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

Diplomová práce

Identifikace a modelování špatných praktik v projektovém řízení

Plzeň 2019 Patrik Bezděk

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 25. června 2019

Patrik Bezděk

Poděkování

Rád bych poděkoval Ing. Petru Píchovi za cenné rady, připomínky a čas, který mi věnoval při vedení diplomové práce.

Abstract

The goal of this thesis is to identify process errors in software development and create their models using SPADe. The SPADe tool collects data from ALM tools and user is then able to search for (anti-)patterns. For the purpose of addressing the goal of the thesis, an analysis and selection of a set of procedural errors and projects for verification was carried out. Furthermore, an analysis of the tool and its data model was performed, and an extension was created to create SQL queries over the tool's data warehouse and thus model process errors. Subsequently, a selected group of process errors was detected using the created extension in the selected project data, which was compared with the separately performed manual check for the presence of process errors in the ALM project data. Detection was successful in 89,8% of cases compared to manual control.

Abstrakt

Cílem této diplomové práce je identifikace procesních chyb v softwarovém vývoji a vytvoření jejich modelů pomocí nástroje SPADe. SPADe slouží ke sběru dat z ALM nástrojů a následnému hledání (anti-)patternů v projektových datech. Za účelem vypracování byla provedena analýza a výběr sady procesních chyb a projektů pro ověření. Dále pak byla provedena analýza nástroje a jeho datového modelu a bylo vytvořeno rozšíření, pomocí kterého lze vytvářet SQL dotazy nad datovým skladem nástroje a tím modelovat procesní chyby. Následně byla provedena detekce vybrané skupiny procesních chyb pomocí vytvořeného rozšíření v datech vybraných projektů, která byla porovnána s odděleně provedenou manuální kontrolou přítomnosti procesních chyb v ALM datech projektů. Detekce byla úspěšná v 89,8% případů oproti manuální kontrole.

Obsah

1 Úvod

Jakákoliv spolupráce většího počtu lidí je náchylná na chyby právě kvůli existenci lidského faktoru. Vývoj software není výjimkou. Členové týmů musí denně činit rozhodnutí, a tak dochází k opakujícím se chybám jak při vývoji, tak v projektovém řízení. Tyto chyby jsou známy pod pojmem „bad practices“ nebo anti-patterny. Obor softwarového inženýrství se neustále vyvíjí, ale problémy v projektovém řízení zůstávají i přes to, že existuje celá řada podkladů poskytujících informace k jejich identifikaci, odstranění a předcházení. Neexistuje však nástroj, který by při řešení problémů pomohl, a proto vzniká na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity nástroj Software Process Anti-pattern Detector (SPADe), jehož konečným účelem bude pomoci managementu řešit problémy v projektovém řízení díky shromažďování dat z Application Lifecycle Management nástrojů a možnosti analýzy těchto dat s cílem identifikovat anti-patterny a poskytnout informace k jejich vyřešení.

Cílem této diplomové práce je zanalyzovat a detekovat procesní chyby (bad practices a anti-patterny) ve vybraných open source projektech a rozšířit nástroj SPADe tak, aby byl schopen detekovat co největší množinu nalezených (anti-)patternů. Za tímto účelem byla provedena analýza nástroje SPADe, jeho grafického uživatelského rozhraní a datového modelu a porovnání dat z ALM (Application Lifetime Management) nástrojů s výstupy implementovaného rozšíření nástroje SPADe. Nástroj byl použit na detekování (anti-)patternů na projektových datech a výsledky jsou vyhodnoceny v závěrečné části této práce.

První část práce, konkrétně kapitola 2, poskytuje teoretický úvod do kontextu projektového řízení a vývoje software. Následuje prozkoumání problematiky patternů a anti-patternů v kapitole 3 a popis teorie disciplíny Application Lifecycle Management v kapitole 4. Aby se mohl nástroj SPADe rozšířit, byl v kapitole 5 zanalyzován současný stav nástroje a sepsány požadavky na rozšíření a v kapitole 6 popsán návrh a jeho implementace. Nakonec byl proveden experiment popsaný v kapitole 7, kde se testovalo rozšíření na projektových datech a byly vyhodnoceny výsledky oproti manuální kontrole.

2 Vývoj software

Vývoj software už není jen o psaní kódu programu, ale zahrnuje spoustu dalších činností. Součástí vývoje je sběr požadavků, návrhy, plánování, implementace, nasazení, podpora a další. Není toho málo, a proto většina projektů, které spadají pod nějakou organizaci, je vyvíjeno týmově a nikoli jednotlivci. V tu chvíli je velmi důležité zapojit do procesu i projektové řízení, které se zaobírá plánováním projektu, jeho vykonáním a kontrolou, komunikací se zúčastněnými stranami a v neposlední řadě řízením lidí a podporou členů týmu. Hlavním úkolem projektového řízení je zařídit, aby byly splněny cíle projektu.

Tato kapitola poskytuje teoretický základ o vývoji software s ohledem na metodiky a praktiky, které se využívají v projektovém řízení, a definuje termíny podstatné pro kontext práce.

2.1 Projekt

Projekt lze definovat jako posloupnost úkolů, které je třeba vykonat, aby bylo dosaženo konkrétního výsledku. Může být definován také jako soubor vstupů a výstupu, které jsou potřeba k dosažení určitého cíle. Projekt je dočasný, takže má jasně stanovený začátek a konec a výsledek projektu by měl být měřitelný, aby se dal porovnat s požadavky. [17]

Projekt [19] se skládá z několika po sobě jdoucích částí. V závislosti na velikosti projektu mohou být nějaké části vypuštěny a nebo sloučeny s jinou částí. Velikosti projektů se mohou lišit s ohledem na cenu, délku trvání a dostupné zdroje. Obecně se projekt skládá z těchto částí:

- **Zahájení** – popis projektu, sběr požadavků a vypracování první dokumentace
- **Plánování** – naplánování rozsahu, trvání, ceny, zdrojů atd.
- **Vykonávání** – vypracování a dokončení projektu podle plánu
- **Kontrola** – společně s částí vykonávání; měření postupu projektu a kontrola výstupů proti požadavkům
- **Uzavření** – dodání produktu a ukončení kontraktu

2.2 Proces

Softwarový proces [29] je reprezentován jako soubor souvisejících činností (aktivit), které vedou k návrhu a tvorbě softwarového produktu. Činnosti mohou zahrnovat vývoj nového softwaru, nebo úpravu již existujícího. Žádný ideální proces neexistuje a mnoho organizací si vyvinulo svůj vlastní přístup k vývoji softwaru. Proces by měl co nejvíce využít schopností lidí v organizaci.

Existují některé činnosti, které jsou společné pro všechny softwarové procesy:

- **Specifikace softwaru** – definování funkčnosti a omezení provozu softwaru
- **Návrh a implementace softwaru** – tvorba softwaru podle specifikace
- **Ověřování softwaru** – kontrola, zda má software všechny funkce, které byly definovány
- **Evoluce softwaru** – zpracování měnících se požadavků

Projekt lze vidět jako nejvíce detailní vykonávanou instanci procesu. Nemusí použít celý proces, ale lze ho upravit podle situace a potřeb.

Existuje několik základních konceptů, které lze považovat za základní stavební kameny jakéhokoli procesu vývoje software. Ty slouží k vyjádření základních spojitostí mezi členy týmu, jejich činnostmi, vstupy a výstupy těchto činností. V následujících pododdílech je stručný popis těchto konceptů.

2.2.1 Aktivita

Aktivity jsou hlavní jednotky práce, které je třeba dokončit při dosahování cílů procesu. Každá aktivita má počáteční a koncové datum a zahrnuje soubor úkolů, které mají být dokončeny a při kterých jsou spotřebovávány zdroje. [3]

2.2.2 Úkol

Úkol je definované zadání práce se specifickým cílem, vstupy, výstupy a místem v procesu. Obvykle je přiřazen jedné osobě, ale může být prováděn i více lidmi (např. popsat požadavky, konkrétní funkčnost testu, provést schůzku týmu atd.).

2.2.3 Role

Roli lze definovat jako soubor kompetencí a odpovědností, které se pojí s určitou aktivitou příslušící jedné nebo více osobám. Jednu roli může zastávat v týmu více lidí a jeden člen může v týmu zastávat více rolí.

V dnešní době se už tým, který pracuje na nějakém projektu, neskládá jen z vývojářů. Projektové týmy se skládají z vývojářů, analytiků, testerů, grafických návrhářů a lidí na psaní technických dokumentů. [15] Pokud se z tohoto seznamu odeberou vývojáři, dá se říci, že práce ostatních nemusí být na první pohled vidět. To ale neznamená, že jsou v projektu nedůležití a postradatelní. V dnešní době už jen vývojáři nestačí.

Aby byl projekt úspěšný, je potřeba pečlivého plánování, talentovaného týmu a hlavně spolupráce mezi jednotlivými členy týmu. Projekt může spolehlivě postupovat kupředu, pokud jsou v týmu obsazeny všechny klíčové role. [24]

2.2.4 Artefakt

V softwarovém inženýrství se za artefakt považují „věci“, které představují vstupy nebo výstupy od lidí nebo procesů při vývoji. Jedná se například o dokumentace, datové modely, workflow diagramy, testovací scénáře a plány, skripty, ale i samotný zdrojový kód. Při vývoji se často specifické artefakty sdílejí, aby si je členové týmu mohli prohlédnout a případně sdílet dále.

2.3 Praktiky v SW vývoji

Praktika je v podstatě způsob, jak vykonat nějaký proces nebo úkol – je to část procesu. Mohou být jak obecné, tak hodně specifické. Praktiky mohou být vyjmuty z procesu a vloženy do jiného procesu a slouží pak jako stavební kameny daného procesu. Proces lze vidět jako soubor praktik. [18]

V softwarovém vývoji nebo projektovém managementu se lze setkat s pojmem „best practices“. Jedná se o techniku, metodu nebo proces, který je při dosahovaní požadovaného cíle účinnější a efektivnější než jiné techniky, metody a procesy. Best practice jsou založeny na zkušenostech a používají se k popisu procesu a následného standardního postupu. Vzhledem k tomu, že projekty jsou vnímány jako jedinečné, nedá se očekávat, že jeden soubor procesů a metod zaručí úspěch každého projektu. [10] Praktik je mnoho a mohou se značně lišit. Mezi praktiky lze řadit například používání ALM nástrojů při vývoji, daily standup a iterační retrospektivy při používání Scrum

metodiky, komunikace se zákazníkem a získání zpětné vazby nebo také formáty psaných dokumentů atd.

2.4 Metodiky vývoje software

Metodika projektového řízení je soubor principů a postupů pro řízení projektu. Jedná se o způsob, jak daný projekt vést. Volba metodiky definuje, jak budou lidé v týmu pracovat a komunikovat.

Jedno z prvních rozhodnutí projektového manažera je právě výběr metodiky. Jakou metodiku zvolit závisí na typu a rozsahu projektu a také složení týmu. Výběr má dopad na to, jak a kdy budou prováděny jednotlivé procesy v projektu. Metodiky mají různé výhody a nevýhody pro různé typy projektů. [4]

Metodiku je možné vidět jako sadu best practices. Může popisovat, jak samotnou metodiku používat, co za software využívat, jak kontrolovat procesy, jaké vytvořit výstupy atd. Součástí metodiky je i procesní model, který je většinou postavený právě na best practices. Jedná se o abstraktní reprezentaci procesu. Proces může procesní model adaptovat celý, ale reálně se většinou upraví a zasadí do kontextu a potřeb organizace. Proces se tak dá popsat jako sada praktik (ne nutně jen best practices), ze kterých je složen. Dále jsou uvedeny příklady několika metodik s jejich popisem a několika praktikami specifickými pro dané metodiky.

2.4.1 Vodopád

Jedná se o jednoduchý a lehce pochopitelný model. Skládá se z několika po sobě jdoucích částí (viz obr. 2.1), které jdou sekvenčně za sebou a nepřekrývají se. Části na sebe navazují a výstup z jedné části je vstupem do části následující. Vodopádový model spadá do sekvenčních metodik. Obecně se model skládá ze šesti následujících částí:

- **Analýza a sběr požadavků** – všechny požadavky zákazníka jsou zapsány do dokumentu specifikace požadavků
- **Návrh systému** – podle specifikace požadavků se připraví návrh systému a definuje se architektura
- **Implementace** – návrh je převeden do kódu ve vybraném programovacím jazyce a jednotlivé části kódu se testují
- **Integrace a testování** – jednotlivé části kódu se spojí do jednoho celku a otestuje se celé fungování systému

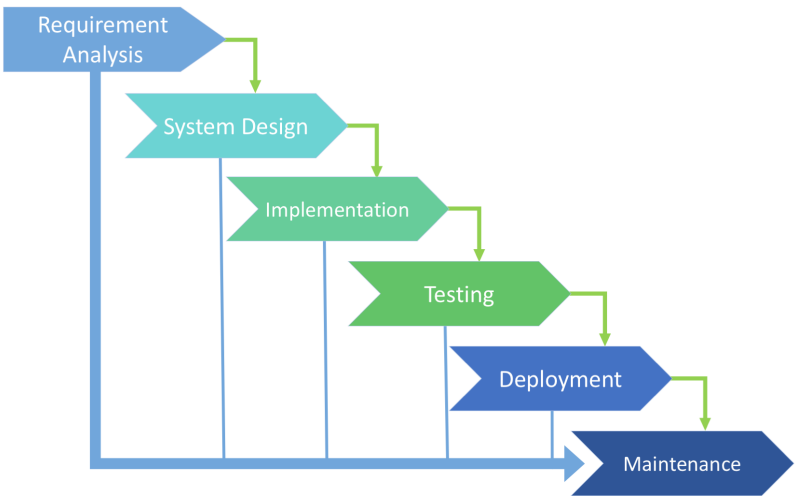
- **Nasazení** – produkt je nasazen k zákazníkovi nebo uveden na trh
- **Údržba** – opravují se chyby nebo se systém upravuje a vylepšuje podle nových požadavků

Výhodou vodopádového modelu je jasné definování jednotlivých částí. Části jsou vypracovávány postupně, mají jasně určený začátek a konec a dobře zdokumentované výstupy. Jednou z nevýhod je, že zákazník neuvidí žádnou ukázku produktu, dokud není projekt téměř u konce, a není tedy možnost zapracovat změny požadavků nebo upravit rozsah projektu. Vodopádový model se hodí na menší a kratší projekty s jasně specifikovanými požadavky.

Praktiky

Vodopádový model obsahuje několik klíčových praktik. Jsou jimi například:

- Specifikace požadavků – důkladná analýza a podrobný sběr požadavků pro vytvoření dokumentace
- Fáze projektu – projekt je předem jasně rozdělen na fáze s definovanou náplní práce
- Sekvenční průběh – nepřekrývající se části navazující na sebe, výstupy z jedné slouží jako vstupy do následující



Obrázek 2.1: Schéma vodopádového modelu

2.4.2 Unified Process

Unified Process [21] je model iterativní a inkrementální. Je rozdělen do čtyř částí: Inception, Elaboration, Construction a Transition (viz obr. 2.2). Každá část se skládá z několika časově ohraničených iterací a je zakončená milníkem, který má předem definováno, jakými výstupy ho lze dosáhnout. Výsledkem každé iterace je přírůstek do systému, který obsahuje nové nebo upravené části produktu.

- *Inception*: V této části se určí základní struktura projektu. Určí se rozsah a náklady na projekt, odhadne se cena, zanalyzují se rizika, naplánují se iterace, navrhnou se možné architektury atd. Na konci této části se vyhodnotí milník *Lifecycle Objectives*. Zkoumá se, jestli se většina zúčastněných stran dohodla na rozsahu projektu, jestli rozumí požadavkům, a rozhodne se, jestli se na projektu bude pokračovat.

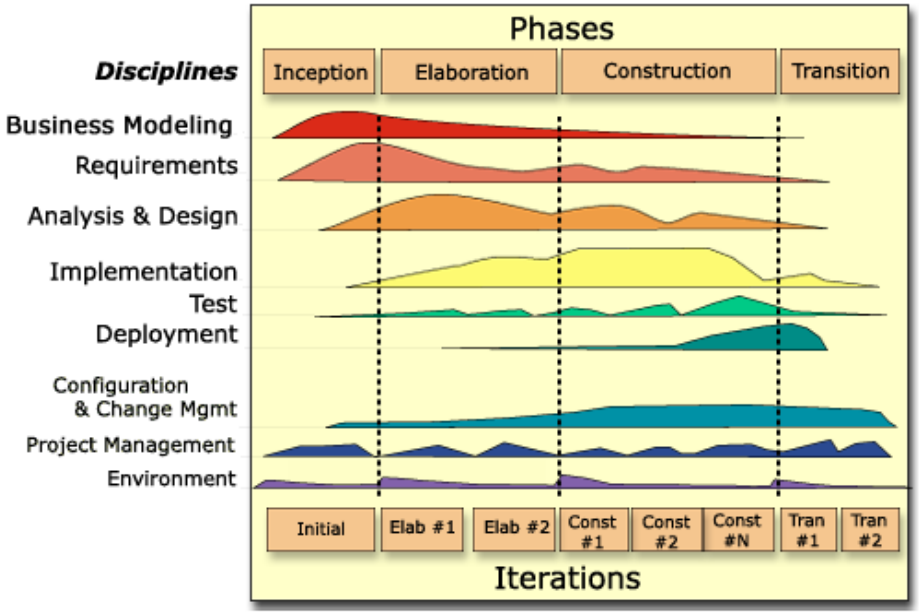
- *Elaboration*: Cílem této části je analýza a sběr zbývajících funkčních požadavků, vytvoření základní architektury a ukázku systému. Vytvoří se podrobný plán iterací a popíšou se detailně případy užití. Na konci se hodnotí milník *Lifecycle Architecture*, kde se zkoumá, jestli je architektura a vize produktu stabilní a zúčastněné strany s vizí souhlasí, jestli je zachycena většina funkčních požadavků na systém a jestli jsou náklady na projekt akceptovatelné a je vytvořen detailní plán další části.

- *Construction*: Jedná se o vývojovou část celého procesu. Hlavním smyslem je porozumění požadavkům a dokončení vývoje na předem připravené architektuře. Vývoj probíhá iterativně a postupně se testují a integrují se napsané části systému. Konec označuje milník *Initial Operational Capability* a otázkou je, jestli produkt už je v takové fázi, že může být nasazen k zákazníkovi.

- *Transition*: V této části probíhá nasazení a dodání produktu zákazníkovi. Během této části se opravují neidentifikované chyby, dodělávají se poslední úpravy a poskytuje se podpora. Milníkem je zde *Product Release* a hlavním bodem zkoumání je spokojenost zákazníka.

Výhodou je zejména možnost zapracování měnících se požadavků v průběhu projektu, ať už se jedná o požadavky od zákazníka, nebo požadavky vzniklé zevnitř projektu. Testování menších částí produktu je snazší a zákazník má možnost sledovat průběh díky přírůstkům na konci každé iterace. Nevýhodou je, že proces je celkem složitý a vyžaduje větší zapojení managementu. Může

vyžadovat i více zdrojů než jiné modely, a nehodí se proto na menší projekty.



Obrázek 2.2: Schéma Unified Process

Praktiky

Kromě popisu uvedeného výše jsou pro Unified Process důležité tyto praktiky:

- Iterativní vývoj – postupné vyvíjení produktu po malých částech, které se spojují do jednoho celku
- Komponentová architektura – základem produktu je stabilní architektura, na kterou se postupně napojují vyvíjené komponenty
- Vizuální modelování – zachycení podoby a chování systému většinou pomocí UML diagramu

2.4.3 Scrum

Scrum je iterativní a inkrementální model a řadí se do agilních metodik. Tým, který Scrum používá, je rozdělen do předem definovaných rolí, má předepsaná pravidla, artefakty a události, kterých by se měl tým účastnit (viz obr. 2.3). Projekt je rozdělen do časově ohraničených iterací a cílem doručit produkt ke spokojenosti zákazníka. Na začátku není jasné, kdy projekt

skončí, protože se počítá s tím, že v průběhu se budou požadavky měnit a zákazník ani nemusí zpočátku vědět, jak přesně má výsledek vypadat.

Oproti ostatním metodikám má Scrum dva specifické artefakty – Product a Sprint Backlog. Product Backlog je seřazený seznam všech známých požadavků. Seznam není statický a v průběhu projektu se požadavky přidávají, odebírají a nově seřazují podle aktuálních priorit. Sprint backlog je podmnožinou vybraných požadavků z product backlogu, které se budou implementovat ve sprintu.

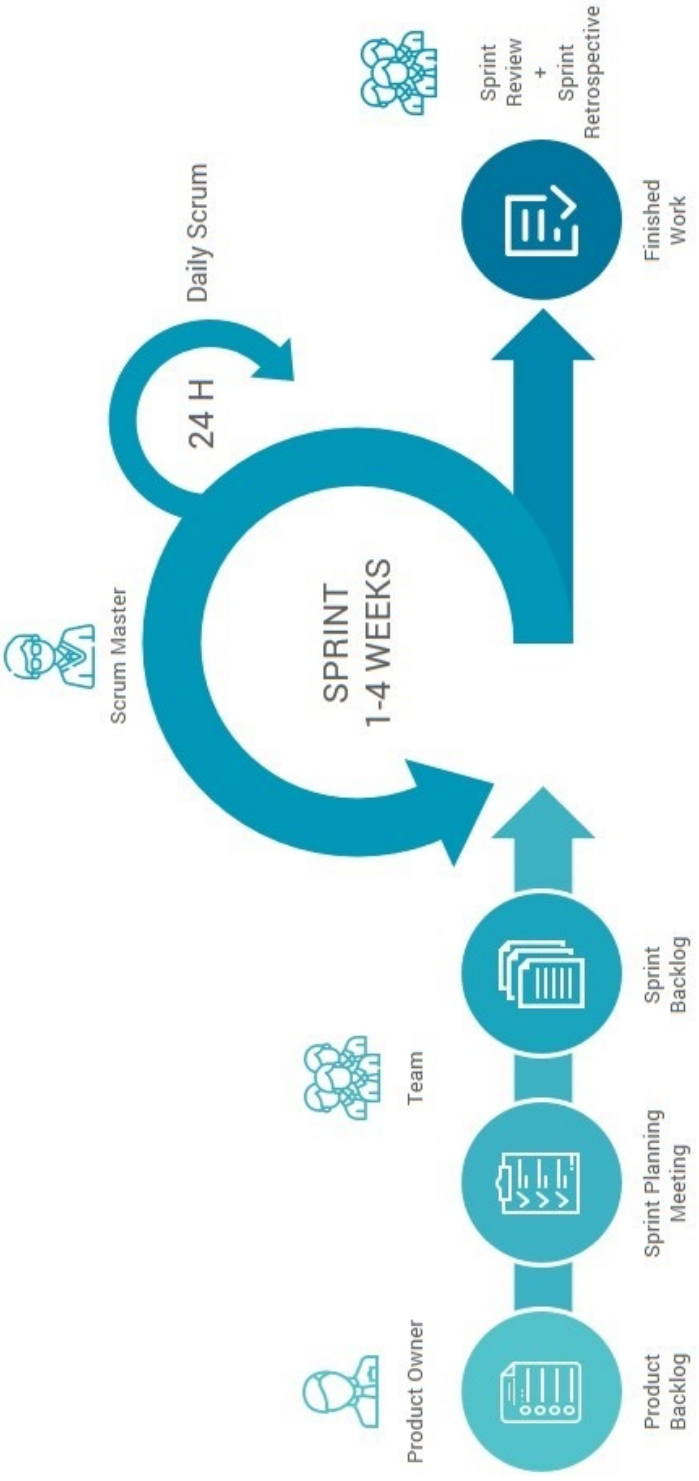
V týmu může existovat více rolí, ale 3 z nich jsou pro každý tým stěžejní. Jsou jimi Product Owner, Scrum Master a Development Team. Product Owner je zodpovědný za product backlog. Má na starost řazení požadavků podle priorit, popisování požadavků a musí se ujistit, že vývojový tým dostatečně rozumí položkám v product Backlog. Product owner by měla být jedna osoba, která bude mít poslední slovo. Development team se stará o dodání produktu po inkrementech, které se dodávají na konci sprintu. Tým se organizuje sám pomocí praktik popsaných v metodice Scrum. Při vývoji se metodika nedívá na jednotlivce, ale na tým jako celek, který by měl umět vše potřebné, aby mohl produkt dodat. Scrum master zodpovídá za to, že každý člen týmu dělá to, co má podle metodiky Scrum. Funguje také jako informační bod pro okolí, aby nebyl tým zbytečně zaneprázdněn

Scrum obsahuje také několik specifických událostí, které se konají v průběhu projektu. Sprint je „srdcem“ Scrumu. Sprint je časově ohraničená doba, za kterou se vytvoří inkrement produktu. Jsou stejně dlouhé a hned jak jeden skončí, začíná další. Každý sprint má předem definovaný cíl toho, co se musí vytvořit, a maximální délka by měla být třicet dní. Na začátku se koná tzv. sprint planning, na kterém se tým dohodne, co se v daném sprintu bude vytvářet. Vstupem je product backlog a výstupem je sprint backlog. Během sprintu se každý den koná daily scrum. Jedná se o 15-ti minutové setkání, kde se probere postup, jakého se dosáhlo od minulého daily scrumu, naplánuje se práce na dalších 24 hodin a zjistí se, jestli něco brání tomu, aby mohl vývojový tým vykonávat svoji práci. Na konci sprintu je sprint review, kde se vytvořený přírůstek ukáže zúčastněným stranám a hovoří se o tom, čeho se ve sprintu dosáhlo. Cílem je získat zpětnou vazbu a případně upravit nebo přidat požadavky. Jednou z posledních událostí je sprint retrospective, kde tým řeší, jak sprint probíhal, co se povedlo a nepovedlo a co by se dalo pro další sprint zlepšit spolu s plánem těchto zlepšení. [23]

Praktiky

Pro Scrum existuje několik specifických praktik, které se v ostatních metodikách nenacházejí. Například:

- Vedení produktového backlogu – seřazený seznam požadavků, který se v průběhu projektu mění podle potřeb
- Daily standup – každodenní setkání týmu s cílem krátkého shrnutí postupu projektu
- Retrospektiva – setkání s cílem zlepšení práce v týmu během projektu
- Zapojení zákazníka – zákazník se aktivně podílí na většině aktivit v průběhu celého projektu
- Rozdělení rolí – členové týmu jsou rozděleni do 3 základních rolí: Product Owner, Scrum Master, Development Team



Obrázek 2.3: Schéma Scrum [28]

3 Patterny a anti-patterny

Patterny a anti-patterny představují koncept v oblasti počítačové vědy a softwarového inženýrství. Během vývoje tohoto oboru vzešla potřeba vyřešit některé zásadní problémy, které vznikají, když lidé převedou obchodní myšlenku do kódu aplikace. [2] Identifikovaný problém může být překážkou, která brání organizacím v úspěchu. Pokud je problém popsán špatně, nemusí být v možnostech organizace tento problém odstranit. Na druhou stranu, pokud je problém identifikován správně, skoro vždy se s nimi lze vypořádat. [14]

3.1 Co to je pattern

V softwarovém inženýrství je pattern (vzor) formalizovaná praktika, kterou lze využít pro řešení známého problému při návrhu nebo implementaci aplikace či systému. Jedná se o popis nebo šablonu, kterou lze použít v závislosti na situaci. Efektivní návrh softwaru vyžaduje zvážení otázek, které nemusí být v počátku vývoje viditelné. Použití patternů pomáhá předcházet problémům, které mohou vyvstat v pozdějších fázích vývoje.

Patterny poskytují obecná řešení, zapsaná ve formátu, který nevyžaduje specifika vázaná na konkrétní kontext, ve kterém se problém objevil. Často lidé chápou, jak aplikovat určité techniky na určité problémy. Neznamená to ale, že jeden pattern vyřeší všechny problémy, se kterými se lze během projektu setkat. [26]

3.1.1 Typy patternů

Patternů je spousta a dělí se na obecné skupiny podle použití. Patří mezi ně například patterny pro vytváření architektury (Layered, Client-server, Master-slave), psaní kódu (Abstract Factory, Builder, Singleton), vytváření struktury programu (Adapter, Bridge, Fasade) nebo komunikaci mezi třídami (Command, Interpreter, Observer).

3.2 Co je to anti-pattern

Anti-patterny je opakem patternu. Je to určitý vzor v softwarovém vývoji, který je považován za špatné (neefektivní) řešení známého problému. Stejně

jako pattern je popsán literární formou, která popisuje běžně se vyskytující řešení problému, který vytváří negativní důsledky. Takto popsán poskytuje plán na zvrácení příčin problému a implementaci řešení. Anti-pattern může být výsledkem toho, že manažer nebo vývojář neznají lepší řešení, nemají dostatečné znalosti nebo zkušenosti při řešení problému, nebo použijí dobrý vzor v nesprávném kontextu. [2]

3.2.1 Typy anti-patternů

Anti-patterny se objevují na všech úrovních vývojových projektů. Může jít například o anti-patterny vývojové (The Blob, Spaghetti Code, Golden Hammer), architektonické (Jumble, Stovepipe System) a projektového managementu. [2] Tato práce se zaobírá hlavně anti-patterny a bad practices v oblasti projektového managementu.

Anti-patterny projektového managementu

V softwarovém inženýrství zahrnuje více než polovina práce mezilidskou komunikaci a řešení problémů mezi lidmi. Anti-patterny projektového managementu popisují právě ty scénáře, ve kterých můžou být problémy pro softwarové procesy až destruktivní. Před příchodem elektronické pošty a intranetu byla úloha manažera především zajistit komunikaci mezi klíčovými stranami a konat důležitá rozhodnutí. V dnešní době tato hranice pravomocí mizí a lidé jsou oprávněni řešit problémy bez zásahu managementu. I tak manažeři hrají důležitou roli v softwarovém procesu, například při řízení zdrojů nebo komunikaci se zákazníky. [2]

Zde je několik anti-patternů, které se mohou vyskytnout v projektovém managementu:

Analysis Paralysis Snaha o dokonalou dokumentaci ve fázi analýzy často vede k oddálení nebo zablokování projektu. Řešením je iterativní vývoj, kde popis požadavků může počkat do doby, než budou opravdu potřeba a dokud nejsou potřebné znalosti. [2]

Death by Planning Na začátku projektu se spousta organizací snaží o detailní plán. Takovéto myšlení je vhodné při například výrobních činnostech, ale ne nutně u softwarových projektů, které na začátku mají spousty neznámých. Řešením je vytvoření plánu, který bude jasně ukazovat to hlavní, čeho se projekt snaží dosáhnout, a bude obsahovat milníky, které ukáží postup projektu. [2]

Corncob Corncob je slangový termín, který se často používá pro popis lidí způsobujících problémy. Jsou to potížisté, kteří se mohou objevit v týmu, jenž vyvíjí software. Jejich postoj a chování může být ovlivněno jejich osobností, ale často vzniká i kvůli osobní motivaci. Vývoj softwaru může být stresující a corncobs tyto problémy ještě více zhoršují a vytváří náročném prostředí. Řešení je více, většinou si vyžadují zásah manažera. Může se jednat například o přenesení plné zodpovědnosti za vyřešení úkolu. [2]

Irrational Management Tento anti-pattern zahrnuje celou řadu běžně se vyskytujících problémů v projektovém řízení, ze kterých lze vysledovat charaktery osob, které projekt vedou. Manažer může mít až nezdravý zájem o nějakou technologii nebo charakterní vadu, díky které jedná iracionálně nebo neefektivně. Jeho rozhodnutí pak vedou softwarový projekt v iracionálním směru bez ohledu na to, jak nesmyslné jsou. Řešení leží v rukou manažera, který by měl zlepšit komunikaci s okolím a přiznat si, že ne všechna rozhodnutí, která dělá, jsou v dané situaci podle něj nejlepší. [2]

Project Mismanagement Anti-pattern se zaobírá kontrolou a monitorováním softwarového projektu. Řádné monitorování a řízení vede k úspěšnému vývoji. Vývoj produktu je stejně složitý jako tvorba plánu a obojí zahrnuje spoustu kroků a procesů. Klíčové aktivity jsou pak často přehlíženy nebo brány na lehkou váhu, což může vést ke zmatkům a případně i nedodržení domluvených podmínek. Řešením je zavést důkladnou analýzu rizik, která budou popisovat jak vyřešit problémy, které může tento anti-pattern způsobit. [2]

4 Application Lifecycle Management

Řízení životního cyklu aplikací (Application Lifecycle Management – ALM) se zabývá řízením aplikace po celou dobu její životnosti. Začíná od sběru požadavků a zaniká po ukončení provozu aplikace. Digitalizace zavádí řadu změn, které ovlivňují obchodní procesy, modely i produkty. IT oddělení musí reagovat na rychle se měnící požadavky. To vyžaduje spolupráci a komunikaci mezi obchodním oddělením a IT. Díky ALM je cesta od fáze potřeb k finálnímu IT systému nebo produktu transparentní a sledovatelná pro všechny účastníky. [16]

Nedílnou součástí řízení životního cyklu aplikace je zvládání příchozích požadavků, změn a priorit. ALM aplikace jsou toto schopny zajistit pomocí nástrojů, které nabízejí pro všechny artefakty životního cyklu aplikace, například repozitáře s možností dotazování a hodnocení, integrovanou správu verzí a správu konfigurace. [16]

Výhody při využívání ALM jsou:

- **Větší efektivita** – možnost sdílet informace a osvědčené postupy při vývoji
- **Rychlejší release** – efektivní plánování rychlejších release cyklů (nasazení nové verze)
- **Monitoring změn** – přehled stavů požadavků a zobrazení aktuálních informací
- **Lepší rozhodování** – integrace ALM nástrojů zlepšuje komunikaci a schopnost lepšího plánování

4.1 ALM nátsroje

Application Lifecycle Management nástroje jsou podpůrné nástroje, které usnadňují procesy jako řízení požadavků, změn, verzí, konfigurací atd. V dnešní době používá ALM nástroje téměř každý projektový tým čítající dva a více lidí. Obvykle se využívá jeden nástroj, který spojuje několik funkčností, nebo například kombinace jednoho nástroje pro issue-tracking (správu změn) a druhého pro správu verzí (VCS – Version Control System). [20]

Rozsah toho, jak se nástroje používají, se může lišit v závislosti na parametrech projektu, jako je například počet lidí pracujících na projektu, rozsah projektu, časový rámec atd. Jednoduchý projekt, který vyvíjí jeden člověk, nebude nejspíše ALM potřebovat vůbec, protože by to přineslo jen více komplikací a zbytečné administrativy. Oproti tomu na projektu, na kterém pracuje mnoho lidí a může trvat i několik let, je zapojení ALM nástrojů nezbytné, aby bylo vůbec možné projekt spravovat.

První ALM nástroje neposkytovaly více funkčností, ale byly většinou zaměřeny na jednu konkrétní. Až později se začaly objevovat nástroje, které jich poskytovaly více na jednou v komplexním provedení tak, jak jsou známy dnes. Pokud je řeč o ALM nástroji, nemluví se o jednom konkrétním, který obsahuje všechny, ale o jednotlivých funkčnostech, které mohou být spojeny k sobě v jednom nástroji.

Níže je popsán výčet několika typů spolu s jejich popisy a příklady.

4.1.1 Systém pro správu verzí

Systémy pro správu verzí (VCS – Version Control System) umožňuje týmu v čase řídit změny zdrojového kódu. Software pro správu verzí sleduje každou změnu kódu ve speciálním druhu databáze a vývojáři se mohou vrátit k dřívější verzi kódu, pokud by došlo k chybě. Chybu pak můžou rychleji vyřešit a zároveň minimalizují narušení práce ostatních členů týmu. Pro téměř všechny softwarové projekty je zdrojový kód cenný majetek a musí být chráněn. Nástroje pro kontrolu verzí chrání zdrojový kód před poškozením, které mohlo vzniknout v důsledku lidské chyby a dalšími nezamýšlenými následky. Vývojáři pracující v týmech neustále přidávají nový kód a upravují starý. Jeden vývojář může pracovat na nové funkci, zatímco jiný vývojář opraví chybu změnou kódu, který s novou funkcí nesouvisí.

Díky řízení verzí může tým sledovat každou jednotlivou změnu a zabránit tomu, aby v práci několika členů pracujících zároveň vznikly konflikty v kódu. Změny provedené v jedné části softwaru mohou být nekompatibilní se změnami provedenými ve stejné době jiným vývojářem. Pokud nastane tento problém, měl by být vyřešen tak, aby se nezablokovala práce zbytku týmu. Kromě toho ve všech fázích vývoje může každá změna zavést nové chyby a díky správě verzí je snazší najít, která změna chybu způsobila.

Dobrý software pro správu verzí podporuje vývojářem a nijak ho neomezuje v práci. Týmy, které nepoužívají žádnou formu řízení verzí, se mohou dostat do problémů, kdy neví, jaké změny byly v projektu provedeny, jaká verze je uživatelům k dispozici, nebo jestli se nevytvořily změny mezi dvěma díly aplikace, které po připojení nebudou kompatibilní. V dnešní době je

software pro správu verzí nezbytnou součástí každodenní praxe bez ohledu na velikost projektu. [1]

Výhody VCS

Používání VCS [1] při vývoji projektu přináší hned několik výhod. Například:

- **Historie změn:** Nástroje obsahují kompletní historii změn každého souboru. To zahrnuje vytváření a mazání souborů a úpravy jejich obsahu. Historie by měla také obsahovat autora, datum vytvoření změny a písemný popis změny. Díky kompletní historii se vývojáři mohou vrátit ke starším verzím aplikace a snáze opravit nalezené chyby.

- **Větvení s slučování:** Souběžná spolupráce mezi členy týmu je samozřejmost. Vytvoření nezávislých pracovních toků („větví“) v nástrojích VCS pomáhá udržovat přehled a zároveň poskytuje možnost pozdějšího sloučení. To vývojářům umožňuje ověřit, že změny v různých větvích nejsou v konfliktu. Běžnou praktikou je vytvářet novou větev pro každou novou funkci, aby změny nenarušily stávající chod aplikace, a každou release verzi. Existuje mnoho různých postupů jak větve využívat a je na týmech, pro který se rozhodnou.

- **Sledovatelnost:** Možnost vysledovat každou změnu v softwaru a připojit ji k softwaru pro správu projektů a sledování chyb může pomoci k budoucí analýze. Mít anotovanou historii kódu na dosah ruky umožňuje vývojářům provádět správné změny, které jsou v souladu se zamýšleným návrhem systému. To může být obzvláště důležité pro efektivní práci se staršími kódy.

Příklady VCS jsou:

- **Git**[1]
- **CVS**[2]
- **SVN**[3]
- **Mercurial**[4]

[1] https://git-scm.com/
[2] https://www.nongnu.org/cvs/
[3] https://subversion.apache.org/
[4] https://www.mercurial-scm.org/

4.1.2 Issue/Bug tracker

Sledování problémů (issue tracking) je zobecněná verze sledování chyb (bug tracking). Fráze „sledování chyb“ může naznačovat, že chyby existují pouze v kódu, ale není tomu tak. Stejně snadné je mít chyby v požadavcích, designu nebo specifikacích. Sledování problémů je navrženo tak, aby pomohlo odhalit nebo zabránit všem těmto chybám.

Sledování problémů může znít trochu dramaticky, ale je to v podstatě jen seznam úkolů, který podporuje proces kontroly. Hlavním úkolem nástrojů pro sledování problémů je zaznamenat na jednom místě plánované úkoly, nalezené chyby a další práci do tzv. ticketů. Jedná se vlastně o formulář, kde jsou zachyceny parametry úkolu, jako je název, popis, priorita, datum vytvoření a splnění, člověk, který má úkol přidělen na vyřešení, odhadnutý a skutečně strávený čas na úkolu, stav atd. To podporuje plánování projektu, sledování postupu a lepší rozhodování. Nástroje jsou schopné různé informace zobrazovat i v grafech pro lepší vizualizaci a některé i podporují metodiky vývoje, jako je například Scrum. [25]

Výhody issue/bug trackingu

Cílem při psaní software je dodat ho co možná nejrychleji a s co nejmenším počtem chyb. Tyto dva koncepty jsou ovšem v rozporu. Snížením kvality lze vždy dodávat rychleji, ale aby se zabránilo chybám a budoucím problémům, musí se investovat i čas. Sledování problémů [25] pomáhá vyvážit tyto dvě strany a ušetřit jak čas, tak peníze. Využívání nástrojů pro sledování chyb může pomoci k:

- **Rychlejšímu odhalení chyb.** Formálním zapsáním a zamyšlením se nad daným úkolem může pomoci k odhalení chyby ještě před implementací. Čím dříve se chyba odhalí, tím menší jsou náklady na její odstranění.
- **Nalezení většího počtu chyb.** Nejen vývojáři používají issue tracking nástroje. Spolupracovníci v oblasti obchodu mohou odhalit chyby v logice, které vývojář nemusí na první pohled vidět.
- **Určení priorit.** Není vždy jednoduché se rozhodnout, na čem zrovna pracovat. Díky zanesené prioritě u úkolů je jasné, co „hoří“, a co lze nechat na později. Tím se odstraní dohady a lidé se mohou lépe zaměřit na vykonávanou práci.
- **Větší spokojenosti zákazníků.** Méně chyb znamená menší náklady

na jejich opravu a rychlejší a kvalitnější produkt. Zákazník ušetří čas a prostředky, protože nemusí hlásit chyby a čekat na jejich opravu.

Příklady issue/bug trackeru jsou:

- **Redmine**[5]
- **GitHub**[6]
- **Mozilla Bugzilla**[7]
- **Jira**[8]

4.1.3 Nástroje znalostní báze

Za znalostní bázi lze označit prostor pro ukládání dat, který obsahuje informace o určitém produktu, službě, tématu nebo koncepci. Organizace vytvářejí vlastní znalostní základny, kam umísťují všechny znalosti v rámci své organizace o konkrétním tématu. Poskytují tak spolupracovníkům jedno místo se všemi potřebnými informacemi. Informace ve znalostní bázi se zaměřují jak na interní zaměstnance, tak externí subjekty, jako je veřejnost, zákazníci nebo potenciální zákazníci, kteří se chtějí o organizaci nebo jejích produktech dozvědět více. Cílem znalostní báze je poskytovat ucelené informace a v případě vnitřního systému zvýšit celkové porozumění v organizaci a šířit znalosti.

Existuje mnoho typů znalostí, ale nejzákladnějším rozlišením je na explicitní a implicitní. Explicitní znalosti se dají snadno formulovat, popsat a pochopit, a proto se dají lehce předat dále. Implicitní znalosti není nijak vyjádřena, ale má potenciál na to, aby byla zdokumentována a sdělena dále. Implicitní znalost je obtížnější se naučit, protože zahrnuje věci jako řeč těla, estetický smysl nebo inovativní myšlení. [22]

Výhody znalostních bází

V interních znalostních bázích uchovává organizace svoje odborné znalosti a poskytuje zaměstnancům přístup k těmto informacím. Odpadá tedy potřeba hledat informace jinde a zrychlí se tak postup učení a hledání řešení při výskytu problému. Externí znalostní báze, které jsou určeny pro širokou

[5] https://www.redmine.org/
[6] https://github.com/
[7] https://www.bugzilla.org/
[8] https://www.atlassian.com/software/jira

veřejnost, mají poskytnout přehledné informace o organizaci a jejích produktech nebo službách. Externí znalostní báze mají často podobu stránky s informacemi o organizaci, často kladených dotazů, návodů nebo určité formy helpdesku. [22]

Příklady knowledge base nástrojů jsou:

- **Confluence**[9]
- **Zendesk**[10]
- **ProProfs**[11]

4.1.4 Komunikační nástroje

V minulosti byli lidé tvořící software společně na jednom místě a vždy byl na blízku nějaký expert. V dnešní době je však situace jiná. V organizacích mohou existovat distribuované týmy pracující na vývoji softwaru. Takové to týmy jsou čím dál tím víc běžnější a nacházejí se často i v různých částech světa. Tým se v tom případě dá vnímat jako komunita „cizinců“, která má za úkol vytvořit společný produktu. Výměna znalostí mezi lidmi v takovémto týmu musí překonat nové překážky a schopnost komunikace může přímo souviset s úspěchem projektu. Pro úspěšnou spolupráci musí mít tým nástroje, které jim umožní snadnou komunikaci. Každý člen týmu může preferovat jiný způsob jak se s ostatními na dálku dorozumět, a proto je volba nástroje velice důležitá. [27]

Při vývoji software je důležité mít dobré komunikační metody a dovednosti, jelikož předávání informací mezi členy týmu je pro hladký průběh projektu klíčové. Zákazník chce vědět stav projektu, manažeři potřebují informace od vývojářů a vývojový tým musí denně komunikovat mezi sebou. [8] Zatímco mluvení nemusí být pro někoho nejlepším způsobem komunikace, existuje několik dalších způsobů, jak si informace předat, jako je například klasický e-mail, videokonference nebo chat.

Mezi komunikační nástroje patří:

- **Skype**[12]
- **Slack**[13]
- **E-mail**

[9] https://cs.atlassian.com/software/confluence/knowledge-base
[10] https://www.zendesk.com/guide/
[11] https://www.proprofs.com/knowledgebase/
[12] https://skype.com/
[13] https://slack.com/

5 SPADe

SPADe, neboli Software Process Anti-pattern Detector, je software, který je vyvíjen na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU). Jeho zamýšleným účelem je dolování dat z ALM nástrojů, jejich ukládání do datového skladu a identifikace (anti-)patternů a „bad practices“ v projektovém řízení. [20]

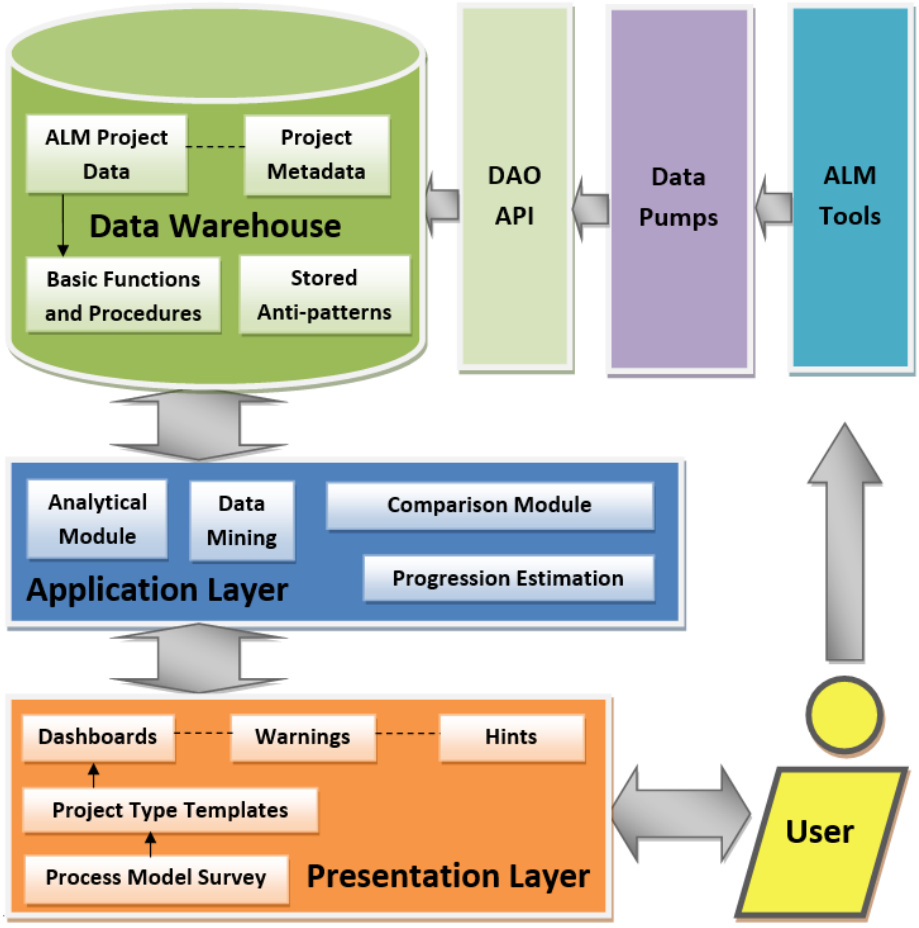
5.1 Popis nástroje

Jednou z hlavních funkčností nástroje je dolování dat ALM nástrojů. Data z ALM nástrojů včetně metadat se přes datové pumpy a rozhraní pro ukládání dat nahrají v jednotném formátu do datového skladu. Data se poté v nástroji dají zkoumat s cílem odhalení (anti-)patternů a porovnávat s projekty o přibližně stejných parametrech a výsledky zobrazit pomocí grafického uživatelského rozhraní (Graphical User Interface – GUI). Zobrazit lze základní statistiky, grafy a nalezené (anti-)patterny. [20] Schéma nástroje SPADe lze vidět na obrázku 5.1.

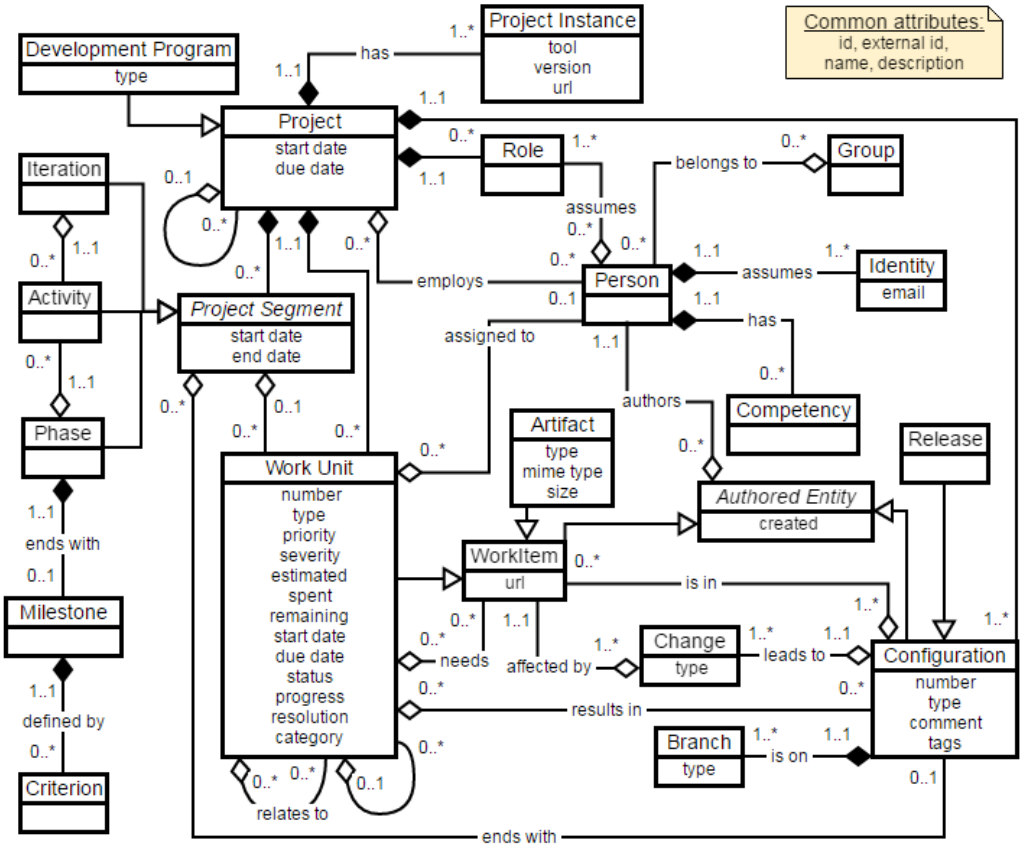
5.1.1 Struktura SPADe

Data [20] se dostávají do datového skladu nástroje přes datové pumpy, které používají veřejné rozhraní (Application Programming Interface – API) ALM nástrojů, a k uložení dat používá aplikace Data Access Object (DAO). V datovém skladu se hromadí data pro účely porovnání zkoumaného projektu s projekty dokončenými. Velký důraz je zde kladen na kontrolu metadat projektů, která ovlivňují celkové dění v projektu. Nelze například hledat patterny související s iteracemi v projektu s vodopádovým procesem nebo v projektu, který začíná od nuly úkoly na seznámení se s současným stavem systému, když žádný není. Datový model nástroje SPADe (viz obr. 5.2) byl vytvořen tak, aby obsahoval co nejmenší redundanci dat. Model obsahuje entity víceméně přímo mapovatelné z ALM nástrojů i entity kompletně vytvořené, jelikož k jejich určení nejsou data z ALM nástrojů uzpůsobena, ale jejich přítomnost může pomoci s detekováním (anti-)patternů. Model byl tvořen tak, aby i názvosloví co nejvíce odráželo zažitou praxi. Výjimku tvoří tyto entity, které byly přejmenovány při tvoření modelu:

- *Configuration:* Tato třída reprezentuje stav aplikace v čase. Jedná se o souhrn změn mezi 2 stavy dat. Změnu stavu dat si lze představit jako



Obrázek 5.1: Schéma nástroje SPADe



Obrázek 5.2: Doménový metamodel datového skladu nástroje SPADe [20]

jednu uživatelskou akci, například commit nebo změnu úkolu v ALM nástroji.

- *Work Item:* Třída reprezentuje pracovní položku a jedná se o rodičovskou třídu pro entity Artifact a Work Unit. Hlavní myšlenkou je, aby se se všemi tickety a artefakty pracovalo v programu SPADe stejně.
- *Work Unit:* Tato třída reprezentuje úkol (ticket) v ALM nástroji a její atributy jsou dané především funkcemi ticketovacích nástrojů.
- *Project Segment:* Jedná se o třídu nadřazenou pro entity Activity, Iteration a Phase.

Aplikační vrstva nástroje poskytuje možnost zkoumání dat analytickými nástroji. Dají se zkoumat detekované anti-patterny, odchylky od procesu projektu a možnost nalézání souvislostí mezi daty. Další možnost je porovnání dat s daty uzavřených projektů se zhruba stejnými metadaty. Účelem tohoto porovnávání je odhadnout další vývoj projektu. Myšlenka při tomto postupu je taková, že pokud podobný projekt s podobnými anti-patterny ve zhruba stejné fázi skončil se zpožděním nebo jinými problémy, je šance, že se u zkoumaného projektu bude situace opakovat.

Nástroj poskytuje i prezentační vrstvu pro zobrazování statistik, grafů a nalezených anti-patternů a zobrazování dat lze upravit podle potřeb a obsah tak personalizovat. Uživatelské rozhraní poskytuje i upozornění ohledně pravděpodobného neúspěchu projektu a rady, které by měly pomoci k odstranění nebo minimalizaci dopadů anti-patternů. [20] Součástí prezentační vrstvy je i část pro zobrazování strukturální závislosti dat pomocí hranových a uzlových grafů (viz [11]).

5.1.2 Současný stav

Nástroj je schopen dolovat data z ALM nástrojů a ukládat je do datového skladu. Nad těmito daty je uživatel schopný provádět dotazy pomocí GUI a data z datového skladu lze v nástroji zobrazit. Data lze vyexportovat pro aplikace IMiGEr (Interactive Multimodal Graph Explorer) [9] a Timeline [6], které jsou vyvíjeny na ZČU v Plzni, a pro aplikaci Codeface [13], která je vyvíjena na vysoké škole v Regensburgu. Nástroj je schopen zobrazit nalezené (anti-)patterny a zobrazit tato data do grafu. SQL dotazy pro nalezení (anti-)patternů jsou psány ručně ve specifickém formátu. Je tedy obtížné vkládat do nástroje další, a proto vznikla potřeba pro vytváření SQL dotazů pro nalezení (anti-)patternů v GUI aplikaci pro SPADe.

5.1.3 Požadavky

Hlavním požadavkem na práci bylo vytvoření nového okna v GUI části aplikace SPADe, kde budou moci uživatelé vytvářet dotazy nad databází a tím vytvářet předpisy pro detekování (anti-)patternů.

- **Lokalizace:** GUI část aplikace SPADe je lokalizována do českého i anglického jazyka. Toto je potřeba dodržet pro větší použitelnost ze stran uživatelů.

- **Konstanty:** Uživatel bude mít možnost vytvořit si pojmenované konstanty s libovolnou hodnotou pro využití při vytváření dotazů. Konstanty se budou ukládat do souboru, aby bylo možné je při dalším puštění aplikace načíst.

- **Proměnné:** Aplikace bude poskytovat možnost vytvořit agregační dotaz, jehož výsledek bude reprezentovat proměnnou pro další využití při vytváření dotazů. Proměnné bude možné přejmenovat a budou nezávislé na projektu, aby se ušetřil čas při jejich znovuvytváření. Proměnné bude možné editovat a mazat přímo v aplikaci a budou uloženy do souboru, ze kterého se budou načítat při opětovném puštění aplikace.

- **Vytváření uživatelských dotazů:** Uživatelsky definované grafy v nástroji existují, ale jejich vytváření je složité přes ručně psané SQL dotazy ve specifickém formátu. Do aplikace se přidá okno pro vytváření dotazů pomocí GUI a možnost zvolení dat pro osu X pro následné vykreslování do grafu. Součástí okna bude i zadávání kritérií pro vyhodnocení (anti-)patternů, kde bude uživatel moci použití dříve vytvořené konstanty a proměnné. Při zadávání kritérií bude k dispozici elementární aritmetika a možnost porovnávat výsledné sloupce mezi sebou.

- **Vykreslení do grafu:** Vytvořené (anti-)patterny bude možno vykreslit do grafu pomocí stávající implementace, kterou nástroj SPADe GUI poskytuje. Jelikož je vykreslování (anti-)patternů do grafu v programu implementováno, není potřeba vytvářet novou verzi vykreslování grafu.

- **Přenositelnost:** Cokoliv, co uživatel díky nové funkčnosti vytvoří (tzn. konstanty, proměnné, patterny), musí být přenositelné mezi projekty a také mezi počítači s aplikací SPADe GUI. Zjednoduší se tak práce s aplikací a ušetří se čas při vytváření stejných (anti-)patternů pro jiná data.

- **Export:** Výsledky z detekce (anti-)patternů bude možno vyexportovat do souboru ve formátu CSV pro případné analýzy v tabulkových softwarech, které nejsou v tuto chvíli v aplikaci možné provést. Součástí aplikace bude logování spuštěných dotazů do souboru pro případnou manuální kontrolu.

6 Realizace řešení

Tato kapitola popisuje návrh a implementaci rozšíření GUI části aplikace. Rozšíření je zapojeno do původní implementace aplikace SPADe s využitím jejíchž existujících prvků. Původní implementace včetně datového modelu a načítání dat zůstala nezměněná. Dále jsou v kapitole informace o testování implementace a popis možných budoucích rozšíření.

6.1 Návrh řešení

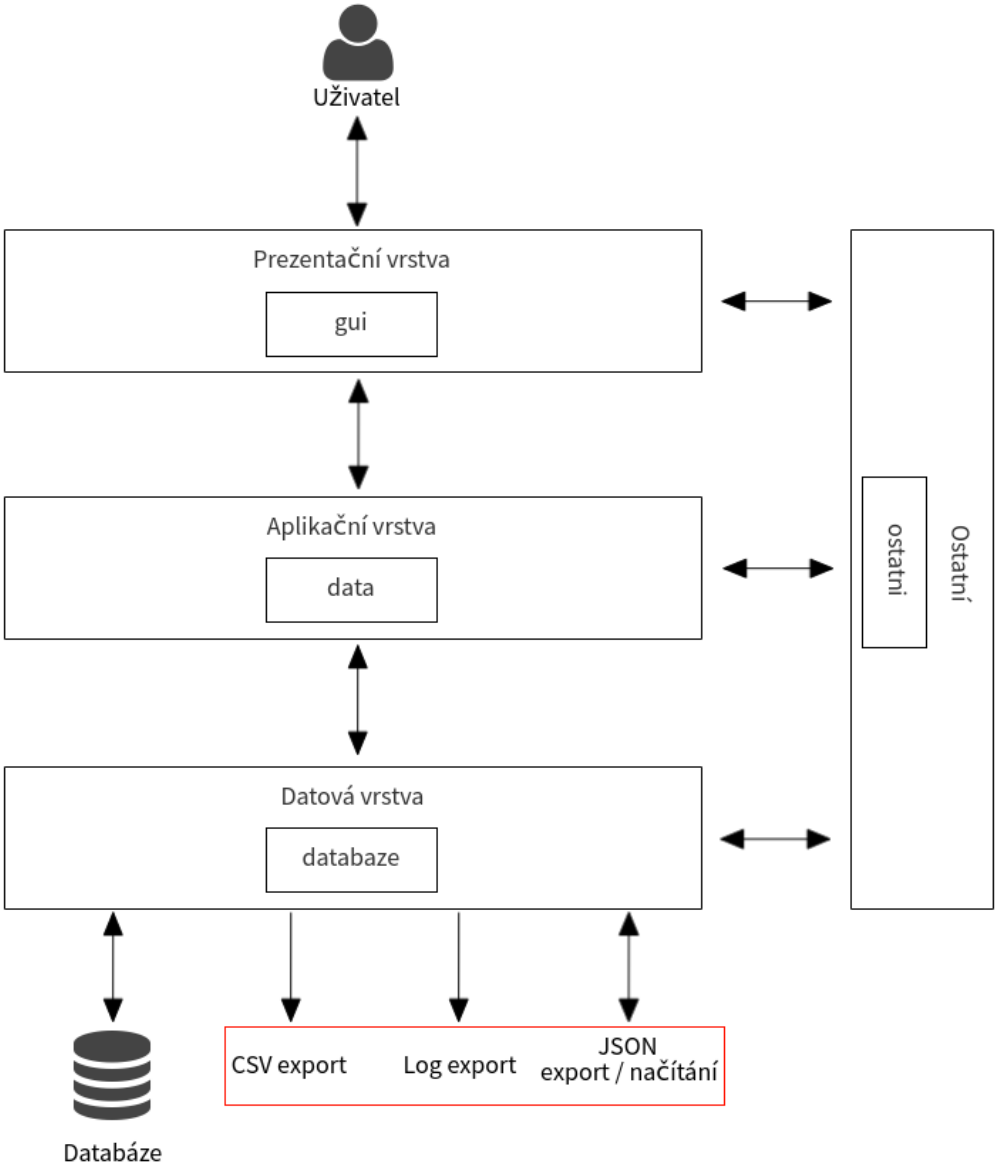
Rozšíření aplikace bude sloužit pro vytváření uživatelsky zadaných dotazů nad databází nástroje SPADe pro detekci (anti-)patternů a bad practices. Po vytvoření dotazů použije rozšíření stávající implementaci pro zobrazení výsledných dat do grafu. Rozšíření bude lokalizováno do anglického a českého jazyka stejným způsobem jako původní aplikace. Obrázek 6.1 zobrazuje vrstvy aplikace a balíky zdrojového kódu. Úpravy aplikace se budou týkat všech vrstev. Nově přidané budou exportní možnosti, označeny na obrázku červeně.

6.1.1 Databázové pohledy

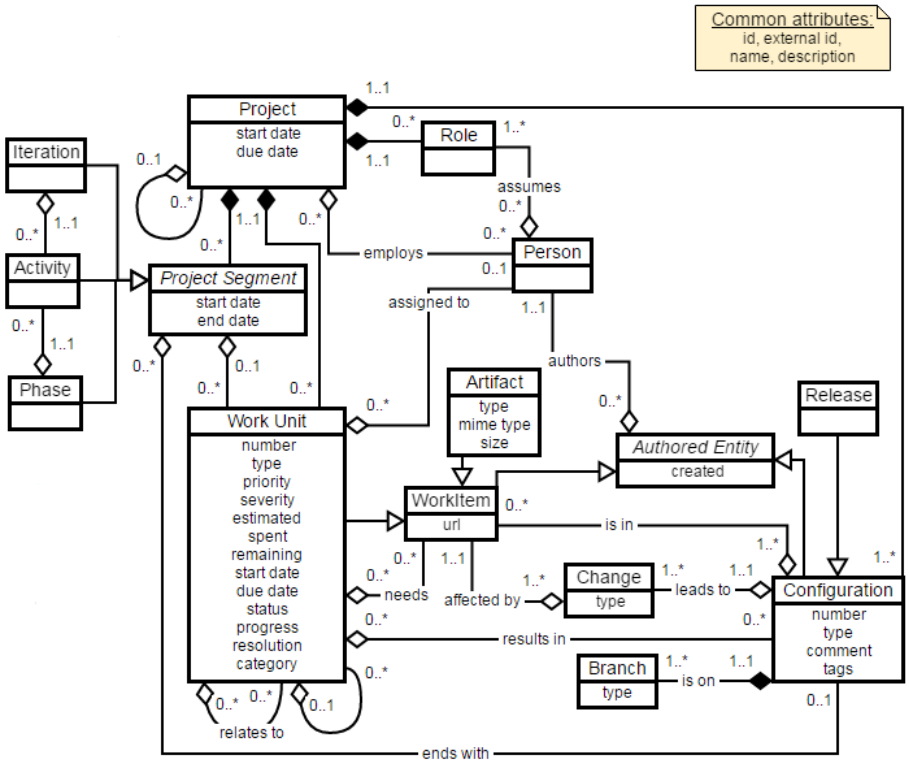
V současné podobě datového modelu by nebylo uživatelsky příjemné vytvářet dotazy kvůli jeho komplexnosti. Za účelem získání dat pro psaní SQL dotazů bude potřeba vytvořit databázové pohledy, které budou kombinovat všechny potřebné informace z datového modelu. Pohledy budou shlukovat informace do logických celků, jako například artefakty, commity, konfigurace atd. Analýza datového skladu ukázala, že pro potřeby této práce se budou zpracovávat data jen z části datového modelu, zobrazeném na obrázku 6.2.

6.1.2 Uživatelsky psané dotazy

Pro psaní SQL dotazů v GUI bude do horní lišty menu přidána položka na vytváření uživatelských dotazů a grafů. Po výběru dané možnosti se otevře nové okno (viz obr. 6.3), kde bude moci uživatel vytvořit novou detekci (anti-)patternů v podobně SQL dotazů. Výsledky dotazů budou zobrazovány do grafu, proto okno poskytne možnost zvolit si, jaká data budou na ose X. Uživatel bude mít možnost vytvořit si i konstanty a proměnné, které lze následně použit při vytváření SQL dotazů. V horní části okna bude seznam s

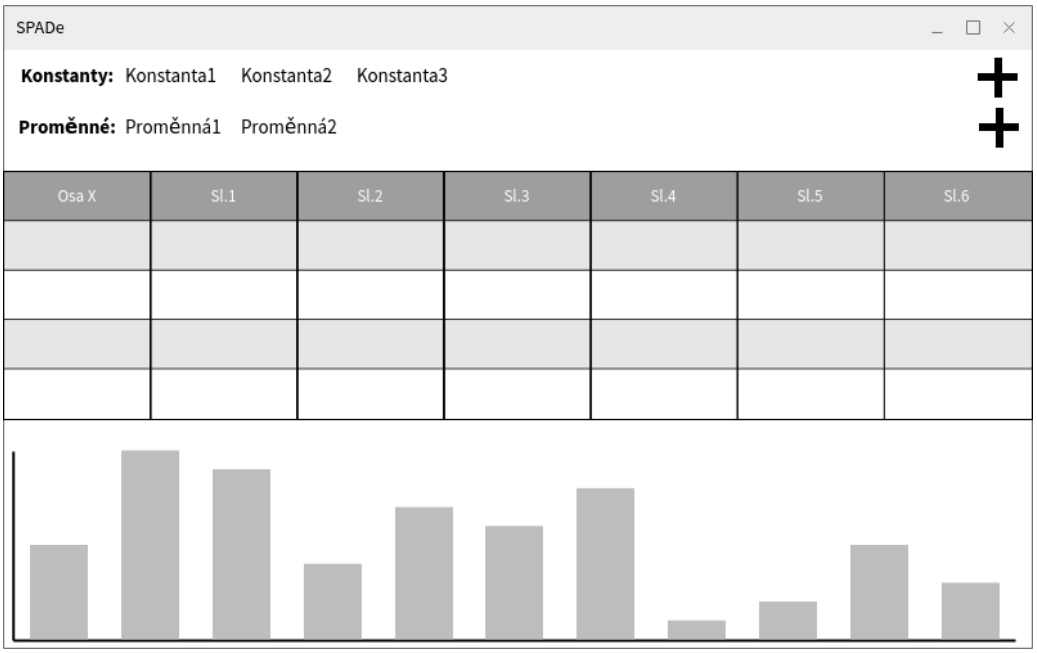


Obrázek 6.1: Cílová architektura aplikace



Obrázek 6.2: Upravený doménový metamodel datového skladu nástroje SPADe

vytvořenými konstantami a proměnnými a ve střední části bude prostor pro vytváření dotazů a na vykreslení grafu. Vytvořené SQL dotazy bude moci uživatel pomocí tlačítka spustit a poté se zobrazí tabulka výsledků, kde uživatel zadá parametry pro detekci (anti-)patternu. Při zadávání kritérií bude u každé hodnoty možnost zadat aritmetický operátor pomocí dropdown seznamu. Pokud tak uživatel učiní, objeví se druhé textové pole pro zadávání hodnot. Uživatel si také bude moci zvolit pomocí přepínače, jestli chce výsledky porovnávat se zadanou hodnotou nebo jiným výsledným sloupcem. U porovnávání sloupců bude také možnost použít elementární aritmetiku. Dále bude možné vybrat zda se budou výsledky porovnání jednotlivých sloupců spojovat pomocí logického AND nebo OR. Po zadání kritérií si může uživatel nechat spočítat výsledek detekcí a zobrazit graf výsledných dat pomocí tlačítka ve spodní části okna. Ve spodní části bude také tlačítko pro ulo-



Obrázek 6.3: Návrh hlavního okna rozšíření

žení detekce (anti-)patternu. Po stisknutí se detekce uloží do souboru ve formátu JSON, který bude obsahovat jak informace o dotazu, tak informace o zvolených parametrech pro detekci (anti-)patternů. Pokud uživatel nezadá název detekce (anti-)patternu, vygeneruje se univerzální název ve tvaru `Query_TIMESTAMP`. JSON soubor bude vypadat následovně:

```
{
  "axis": "Year",
  "dateTo": "2017-06-30",
```

```
  "dateFrom": "2017-05-01",
  "invertValues": false,
  "queries": [
    {
      "joinColumn": "columnName",
      "detection": {
        "compareColumns": false,
        "detect": true,
        "firstInput": {
          "value2": "10",
          "value1": "0",
          "arithmetic": "+"
        },
        "operator": "between",
        "secondInput": {
          "value2": "5",
          "value1": "10",
          "arithmetic": "-"
        }
      },
      "agrColumn": "columnName",
      "conditions": [{
        "name": "columnName",
        "type": "text",
        "value": "value",
        "operator": "like"
      }],
      "table": "viewName",
      "aggregate": "SUM",
      "columnName": "columnName"
    }
  ]
}
```

Každý vytvořený SQL dotaz bude odpovídat jednomu sloupci v tabulce výsledků pro vykreslení grafu. Jak konstanty, tak proměnné budou přenositelné mezi projekty, aby je uživatel nemusel po každém přepnutí projektu nebo novém zapnutí aplikace definovat znovu.

V hlavním okně bude tlačítko pro načítání detekcí a po jeho stisknutí se do hlavní části okna načtou všechny vytvořené dotazy, které byly součástí

detekce, spolu s kritérii pro detekování (anti-)patternů. Dotazy bude možno upravovat a mazat pomocí tlačítek v daném sloupci dotazu. Při úpravě se otevře formulář pro vytváření a předvyplní se hodnoty.

6.1.3 Vytváření konstant

Pro vytváření konstant bude sloužit jednoduchý formulář (viz obr. 6.4), kam uživatel zadá název konstanty a její hodnotu. Pokud uživatel nezadá název proměnné, vygeneruje se univerzální název spolu s časovou značkou ve formátu `Constant_TIMESTAMP`. Formulář nebude rozlišovat typy hodnot, takže bude možno zadat jak text, tak čísla.

Konstanty se po vytvoření uloží do souboru, aby bylo možné je při zapnutí aplikace načíst bez nutnosti opětovného vytváření. Konstanty se budou moci za běhu upravovat(uživateli se objeví stejný formulář jako při vytváření) a na pozadí se po potvrzení úpravy přepíše i konkrétní soubor. Konstanty bude uživatel oprávněn i mazat, a pokud tak učiní, smaže se i příslušný soubor, který ke konstantě patří. Soubor, do kterého se konstanta uloží, bude ve formát JSON [5] a s následným formátováním:

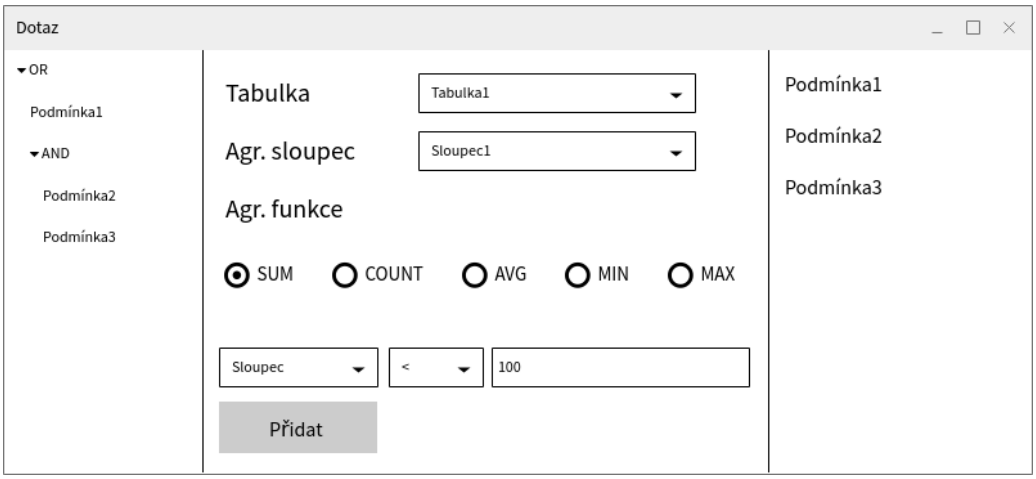
```
{
  "name": "constName",
  "value": "constValue"
}
```

Obrázek 6.4: Návrh okna pro vytváření konstant

6.1.4 Vytváření proměnných

Na vytváření proměnných bude poskytovat rozšíření aplikace formulář (viz obr. 6.5), kde uživatel vytvoří agregační SQL dotaz včetně podmínek a vý-

sledkem dotazu bude vždy číslo, nebo hodnota `null`, pokud je SQL dotaz sestaven špatně. Při vytváření podmínek v SQL dotazu bude moci uživatel použít již existující konstanty nebo proměnné. Po následném potvrzení formuláře se uživateli zobrazí hlavní okno rozšíření spolu se souhrnným zobrazením právě vytvořeného dotazu. SQL dotaz bude možno otestovat a při testu dotazu se uživateli zobrazí dialogové okno s výsledkem dotazu. Proměnnou bude možno pro větší přehlednost při použití pojmenovat. Pokud tak uživatel neučiní, vygeneruje se univerzální název spolu s časovou značkou uložení proměnné ve formátu `Variable_TIMESTAMP`. Takto definovanou proměnnou bude muset uživatel manuálně uložit kliknutím na příslušné tlačítko v dolní části okna. Vytvořené proměnné nebudou statické. Nebylo by uživa-



Obrázek 6.5: Návrh okna pro vytváření dotazů

telsky příjemné proměnné při každém přepnutí projektu znovu definovat, a proto budou proměnné nezávislé na projektu, takže jejich hodnota se může vzhledem k datům vybraného projektu měnit. Proměnné se po vytvoření uloží do souboru JSON stejně jako konstanty. Vytvořenou proměnnou bude možno upravovat skrze stejný formulář jako při vytváření, do kterého se zobrazí hodnoty upravovaného dotazu. JSON formát pro ukládání proměnných bude vypadat následovně:

```
{
  "queries": [
    {
      "joinColumn": "columnName",
      "agrColumn": "columnName",
      "conditions": [{
        "name": "columnName",
```

```
          "type": "number",
          "value": "value",
          "operator": ">"
        }],
        "table": "viewName",
        "aggregate": "SUM",
        "columnName": "columnName"
      }
    ]
}
```

6.1.5 Export výsledků

Výsledky dotazů pro nalezení (anti-)patternů bude nástroj schopen vyexportovat ve formátu CSV pro další použití v tabulkových nástrojích, jako je např. MS Excel. Bude se jednat o stejnou tabulku, jakou uvidí uživatel při zadávání kritérií detekce (anti-)patternu.

6.2 Implementace

Tato sekce popisuje implementaci rozšíření. Rozšíření je napsáno v jazyce Java a vychází z implementace nástroje SPADe GUI. Program byl vyvíjen ve vývojovém prostředí IntelliJ IDEA spolu s verzovacím nástrojem GitHub.

6.2.1 Použité knihovny

Rozšíření používá knihovny, které už jsou v aplikaci použity. Jedná se o knihovnu JFreeChart pro vykreslování grafů a knihovnu Swing pro vytváření grafického uživatelského rozhraní. K databázi se aplikace i rozšíření připojuje pomocí rohzraní JDBC.

Knihovna JFreeChart

JFreeChart [12] je bezplatná open source grafová Java knihovna, která vývojářům usnadňuje zobrazení grafů v jejich aplikacích. Rozsáhlá sada funkcí JFreeChart zahrnuje:

- konzistentní a dobře zdokumentované API podporující širokou škálu typů grafů

- flexibilní design, který lze snadno rozšířit a zaměřuje se na aplikace na straně serveru i na straně klienta
- podpora mnoha typů výstupů včetně komponent Swing a JavaFX, obrazových souborů (včetně PNG a JPEG) a formátů vektorových grafických souborů (včetně PDF a SVG)

Knihovna Swing

Java Swing je sada nástrojů pro vytváření grafického uživatelského rozhraní (GUI), která obsahuje bohatou sadu widgetů. Obsahuje balíček, který umožňuje vytvářet komponenty GUI pro Java aplikace a je nezávislý na platformě.

Knihovna Swing je postavena na Java Abstract Widget Toolkit (AWT), starší a na platformě závislé sadě nástrojů pro vytváření GUI. Díky knihovně Swing lze použít Java GUI komponenty, jako je tlačítko, textbox, checkbox atd. bez nutnosti vytvářet komponenty od nuly. [7]

6.2.2 Balík `databáze`

Do tohoto balíku je přidána třída `PohledDAO`, která slouží k výběru dat z databázových pohledů.

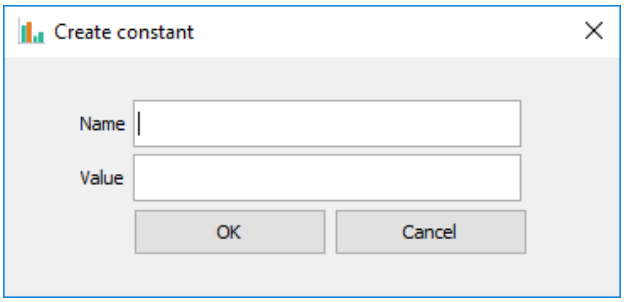
6.2.3 Balík `gui`

V tomto balíku se nachází většina úprav aplikace. Jsou zde přidány třídy pro nové okno a veškeré formuláře pro vytváření konstant, proměnných a dotazů.

Třída `FormCreateConstant`

Tato třída poskytuje formulář (viz obr. 6.6) pro vytváření konstant a dědí od třídy `JDialog`. Obsahuje atributy `name`, `value` a `wasCanceled`, který indikuje, jestli uživatel okno zavřel bez potvrzení formuláře. Třída využívá standardní `get` metody pro získání hodnot těchto atributů.

Do konstanty si může uživatel uložit cokoliv. Může se tedy stát, že při následném použití při vytváření detekcí nebude dotaz fungovat kvůli nesmyslné nebo neexistující hodnotě. Konstanty lze v aplikaci upravit změnou názvu nebo hodnoty. Pokud uživatel ponechá stejný název, přepíše se hodnota konstanty v aplikaci a v souboru, který k ní přísluší. Pokud uživatel smaže konstantu v aplikaci, smaže se soubor na disku, který ke konstantě patří.



Obrázek 6.6: Okno pro vytváření konstant

Třída `FormCreateQuery`

Třída uživateli zobrazí formulář (viz obr. 6.7) pro vytváření dotazů jak pro proměnné, tak pro detekce (anti-)patternů a dědí od třídy `JDialog`. Uživatel zde má možnost zadat, z jaké tabulky (pohledu) se budou data brát, přes jaký sloupec se bude tabulka pojit na osu X při vytváření (anti-)patternů, jaký sloupec bude sloužit pro agregační funkci, samotnou agregační funkci a podmínky spojené implicitně pomocí AND pro konečný SQL dotaz. Nabízené sloupce jsou vždy aktuální pro vybraný pohled a při přepnutí pohledu se formulář vrátí do původního stavu před zadáním hodnot, aby se co nejvíce zabránilo chybám při vytváření dotazů. Po potvrzení formuláře aplikace vytvoří dotaz pomocí obecné šablony, do které se za klíčová slova dosadí hodnoty z formuláře a vyplní se `id` projektu, který se právě zkoumán v aplikaci.

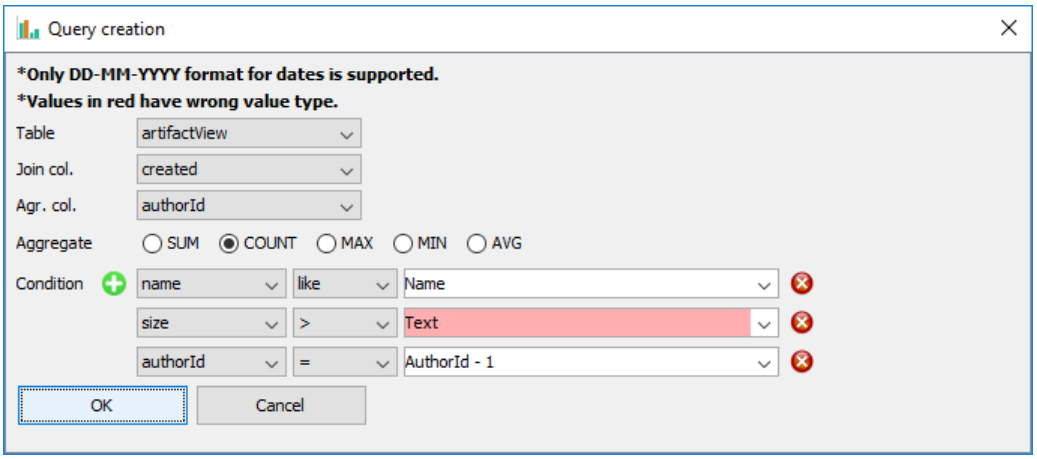
Šablona SQL dotazu vypadá následovně:

```
SELECT
    joinColumn, aggrFunc(aggrColumn)
FROM
    tableName
WHERE
    condition AND condition AND ...
    ... AND projectId = ?
GROUP BY
    joinColumn;
```

Stěžejní metodou v této třídě je metoda `getFormData`, která vrátí informace zadané ve formuláři v objektu JSON, ze kterého se ukládá do souboru ve stejném formátu, a skládá se z něj dotaz pro získání výsledné hodnoty dotazu. Třída dále obsahuje vnitřní třídu `ConditionPanel`, která reprezentuje podmínky při vytváření dotazu. Třída dále obsahuje metodu `resize`, která se stará o dynamické chování okna, jako je zvětšování a zmenšování výšky při přidávání a odebírání podmínek.

Vytvořené proměnné lze upravit, ale je třeba mít na paměti, že tato akce může ovlivnit výslednou hodnotu proměnné a tím případně i výsledek detekce anti-patternů v jiných projektech, než na kterém uživatel právě pracuje. Při úpravě je možné změnit i název proměnné. Pokud název zůstane stejný, soubor s proměnnou se přepíše, a pokud zadá uživatel název jiný, vytvoří se soubor nový. Tímto způsobem je uživatel schopen zachovat původní proměnnou, pokud by ji chtěl rozšiřovat, nebo lehce upravit její definici. Mazání proměnné funguje stejně jako u konstanty. Po smazání proměnné se z disku odstraní i soubor vázaný k proměnné.

Od původního návrhu se v implementaci ustoupilo z důvodu přílišné složitosti řešení a nepotřebě takto komplexní logiky pro demonstraci nového rozšíření.

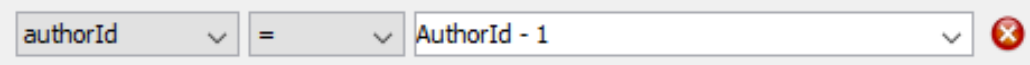


Obrázek 6.7: Okno pro vytváření dotazů

Vnitřní třída `ConditionPanel`

Tato třída obsahuje ovládací prvky (viz obr. 6.8) pro vytvoření podmínky a hlavním atributem je třída `Condition`, která reprezentuje jednu SQL podmínku při vytváření dotazů. Při přidání nové podmínky má uživatel možnost zadat sloupec, operátor a hodnotu pro vytvoření podmínky. Typy sloupců se načtou z databáze při spuštění aplikace a ty jsou poté pro příslušný pohled předány třídě pro vytváření dotazů. Operátory se vždy vztahují k typu sloupce, na který se podmínka vytváří, a není tedy možné použít aritmetiku při výběru sloupce, který v databázi obsahuje text a na opak. Tím je zajištěna částečná ochrana před špatně nadefinovanou podmínkou, která by vedla k chybě při spuštění dotazu. Třída `Condition` dále poskytuje metodu `validate`, která slouží k validaci uživatelem zadané hodnoty. Pokud

uživatel zadá nevalidní hodnoty, jako například text do pole, kde program očekává číslo, zobrazí se při potvrzení formuláře dialogové okno, které informuje uživatele o chybě, a špatně zadané hodnoty se podbarví červenou barvou. Formulář nepůjde potvrdit, dokud uživatel chyby neopraví. Tato validace se provádí pouze mezi typem sloupce a zadanou hodnotou. Pokud tedy například zadá, že chce, aby `size = -100`, jedná se o validní SQL podmínku, která ale pravděpodobně nic nevrátí vzhledem k tomu, že číselně vyjádřená velikost by neměla být záporná.



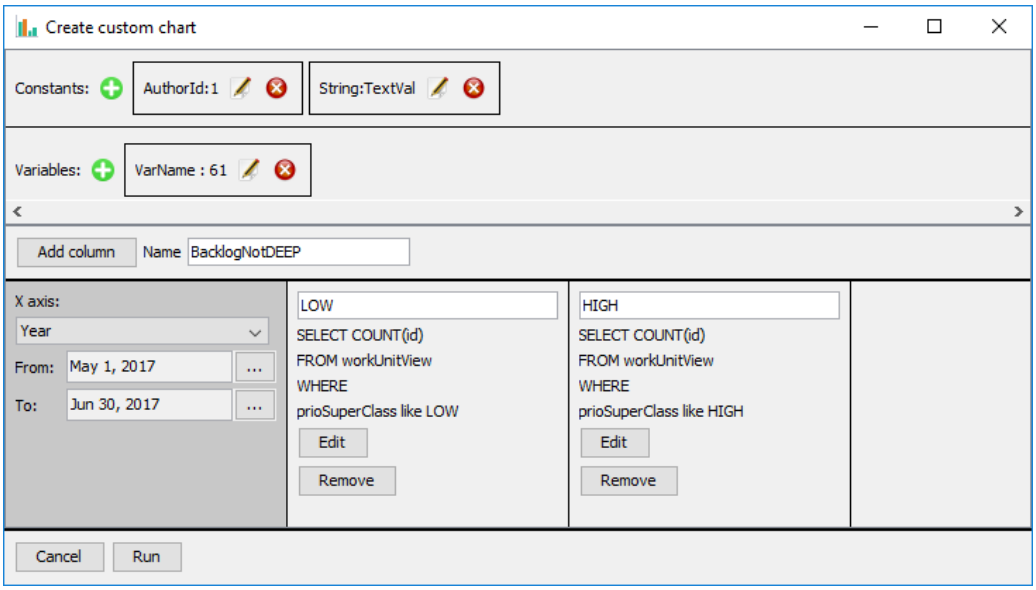
Obrázek 6.8: Panel podmínky

Třída `WindowCreatePatternDetection`

Tato třída reprezentuje hlavní okno rozšíření aplikace (viz obr. 6.9). V horní části okna je seznam s vytvořenými konstantami a proměnnými. Panel zobrazující konstanty a proměnné je „nekonečný“, takže pokud uživatel vytvoří například větší počet konstant než by se do okna vešlo, zobrazí se horizontální scrollbar a uživatel tak neztratí možnost konstanty a proměnné, které by byly mimo okno, upravovat a mazat.

Ve střední části okna je prostor pro vytváření dotazů. Střední část slouží jak pro vytváření proměnných, tak pro vytváření (anti-)patternů. Po spuštění dotazů při vytváření patternu se střední část překreslí na tabulku s možností zadávání kritérií pro detekci (anti-)patternu.

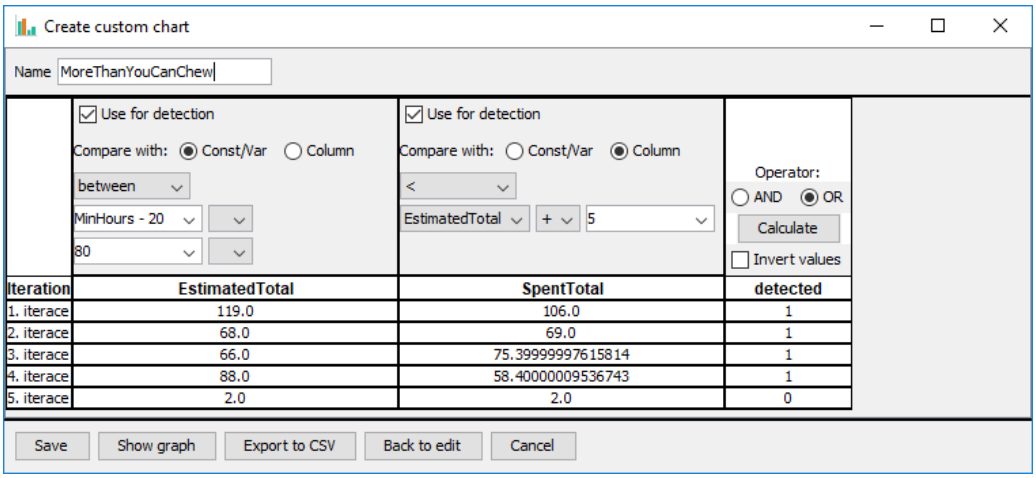
Dolní část okna obsahuje navigační a ovládací prvky při různých stavech hlavního okna. Při přechodu do okna rozšíření z hlavního okna se z aplikace přebere připojení k databázi a načtou se data o strukturách pohledů pro použití při vytváření dotazů. Data zůstanou načtena, dokud se okno nezavře. Pokud se tedy změní struktura pohledu v čase, kdy je otevřené okno aplikace, neproběhne aktualizace dat. Poté se ze souborů načtou uložené konstanty a proměnné, pokud nějaké existují, a vytvoří se příslušné panely v horní části okna. K vytvoření nové konstanty nebo proměnné slouží tlačítko se znaménkem plus, které otevře příslušný formulář `FormCreateConstant` (6.2.3) nebo `FormCreateQuery` (6.2.3). Uživatel má možnost vytvořit nový (anti-)pattern, nebo už vytvořený načíst ze souboru. Při vytváření (anti-)patternu se ve střední části okna zobrazí barevně odlišený sloupec, který reprezentuje výběr dat pro osu X při vykreslování do grafu a mapování hodnot, tlačítko na přidávání dotazů a textové pole pro zadání názvu (anti-)patternu. Uživatel si může vybrat ze tří hlavních skupin dat a těmi jsou iterace, členové



Obrázek 6.9: Hlavní okno rozšíření

projektu nebo zadání časového rozmezí rozděleného na dny, týdny, měsíce nebo roky. Pokud uživatel vybere časové rozmezí a nezadá žádné datum, přiřadí se na začátek intervalu datum začátku projektu a na konec intervalu aktuální datum.

Po výběru dat pro osu X může uživatel vytvářet dotazy. Každý vytvořený sloupec odpovídá jednomu SQL dotazu a zároveň sloupci při zadávání kritérií, a proto je nutné sloupce unikátně pojmenovat. Důvodem je použití stávající implementace vykreslování grafů, která počítá s tím, že každý sloupec zadaných dat bude mít unikátní jméno. Po zadání všech potřebných sloupců lze spustit zobrazení dat tlačítkem v dolní části okna. V tu chvíli se provedou všechny vytvořené dotazy, data z nich se namapují na data pro osu X a ve střední části okna se vykreslí tabulka s daty(viz obr. 6.10). V každém sloupci s číselnými hodnotami je záhlaví, které poskytuje možnost zapojit sloupec do počítání detekce. Dále lze zvolit, jestli se budou porovnávat hodnoty zadané uživatelem včetně použití proměnných a konstant, nebo se budou porovnávat hodnoty mezi jednotlivými sloupci. Ke každé hodnotě, ať už zadané, nebo hodnotě ze sloupce, lze pomocí aritmetických operátorů připojit hodnotu další a všechny zadané vstupy se porovnají s hodnotou ve sloupci pomocí relačního operátoru. Nakonec lze pomocí přepínače zvolit, jestli se mají výsledky porovnání spojovat pomocí AND nebo OR. Spočítání detekce se spustí po kliknutí na tlačítko v posledním sloupci. Výsledné hodnoty spočítané detekce lze invertovat. To poskytuje větší rozsah zadávání kritérií a rozšiřuje možnost modelování (anti-)patternů.



Obrázek 6.10: Okno pro zadávání kritérií detekce

Data z tabulky se dají zobrazit do grafu, který aplikace poskytuje. Uživatel si také může tabulku vyexportovat do souboru ve formátu CSV, vrátit se na předchozí stav vytváření dotazů nebo kompletně zrušit celé vytváření (anti-)patternu. Formát CSV dat vypadá následovně:

```
X axis , ColName1, ColName2, ..., ColNameN, Detection
axisData, colData , colData , ..., colData , detectData
axisData, colData , colData , ..., colData , detectData
```

Poslední možností je uložení (anti-)patternu do souboru. Při uložení se do souboru JSON zapíšou veškeré vytvořené dotazy a zadané hodnoty a kritéria. Je tak zajištěno, že se uživatel může případně vrátit k rozdělané práci, nebo v budoucnu upravit zadání. Takto uložený (anti-)pattern je přenositelný mezi projekty stejně jako proměnné, není proto potřeba jej pro každý projekt vytvářet znovu.

Vnitřní třída `ConstantPanel`

Třída slouží pro vykreslení informačního panelu vytvořené konstanty v horní části hlavního okna. Panel obsahuje tlačítka pro editaci a smazání konstanty. Při volbě editace se zobrazí formulář pro vytváření konstant s vyplněnými hodnotami upravované konstanty.

Vnitřní třída `VariablePanel`

Třída vytváří panely pro zobrazení vytvořených proměnných. V panelu jsou jako u konstant dvě tlačítka pro editaci a smazání proměnné. Pokud se uživatel rozhodne upravovat proměnnou, zobrazí se mu formulář pro vytváření

dotazů a do něj se vyplní hodnoty upravované proměnné.

Vnitřní třída `QueryPanel`

Třída zobrazuje panel se souhrnnými informacemi vytvořeného dotazu přes formulář třídy `FormCreateQuery`. Panel obsahuje dvě tlačítka, jedno pro editaci dotazu a jedno pro smazání panelu z hlavního okna. Pokud se uživatel rozhodne dotaz editovat, zobrazí se formulář pro vytváření dotazů s vyplněnými hodnotami upravovaného dotazu.

Třída `PanelDetectionColumn`

Tato třída vykresluje data do sloupce tabulky při zadávání kritérií pro detekce (anti-)patternů. V aplikaci se rozlišují tři typy sloupců. První na vypsání dat pro osu X, druhý se záhlavím pro zadávání kritérií a třetí pro detekční sloupec s možností invertování hodnot.

6.2.4 Balík `ostatni`

V balíku ostatní jsou přidány třídy `ComboBoxItem`, `CustomComboBoxEditor`, `Condition`, `Iteration` a `ViewEnum`. Jedná se o pomocné třídy pro uložení dat a nastavení ovládacích prvků v hlavním okně rozšíření aplikace.

6.2.5 Vytvořené pohledy

Pro jednodušší vytváření dotazů jsou v databázi vytvořeny pohledy reprezentující logické celky pro snadnější získání dat.

`ArtifactView`

Pohled spojuje záznamy z tabulek `work_item`, `artifact` a `personView` (viz tab. 6.1). Obsahuje veškeré potřebné informace o vytvořených artefaktech v projektu, jako je např. `name`, `description`, `mimeType`, reprezentující typ souboru artefaktu, `authorName` a další.

`ConfigurationView`

Tento pohled poskytuje informace o konfiguracích v projektech a o tom, kdo změnu konfigurace provedl. Hlavní tabulkou je `work_item`, na kterou jsou napojeny `configuration`, `configuration_person_relation` a pohled `personView`, jak popisuje tabulka 6.2.

Sloupec pohledu	Původní tabulka	Původní sloupec
name	work_item	name
description	work_item	description
created	work_item	created
url	work_item	url
artifactClass	artifact	artifactClass
mimeType	artifact	mimeType
size	artifact	size
authorId	personView	id
authorName	personView	name
projectId	personView	projectId

Tabulka 6.1: Vytváření sloupců pohledu `ArtifactView`

Sloupec pohledu	Původní tabulka	Původní sloupec
id	work_item	id
type	work_item	workItemType
name	work_item	name
description	work_item	description
created	work_item	created
authorId	personView	id
authorName	personView	name
relationName	configuration_person_relation	name
relatedId	personView	id
relatedName	personView	name
projectId	configuration	projectId

Tabulka 6.2: Vytváření sloupců pohledu `ConfigurationView`

`CommittedConfigView`

Pohled rozšiřuje pohled `configurationView` o tabulku `committed`, která poskytuje informace o tom, kdy byla konfigurace upravena (viz tab. 6.3).

Sloupec pohledu	Původní tabulka	Původní sloupec
*	configurationView	*
committed	committed_configuration	committed

Tabulka 6.3: Vytváření sloupců pohledu `CommittedConfigView`

`CommitView`

Pohled dále rozšiřuje pohled `committedConfigView` o tabulku `commit`, `tag` a `branch` a tím dává dohromady ucelenou informaci o commitech projektu. Přidává k záznamům název tagu a větve(branch) a indikátor toho, jestli je větev hlavní nebo ne (viz tab. 6.4).

Sloupec pohledu	Původní tabulka	Původní sloupec
*	commitedConfigView	*
isRelease	commit	isRelease
tag	tag	name
branch	branch	name
main	branch	isMain

Tabulka 6.4: Vytváření sloupců pohledu `CommitView`

`FieldChangeView`

`FieldChangeView` vychází z pohledu `configurationView` a připojuje tabulky `configuration_change`, `work_item_change` a `field_change`, jak je vidět v tabulce 6.5. Doplňuje informace o tom, kdy byla konfigurace upravena, jaká byla stará a nová hodnota a jaký `work_item` se upravil.

`PersonWithRolesView`

Pohled obsahuje záznamy z tabulky `person` a připojují s k ní tabulky `role` a `role_classification` (viz tab. 6.6). Tím poskytuje kompletní informace o členech týmu.

Sloupec pohledu	Původní tabulka	Původní sloupec
*	configurationView	*
changeName	work_item_change	name
changeDesc	work_item_change	description
itemId	work_item	id
itemType	work_item	workItemType
itemName	work_item	name
itemDesc	work_item	description
itemCreated	work_item	created
field	field_change	name
newValue	field_change	newValue
oldValue	field_change	oldValue

Tabulka 6.5: Vytváření sloupců pohledu `FieldChangeView`

Sloupec pohledu	Původní tabulka	Původní sloupec
id	person	id
name	person	name
role	role	name
roleClass	role_classification	class
roleSuperClass	role_classification	superClass
projectId	person	projectId

Tabulka 6.6: Vytváření sloupců pohledu `PersonWithRolesView`

`PersonView`

Pohled obsahuje záznamy z tabulky `person` a nenapojují se na ní další tabulky (viz tab. 6.7). Původním záměrem bylo připojit k ostatním pohledům pohled `personWithRoles`. Vzhledem ke struktuře datového modelu vracely ostatní pohledy, které tento pohled používaly, duplicitní výsledky. Proto byl tento pohled nahrazen pohledem `personView`.

Sloupec pohledu	**Původní tabulka**	**Původní sloupec**
id	person	id
name	person	name
projectId	person	projectId

Tabulka 6.7: Vytváření sloupců pohledu `PersonView`

`WorkUnitView`

Work unit představuje v datovém modelu ticket z ALM nástroje. Tento pohled je nejkomplexnější ze všech a obsahuje souhrnné informace o ticketu, jako je například, do jaké iterace a fáze patří, zakladatel ticketu a ten, kdo ho má přiřazen, datum vytvoření a splnění, odhadovaný a fakticky strávený čas na ticketu atd. Všechny tabulky potřebné na vytvoření pohledu lze najít v tabulce 6.8.

Sloupec pohledu	**Původní tabulka**	**Původní sloupec**
id	work_item	id
name	work_item	name
description	work_item	description
created	work_item	created
dueDate	work_unit	dueDate
estimatedTime	work_unit	estimatedTime
progress	work_unit	progress
spentTime	work_unit	spentTime
startDate	work_unit	startDate
projectId	work_unit	projectId
authorId	personView	id
authorName	personView	name
assigneeId	personView	id
assigneeName	personView	name
activityName	activity	name
activityDesc	activity	description

activityEndDate	activity	endDate
activityStartDate	activity	startDate
iterationName	iteration	name
iterationDesc	iteration	description
iterationStartDate	iteration	startDate
iterationEndDate	iteration	endDate
iterationCreated	iteration	created
phaseName	phase	name
phaseDesc	phase	description
phaseStartDate	phase	startDate
phaseEndDate	phase	endDate
phaseCreated	phase	created
priorityName	priority	name
priorityDesc	priority	description
prioClass	priority_classification	class
prioSuperClass	priority_classification	superClass
severityName	severity	name
severityDesc	severity	description
severityClass	severity_classification	class
severitySuperClass	severity_classification	superClass
resolutionName	resolution	name
resolutionDescription	resolution	description
resolutionClass	resolution_classification	class
resolutionSuperClass	resolution_classification	superClass
statusName	status	name
statusDescription	status	description
statusClass	status_classification	class
statusSuperClass	status_classification	superClass
wuTypeName	wu_type	name
wuTypeDescription	wu_type	description
wuTypeClass	wu_type_classification	class
categoryName	category	name
categoryDesc	category	description

Tabulka 6.8: Vytváření sloupců pohledu `WorkUnitView`

6.3 Testování

Během vývoje byl kód aplikace testován pomoci jednotkových (JUnit) testů. Testů je napsáno 23 a pokrývají 45% kódu rozšíření.

Pro uživatelské testování bylo osloveno deset testerů, kde polovina byla obeznámena s původní aplikací a nástrojem SPADe a druhá polovina neznala nástroj vůbec. Všem testerům byl poskytnut návod na ovládání rozšíření s popisem nových funkčností. Úkolem testerů bylo vyzkoušet uživatelskou přívětivost, intuitivnost ovládání a otestovat nové funkce, jako je vytvoření proměnné, (anti-)patternu a detekce.

Během testování bylo nalezeno několik vizuálních a dvě funkční chyby. První funkční chyba byla nemožnost zavření vyskakovacího okna a druhá smazání názvu sloupce při jeho editaci, který musel být následně znovu vyplněn. Obě funkční a všechny vizuální chyby byly opraveny. Dva testeři dále navrhovali doplnit text vyskakovacího okna při exportu CSV o cestu k souboru a čtyři by považovali za užitečné moci potvrdit formuláře pomocí klávesy `enter`.

U testu ovladatelnosti a přehlednosti se čtyři testeři, ti, kteří nevěděli k čemu nástroj SPADe slouží, shodli na tom, že aplikace není příliš intuitivní na ovládání. I s návodem bylo prý složité porozumět konceptu aplikace a účelu celého rozšíření. Tento výsledek lze připsat nedostatečné znalosti problematiky, kterou se nástroj SPADe zabývá, a k čemu má být určený.

Akceptační testování bylo prováděno vedoucím práce v průběhu celého vývoje.

6.3.1 Rozsah změn implementace

Ke zjištění rozsahu implementace byl použit plugin Statistic[1] nainstalovaný do vývojového prostředí IntelliJ IDEA[2] a pomocí něho byl porovnán poslední commit původní implementace a poslední commit nové implementace. Původní implementace měla 8397 řádek zdrojového kódu a nová implementace 11687 řádek. V nové implementaci přibylo 3290 řádek zdrojového kódu a 9 řádek bylo smazáno. Oproti 74 zdrojovým souborům ve výchozí verzi aplikace má nová implementace souborů 92, přičemž 32 původních souborů bylo změněno.

[1] https://plugins.jetbrains.com/plugin/4509-statistic
[2] https://www.jetbrains.com/idea/

6.3.2 Budoucí rozšíření

Jelikož se jedná o první iteraci rozšíření na zadávání uživatelských dotazů pro tvorbu detekcí (anti-)patternů, počítá se u aplikace s budoucím rozšířením.

Jedním z prvních rozšíření by mohlo být přidání dalších logických operátorů při vytváření podmínek a při vytváření kritérií pro detekce (anti-)patternů. Pokud by se toto rozšíření implementovalo, bylo by dobré přidat možnost uzavřít například podmínky do závorek pro ještě širší škálu využití. To samé by platilo při vytváření kritérií. Jednou z možností by bylo implementovat vizuální stromovou strukturu podmínek, do které by se jednotlivé podmínky přetahovaly pomocí drag-and-drop, a nebo vytvořit parser, který by podmínky postupně zpracoval.

Další možností by bylo přidání automatického provedení detekce při jakékoliv změně dat při zadávání kritérií, což by přineslo větší uživatelskou přívětivost.

Zobrazování grafů by se dalo rozšířit o větší výběr typů grafů a o možnost exportování do obrázku pro případné analýzy a předvádění výsledků.

Dále by bylo možné upravit pohledy nebo datový model tak, aby bylo možné použít pohled `personWithRolesView`, a nebo upravit načítání dat tak, aby se nemusely pohledy využívat vůbec a dotazy by se skládaly přímo z tabulek datového modelu.

7 Experiment

Tato kapitola popisuje projekty a praktiky vybrané k testování, průběh a výsledky testování. Na konci kapitoly je diskuze k výsledkům celého experimentu. Experiment probíhal ve třech fázích.

V první fázi se vybraly projekty, na kterých se nové rozšíření testovalo. Vzhledem k tomu, že se nástroj vyvíjí na ZČU, zkoumaly se školní projekty vypracované studenty v rámci předmětu Pokročilé softwarového inženýrství (KIV/ASWI). Následoval výběr anti-patternů (viz 7.1.2). Vzhledem k tomu, že existují jistá omezení v implementaci a natěžených datech v nástroji, vybraly se praktiky (ne nutně komplexní anti-patterny), které lze v aplikaci namodelovat a mají dostatečnou vypovídající hodnotu při řízení vývoje software.

Praktiky se upravily podle potřeb projektů, které probíhaly iteračně. Poté se vybrané praktiky namodelovaly (viz 7.2), spustily na projektech a výsledky se zpracovaly do tabulky (viz tab. 7.1).

Poslední fáze experimentu zahrnovala manuální analýzu praktik v ALM nástroji, který studenti při vývoji používali. Výsledky kontroly lze najít v tabulce 7.2, kde jsou barevně označeny hodnoty, ve kterých se manuální kontrola lišila.

7.1 Výběr dat

7.1.1 Projekty

V době testování rozšíření aplikace jsou v datovém skladu SPADe nahrána data ze školních projektů z předmětu ASWI vyučovaném na Katedře informatiky a výpočetní techniky. Při projektech, které žáci během semestru vytvářeli, používali různé praktiky z různých metodik a ALM nástroj Redmine. Je tedy možnost vyzkoušet škálu různých praktik. Testování dat ze školních projektů přinese také hodnotné informace pro výuku v budoucích letech.

7.1.2 Obecné praktiky

Jelikož nástroj ještě nezvládne tak detailní vytváření dotazů, aby bylo možné modelovat (anti-)patterny, testují se v této práci praktiky, které namodelovat lze.

Zde je popis vybraných praktik:

Poor Work Decomposition

Zavedenou praktikou je rozdělovat velké úkoly na menší části. Úkol, který má více než 8h, by se měl rozdělit na menší, lépe popsatelné úkoly. Lze tak práci lépe pochopit a případně rozdělit mezi více členů týmu. Tento jev sleduje rozložení práce a rozpad časově náročných úkolů.

The Flash

Členové týmu využívající ALM nástroje by měli řádně pracovat s úkoly a správně měnit jejich informace, jako je například stav úkolu. Zde se sleduje, jestli v projektech existují úkoly se stavem CLOSED, ale nulovým odpracovaným časem.

Continuous Documentation

V iterativních procesech není dokumentace statická, ale mění se v průběhu projektu a měla by se neustále upravovat a zpřesňovat. Tato praktika sleduje aktivní práci s artefakty, kromě souborů zdrojového kódu. Jedná se například o dokumentace a wiki stránky projektů.

More Than You Can Chew

Iterace jsou časově ohraničené a lze předem spočítat, kolik hodin na iteraci tým má. Obvykle se počítá s 8 hodinami na člověka na den a podle toho by se měla plánovat časová náročnost a počet úkolů v iteraci. Tuto hodnotu není dobré překračovat, a proto tato praktika detekuje naplánování úkolů v iteraci a sleduje odhadovaný a skutečně strávený čas při plnění každého úkolu.

Podobným případem je praktika **Underutilization**, kde figuruje nedostatečné zapojení členů týmu při špatně naplánované iteraci, kde je úkolů málo a čas se dostatečně nevyužije.

Backlog not DEEP

Produktový backlog se v průběhu projektu neustále mění a měla by se dodržovat zásada DEEP. To znamená, že úkoly v backlogu mají být seřazeny podle priorit, dostatečně popsány v dané situaci, má být odhadnuta časová náročnost a nemá být statický, jelikož v průběhu se budou úkoly přidávat,

odebírat a měnit. Praktika zde detekuje část této zásady sledováním priorit u úkolů, jejich popisem a odhadovaným časem.

Backlog Grooming

Backlog grooming je setkání, kde členové týmu přezkoumají backlog, aby zajistili mimo jiné i zásadu DEEP. Úkoly se prochází podle priorit a připravuje se práce na další sprinty. Přidávají se nové úkoly nebo se staré odebírají, upravují se odhady, rozdělují se velké úkoly na menší atd. Jedná se o dobrou praktiku, kde se sledují úpravy úkolů v backlogu v určitém časovém období. Zkoumají se změny v prioritách, odhadech, přesunutí do jiné iterace a přiřazení úkolu některému ze členů týmu.

Patchy Engagement

Při práci v týmu se očekává, že se všichni zapojí do práce stejně. Rozdělení práce je nedílnou součástí jakéhokoliv vývoje software. Zajistí se tak plynulý průběh a odstraní se případné neshody mezi jednotlivými členy týmu. Praktika zkoumá rozdělení práce pomocí sledování commitů jednotlivých členů týmu v daném časovém období.

Poor Ticket Workflow

Pokud tým využívá některý ALM nástroj, je potřeba aby s ním všichni uměli pracovat a dodržovali zavedený pracovní postup. Úkoly v průběhu implementace mění svůj stav podle předem domluvených konvencí. Pomáhá to s plánováním a odhadováním budoucí práce. Tato praktika sleduje postupné přiřazování stavů úkolu v ALM nástroji.

7.2 Modelování praktik

V rozšíření se vytvořily modely na základě popisů praktik uvedených v sekci 7.1.2. Informace o modelech obsahují data pro osu X, popis vytvořených dotazů (sloupce) a popis kritérií pro detekci.

Poor Work Decomposition

Data pro osu X jsou iterace projektu a praktika obsahuje dva sloupce. Jeden s počtem work unitů (tasků) s odhadnutým časem větším než 8h a druhý s celkovým počtem work unitů v iteraci. Detekce je pozitivní, pokud je počet

work unitů s odhadnutým časem větší než 10% všech work unitů v iteraci, a nebo je počet work unitů v iteraci menší než 12.

- **Osa X:** Iterace
- **Sloupec1:** Počet work unitů v iteraci s odhadovaným časem větším než 8h
- **Sloupec2:** Počet work unitů v iteraci
- **Detekční kritérium:** Sloupec1 > Sloupec2*0.1 OR Sloupec2 < 12

The Flash

U této praktiky jsou na ose X jednotliví členové týmu. V jednom sloupci se sleduje stav work unitů u každého člena a sčítají se úkoly, které mají strávený čas 0h a jejich statusSuperClass je CLOSED a statusClass není DELETED a zároveň není INVALID. Detekce je pozitivní, pokud je součet větší než 0.

- **Osa X:** Členové týmu
- **Sloupec1:** Počet work unitů, kde statusSuperClass je CLOSED, statusClass není DELETED ani INVALID a strávený čas na úkolu je 0h.
- **Detekční kritérium:** Sloupec1 > 0

Continuous Documentation

Zde jsou na ose X jednotlivé kalendářní dny. Detekování je rozděleno do několik sloupců, které sčítají vytvořené artefakty s příponami (mimeType) jako doc, docx, pdf a txt. Detekce zde není provedena, jelikož se jedná pouze o sběr souhrnných dat do výsledkové tabulky.

- **Osa X:** Kalendářní dny
- **Sloupec1:** Počet artefaktů s mimyType docx
- **Sloupec2:** Počet artefaktů s mimyType doc
- **Sloupec3:** Počet artefaktů s mimyType txt
- **Sloupec4:** Počet artefaktů s mimyType pdf
- **Detekční kritérium:** Neprovádí se

More Than You Can Chew

Data pro osu X jsou iterace projektu. Praktika obsahuje dva sloupce, kdy jedním je součet odhadovaných časů (estimatedTime) u všech úkolů pro danou iteraci a druhý je skutečně strávený čas (spentTime) na těchto úkolech. Detekce je pozitivní, pokud je jeden ze součtů času v nějaké iteraci větší než 80h.

- **Osa X:** Iterace
- **Sloupec1:** Součet estimatedTime pro work unity iterace
- **Sloupec2:** Součet spentTime pro work unity iterace
- **Detekční kritérium:** Sloupec1 $>$ 80 OR Sloupec2 $>$ 80

U **Underutilization** je model stejný, jen se zde sleduje a detekuje, jestli součty odhadnutých a strávených hodin na úkolech jsou menší než 20h, čímž se odhalí špatně naplánovaná nebo odhadnutá práce.

- **Detekční kritérium:** Sloupec1 $<$ 20 OR Sloupec2 $<$ 20

Backlog not DEEP

Praktika používá na ose X roky, po který projekt běží, a čtyři sloupce obsahují počet work unitů s prioritou low, high, high a odhadovaným časem 0h a high a neexistujícím popisem úkolu. Detekce je pozitivní, pokud u prvních dvou sloupců je počet 0 a pro třetí a čtvrtý sloupec je počet větší než 0.

- **Osa X:** Kalendářní dny
- **Sloupec1:** Počet artefaktů s prioritou Low
- **Sloupec2:** Počet artefaktů s prioritou High
- **Sloupec3:** Počet artefaktů s prioritou High a estimatedTime = 0
- **Sloupec4:** Počet artefaktů s prioritou High a prázdným popisem
- **Detekční kritérium:** Sloupec1 = 0 OR Sloupec2 = 0 OR Sloupec3 $>$ 0 OR Sloupec4 $>$ 0

Patchy Engagement

Tento jev má na ose X týdny kalendářního roku a každý sloupec představuje jednoho člena týmu. U každého člena se poté spočítá počet committed configuration. Detekce je pozitivní, pokud pro jakoukoliv kombinaci autor/týden je políčko v tabulce 0. Tento model je zatím jako jediný nepřenositelný, jelikož jednotlivé sloupce jsou vázané na konkrétní jméno člena týmu.

- **Osa X:** Kalendářní týdny
- **Sloupec1:** Počet committed configuration, kde jméno autora je ...
- **SloupecN:** Počet committed configuration, kde jméno autor je ...
- **Detekční kritérium:** Sloupec1 = 0 OR ... OR SloupecN = 0

Poor Ticket Workflow

Data pro osu X jsou iterace projektu. Kontrola stavů probíhá v pěti sloupcích, kde se sčítají výskyty záznamů field change, jejichž status se změnil na resolved, verified, accepted, in progress nebo z hodnoty new ihned na closed, kde příslušný work unit nemá v názvu řetězce „schůz“, „meeting“, „stand“, „revi“, „retro“. Detekce je pozitivní, pokud v jednom ze sloupců je výsledná hodnota 0.

- **Osa X:** Iterace
- **Sloupec1:** Počet field change na work unitech, kde nová hodnota statusu je resolved
- **Sloupec2:** Počet field change na work unitech, kde nová hodnota statusu je verified
- **Sloupec3:** Počet field change na work unitech, kde nová hodnota statusu je accepted
- **Sloupec4:** Počet field change na work unitech, kde nová hodnota statusu je in progress
- **Sloupec5:** Počet field change na work unitech, kde stará hodnota statusu je new a nová closed a název work unitů neobsahuje řetězec „schůz“
- **Sloupec6:** Počet field change na work unitech, kde stará hodnota statusu je new a nová closed a název work unitů neobsahuje řetězec „meeting“

- **Sloupec7:** Počet field change na work unitech, kde stará hodnota statusu je new a nová closed a název work unitů neobsahuje řetězec „stand“
- **Sloupec8:** Počet field change na work unitech, kde stará hodnota statusu je new a nová closed a název work unitů neobsahuje řetězec „revi“
- **Sloupec9:** Počet field change na work unitech, kde stará hodnota statusu je new a nová closed a název work unitů neobsahuje řetězec „retro“
- **Detekční kritérium:** Sloupec1 = 0 OR Sloupec2 = 0 OR Sloupec3 = 0OR Sloupec4 = 0 OR Sloupec5 = 0 OR Sloupec6 = 0 OR Sloupec7 = 0OR Sloupec8 = 0 OR Sloupec9 = 0

Backlog Grooming

Data pro osu X jsou kalendářní dny, po které trval projekt. Ve čtyřech sloupcích se sčítají záznamy field change, kde se úkolu mění priorita, iterace, odhadovaný čas nebo člen týmu zodpovědný za splnění úkolu.

Momentálně se jev nedá detekovat a výsledky jsou pozorovatelné ve výsledné tabulce. Pravidla detekce jsou zatím mimo možnosti rozšíření aplikace.

- **Osa X:** Kalendářní dny
- **Sloupec1:** Počet field change s názvem „priority“
- **Sloupec2:** Počet field change s názvem „iteration“
- **Sloupec3:** Počet field change s názvem „estimate“
- **Sloupec4:** Počet field change s názvem „assignee“
- **Detekční kritérium:** Neprovádí se

7.3 Výsledky

V této sekci se nachází výsledky kontroly pomocí aplikace v tabulce 7.1 a výsledky manuální kontroly v tabulce 7.2. V tabulce manuální kontroly jsou červeně označeny hodnoty, které se oproti výsledkům z aplikace lišily.

Praktika	**Projekt1**	**Projekt2**	**Projekt3**	**Projekt4**	**Projekt5**	**Projekt6**	**Projekt7**
Backlog Not DEEP	✓	✓	×	✓	✓	✓	✓
More Than You Can Chew	✓	✓	✓	✓	✓	×	×
Patchy Engagement	✓	✓	✓	✓	✓	✓	✓
Poor Ticket Workflow	✓	✓	✓	✓	✓	✓	✓
Poor Work Decomposition	✓	✓	✓	✓	×	✓	✓
The Flash	✓	×	×	×	×	✓	×
Underutilization	✓	×	✓	×	×	✓	✓

Tabulka 7.1: Výsledky detekce přes aplikaci

Praktika	**Projekt1**	**Projekt2**	**Projekt3**	**Projekt4**	**Projekt5**	**Projekt6**	**Projekt7**
Backlog Not DEEP	✓	✓	×	✓	✓	✓	✓
More Than You Can Chew	✓	✓	×	×	×	×	×
Patchy Engagement	✓	✓	✓	✓	✓	✓	✓
Poor Ticket Workflow	✓	✓	✓	✓	✓	✓	✓
Poor Work Decomposition	✓	✓	×	✓	×	✓	✓
The Flash	✓	×	×	×	×	✓	×
Underutilization	✓	×	×	×	×	✓	✓

Tabulka 7.2: Výsledky manuální detekce

7.4 Diskuze

Detekce praktik v aplikaci byla úspěšná na 89,8%. Výsledky manuální kontroly se oproti výsledkům z aplikace lišily u tří praktik – **More Than You Can Chew**, **Underutilization** a **Poor Work Decomposition**. Po prozkoumání se ukázalo, že chyba je v datech datového skladu. Všechny tři praktiky čerpají data z pohledu `workUnitView` a na ose X mají iterace. V datech existují úkoly, které mají jiné datum začátku než datum začátku iterace, do které patří, a tak úkoly spadají do jiných iterací, než lze vidět v ALM nástroji. Toto špatné rozdělení zkresluje data, a proto se výsledky lišily.

Problémem i tak zůstávají limity celého procesu. Data se z ALM nástrojů se nahrávají do unifikovaného modelu datového skladu a může se stát, že při přenosu a konverzi vzniknou v datech chyby. Dalším faktorem jsou lidé používající ALM nástroje, kteří můžou nástroj používat špatně nebo jen zadávat chybná data. Například praktika Poor Ticker Workflow zkoumala výskyty řetězců v názvech úkolů. Problém by pak nastal, pokud by projekt měl zavedenou konvenci na pojmenování úkolů a ostatní informace byly vkládány do popisu úkolu. Model praktiky by pak nebyl schopen správně detekovat chybu.

Modelování funguje do určité úrovně složitosti, a pokud je praktika dobře namodelovaná, dokáže posloužit jako indikátor potencionálně slabého místa. Jednotlivé výsledky se lehce lišily, i tak ale experiment prokázal, že rozšíření je i ve své zatím omezené podobě použitelné, a pokud se bude nástroj nadále vyvíjet, bude nápomocné při používání nástroje SPADe.

Stále ale platí, že i kdyby byla data v pořádku a praktiky namodelovány perfektně, není účelem nástroje SPADe anti-patterny odstranit, ale upozornit uživatele, že se v projektu může nacházet problém.

Jelikož nebyly výsledky nalezené aplikací kvůli chybě v datech průkazné na 100%, jsou zde popsány jednotlivé praktiky a nalezené důvody jejich výskytu při manuální kontrole:

Backlog Not DEEP

Chybou u této praktiky bylo u téměř všech nálezů nevyplnění popisu úkolů s vysokou prioritou. Popis úkolu byl většinou přímo v názvu. Jednalo se úkoly, jako například „Předání produktu zákazníkovi“, „Navrhnout strukturu GUI“, „Schůzka se zadavatelem“ apod.

More Than You Can Chew

Zde se u téměř všech projektů jednalo o první nebo druhou iteraci. Důvodem mohla být povaha projektu, kdy chtěli studenti stihnout co nejvíce práce co nejdříve. Výjimku tvořily úkoly typu „Seznámení s dosavadní implementací“, kterým studenti odhadli souhrnně například 20 hodin, místo aby úkol rozdělili na menší pro jednotlivé členy.

Patchy Engagement

Z ALM nástroje bylo vidět, že týmy se často rozdělily na programátory a většinou jednoho člověka, který psal dokumentace a moc do kódu nepřispíval. Dalším důvodem mohla být povaha vývoje projektu. Jelikož se jednalo o studenty, kteří mají více předmětů v semestru, bylo z dat vidět, že se v práci na projektu střídali, pravděpodobně kvůli časové náročnosti jiných předmětů.

Poor Ticket Workflow

Hlavním důvodem pro výskyt tohoto jevu bylo nepoužívání stavu verified a in progress. Problém nastával i při porovnávání řetězců v názvu úkolu. Občas byly úkoly pojmenovány jinak a model je nedokázal zachytit.

Poor Work Decomposition

Špatné rozdělení práce se objevovalo na začátku projektů, kde byly zadány úkoly jako „Seznámení s aplikací“, „Příprava architektury“ atd.

The Flash

U této praktiky byl důvod vždy stejný a tím bylo nezalogování času u úkolů trvajících například jen 15 minut a označení úkolu za hotový.

Underutilization

Tento jev se vyskytoval jednou na začátku projektu, kdy v první iteraci bylo jen několik úkolů na seznámení se s aplikací a zadáním a jinak spíše na konci projektu, kdy v iteraci probíhalo poslední testování a předával se hotový produkt.

8 Závěr

Cílem této diplomové práce byla identifikace procesních chyb v softwarovém vývoji a vytvoření jejich modelů pomocí nástroje SPADe. Za tímto účelem bylo potřeba prostudovat problematiku softwarového vývoje a (anti-)patternů (kapitola 2 a 3). Dále bylo nutné se seznámit s disciplínou Application Lifecycle Management a ALM nástroji (kapitola 4), samotným nástrojem SPADe, jeho uživatelským rozhraním a datovým modelem (kapitola 5) a navrhnout a rozšířit nástroj SPADe GUI o vytváření modelů praktik (kapitola 6). Posledním bodem bylo vybrat projekty, vhodné praktiky, které lze v rozšíření namodelovat, a provést experiment hledání praktik. Hledání proběhlo přes nově upravený nástroj SPADe a pomocí manuálního procházení dat v ALM nástroji. Poté byly popsány výsledky a diskuze nad nimi (kapitola 7).

Změny vytvořené v této práci tvoří 28,2% nové implementace a aplikace byla též funkčně i uživatelsky otestována. Při testech se nenalezly žádné závažné chyby. Během experimentu prováděném v této práci se zkoumalo sedm praktik a sedm projektů. Nově vytvořené rozšíření se oproti manuální kontrole lišilo v 10,2% případů. Ve všech případech byla chybně natěžená data z ALM nástrojů do datového skladu SPADe. Jednalo se o úkoly jejichž datum začátku se neshodovalo s datem začátku iterace, do které úkoly patřily. Tato chyba zkreslila rozložení práce v jednotlivých iteracích. Pokud by byla data natěžena správně, odhalila by aplikace všechny zkoumané praktiky bez chyby.

Výstupy této práce představují výsledky při hledání špatných praktik ve vybraných projektech a implementace rozšíření aplikace SPADe s jeho popisem, dokumentací ve zdrojovém kódu a uživatelským manuálem v příloze B. Nástroj v současném stavu představuje pomoc na základní úrovni při hledání (anti-)patternů v projektových datech. I přes některé nedostatky, které jsou popsané v této práci, implementovaná rozšíření výrazně zvyšují použitelnost nástroje pro jeho primární účely, tj. snadné definování procesních vzorů a jejich vyhledávání v databázi projektových dat. Provedený experiment může při pouhé obměně detekovaných vzorů a sady projektů sloužit jako šablona pro další experimenty, ať už za účelem lazení nástroje SPADe a všech jeho komponent, využití ve výuce, nebo přímo pro vědecké studie v oblasti dopadu procesních vzorů na úspěšnost vývojových projektů a kvalitu jejich výstupních softwarů.

Zadání práce bylo splněno v celém rozsahu.

Literatura

[1] Atlassian. *What is version control* [online]. [cit. 2019/04/17]. Dostupné z: https://cs.atlassian.com/git/tutorials/what-is-version-control#benefits-of-version-control.

[2] Brown, W. J. et al. *AntiPatterns (Refactoring Software, Architectures, and Projects in Crisis).* John Wiley & Sons, Inc., 1998. ISBN 0-471-19713-0.

[3] Chambers & Associates. *Activity* [online]. [cit. 2019/04/10]. Dostupné z: http://www.chambers.com.au/glossary/activity.php.

[4] Cohen, E. *The Definitive Guide to Project Management Methodologies* [online]. 2017. [cit. 2019/04/10]. Dostupné z: https://www.workamajig.com/blog/project-management-methodologies.

[5] Ecma International. *The JSON Data Interchange Syntax* [online]. [cit. 2019/04/30]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[6] Fiala, M. *Implementace testovacího nástroje pro časovou osu, Bakalářská práce.* 2019.

[7] Guru99. *Java Swing Tutorial: Examples to create GUI* [online]. [cit. 2019/04/30]. Dostupné z: https://www.guru99.com/java-swing-gui.html#1.

[8] Hellgren, H. *Communicating in software development* [online]. [cit. 2019/04/18]. Dostupné z: https://hackernoon.com/communicating-in-software-development-f3434c52eb23.

[9] Holy, L. et al. Software Engineering Projects Analysis using Interactive Multimodal Graph Explorer – IMiGEr. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP,*, s. 330–337. INSTICC, SciTePress, 2019. doi: 10.5220/0007579803300337. ISBN 978-989-758-354-4.

[10] ILIEŞ, L. – CRIŞAN, E. – MUREŞAN, I. N. Best Practices in Project Management. *Review of International Comparative Management.* March 2010, 11, 1. Dostupné z: https://www.researchgate.net/publication/46567671_Best_Practices_in_Project_Management.

[11] Janoch, V. *Nástroj pro grafický popis procesů vývoje software, Bakalářská práce.* 2017.

[12] JFree. *JFreeChart* [online]. [cit. 2019/04/30]. Dostupné z: http://www.jfree.org/jfreechart/.

[13] Joblin, M. et al. From developer networks to verified communities: a fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, s. 563–573, Piscataway, NJ, USA, 2015. IEEE Press.

[14] Laplante, P. A. – Neill, C. J. *AntiPatterns (Identification, Refactoring, and Management).* Taylor & Francis Group, 2006. ISBN 0-8493-2994-9.

[15] Lotz, M. *The Project Manager's Role in Software Development* [online]. 2013. [cit. 2019/04/09]. Dostupné z: https://www.seguetech.com/the-project-managers-role-in-software-development/.

[16] Microtool. *What is ALM?* [online]. [cit. 2019/04/17]. Dostupné z: https://www.microtool.de/en/knowledge-base/what-is-application-lifecycle-management/.

[17] MyManagementGuide. *What is a Project?* [online]. [cit. 2019/05/28]. Dostupné z: https://mymanagementguide.com/basics/what-is-a-project/.

[18] Object Management Group. Software & Systems Process Engineering Meta-Model Specification. 2008. Dostupné z: http://www.omg.org/spec/SPEM/2.0/PDF.

[19] Pathak, R. *Top 5 Project Management Phases* [online]. [cit. 2019/05/28]. Dostupné z: https://project-management.com/top-5-project-management-phases/.

[20] Pícha, P. – Brada, P. ALM tool data usage in software process metamodeling. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, s. 1–8, Piscataway, NJ, USA, 2016. IEEE.

[21] Powell-Morse, A. *Rational Unified Process: What Is It And How Do You Use It?* [online]. [cit. 2019/06/01]. Dostupné z: https://airbrake.io/blog/sdlc/rational-unified-process.

[22] Putano, B. *A guide to knowledge base systems* [online]. [cit. 2019/04/18]. Dostupné z: https://www.smartsheet.com/knowledge-base-systems-and-templates.

[23] Schwaber, K. – Sutherland, J. *The Scrum Guide* [online]. 2017. [cit. 2019/04/11]. Dostupné z: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

[24] Sherrie, R. *Software Project Team Roles and Responsibilities* [online]. 2017. [cit. 2019/04/09]. Dostupné z: https://medium.com/@SherrieRose/software-project-team-roles-and-responsibilities-152a7d575759.

[25] Sifter. *What is issue tracking?* [online]. [cit. 2019/04/18]. Dostupné z: https://sifterapp.com/academy/overview/why/.

[26] SourceMaking. *Design Patterns* [online]. [cit. 2019/04/24]. Dostupné z: https://sourcemaking.com/design_patterns.

[27] Thissen, M. R. et al. Communication Tools for Distributed Software Development Teams. 2007. Dostupné z: https://www.rti.org/sites/default/files/resources/rti_cpr07_virtualteam.pdf.

[28] Verbindungszentrum der Bildung und Forschung. *Scrum Process* [online]. 2019. [cit. 2019/06/06]. Dostupné z: https://www.verbindungszentrum.com/scrum-en/.

[29] Zsolt, U. *Software processes* [online]. [cit. 2019/04/11]. Dostupné z: http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/szoftverfejlesztesi_folyamatok_angol/ch03.html.

Seznam zkratek

ALM – Application Lifecycle Management
API – Application Programming Interface
ASWI – Pokročilé softwarové inženýrství
AWT – Abstract Window Toolkit
CSV – Comma Separated Values
DAO – Data Access Object
FAV – Fakulta Aplikovaných Věd
GUI – Graphical User Interface
IT – Informační technologie
JDBC – Java Database Connectivity
JPG – Joint Photographic Experts Group
JSON – JavaScript Object Notation
KIV – Katedra informatiky a výpočetní techniky
PDF – Portable Document Format
PNG – Portable Network Graphics
SPADe – Software Process Anti-pattern Detector
SQL – Structured Query Language
SVG – Scalable Vector Graphics
UWB – University of West Bohemia
VCS – Version Constrol System
ZČU – Západočeská univerzita

Seznam obrázků

Seznam tabulek

A Obsah CD

- **Obrázky**
  - **StrukturaAplikace.png** – obrázek struktury aplikace
  - **Doménový metamodel.png** – obrázek doménového metamodelu nástroje SPADe
  - **Schéma nástroje.png** – obrázek se schématem nástroje SPADe
- **Projekt**
  - **res** – JSON soubory uložených konstant, proměnných a dotazů
  - **src** – zdrojové kódy aplikace
  - **test** – zdrojové kódy testů
  - **target** – přeložené class soubory
  - **pom.xml** – soubor obsahující závislosti pro spuštění projektu
- **Spustitelná aplikace**
  - **res** – zdrojové soubory
  - **app.properties** – soubor s cestami pro ukládání exportů
  - **czech.properties** – soubor textů pro českou lokalizaci
  - **english.properties** – soubor textů pro anglickou lokalizaci
  - **sql.properties** – SQL dotazy pro uživatelsky definované grafy
  - **sqlVar.properties** – parametry pro SQL dotazy
  - **SPADe-GUI.jar** – spustitelná aplikace
  - **CustomCharts.dat** – soubor pro uložení uživatelsky definovaných grafů
  - **Templates.dat** – soubor pro uložení šablon uživatelsky definovaných grafů
- **Text**
  - **exports** – CSV soubory s výsledky detekcí praktik
  - **JavaDoc** – dokumentace aplikace
  - **log** – vygenerované SQL dotazy při detekování praktik

  - **pdf** – PDF soubor s textem práce
  - **poster** – PDF soubor s textem posteru
  - **practices** – JSON soubory s modely praktik
  - **res** – zdrojové soubory práce spolu s obrázky
  - **scripts** – soubory s SQL skripty vytvářející pohledy
- **CTI_ME.txt** – obsahuje popis obsahu na disku

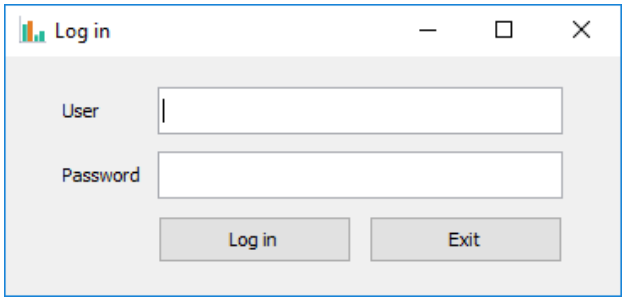
B Uživatelský manuál

Manuál popisuje jednotlivé prvky na nově přidaných oknech v uživatelském rozhraní pomocí číselného označení a příslušného popisu.

Pro spuštění aplikace je potřeba mít nainstalované běhové prostředí Java JRE verze 8 a vyšší. Aplikaci není nutné instalovat, pouze spustit pomocí souboru s příponou `jar`.

B.1 Přihlašování

Po spuštění aplikace se zobrazí okno přihlášení (viz obr. B.1) pro zadání jména a hesla. Po zadání správných přihlašovacích údajů se načtou data do aplikace a zobrazí se hlavní okno.



Obrázek B.1: Přihlašovací okno

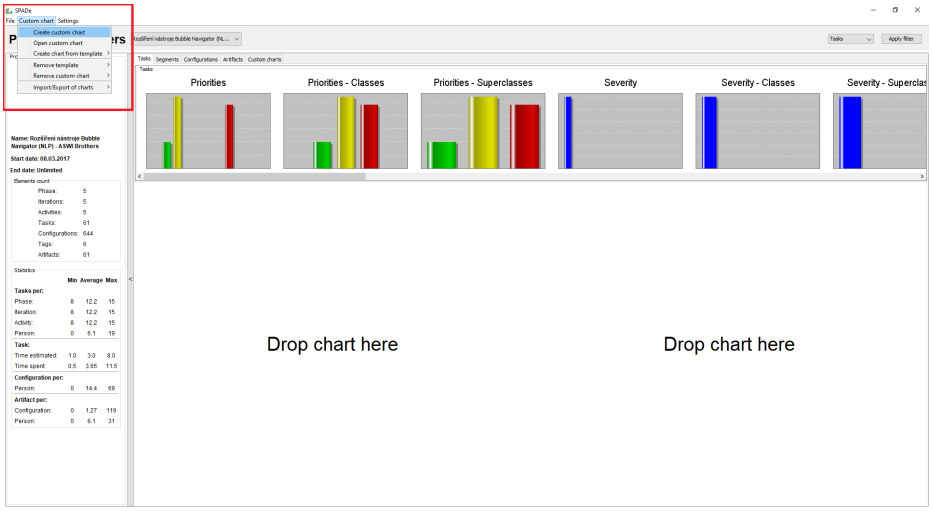
B.2 Hlavní okno

Na nové okno rozšíření se lze dostat přes hlavní okno nástroje pomocí kontextového menu (viz obr. B.2 a obr. B.3).

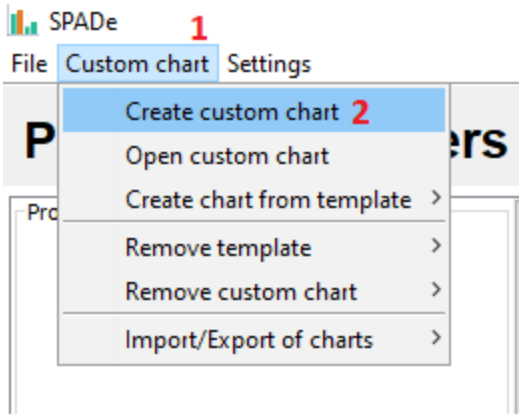
- **1** – Kontextové menu pro uživatelsky definované grafy
- **2** – Položka menu pro otevření nového okna rozšíření

B.3 Okno rozšíření

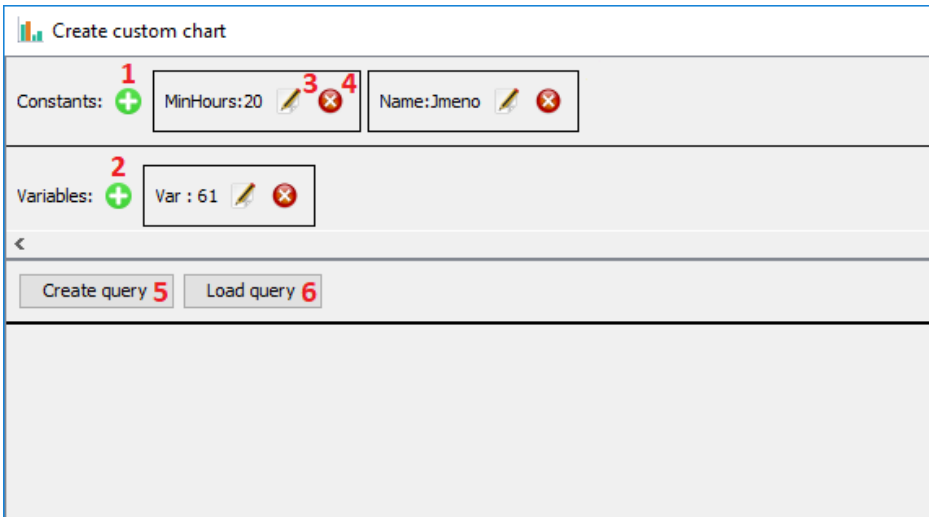
V hlavním okně rozšíření lze vytvářet konstanty, proměnné a dotazy nebo lze dotaz načíst ze souboru, jako popisuje obrázek B.4.



Obrázek B.2: Hlavní okno aplikace



Obrázek B.3: Detail menu v hlavním okně



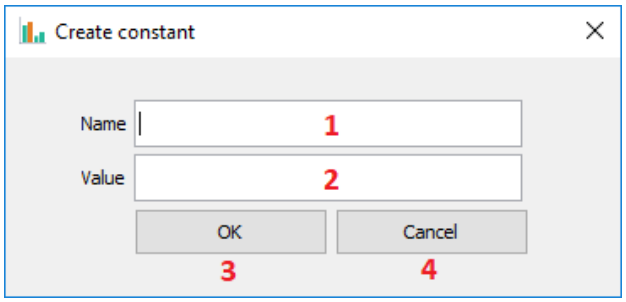
Obrázek B.4: Detail okna rozšíření

- **1** – Tlačítko pro vytvoření nové konstanty. Otevře formulář popsaný v sekci B.4.
- **2** – Tlačítko pro vytvoření nové proměnné. Otevře formulář popsaný v sekci B.5.
- **3** – Tlačítko pro editaci proměnné.
- **4** – Tlačítko pro smazání proměnné.
- **5** – Tlačítko pro vytvoření nového dotazu. Otevře formulář popsaný v sekci B.5.
- **6** – Tlačítko pro načtení praktiky ze souboru.

B.4 Vytváření konstant

Obrázek B.5 popisuje formulář pro vytváření konstant.

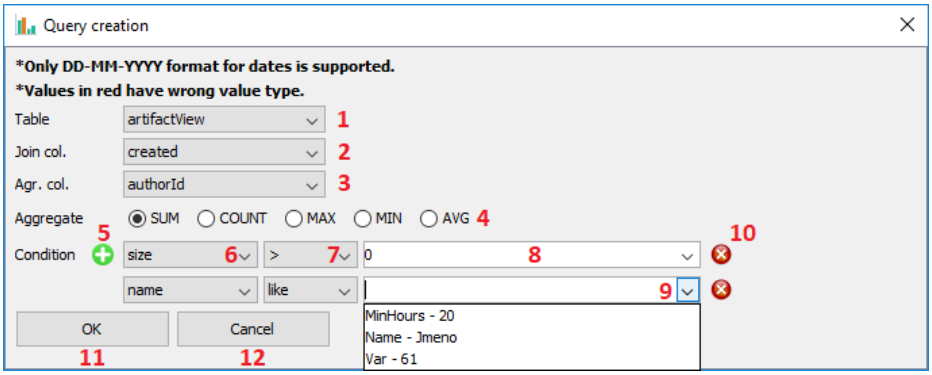
- **1** – Textové pole pro zadání názvu.
- **2** – Textové pole pro zadání hodnoty.
- **3** – Tlačítko pro potvrzení formuláře a vytvoření konstanty.
- **4** – Tlačítko pro zrušení formuláře.



Obrázek B.5: Okno pro vytváření konstant

B.5 Vytváření dotazů

Na obrázku B.6 lze vidět formulář pro vytváření SQL dotazů.



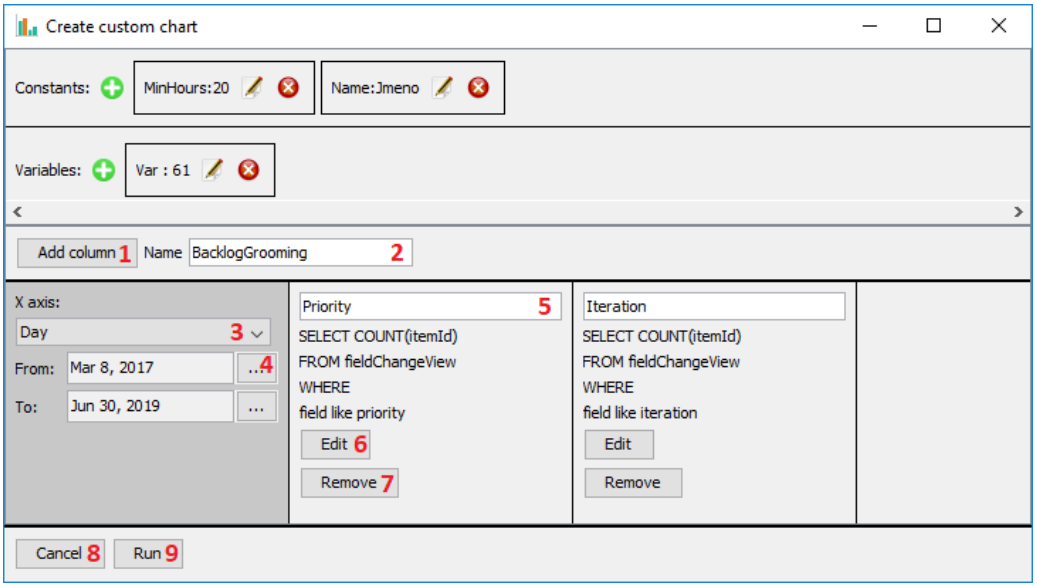
Obrázek B.6: Okno pro vytváření dotazů

- **1** – Dropdown menu pro zvolení pohledu.
- **2** – Dropdown menu pro zvolení sloupce, přes který se budou data mapovat na osu X.
- **3** – Dropdown menu pro zvolení sloupce, přes který se bude počítat agregační funkce.
- **4** – Přepínač na zvolení agregační funkce.
- **5** – Tlačítko pro přidání podmínky do SQL dotazu.
- **6** – Dropdown menu pro zvolení sloupce v podmínce.
- **7** – Dropdown menu pro zvolení operátoru v podmínce.
- **8** – Textové pole pro zadání hodnoty.

- **9** – Dropdown menu pro zadání hodnoty vytvořených konstant a proměnných.
- **10** – Tlačítko pro smazání podmínky.
- **11** – Tlačítko pro potvrzení formuláře a vytvoření dotazu.
- **12** – Tlačítko pro zrušení formuláře.

B.6 Okno rozšíření s vytvořenými dotazy

Obrázek B.7 popisuje okno rozšíření s vytvořenými dotazy nebo praktikou načtenou ze souboru.



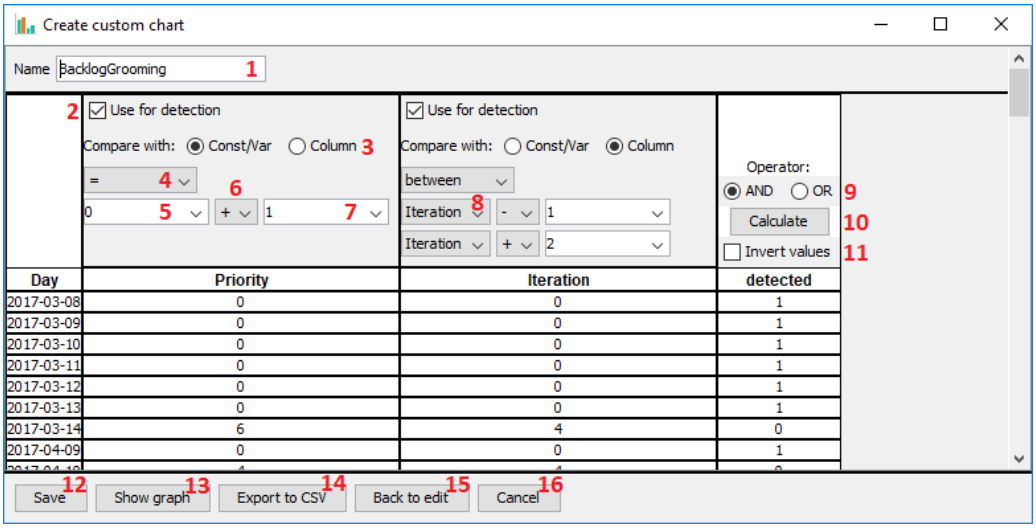
Obrázek B.7: Okno rozšíření s vytvořenými dotazy

- **1** – Tlačítko pro přidání dotazu. Otevře formulář popsaný v sekci B.5.
- **2** – Textové pole pro zadání názvu praktiky
- **3** – Dropdown menu pro zvolení typu dat na ose X.
- **4** – DatePicker pro zvolení dat, pokud jsou na ose X časová data.
- **5** – Textové pole pro zadání názvu sloupce. Důležité pro zobrazení dat do grafu.
- **6** – Tlačítko pro editaci SQL dotazu.

- **7** – Tlačítko pro smazání SQL dotazu.
- **8** – Tlačítko pro zrušení vytváření praktiky.
- **9** – Tlačítko pro spuštění dotazů a zobrazení dat.

B.7 Zadávání kritérií

Na obrázku B.8 je popsáno okno při zadávání kritérií detekce.



Obrázek B.8: Okno rozšíření při zadávání kritérií

- **1** – Textové pole pro zadání názvu praktiky.
- **2** – Přepínač pro použití sloupce při počítání detekce.
- **3** – Přepínač pro zvolení typu hodnoty, se kterou se má sloupec porovnávat.
- **4** – Dropdown menu pro výběr operátoru pro porovnání zadaných hodnot s hodnotami sloupce.
- **5** – Textové pole pro zadání první hodnoty nebo výběru vytvořené konstanty a proměnné pomocí dropdown menu.
- **6** – Dropdown menu pro zvolení aritmetického operátoru.
- **7** – Textové pole pro zadání druhé hodnoty, nebo výběru vytvořené konstanty a proměnné pomocí dropdown menu při zvolení aritmetického operátoru.

- **8** – Dropdown menu pro zvolení sloupce, se kterým se mají hodnoty porovnávat.
- **9** – Přepínač logického operátoru při vyhodnocování podmínek všech sloupců.
- **10** – Tlačítko pro vypočítání detekce pro jednotlivé hodnoty.
- **11** – Přepínač pro inverzi výsledných hodnot detekce.
- **12** – Tlačítko pro uložení celé praktiky do souboru.
- **13** – Tlačítko pro zobrazení výsledné tabulky hodnot do grafu.
- **14** – Tlačítko pro export tabulky výsledků do CSV souboru.
- **15** – Tlačítko pro návrat na předchozí okno, kde lze vytvářet dotazy, konstanty a proměnné.
- **16** – Tlačítko pro zrušení celého procesu vytváření praktiky.