2016

Datový model nástroje SPADe a jeho mapování na projektová data z ALM nástrojů

PRÁCE KE ZKOUŠCE Z PŘEDMĚTU MODERNÍ PROGRAMOVACÍ STYLY A METODY

PETR PÍCHA

Obsah

1. Úvod

Softwarové inženýrství je obsáhlým a stále se rozšiřujícím oborem, jež zasahuje do stále většího podílu lidské činnosti. Jednou z jeho nedílnou součástí je vývoj software a s tím související oblast a problematika projektového řízení (PM – Project Management) se stále větší škálou metodologií pro řízení procesů spjatých s řízením softwarového projektu a podpůrných nástrojů. Přesto, že procesy vývoje softwaru ve všech svých variacích jsou stále lépe, detailněji a doménově specifičtěji popisovány, problémy v oblasti PM v praxi stále přetrvávají.

Jednou ze skupin těchto problémů jsou neustále, nebo často se opakující chyby projektového managementu i vývojových týmů. Tyto se obvykle nazývají „bad practices“ nebo také anti-patterny. Existuje velké množství literatury, možností mentorování a školení záměrně cílené na předcházení, identifikaci a odstraňování takovýchto problémů, jakož i nástrojů (např. ALM – Application Lifecycle Management nástroje), které k jejich odhalení mohou poskytnout určité relevantní informace. Neexistuje však ucelený automatizovaný nástroj umožňující přímou detekci definované množiny anti-patternů PM a pomoc při jejich odstranění.

K těmto účelům vzniká na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU) takovýto nástroj nazvaný SPADe (Software Process Anti-pattern Detector). Ten by měl ve své finální podobě agregovat data z ALM nástrojů do jednotného datového skladu a analyzující jejich strukturu za účelem identifikace anti-patternů objevujících se při vývoji softwaru, porovnání s ostatními již uzavřenými projekty podobného typu a nasměrování uživatelů, nejčastěji projektových manažerů či vedoucích vývojových týmů, k jejich možnému řešení nebo odstranění.

Kapitola 2 popisuje celkovou koncepci nástroje SPADe, motivaci k jeho vytvoření, hlavní myšlenky funkčnosti, ALM nástroje, ze kterých čerpá data, a technologie jeho implementace. Kapitola 3 popisuje postup tvorby doménového metamodelu databáze nástroje SPADe a jeho vazby na jiné metamodely procesů vývoje, metodiky a data uchovávaná v ALM nástrojů. Kapitola 4 podrobně popisuje fyzický datový model databáze nástroje SPADe. Kapitola 5 nastiňuje budoucí práce na nástroji SPADe a jeho další možná rozšíření a využití a kapitola 6 celý dokument shrnuje.

2. SPADe nástroj

Tato kapitola popisuje obecně nástroj SPADe, motivaci vedoucí k jeho vzniku, jeho základní koncept a nástroje, s nimiž by měl mít rozhraní.

SPADe neboli Software Process Anti-pattern Detector je softwarový nástroj v současné době vyvíjený na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU). Jeho podstatou je dolování dat z tzv. Application Lifecycle Management (ALM) nástrojů [1] [2] z různých projektů, jejich ukládání do jednotného datového skladu a následná identifikace „bad practices“ neboli anti-patternů projektového řízení (PM – Project Management) a podání informace o jejich výskytu příslušnému uživateli (projektovému manažerovi, vedoucímu vývojového týmu) spolu s porovnáním s podobnými již uzavřenými projekty a radami, jak anti-patterny odstranit či jinak vyřešit.

2.1. Motivace

Hlavní motivací pro vznik SPADe je na jedné straně množství teoretické pomoci s identifikací a předcházením výskytu anti-patternů v odborné literatuře, konzultantské praxi i obsahu různých školení. Na straně druhé pak množství analyzovatelných dat o projektech v ALM nástrojích, které v dnešní době používá téměř každý vývojový projekt bez ohledu na velikost, výsledný produkt nebo použitou metodologii. Tato data, ač velmi užitečná při řízení a sledování stavu projektu, nejsou většinou dále nijak používána za hranici analytických modulů v samotných ALM nástrojích. Ty ale v nabízených analýzách často nejdou za hranici současného projektu a neposkytují objektivní posouzení zobrazených statistik a grafů. Přítomnost anti-patternů, které mohou výrazně ohrozit úspěch daného projektu tak závisí jen na samotném úsudku a zkušenosti projektového manažera či vedoucího vývojového týmu.

Tak vznikla myšlenka vytvořit softwarové nástroj, který by dokázal načíst data o projektu z ALM nástrojů bez ohledu na konkrétní zdrojový nástroj, velikost nebo metodologii, analyzovat je kvůli detekci anti-patternů a porovnat tato data s údaji již uzavřených podobných projektů. Výsledky těchto analýz by pak nástroj zobrazil uživateli ve srozumitelné formě včetně varování před hrozícími problémy projektu spojenými s anti-patterny a návrhů řešení dané situace.

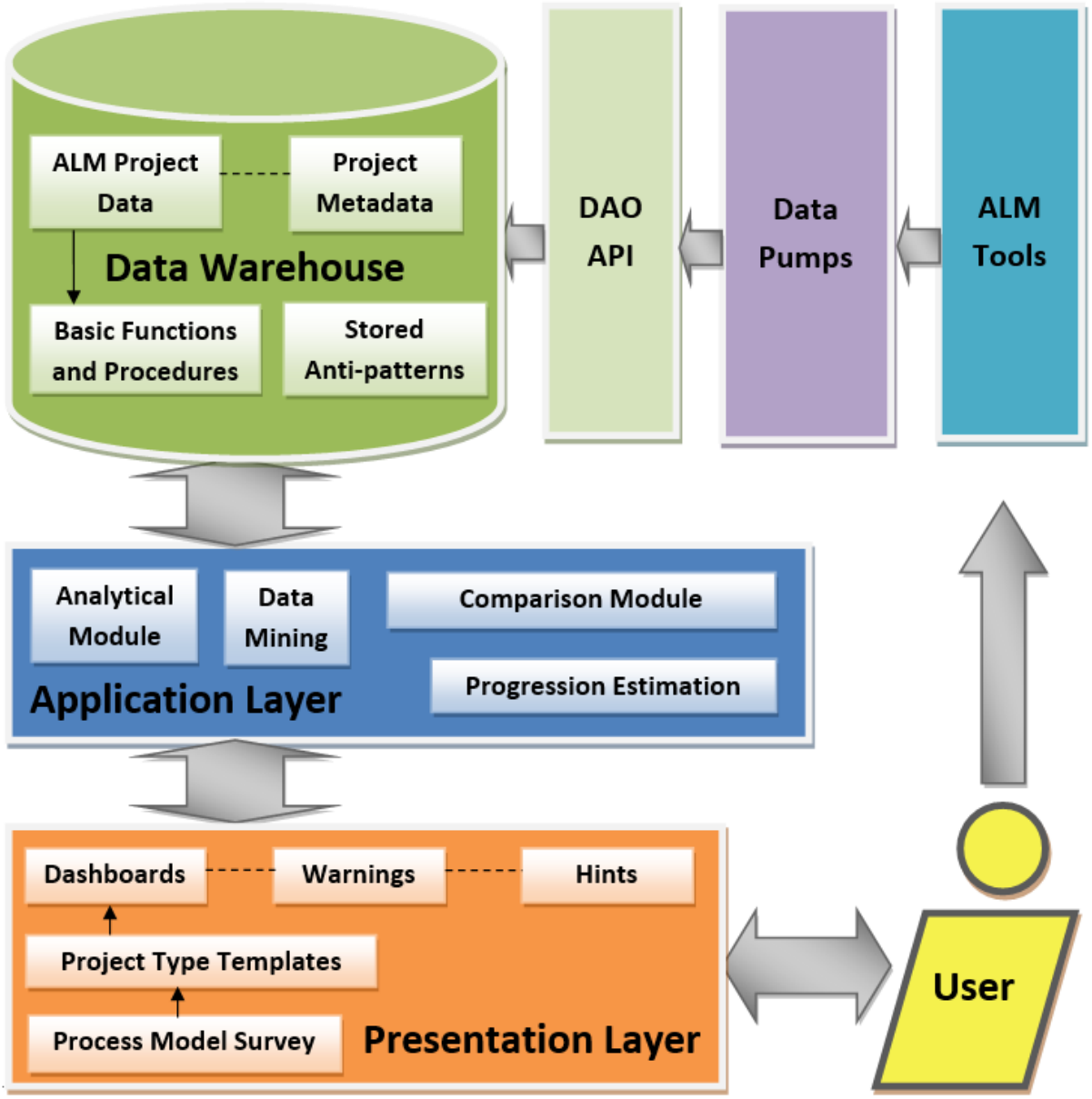
2.2. Koncept

Obrázek 1 demonstruje celkový architektonický princip nástroje SPADe.

2.2.1. Datová vrstva

Funkce SPADe začíná u ALM nástrojů. Z nich SPADe doluje data o jednotlivých projektech skrze datové pumpy. Každá taková pumpa je malým kouskem softwaru na principu Extract – Transform – Load (ETL) specificky implementovaná pro extrakci dat z patřičného ALM nástroje (každý ALM nástroj má tedy vlastní pumpu), jejich převod na unifikovaný formát datového skladu SPADe a jejich uložení do tohoto skladu skrze příslušné Data Access Object (DAO) rozhraní.

Data o jednotlivých projektech v datovém skladu mají strukturu uvedenou v kapitole 4. Kromě nich datový sklad obsahuje také uložená metadata projektů (velikost, doména produktu, metodologie/proces vývoje), šablony pro jednotlivé metodologie a procesy (např.: Vodopád, V-model [3], iterativní, Rational Unified Process [4], agilní [5], Scrum [6], Disciplined Agile Delivery [7], atd.) a jednotlivé anti-patterny, jejichž přítomnost v projektu je nutno detekovat. Formát a struktura metadat, šablon procesů a anti-patternů není předmětem tohoto dokumentu.



Obrázek 1 – Koncept architektury nástroje SPADe

V datovém skladu se tudíž data takto hromadí pro účely porovnání aktuálně zkoumaného projektu s těmi již dokončenými, které jsou přibližně stejné, co se týče metadat. To je nutné zejména proto, že metadata silně ovlivňují dění v projektu. Např. u projektu se sekvenčním procesem (vodopád) nemůžeme hledat patterny související s iteracemi, nebo v projektu, který začíná vývoj „na zelené louce“ (tzv. „green-field project“; tzn. zcela od začátku bez návaznosti na již existující systém) nebudou v prvních fázích figurovat úlohy jako „Seznámení se s dosavadním stavem systému“ či „Pročtení existující dokumentace systému“.

V závislosti na technologii, může datový sklad obsahovat i základní jednoduché agregační funkce a procedury nad daty, jako průměry, sumy, atd. (např. při implementaci v DBMS[1] Oracle).

[1] Data Base Management System – Systém řízení báze dat.

2.2.2. Aplikační vrstva

V aplikační vrstvě jsou data o projektu zkoumána analytickým modulem a dolováním dat. Jednak jsou detekovány anti-patterny a odchylky od deklarovaného procesu, jímž se projekt má řídit, jednak jsou nalézány nové souvislosti mezi daty a další relevantní informace.

Další částí aplikační vrstvy SPADe nástroje je modul pro porovnání s projekty již uzavřenými o přibližně stejných parametrech (metadatech). Toto porovnání slouží další části aplikační vrstvy, kterou je modul pro odhad dalšího vývoje projektu. Filozofie je taková, že pokud podobné projekty s podobnými anti-patterny ve srovnatelné fázi, jako je aktuální zkoumaný projekt, skončily se zpožděním nebo hromaděním dalších problémů, situace se zřejmě bude opakovat i u tohoto projektu.

2.2.3. Prezentační vrstva

Základem prezentační vrstvy je grafické uživatelské prostředí (GUI – Graphical User Interface) v podobě dashboardů. Ty by měly uživateli prezentovat základní statistiky, grafy a nalezené anti-patterny s možností úpravy a personalizace zobrazovaných dat. Dále pak by měly být zobrazeny varování ohledně pravděpodobného neúspěchu projektu (např. překročení mezního termínu ukončení) a rady, které by měly uživateli pomoci s odstraněním či minimalizací dopadů anti-patternů.

Další částí GUI je pak dotazník zobrazený uživateli k vyplnění vždy při importu nového projektu do nástroje. Tento dotazník je cílem na zjištění konkrétního procesu, jímž se projekt řídí. Oproti jednoduché otázce, kde by uživatel vyplnil pouze „RUP“ nebo „DAD“ má tento dotazník za cíl zjistit podrobnější informace a určit proces nebo metodologii na základě odpovědí. Ne se tedy pouze spoléhat na to, co uživatel deklaruje jako proces svého projektu, ale pokud se zjistit, jaká je realita.

2.3. ALM nástroje

ALM (neboli Application Lifecycle Management) nástroje jsou podpůrné softwarové nástroje pro řízení životního cyklu aplikace. Usnadňují procesy jako řízení požadavků, změn, verzí, konfigurací, evidence práce na projektu, plánování, projektové řízení, atd. V současné praxi v podstatě každý softwarové projekt čítající více než jednoho člověka využívá některý ALM nástroj, mnohdy i několik z nich současně. Obvykle je využíván jeden plně kvalifikovaný nástroj, nebo kombinace jednoho pro ticketing (správu změn) a druhého pro správu konfigurací (VCS – Version Control System).

2.3.1. Zvolená sada

Jako základní sada pro vývoj nástroje SPADe, byly zvoleny následující ALM nástroje:

- Apache Subversion [8],
- Assembla [9],
- Atlassian Jira [10],
- Bugzilla [11],
- Git [12],
- GitHub [13],
- IBM Rational Team Concert (RTC) [14]
- a Redmine [15].

Tato sada byla zvolena díky jejich rozšířenosti na trhu [16], přístupnému a využitelnému programovacímu rozhraní (API – Application Programming Interface) k dolování dat, využívání na KIV v rámci výuky nebo partnery KIVu v praxi, nebo (v případě GitHubu) aktuální popularitě. SVN a Git jsou zástupci čistě VCS, Assembla a Bugzilla jsou pouze ticketovací systémy, GitHub, RTC a Jira pak plní obě

funkce. V případě Redmine jde sice pouze o ticketovací systém, který lze ale přímo propojit s SVN, Gitem i GitHubem pro potřeby řízení konfigurací.

Sada nástrojů se může dále rozvíjet po stabilizaci prototypu celého SPADe nástroje. To je také v plánu ať už na straně KIVu, nebo při využití prototypu třetí stranou. V podstatě jediným úkonem nutným k zařazení dalšího nástroje do sady je implementace příslušné datové pumpy, využívající vstupní DAO datového skladu SPADe a transformující data to formátu popsaného v kapitole 4.

2.4. Technologie

Jako technologie implementace nástroje SPADe byl zvolen programovací jazyk Java. Hlavním důvodem je existence API pro dolování dat pumpami v Javě nebo REST technologii u všech ALM nástrojů z vybrané sady. Pro realizaci Datového skladu je prozatím používána MySQL databáze. Na konkrétní DBMS technologii je realizace ale co možná nejvíce nezávislá, díky implementaci doménových a DAO tříd v JPA, konkrétněji Hibernate.

3. SPADe metamodel

Protože nástroj SPADe by měl sbírat a ukládat data o projektech z různých ALM nástrojů za účelem nejen porovnání, ale i stejného systému hledání případných anti-patternů, je nutné, aby data byla ukládána v jednotné databázové struktuře. Tuto strukturu dat projektů od výchozích myšlenek a bodů přes doménový model popisuje tato kapitola.

Článek [17] blíže popisující doménový SPADe metamodel a jeho sestavení bude publikován na přelomu srpna a září na konferenci Euromicro Conference series on Software Engineering and Advanced Applications (SEAA).

3.1. Hledání vhodného metamodelu

Primární snahou bylo najít již existující metamodel schopný pokrýt veškeré potřeby nástroje SPADe a přitom bez zbytečně redundantních dat. Protože neexistuje aplikace přesně plnící funkci SPADe, byla existence takového metamodelu nepravděpodobná.

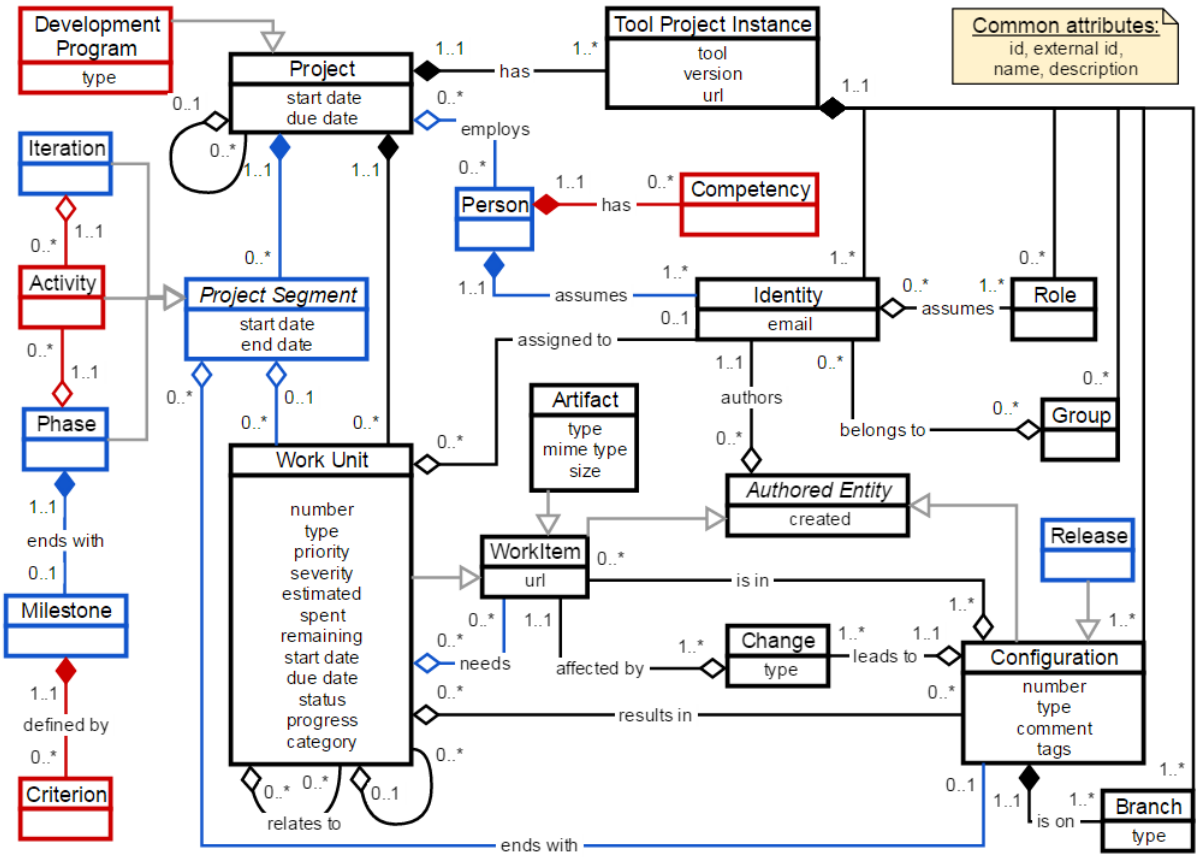
Přesto průzkum výzkumu v oboru odhalil některé podobné projekty. Framework vyvíjený na Aristotelově Univerzitě v Thessaloniki se zaměřuje na anti-patterny spolupráce vývojářů založené na jejich osobnostních charakteristikách a rozdílech v temperamentech [18] [19] [20] [21] [22] [23]. Výzkumná skupina v Iránu studuje procesní patterny v různých doménách jako aspektově orientovaný vývoj [24] a komponentový vývoj [25]. Další výzkumná skupina v Řecku prezentuje proceduru pro extrakci procesních patternů z různých softwarových metodologií [26]. Oberhauser a spol. vyvíjejí nástroj CoSEEEK pro detekci volných časových úseků v osobním workflow vývojáře a následné vkládání vhodných QA úkolů a aktivit [27] [28] [29] [30] [31]. CoSEEEK používá Software Engineering Workflow Language (SEWL) pro modelování osobních procesů. Žádný z nalezených projektů však nepoužívá model procesů vhodný k přímému převzetí.

To potvrdila i studie revidující všechny prostředky používané k modelování procesů vývoje software pokrývající všechny metajazyky, metamodely a další prostředky publikované mezi lety 2000 a 2011 [32]. Konkrétně byly z této studie zkoumány modely Software and Systems Process Engineering Metamodel (SPEM) 2.0 [33], extensible-SPEM (eSPEM) [34], Variability SPEM (VSPEM) [35] [36], Process Meta Model (PMMM) [37] a ISO 24744 [38] [39].

Z výše uvedených faktů vyplynula potřeba tvorby vlastního metamodelu s využitím maxima konceptů z jiných zkoumaných metamodelů a dalších zdrojů (realita vývoje pomocí ALM nástrojů a metodiky vývoje software) a zároveň pokrývající požadavky na SPADe a odpovídající datům dolovatelným z ALM nástrojů. Konkrétně SEWL, Open Services for Lifecycle Collaboration (OSLC) [40] [41] a ISO 29110 [42] [43] byly použity pro převzetí některých konceptů a terminologie.

3.2. Tvorba vlastního metamodelu

Zde je popsán myšlenkový proces za sestavením doménového metamodelu nástroje SPADe, který je vyobrazen na obrázku 2. Černě vyznačené entity jsou víceméně přímo mapovatelné z ALM nástrojů. Modré vyžadují jistou míru analýzy nebo rozhodnutí uživatele. Červené elementy pak musí uživatel definovat téměř celé, protože ani k jejich přibližnému určení nejsou ALM data uzpůsobena, přesto jejich přítomnost může významně rozšířit sadu identifikovatelných anti-patternů a pomoci jejich detekci.



Obrázek 2 – Doménový datový metamodel nástroje SPADe

3.2.1. Původ jednotlivých konceptů

Každý model procesu vývoje software spočívá kromě projektu samotného (*Project*) na třech základních konceptech – úkol (*Work Unit*[2]), role (*Role*) a artefakt[3] (*Artifact*).

Projekt je možné dělit na menší celky v závislosti na metodologii. Segmenty projektu, které je pro účely nástroje SPADe nutné modelovat tak, aby výsledný model byl schopen pojmout všechny základní metodiky vývoje, jsou fáze (*Phase*) pro sekvenční metodiky (Vodopád, V-model) a Unified Process (UP) a iterace[4] (*Iteration*) pro iterativní a agilní metodiky. Fáze jsou navíc většinou ukončeny definovaným milníkem (*Milestone*). Ad-hoc procesy nemusejí používat ani jeden z typů segmentů.

Dále je nutné zachytit reality vývoje software jako konfigurace[5] (*Configuration*), změna (*Change*), release[6] (*Release*) a osoba[7] (*Person*).

Kvůli specifikům dat z ALM nástrojů je třeba přidat samotné instance projektu v jednotlivých nástrojích (*Tool Project Instance*), jelikož projekt jich může používat více. Dalšími elementy z ALM nástrojů jsou identita (*Identity*) osoby v daném nástroji, skupiny těchto identit (*Group*) a vývojová větev (*Branch*). V ALM nástrojích je záznamem o úkolu a jeho plnění tzv. ticket. Ten je ale také možné

[2] Název převzat z SEWL.
[3] Terminologie z Rational Unified Process (RUP) ale i běžně používaná.
[4] V agilních metodikách (jako například) též nazývané sprinty.
[5] Verze aplikace zapouzdřující stav všech jejích položek (artefaktů) v daném časovém okamžiku.
[6] Stabilizovaná verze vyvíjené aplikace obvykle na konci některé fáze či iterace
[7] Samotné role pro účely hledání anti-patternů nestačí, je nutno mapovat činnosti a jejich výstupy na konkrétní lidi.

považovat za artefakt, jelikož podléhá změnám a jedná se o záznam nikoli úkol samotný. Proto byla vytvořena entity *Work Item*[8], která umožňuje zacházet s artefakty a tickety stejně.

Poslední skupinou objektů jsou červené, z většiny uživatelem definované elementy. Jsou to kompetence (*Competency*) osoby, kritéria (*Criterion*) dosažení milníku, skupiny spřízněných úkolů (*Activity*) a série příbuzných projektů (*Development Program*).

3.2.2. Výběr dat dolovatelných z ALM nástrojů

Celková množina elementů a atributů metamodelu SPADe je založena na průniku dat z ALM nástrojů. To znamená, že pokud všechny nástroje (samozřejmě s přihlédnutím na rozdělení ticketovací/VCS nástroje) obsahovaly stejné nebo srovnatelné údaje, byl daný element nebo atribut do metamodelu zahrnut. Následně byly vybrány a do metamodelu doplněny i některé element/atributy obsažené pouze ve většině, nebo i menšině nástrojů, pokud byly uznány za užitečné pro potřeby analýz a detekce anti-patternů.

Vybraná API ALM nástrojů

Některé z vybraných ALM nástrojů disponují více než jedním přijatelným API pro extrakci dat. V tomto dokumentu jsou uvedeny údaje z primárně vybrané sady, vždy jedno API pro jeden nástroj. Pro další doplnění údajů však bude zjevně třeba čerpat pro minimálně některé z nástrojů z více API, jelikož v některých případech žádné jedno API neposkytuje data kompletní. Primárně vybraná sada API se skládá z:

- **Assembla** – Assembla API Reference;
- **Bugzilla** – Bugzilla for Java 2.0.3 API;
- **Git** – JGit – Parent 4.3.1.201605051710-r API;
- **GitHub** – GitHub API for Java 1.76 API;
- **Jira** – Atlassian Jira 7.1.7 API;
- **RTC** – Jazz Client API Specification 5.0.1;
- **Redmine** – redmine-java-api 2.6.0 API;
- **SVN** – SVNKit 1.7.4.

3.2.3. Možnosti popisu metodik

Pokud se schopnosti metamodelu pokrýt procesy postavené na všech základních metodikách týče, jde vlastně jen o problém segmentace projektu.

Vyšli jsme od základních konceptů (artefakt, role, úkol), ze kterých se nevyhnutelně skládají všechny procesy a přidali jen koncepty používané ve vývoji obecně (změna, konfigurace, osoba, apod.) a koncepty přímo z ALM nástrojů (větev, identity skupina, apod.). Proto jediný způsob, jak by tyto koncepty nemodelovali projekt, je, že by tento nepoužíval ALM nástroje, a tudíž nepatřil do oblasti našeho zájmu. Metodiky a procesy od nich odvozené se na úrovni těchto konceptů mohou lišit ve specifikách jednotlivých úkolů a artefaktů a jejich uspořádání, ale nikoli v přítomnosti těchto abstrakcí a jejich základních atributech.

Zásadním rozdílem mezi metodikami z tohoto pohledu je tedy jen rozdělení úkolů do fází (sekvenční metodiky), iterací (iterativní a agilní metodiky), obojího (RUP, DAD[9], apod.) nebo ani

[8] Termín převzat ze SPEM 2.0.
[9] Disciplined Agile Delivery

jednoho (ad-hoc procesy). Některé metodiky sice specifikují úkoly každodenní nebo každotýdenní, tyto segmenty však lze v modelu jasně určit pomocí dat v jednotlivých úkolech a není proto pro ně třeba vytvářet speciální abstrakce.

Existuje tedy dostatečná míra jistoty, že metamodel může zachytit data projektu řízeného libovolným procesem podle jakékoli z nejznámějších metodik. Tuto skutečnost je ale samozřejmě nutné validovat praktickými experimenty po dokončení implementace datové vrstvy nástroje SPADe.

4. Mapování ALM nástrojů na SPADe metamodel

Tato kapitola popisuje strukturu již přímo datového modelu SPADe. Návaznosti na doménový model z předchozí kapitoly jsou zřejmé a nejsou dále rozebírány.

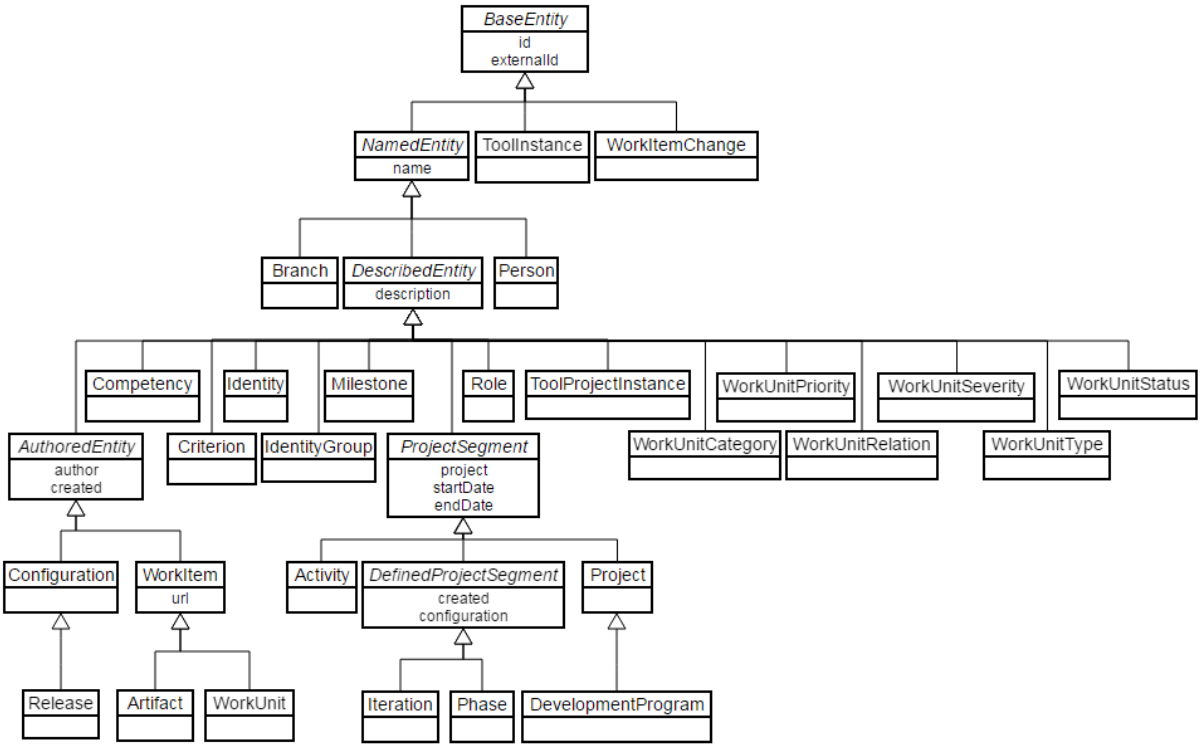
4.1. Obecně

Zde jsou popsány všechny relevantní informace nutné před popisem samotného datového modelu a jeho jednotlivých tříd. Je znázorněno schéma dědičnosti a pospány abstraktní třídy a problém výčtových datových typů.

4.1.1. Schéma dědičnosti tříd a abstraktní třídy

Obrázek 3 zobrazuje schéma dědičnosti mezi jednotlivými třídami datového modelu nástroje SPADe. Třídy, od kterých jiné dědí, mají zobrazeny děděné atributy. Výjimkou jsou *Project* a *Configuration*, od kterých jejich potomci dědí všechny atributy uvedené v příslušných podkapitolách této kapitoly. Názvy abstraktních tříd jsou zvýrazněny kurzívou.

Dále jsou jednotlivé abstraktní třídy popsány jejich významem a seznamem a datovým typem jejich atributů.



Obrázek 3 – Schéma dědičnosti tříd datového modelu nástroje SPADe

4.1.2. BaseEntity

Základní třída, od níž dědí všechny ostatní třídy. Neděděné atributy třídy popisuje tabulka 1.

Tabulka 1 – Atributy třídy BaseEntity

Atribut	Datový typ	Význam
id	int	primární klíč
externalId	String	identifikátor elementu ze zdrojového ALM nástroje; může být prázdný, pokud objekt není přímo převzat z ALM nástroje

4.1.3. NamedEntity

Nadřazená třída pro všechny, které potřebují jméno jako atribut. Neděděné atributy třídy popisuje tabulka 2.

Tabulka 2 – Atributy třídy NamedEntity

Atribut	Datový typ	Význam
name	String	jméno objektu

4.1.4. DescribedEntity

Nadřazená třída pro všechny, které potřebují textový popis jako atribut. Neděděné atributy třídy popisuje tabulka 3.

Tabulka 3 – Atributy třídy DescribedEntity

Atribut	Datový typ	Význam
description	String	popis objektu

4.1.5. AuthoredEntity

Nadřazená třída pro *Artifact*, *Configuration* a *WorkUnit*, které uchovávají autora a datum vytvoření. Neděděné atributy třídy popisuje tabulka 4.

Tabulka 4 – Atributy třídy BaseEntity

Atribut	Datový typ	Význam
author	Identity	autor objektu
created	Date	datum vytvoření

4.1.6. ProjectSegment

Obalující třída pro *Project* a všechny jeho menší části i vyšší celky. Neděděné atributy třídy popisuje tabulka 5.

Tabulka 5 – Atributy třídy ProjectSegment

Atribut	Datový typ	Význam
project	Project	u *Activity*, *Iteration* a *Phase* projekt, do kterého patří; u *Project* nadřazený projekt; u *DevelopmentProgram* potenciálně nadřazená série (sérií) projektů
startDate	Date	datum započetí
endDate	Date	datum ukončení

4.1.7. DefinedProjectSegment

Obalující třída pro *Iteration* a *Phase*. Neděděné atributy třídy popisuje tabulka 6.

Tabulka 6 – Atributy třídy DefinedProjectSegment

Atribut	Datový typ	Význam
created	Date	datum vytvoření
configuration	Configuration	konfigurace odpovídající konci dané části projektu (potenciální *Release*)

4.1.8. Problém výčtových typů

Mnohé atributy objektů z ALM nástrojů je vhodné reprezentovat jako výčtové typy. Konkrétně se jedná o:

- priority,
- závažnosti,
- stavy
- a typy úkolů,
- typy vztahu mezi úkoly
- a role.

Množiny hodnot těchto atributů stejně jako hodnoty samy se ale mohou mezi jednotlivými ALM nástroji lišit a některé nástroje dokonce povolují uživateli definovat vlastní. Proto je vhodné mít ve SPADe nástroji stabilní množinu hodnot, které tyto atributy mohou nabývat a na které se hodnoty z ALM nástrojů mapují. Výsledné hodnoty výčtových typů ve SPADe nazýváme třídy, neboť více hodnot atributu z ALM nástrojů se může mapovat na stejnou třídu hodnot ve SPADe. Z tohoto důvodu jsou v datovém modelu vytvořeny následující výčtové typy:

- *RoleClass* – třída role,
- *WorkUnitPriorityClass* – třída priority úkolu,
- *WorkUnitRelationType* – třída vztahu mezi dvěma úkoly,
- *WorkUnitSeverity*Class – třída závažnosti úkolu,
- *WorkUnitStatusClass* – třída stavu úkolu,
- *WorkUnitTypeClass* – třída typu úkolu.

Navíc priorita a stav úkolu vyžadují ještě druhou úroveň sdružování z důvodu jak dat ALM, tak pro lepší analýzu anti-patternů. Proto jsou zřízeny výčtové typy:

- *WorkUnitPrioritySuperclass* – nadtřída priority úkolu,
- *WorkUnitStatusSuperclass* – nadtřída typu úkolu.

Hodnoty a mapování všech výše uvedených výčtových datových typů je uvedeno v přílohách.

V datovém modelu SPADe nástroje jsou použity i další výčtové datové typy, které ale nemapují výčtové typy ALM nástrojů a slouží pouze pro omezení hodnot některých atributů, které by jinak byly textové. Jsou to:

- *ArtifactClass* – třída (typ) artefaktu
  - hodnoty: „*File*“, „*Folder*“, „*Wikipage*“, „*Email*“
- *ProgramType* – type série projektů (*DevelopmentProgram*)
  - hodnoty: zatím neurčeny
- *Tool* – zdrojový ALM nástroj instance projektu

4.2. Jednotlivé třídy SPADe

Následující podkapitoly popisují jednotlivé třídy datového modelu nástroje SPADe, přesněji entitní třídy mapované pomocí JPA na databázi nástroje, a jejich atributy. Každá tabulka ze sloupců pro název, datový typ a význam atributu objektu třídy v nástroji SPADe. Atributy děděné od abstraktních tříd nejsou uváděny, takže některé třídy nemají popsány žádné dodatečné atributy. Výjimkou jsou atributy, které v kontextu třídy slouží ke specifičtějšímu účelu, než jak bylo popsáno u abstraktní třídy (např. *description* u *Identity*).

Mapování dat ALM nástrojů na třídy SPADe je uvedeno v příloze B.

4.2.1. Activity

Třída reprezentuje aktivitu, skupinu spřízněných úkolů se společným cílem uvnitř jedné fáze nebo iterace. Příkladem může být aktivita „*Zpřesnění požadavků*“ skládající se z úkolů „*Schůzka se zákazníkem*“, „*Úprava modelu požadavků*“, „*Sepsání specifikací požadavků*“, atd.

Identifikace kandidátních ALM dat pro naplnění instancí této třídy je předmětem hlubší analýzy, kterou bude pravděpodobně vykonávat až analytický modul na aplikační vrstvě a jejich definitivní potvrzení bude nejspíše uživatelskou záležitostí. Nicméně prvotními kandidáty jsou sady spřízněných úkolů z ALM nástrojů v rámci jedné iterace nebo fáze. Především pak nadřazené úkoly a sady úkolů se vzájemnými vztahy typu „*related*“, „*depends on*“ nebo „*blocks*“ (viz mapování třídy *WorkUnitRelation* v přílohách)

4.2.2. Artifact

Třída reprezentuje artefakty jak z VCS nástrojů (soubory a složky; tzv. konfigurační položky), tak z ticketovacích nástrojů (wiki stránky, přílohy ticketů, položky vnitřního systému správy dokumentů, apod.). V budoucnosti by měla pokrývat i emaily zasílané v rámci vývojového týmu. Neděděné atributy třídy popisuje tabulka 7.

Tabulka 7 – Atributy třídy Artifact

Atribut	Datový typ	Význam
url	String	cesta k artefaktu v patřičném nástroji
artifactClass	ArtifactClass	třída (typ) artefaktu
mimeType	String	typ obsahu souboru
size	long	paměťová velikost artefaktu

4.2.3. Branch

Třída reprezentuje vývojovou větev ve VCS nástroji. Neděděné atributy třídy popisuje tabulka 8.

Tabulka 8 – Atributy třídy Branch

Atribut	Datový typ	Význam
isMain	boolean	„*true*“ pokud se jedná o hlavní vývojovou větev (*master*/*trunk*), jinak „*false*“

4.2.4. Competency

Třída představuje kompetence osob. Může zahrnovat zkušenosti, certifikace, školení, atd. Přesná implementace zatím není ustanovena, zřejmě se bude jednat o textový zápis (v atributu *description*) zadaný uživatelem podle pevně daného schématu/gramatiky tak, aby byl zpracovatelný a jeho obsah mapovatelný na ostatní data (pro kontrolu adekvátnosti kompetencí osoby vzhledem k její roli v projektu).

4.2.5. Configuration

Třída reprezentuje stav vyvíjené aplikace v daném časovém bodě, tedy souhrn jejích artefaktů a úkolů v dané verzi (konfiguraci). Pokud jde o konfiguraci přímo z VCS nástroje, jedná se o revizi. Uchovávají se však veškeré konfigurace, tzn. stavy po jakékoli změně i ticketu. Neděděné atributy třídy popisuje tabulka 9.

Atribut *description* u objektů třídy *Configuration* uchovává primárně komentář zadaný uživatelem ALM nástroje při commitu do VCS/úpravě ticketu.

Atribut *url* je zamýšlen jako zdroj odkazu, kterým bude možné přejít z GUI SPADe přímo na zobrazení příslušné konfigurace v GUI ALM nástroje.

Tabulka 9 – Atributy třídy Configuration

Atribut	Datový typ	Význam
Number	String	číslo revize, pokud konfigurace je revizí (může být jiné než ID v daném nástroji)
changes	Collection<WorkIntemChange>	soubor změn od minulé konfigurace, které vedly ke stavu, který tato konfigurace popisuje (ve VCS systému je možné změnit více souborů najednou)
isRevision	boolean	„*true*“, pokud jde o revizi z VCS, jinak „*false*“
artifacts	Collection<Artifact>	artefakty přítomné v konfiguraci
workUnits	Collection<WorkUnit>	tickety přítomné v konfiguraci
Branch	Branch	vývojová větev, na které revize leží, pokud jde o revizi
Tags	Collection<String>	sada značek, které na sobě revize má, pokud jde o revizi

4.2.6. Criterion

Třída reprezentuje kritérium dosažení milníku. Přesná implementace zatím není ustanovena, zřejmě se bude jednat o textový zápis (v atributu *description*) zadaný uživatelem podle pevně daného schématu/gramatiky tak, aby byl zpracovatelný a jeho obsah mapovatelný na ostatní data (pro kontrolu daty prokázaného dosažení milníku).

4.2.7. DevelopmentProgram

Třída reprezentuje sérii spřízněných projektů. Může se jednat o vývojovou řadu jednotlivých verzí stejné aplikace, sadu projektů pro vývoj též aplikace pro různé platformy apod. Hlavním cílem je možnost analyzovat jak jednotlivé dílčí projekty, tak nadřazený celek. Ten by měl podléhat různým zkoumáním v závislosti na typu sady projektů. Neděděné atributy třídy popisuje tabulka 10.

Třída dědí od třídy *Project*, tudíž další její atributy jsou uvedeny v popisu této třídy.

Tabulka 10 – Atributy třídy DevelopmentProgram

Atribut	Datový typ	Význam
type	ProgramType	typ sady projektů
projects	Collection<Project>	projekty

4.2.8. Identity

Třída reprezentuje identitu/profil daného člena vývojového týmu v jednom z ALM nástrojů. Protože každý projekt může využívat více nástrojů, je zapotřebí uchovávat všechny identity v nich a následně je sdružovat do objektů třídy *Person* pro reprezentaci jednotlivých osob. Neděděné atributy třídy popisuje tabulka 11.

Atribut *description* v případě objektů třídy Identity uchovává celé jméno osoby, zatímco atribut *name* jméno přihlašovací (*login*/*username*). Obojí je pak základem pro agregaci identit a jejich slučování do objektů třídy *Person*.

Tabulka 11 – Atributy třídy Identity

Atribut	Datový typ	Význam
roles	Collection<Role>	seznam rolí sdružených s identitou
email	String	emailová adresa sdružená s identitou

4.2.9. IdentityGroup

Třída reprezentuje skupinu uživatelů některého z ALM nástrojů. Některé ticketovací nástroje umožňují sdružovat uživatele do skupin. Tento údaj může být užitečný pro analýzy dat a detekci především sociálních anti-patternů. Neděděné atributy třídy popisuje tabulka 12.

Tabulka 12 – Atributy třídy IdentityGroup

Atribut	Datový typ	Význam
members	Collection<Identity>	členové skupiny

4.2.10. Iteration

Třída reprezentuje iteraci vývoje produktu v rámci projektu. Neděděné atributy třídy popisuje tabulka 13.

Atribut *activities* je stejný jako v případě třídy *Phase*, která společně s *Iteration* dědí od stejné třídy *DefinedProjectSegment*. Atribut *activities* však nemohl být přesunut do nadřazené třídy kvůli možnostem mapování JPA na databázi. Jelikož aktivita může patřit nejvýše do jedné fáze a zároveň nejvýše do jedné iterace je nutné tento atribut přesunout až to tříd potomků.

Tabulka 13 – Atributy třídy Iteration

Atribut	Datový typ	Význam
activities	Collection<Activity>	aktivity v rámci iterace

4.2.11. Milestone

Třída reprezentuje milník, tj. konec některé fáze. Neděděné atributy třídy popisuje tabulka 14.

Tabulka 14 – Atributy třídy Milestone

Atribut	Datový typ	Význam
criteria	Collection<Criterion>	kritéria dosažení milníku

4.2.12. Person

Třída reprezentuje osobu podílející se na projektu. Typicky člena vývojového týmu, projektového manažera, mentora nebo další osoby s přístupem do ALM nástrojů používaných v projektu. Objekt pod sebou sdružuje všechny identity dané osoby z různých ALM nástrojů. Neděděné atributy třídy popisuje tabulka 15.

Tabulka 15 – Atributy třídy Person

Atribut	Datový typ	Význam
identities	Collection<Identity>	identity/profily osoby v jednotlivých ALM nástrojích
competencies	Collection<Competency>	kompetence osoby

4.2.13. Phase

Třída reprezentuje fázi vývoje produktu. Jedná se o fáze obvykle pevně definované v metodikách vývoje, např. Vodopád, RUP. Neděděné atributy třídy popisuje tabulka 16.

Atribut *activities* je stejný jako v případě třídy *Iteration*, která společně s *Phase* dědí od stejné třídy *DefinedProjectSegment*. Atribut *activities* však nemohl být přesunut do nadřazené třídy kvůli možnostem mapování Java Persistence API na databázi. Jelikož aktivita může patřit nejvýše do jedné fáze a zároveň nejvýše do jedné iterace je nutné tento atribut přesunout až to tříd potomků.

Tabulka 16 – Atributy třídy Phase

Atribut	Datový typ	Význam
activities	Collection<Activity>	aktivity v rámci fáze
milestone	Milestone	milník fáze

4.2.14. Project

Třída reprezentuje projekt jako takový. Jedná se o agregaci reprezentací projektů z jednotlivých ALM nástrojů (objekty *ToolProjectInstance*), které se v projektu používají. Centrální prvek celého datového modelu, jelikož data se budou nahrávat do úložiště po projektech a analýzy budou z většiny probíhat na úrovni jednotlivých projektů a porovnání mezi nimi. Neděděné atributy třídy popisuje tabulka 17.

Tabulka 17 – Atributy třídy Project

Atribut	Datový typ	Význam
personnel	Collection<Person>	lidé zapojení v projektu

4.2.15. Role

Třída reprezentuje roli některé z osob pracujících na projektu. Neděděné atributy třídy popisuje tabulka 18.

Tabulka 18 – Atributy třídy Role

Atribut	Datový typ	Význam
roleClass	RoleClass	třída role

4.2.16. Release

Třída reprezentuje speciální konfiguraci vyvíjeného produktu. Obvykle se jedná o stabilní, ač většinou prozatímní verzi vhodnou k předání zákazníkovi. Release se obvykle pojí ke konci iterace nebo fáze projektu. Všechny atributy dědí od *Configuration*.

4.2.17. ToolInstance

Třída reprezentuje instanci obvykle serverové části ALM nástroje. Zatímco GitHub má jen jednu instalaci a funguje jako cloudová služba, většina nástrojů (např. RTC, Redmine, SVN) mají v obvyklém případě vždy svou instanci instalovanou na serveru organizace nebo společnosti. U těchto instalací je krom nástroje samotného nutné identifikovat verzi nástroje, neboť ta může mít vliv na dostupná data a možnosti API. Rovněž je nutné identifikovat zdrojové instalace u instancí projektů, identit a dalších objektů pro případ, že by v několika různých instalacích nástrojů existovaly objekty se stejnými jmény a ID, nebo takové, které ID nemají. Neděděné atributy třídy popisuje tabulka 19.

Tabulka 19 – Atributy třídy ToolInstance

Atribut	Datový typ	Význam
tool	Tool	nástroj
version	String	verze

4.2.18. ToolProjectInstance

Třída reprezentuje údaje o projektu v jedné konkrétní instalaci ALM nástroje. Projekt může mít těchto reprezentací víc a údaje z nich agreguje objekt třídy *Project*. Neděděné atributy třídy popisuje tabulka 20.

Některé výčtové typy jako priority, role a skupiny mohou některé nástroje vztahovat přímo k instalaci své serverové části, některé však přímo k reprezentaci projektu v ní. Proto musí být tyto

kolekce přiřazeny k této třídě. Obdobně atribut *identity* nereprezentuje v tomto případě všechny profily uživatelů v instalaci nástroje, ale jen ty zainteresované v tomto projektu.

Atribut *url* (cesta k serveru instalace a reprezentaci projektu v ní) je ten, který předává uživatel SPADe jako ukazatel na projektová data, která chce do nástroje SPADe přenést a analyzovat.

Atribut *configuration* buď znamená revize (u VCS nástrojů), všechny stavy po jednotlivých změnách ticketů, wiki stránek, apod. (u ticketovacích nástrojů), nebo směs obojího (u nástrojů se funkcemi obojího).

Tabulka 20 – Atributy třídy ToolProjectInstance

Atribut	Datový typ	Význam
toolInstance	ToolInstance	Instalace ALM nástroje, která je zdrojem reprezentace projektu
project	Project	projekt, k němuž tato reprezentace náleží
url	String	cesta k reprezentaci projektu v ALM nástroji na serveru
priorities	Collection<WorkUnitPriority>	priority definované v této reprezentaci projektu
severities	Collection<WorkUnitSeverity>	závažnosti definované v této reprezentaci projektu
workUnitTypes	Collection<WorkUnitTypes>	typy úkolů definované v této reprezentaci projektu
statuses	Collection<WorkUnitStatus>	stavy úkolů definované v této reprezentaci projektu
categories	Collection<WorkUnitCategory>	kategorie úkolů definované v této reprezentaci projektu
identities	Collection<Identity>	identity uživatelů v této reprezentaci projektu
groups	Collection<IdentityGroup>	skupiny uživatelů v této reprezentaci projektu
roles	Collection<Role>	role definované v této reprezentaci projektu
configuration	Collection<Configuration>	konfigurace v této reprezentaci projektu (revize ve VCS/stavy po změnách ticketů nebo wiki stránek v ticketovacím nástroji)
branches	Collection<Branch>	vývojové větve ve VCS reprezentaci projektu
tags	Collection<String>	značky revizí ve VCS reprezentaci projektu

4.2.19. WorkItem

Třída reprezentuje pracovní položku. Rodičovská třída pro objekty tříd *Artifact* a *WorkUnit*. Jejím účelem je, aby se se všemi artefakty a tickety a hlavně jejich změnami mohlo v programu SPADe

zacházet stejně a nevznikla tak potřeba pro rozdělování tříd pro změny a konfigurace pro každou skupinu zvlášť. Neděděné atributy třídy popisuje tabulka 21.

Atribut *url* je zamýšlen jako zdroj odkazu, kterým bude možné přejít z GUI SPADe přímo na příslušnou položku v GUI ALM nástroje.

Tabulka 21 – Atributy třídy WorkItem

Atribut	Datový typ	Význam
url	String	cesta k pracovní položce na serveru instalace ALM nástroje

4.2.20. WorkItemChange

Třída reprezentuje změnu na jedné pracovní položce (ticketu/wiki stránce/artefaktu ve VCS). Každá taková změna (nebo v případě commitu do VCS více změn) vede k přechodu mezi dvěma konfiguracemi. Neděděné atributy třídy popisuje tabulka 22.

Tabulka 22 – Atributy třídy WorkItemChange

Atribut	Datový typ	Význam
description	String	popis změny
changedItem	WorkItem	změněná pracovní položka (artefakt/ticket/wiki stránka)

4.2.21. WorkUnit

Třída reprezentuje ticket v ALM nástroji, který reprezentuje úkol v projektu. Atributy jsou z většiny dané obvyklými funkcemi ticketovacích nástrojů. Ty umožňují kromě typů klasifikovat úkoly také kategoriemi. Neděděné atributy třídy popisuje tabulka 23.

Atribut *projectSegments* by měl obsahovat nejvýše jednoho zástupce každé ze tříd *Activity*, *Iteration* a *Phase*, jelikož jeden úkol nemůže být součástí více iterací ani fází (k tomu by byla vytvořena jeho kopie, tudíž jiný ticket) a aktivity by měly být co do obsažených úkolů také disjunktní množiny, jinak jsou špatně identifikovány.

Atribut *prerequisites* označuje ty artefakty a úkoly, které musí být přítomné, respektive dokončené před započetím prací na tomto úkolu.

Tabulka 23 – Atributy třídy WorkUnit

Atribut	Datový typ	Význam
number	int	číslo ticketu (může být jiné než ID)
type	WorkUnitType	typ úkolu (bug/task/apod.)
priority	WorkUnitPriority	priorita úkolu
severity	WorkUnitSeverity	závažnost úkolu
estimatedTime	double	odhadovaný čas na zpracování úkolu
spentTime	double	čas strávený zpracováním úkolu
startDate	Date	začátek práce na úkolu
dueDate	Date	mezní termín splnění úkolu
status	WorkUnitStatus	stav úkolu
progress	int	poměr zpracované části úkolu
assignee	Identity	řešitel úkolu
prerequisites	WorkItem	úkoly a artefakty nutné k započetí úkolu
category	WorkUnitCategory	kategorie úkolu
toolProjectInstance	ToolProjectInstance	reprezentace projektu v ALM nástroji, ke které úkol patří
projectSegments	Collection<ProjectSegment>	projektové segmenty (iterace, fáze a aktivita), do kterých úkol patří)

4.2.22. WorkUnitCategory

Třída reprezentuje kategorii úkolu. Ve většině ticketovacích nástrojů lze kromě samotného typu (*WorkUnitType*) klasifikovat a seskupovat i pomocí kategorií nebo obdobného konceptu. Tyto skupiny jsou pak dalším kandidátem na identifikaci objektů *Activity*. Kategorie mohou mít i vlastní popis, proto je nelze reprezentovat pouhým textovým atributem.

4.2.23. WorkUnitPriority

Třída reprezentuje prioritu úkolu. Množina implicitně definovaných priorit se liší jak mezi jednotlivými nástroji, tak mezi jejich verzemi, a v některých jsou nahrazeny jinými prostředky (značkami, labely, uživateli definovanými atributy ticketů). Navíc je priority v mnoha ticketovacích nástrojích možné vytvářet uživateli a nelze se tedy spoléhat na jejich konečnou a přesně definovanou sadu ani v jednotlivých instalacích nástrojů. Aby ale bylo možné priority úkolů porovnávat napříč projekty z různých instalací různých nástrojů, je třeba každé z nich přiřadit jednu ze tříd pevně stanovených v nástroji SPADe (viz příloha). Ta se pak přímo mapuje i na nadtřídu pro zjednodušenou klasifikaci (pouze hodnoty „*low*“, „*normal*“ a „*high*“). Neděděné atributy třídy popisuje tabulka 24.

Atribut *name* v tomto případě obsahuje původní označení priority z ALM nástroje pro možnost přesnějšího porovnání priorit úkolů ze stejného projektu, instalace nebo verze nástroje bez zásahu SPADe tříd.

Tabulka 24 – Atributy třídy WorkUnitPriority

Atribut	Datový typ	Význam
proirityClass	WorkUnitPriorityClass	třída priority úkolu
prioritySuperClass	WorkUnitPrioritySuperClass	nadtřída priority úkolu

4.2.24. WorkUnitRelation

Třída reprezentuje vztah mezi dvěma úkoly. Pro každý nalezený vztah ticketů v ALM nástroji (rodič-potomek, blokující-blokovaný, předchůdce-následník) je v nástroji SPADe vytvořen vztah popsatelný jako:

leftUnit type rightUnit,

a obdobně zrcadlový vztah, kde si úkoly vymění pozice a typ je změněn na inverzní (např. z „*blocks*“ se stane „*is blocked by*“). Místo ukládání každého ze vztahů u jednoho z úkolů (např. toho, který v dané verzi vztahu vystupuje jako levý), což by vedlo k dalšímu atributu typu kolekce u *WorkUnit*, jsou tyto vztahy ukládány s vazbou na příslušný projekt. Neděděné atributy třídy popisuje tabulka 25.

Typy vztahů a jejich různé vyjadřovací prostředky v jednotlivých ALM nástrojích jsou mapovány na přesně definovanou omezenou sadu tříd typů vztahů nástroje SPADe (viz příloha).

Tabulka 25 – Atributy třídy WorkUnitRelation

Atribut	Datový typ	Význam
type	WorkUnitRelationType	typ vztahu úkolů
leftUnit	WorkUnit	levý úkol vztahu
rightUnit	WorkUnit	pravý úkol vztahu úkolu
project	Project	projekt, k němuž vztah úkolů náleží

4.2.25. WorkUnitSeverity

Třída reprezentuje závažnost úkolu. Množina implicitně definovaných závažností se liší jak mezi jednotlivými nástroji, tak mezi jejich verzemi, a v některých je nahrazena jinými prostředky (značkami, labely, uživateli definovanými atributy ticketů). Navíc je závažnosti v mnoha ticketovacích nástrojích možné vytvářet uživateli a nelze se tedy spoléhat na jejich konečnou a přesně definovanou sadu ani v jednotlivých instalacích nástrojů. Aby ale bylo možné závažnosti úkolů porovnávat napříč projekty z různých instalací různých nástrojů, je třeba každé z nich přiřadit jednu ze tříd pevně stanovených v nástroji SPADe (viz příloha). Neděděné atributy třídy popisuje tabulka 26.

Atribut *name* v tomto případě obsahuje původní označení závažnosti z ALM nástroje pro možnost přesnějšího porovnání priorit úkolů ze stejného projektu, instalace nebo verze nástroje bez zásahu SPADe tříd.

Tabulka 26 – Atributy třídy WorkUnitSeverity

Atribut	Datový typ	Význam
severityClass	WorkUnitSeverityClass	třída závažnosti úkolu

4.2.26. WorkUnitStatus

Třída reprezentuje stav úkolu. Množina implicitně definovaných stavů se liší jak mezi jednotlivými nástroji, tak mezi jejich verzemi, a v některých je nahrazena jinými prostředky (značkami, labely, uživateli definovanými atributy ticketů). Navíc je stavy v mnoha ticketovacích nástrojích možné vytvářet uživateli a nelze se tedy spoléhat na jejich konečnou a přesně definovanou sadu ani v jednotlivých instalacích nástrojů. Aby ale bylo možné stavy úkolů porovnávat napříč projekty z různých instalací různých nástrojů, je třeba každému z nich přiřadit jednu ze tříd pevně stanovených

v nástroji SPADe (viz příloha). Ta se pak přímo mapuje i na nadtřídu pro zjednodušenou klasifikaci, které používají i ALM nástroje (pouze hodnoty „*open*“ a „*closed*“). Neděděné atributy třídy popisuje tabulka 27.

Atribut *name* v tomto případě obsahuje původní označení stavu z ALM nástroje pro možnost přesnějšího porovnání priorit úkolů ze stejného projektu, instalace nebo verze nástroje bez zásahu SPADe tříd.

Tabulka 27 – Atributy třídy WorkUnitStatus

Atribut	Datový typ	Význam
statusClass	WorkUnitStatusClass	třída stavu úkolu
statusSuperClass	WorkUnitStatusSuperClass	nadtřída stavu úkolu

4.2.27. WorkUnitType

Třída reprezentuje typ úkolu (např. *bug*, *task*, *support*, apod.). Množina implicitně definovaných typů se liší jak mezi jednotlivými nástroji, tak mezi jejich verzemi, a v některých je nahrazena jinými prostředky (značkami, labely, uživateli definovanými atributy ticketů). Navíc je typy v mnoha ticketovacích nástrojích možné vytvářet uživateli a nelze se tedy spoléhat na jejich konečnou a přesně definovanou sadu ani v jednotlivých instalacích nástrojů. Aby ale bylo možné typy úkolů porovnávat napříč projekty z různých instalací různých nástrojů, je třeba každému z nich přiřadit jednu ze tříd pevně stanovených v nástroji SPADe (viz příloha). Neděděné atributy třídy popisuje tabulka 28.

Atribut *name* v tomto případě obsahuje původní označení typu z ALM nástroje pro možnost přesnějšího porovnání priorit úkolů ze stejného projektu, instalace nebo verze nástroje bez zásahu SPADe tříd.

Tabulka 28 – Atributy třídy WorkUnitType

Atribut	Datový typ	Význam
typeClass	WorkUnitTypeClass	třída typu úkolu

4.2.28. Ostatní data

ALM nástroje obsahují větší množinu dat, než kterou je možno namapovat na prezentovaný datový model díky tomu, že ten vychází z průniku dat dolovatelných ze všech vybraných ALM nástrojů. Údaje v jednotlivých nástrojích tak mohou být značně nesourodé. Nicméně jakýkoli údaj vhodný k další analýze a nereprezentovaný ve SPADe modelu lze specifickým způsobem převést na text a připojit do atributu *description* příslušného objektu, jak demonstruje tabulka 29. Ten pak může být předmětem například analytického zpracování textu.

Tabulka 29 – Ostatní mapovatelná data

Atribut	Datový typ	Význam
description	String	doplněk popisu příslušného objektu

5. Budoucí postup prací

V nejbližší době je potřeba prozkoumat možnosti dalších API vybraných nástrojů pro vydolování většího a přesnějšího objemu dat. Zároveň je nutná implementace datových pump pro ověření funkčnosti a konzistence metamodelu.

V příštím akademickém roce (2016/2017) by měly proběhnout tři bakalářské práce, jejichž předmětem je nástroj SPADe. Cílem první je příprava testovacích dat pro datové pumpy, tzn. vytvoření fiktivního referenčního projektu ve všech vybraných nástrojích a dalších projektů soustředěných na specifika jednotlivých nástrojů. V druhé práci by měl vzniknout grafický editor umožňující vytvářet šablony procesů10 a anti-patternů ve formě XML souborů podle předem připraveného XSD schématu zachycující SPADe metamodel. Třetí práce by měla vytvořit jednoduché GUI pro zobrazení obsahu dat v datovém skladu SPADe a základních statistik (sumy, průměry, atd.) a grafů nad nimi.

Dalším nutným krokem je vytvoření souboru anti-patternů k detekci a navržení jejich vhodné reprezentace. Pak samozřejmě budou následovat další výzkumné a implementační práce týkající se aplikační a prezentační vrstvy celého nástroje SPADe.

Nástroj má do budoucna velké možnosti rozšíření. Prvním je samozřejmě rozšíření sady ALM nástrojů, které obnáší pouze implementace příslušných datových pump, a rozšíření o další dolovaná data pro zpřesnění analýz. Další možností je využití celé datové části, tzn. datového skladu plněného pumpami, pro jakoukoli jinou aplikaci zpracovávající ALM data projektů za účelem jiného výsledku, než detekce anti-patternů. Další možností rozšíření je analýza názvů a popisů všech objektů a obsahu artefaktů (souborů, wiki stránek, emailů, apod.) pomocí technik zpracování přirozeného textu (NLP – Natural Language Processing) za účelem vydolování více informací a snížení nutnosti zásahu uživatele. Ten pak může výsledky analýz pouze upravit nebo odsouhlasit.

Na základě výzkumu vedoucího k tvorbě nástroje SPADe byl také navázán kontakt s podobně zaměřenými výzkumnými pracovníky Ostbayerische Technische Hochschule (OTH) Reggensburg a Milánské polytechniky, jejichž výzkum se zabývá detekcí sociálních anti-patternů v komunikaci a spolupráci členů vývojového týmu na základě dat nejen z VCS nástrojů [44] [45], s možnou budoucí spoluprací na evropském projektu v rámci iniciativy Horizont 2020.

[10] Podobné funkce poskytují IBM Rational Method Composer (RMC) [46] [47] a Eclipse Process Framework (EPF) [48], ovšem nad jiným metamodelem procesů.

6. Závěr

Tento dokument popisuje celkový koncept nástroje SPADe, který je v současnosti vyvíjen na Katedře informatiky a výpočetní techniky na FAV ZČU, a především pak mapování jeho datového metamodelu procesů vývoje software na data dolovatelná z vybrané sady ALM nástrojů.

Hlavní funkcí SPADe je extrakce maxima agregovatelných dat z ALM nástrojů používaných při projektech vývoje software, jejich ukládání v univerzální podobě (struktuře metamodelu) a následnou detekci tzv. anti-patternů, tj. častých chyb objevujících se v projektovém řízení. Tyto anti-patterny pak má nástroj přehledně zobrazit uživateli (nejčastěji projektovému manažerovi nebo vedoucímu vývojového týmu) spolu s návrhy pro jejich odstranění nebo eliminace dopadů na projekt a porovnání s jinými již dokončenými projekty, jejichž data jsou již uložená v databázi SPADe.

Stěžejní částí tohoto dokumentu jsou kapitola 4 popisující podrobně entity a atributy datového modelu SPADe a tabulky v přílohách, které zobrazují mapování dat dolovatelných z ALM nástrojů pomocí primární sady zvolených API na data SPADe.

Prezentovaný model je stabilní a v současné době implementovaný entitními třídami a vstupním DAO rozhraním a čeká se na jeho validaci pomocí datových pump, které zatím implementovány nejsou. Mapování je v mnoha místech nekompletní a nejednoznačné, což je důsledek dosavadní nemožnosti experimentální validace datovými pumpami a faktem, že zahrnuje pouze prvotní sadu API ALM nástrojů, jedno rozhraní pro každý. Tyto nedostatky budou v dohledné době odstraněny použitím více API pro každý nástroj pro zkompletování potřebných dat a implementací datových pump pro praktickou validaci metamodelu a mapování.

Článek popisující metamodel SPADe a jeho sestavení byl přijat na konferenci SEAA, kde bude v blízké době prezentován. Koncept nástroje SPADe také tvoří potenciální základ pro spolupráci s výzkumnými pracovníky OTH Reggensburg a Milánské polytechniky a možný společný evropský projekt.

Citovaná literatura

[1] P. Pícha, „Diplomová práce - Popis softwarových procesů použitelný pro nástroje řízení projektů," ZČU, 2013.

[2] rommanasoftware.com, „Integraten Application Lifecycle Management," 2011. [Online]. Available: http://rommanasoftware.com/. [Přístup získán 14 červen 2016].

[3] ISTQB GUIDE, „What is V-model-advantages, disadvantages and when to use it?," 2016. [Online]. Available: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/. [Přístup získán 14 červen 2016].

[4] P. Kroll a P. Kruchten, The rational unified process made easy: a practitioner's guide to the RUP, Addison-Wesley Professional, 2003.

[5] S. W. Ambler a M. Holitza, Agile for Dummies, John Wiley & Sons, inc., 2012.

[6] K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004.

[7] S. W. Ambler a M. Lines, Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise, IBM Press, 2012.

[8] Apache Software Foundation, „Apache Subversion," [Online]. Available: https://subversion.apache.org/. [Přístup získán 14 červen 2016].

[9] Assembla Inc., „Assembla Keeps Code, Tasks, and Teams Happily Together," 2016. [Online]. Available: https://www.assembla.com/home. [Přístup získán 14 červen 2016].

[10] Atlassian, „Jira Software," 2016. [Online]. Available: https://www.atlassian.com/software/jira. [Přístup získán 14 červen 2016].

[11] bugzilla.org contributors, „About - Bugzilla," 27 únor 2015. [Online]. Available: https://www.bugzilla.org/about/. [Přístup získán 14 červen 2016].

[12] „git --everything-is-local," [Online]. Available: https://git-scm.com/. [Přístup získán 14 červen 2016].

[13] GitHub, Inc., „About," 2016. [Online]. Available: https://github.com/about. [Přístup získán 14 červen 2016].

[14] IBM Corporation, „Rational Team Concert," 2013. [Online]. Available: https://jazz.net/products/rational-team-concert/. [Přístup získán 14 červen 2016].

[15] J.-P. Lang, „Redmine - Overview," 2014. [Online]. Available: http://www.redmine.org/. [Přístup získán 14 červen 2016].

[16] The Eclipse Foundation, „The open source developer report: 2011 Eclipse community survey," 2011.

[17] P. Pícha a P. Brada, „ALM Tool Data Usage in Software Process Metamodeling (přijato na konferenci),“ 2016.

[18] D. Settas a I. Stamelos, „Resolving complexity and interdependence in software project management antipatterns using the dependency structure matrix,“ *Software Engineering Research, Management and Applications,* pp. 205-217, 2008.

[19] D. Settas a I. Stamelos, „Towards a dynamic ontology based software project management antipattern intelligent system,“ v *19th IEEE International Conference on Tools with Artificial Intelligence*, říjen 2007.

[20] D. Settas a I. Stamelos, „Using ontologies to represent software project management antipatterns,“ v *19th International Conference on Software Engineering & Knowledge Engineering*, 2007.

[21] D. Settas, S. Bibi, P. Sfestos a I. Stamelos, „Using bayesian belief networks to model software project management antipatterns,“ v *4th International Conference on Software Engineering Research, Management and Applications*, srpen 2006.

[22] D. Settas, S. K. Sowe a I. Stamelos, „Addressing software project management antipattern ontology similarity using semantic social networks,“ *The Knowledge Engineering Review,* sv. 24, č. 3, pp. 287-308, 2009.

[23] I. Stamelos, „Software project management anti-patterns,“ v *20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, červenec 2008.

[24] M. Khaari a R. Ramsin, „Process patterns for aspect-oriented software development,“ v *17th IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS)*, březen 2010.

[25] E. Kouroshfar, H. Y. Shahir a R. Ramsin, „Process patterns for component-based software development,“ *Component-Based Software Engineering,* pp. 54-68, 2009.

[26] M. F. Gholami, P. Jamshidi a F. Shams, „A procedure for extracting software development process patterns,“ v *4th UKSim European Symposium on Computer Modeling and Simulation (EMS)*, listopad 2010.

[27] G. Grambow, R. Oberhauser a M. Reichert, „Contextual generation of declarative workflows and their application to software engineering processes,“ *International Journal on Advances in Intelligent Systems,* sv. 4, č. 3 a 4, pp. 158-179, 2011.

[28] G. Grambow, R. Oberhauser a M. Reichert, „Contextual injection of quality measures into software engineering processes,“ *International Journal on Advances in Software,* sv. 4, č. 1 a 2, pp. 76-99, 2011.

[29] G. Grambow, R. Oberhauser a M. Reichert, „Knowledge provisioning: a context-sensitive process-oriented approach applied to software engineering environments,“ v *7th Int'l Conf. on Software Paradigm Trends (ICSOFT'12)*, červenec 2012.

[30] G. Grambow, R. Oberhauser a M. Reichert, „Towards a workflow language for software engineering," v *10th International Confonference on Software Engineering (SE'11)*, únor 2011.

[31] G. Grambow, R. Oberhauser a M. Reichert, „Towards automated process assessment in software engineering," v *7th Int'l Conf on Software Engineering Advances (ICSEA'12)*, listopad 2012.

[32] L. García-Borgoñón, M. A. Barcelona, J. A. García-García, M. Alba a M. J. Escalona, „Software process modeling languages: A systematic literature review," *Information and Software Technology,* sv. 56, pp. 103-116, únor 2014.

[33] OMG Group, „Software & Systems Process Engineering Meta-Model Specification," OMG Std., 2008.

[34] R. Ellner a et al., „eSPEM–A SPEM extension for enactable behavior modeling," v *Modelling Foundations and Applications*, červen 2010.

[35] T. Martinez-Ruiz, F. García, M. Piattini a J. Münch, „Applying AOSE concepts to model crosscutting variability in variant-rich processes," v *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, září 2011.

[36] T. Martinez-Ruiz, F. García, M. Piattini a J. Münch, „Modelling software process variability: an empirical study," *IET Software,* sv. 5, pp. 172-187, duben 2011.

[37] S. Jablonski, B. Volz a S. Dornstauder, „A meta modeling framework for domain specific process management," v *32nd Annual IEEE International Computer Software and Applications (COMPSAC'08)*, 2008.

[38] F. B. Ruy, R. A. Falbo, M. P. Barcellos a G. Guizzardi, „An ontological analysis of the ISO/IEC 24744 metamodel," v *Proc. Eighth Int. Cpnf. Formal Ontology in Information (FOIS 2014)*, září 2014.

[39] C. Gonzalez-Perez, „Supporting situational method engineering with ISO/IEC 24744 and the work product pool approach," *Situational Method Engineering: Fundamentals and Experiences,* sv. 244, pp. 7-18.

[40] Open Services for Lifecycle Collaboration, „Open Services for Lifecycle Collaboration Change Management Specification Version 3.0," 2014.

[41] Open Services for Lifecycle Collaboration, „Open Services for Lifecycle Collaboration Configuration Management 1.0, OASIS Editor's Draft," 2016.

[42] R. V. O'Connor a C. Y. Laporte, „Software project management in very small entities with ISO/IEC 29110," *Systems, Software and Services Process Improvement,* pp. 330-341, červen 2015.

[43] R. V. O'Connor a C. Y. Laporte, „Using ISO/IEC 29110 to harness process improvement in very small entities," *Systems, Software and Services Process Improvement,* pp. 225-235, červen 2011.

[44] M. Joblin, S. Apel a W. Mauerer, *Evolutionary trends of developer coordination: A network approach (nepublikováno).*

[45] M. Joblin, W. Mauerer, S. Apel, J. Siegmind a D. Riehle, „From developer networks to verified communities: a fine-grained approach," v *Proceedings of the 37th International Conference on Software Engineering*, květen 2015.

[46] P. Kroll, Introducing IBM Rational Method Composer, The Rational Edge, leden 2005.

[47] P. Haumer, „IBM Rational Method Composer: Part 1: Key concepts," IBM Report, prosinec 2005.

[48] The Eclipse Foundation, „Eclipse Process Framework (EPF)," 2016. [Online]. Available: https://eclipse.org/epf/. [Přístup získán 14 červen 2016].

Seznam zkratek

ALM	–	Application Lifecycle Management	–	Řízení životního cyklu aplikace
API	–	Application Programming Interface	–	Programovací rozhraní aplikace
DAD	–	Disciplined Agile Delivery	–	jedna z agilních metodik řízení vývoje software
DAO	–	Data Access Object	–	Objekt pro přístup k datům
DBMS	–	Data Base Management Systém	–	Systém řízení báze dat
EPF	–	Eclipse Process Framework	–	nástroj pro modelování procesů
ETL	–	Extract-Transform-Load	–	způsob migrace dat mezi databázemi
FAV	–	Fakulta aplikovaných věd		
GUI	–	Graphical User Interface	–	Grafické uživatelské rozhraní
JPA	–	Java Persistence API		
KIV	–	Katedra informatiky a výpočetní techniky		
OTH	–	Ostbayerische Technische Hochschule	–	Východobavorská vysoká škola technická
PM	–	Project Management	–	Projektové řízení
RMC	–	Rational Method Composer	–	nástroj pro modelování procesů od IBM
RTC	–	Rational Team Concert	–	ALM nástroj od IBM
RUP	–	Rational Unified Process	–	iterativní metodika od IBM založená na UP
SEAA	–	Software Engineering and Advanced Applications	–	mezinárodní vědecká konference z cyklu EUROMICRO
SVN	–	Subversion	–	VCS nástroj
SPADe	–	Software Process Anti-patterns Detector		
UP	–	Unified Process	–	iterativní metodika vývoje software
VCS	–	Version Control System	–	Systém kontroly verzí
ZČU	–	Západočeská univerzita		

Seznam obrázků

Seznam tabulek

Příloha A – Mapování výčtových datových typů

Následující tabulky mapují hodnoty výčtových datových typů používaných jako atributy některých entit v ALM nástrojích na obdobné hodnoty ve SPADe metamodelu.

Konkrétně se jedná o atributy ticketů, tj. úkolů (typ, stav, priorita a závažnost), role a vztahy mezi tickety. Pro všechny tyto koncepty jsou ve SPADe metamodelu použity třídy (*WorkUnitType*, *WorkUnitStatus*, *WorkUnitPriority*, *WorkUnitSeverity*, *Role* a *WorkUnitRelation*), které v atributu *name* uchovávají původní hodnotu převzatou přímo z patřičné reprezentace projektu v ALM nástroji (pro případ porovnávání projektů, jejichž reprezentace disponují stejnou sadou hodnot), a v atributu označovaném jako „třída“ (třídy s příponou –*Class* nebo -*Type*) hodnotu ze stabilní sady nástroje SPADe (pro porovnávání reprezentací projektů z různých nástrojů). V případě stavu a priority úkolu mají třídy ještě tzv. „nadtřídu“ (třídy s příponou -*Superclass*) určené pro klasifikaci úkolů s menší granularitou. Všechny tyto entity a atributy se týkají pouze ticketovacích nástrojů, takže Git a SVN nejsou v tabulkách uváděny.

Tabulky užívají hodnoty výchozí v instalacích nástrojů nebo nejčastěji se objevujících v jejich dokumentacích a příkladech. Více hodnot v jedné buňce tabulek znamená neurčitost mapování, pro jejíž odstranění je nutná hlubší analýza nebo zásah uživatele. Buňka zabírající více řádek tabulky znamená stejnou hodnotu pro všechny dané řádky. Pro rozhodnutí některých neurčitých nebo v nástrojích neobsažených (prázdné buňky tabulky) hodnot je možné použít i jiných atributů ticketů nebo jejich kontextu. Např. stav „*assigned*“ může být reprezentován přiřazenou hodnotu atributu *assignee* třídy *WorkUnit*. Tyto pravidla nejsou v tabulkách kvůli prostorovým restrikcím uváděna.

Některé atributy mohou mít nepřiřazenou hodnotu („*unassigned*“/“-“/“*none*“) a některé hodnoty mohou přidávat a definovat uživatelé ALM nástrojů sami („*custom*“/„*label*“). V případě uživatelský definovaných atributů, se uvažují jen takové, které odpovídají danému atributu. Jejich přiřazení k určité třídě a/nebo nadtřídě rozhoduje analýza jejich textu a/nebo uživatelský zásah.

O konkrétní implementaci tříd a nadtříd není definitivně rozhodnuto. V úvahu připadají konstrukce jazyka Java přímo určené pro výčtové datové typy (*enum*), nebo soubor textových nebo jiných konstant.

Tabulka 30 – Mapování typů úkolů

Jira	Bugzilla	Redmine	Assembla	RTC	GitHub	SPADe
						třída
bug	bug	bug	task	defect	label	bug
improvement		enhancement	story task	enhancement		enhancement
new feature		feature		story epic		feature
task		task	task	task		task
		support	support	impediment retrospective etc.		other

Tabulka 31 – Mapování stavů úkolů

<table>
<tr><th rowspan="2">Jira</th><th rowspan="2">Bugzilla</th><th rowspan="2">Redmine</th><th rowspan="2">Assembla</th><th rowspan="2">RTC</th><th rowspan="2">GitHub</th><th colspan="2">SPADe</th></tr>
<tr><th>třída</th><th>nadtřída</th></tr>
<tr><td rowspan="2">open</td><td>new
open
unconfirmed</td><td>new</td><td>new
open</td><td>new
open</td><td rowspan="6">open</td><td>new</td><td rowspan="5">open</td></tr>
<tr><td>confirmed</td><td>accepted</td><td>accepted</td><td>approved</td><td>accepted</td></tr>
<tr><td rowspan="2">in progress</td><td>assigned</td><td rowspan="3">assigned</td><td>accepted
in progress</td><td>approved
in progress</td><td>assigned</td></tr>
<tr><td rowspan="3">in progress</td><td>deferred</td><td>deferred</td><td>stalled</td></tr>
<tr><td rowspan="2">in progress</td><td>in progress</td><td>in progress</td></tr>
<tr><td>reopened</td><td>reopen</td><td>reopened</td></tr>
<tr><td>in progress</td><td rowspan="2">resolved</td><td>test
qtest</td><td>implemented</td><td rowspan="3">resolved</td></tr>
<tr><td rowspan="2">resolved</td><td rowspan="2">resolved</td><td>closed
fixed
won't fix</td><td>complete</td><td>closed</td><td>closed</td></tr>
<tr><td>verified</td><td>test
qtest</td><td>implemented</td><td>open</td><td rowspan="2">verified</td><td>open</td></tr>
<tr><td>resolved
closed</td><td>verified</td><td rowspan="2">closed</td><td rowspan="2">closed
fixed
won't fix</td><td>verified</td><td rowspan="3">closed</td><td rowspan="3">closed</td></tr>
<tr><td rowspan="2">closed</td><td rowspan="2">closed</td><td>done</td><td>done</td></tr>
<tr><td>invalid</td><td>invalid</td><td>invalid
rejected</td><td>invalid</td></tr>
</table>

V případě priorit je nutné uvažovat u uživatelem definovaných hodnot i jejich zařazení mezi hodnoty v nástroji dané jako výchozí, jelikož se jedná o uspořádanou množinu.

Tabulka 32 – Mapování priorit

<table>
<tr><th rowspan="2">Jira</th><th rowspan="2">Bugzilla</th><th rowspan="2">Redmine</th><th rowspan="2">Assembla</th><th rowspan="2">RTC</th><th rowspan="2">GitHub</th><th colspan="2">SPADe</th></tr>
<tr><th>třída</th><th>nadtřída</th></tr>
<tr><td rowspan="2">highest</td><td>immediate</td><td rowspan="2">urgent</td><td rowspan="2">highest</td><td rowspan="2">critical</td><td rowspan="6">label</td><td>critical</td><td>high</td></tr>
<tr><td>highest</td><td>highest</td><td>high</td></tr>
<tr><td>high</td><td>high</td><td>high</td><td>high</td><td>high</td><td>high</td><td>high</td></tr>
<tr><td>medium</td><td>normal</td><td>normal</td><td>normal</td><td>medium</td><td>normal</td><td>normal</td></tr>
<tr><td>low</td><td>low</td><td rowspan="2">low</td><td>low</td><td>low</td><td>low</td><td>low</td></tr>
<tr><td>lowest</td><td>lowest</td><td>lowest</td><td>unlikely</td><td>lowest</td><td>low</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td><td>unassigned</td><td>none</td><td>unassigned</td><td>unassigned</td></tr>
<tr><td>custom</td><td></td><td></td><td></td><td></td><td>label</td><td></td><td></td></tr>
</table>

Nástroj Assembla nedisponuje prostředky pro určení závažnosti úkolu.

Tabulka 33 – Mapování závažností

<table>
<tr><th rowspan="2">Jira</th><th rowspan="2">Bugzilla</th><th rowspan="2">Redmine</th><th rowspan="2">RTC</th><th rowspan="2">GitHub</th><th>SPADe</th></tr>
<tr><th>nadtřída</th></tr>
<tr><td>blocker</td><td>blocker</td><td></td><td>blocker</td><td rowspan="5">label</td><td>blocker</td></tr>
<tr><td>critical</td><td>critical</td><td>critical</td><td>critical</td><td>critical</td></tr>
<tr><td>major</td><td>major</td><td>big</td><td>major</td><td>major</td></tr>
<tr><td>minor</td><td>normal</td><td>common</td><td>normal</td><td>normal</td></tr>
<tr><td>trivial</td><td>trivial</td><td>small</td><td>minor</td><td>minor</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>unclassified</td><td>none</td><td>unassigned</td></tr>
<tr><td>custom</td><td>enhancement</td><td></td><td></td><td>label</td><td></td></tr>
</table>

Tabulka 34 – Mapování rolí

Jira	Bugzilla	Redmine	Assembla	RTC	GitHub	SPADe
						třída
administrator	custom	administrator	owner	administrator	owner	administrator
project lead		administrator manager		project manager		project manager
			owner member	team lead	owner member	team leader
developer		developer	member	team member developer	member	developer
				analyst		analyst
				architect		architect
				tester test team member test team contributor		tester
				release engineer data migration administrator project baseline administrator		operations support
user		manager reporter	watcher	product owner	owner/member	customer
		reporter		stakeholder author commenter		stakeholder
user developer				scrum master	member	mentor
administrator project lead		manager	watcher owner	stakeholder	owner/member	manager
		non member anonymous	non menber			non member
custom		custom		custom		other

Pro určení vztahů mezi úkoly používají nástroje kromě explicitních konceptů (např. *WorkItemReference* v RTC a *TicketAssociation* v Assembla) i jiné prostředky, jako speciální pole „*parent task*“ nebo „*depends on*“ (viz příloha B). Nástroj GitHub nedisponuje žádnými prostředky pro vyjádření vztahů mezi úkoly. Šedé buňky označují typy vztahů, pro něž daný nástroj nemá ekvivalent.

Tabulka 35 – Mapování vztahů mezi úkoly

Jira	Bugzilla	Redmine	Assembla	RTC	SPADe
					třída
duplicate	dupicate		duplicate	duplicates	dupicated by
			dupicate of	duplicate of	duplicates
			block	blocks	blocks
block			blocked by		blocked by
	depends on		dependent	depends on	depends on
relate			related	related mentions	relates to
		precedes			precedes
		follows			follows
				copies	copied to
				copied from	copied from
	parent	parent	parent	parent	parent
	children	children	child	children	child
			sibling		sibling
subtask			subtask		subtask
	unspecified				unspecified

Příloha B – Mapování entit a atributů

Následující tabulka popisuje mapování entit a atributů datových modelů ALM nástrojů dostupných přes zkoumaná API. V API ALM nástrojů většinou najdeme příslušnou metodu přidáním klíčového slova *get-* (v případě logické proměnné *is-*) před název atributu. Výjimkou je nástroj Assembla jejíž API využívá technologii REST. Datové typy atributů v jednotlivých nástrojích jsou z důvodu prostorových restrikcí vynechány. Drobné úpravy v mapování, hlavně v datových typech se očekávají po prozkoumání dalších API jednotlivých nástrojů, implementaci datových pump a praktickém otestování plnění datového skladu.

Řádky mapování jednotlivých entit jsou zvýrazněny orámováním a následované řádky mapování jejich atributů. Pokud se v mapování atributů neobjeví v tečkové notaci na začátku zápisu jiná třída dat ALM nástroje, patří atribut ke třídě ve zvýrazněném řádku.

Řádky obsahující očíslované možnosti označují agregaci těchto entit (nebo jejich částí) do jedné třídy metamodelu SPADe. V takovém případě je nutné v implementaci datových pump vytvářet instance příslušné třídy SPADe pro každou z popsaných entit dat ALM nástroje. Atributy jsou číslovány podle příslušnosti ke stejně očíslované třídě. Pokud se v buňce objevují dva záznamy se stejným očíslováním, jedná se o alternativy, které je nutné mapovat obě. V některých buňkách se může objevit tečková notace nezačínající slovem označujícím třídu (tzn. nezačínající velkým písmenem). To značí nutnost užití zřetězení metod k zpřístupnění patřičné hodnoty.

Žlutě označené buňky obsahují klauzuli „*where*“ nebo „*if*“. To obvykle značí, že vazba mezi entitami vyjádřená daným atributem SPADe metamodelu je v datech ALM nástroje koncipována opačným směrem. Např. iterace má ve SPADe odkaz na projekt, ale v ALM má projekt kolekci iterací. Pravidlo za klauzulí pak popisuje vyhledání příslušné hodnoty. Pseudometoda „*contains()*“ označuje přítomnost prvku v kolekci. Klíčové slovo „*this*“ označuje samotnou instanci třídy v ohraničeném řádku (popř. její ID, jméno, atd.). Zápis „*[i]*” nebo „*(i)*“ značí jeden, v tuto chvíli ne přesně specifikovaný, prvek pole nebo kolekce.

Notace „//“ reprezentuje dosud nevyřešenou nejednoznačnost v mapování, k jejímuž rozhodnutí je třeba experimentální otestování implementovaných datových pump.

Atributy spojené pomocí symbolu „*+*“ ukazují na spojení jejich hodnot do jedné hodnoty jednoho objektu v metamodelu SPADe. Většinou se týká spojení kolekcí, řetězců (často u atributu *description*) nebo faktu, že tyto atributy přisívají k analýze výsledné hodnoty atributu SPADe objektu, i když v ní nemusí být přímo obsažené.

Buňky mapování rozpínající se napříč sloupci více nebo všech ALM nástrojů jsou platné pro všechny tyto nástroje a často ve zkráceném zápisu slovně v angličtině popisují analytické možnosti mapování příslušných atributů nebo fakt, že vyplnění jejich hodnot je předmětem hlubší analýzy nebo uživatelského zásahu (často se týká červeně označených entit na obrázku 2). tyto buňky jsou podbarveny červeně, pokud nebyly dříve podbarveny žlutě (viz výše).

Některé atributy *WorkUnit* mohou být nahrazeny koncepty jako *Label*, *Tag* nebo *Custom Field*. Ty, které odpovídají (jménem nebo hodnotou) některému atributu jsou na něj mapovány (v tabulce označeno „*(where/if fits)*“), ostatní se přidávají v textové podobě do *description* daného objektu (viz poseldní sekce tabulky).

Tabulka 36 – Mapování atributů mezi SPADe a ALM nástroji

SPADe	Jira	Bugzilla	Redmine	Assembla	RTC	GitHub	Git	SVN
Activity	sets of related tasks or WorkUnits of the same category inside the same iteration or phase							
externalId	unused							
name	user input							
descritption								
startDate	min WorkUnit.startDate							
endDate	max WorkUnit.endDate							
Artifact	Attachment	Attachment	1) Attachment 2) WikiPage	1) Document 2) Project	IAttachment	GHAsset	RevBlob	SvnCommitItem
externalId	id	id	1) id	1) id	id	id	id	
name	fileName	fileName	1) token || fileName 2) title	1) filename || name, 2) wiki_name	name	name	name	path
descritption	properties	description	1) description 2) version	1) description				flags
created	created	date	1) createdOn 2) createdOn	1) created_at	creationDate			
author	authorObject		1) author	1) created_by	creator			
url		uri	1) contentUrl	1) url		url		url
artifactClass	„File“	„File“	1) „File“ 2) „Wikipage“					kind
mimeType	mimeType	type	1) contentType	1) content_type				
size	fileSize		1) fileSize	1) filesize				
Branch						GHBranch	Ref	
externalId						sha1	objectId	
name						name	name	"trunk" or after "/branches/" in path
isMain						if GHRepository.defaultBranch = this	Repository.getRef("refs/heads/master")	if "trunk" in path
Competency	user input							
externalId	unused							
name	user input							
descritption								
Configuration	1) ChangeHistory 2) Worklog 3) Comment 4) Issue	Comment	1) Changeset 2) Journal 3) TimeEntry 4) WikiPage	1) TicketComment 2) Document 3) Ticket	1) IChangeset 2) IComment 3) IAttachment 4) ITimeSheetEntry 5) IWorkItem	1) GHIssueComment 2) GHCommit	RevCommit	1) SVNRevision 2) SvnCommit 3) SVNCommitInfo
externalId	1) id 2) id 3) id	id	1) revision, 2) id 3) id	1) id	1) id	1) id 2) sha1	id	1) id
name							name	1) name
description	1) comment 2) comment 3) body	theText	1) comments, 2) comment, 3) notes	1) comment	1) comment 2) HTMLContent	1) body, 2) shortinfo + comments + status	fullMessage	2) commitMessage
number								1) number
created	1) timePerformed 2) created 2) updated 3) created 3) updated 4) resolutionDate	creationTimestamp updateTimestamp	1) commitedOn 2) createdOn 3) createdOn 3) updatedOn 4) updatedOn	1) created_on 1) updated_at 2) updated_at 3) completed_date	1) lastChangeDate 2) creationDate 3) modified 4) creationDate, 5) resolutionDate	1) createdAt 1) updatedAt 2) lastStatusCreatedAt	authorIdent.when || committerIdent.when	2) date
author	1) authorObject 2) authorObject 2) updateAuthorObject 3) authorApplicationUser 3)	Author updateAuthor	1) user 2) user 3) userId	1) user_id, 2) updated_by	1) author 2) creator 3) modifiedBy 4) creator 5) resolver	1) user 2) commiter || author	authorIdent || commiterIdent	3) author

	updateAuthorApplicationUser							
changes	1) changeItemBeans	id	2) details	1) ticket_changes	1) changes	2) files	tree	2) commitItems.items
isRevision			„true“ if 1)		„true“ if 1)	„true“ if 2)	„true“	„true“
artifacts	all created and not deleted untill this.created							
workUnits								
branch							if type = branch	"trunk" or after "/branches/" in path
tags						GHTag.name where GHTag.commit == this	if type = tag where RevTag.object = this	if "tags" in path
Criterion	user input							
externalId	unused							
name	user input							
descritption								
DevelopmentProgram	ProjectCategory		Project (if has children && not parent)	1) Space (if has children && not parent) 2) Group (if type = space)	IProjectArea (if has children && not parent)	GHRepository (if has children && not parent, if source of any repo)		
externalId	id		id	1) id 2) id		id		
name	name		name	1) name, 2) name	name	name		
descritption	description		description	1) description, 2) portfolio_id	description	description		
startDate	min Project.startDate	min Project.startDate	createdOn || min Project.startDate	created_at || min Project.starDate	Modified || Project.startDate	createdAt || Project.startDate		
endDate	max from all dates or user input							
personnel	sum of all member projects personell							
project			parentId	parent_id	projectLinks	parent		
type	user input							
projects	where Project.projectCategory = this			2) items				
Identity	ApplicationUser	User	User	User	IContributor	GHUser	PersonIdent	
externalId	id	id	id	id	userId	id		
name	displayName	name	login	login	name	login		
descritption	key + name	realName	fullName	Name + im + im2	details	name + organizations + company	name	
roles	ProjectRoleActor.name (where users.contains(this)) + Project.projectLead		Memebership.roles (where user = this)		IProjectArea.administrators + ITeamArea.roleAssignments			
email	emailAddress	email	mail	email	emailAddress	email	emailAddress	
IdentityGroup	GroupBean		Group	Group (if type = user)	ITeamArea	GHTeam		
externalId	self		id	id		id		
name	name		name	name	name	name		
descritption				portfolio_id	descritption + teamData	organization		
members	GroupMembership.users	where BugzillaUser.team = this	where User.groups.contains(this)	items	members	members + where GHUser.organizations.contain(this) + where GHUser.company = this		
Iteration	Version	BugzillaVersion	Version	Milestone	IIteration	GHMilestone		
externalID	id	id	id	id	id	id		
name	name	name	name	title	name	title		
descritption	description + archived + released + sequence		Description + status	Description + is_completed + releas_notes	Description + archived + iterrationType + parent + children	description + state + number		
startDate	startDate	min Issue.creationTimestamp	min Issue.startDate	start_date	startDate	min GHIssue.createdAt		

endDate	releaseDate	releaseDate	dueDate	due_date	endDate	dueOn		
project	project	project	project	space_id	developmentLine	owner		
created	min Issue.created	min Issue.creationTimestamp	createdOn	created_at	modified	createdAt		
activities	Activity inside this							
configuration	closest to endDate or user input							
Milestone	user input							
externalID	unused							
name	user input							
descritption								
criteria								
Person	agregated from Identities							
externalID	unused							
name	select from identities.description							
identities	similar Identity.name or Identity.description or user input							
competencies	user input							
Phase	Version	BugzillaVersion	Version	Milestone	IIteration	GHMilestone		
externalID	id	id	id	id	id	id		
name	name	name	name	title	name	title		
descritption	description + archived + released + sequence		Description + status	Description + is_completed + releas_notes	Description + archived + iterrationType + parent + children	description + state + number		
startDate	startDate	min Issue.creationTimestamp	min Issue.startDate	start_date	startDate	min GHIssue.createdAt		
endDate	releaseDate	releaseDate	dueDate	due_date	endDate	dueOn		
project	project	project	project	space_id	developmentLine	owner		
created	min Issue.created	min Issue.creationTimestamp	createdOn	created_at	modified	createdAt		
activities	Activity inside this							
configuration	closest to endDate or user input							
milestone	user input							
Project	Project	BugzillaProject	Project	Space	IProjectArea	GHRepository		
externalID	unused							
name	select from ToolProjectInstances names							
descritption	sum of ToolProjectInstances description							
startDate	min Issue.created	min Issue.creationTimestamp	createdOn || min Issue.startDate	created_at || min Ticket.created_on	Modified || WorkItem.creationDate	createdAt || GHIssue.createdAt		
endDate	max from all dates or user input							
personnel	where Person.identities(i) = ToolProjectInstance.identities(i) + where ToolProjectInstance.project = this							
project			parentId	parent_id	projectLinks	parent		
Role	ProjectRoleActor		Role	UserRole	IRole2			
externalID	id		id	id	id			
name	name + Project.projectLead		name	role	name			
descritption	descriptor			title + status	description + label			
roleClass	see table 34							
Release	1) ChangeHistory 2) Worklog 3) Comment 4) Issue	Comment	1) Changeset 2) Journal 3) TimeEntry 4) WikiPage	1) TicketComment 2) Document 3) Ticket	1) IChangeset 2) IComment 3) IAttachment 4) ITimeSheetEntry 5) IWorkItem	1) GHIssueComment 2) GHCommit 3) GHRelease	RevCommit	1) SVNRevision 2) SvnCommit 3) SVNCommitInfo
externalID	1) id 2) id 3) id	id	1) revision, 2) id 3) id	1) id	1) id	1) id 2) sha1 3) id	id	1) id
name						3) name	name	1) name

description	1) comment 2) comment 3) body	theText	1) comments, 2) comment, 3) notes	1) comment	1) comment 2) HTMLContent	1) body 2) shortinfo + comments + status 3) body + tagName + draft	fullMessage	2) commitMessage
number								1) number
created	1) timePerformed 2) created 2) updated 3) created 3) updated 4) resolutionDate	creationTimestamp updateTimestamp	1) commitedOn 2) createdOn 3) createdOn 3) updatedOn 4) updatedOn	1) created_on 1) updated_at 2) updated_at 3) completed_date	1) lastChangeDate 2) creationDate 3) modified 4) creationDate, 5) resolutionDate	1) createdAt 1) updatedAt 2) lastStatusCreatedAt 3) publishedAt || createdAt 3) updatedAt	authorIdent.when || committerIdent.when	2) date
author	1) authorObject 2) authorObject 2) updateAuthorObject 3) authorApplicationUser 3) updateAuthorApplicationUser	Author updateAuthor	1) user 2) user 3) userId	1) user_id, 2) updated_by	1) author 2) creator 3) modifiedBy 4) creator 5) resolver	1) user 2) commiter || author	authorIdent || committerIdent	3) author
changes	1) changeItemBeans	id	2) details	1) ticket_changes	1) changes	2) files	tree	2) commitItems.items
isRevision			„true“ if 1)		„true“ if 1)	„true“ if 2) or 3)	„true“	„true“
artifacts	all created and not deleted untill this.created					3) assets	all created and not deleted untill this.created	
workUnits	all created and not deleted untill this.created							
branch							if type = branch	"trunk" or after "/branches/" in path
tags						1), 2) GHTag.name where GHTag.commit == this 3) tagName	If type = tag where RevTag.object = this	if "trunk" in path
ToolInstance	from main user input							
externalID	unused							
tool version	from main user input							
ToolProjectInstance	Project	BugzillaProject	Project	Space	IProjectArea	GHRepository	Repository	
externalID	id	id	id	id		id		
name	name	name	name	name	name	name		
descritption	description + key + originalKey + projectTypeKey	description	description + identifier	description	description	fullName + description		
priorities					where IPriority.projectArea = this	labels (where fits)		
severities					where ISeverity.projectArea = this	labels (where fits)		
workUnitTypes	issueTypes				where IWorkItemType.projectArea = this	labels (where fits)		
statuses						labels (where fits)		
categories	components + where ProjectComponent.projectId = this	components + where BugzillaComponent.project = this	where IssueCategory.project = this		where ICategory.projectArea = this	labels (where fits)		
identities			Membership.user + where User.memberships.contains(this)		members	collaborators		
groups				team_permissions	teamAreas	teams		
roles					rolesAssignments			
configurations			where TimeEntry.projectId = this		where ITimeSheetEntry.projectArea = this	commits	RevWalk(Repository).iterator	
branches						Branches + where GHBranch.owner = this	branchList	
tags						tags	tags	
toolInstance url project	from main user input							

WorkItemChange	1) ChangItemBean 2) Worklog 3) ChangeHistory 4) Comment 5) Resolution 6) Issue	1) BugzillaResolution 2) Comment 3) Issue	1) JournalDetail 2) TimeEntry	1) TicketComment 2) Document 3) Ticket	1) IChange 2) ITimeSheetEntry 3) IChangeset 4) IResolution 5) IWorkItem	1) GHIssueComment 2) GHCommit		
externalID	5) id				4) identifier2	1) id		
description	1) (all attributes) 2) timeSpent 2) startDate 5) name + description	1) name + cancelled + duplicate + resolved	1) (all attributes) 2) spentOn + hours + activityName	1) "comment added" 2), 3) "modified"	1) (all attributes) 2) startDate + timeSpent + workType 4) "resolved"	1) "comment added"		
changedItem	3) issue 4) issue 5) (if has Resolution)	2) issueId 3) (if has Resolution)	2) issueId	1) ticket_id 2) 3) (if has completed_date)	3) component 5) (if has Resolution)	1) parent 2) files		
WorkUnit	Issue	Issue	Issue	Ticket	IWorkItem	GHIssue		
externalID	id	id	id	id	id	id		
name	summary	summary	subject	summary	htmlSummary	title		
descritption	description + key + customFieldValue + labels	description + customFieldNames + classification	description + customFields	description + custom_fileds + tags	htmlDescription + customAttributes + tags2	body + labels		
created	created	creationTimestamp	createdOn	created_on	creationDate	createdAt		
author	creator || reporter	reporter	author	reporter_id	creator	user		
url		uri				htmlUrl || url		
number	number			number		number		
type	issueType	type	tracker	is_story	workItemType	labels (where fits)		
priority	priority	priority	priorityId + priorityText	priority	priority	labels (where fits)		
severity	customFieldValue (where name = severity) || where Label.issue = this (if fits)	severity	customFields (where name = severity)	custom_fields + tags + importance + story_importance	severity	labels (where fits)		
estimatedTime	originalEstimate || estimate		estimatedHours	total_working_hours || estimate || total_estimate || working_hours	duration			
spentTime	timeSpent		spentHours	total_invest_hours				
startDate	created	creationTimestamp	startDate	created_on	creationDate	createdAt		
dueDate	dueDate		dueDate		dueDate	milestone.dueOn		
status	status + resolution	status + cancelled + closed + inProgress + open + resolved + resolution	statusId + statusName	status + state	state2 + resolution2	state		
progress			doneRatio					
assignee	assignee	assignee	assignee	assigned_to_id	Resolver || owner	assignee		
prerequisites	attachments || where Attachment.issue = this	attachments || where Attachment.issueId = this	(relations.type = depends || blocks) + attachments	where Document.ticket_id = this	WorkItemReference.target (where source = this && name = dependsOn || blockedBy)			
category	components	components + classification	category	component_id	category	labels (where fit)		
toolProjectInstance	projectObject	project	project	space_id	projectArea	repository || if GHRepository.issues.contains(this)		
projectSegments	fixVersions + affectedVersions	fixVersions + plannedVerions + affectedVersions	targetVersion	milestone_id	target	milestone		
WorkUnitCategory	ProjectComponent	1) BugzillaComponent 2) Classification	IssueCategory		ICategory	GHLabel		
externalId	id	1) id 2) id	id		categoryId			
name	name	1) name 2) name	name		name	name		

description	description	1) description 2) description			HTMLDescription + unassigned	color (if fits)		
WorkUnitPriority	Priority	BugzillaPriority	IssuePriority	1) CustomField (if type = priority) 2) Tag (if fits)	IPriority	GHLabel (if fits)		
externalID	id		id	1) id 2) id	identifier2			
name	name	name	name	1) title 2) name	name	name		
descritption	description			1) type 2) state		color		
priorityClass prioritySuperclass	see table 32							
WorkUnitRelation	1) IssueLink 2) Issue 3) Issue	1) IssueLink 2) Issue 3) Issue	1) IssueRelation 2) Issue	TicketAssociation	WorkItemReference			
externalID	1) id		1) id	id				
name	1) issueLinkType.name 2) if is parent 3) if is subtask	linkTypeName	1) type	relationship	name			
type	see table 35							
leftUnit	1) sourceObject 2) parentObject = this 3) subTaskObjects.contains(this)	1) where Issue.links.contains(this) 2) where Issue.parentId = this 3) where Issue.children.contains(this)	1) issueId 2) this	ticket2_id	source			
rightUnit	1) destinationObject 2) this 3) this	1) issueId 2) this 3) this	1) issueToId 2) parentId	ticket1_id	target			
description	1) issueLinkType + inward + outward + subTaskLinkType 2) parent 3) subtask	1) linktype + linkTypeDescription + inward 2) „parent“ 3) „child“	1) type 2) „child“					
WorkUnitSeverity	Label (if fits)	BugzillaSeverity	CustomField (if fits)	1) CustomField (if type = severity) 2) Tag (if fits)	ISeverity	GHLabel (if fits)		
externalID	id		id	1) id 2) id	identifier2			
name	label	name	value	1) title 2) name	name	name		
descritption	customFieldId		name	1) type 2) state		color		
severityClass	see table 33							
WorkUnitStatus	1) Status 2) Resolution	1) BugzillaResolution 2) BugzillaStatus	IssueStatus	TicketStatus		GHIsseuState		
externalID	1) id 2) id		id	id				
name	1) name 2) name	1) name, 2) name	name	name		name		
descritption	1) description 2) description	1) cancelled + dupicate + resolved 2) cancelled + duplicate +		list_order				

		resolved + closed + open						
statusClass	1) simpleStatus + see table 31	+ see table 31						
statusSuperclass	2) statusCategory + see table 31	see table 31	closed + see table 31	state + see table 31	+ see table 31			
WorkUnitType	IssueType	BugzillaIssueType	Tracker	1) CustomField (if type = severity) 2) Tag (if fits)	IWorkItemType	GHLabel (if fits)		
externalID	id		id	1) id 2) id	identifier			
name	name	name	name	1) title 2) name	displayName	name		
descritption	description + subTask			1) type 2) state	aliases	color		
typeClass	see table 30				category + see table 30	+ see table 30		
any	Label			Tag		GHLabel		
description+	id			id				
	label			name		name		
	customFieldId			state		color		