zde ale využíván návrhový vzor Adapter. Samotný binding dat mezi View a ViewModelem je tvořen pomocí frameworku dané technologie. Případně můžeme aplikaci doplnit o separátní kontrolery [4].



Obrázek 5.3: Ukázka prezentačního vzoru MVVM [11]

PAC (Presentation - Abstraction - Control) rozděluje GUI aplikaci jako hierarchii splupracujících agentů, viz obrázek 5.4. Agent je tvořen pomocí tří odlišných komponent [23]. Komponenta prezentace zobrazuje

data a zpracovává vstupy od uživatele. Komponenta abstrakce obsahuje data specifická pro agenta. Komponenta Control slouží ke komunikaci s ostatními agenty. Následná komunikace mezi agenty vypadá tak, že agent v nejspodnější vrstvě odešle svůj požadavek na svého rodičovského agenta, který se nachází o vrstvu výše v dané hierarchii. Tento agent buď předá zprávu některému ze svých dětí, nebo ji předá o vrstvu výše svému rodiči. Agent na nejvyšší úrovni odpovídá za prvky GUI na úrovni aplikace, jako jsou menu nebo lišty nástrojů. Druhou funkcí agenta na nejvyšší úrovni je koordinace všech ostatních agentů.



Obrázek 5.4: Ukázka prezentačního vzoru PAC [23]

Po prozkoumání jednotlivých architektonických vzorů MVC, byl vybrán přechod na styl MVP ve formě Passive View převážně z důvodu možnosti odděleného testování jednotlivých částí GUI, Modelu a Presenteru. Za účelem přepracování architektury aplikace bude aplikace rozdělena do nové struktury balíků.

Pro odstranění problému s dekompozicí kódu zjištěným taktéž při analýze v podkapitole 4.2. Bude přistoupeno k úplnému odstranění tříd `FillCopyForms` a `FillFormsXML`. Funkce, které tyto třídy umožňovaly například kopírování a vyjmutí jednotlivých prvků na plátně, budou přesunuty do třídy `ManipulationController`, která bude následně sloužit k volání metod třídy `FormFillController` a převezme metody k editaci datového modelu, pokud se jedná o akce vložení nebo vyjmutí prvku obsaženého na modelovacím plátně. Další z funkčností, kterou tato třída bude plnit v případě načítání

modelu z XML souboru, je nastavení datových struktur, jako jsou počítadla jednotlivých instancí datového modelu. Dále bude řešit vytvoření a vložení aliasů z datového modelu do seznamů pro konkrétní `ComboBox`, respektive `CheckComboBox`. Dekompozice jednotlivých částí metod třídy `FormFillController` umožní jednotný přístup pro obě zmíněné funkčnosti a napomůže k přehlednosti jak z pohledu programového kódu, tak i z celkového pohledu na aplikaci.

5.1.3 Odstranění nalezených chyb a nedostatků

Během analýzy existující implementace editoru procesů vývoje softwaru, byly také zjištěny grafické a funkční chyby viz podkapitola 4.2. Grafické chyby **VD1, VD2, VD3 a VD4** budou vyřešeny změnou GUI. Jednak bude sjednocena velikost editačního okna, konkrétně přechodem na přehledový a editační `Drawer` panel, jednak budou zjednodušeny přehledové tabulky pouze na dva sloupce – sloupec s aliasem (uživatelem definovaný název instance) a příznakem (ne-)existence konkrétní instance vytvářeného modelu. Pro odstranění grafické chyby **VD3**, vznikne jednotný systém pro vytváření editačního panelu, dle dat obsažených v datovém modelu, na místo vytváření editačního pole pro každý formulář staticky. Bude také proveden menší zásah do logiky vytváření a práce s ikonami reprezentujícími jednotlivé instance obsažené na modelovacím plátně (spolu s jejich grafickou úpravou) pro odstranění grafického nedostatku **VD5**.

Většinu funkčních chyb by mělo odstranit přepracování architektury a dekompozice balíků a programového kódu aplikace. Funkční chyby **FD2 a FD3** budou odstraněny dekompozicí třídy `FillForms` a přechodem na vykreslení grafických prvků (ale i samotných editačních panelů) přímo z dat v datovém modelu. To bude v kontrastu s předchozím stavem držení dat ve formulářích a jejich editace při každé změně datového modelu. Tato dekompozice také odstraní problém s funkční chybou **FD1** (kopírováním jednotlivých prvků mezi plátny). K vyřešení této chyby napomůže zjednodušení systému s modelovacími plátny na jedno hlavní plátno, nacházející se v hlavní obrazovce aplikace.

V důsledku funkční chyby **FD5** bude vytvořen nový systém kontroly vstupů, kdy vzniknou přetížené metody pro kontroly jak konkrétních hodnot, tak i například možnost ověření, zda je číselná hodnota z určeného intervalu. S kontrolou vstupních dat také souvisí přepracování událostí vyvolávajících varovná okna včetně jejich obsahu.

5.1.4 Nová množina prvků plátna

Mezi nové funkční požadavky patří změna elementů a segmentů určených pro plátno. Množina by měla být omezena na elementy typu konfigurace (**Configuration**), osoba (**Person**), artefakt (**Artifact**), commit (**Commit**) a **Committed Configuration** (představující konfiguraci s odlišnými časy vytvoření a potvrzení). Předešlá množina segmentů a elementů bude muset být přesunuta do výběrového seznamu, viz podkapitola 5.1.1. Tímto krokem vznikne zjednodušení logiky pro vytváření spojnic mezi jednotlivými prvky na modelovacím plátně. Odpadne nutnost vytvářet dva typy spojnic a bude nutno vytváře pouze jednoduchý typ spojení bez možnosti vybrání typu relace, jako tomu bylo v případě spojování prvků typu **Work Unit**. Odstraněním nutnosti hlídat, ze kterého prvku určitá vazba vychází, lze vytvořit funkci hlídání směru spojnice a tím dosáhnout zlepšení přehlednosti prvků a jejich spojnic na plátně.

Pro odstranění nutnosti kopírování či vytváření nového prvku obsaženého na modelovacím plátně, který má stejné vlastnosti jako některý z jiných prvků. Vznikne možnost zadání počtu instancí konkrétního elementu v editačním panelu. Zadaný počet by se následně měl zobrazit u obrázku představující prvek plátna.

Možnost volby počtu existujících elementů poskytuje více možností, jak pracovat s jednotlivými instancemi objektů v datovém modelu a s relacemi mezi nimi při vyhledávání modelu v projektových datech. Jednou z možností, jak s relacemi pracovat, je vyžadovat vazbu M:N, tedy požadovat, aby počet instancí jednoho elementu byl v relaci s určitým počtem druhého elementu. Jako další možnost lze využít vazbu 1:M, kdy by jedna instance elementu byla v relaci s určitým počtem relací druhého elementu. V tomto případě by musel být opět hlídán počáteční a koncový prvek. Poslední možnost je relace 1:1, tedy skutečnost, aby alespoň jedena instance z množiny jednoho elementu byla v relaci s jednou instancí z množiny druhého elementu. Pro zjednodušení práce s datovým modelem byla vybrána poslední možnost relace tedy 1:1.

5.1.5 Napojení na datový sklad SPADe

Funkčnost nástroje bude rozšířena o možnost napojení se na datový sklad nástroje SPADe. Datový sklad je v současné době implementován pomocí MySQL databáze. Pro vytvoření databázového spojení jsou k dispozici následující možnosti.

První je využití frameworku `Hibernate`[18], který je napsán v programovacím jazyce Java a umožňuje objektově-relační mapování (Object-Relational Mapping – ORM). Tedy mapování Java objektů na entity v relační databázi. Mapování může být provedeno dvěma způsoby. Mohou být využity mapovací soubory, ve kterých je popsána transformace dat z objektů do tabulek a naopak. Druhým způsobem může být využití anotací. Hibernate pracuje s klasickými objekty Plain Old Java Object (POJO)[8] a využívá dotazovací jazyk Hibernate Query Language (HQL)[21] odvozený od klasického SQL. Framework může být využit ve více prostředích podporujících Java Persistence API například Java SE, prostředí aplikačních serverů Java EE nebo kontejnerů Enterprise OSGi[34]. Hibernate je vysoce konfigurovatelný, rozšířitelný a umožňuje pracovat s databázovým systémem nezávisle na jeho typu.

Druhou možností je přímé využití rozhraní pro přístup k relační databázi Java Database Connectivity (JDBC). Pomocí JDBC lze vytvořit spojení s databází a následně přes SQL dotaz měnit či získávat obsah z databáze. Při použití JDBC je potřeba implementovat specifický ovladač požadované databáze. Pro připojení k MySQL databázi je potřebné implementovat ovladač MySQL JDBC, který se nazývá MySQL Connector/J [29].

5.1.6 Možnost vyhledání vzorů v datovém skladu

Aplikace bude rozšířena o možnost vyhledání namodelovaného procesu vývoje softwaru, nebo anti-patternu v již existujících datech, uložených v datovém skladu nástroje SPADe. Pro možnost hledání vzorů v existujících datech bude využito připojení do databázového systému popsaného v 5.1.5. Vznikne několik Data Access Object (DAO) tříd, ve kterých budou vytvářeny dotazy v závislosti na datech obsažených v jednotlivých entitách datového modelu.

Pro samotný převod modelu do SQL a získání jednotlivých dotazů bude potřeba projít veškeré instance datového modelu a pro každý prvek poskládat jeho parametry do textového řetězce. Pro jednotlivé prvky datového modelu vznikne statické spojení potřebných tabulek pomocí příkazu `join`, ke kterému bude přidán textový řetězec z předešlého kroku. Pro jednotlivé elementy a segmenty datového modelu, které jsou v relaci, bude potřeba získat jejich konkrétní identifikátory v databázi a následně je přidat výslednému textovému řetězci představující SQL dotaz. Jednotlivé dotazy bude potřeba omezit na konkrétní projekt, bude tedy potřeba uchovávat jeho *ID*. Po spuštění dotazu

budou výsledky uloženy do seznamu, který bude následně využíván i k určení počtu existujících elementů či segmentů v datovém skladu nástroje SPADe.

Funkce ověření vytvořeného modelu v databázi bude umožněna pomocí tlačítka umístěného do hlavního panelu aplikace. Nejprve bude požadováno zadání přístupových údajů do staticky zvolené databáze. Dalším krokem bude zvolení projektu, ve kterém se má vzor hledat. Vznikne logika vyhodnocení dat vrácených z databáze a následné zobrazení výsledků do nového okna, viz obrázek 5.5. Okno bude obsahovat tabulku s následujícími sloupci: alias elementu nebo segmentu obsaženého v modelu spolu s informací o existenci či neexistenci v daném vzoru, počet instancí (v případě elementů obsažených na plátně), informaci o tom, zda byl konkrétní element nalezen v projektu, výsledek vyhledání a samotný SQL dotaz, pomocí kterého byl získán výsledek. Pro přenos SQL dotazů a jejich použití mimo aplikaci bude umožněno jejich vykopírování rozkliknutím konkrétního řádku tabulky a vyvoláním informačního okna s dotazem.

Model elemet alias	Model exist	Instance count	Project exist	Result	SQL query
Phase1	YES	----	YES	✓	Select id FROM phase

Obrázek 5.5: Návrh okna s výsledky po vyhledání modelu

5.2 Implementace

V této kapitole bude popsána implementace aplikace, která je výsledkem praktické části této práce (dále jako nástroj). Nástroj byl vyvíjen ve vývojovém prostředí IntelliJ Idea 2018 a bylo navázáno na předchozí implementaci v jazyce Java spolu s frameworkem JavaFX, verzovacím systémem Git a z velké části také na datový model. K vývoji bylo také využito Apache Maven

pro správu, řízení a automatizaci sestavení aplikací. Soubor Project Object Model (POM) obsahuje definice závislostí externích knihoven, kterými jsou:

- `SPADEPAC` – JAR archiv obsahující třídy datového modelu vygenerovaného z XML pomocí XSD schématu,
- `Controlsfx` – knihovna potřebná k rozšíření funkčnosti frameworku JavaFx,
- `JFoenix` – knihovna umožnující funkčnost `Drawer` panelu,
- `JUnit` – knihovna pro jednotkové testovaní aplikace,
- `MySQL` – knihovna pro komunikaci s databázovým systémem.

Pro práci s datovým skladem nástroje SPADe, který je v současné době realizován pomocí MySQL databáze, bylo využito rozhraní JDBC.

5.2.1 Struktura aplikace

Oproti původní implementaci přibyly balíky:

- `abstractControlPane`,
- `abstracontrolPanels`,
- `controllers`,
- `model`,
- `database`.

V balíku `graphics` přibyly nové pod-balíky v závislosti na konkrétní grafické funkčnosti obsažených tříd. Balík `controller` je dále dělen na `formController` pro řízení dat formulářů a `graphicsComponentsControllers` pro logiku grafických komponent. Následující obrázek 5.6 ukazuje rozložení balíků aplikace do jednotlivých vrstev vybraného architektonického stylu MVP. Pro zpřehlednění a snazší možnost případné úpravy nebo rozšíření aplikace vzniklo k již existujícím rozhraním několik nových. Rozhraní vznikla jak pro jednotlivé kontrolery, tak hlavně pro práci s datovým modelem. Jedná se o rozhraní:

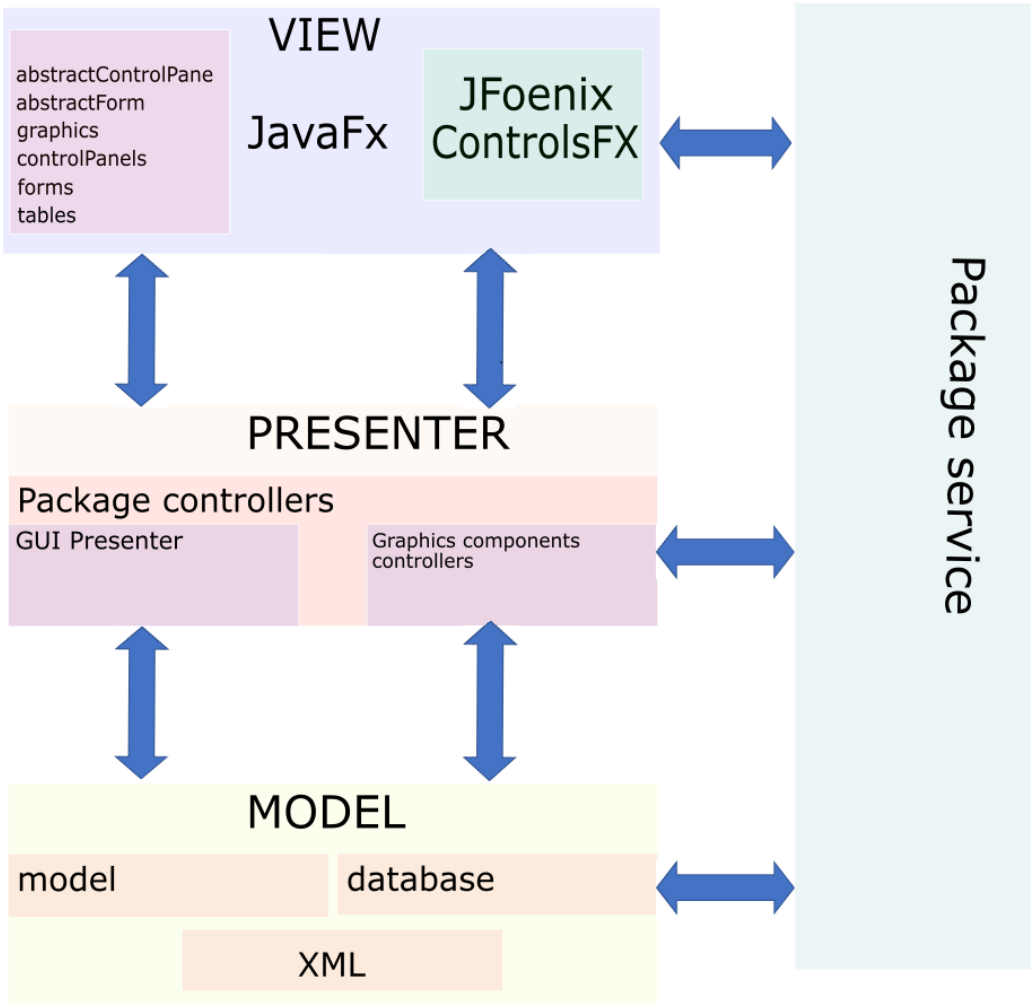
- `IControlPanel` – rozhraní obsahující metodu pro vyvolání Drawer panelu,

- `IDeleteDataModel` – rozhraní obsahující metody umožnující odstranění jednotlivých instancí elementů a segmentů ze seznamů datového modelu,
- `IDeleteFormController` – rozhraní obsahující metody, které slouží k odstranění dat závislých na formuláři,
- `IEditDataModel` – rozhraní obsahující metody k editaci jednotlivých instancí elementů a segmentů ze seznamů datového modelu,
- `IEditFormController` – rozhraní obsahující metody, které slouží k editaci dat závislých na formuláři,
- `IFormDataController` – rozhraní obsahující metody využívané pro vytvoření jednotlivých instancí tabulkových elementů a segmentů, společně s naplněním potřebných datových struktur (přehledové seznamy a tabulky).
- `ISavaDataModel` – rozhraní využívané pro vytvoření instancí v datovém modelu.

5.2.2 Datový model a kontrolery

Implementace jednotlivých rozhraní spadajících pod datový model se nachází v balíku `model`. Rozhraní `IDeleteDataModel` je implementováno třídou `DeleteDataModel`. Metody uvedené v této třídě nejprve odzadu projdou seznam indexů jednotlivých prvků určených k odstranění předaný parametrem metody a vždy odstraní konkrétní instanci na daném indexu ze seznamu uloženém ve třídě `DataModel`. V případě, že se jedná o objekty umístěné na modelovacím plátně, je požadováno pouze *ID* objektu místo seznamu. Třída také obsahuje metody, které slouží k odstranění relací mezi jednotlivými objekty plátna. Metoda pomocí identifikátorů relace počátečního a koncového *ID* prvku odstraní spojení z datových struktur.

Třída `EditDataModel` je implementací rozhraní `IEditDataModel`. Metody pro editaci elementů a segmentů vždy najdou konkrétní instanci v seznamu dle *ID* předaného parametrem a následně zavolají metodu z datového modelu pro naplnění dané instance konkrétními daty, která jsou předána pomocí parametrů. Metoda `updateItemList` a její přetížené varianty slouží k odstranění referencí na mazaný objekt z datových struktur. Pomocí konstrukce `switch-case` je nalezena metoda pro editaci konkrétních objektů. Následně je pomocí konstrukce `foreach` prozkoumán celý seznam existujících instancí závislého



Obrázek 5.6: Ukázka rozložení balíků do MVP

prvku a hledán konkrétní index tohoto prvku. Ten je posléze odstraněn z datových struktur.

Implementace rozhraní `ISaveDataModel` je realizována ve třídě `SaveDataModel`. Obsahuje metody pro vytvoření nové instance segmentu nebo elementu. V každé metodě je nejprve zavolána tovární třída `ObjectFactory` z knihovny `SPADEPAC` pro vytvoření konkrétního objektu. Pokud se jedná o prvek patřící do tabulkových formulářů, je mu následně přiřazeno ID vytvořené třídou `IdentificatorCreater`, nastaven příznak existence a alias. Instance je následně přidána do seznamu v datovém modelu. Pokud se jedná o prvek určený na plátno, jsou mu také nastaveny souřadnice na defaultní pozici a počet instancí na 1. Metody pro vytvoření relací mezi prvky na plátně nejprve vytvoří instanci objektu `Link` z knihovny `SPADEPAC` a nastaví do ní

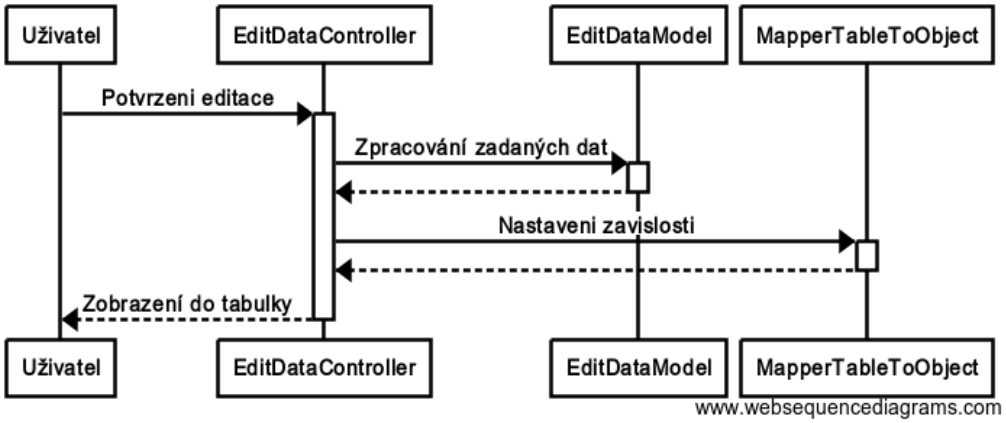
identifikátory spojovaných prvků a typ relace. Následně přiřadí identifikátory do příslušných instancí prvků v datovém modelu.

Třída `DataManipulator` obsahuje metody pro kopírování. Nejprve je vytvořen nový prvek v datovém modelu a nahrána do něho data z kopírovaného objektu. Dále se zde nachází metody pro převod dat obsažených v datovém modelu do datového seznamu pro zobrazení v GUI.

Třída `DataModel` obsahuje instanci třídy `Project` z knihovny `SPADEPAC`, která obsahuje seznamy s jednotlivými elementy a segmenty. Metody obsažené v `DataModel` umožňují získat instance těchto prvků jednak pomocí vytvořených identifikátorů, jednak i pomocí jednotlivých indexů v seznamu, případně lze získat index v seznamu pomocí *ID*.

Implementace rozhraní `IDeleteFormController` je realizována ve třídě `DeleteFormController` a obsahuje metody pro odstranění jednotlivých elementů a segmentů. Na začátku každé metody je zavoláno informační okno s *aliasy* prvků, které se budou mazat. Pod identifikací mazaného prvku jsou následně do řádků vypsány skupiny *aliasů* segmentů a elementů, které jsou na mazané instanci závislé. Pokud je potvrzeno smazání dané instance, je následně smazána z přehledového seznamu určeného pro `ComboBox` nebo `CheckComboBox` a jsou aktualizovány reference v `HashMap` určené pro daný typ prvku datového modelu ve třídě `MapperTableToObject`. Dále jsou zavolány metody pro úpravu referencí v objektech, které jsou závislé na mazané instanci z rozhraní `IEditDataModel` a smazání samotné instance z datového modelu pomocí metody rozhraní `IDeleteDataModel`.

Implementace rozhraní `IEditFormController` je realizována třídou `EditFormController`. Metody v této třídě slouží ke kontrole a ukládání dat z editačních panelů do datového modelu. Nejprve je provedena kontrola textových vstupů. V případě špatně zadaného vstupu, je vyvolána výjimka a vyvoláno informační okno s nastalým problémem. Pokud je vstup v pořádku, je provedena konverze zadaných dat pro datový model a zavolána konkrétní metoda z rozhraní `IEditDataModel`. Následně jsou změněny informace v přehledových seznamech a reference v `HashMap` ve třídě `MapperTableToObject`. Pokud se jedná o editaci prvku obsaženého na modelovacím plátně, jsou danému prvku nastaveny vlastnosti jako je alias, počet instancí a nastavení barevného provedení zobrazujícího příznak (ne-)existence konkrétního elementu v procesu. Tok dat mezi jednotlivými třídami v případě editace je zobrazen na obrázku 5.7.



Obrázek 5.7: Sekvenční diagram volaných tříd při editaci elementu

Třída `FormDataController` slouží jako implementace rozhraní `IFormDataController`. Jedna část metod obsažených v této třídě, potažmo v rozhraní, slouží k vytvoření jednotlivých elementů a segmentů v datovém modelu (pomocí metod třídy `ISaveDataModel`) společně s jejich inicializací v přehledových seznamech. Dále pak v závislosti na tom, o jaký typ entity se jedná, jsou inicializována data v přehledových tabulkách, respektive v prvku modelovacího plátna. Druhou oblastí metod obsažených v této třídě je vytvoření relací mezi jednotlivými instancemi obsaženými na plátně. Jsou volány konkrétní metody z rozhraní `ISaveDataModel` a vytvořeny reference v `HashMap` ve třídě `MapperTableToObject`.

5.2.3 Přehledový a editační panel

Přehledový a editační postranní panel byly implementovány pomocí knihovny `JFoenix`. Hlavní okno aplikace vytvořené pomocí třídy `MainWindow` obsahuje komponentu `JFXDrawersStack`, do které je předána instance objektu `DragAndDropCanvas` představující modelovací plátno. Komponenta `JFXDrawersStack` umožňuje automatickou úpravu velikosti modelovacího plátna v závislosti na tom, zda je zobrazen některý z `Drawer` panelů.

Oba vytvořené `Drawer` panely jsou ovládány pomocí kontroleru `DrawerPanelController`, ve kterém je každému panelu nastaven směr zobrazení, jeho velikost a pro zlepšení uživatelské přívětivosti funkce, schování panelu po kliknutí mimo konkrétní `Drawer` panel. Pro vyvolání levého panelu s přehledovými tabulkami slouží metoda `showLefPanel`, do které je ze třídy `FormController` předán konkrétní formulář, jenž je následně přidán do panelu a zobrazen. Pravý panel je vyvolán pomocí metody `showRightPanel`.

Tato metoda je volána z jednotlivých formulářů obsahující přehledovou tabulku zvolením jedné z položek obsažených v tabulce. `Drawer` panel obsahuje instanci třídy `ControlPanel`.

Změna GUI, konkrétně systému samostatných oken pro editační a přehledové formuláře do jednotného tvaru, umožnila zjednodušení hierarchie dědění formulářových tříd. Zůstala základní abstraktní třída `BasicForm`, která dědí vlastnosti od layout panelu `BorderPane` a ve které se nachází také abstraktní metoda `createForm`, kterou musí jednotlivé formulářové třídy implementovat. Dále byla upravena abstraktní třída `TableBasicForm`, která je nově oděděna od třídy `BasicForm` a obsahuje čtyři editační tlačítka (pro přidání prvku do tabulky, odstranění prvku z tabulky, duplikace prvku a vyvolání editačního panelu pro vybraný řádek v tabulce). Tato tlačítka a jejich funkčnost jsou vytvořeny v metodě `createPanel`. Metoda `deleteItem` je univerzální metodou pro mazání dat z přehledové tabulky a datového modelu pomocí metod rozhraní `IDeleteFormController`. Nachází se zde také abstraktní metoda `setEventHandler` pro nutnost nastavení obsluhy událostí (instanci `EventHandler`) pro konkrétní přehledovou tabulku. Na další úrovni hierachie dědění je třída `TableClassBasicForm` sloužící ke sjednocení funkčnosti formulářů představujících výčtové typy elementu **WorkUnit**. Na nejnižší úrovni hierarchie se nachází samotné třídy představující jednotlivé formuláře pro segmenty a elementy. Tyto formuláře implementují rozhraní `ISegmentTableForm` a obsahují instanci přehledové tabulky `TableView` a panelu `ControlPanel` pro editaci dat o daném prvku datového modelu. Metoda `addItem` slouží k vytvoření nové položky do přehledové tabulky pomocí kontroleru `FormController`, což automaticky otevře editační panel.

Editační panel byl implementován tak, že do něho lze dynamicky přidávat parametry konkrétního elementu a segmentu. Základem dynamického systému pro přidání parametrů je třída `ControlPanelLine`, která slouží jako kontejner pro grafické prvky vkládané do `GridPane` panelu třídy `ControlPanel`. Obsahuje tedy `ComboBox` pro výběr parametru prvku. Rozlišujeme sedm typů parametrů: textový vstup, číselný vstup, výběr kalendářního data, výběr z `ComboBoxu`, výběr z `CheckComboxu`, volbu pomocí `RadioButton` komponenty a výběr typu relace mezi prvky typu **Work Unit**. Tyto možnosti parametrů jsou popsány ve třídě výčtového typu `ControlPanelLineType`. Pro sjednocení vlastností řádků jednotlivých parametrů vznikla opět hierarchie tříd v závislosti na typu parametru (popsaná níže v této podkapitole).

Kontejner `ControlPaneLine` může být vytvořen pomocí dvou konstruktorů. První typ konstruktoru slouží k inicializaci objektů určených pro přidání do editačního panelu a druhý konstruktor umožňuje přidat výběrový seznam do komponenty `ComboBoxItem` a je určený pro výčtové atributy elementu typu **Work Unit** (**Priority, Severity, Relation, Resolution, Status, Type**). Metoda `setType` slouží k nastavení typu parametru, který bude vybraný pro editaci. Využívá k tomu data uložená v objektu třídy `ControlPanelLineObject`. Tato třída slouží jako kontejner uchovávající informace o typu editačního řádku zvoleného parametru spolu s referencemi na přehledové seznamy určené pro výběrové komponenty. Každý řádek obsažený v editačním panelu obsahuje možnost volby, zda mají být data v něm obsažená přidána do datového modelu, pomocí komponenty `RadioButton`. Metody `fillTextLine`, `fillNumberLine`, `fillCheckComboBoxLine`, `fillComboBoxLine` a `fillDateLine` třídy `ControlPanelLine` slouží k nastavení editačního panelu z již existujících dat v datovém modelu.

Jak bylo zmíněno výše, pro jednotlivé typy parametrů řádku vznikla hierarchie tříd. Základ hierarchie tvoří třída `ItemBox`, která obsahuje komponentu `ComboBox` pro vybrání indikátoru označujícího v jaké formě se má daná hodnota vyskytovat v modelu. V případě, že se jedná o číselnou hodnotu nebo kalendářní datum může být hodnota větší, menší, rovna. V případě textových dat existují čtyři možnosti jak s hodnotou pracovat: existence textu jako celku (IS), neexistence textu jako celku (IS NOT), obsahuje daný text (CONTAINS), neobsahuje daný text (NOT CONTAINS). Změna operátoru je zaznamenána pomocí `ChangeListener`. Seznam je do výběrové komponenty vkládán pomocí konstruktoru třídy. `ItemBox` obsahuje také tlačítko pro vytvoření nového řádku pomocí kontroleru `ControlPanelController` a nakopírování dat z dan ého řádku do nově vytvořeného.

Pro jednotlivé typy řádků v závislosti na potřebné grafické komponentě vznikly následující třídy: `CheckComboBoxItem`, `ComboBoxItem`, `DateItem`, `TextFieldItem` a `RadioButtonItem`. K odlišení od grafických prvků nacházejících se v JavaFX, respektive v knihovně `ControlFX`, byl název třídy doplně o koncovku `-Item`. Objekty obsahují instanci konkrétní komponenty, kterou zapouzdřují, společně s vlastnostmi objektu `ItemBox`.

Dle vlastností vycházejících z jednotlivých prvků datového modelu byl vytvořen systém postupně dědících abstraktních tříd a tříd editačních panelů. Základní abstraktní třída je označena jako `ControlPanel` a obsahuje grafické prvky reprezentující všechny společné vlastnosti z datového modelu. Objekt

`ControlPanel` dědí vlastnosti od `ScrollPane` pro možnost posuvu při velkém množství dat obsažených v panelu. Pro rozložení grafických prvků do panelu je využito layout panelu `GridPane`. Metoda `createMainPanel` vytvoří konkrétní instanci `GridPane`, které nastaví požadované vlastnosti, jako jsou mezery mezi grafickými prvky a jejich vycentrování na střed panelu. Instance `GridPane` je následně přidána do panelu `ScrollPane`. V metodě `createMainPanel` je také vytvořeno editační tlačítko, které je pro všechny editační panely stejné, spolu s nastavením konkrétní barvy a vložením do `GridPane`. Pro udržení informací o jednotlivých řádcích obsažených v editačním panelu byl zvolen `ArrayList` s instancemi třídy `ControlPanelLine`. Nachází se zde také seznam uchovávající instance třídy `ControlPanelLineObject` pro ukládání typů jednotlivých řádků. Do tohoto seznamu je v konstruktoru každé třídy oddědění od `ControlPanel` přidán záznam o tom, jaké parametry mohou být editovány a přidány do panelu.

Pro zpřehlednění a zjednodušení práce s editačními formuláři vznikly další abstraktní třídy využívající základní funkčnost od `ControlPanel`. Jedná se konkrétně o třídy:

- `NameControlPanel` – editační panel bude obsahovat parametr *Name*. Parametr *Name* je obsažen v každém editačním panelu, proto ostatní třídy vycházejí z vlastností této třídy a musí implementovat abstraktní metodu `createBaseControlPanel`,
- `DateControlPanel` – editační panel bude obsahovat parametry *Name* a *Date*,
- `DateDescControlPanel` – editační panel bude obsahovat parametry *Name*, *Date* a *Description*,
- `DescriptionControlPanel` – editační panel bude obsahovat parametry *Name* a *Description*,
- `WorkUnitControlPanel` – editační panel bude obsahovat parametry *Name* a *Work Unit*,
- `WorkUnitDateControlPanel` – editační panel bude obsahovat parametry *Name*, *Date* a *Work Unit*.

Třídy obsahují také dvě abstraktní metody `showEditControlPanel` a `addItemsToControlPanel`. První jmenovaná metoda slouží k získání dat z datového modelu a pomocí kontroleru `ControlPanelController` vytváří potřebný počet řádků a dané řádky plní získanými daty. Druhá jmenovaná

metoda slouží k vytvoření jednoho nového prázdného řádku v panelu, opět za pomoci kontroleru `ControlPanelController`.

Třídy představující vytvořené editační panely získají parametry z již zmíněných abstraktních tříd a dále mohou v konstruktoru přidat své konkrétní parametry do seznamu parametrů, získaného ze třídy `ControlPanel`. Následně je v konstruktoru zavolána metoda `addItemsToControlPanel`. Každá třída také obsahuje metodu pro uložení dat z panelu do datových struktur aplikace. Data jsou nejprve zpracována v kontroleru `ControlPanelController` a následně předána kontroleru `EditFormDataController`.

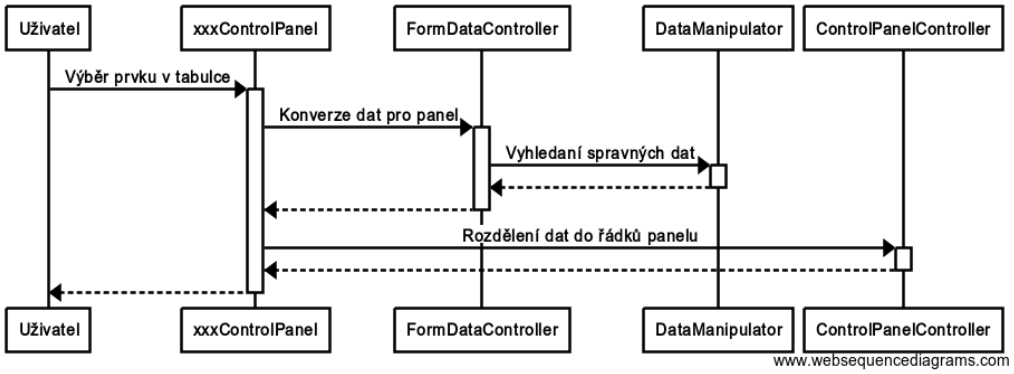
Grafická část editačního panelu je řízena převážně kontrolerem `ControlPanelController`. První funkcí kontroleru je přidání grafických komponent, obsažených v kontejneru `ControlPanelLine` do lyoutu editačního panelu. K tomuto účelu slouží metody `setParamBoxToControlPanel`, `setComboBoxItemToControlPanel`, `setRelationComboBoxItemToControlPanel`, `setDateItemToControlPanel`, `setTextItemToControlPanel`, `setNumberItemToControlPanel` a `setCheckComboBoxItemToControlPanel`. Pozice jednotlivých komponent v panelu je získána pomocí parametrů konkrétní metody.

V kontroleru jsou dále obsaženy metody pro zpracování dat z jednotlivých typů řádků: `processTextLines`, `processNumberLines`, `processComboBoxLines`, `processRelationComboBoxLines`, `processCheckComboBoxLines` a `processDateLines`. Metoda vždy projde veškeré řádky obsažené v panelu a v případě, že se jedná o řádek, jehož data mohou být přidána do datového modelu a souhlasí typ řádku se zpracovávaným typem řádků, jsou vráceny data z jednotlivých komponent spolu s možnými operátory (vhodnými pro datový typ) daných hodnot.

Další činností kontroleru je nastavení dat z datového modelu do grafických prvků pomocí metod `setValueCheckComboBox`, `setValueDatePicker`, `setValueComboBox`, `setValueRelationBox` a `setValueTextField`. Uvnitř každé metody je vytvořen nový řádek, kterému jsou následně předána konkrétní data. Data jsou v daném řádku dále zpracována a nastavena pomocí kontroleru `ControlPanelLineController` do konkrétních prvků řádku.

Poslední funkčností tohoto kotroleru je vytváření nových řádků spolu s logikou posunu statických prvků v panelu, konkrétně editačního tlačítka a volby možnosti existence elementu nebo segmentu. Do panelu lze přidávat různé

statické řádky jednoduchou registrací daného řádku v metodě `addItemsToControlPanel` konkrétního panelu. Registraci lze porovézt pomocí metod `setRadioButton`, `setStaticButton`, `setStaticClassBoxes`, `setStaticComboBox`. Posun řádků je řízen pomocí metody `shiftStaticObjects`. Přechod mezi jednotlivými třídami v případě výběru prvku datového modelu, jeho zpracování a zobrazení do panelu je ukázán na obrázku 5.8.



Obrázek 5.8: Sekvenční diagram volaných tříd při zobrazení editačního panelu

5.2.4 SQL a vyhledání vzorů

Pro komunikaci s datovým skladem nástroje SPADe je využíváno rozhraní JDBC. V aplikaci je zavedeno statické připojení na adresu databázového serveru. Adresa serveru se nachází ve třídě `Constans` a je tvořena konstantou `DATABASE_PATH`. Pro přihlášení do datového skladu, je ve třídě `LogInWindow` vytvořeno dialogové okno pro zadání přihlašovacích údajů. Tato třída také slouží jako přehledové oko pro výběr z konkrétních projektů obsažených v datovém skladu. Okno je tvořeno pomocí komponenty `TableView`, do které jsou vkládány instance třídy `VerifyTable` obsahující atributy *ID* a *name*. Jak následná práce s přihlašovacími údaji, tak následné získání dat o projektech je řízeno pomocí kontroleru `DatabaseController`. Metoda `logIn` nejprve zkontroluje validnost vstupů a pokusí se navázat spojení. Při zadání špatného vstupu nebo nenavázání spojení jsou vyvolány výjimky popisující nastalou událost.

Balík `database` obsahuje DAO třídy, které slouží k výběru konkrétních dat z databáze nástroje SPADe. Vznikly DAO třídy pro veškeré elementy a segmenty datového modelu SPADe. Pouze pro výčtové atributy elementů typu **Work Unit** (**Priority, Severity, Status, Type, Relation, Resolution**)

vznikla společná třída `WorkUnitElementDAO`. V konstruktoru každé třídy se přiřadí připojení k databázi, které se navazuje po přihlášení uživatele.

Všechny třídy obsahují metodu, ve které je automaticky vytvořen SQL dotaz v závislosti na datech obsažených v datovém modelu. Jednotlivé parametry jsou do metody předávány pomocí parametrů. Tyto parametry jsou zpracovány pomocí třídy `SQLAtributeCreator` a z výsledných výstupů je poskládána podmínka pro konkrétní dotaz. V druhé části metody je staticky popsáno sloučení potřebných tabulek databáze pro získání výsledku a popis je následně sloučen s dříve vytvořenými parametry. Metody pro získání dat z datového skladu se nachází v ukázce 5.1. Vzniklý dotaz je předán do metody `findInstanceInDB` společně s *ID* projektu, kde je následně zpracován.

V balíku `database` se také nachází třída `SQLVerifyObject`, která slouží jako kontejner pro uchování informací o existenci konkrétního segmentu nebo elementu ve vybraném projektu, konkrétního SQL dotazu, počet instancí v modelu a projektu.

```
ArrayList<List<Integer>> paramIds = new ArrayList<>();
    paramIds.add(classId);
    ArrayList<Integer> idList = new ArrayList<>();
    idList.add(projectVerifyId); paramIds.add(idList);
    String atributeSection = "";
    atributeSection += SQLAtributeCreator.createClassAttribute("
        role_classification", classId, projectVerifyId);
    atributeSection += SQLAtributeCreator.createStringAttribute("
        p.name", name, nameIndicators);
    String sql = "SELECT p.id FROM role p " + atributeSection;
```

Ukázka kódu 5.1: Ukázka metody pro vytroření SQL dotazu

Třída `SQLAtributeCreator` obsahuje statické metody pro vytvoření konkrétního atributu v závislosti na datovém typu. Do metody je předáno jméno atributu, seznam hodnot a seznam ukazatelů rovnosti. Na konci metody je vrácen textový řetězec obsahující podmínky pro konkrétní atribut. Druhou sadou metod nacházejících se v této třídě jsou metody pro získání identifikátorů, segmentů či elementů pro konkrétní instance. Pro vytvoření těchto atributů je v metodách znovu využito DAO objektů konkrétních segmentů a elementů. Metoda `findInstanceInDB` slouží k samotnému výběru dat pomocí třídy `PreparedStatement`. Z výsledku dotazu je naplněna nová instance třídy `SQLVerifyObject` dle toho, zda dopadl úspěšně či ne. Nová instance je přidána do seznamu, který je na konci metody vrácen.

Získané výsledky jsou zobrazeny pomocí třídy `VerifyWindow`, která vychází ze třídy `Stage` a obsahuje komponentu `TabelView`. Do té lze přidat instance třídy `VerifyTable`. Sloupec tabulky určený pro SQL dotaz využívá automatické úpravy velikosti sloupce a textu v něm obsaženém v závislosti na šířce okna pomocí metody `repaintText`. Pro možnost získání SQL příkazu má tabulka nastavený listener, který po dvojkliku do tabulky vyvolá informační okno, do kterého je daný dotaz přidán.

Veškerá logika vyhledávání vzorů v projektu je implementována ve třídě `VerifyController`. Metoda `verifyInstanceInProject` je volána z kontroleru `DatabaseController` po vybrání projektu z databáze. Umožňuje zobrazení okna s výsledky verifikace a naplnění přehledové tabulky daty získanými pomocí metody `fillVerifyTables`. V této metodě jsou volány jednotlivé metody pro vyzvednutí dat z datového skladu. Metody vracejí seznamy instancí třídy `VerifyTable`, které jsou následně sloučeny do jednoho a přidány do přehledové tabulky. Metody volané pro vyzvednutí dat jsou označeny `addXXXTOVerifyTable`, kdy místo XXX se nachází název konkrétního elementu nebo segmentu. Do metody je pomocí parametru předán seznam instancí obsažených v datovém modelu. V každé metodě se pomocí konstrukce `foreach` projde celý seznam a skrze DAO objekt jsou vyzvednuta konkrétní data. Tyto data jsou následně zpracována pomocí metody `createVerifyTable`, vracející instanci `VerifyTable`, která je následně přidána do seznamu získaných výsledků. V případě, že je prvek závislý na některém z elementů nebo segmentů (např. existuje vazba mezi **Iteration** a **Configuration**) je zavolána statická metoda z třídy `SQLAtributeCreator` pro získání *ID* řádků obsažených v databázi a shodujících se s objekty v datovém modelu nástroje. Tyto *ID* jsou dále předány do metody pro vyzvednutí dat konkrétního prvku. Metoda `createVerifyTable` nejprve porovná, zda se shoduje existence prvku v modelu s existencí prvku v datovém skladu, dále porovnává počet instancí v modelu a databázi. Dle výsledků porovnání následně vytvoří instanci `VerifyTable` buď s kladným nebo negativním výsledkem.

5.2.5 Modelovací plátno a manipulace s prvky

Modelovací plátno je tvořeno pomocí třídy `DragAndDropCanvas` převzaté z předchozí implementace nacházející se v balíku `graphics.canvas`. Logika plátna byla přesunuta do kontroleru `CanvasController` a operace nad prvky plátna jsou řízeny pomocí kontroleru `ManipulationController`. Kontroler `CanvasController` řídí přesun prvků po plátně a obsahuje metodu pro reakci na stisknutí klávesových zkratek. `ManipulationController` obsahuje

metody pro zkopírování prvku `copyItem`, vyjmutí prvku `cutItem`, odstranění prvku `deleteItem` a vložení prvku do plátna `pasteItem`. Metoda `createCopyForm` následně v případě operace kopírování zavolá potřebnou metodu z kontroleru `FormFillController` v závislosti na konkrétním typu prvku. Metoda `deleteForm` slouží k zavolání metod `DeleteFormController` pro smazání konkrétního prvku. Třída dále obsahuje globální proměnnou typu `NodeLink`, která napomáhá při vytváření relace mezi jednotlivými prvky.

Samotná logika vytváření relací mezi prvky se nachází ve třídě `LinkControl`. Metoda `ArrowManipulation` rozhoduje o tom, zda relace začíná či končí. Pokud relace začíná, je zavolána metoda `createLink` v té se zaregistruje počáteční prvek relace a jeho typ vytvoří instance `ElementsLink`, oddděné od třídy `NodeLink`. Instance je následně přidána do seznamů relací a je přidána grafická podoba do plátna s nastaveným počátečním bodem. V případě, že je relace dokončována, je ve třídě `segmentControl` zkontrolována relevance relace mezi danými prvky. Pokud je vytvářená relace nerelevantní, je vypsáno informační okno s textem o nemožnosti spojení. V opačném případě je zavolána metoda `finishLinkFromOperation`, ve které je zavolána konkrétní metoda z kontroleru `FormController` pro vytvoření relace v datových strukturách. Následně je zavolána metoda `finishLink` pro dokončení vytvářené relace. V té je aktuální instanci `ElementLink` nastaven konečný bod spojení, typ a je zobrazen na plátně. Metoda `deleteArrow` umožňuje odstranění spojnice z plátna a volá kontroler `DeleteFormData` pro odstranění relace z datových struktur. Pro překreslení spojnice mezi prvky v relaci slouží metody `repaintArrow`. Tato metoda projde seznamy spojnic a v závislosti na identifikátoru prvků překreslí počáteční nebo koncový bod spojnice.

6 Testování

Aplikace byla testována během celého vývoje pomocí jednotkových testů existujících ve výchozí implementaci, které byly dále rozšiřovány a měněny s přibývající funkcionalitou. Grafické uživatelské rozhraní bylo testováno za pomocí uživatelský testů. Komplexní funkčnost a uživatelská přívětivost aplikace byla otestována namodelováním množiny příznaků některých anti-patternů a základními prvky vybraného procesu vývoje softwaru.

6.1 Jednotkové testování

Pro vytvoření jednotkových testů bylo využito vývojové prostředí IntelliJ IDEA s knihovnou JUnit 4, která je do projektu připojena pomocí Maven dependency. Jednotkové testy vznikly pro implementace jednotlivých rozhraní datového modelu, tedy pro `EditDataModel`, `SavaDataModel` a `DeleteData-Model` a z větší části také pro třídy kontrolerů.

Testovací třídy jsou umístěny v balíku `test` a jsou rozděleny do dalších čtyř pod-balíků dle toho, která třída z datového modelu je testována. Pro otestování datového modelu a kontrolerů vzniklo tedy 470 testů. Datový model je jednotkovými testy pokryt z 87%, kontrolery aplikace jsou pokryté z 66%. Převážná část zbylé funkčnosti byla pokryta pomocí uživatelských testů. Předešlá implementace aplikace byla testována pomocí 95 testů. Z důvodu celkového přepracovaní architektury aplikace na MVC architekturu, nebylo možné tyto testy použít.

6.2 Uživatelské testování

Pro testování uživatelského rozhraní byl zvolen přístup pomocí uživatelských testů. Byla vybrána skupina čtyř testerů. Všichni členové testovací skupiny pracují jako SW projektanti, a proto lze považovat jejich schopnosti využívání PC na dostatečně vysoké úrovni. Tato skupina byla zvolena z důvodu, že aplikace bude v konečné podobě určena pro uživatele s vyšším stupněm využívání PC a znalostmi v oblasti softwarového inženýrství. Testeři obdrželi testovací scénáře, pomocí kterých procházeli aplikaci a hledali chyby.

Pro řízení uživatelských testů byl zvolen on-line nástroj pro správu testů TestLodge. Pomocí tohoto nástroje vznikly čtyři testovací sady. První sada testů obsahovala scénáře pro vytvoření a editaci jednotlivých elementů či segmentů v aplikaci a následnou kontrolu zobrazení dat v přehledových panelech. Druhá sada obsahovala scénáře pro spojování a odstranění elementů a segmentů a ve třetí sadě se nacházely scénáře pro uložení modelu do XML a zpětné nahrání do aplikace. Ve čtvrté sadě se nacházejí scénáře pro kopírování a následné vkládání jak tabulkových prvků, tak prvků na modelovacím plátně. SQL funkčnost byla testována pomocí vytvořených modelů, popsaných níže v této kapitole. Celkem tedy vzniklo 90 testovacích případů. Vzniklé testovací sady se nacházejí na CD nosiči přiloženém k této práce.

Nástroj dále umožňuje snadný přehled o testech testery již vykonaných pomocí grafů a statistik. Výsledky testů testeři zapisovali přímo do jednotlivých případů spolu s případnými přílohami. Testeři objevili několik funkčních a grafických chyb, které byly následně opraveny a zapracovány do implementace. Po těchto úpravách byla aplikace znovu otestována a opět byly následné připomínky zapracovány do implementace. Testeři měli také několik připomínek k funkčnosti a grafické podobě aplikace. Návrhy na funkční změny jsou popsány v kapitole 7

6.3 Modely příznaků anti-patternů

Pro otestování uživatelské přívětivosti, generování modelu do SQL a následné vyhledání modelu v datovém skladu, vzniklo několik ukázkových anti-patternů nebo alespoň jejich příznaků. Vznikl také částečný model procesu ASWI. Detail modelu byl omezen možnostmi současné implementace aplikace. Vzniklé modely se nacházejí jako přílohy této práce ve formátu XML.

S ohledem na aktuální možnosti aplikace byly modelovány pouze některé příznaky vývoje softwaru potenciálně vedoucí ke vzniku různých anti-patternů nebo indikujícíc jejich přítomnost. Vytvořené modely mohou sloužit jako dílčí symptomy těchto anti-patternů. Detekce těchto symptomů v datovém modelu SPADe využívá stejných postupů a pravidel, které jsou potřebné pro identifikaci komplexních anti-patternů. Zvolené modely vycházejí z definice anti-patternů, popsané v podkapitole 2.4. Detekce těchto příznaků je žádoucí, stejně jako detekce komplextních anti-patternů a umožňuje nalezení chyb, nebo slabých míst v procesu vývoje softwaru. Následující výčet popisuje namodelované symptomy vycházející z praxe společně s popisem jejich

vlastností a funkcí, které jsou pomocí nich testovány. Tyto symptomy byly zvoleny pro jejich schopnost otestovat širší množinu vlastností aplikace. V závorce za každým názvem příznaku se nachází jeho identifikátor, používaný ve výsledkových tabulkách.

Absent Artifact (SNT1)

Tento symptom se vyznačuje neexistencí důležitých dokumentů v projektu. Odhalení tohoto příznaku je vyšší pravděpodobnost kvalitnější implementace. Je důležité, aby tyto dokumenty existovaly před začátkem implementační fáze projektu, případně po každém ukončení iterace pro její vyhodnocení. Příznak lze v aplikaci namodelovat vytvořením následujících artefaktů (Vize, Architektura, Dokument specifikace, Retrospektiva) s volbou neexistence v projektu. Modelace tohoto anti-patternu kontroluje schopnost aplikace vyhledávat patterny, potažmo anti-patterny na základě části textu a dále testuje funkčnost existence a neexistence prvku v modelu.

No Knowledge Base (SNT2)

V projektu se nachází minimální množství artefaktů pro uchování znalostí typu `WIKIPAGE` (tzn. wiki stránek). Tato skutečnost může mít za následky fakt, že při budoucích pracích na projektu nebude k dispozici dostatek informací. Pomocí modelu je ukázána schopnost vytvořit neexistující prvek společně počtem jeho instancí v projektu. Následně bylo pomocí modelu zkoumáno vyhodnocení dat z databáze při obrácené logice existence modelu.

The Flash (SNT3)

Příznak The Flash se objevuje v situacích, kdy jsou některé aktivity ukončeny v neadekvátním časném rozsahu. V praxi tato skutečnost může znamenat nedostatečnou implementaci příslušné činnosti nebo nesprváné používání projektového nástroje. Tato skutečnost může vést ke zkreslování statistik o projektu, případně k problém s plánováním. Lze ho namodelovat pomocí existence uzavřeného **Work Unit**, na jehož dokončení nebyl potřeba žádný čas. Model testuje, stejně jako následující příznak **Unfinished Bussiness** možnost pracovat s **Work Unit** a jeho výčtovými atributy.

Unfinished Business (SNT4)

Tento příznak se soustředí na jednotlivé aktivity v projektu a lze za něho považovat situaci, kdy je aktivita ukončena, ale není ze 100% dodělána. Tento stav může znamenat nedokončené činnosti v projektu, ale také může ukazovat

na nesprávné používání projektových nástrojů. Tato situace vede stejně jako v případně The Flash ke zkreslování statistik. Modelace tohoto příznaku testuje možnost práce s elementy **Work Unit**.

What did I do? (SNT5)

Příznak se zaměřuje na nedostatečně popsané jednotlivé činnosti a události v průběhu vývoje softwaru. Nedostatečný popis může vést ke špatnému pochopení nově implementovaných částí, případně obtížnému budoucímu navázání prací na projektu. Pomocí aplikace lze příznak namodelovat následujícím postupem. V projektu existuje určitý počet elementů (**Work Unit, Configuration, Commit, Committed Configuration**), většinou přesahující počet 10, ve kterých se nachází nevyplněná informace o parametru *description*. U elementu **Work Unit** *description* znamená popis konkrétní činnosti. V případě **Configuration** a **CommittedConfiguration** se jedná o popis daného elementu. U **Commitu** tento atribut obsahuje commit massage.

6.4 Model procesu ASWI

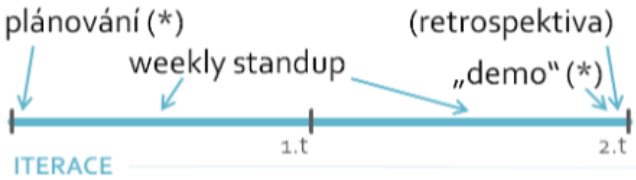
Pro otestování uživatelské použitelnosti aplikace a její celkové funkčnosti vznikl model procesu vývoje softwaru využívaný na Západočeské univerzitě v Plzni (ZČU), Fakultě aplikovaných věd (FAV), Katedře informatiky a výpočetní techniky (KIV) pro potřeby předmětu Pokročilé softwarové inženýrství (KIV/ASWI), více popsaný v následujícím textu.

„Jedná se o iterativní, agilně orientovaný proces pro řízení tvorby malých až středně velkých softwarových projektů, založený na metodice Scrum a RUP.“[6]

Proces je tvořen jednotlivými iteracemi, které vycházejí z již zmíněné metodiky Scrum. Projekt je strukturován do fází ukončených konkrétními milníky, jejichž cílem je eliminovat nejrizikovější oblasti projektu v konkrétním čase a ověřit správnost jeho průběhu. Iterace jsou časově omezené se standardní délkou 2 týdnů. Iterace prochází následujícími pěti body, viz obrázek 6.1:

- naplánování iterace,
- práce na iteraci + týdenní schůzky,
- předání výsledků,
- retrospektiva,

- schůzka s mentorem.



Obrázek 6.1: Průběh ASWI iterace [6]

Projekt je tvořen pomocí čtyř milníků, převzatých z metodiky RUP a [5] a lehce upravených pro výukové účely. Konkrétně se jedná o:

- **„Project Initiation" (PRI)** – ukončuje fázi zahájení projektu a produkuje artefakt Vize,
- **„Lifecycle Objectives and Architecture" (LCOA)** – ukončuje fázi určení architektury,
- **„Initial Operational Capability" (IOC)** – ukončuje hlavní realizační práce,
- **„Product Release" (REL)** – ukončuje celý semestrální projekt.

Návaznost jednotlivých milníků je ukázána na obrázku 6.2. Proces ASWI je tvořen oproti Scrumu poměrně značným množstvím rolí a artefaktů. Stále ale platí, že rozdělení rolí je plně v pravomoci týmu.

Obrázek 6.2: Makro proces ASWI projektů [6]

Jako testovací množina dat bylo zvoleno pět studentských projektů z ročníku 2016/2017 uložených v projektovém nástroji Redmine a následně zpracovány pomocí nástroje SPADe a uloženy do jeho datového skladu. Data posloužila jednak k otestování samotné logiky dynamického vytváření SQL dotazů, tak i ke kontrole správnosti datového modelu a správného vyhodnocení vrácených dat.

Model zachycuje základní aspekty procesu vývoje ASWI, jako je pět instancí segmentu Iteration, čtyři typu Phase, důležité týmové role (Admin projektu, Vývojář, Mentor a Zákazník). Jsou zachyceny také hlavní artefakty a to Vize a Architektura. Dále artefakty představující výstupy z jednotlivých schůzek, dokumenty retrospektivy. Model také obsahuje určený počet **Commit**, který se musí v projektu nacházet a několik aktivity.

6.5 Výsledky testování pomocí modelů

Jak modely jednotlivých bad practices vývoje softwaru, tak i vytvořený model ASWI procesu vývoje softwaru odhalily některé nedostatky jak samotné aplikace, tak i možná budoucí rozšíření, která by mohla sloužit k ještě úspěšnějšímu vyhledání modelu v datovém skladu.

Výsledky vyhledání jednotlivých namodelovaných bad practices pomocí aplikace se nacházejí v tabulce 6.1 a ukazují, které symptomy byly odhaleny v konkrétních projektech. Projekty jsou označeny pomocí identifikátorů obsažených v datovém skladu nástroje SPADe. Pokud se konkrétní symptom v projektu nevyskytl, je na příslušném místě uvedeno „NF“ (Not Found), v opačném případě „F“ (Found). Výsledky vyhledání příznaků anti-patternů byly dále ručně porovnávány přímo s daty uloženými v datovém skladu, respektive v nástroji Redmine. Konkrétní výstup z ručního vyhledávání v datovém skladu a Redmine se nachází v tabulce 6.2.

Tabulka 6.1: Výsledky vyhledání příznaků pomocí aplikace

Projekt	SNT1	SNT2	SNT3	SNT4	SNT5
4	NF	NF	NF	NF	NF
5	NF	NF	NF	NF	NF
6	NF	F	NF	NF	NF
7	F	NF	NF	NF	NF
8	F	NF	NF	NF	NF

Výsledky vyhledání příznaku aplikací se ve většině případů shodují s daty obsaženými v datovém skladu a daty v nástroji Redmine. Aplikací byly detekovány bad practicies: Absent Artifact v projektech 7 a 8, dále pak No Knowlege Base v projektu s identifikátorem 8. V případě nalezení příznaku v SNT1 v projektu 7, se v projektu nacházeli všechny dokumenty potřebné pro potlačení daného příznaku. Aplikace, ale existenci těchto dokumentů vyhodnotila jako falešnou z důvodu existence nekorespondujícího počtu

Tabulka 6.2: Výsledky vyhledání příznaků v datovém skladu a Redmine

Projekt	SNT1	SNT2	SNT3	SNT4	SNT5
4	NF	NF	NF	NF	NF
5	NF	NF	NF	NF	NF
6	NF	NF	NF	NF	NF
7	NF	NF	NF	NF	NF
8	F	NF	NF	NF	NF

instancí prvku v modelu a dat v databázi. V případě **SNT2** v projektu 8 se v databázi nacházejí položky tabulky **Artifact**, které ale mají v poli pro typ artefaktu hodnotu `NULL`, proto byly aplikací vyhodnoceny jako neodpovídající. V případě Redmine bylo nalezeno více než 3 stránky typu `WIKIPAGE`. Pro nalezení příznaku musely být nalezeny (nebo nenalezeny) všechny prvky daného modelu. Z tohoto pohledu aplikace zaregistrovala existenci, například při vyhledání příznaku **SNT1** v projektu 1, pouze dvou prvků modelu. Při detailnějším prohledání se výsledek shodoval s daty v databázi, ale v Redmine se nacházeli všechny potřebné dokumenty (data).

Vyhledání modelu ASWI v konkrétních studentských projektech odhalilo několik syntaktických chyb při skládání SQL dotazů i v samotném vyhodnocení vráceného dotazu. Model dále poukázal na vlastnost aplikace při práci s kalendářními daty. Některé tabulky datového skladu využívají různé formy datových typů pro uchování data. Například entita **Work Item** využívá pro pole *created* datový typ s časovou značkou. Tato značka znemožňuje vyhledání konkretního data z důvodu nemožnosti aplikace zadávat datum spolu s časovou značkou. Je tedy nutné pro získání konkrétního data uzavřít požadovaný den do intervalu, který ho ohraničí. Druhým zjištěným nedostatkem je absence možnosti vytvoření složené podmínky pro konkrétní parametr elementu či segmentu.

Tabulka 6.3: Výsledky detekce modelu procesu ASWI pomocí aplikace

Projekt	Nalezeno	Nenalezno	Poměr (%)
4	22	37	37%
5	24	35	41%
6	24	35	41%
7	17	42	29%
8	18	41	31%

Vytvořený model byl v průměru identifikován v datech z 36%. Přesné poměry nalezených a nenalezených částí vytvořeného modelu se nachází v tabulce 6.3. V některých případech nenalezení prvků datového modelu v datovém skladu bylo zapříčiněno neexistencí těchto hodnot v databázi, například u elementu **Milestone**. Dalším z důvodů nenalezení dat je možnost velké variability textových řetězců, obsažených v parametrech jednotlivých elementů a segmentů, uložených v datovém skladu. Hlavním problémem při vyhledávání se ukázaly parametry určitých kalendářních dat. I přes možnost zadání intervalu, v jakém se mají například jednotlivé **Iteration** nebo **Phase** vyskytovat, není tato možnost dostatečně silná pro pokrytí všech možností vytvoření a ukončení těchto segmentů.

7 Možná rozšíření aplikace

Aplikace by měla být dále rozšiřována pro možnost modelace plnohodnotných anti-patternů a patternů a jejich následné vyhledání v datovém skladu nástroje SPADe. Následující výčet popisuje možná funkční rozšíření aplikace:

- odkazování na atributy existujících prvků,
- složitější logika pro kritéria (attributy),
- základní aritmetika v kritériích,
- přítomnost úkolu/konfigurace/artefaktu v každé iteraci/fázi,
- kritérium na délku textu,
- doplnění zbývajících entit z datového modelu nástroje SPADe
- optimalizace SQL dotazů.

Možná rozšíření od testerů pokrývají jak funkční stránku aplikace, tak i její grafickou podobu a uživatelskou přívětivost. Zde jsou uvedené některé připomínky od testerů:

- rozšíření formátů pro export datového modelu,
- umožnit zoom modelovacího plátna na místo současného posunu pomocí scrollbaru,
- umožnit v aplikaci využívat více jazyků,
- umožnit výběr barevné kombinace aplikace,
- zlepšit možnost odstranění samotné spojnice z plátna.

8 Závěr

Cílem této práce bylo zlepšit uživatelskou přívětivost, opravit a rozšířit stávající funkčnost grafického editoru modelů procesu vývoje softwaru, který je rozšířením nástroje SPADe.

Pro splnění tohoto cíle bylo potřeba analyzovat některé metologie a postupy vývoje softwaru spolu s možnostmi zachycení vzorů, vycházejících z daných metodologií (kapitola 2). Dále bylo nutné seznámit se s možnými chybami v procesu vývoje softwaru (anti-patterny; kapitola 2.4.1). Pro pochopení kontextu práce bylo potřeba seznámit se s možnostmi nástroje SPADe (kapitola 3) a provézt analýzu samotné implementace nástroje pro grafické modelování procesů za účelem identifikace funkčních nedostatků a možných rozšíření kapitola 4.

V rámci práce byla implementována funkcionalita popsaná v kapitole 4.4. Pro zlepšení uživatelské přívětivosti byl přepracován systém vyskakujících oken s editačními formuláři do `Drawer` panelů, umístěných po stranách hlavního modelovacího plátna. Dále vznikly ikony pro tlačítka umístěná v horním panelu aplikace. Aplikace byla přepracována do třívrstvé architektury MVC, konkrétně vzoru MVP. Značné množství nalezených grafických a funkčních chyb bylo odstraněno přepracováním grafické stránky nebo úpravou architektury aplikace. Hlavním rozšířením aplikace bylo napojení na datový sklad nástroje SPADe a vyhledání jednotlivých modelů v datech v něm obsažených. Pro naplnění tohoto požadavku, bylo využito rozhraní JDBC a vznikl systém dynamického vytváření SQL dotazů na základě dat uložených v datovém modelu aplikace. Tyto dotazy jsou následně používány pro ověření existence či neexistence modelovaného procesu nebo jeho části v datech projektů uložených v datovém skladu.

Aplikace byla testována jednotkovými testy pomocí knihovny JUnit 4 a GUI bylo testováno za pomoci uživatelských testů řízených pomocí on-line nástroje TestLodge. Jednotlivé testovací scénáře byly předány 4 nezávislým testerům a nalezené nedostatky byly zapracovány do implementace. Pro ověření možnosti vytvoření anti-patternů vzniklo několik ukázkových modelů popsaných v kapitole 6.

Hlavním cílem práce bylo rozšíření a opravení stávající implementace grafického nástroje pro vytváření modelů procesu vývoje softwaru spolu s analýzou existujících metodik a anti-patternů vývoje softwaru. Tento cíl práce byl splněn ve všech bodech.

Literatura

[1] Alexander, C. *A pattern language: towns, buildings, construction.* Oxford University Press, 1977.

[2] Ambler, S. W. *Process patterns: building large-scale systems using object technology.* Cambridge University Press, 1998.

[3] Ambler, S. W. – Nalbone, J. – Vizdos, M. *The Enterprise Unified Process: extending the Rational Unified Process.* Prentice Hall Press, 2005.

[4] Bernard, B. *Prezentační vzory z rodiny MVC* [online]. 2009. Dostupné z: `https://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/`.

[5] Boehm, B. *Anchoring the software process.* IEEE Software, 1996.

[6] Brada, P. *Agilní proces pro výuku softwarovéhoinženýrství v předmětu ASWI* [online]. 2011. Dostupné z: `http://www.kiv.zcu.cz/~brada/files/aswi/aswi-proces/cz.zcu.kiv.aswi_plug-in_cz/guidances/whitepapers/resources/aswi-proces-2011-spec.pdf`.

[7] Brown, W. H. et al. *AntiPatterns: refactoring software, architectures, and projects in crisis.* John Wiley & Sons, Inc., 1998.

[8] Chris, R. *POJOs in action : developing enterprise applications with lightweight frameworks.* Manning Publications, 2006.

[9] Coder, G. *How to make Navigation Drawer (Side Panel) in JavaFX – JavaFX Drawer* [online]. 2017. Dostupné z: `https://www.genuinecoder.com/javafx-navigation-drawer-side-panel-html/`.

[10] Corporation, F. *What is Application Lifecycle Management?* [online]. 2017. Dostupné z: `https://www.inflectra.com/spirateam/highlights/understanding-alm-tools.aspx`.

[11] Corporation, P. *MVVM Tutorial* [online]. 2019. Dostupné z: `https://www.zkoss.org/zkdemo/getting_started/mvvm`.

[12] Cunningham, W. *Anti Patterns Catalog* [online]. 2013. Dostupné z: `http://wiki.c2.com/?AntiPatternsCatalog`.

[13] ELGABRY, O. *Software Engineering - Software Process and Software Process Models (Part 2)* [online]. 2017. Dostupné z: `https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d0621`

[14] G. Grambow, R. O. – Reichert, M. Process patterns: building large-scale systems using object technology. *Process patterns*. Nov. 2012.

[15] Giles, J. *ControlsFX* [online]. 2017. Dostupné z: `http://fxexperience.com/controlsfx/`.

[16] Gluonhq. *Charm Glisten* [online]. 2018. Dostupné z: `https://docs.gluonhq.com/charm/5.0.1/#_charm_glisten`.

[17] Herout, P. *Java a XML*. KOPP, 2012. ISBN 978-80-7232-307-4.

[18] Hibernate. *Object/Relational Mapping* [online]. 2019. Dostupné z: `https://hibernate.org/orm/`.

[19] Interactive, N. R. *Agile Scrum for Web Development* [online]. 2019. Dostupné z: `https://www.neonrain.com/agile-scrum-web-development/`.

[20] JANOCH, V. Nástroj pro grafický popis procesů vývoje software, 2017. Dostupné z: `DostupnézWWW<https://theses.cz/id/g44zj1/>`.

[21] JavaTpoint. *Hibernate Query Language (HQL)* [online]. 2018. Dostupné z: `https://www.javatpoint.com/hql`.

[22] JFoeniX. *Drawer* [online]. 2017. Dostupné z: `http://www.jfoenix.com/documentation.html#Drawer`.

[23] Karagkasidis, A. Developing GUI Applications: Architectural Patterns Revisited. *Developing*. 2008.

[24] Kosek, J. *XML schémata* [online]. 2014. Dostupné z: `http://www.kosek.cz/xml/schema/wxs.html#wxs-atributy`.

[25] Kroll, P. – Kruchten, P. *The Rational Unified Process made easy: A practitioner's guide to the RUP*. Addison-Wesley Professional, 2003.

[26] Laplante, P. A. – Neill, C. J. *Antipatterns: identification, refactoring, and management*. Auerbach Publications, 2005.

[27] L.García-Borgoñónab, J.-G. M. M. M. Software process modeling languages: A systematic literature review. *Software process modeling*. 2013.

[28] Lipka, R. *Návrhové vzory pro GUI* [online]. 2018. Dostupné z: `https://courseware.zcu.cz/portal/studium/courseware/kiv/uur/prednasky.html`.

[29] mysqlConnector. *MySQL Connector/J 8.0 Developer Guide* [online]. 2019. Dostupné z: `https://dev.mysql.com/doc/connector-j/8.0/en/`.

[30] Object Management Group. Software & Systems Process Engineering Meta-Model Specification.

[31] OSLC. *OSLC Change Management Version 3.0. Part 1: Specification* [online]. 2018. Dostupné z: http://docs.oasis-open.org/oslc-domains/cm/v3.0/cm-v3.0-part1-change-mgt.htm.

[32] Petr Pícha, P. B. ALM Tool Data Usage in Software Process Metamodeling. *ALM Tool*. 2016.

[33] Petr Pícha, P. B. Towards Architect's Activity Detection Through a Common Model for Project Pattern Analysis. *Towards Architect's*. 2016.

[34] Pishchukhin, D. *Using JPA in an OSGi Environment* [online]. 2011. Dostupné z: https://jaxenter.com/tutorial-using-jpa-in-an-osgi-environment-103494.html#authors-block.

[35] Pícha, P. Popis softwarových procesů použitelný pro nástroje řízení projektů. Master's thesis, ZCU, 2013.

[36] Pícha, P. Datový model nástroje SPADe a jeho mapování na projektová data z ALM nástrojů. *Datový model nástroje SPADe*. 2016.

[37] SourceMaking. *AntiPatterns* [online]. 2019. Dostupné z: https://sourcemaking.com/antipatterns/.

[38] Sutherland, J. V. – Schwaber, K. The SCRUM methodology. In *Business object design and implementation: OOPSLA workshop*, Londen, UK, 1995. Springer-Verlag London.

[39] Team, L. C. *What the Waterfall Project Management Methodology Can (and Can't) Do for You* [online]. 2017. Dostupné z: https://www.lucidchart.com/blog/waterfall-project-management-methodology.

Seznam použitých zkratek a výrazů

- AD – Architecture Defect,
- ALM – Aplication Lifecycle Management,
- API – Application Programming Interface,
- ASWI – Pokročilé softwarové inženýrství,
- DAO – Data Access Object,
- DBMS – Database Management System,
- DD – Decomposition Defect,
- DMD – Data Model Defect,
- ETL – Extract-Transform-Load,
- EUP – Enterprise Unifed Process,
- FAV – Fakulta aplikovaných věd,
- GUI – Graphic User Interface,
- FD – Functional Defect,
- HQL – Hibernate Query Language,
- IOC – Initial Operational Capability,
- JDBC – Java Database Connectivity,
- KIV – Katedra informatiky a výpočetní techniky,
- LCOA – Lifecycle Objectives and Architecture,
- MVC – Model-View-Controler,
- MVP – Model-View-Presenter,
- MVVM – Model-View-ViewModel,

- ORM – Object-Relation Mapping,
- OSLC – Open Services for Lifecycle Collaboration,
- XJC – XML to Java Compiler,
- XML -– eXtensible Markup Language,
- XSD – XML Schema Definition,
- PAC – Presentation-Abstraction-Control,
- POJO – Plain Old Java Object,
- POM – Project Object Model,
- PRI – Project Initiation,
- REL – Product Release,
- REST – Representational State Transfer,
- RUP – Rational Unifed Process,
- SEWL – Software Workflow Language,
- SNT – Sympton of Anti-pattern
- SPADe – Software Process Anti-pattern Detector,
- SPEM – System Process Engineering Meta-Model,
- UFD – User Friendly Defect,
- UML – Unified Modeling Language,
- UP – Unifed Process,
- VD – Visual Defect,
- VCS – Version Control System,
- ZČU -– Západočeská Univerzita,

Seznam obrázků

Seznam tabulek

Přílohy

A Obsah CD

Zde je rozepsán obsah CD nosiče přiloženého k této diplomové práci.

- Složka **src** obsahuje zdrojové kódy aplikace rozdělené do balíků.

- Složka **text** obsahuje tento dokument ve formátu pdf spolu s jeho zdrojovými LaTeXsoubory ve složce **Latex**.

- Složka **dokumentace** obsahuje JavaDoc ke zdrojovým kódům

- Složka **aplikace** obsahuje spustitelný .jar soubor s aplikací, XSD schéma popisu datového modelu a příkladový proces vývoje softwaru a jednotlivé příznaky anti-patternů v XML souborech.

- Soubor **README.txt** obsahuje tento rozpis obsahu CD nosiče

B Uživatelský manuál

Uživatelský manuál obsahuje informace o instalaci aplikace, požadavcích pro její spuštění, její funkcionalitu, popis ovládání a chybových hlášení.

B.1 Instalace aplikace

Aplikaci nelze spustit bez nainstalovaného běhového prostředí Java JRE nejméně verze 8. Se staršími verzemi nebude aplikace vůbec fungovat. Nejnovější verzi prostředí lze stáhnout ze stránek společnosti Oracle.

Aplikace se nijak neinstaluje, jednoduše se spustí po kliknutí na JAR soubor přiložený k tomuto dokumentu, nebo je k získání v několika verzích na stránkách verzovacího nástroje GitHub, konkrétně na adrese[1].

B.2 Rozložení aplikace

B.2.1 Hlavní obrazovka

Hlavní obrazovka je také úvodní obrazovkou aplikace, viz obrázek B.1. Ta umožňuje tři typy funkčnosti. V horní části se nachází kontextové menu, popsané v kapitole B.2.3. Prostřední část, označená modrou barvou, představuje pás tlačítek. Tento pás obsahuje tlačítka, která slouží k vytvoření pohyblivého prvku na plátně. To lze provést klasickým stiskem tlačítka, které je zobrazeno pomocí piktogramu daného prvku společně s popisem. Nachází se zde také tlačítko Project, které slouží k vyvolání editačního panelu popsaného v kapitole B.3. Pod těmi tlačítky je umístěna komponenta pro výběr konkrétního elementu nebo segmentu a vyvolání jeho přehledové tabulky viz kapitola B.3.1. Plátno je umístěno do spodní části obrazovky, označené bílou barvou. Plátno slouží k rozmístění vytvořených elementů. Objekty na plátně lze libovolně přemisťovat a skládat.

B.2.2 Prvek plátna

Prvek obsahuje v horní části identifikaci s elementu, který představuje a ve spodní části se nachází alias vytvořený uživatelem. Velikost spodního

[1] github.com/VenaJanoch/BakalarkaSPADEUI/releases

obdélníku je automaticky měněna na základě délky textu zadaného uživatelem. Vyvolání editačního panelu proběhne po dvojkliku levým tlačítkem myši. Pravím tlačítkem je vyvoláno kontextové menu, umožňující kopírování daného objektu, jeho odstranění a vyjmutí z plátna. Identifikaci jednotlivých objektů lze zjistit i po najetí na objekt. Ukázka elementu Person na obrázku B.3. Jednotlivé prvky lze kopírovat a vkládat jak jednotlivě, tak po celých celcích. Pro výběr prvků jsou k dispozici klasické způsoby výběru (Výběr klikem s klávesou shift nebo výběr tahem).

B.2.3 Menu

Menu je složené z kořenové položky, obsahující další podpoložky. Hlavní položka má označení File, zde se nachází možnost New pro vytvoření nového projektu, Open... pro načtení již vytvořeného procesu uloženého v XML, Save uložení procesu, Save as... uložení procesu s výběrem názvu souboru. Validate umožní ruční kontrolu procesu. A Close umožňující ukončení celé aplikace, viz obrázek B.2.

B.3 Editační panel

Postranní panel umožňující zadávání konkrétních informací o jednotlivých segmentech či elementech procesu vývoje softwaru je umístěn do pravé části obrazovky. Panel je zobrazen pouze tehdy, je-li potřeba zadat informace o prvku do vstupních polí nebo vybrat již z předdefinovaného prvku z nabídky. Panel se zobrazuje po stisku konkrétního řádku v přehledové tabulce, či dvojklikem na konkrétní prvek umístěný na plátně. Panel vždy obsahuje políčko pro zadání aliasu, pod kterým bude v aplikaci vystupovat a volbu o existenci či neexistenci v rámci modelu. Ostatní políčka jsou závislá na konkrétním typu elementu či segmentu a lze dynamicky přidávat potřebné parametry, při nutnosti odstranit některou z položek, stačí jednoduché odznačení řádku na jeho začátku. Data jsou uložena po stisknutí tlačítka Edit. V případě že se jedná o editační panel pro výčtové typy Priority, Severity, Relation, Resolution, Status a Type pro Work Unit. Nachází se v panelu také výběrový seznamem typu Class patřícího pod daný element. Po vybrání položky ze seznamu se automaticky zvolí hodnota v seznamu představujícím Super Class. Ukázka formuláře Priority na obrázku B.4.

B.3.1 Panel s přehledovou tabulkou

Panel s přehledovou tabulkou existujících instancí elementu či segmentu se nachází v levé části obrazovky a může být vyvolán v případě výběru prvku z komponenty umístěné v horním panelu nebo vytažením pomocí myši. Pod přehledovou tabulkou se nacházejí tři tlačítka. Tlačítko označené znakem + slouží k vytvoření nové instance prvku, který je následně zobrazen v tabulce a automaticky vyjede editační panel z pravé strany. Tlačítko označené znakem - odebere prvek z tabulky a celé aplikace. Pomocí tlačítka označeného D, lze jednotlivé prvky tabulky kopírovat. Tlačítko Edit slouží k vyvolání editačního panelu pro zvolený řádek tabulky, panel lze také vyvolat dvojklikem na daný řádek. Odstranění prvku z tabulky lze také pomocí klávesy Delete. Po stisku Delete se zobrazí okno s výzvou, zda uživatel chce opravdu data z tabulky smazat a jaké další prvky jsou na nich závislé, konkrétní podoba výzvy je popsána v kapitole B.5.

B.4 Propojování objektů

Propojování objektů slouží k vytvoření vazby mezi konkrétními objekty. Aplikace umožňuje vytvářet vazbu mezi elementy Person-Artifact, Person-Configuration, Artifact-Configuration, Commit-Committed Configuration, Committed Configuration-Configuration.

Vazbu mezi objekty lze vytvořit v režimu kreslení čar, do kterého se přepneme stiskem tlačítka označeného šipkou a umístěného nad kreslícím plátnem. Režim kreslení poznáme dle změny kursoru ze šipky na křížek nebo barevnou změnou tlačítka označeného šipkou. Kliknutím na některý prvek umístěný na plátně, nejprve vybereme počáteční prvek, ze kterého se vykreslí čára a až po kliknutí na druhý prvek, je čára vykreslena. Režim kreslení ukončíme opětovným stisknutím tlačítka, označeného šipkou nebo klávesou ESC. Již nakreslenou čáru lze odstranit dvojitým klikem na ni. Ukázka režimu kreslení na obrázku B.6.

B.5 Klávesové zkratky

Aplikace reaguje na různé kombinace stisku kláves klávesnice.

- Klávesa ESC -> vypnutí režimu kreslení, zavření editačních formulářů,
- Delete -> smazání prvku z plátna či tabulky.

- Kombinací Ctrl + n -> vytvoření nového projektu,
- Ctrl + o -> open,
- Ctrl + s -> save,
- Alt + f4 -> ukončení aplikace,
- Ctrl + c -> kopírování prvku plátna,
- Ctrl + v -> vložení prvku do plátna,
- Ctrl + x -> vyjmutí prvku z plátna,
- Crtl + f -> pro validaci procesu.

B.6 Aplikační hlášení

Hlášení jsou zobrazována na principu vyskakujícího okna. Toto okno ve většině hlášení, kromě informování o počtu mazaných položek z tabulky, je rozděleno do dvou částí. V horní části hlášení je popsána nastalá chyba. Spodní část udává jednoduchý návod, jak chybu napravit. S oknem, které hlášení vyvolalo, se nedá pracovat, dokud není zmáčknuto potvrzovací tlačítko.

B.6.1 Tabulková hlášení

Do této skupiny spadají hlášení, týkající se mazání prvků z přehledových tabulek segmentů.

První hlášení je uživateli zobrazeno, pokud není v tabulce s výčtem elementů vybrána žádná položka pro smazání, viz obrázek B.7.

Druhý formát hlášení je zobrazen, pokud je uživatelem zvolena alespoň jedna položka z tabulky a stisknuto tlačítko nebo klávesa Delete. Informační okno obsahuje kontrolní seznam označených položek ke smazání a postupně vypsány elementy či segmenty, které jsou závislé na mazané instanci. Pokud uživatel souhlasí se smazáním, stiskne tlačítko OK, v opačném případě Cancel, a označené položky zůstanou v dané tabulce, viz obrázek B.8.

B.6.2 Zavření Aplikace

Při jakémkoliv způsobu vypnutí aplikace je uživatel informován o ukončení programu a možností uložení stávajícího stavu projektu. Projekt lze uložit stiskem tlačítka Save umístěného do levé spodní části okna, po uložení je aplikace vypnuta. Zavření aplikace lze odvolat tlačítkem Cancel, případně zavřít bez ukládání tlačítkem OK, ukázka hlášení na obrázku B.13.

B.7 Validace projektu

Validaci projektu z pohledu XML může každý uživatel provést dle vlastní potřeby volbou umístěnou v menu hlavní obrazovky File položkou Validate. Na výskyt chyby v obsahu procesu vývoje software je uživatel upozorněn hlášením s informací o problému, viz obrázek B.15. Při úspěšné validaci hlášení obsahuje informaci o úspěchu validace, viz obrázek B.14.

Automatická validace probíhá při každém ukládání vytvořeného procesu nebo jeho načtení ze souboru do aplikace. Pokud validace proběhne úspěšně, soubor je vytvořen pod zadaným jménem, případně jsou data načtena z již vytvořeného souboru bez dalších informací pro uživatele. Pří výskytu chyby umožňuje chybové hlášení obsahující popis chyby danou akci vykonat i s vyskytlou chybou. Pro tuto akci slouží tlačítko Accept, pro odvolání tlačítko Cancel, viz obrázek B.16.

B.8 Práce se soubory

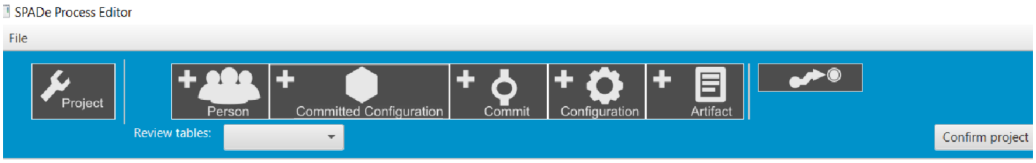
Aplikace umožňuje zpracovávat jak již uložené soubory typu XML pro jejich další úpravu, tak ukládat nově vytvořený proces rovněž do XML. Obě tyto akce lze vyvolat z menu popsaného v kapitole B.2. V obou případech je vyvoláno klasické okno operačního systému, pro výběr souboru k načtení nebo zadání jména a vybrání složky pro uložení. Typ souboru pro načtení i uložení je automaticky XML. Toto nastavení nelze změnit na jiné typy souborů.

B.9 Převod modelu do SQL a vyhledání modelu v projektových datech

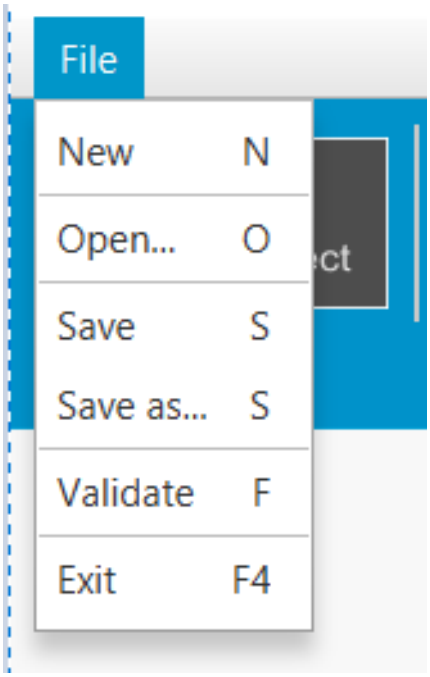
Pro samotný převod modelu do SQL je potřeba internetové připojení, kvůli nutnosti napojení aplikace na datový sklad nástroje SPADe. Pro navázání

spojení slouží tlačítko umístěné v horním panelu aplikace na pravé straně. Při prvním stisku toho tlačítka je uživatel vyzván k zadání přihlašujících údajů do datového skladu, viz obrázek B.9, při následném stisknutí je automaticky zobrazováno okno s přehledem existujících projektů v datovém skladu. Aplikace neumožňuje změnu vybraného databázového systému, pouze změnu uživatele. Uživatel má možnost vybrat v přehledovém okně projekt, ukázka na obrázku B.10, který ho zajímá a tlačítkem OK zahájit převod modelu do SQL a vyhledání modelu ve zvoleném projektu. Výsledky hledání modelu jsou následně zobrazeny v přehledové tabulce, která obsahuje jednotlivé sloupečky pro alias elementu či segmentu, existenci v modelu, počet instancí v modelu, existenci v projektu, počet instancí v projektu, výsledek vyhledání a sloupec s SQL dotazem viz obrázek B.11. Po rozkliknutí konkrétního řádku tabulky je vyvoláno okno, ze kterého lze konkrétní SQL dotaz zkopírovat, viz obrázek B.12.

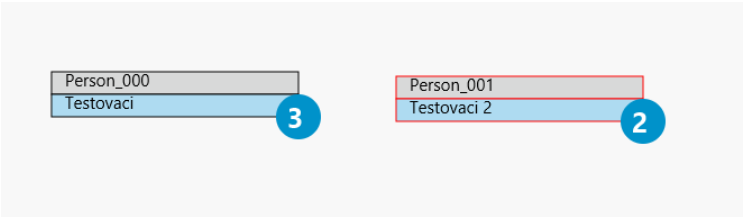
B.10 Obrázky



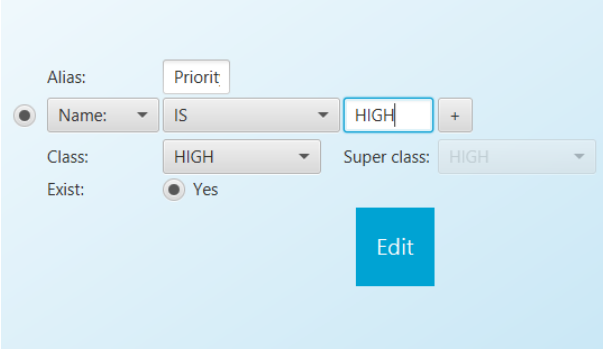
Obrázek B.1: Hlavní obrazovka aplikace



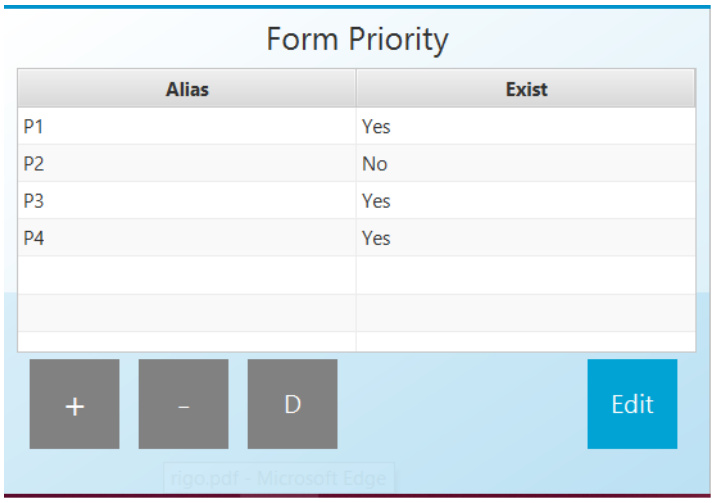
Obrázek B.2: Hlavní menu aplikace



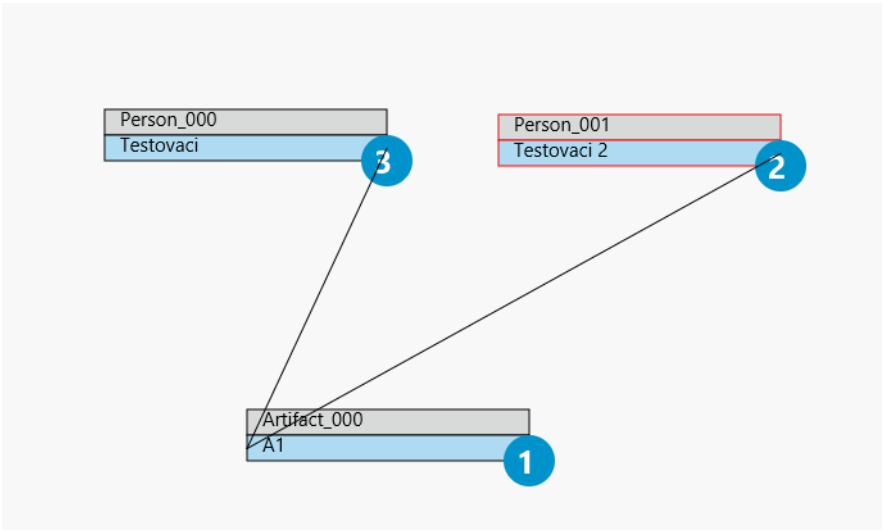
Obrázek B.3: Ukázka prvku na plátně



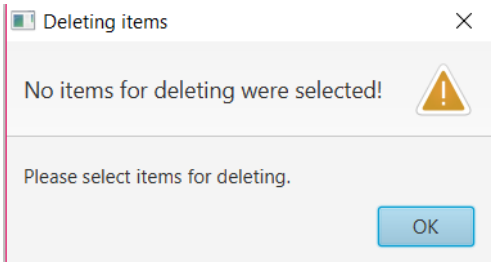
Obrázek B.4: Ukázka editačního panelu



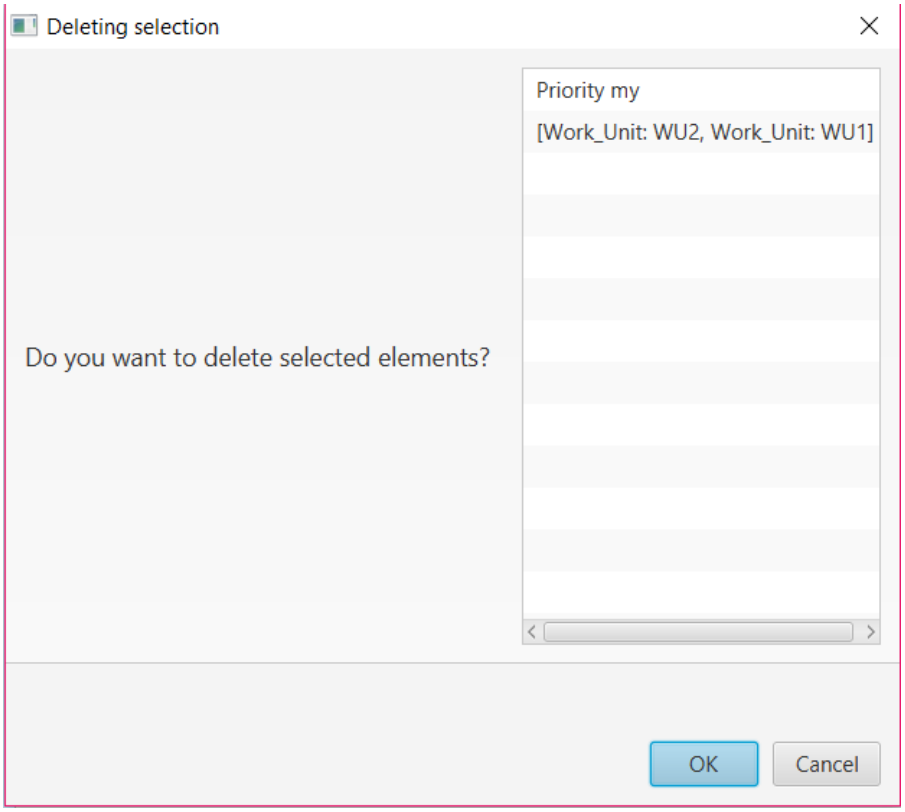
Obrázek B.5: Ukázka přehledového panelu



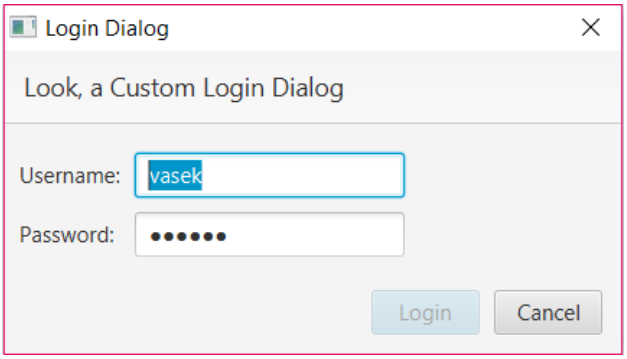
Obrázek B.6: Ukázka spojení prvků



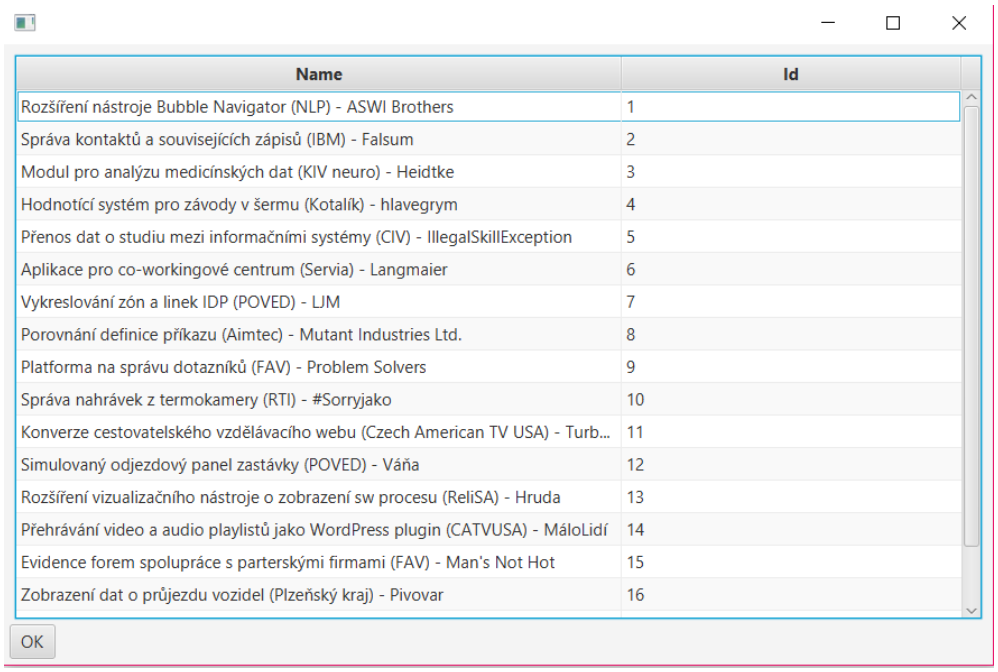
Obrázek B.7: Chybové hlášení o nevybrání položky z tabulky



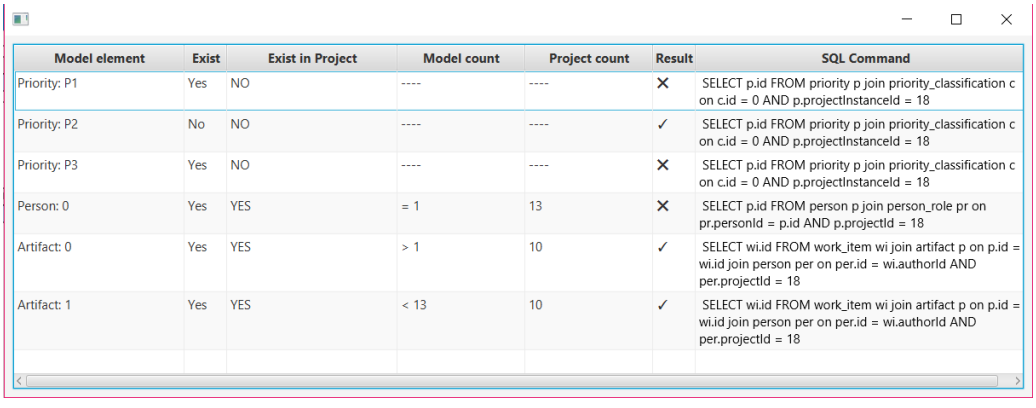
Obrázek B.8: Hlášení s informací o vybraných položkách z tabulky



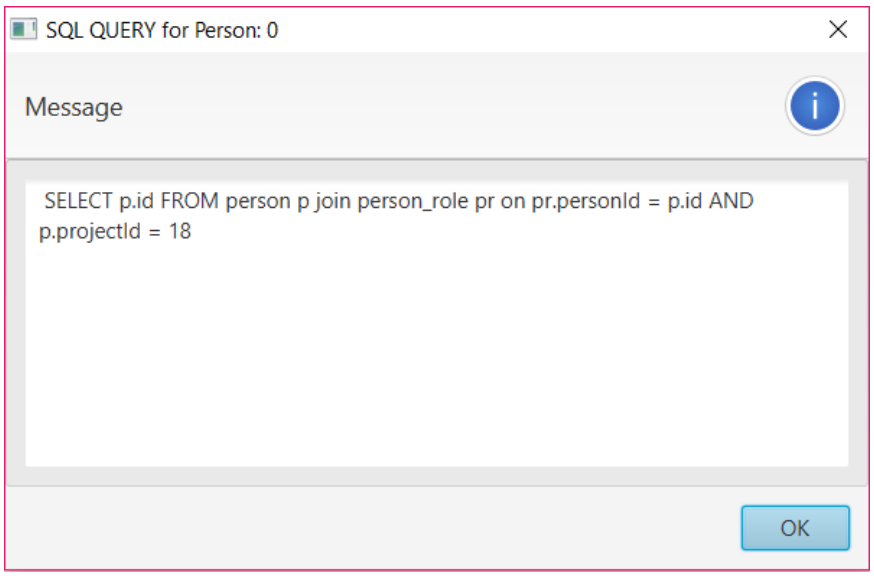
Obrázek B.9: Ukázka okna pro přihlášení



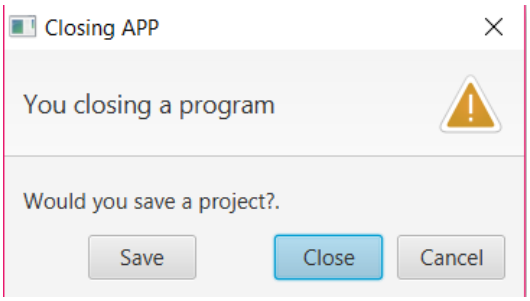
Obrázek B.10: Ukázka okna s projekty



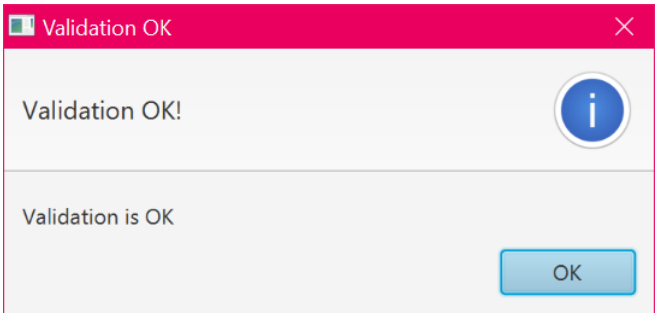
Obrázek B.11: Ukázka okna s verifikací modelu



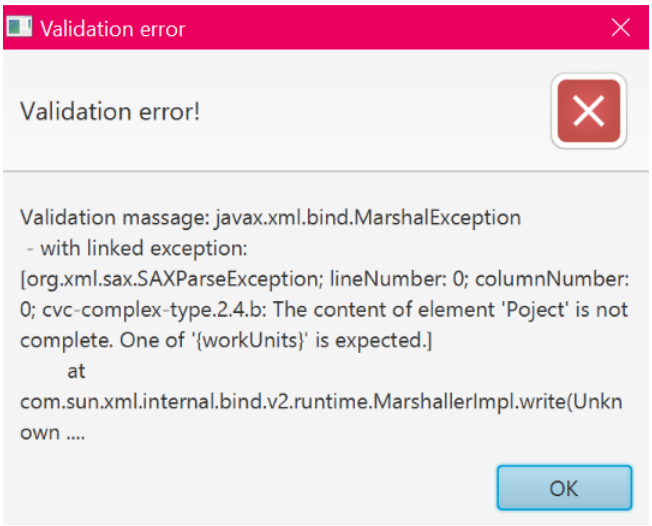
Obrázek B.12: Ukázka okna pro vyjmutí dotazu



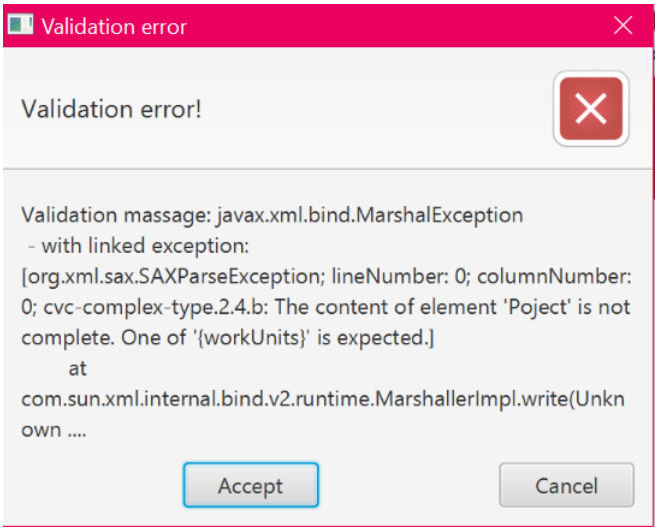
Obrázek B.13: Hlášení o uzavření programu s možností uložení



Obrázek B.14: Úspěšná Validace procesu



Obrázek B.15: Hlášení chyby po ruční validaci



Obrázek B.16: Hlášení chyby při ukládání procesu