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

Diplomová práce

Analýza přítomnosti anti-vzorů v datech nástrojů pro řízení projektů

Plzeň, 2021 Ondřej Váně

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Ondřej VÁNĚ**
Osobní číslo:	**A19N0096P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační systémy**
Téma práce:	**Analýza přítomnosti anti-vzorů v datech nástrojů pro řízení projektů**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou metodik řízení softwarových projektů a jejich popisů, anti-vzorů v projektovém řízení a strukturou projektových dat v datovém skladu nástroje SPADe.
2. Navrhněte vhodnou sadu anti-vzorů a dostupných projektů (včetně kritérií pro výběr) pro studii výskytu anti-vzorů v projektových datech. Dále pro tyto anti-vzory navrhněte metriky, které umožní jejich detekci v projektových datech.
3. Implementujte soustavu dotazů nad těmito daty odpovídajících navrženým metrikám a ověřte jejich správnou funkčnost.
4. Ověřte formou empirické studie s použitím výsledků bodu 2 výslednou metodu detekce přítomnosti anti-vzorů v datech projektů a diskutujte výsledky studie.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Pícha**
Nové technologie pro informační společnost

Datum zadání diplomové práce: **11. září 2020**
Termín odevzdání diplomové práce: **20. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 24. září 2020

Prohlášení

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

V Plzni dne 19. května 2021

Ondřej Váně

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce Ing. Petru Píchovi za cenné rady, připomínky a především za čas, který strávil vedením této diplomové práce.

Abstract

The goal of this thesis is to analyze and automatically detect anti-patterns in the data of project management tools using SPADe tools. The SPADe tool is used to collect data from ALM tools and search for bad practices (anti-patterns) in project data. In order to develop this thesis, an analysis of the available anti-patterns was performed and then a subset was selected for further investigation. In the next part, the detection of the analyzed anti-patterns was implemented using SQL queries and Java programming language was used to process the results of these queries. As an extra feature of this thesis, going beyond the scope of the original assignment, a support web application for running the detection process of implemented anti-patters and results presentation was also developed. Furthermore, an experiment was performed on a selected sets of anti-patterns and project data. The success of the detection was verified by comparing the results to those from a manual anti-pattern detection in project data from the source ALM tools. Detection success was successful at 93.65 % compared to manual control.

Abstrakt

Cílem této diplomové práce je analyzovat a automaticky detekovat vybrané anti-vzory v datech nástrojů pro projektové řízení s využitím nástroje SPADe. Nástroj SPADe slouží ke sběru dat z jednotlivých ALM nástrojů a následnému hledání špatných praktik (anti-vzorů) v projektových datech. Za účelem vypracování této práce byla provedena analýza dostupných anti-vzorů a následně vybrána omezená podmnožina pro další zkoumání. V další části byla provedena implementace detekce analyzovaných anti-vzorů pomocí SQL dotazů a následným vyhodnocením pomocí jazyka Java. Součástí této práce, nad rámce jejího zadání, je také podpůrná webová aplikace pro snadné spouštění detekce implementovaných anti-vzorů a zobrazení jejích výsledků. Následně byl proveden experiment nad vybranou množinou anti-vzorů a projektových dat. Úspěšnost detekce byla ověřena porovnáním výsledků s výstupy manuální detekce anti-vzorů v datech projektů ve zdrojových ALM nástrojích. Úspěšnost detekce byla úspěšná v 93,65 % případů oproti manuální kontrole.

Obsah

1 Úvod **1**

2 Proces vývoje software **2**
2.1 Proces . . . 2
2.2 Projekt . . . 3
2.3 Metodiky vývoje softwaru . . . 4
2.3.1 Vodopádový model . . . 4
2.3.2 Rational Unified Process . . . 6
2.3.3 Scrum . . . 8
2.3.4 Extrémní programování . . . 10
2.3.5 Kanban . . . 12
2.4 Vzory . . . 14
2.4.1 Příklady vzorů . . . 15
2.5 Anti-vzory . . . 16
2.5.1 Příklady anti-vzorů . . . 16
2.6 Shrnutí . . . 17

3 Projektová data, ALM nástroje a SPADe **18**
3.1 Nástroje pro správu verzí . . . 18
3.2 Nástroje pro správu nasazení . . . 19
3.3 Nástroje pro správu změn a problémů . . . 20
3.4 Nástroje pro správu dokumentů . . . 21
3.5 Nástroje pro komunikaci . . . 21
3.6 SPADe . . . 22
3.6.1 Popis nástroje SPADe . . . 22
3.6.2 Datový sklad . . . 24
3.7 Shrnutí . . . 31

4 Návrh detekce anti-vzorů **32**
4.1 Výběr vhodných anti-vzorů . . . 32
4.2 Popis a návrh detekce pro vybrané anti-vzory . . . 33
4.2.1 Business As Usual . . . 33
4.2.2 Long Or Non-Existant Feedback Loops . . . 35
4.2.3 Ninety-Ninety Rule . . . 39
4.2.4 Road To Nowhere . . . 41
4.2.5 Specify Nothing . . . 44

4.2.6 Too Long Sprint . . . 46
4.2.7 Varying Sprint Length . . . 49
4.3 Návrh pomocné aplikace . . . 51
4.3.1 Klíčové vlastnosti . . . 51
4.3.2 Struktura aplikace . . . 52

5 Implementace **54**
5.1 Implementace detekce anti-vzorů . . . 54
5.2 Implementace pomocné aplikace . . . 58
5.2.1 Zvolené technologie . . . 58
5.2.2 Struktura projektu . . . 59
5.2.3 Balíky tříd . . . 59
5.2.4 Rozšíření o další anti-vzory . . . 60
5.2.5 Testování . . . 61

6 Experiment **62**
6.1 Postup experimentu . . . 62
6.2 Výběr vhodné sady projektů . . . 62
6.2.1 ASWI projekty . . . 63
6.2.2 ASWI proces . . . 64
6.3 Nastavení prahových hodnot . . . 65
6.4 Výsledky . . . 68
6.5 Diskuze . . . 72
6.6 Možná rozšíření . . . 75

7 Závěr **76**

Literatura **77**

Seznam zkratek **79**

Seznam obrázků **81**

Seznam tabulek **83**

Přílohy **84**
A Obsah ZIP souboru . . . 85
B Příručka nasazení . . . 86
B.1 Potřebné nástroje . . . 86
B.2 Postup spuštění aplikace . . . 86
B.3 Obnova a konfigurace databáze . . . 87
B.4 Konfigurace aplikace . . . 90

B.5 Spuštění aplikace s napojením na SPADe 91
C Uživatelská příručka 93
C.1 Detekce anti-vzorů 93
C.2 Konfigurace prahových hodnot 97
C.3 Zobrazení informací o projektu 100
C.4 Zobrazení informací o anti-vzoru 101

1 Úvod

Vývoj softwaru obnáší mnoho rozdílných činností, které je nutné provést k úspěšné realizaci výsledného produktu. Jedná se například o testování, vývoj, ale také o podpůrné činnosti, kam lze zařadit například řízení celého projektu a jednotlivých zainteresovaných osob. Pokud všechny, nebo alespoň většina činností v rámci vývoje projektu neprobíhají správně, je možné, že bude celý projekt neúspěšný. Nejčastějším problémem neúspěšných projektů je špatné řízení, které má dopad na celý projektový tým a následně i na výsledný produkt. Ve většině případů se jedná o opakující se chybné praktiky, které vedou ke špatným výsledkům, tzv. anti-vzorům. V případě detekce těchto chybných praktik by bylo možné včasně identifikovat problém a následně ho napravit.

Tato práce se zabývá automatickou detekcí vybraných anti-vzorů s využitím nástroje Software Process Anti-pattern Detector (SPADe). Tento nástroj je vytvářen na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU). SPADe uchovává komplexní informace z různých Application Lifecycle Management (ALM) nástrojů, ze kterých je možné identifikovat některé vhodné anti-vzory. Zabývat se automatickou detekcí je vhodné z důvodu toho, že manuální detekce je subjektivní, příliš dlouhá a náročná na zdroje. Kombinací nástroje SPADe a automatické detekce anti-vzorů by následně mohl vzniknout podpůrný nástroj pro identifikaci nevhodných praktik. Tento nástroj by byl vhodný pro projektové manažery, kteří by mohli v průběhu projektu detekovat různé anti-vzory a patřičně na ně zareagovat. Tím by se následně zvýšila šance k úspěšnému dokončení projektu.

Práce je rozčleněna do čtyř hlavních částí. První část se zabývá úvodem do projektového řízení, jednotlivých metodik ve vývoji softwaru a problematikou vzorů a anti-vzorů. Ve druhé části jsou představeny ALM nástroje a také nástroj SPADe. Ve třetí části jsou vybrané vhodné anti-vzory, které by bylo možné detekovat s využitím SPADe, a navrhnuty jejich možné detekce. V poslední části práce je implementována detekce vybraných anti-vzorů na základě předchozí analýzy. Následně je vybrána vhodná sada projektových dat, na kterých je provedena automatická a manuální detekce vybraných anti-vzorů. Porovnáním výsledků z obou detekcí je vyčíslena výsledná úspěšnost automatické detekce. Na závěr je provedena diskuze nad výsledky a jsou zde navrhnuta další možná rozšíření této práce.

2 Proces vývoje software

Vývoj softwaru v dnešní době představuje soubor mnoha různých činností, které napomáhají k úspěšnému dokončení projektu, a tím i uspokojení zákazníka. Mezi tyto aktivity již zdaleka nepatří pouze psaní programového kódu, ale patří sem také například sběr požadavků, kapacitní plánování, návrh architektury, návrh nasazení, testování atd. Všechny tyto činnosti můžeme jednotně nazvat proces vývoje softwaru.

V této kapitole jsou představeny základní pojmy týkající se procesu vývoje softwaru a jednotlivé metodiky, které se v dnešní době nejčastěji využívají k řízení vývoje softwaru.

2.1 Proces

Proces je sada jednotlivých činností, které na sebe navazují a mají určitou posloupnost. Při jednotlivých činnostech dochází k přeměně vstupů na požadované výstupy. Proces se snažíme stále zdokonalovat, optimalizovat a vylepšovat.

Proces může být lineární nebo komplikovanější. O lineární proces se jedná, pokud za každou ukončenou činností následuje pouze jedna další činnost nebo konec procesu, tedy výsledek. Oproti lineárnímu procesu může být proces složitější, kde po jedné ukončené činnosti může navazovat několik dalších činností. Dojde tak k větvení procesu, který může být prováděn paralelně několika pracovníky. Následně se opět sbíhá do jednoho výsledku. [5].

V procesu vývoje softwaru se vyskytují čtyři základní prvky, kterými jsou:

- úkol,
- fáze,
- artefakt,
- role.

Celý projekt se rozpadá na jednotlivé činnosti nazývané úkoly. Tyto úkoly můžeme rozdělit do dvou základních skupin. Úkoly technické a podpůrné. Technické úkoly jsou spojeny s přímým vývojem produktu jako například

analýza, návrh či kódování. Podpůrné úkoly mají za cíl například řídit celkový vývoj nebo kontrolovat kvalitu produktu.

Dalším základním prvkem jsou fáze projektu. Každý softwarový projekt lze rozdělit na menší části na tzv. fáze. V rámci každé fáze se vykonávají jednotlivé úkoly. Výstupem z každé fáze by měl být jeden nebo více artefaktů.

Artefakt je libovolný objekt, dokument či programový kód, který může figurovat jako vstup nebo výstup v jednotlivých fázích procesu. Jednotlivé procesy mohou předepisovat formu nebo úroveň formality. Artefakty můžeme dělit dle jejich účelu na technické, komunikační nebo obchodní.

Role je sada kompetencí a povinností osoby v rámci projektového týmu. Obvykle jsou jednotlivé role a jejich činnosti odvozené od konkrétního procesu. Každý, kdo se účastní vývoje softwaru, by měl mít jasně definovanou roli (či více rolí) a k ní odpovídající pracovní náplň. Role můžeme rozdělit do třech základních skupin: technické, manažerské a podpůrné. Pracovníci v technické skupině se starají o analýzu, vývoj, testování a nasazení produktu. Manažerské role se starají o správný průběh celého projektu nebo o ostatní manažerské činnosti ve firmě. Dále je zde skupina podpůrná. Tato skupina se stará například o dokumentaci, mentorování nebo o uživatelskou podporu.

2.2 Projekt

V předchozí kapitole byl popsán proces, který lze definovat také jako opakovatelný předpis daných činností. Projekt je unikátní instancí tohoto procesu s konkrétními vstupy a výstupy. Jedná se o jednorázový jev, který slouží k vytvoření či změně něčeho jedinečného, např. vytvoření softwaru, implementace nového informačního systému, uvedení nového výrobku na trh a podobně. Průběh každého projektu bývá unikátní a dopředu ho nelze dokonale popsat a naplánovat. U většiny projektů lze definovat fáze, které obsahují základní popis, co by se mělo v dané fázi odehrát. Konkrétní činnosti v každé fázi se však mohou v průběhu projektu měnit z důvodu nepředvídatelnosti.

Z tohoto důvodu zavádíme pojem projektové řízení. Jedná se o činnosti související s plánováním a kontinuálním řízením, jak jednotlivých činností projektu, tak řízení celého projektu. Nejčastěji se projektové řízení využívá při dodávání unikátního softwaru zákazníkům např. implementace informačního systému, vývoj nové podnikové aplikace, konfigurace stávajícího systému dle požadavků zákazníka apod. [10].

2.3 Metodiky vývoje softwaru

Pro zpracování jednotlivých projektů se používají procesní modely nebo také metodiky. Metodika představuje soubor doporučených postupů, nástrojů, pravidel a praktik, které pokrývají celkový životní cyklus vývoje softwaru. Do těchto metodik můžeme zařadit například Scrum, vodopádový model, nebo RUP. Metodiky lze rozčlenit do základních skupin:

- sekvenční,
- iterativní,
- agilní,
- štíhlé (lean).

2.3.1 Vodopádový model

Typickým představitelem sekvenčních metodik je vodopádový model, který představuje přístup k vývoji nebo řízení projektu. Hlavním charakteristickým rysem vodopádového modelu je sekvenční postup fází, z nichž se každá soustředí na poměrně úzce specifickou činnost. Jednotlivé činnosti se neprolínají a jsou pevně řízené. Z tohoto důvodu jsou náchylné na změny v pokročilých fázích projektu. Vodopádový model je vhodný použít pro projekty s jasným cílem, jasně definovaným postupem a rozdělenou prací.[29]

Vodopádový přístup lze rozdělit do pěti základních fází:

- sběr požadavků,
- analýza a návrh,
- vývoj a testování,
- schválení a předání,
- údržba.

Jako první fáze je sběr požadavků. V této fázi dochází ke komunikaci se zákazníkem a prioritou je zjistit co nejvíce informací, které poslouží k vytvoření potřebného produktu. Tyto informace a požadavky zákazníka jsou většinou sepsány do závazného dokumentu, jako je například dokument specifikace projektu.

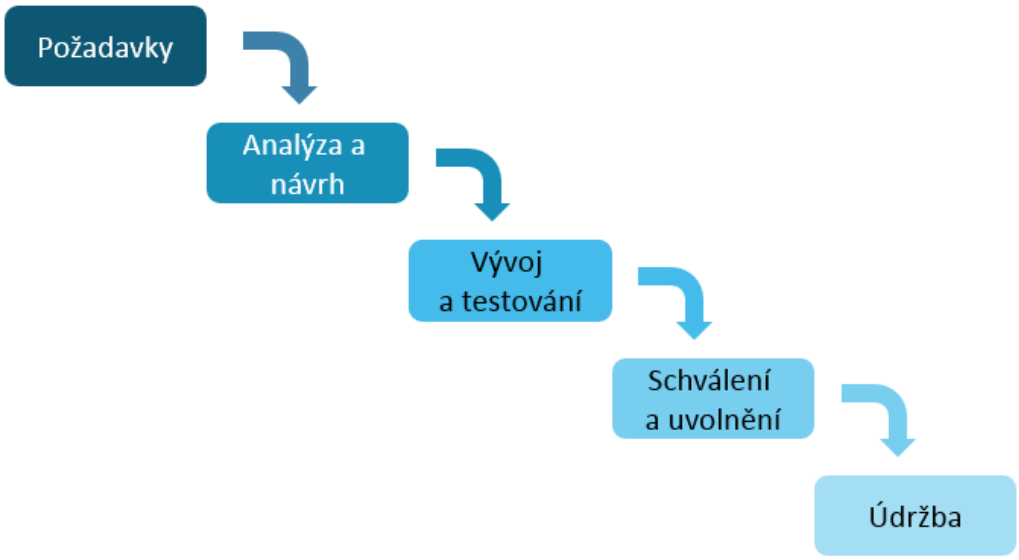
Následuje fáze analýzy a návrhu, ve které se zabýváme návrhem jednotlivých komponent systému tak, aby byl celý systém funkční a použitelný. Využijí se znalosti z předchozí fáze.

Další fází je vývoj a testování, ve které vzniká samotný produkt na základě dokumentace, která byla vytvořena v předchozí části. V průběhu této fáze se také provádí testování jednotlivých funkcí produktu.

Předposlední fáze je schválení a předání hotového produktu. V této fázi opět dochází ke kontaktu se zákazníkem, kdy je mu představen výsledný produkt. Zde může dojít k výhradám ze strany zákazníka, který není s výsledným produktem spokojen. V opačném případě dochází ke konsenzu, předání produktu a uvedení do provozu.

Poslední fází je údržba výsledného systému. Jedná se o nejdelší fázi, kdy je systém plně využíván zákazníkem. Dochází pouze k drobným změnám, opravám chyb nebo k úpravám některých funkcionalit na základě požadavků zákazníka.

Jednotlivé fáze na sebe bezprostředně navazují, a jakmile je jedna fáze dokončena, již se k této fázi nevracíme. Jakmile se tedy udělá chyba na začátku projektu, tak je velice těžké, nebo dokonce nemožné ji napravit. Velkou výhodou je, že na začátku vývoje se určí přesně požadavky, podle nich se stanoví harmonogram prací, termíny dokončení a rozpočet. Díky kvalitním analýzám a upřesnění na začátku vývoje se dá mnohé předpovědět a rovnou navrhnout funkční model aplikace. Tím se dají ušetřit nemalé prostředky. Vodopádový přístup je zobrazen na obrázku číslo 2.1.



Obrázek 2.1: Vodopádový model [29]

2.3.2 Rational Unified Process

Hlavní výsadou iterativních metodik je, že vývoj softwaru probíhá v jednotlivých iteracích (jejíž délka je obvykle několik týdnů). V každé iteraci se implementují naplánované funkce a na konci každé iterace vznikne funkční program, který je představen zákazníkovi (Potentially Shippable Product). V tom je velký rozdíl oproti sekvenčním metodikám, kde většinou dochází ke kontaktu se zákazníkem na začátku a na konci celého projektu.

Jeden ze zástupců iterativních metodik je Rational Unified Process (RUP). RUP je metodika vycházející z obecnější metodiky Unified Process (UP), která byla vyvinuta společností IBM. Tato metodika určuje mnoho kroků a úkolů, které by měly vést k úspěšnému dokončení projektu. Do těchto aktivit můžeme například zařadit správu uživatelských požadavků, rozsáhlou analýzu nebo ověřování jednotlivých částí vytvořeného softwaru. Tato metodika je díky velké míře plánování a analýzám vhodná spíše pro projekty většího rozsahu s velkými týmy [16].

Fáze

Celý proces vývoje softwaru pomocí metodiky RUP můžeme rozdělit do čtyř základních fází:

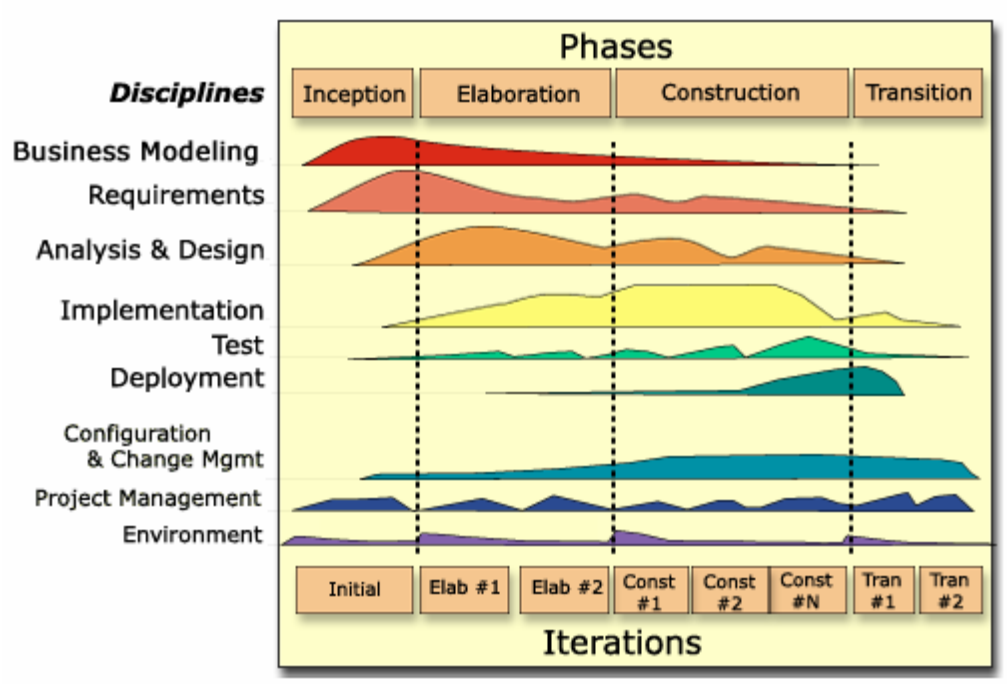
- zahájení – Během zahajovací fáze je vytvořen obchodní záměr systému a ohraničen rozsah projektu. Definujeme zde kritéria úspěchu, rizika projektu nebo například časový odhad projektu. V této fázi nejvíce komunikujeme se zákazníkem a snažíme se zjistit jeho požadavky.

- příprava – Hlavním cílem fáze přípravy je analýza problémové oblasti, vytvoření vhodného architektonického návrhu, vypracování projektového plánu. Architektonický návrh musí být navrhnut tak, aby odpovídal jeho rozsahu, hlavním funkcím, ale třeba i mimofunkčním požadavkům.

- konstrukce – Během fáze konstrukce jsou implementovány a integrovány všechny komponenty a funkce aplikace a všechny jeho funkce jsou důkladně otestovány. Jedná se o časově nejnáročnější fázi projektu.

- předávání – Cílem poslední fáze je předání softwarového produktu zákazníkovi, nebo uvedení do provozu. Jakmile je produkt uveden do provozu, tak obvykle vznikají problémy, které vyžadují vývoj opravných verzí [16].

Na obrázku číslo 2.2 jsou zobrazeny fáze a jednotlivé disciplíny, které se v každé fázi vykonávají a s jakou intenzitou.



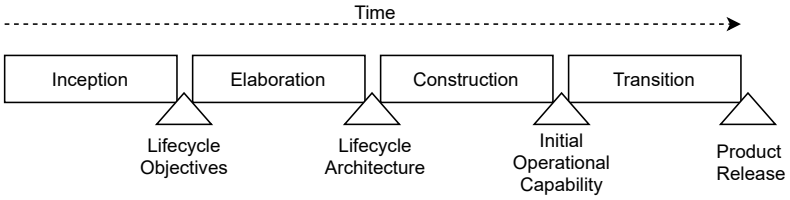
Obrázek 2.2: Fáze RUP [16]

Milníky

Dalším pojmem v metodice RUP jsou tzv. milníky, z nichž je každý přiřazen k jedné z fází. Milník je sadou hodnotících kritérií stavu projektu a jeho dosažení (tzn. splnění kritérií), značí konec příslušné fáze. Sady hodnotících kritérií jsou velice rozsáhlé, a proto zde bude nestíněn pouze základní cíl. Jednotlivé milníky jsou:

- Lifecycle Objectives (LCO) – Pro dosažení milníku LCO by měl být primárně jasný záměr a vize projektu.
- Lifecycle Architecture (LCA)– Pro splnění milníku LCA by měla být hotova základní architektura výsledného systému.
- Initial Operational Capability (IOC)– Pro dosažení milníku IOC by měl být produkt připraven k nasazení do prostředí beta-testu.
- Product Release (PR)– Pro vyhovění milníku PR je nutné mít hotový produkt, a tím uspokojit potřeby zákazníka[16].

Posloupnost jednotlivých milníků a návaznost na fáze metodiky RUP jsou zobrazeny na obrázku číslo 2.3.



Obrázek 2.3: Milníky metodiky RUP

2.3.3 Scrum

V průběhu let se objevilo mnoho metodik pro vývoj softwaru, v dnešní době jsou často používané tzv. agilní metodiky vývoje softwaru. Tyto metodiky jsou založeny na Manifestu Agilního vývoje software (Manifesto for Agile Software Development), který byl vydán v roce 2001 několika programátory na základě jejich předchozích zkušeností. Autoři Agilního manifestu dospěli k následujícím hodnotám:

- upřednostnění jednotlivců a interakce před procesy a nástroji,
- upřednostnění fungujícího softwaru před vyčerpávající dokumentací,
- upřednostnění spolupráce se zákazníkem před vyjednáváním o smlouvě,
- upřednostnění reagování na změny před dodržováním plánu [3].

Hlavním cílem agilních metodik je rychlé a efektivní reagování na změny zákazníka, časté dodávky produktu v jednotlivých jeho fázích a dobrá komunikace uvnitř týmu i se zákazníkem. Do pozadí jsou postaveny například smlouvy, dokonalá dokumentace a dodržování plánů.

Jeden z hlavních zástupců agilních metodik je metodika s názvem Scrum. Jedná se o iterativní a inkrementální způsob řízení vývoje softwaru, který se řadí mezi agilní metodiky. Jedna z klíčových vlastností je vysoká adaptace na změny zákazníka. Důležitá je také transparentnost. Všichni by měli mít jasný přehled o tom, co a proč se dělá a v jakém je to stavu. Na obrázku číslo 2.4 jsou znázorněny hlavní role a činnosti v metodice Scrum, které budou dále představeny.

Vývoj probíhá v jednotlivých iteracích podobně jako v metodice RUP, popsána v sekci číslo 2.3.2. Avšak v terminologii Scrumu se iterace nazývají jako sprint. Obvyklá délka doporučená metodikou Scrum je mezi třemi až čtyřmi týdny [1].

Role

V metodice Scrum se vyskytují tři role, které mají za cíl vznik produktu. Při reálném využití Scrum metodiky mohou být definovány některé další role, ale Scrum definuje pouze tři následující:

- vývojový tým,
- vlastníka produktu (Product Owner),
- Scrum Mastera.

Vývojový tým má za úkol dokončit všechny úkoly, které měl v dané iteraci naplánované. Tým musí být sebeorganizující a každý člen týmu by měl být specialistou na nějakou oblast technologií. Tím je dosaženo, že tým si dokáže poradit s jakýmkoliv problémem. Velikost Scrum týmu není nijak striktně omezena či určena, většinou se však udává, že velikost týmu by měla být od pěti do devíti členů [19].

Vlastník produktu neboli Product Owner by měl být zástupce zákazníka. Měl by vědět, co je nutné v další fázi implementovat či změnit. Vlastník produktu je v podstatě komunikační prostředek mezi vývojovým týmem a zákazníkem.

Scrum Master je osoba, která dohlíží na správné dodržování metodiky a také má za cíl odstraňování překážek, které brání vývojovému týmu dokončit některé úkoly. [28].

Události

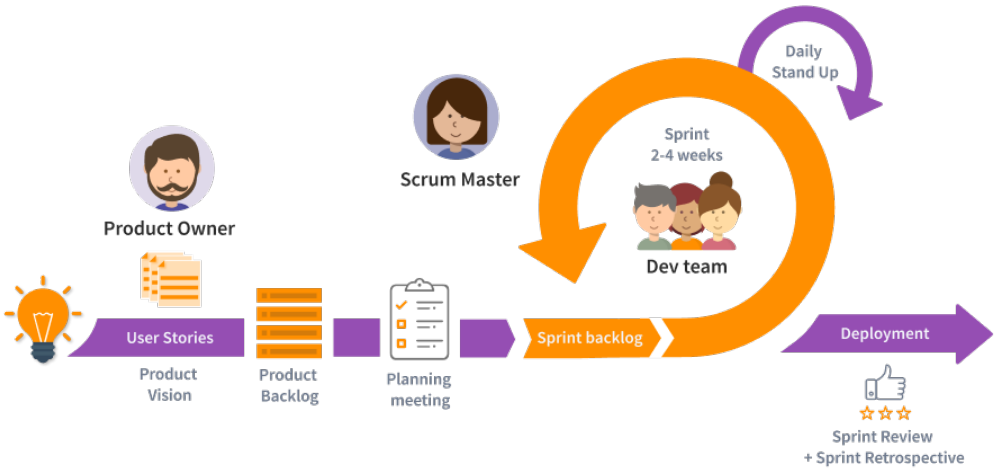
V metodice Scrum se používají tři základní události, které probíhají periodicky v každém sprintu:

- plánování,
- Daily Srum,
- retrospektiva.

Na začátku každého sprintu dochází k tzv. plánování. Při plánování se pracuje s backlogem. Backlog je v podstatě seznam časově ohodnocených úkolů ke zpracování v rámci projektu, které mohou mít přiřazeny priority. V rámci plánování se sejdou všichni členové týmu, Scrum Master a vlastník produktu. Následně vlastník produktu vybere z backlogu jednotlivé úkoly, které mohou být vybrány na základě priority nebo časové kapacity jednotlivých členů týmu. Následně si členové týmu rozdělí úkoly dle jejich časových kapacit.

Daily Srum je krátká schůzka vývojového týmu, která se koná každý pracovní den. Délka této schůzky by měla být od 10 do 15 minut. Účelem této schůzky je předat si poznatky mezi celým týmem o rozdělané, hotové či nedokončené práci. Tím se zajistí dostatečná transparentnost, synchronizace a vzájemná zastupitelnost ve vývojovém týmu.

Poslední událostí je retrospektiva. Retrospektiva je schůzka, která se koná po ukončení každého sprintu. Hlavní účel retrospektivy je identifikovat problémy, zjistit, zda jsou příčiny problému systematické nebo systematicky řešitelné. Při detekci problémů dochází k úpravě postupů, které vedou k jejich odstranění, minimalizaci dopadů, zamezení opakování a dalším procesním vylepšením. V literatuře se toto označuje jako metodika pro vylepšení softwarových procesů (Software Process Improvement; SPI). SPI je definována jako sled úkolů, nástrojů a technik pro plánování a implementaci aktivit zlepšování k dosažení konkrétních cílů, jako je zvýšení rychlosti vývoje, dosažení vyšší kvality produktu nebo snížení nákladů [15]. Z retrospektivy mohou vyplynout nějaké další úkoly, které bude nutné v následujících sprintech vyřešit [26].



Obrázek 2.4: Scrum proces [28]

2.3.4 Extrémní programování

Extrémní programování (XP) je agilní metodikou pro vývoj softwaru, která nespočívá v zavádění nových postupů, ale extremizuje známé postupy či metody, které se osvědčily. Hlavním cílem extrémního programování je co nejrychleji se přizpůsobit na měnící se požadavky ze strany zákazníka, a tím vyvíjet a dodávat kvalitnější software.

XP spočívá v extremizování známých a ověřených postupů. V minulosti se například osvědčila revize programového kódu, tudíž extremizací tohoto postupu je práce ve dvojicích. Ve dvojicích dochází k bezprostřední revizi výsledného kódu, a tím vzniká kvalitnější a lepší programový kód. V minulosti se také osvědčilo testování jednotlivých funkcionalit, takže v extrémním programování dochází k velmi častému testování všeho. V podstatě každý řádek programového kódu by měl být pokrytý testem. Dalším znakem extrémního programování je psát co nejméně složité úseky kódu. Toho lze dosáhnout tím, že implementujeme pouze to, co je v danou chvíli nejdůležitější [14].

Postup vývoje

Extrémní programování funguje podobně jako Scrum v jednotlivých iteracích. Zákazník může v průběhu iterace doplnit nebo pozměnit požadavky. Pokud k tomuto dojde, tak je prohlášená iterace za ukončenou a začíná se opět od začátku. Iterace v extrémním programování má nejčastěji následující fáze:

- zadávání,
- plánování,
- designování,
- programování,
- testování.

Nejprve se začne zadáváním, kdy se sepíší jednotlivé případy užití a k tomu odpovídající akceptační scénáře. Jak již bylo zmíněno, zákazník může kdykoliv tyto případy užití pozměnit, a tím může dojít k restartu cyklu vývoje.

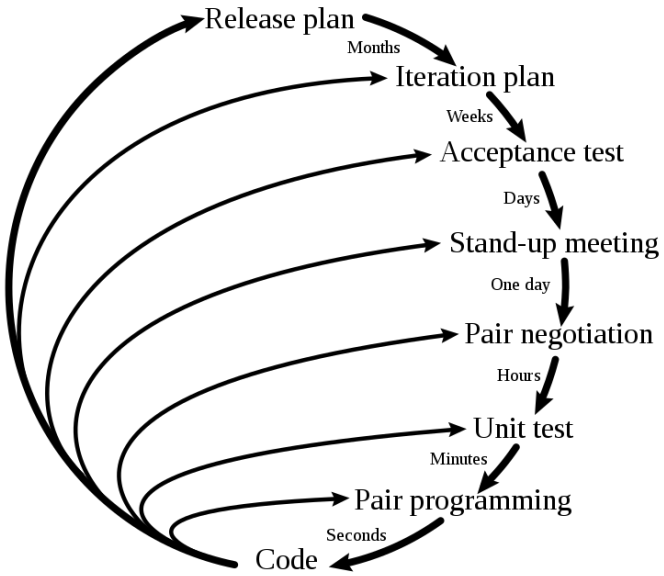
Další fází je plánování, při kterém se vytvoří časový harmonogram jednotlivých iterací a naplánuje se vydání výsledného produktu.

Fáze designování v sobě zahrnuje návrh jednotlivých funkcionalit, ty se snažíme navrhnout co nejjednodušeji. Nesnažíme se přidávat funkcionality, které zákazník výslovně nepotřebuje.

Ve fázi programování dochází k implementaci jednotlivých funkcionalit, které se v předchozí fázi navrhly. Nejprve dojde k sepsání jednotkových testů a až poté se píše programový kód, který bude testům vyhovovat, tzv. vývoj řízený testy (test driven development). Programový kód se píše vždy ve dvojicích, a tím je docíleno jeho vysoké kvality. V rámci této fáze také probíhá integrace jednotlivých částí, kterou provádí vždy pouze jedna dvojice programátorů. Na závěr probíhá optimalizace a úprava vzniklého kódu.

Poslední fází je fáze testování. Jelikož testování je osvědčená metodika pro ověřování kvality softwaru, tak v extrémním programování se testuje všechno a ve velkém množství. Každý úsek programového kódu tedy musí být popsán testem. Pokud se v této fázi vyskytne chyba, tak dojde k opravení a napsání testu na tuto chybu. Dále také vznikají další sady testů, které jsou relevantní pro daný projekt například funkční nebo integrační. [14].

Proces vývoje softwaru pomocí metodiky extrémního programování je zobrazen na obrázku 2.5



Obrázek 2.5: Proces vývoje v extrémní programování [14]

2.3.5 Kanban

Kanban je metodika, která se řadí mezi tzv. štíhlé (lean) metodiky. Štíhlá výroba znamená eliminování nadbytečných činností a soustředí se na nejdůležitější faktory, které zákazník opravdu vyžaduje. Tato metoda byla vyvinuta primárně pro výrobní podniky, později se však začala aplikovat i na vývoj softwaru, kde se velice osvědčila. S metodikou Scrum se řadí mezi nejpoužívanější metodiky v dnešní době [14].

Kanban tabule

V této metodice se klade důraz na vizualizaci. S její pomocí je snadno rozpoznatelné, v jaké fázi se nacházejí jednotlivé úkoly. K této vizualizaci se používá Kanban tabule (Kanban Board). Tato tabule je rozdělena na mřížku

složenou ze stavů životního cyklu úkolu (sloupce) a priorit (řádky). Úkoly ve formě štítků, nálepek nebo papírků se po tabuli pohybují podle svého aktuálního stavu. Tabule tak zachycuje celkový stav projektu v dané chvíli. Tabuli lze vést ve fyzické nebo elektronické podobě.

Každý sloupec ukazuje, v jaké fázi se nachází každý úkol. Pokud se změní rozpracovanost daného úkolu, tak se úkol přesune do dalšího sloupce. V této metodice je kladen důraz na omezení počtů úkolů v jednotlivých fázích. V jednom sloupci může být pouze omezený počet úkolů. Tím může nastat problém, že pracovník po dokončení fáze úkolu nemůže přesunout úkol do dalšího sloupce. Avšak díky Kanban tabuli je zřejmé, kde nastal problém, a mohou pomoci s řešením tohoto problému ostatním členům týmu.

Řádky na Kanban tabuli mohou znázorňovat míru priority jednotlivých úkolů. Čím výše se úkol nachází, tím větší má prioritu a naopak. Úkoly s větší prioritou by měly být plněny přednostně. Prioritu úkolu může určovat nadřízený, ostatní členové týmu nebo přímo zákazník.

Nejčastěji se používá tabule se třemi stavy úkolů: k vyřešení, ve vývoji a dokončené (To Do, In Progress a Done). Avšak tabule může být mnohem složitější a může mít více stavů. Příklad složitější Kanban tabule je zobrazena na obrázku číslo 2.6. Jednotlivé stavy závisejí na charakteru projektu, který se pomocí Kanbanu snažíme dokončit a je možno si je upravovat dle potřeby.

Obrázek 2.6: Ukázka Kanban tabule [27]

2.4 Vzory

V jednotlivých fázích vývoje softwaru můžeme narazit na velké množství potíží nebo problémů. Může se jednat o problém návrhu aplikace, implementace nebo pouze o personální problém. Pokud se tento problém opakuje častěji a je veřejně známý, je možné, že existuje nějaké jeho obecně známé řešení nazývané jako vzor.

Vzor lze formulovat jako definované a praxí ověřené řešení známého problému, které můžeme aplikovat opakovaně v daném kontextu, a tím vyřešit daný problém. Nutné je však porozumět kontextu použití vzoru, který při správné aplikaci vzoru ve správném kontextu vede k vyřešení problému. Tento fakt mnohonásobně ověřený v praxi vede k ustálení tohoto řešení, které se stane známým vzorem v celém oboru softwarového inženýrství.

Vzor lze také definovat jako pojmenovanou dvojici popisu problému a jeho řešení. [17].

2.4.1 Příklady vzorů

Vzory se mohou vyskytovat na všech úrovních a ve všech disciplínách a aktivitách kolem projektu. Konkrétní příklady vybraných vzorů jsou popsány v následující části.

Singleton

V programu je někdy nutné sdílet jednu instanci objektu ostatním třídám. Pokud by se v programu vytvářely stále nové instance tohoto objektu, tak by mohlo dojít k přetečení paměti. Proto lze využít vzor s názvem Singleton (jedináček). Jedináček je tvořen třídou, která se stará o to, aby instance této třídy existovala pouze jednou. V případě, že je instance této třídy potřeba, nejprve dojde ke kontrole, zda již instance neexistuje. Pokud existuje, je vrácena instance této třídy, ale pokud neexistuje, vytvoří se nová instance. Praktické využití může být například instance připojení k databázi aplikace. Singleton lze zařadit do tzv. návrhových vzorů.

Client-Server

Vzor Client-server (klient-server) řeší problém komunikace a výměny dat mezi jednotlivými instancemi programu. Jedná se v podstatě o síťovou architekturu, která nám definuje několik klientských stanic, které komunikují pomocí různých protokolů se serverem. Server následně řídí komunikaci nebo výměnu dat mezi jednotlivými klienty. Nejčastější klienty si v dnešní době můžeme představit jako webové prohlížeče, kde je zobrazena webová aplikace. V tomto případě se jedná o architektonický vzor.

Semantic Versioning

Vzor popisující sémantické označování jednotlivých verzí softwaru tak, aby bylo z verze možné odhadnout, k jak velkým změnám došlo, například oproti předchozí verzi. Tento vzor se také někdy označuje jako tříúrovňová tečková notace k označení verzí softwaru. Název verze je rozdělen do tří sekcí oddělených tečkou. První sekce označuje major verze (hlavní verze), za tečkou následuje minor verze (vedlejší verze) a třetí číslice označuje revizi (6.3.0) [20].

2.5 Anti-vzory

Pojem anti-vzor představuje popis řešení běžně vyskytujícího se problému, který má negativní důsledky. Anti-vzor může být špatným výsledkem práce manažera nebo vývojáře, který nemá dostatečné znalosti nebo zkušenosti při řešení daného problému. Špatný výsledek může být také způsoben chybným pochopením kontextu řešeného problému [8]. Myšlenka anti-vzorů byla zavedena brzy potom, co byl představen koncept vzorů [17]. Anti-vzor je podobný jako vzor, rozdíl je v tom, že řešení popsané anti-vzorem nefunguje. Důsledek použití anti-vzoru může měnit problém v jiný, přidělávat další problémy, být neefektivní nebo problém pouze maskovat. To může vést k ohrožení kvality, harmonogramu, rozpočtu nebo výsledné funkcionality projektu.

2.5.1 Příklady anti-vzorů

Absentee manager

Tento anti-vzor nastává v případě, že má vývojový tým manažera, který není většinu času k dispozici. Pracovníci jsou nucení dělat rozhodnutí, na něž nemají kompetence, oprávnění nebo schopnosti, bez přítomnosti manažera, nebo musejí čekat, až bude k dispozici. Tím může dojít k nejrůznějším problémům jako například opoždění celkového projektu. [17].

Stovepipe System

Anti-vzor, který hovoří o špatné integrací více subsystémů. Jednotlivé subsystémy jsou napojeny přímo na sebe s žádnou mírou abstrakce. Tím je pak nemožné využít existující rozhraní pro komunikaci s ostatními subsystémy. Navíc je rozhraní silně závislé na dané konfiguraci systému [25].

Spaghetti Code

Jedná se o anti-vzor, který označuje nevhodnou strukturu zdrojového kódu, kde každá třída komunikuje s velkým množstvím ostatních tříd. V případě, že uděláme změnu v jedné třídě, dojde k chybám v ostatních třídách. Tím činí výsledný zdrojový kód velice špatně udržitelný a nesrozumitelný.

Architects Don't Code

Tento anti-vzor je založen na tom, že se systémoví architekti nepodílí na vývoji, např. protože jejich čas je drahý. Nakonec vzniká výsledná architektura pouze jako teoretický návrh a není podložena implementací. Při následné

implementaci můžou nastat některé problémy, které při teoretickém návrhu nebylo možno odhadnout [6].

2.6 Shrnutí

V této kapitole byly popsány základní pojmy z oblasti procesu vývoje softwaru. Dále zde byly představeny nejpoužívanější metodiky pro vývoj softwaru. V praxi se často můžeme setkat s některou z uvedených metodik, která je však upravena podle potřeb společnosti. Někdy může docházet i ke kombinaci určitých částí jednotlivých metodik. Na závěr této kapitoly byl představen koncept vzorů a anti-vzorů a některé reálné příklady. Později v práci budou vybrány jednotlivé vhodné anti-vzory, které budou analyzovány.

3 Projektová data, ALM nástroje a SPADe

S velkým nárůstem využívání softwarových produktů roste jejich velikost a složitost, tím dochází k rostoucímu počtu vývojových týmů a jednotlivých vývojářů, kteří spolu musejí spolupracovat. Často se jedná i o týmy, které jsou geograficky oddělené. Pro zajištění efektivního vývoje softwaru v rámci týmů je možné využít podpůrných Application Lifecycle Management (ALM) nástrojů. Tyto nástroje slouží primárně k vylepšení koordinace všech členů podílejících se na projektu. ALM nástroje mohou sloužit k následujícím činnostem: správa požadavků, návrh softwarové architektury, vývoj, testování, údržba, správa změn, integrace, nasazení nebo správa verzí.

V jednotlivých ALM nástrojích vzniká velké množství dat, která však většinou nejsou logicky propojena. Proto je vyvíjen nástroj Software Process Anti-pattern Detector (SPADe), který se snaží dolovat data z jednotlivých ALM nástrojů a následně je logicky propojovat.

V následující kapitole jsou představeny jednotlivé typy ALM nástrojů a reálné příklady, které se často využívají. V druhé části kapitoly je představen nástroj SPADe.

3.1 Nástroje pro správu verzí

Nástroj pro správu verzí (anglicky Version Control System; VCS) je systém, který zaznamenává změny souboru nebo sady souborů v čase tak, aby bylo možné se později k potřebné verzi vrátit. Umožní vrátit jednotlivé soubory nebo celý projekt zpět do předchozího stavu, porovnávat změny provedené v průběhu času, zjistit, kdo naposledy upravil něco, co nyní možná způsobuje problémy, kdo a kdy vytvořil diskutabilní část atd.

V dnešní době se můžeme nejčastěji setkat s nástrojem Git. Jedná se o konkrétní systém, sloužící ke správě verzí softwaru. Tento systém se ovládá pomocí příkazového řádku, a to může být pro některé uživatele nekomfortní. Často je tedy ovládání integrováno přímo do vývojového prostředí. Další variantou je použití externích nástrojů jako například SourceTree[1]. Dále také existují webové služby, které podporují vývoj softwaru za pomoci verzovacího

[1] `http://www.sourcetreeapp.com`

nástroje Git jako například GitHub[2], GitLab[3] atd.

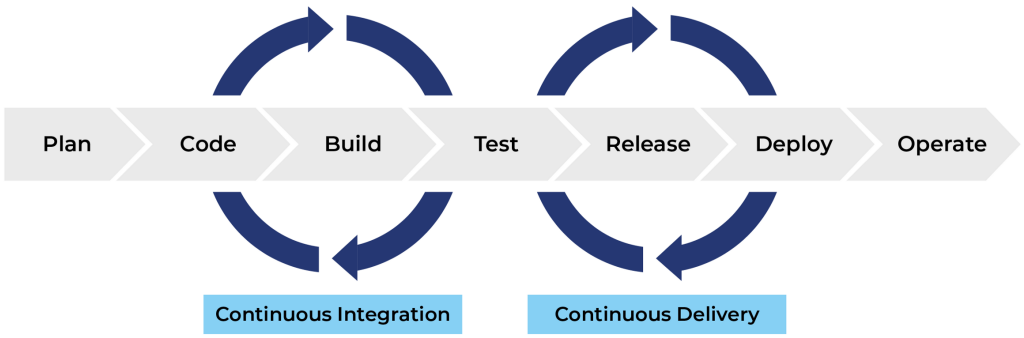
3.2 Nástroje pro správu nasazení

Nástroje pro správu nasazení nové verze produktu jsou převážně označovány jako Continues Integration a Continues Delivery (CI/CD). Tato zkratka představuje spojení průběžné integrace a průběžného doručení. Tato spojení si lze představit jako dvě fáze.

Fáze průběžné integrace (CI) je posloupnost kroků, které se provádí při jakékoli změně aplikace. Pokud dojde ke změně v aplikaci, mohlo také dojít k zanesení nových chyb. K ověření správnosti změny dojde v rámci průběžné integrace k sestavení a otestování všech komponent aplikace. Tím se mohou včasně odhalit jednotlivé chyby a ihned se napravit.

Pokud je ověřena funkčnost výsledné aplikace, dochází většinou k nasazení této aplikace do produkčního prostředí. Jedná se často o nahrání nové verze na server, konfiguraci nové verze nebo třeba i o otestování na produkčním prostředí. Je tedy zřejmé, že tato fáze může obsahovat spoustu manuálních kroků, při kterých může vzniknout chyba vlivem lidského faktoru. Nástroje průběžného doručení (CD) se snaží všechny kroky spojené s nasazením nové verze automatizovat, a tak i eliminovat možnost vzniku chyb. Proces CI/CD je zobrazen na obrázku číslo 3.1.

K nejčastěji používaným CI/CD nástrojům lze zařadit například Jenkins[4], Travis CI[5] nebo Circle CI[6] [12].



Obrázek 3.1: Proces CI/CD [12]

[2] http://www.github.com
[3] http://www.gitlab.com
[4] https://www.jenkins.io
[5] https://travis-ci.org
[6] https://circleci.com

3.3 Nástroje pro správu změn a problémů

Nástroje pro správu změn a problémů se anglicky označují jako issue tracker nebo také bug tracker. Issue lze chápat jako požadavek na změnu, úpravu či vylepšení produktu. Bug lze chápat jako požadavek na opravu chyby. Tyto nástroje slouží k zajišťování kvality, sledování hlášených softwarových chyb a požadavků při práci programátorů. Tyto nástroje můžeme rozdělit do dvou skupin, a to na veřejné a interní. Do veřejných nástrojů pro správu chyb mohou zadávat chyby koncoví uživatelé. Oproti tomu jsou systémy interní, kam jsou zadávány chyby pouze týmem, který se účastní realizace projektu. Každý záznam se v takovémto systému obvykle nazývá jako issue, ticket nebo také úkol.

Úkol může obsahovat nejrůznější atributy dle povahy projektu. Mezi nejčastější atributy lze zařadit následující:

- název,
- podrobný popis,
- předpokládaná délka řešení úkolu,
- dosud odpracovaný čas na úkolu,
- jméno zadavatele,
- jméno řešitele,
- stav úkolu (např. rozpracovaný, dokončený nebo k řešení),
- priorita úkolů.

Jedny z hlavních představitelů nástrojů pro správu změn a chyb jsou například Jira[7], Redmine[8] nebo BugZilla[9]. Všechny tyto nástroje obsahují základní správu změn a problémů, ale také obsahují spoustu dalších podpůrných funkcionalit pro komplexní vedení projektu. Mezi tyto funkcionality lze například zařadit tvorbu Gantova diagramu, vedení poznámek o celkovém projektu, integrace s testovacími nástroji nebo také integrace s nástroji pro správu verzí.

[7] `https://www.atlassian.com/software/jira`
[8] `https://www.redmine.org`
[9] `https://www.bugzilla.org`

3.4 Nástroje pro správu dokumentů

Jelikož v každém projektu dochází k častým změnám, je zapotřebí tyto změny formálně zanést do příslušných dokumentů. Pro tento účel by bylo možné využít nástrojů ke správě verzí. Ty jsou však primárně určeny pro programový kód. K tomuto účelu slouží nástroje pro správu dokumentů anglicky označovány jako document management systems zkráceně DMS.

Jedná se o nástroje, které slouží ke správě formálních či neformálních dokumentů souvisejících s projektem. Tyto nástroje lze využít k verzování, úpravě či pouze k nahlížení aktuálně platného dokumentu. V takovýchto systémech jsou také uchovávány informace o tom, kdo dokument, odstavec či úsek textu upravil nebo vytvořil.

Mezi konkrétní DMS nástroje patří například Google Docs[10], Dropbox[11] nebo Krystal DMS[12].

3.5 Nástroje pro komunikaci

Komunikace je jedním ze základních faktorů pro úspěch projektu. Bez dostatečné komunikace se razantně snižuje pravděpodobnost úspěchu projektu. Trendem dnešní doby je řešení většiny problémů distanční formou, tedy za pomocí audio hovorů, video hovorů nebo například pomocí sdílené obrazovky. Tím se dají razantně snížit náklady spojené s cestováním dotyčné osoby.

Komunikaci můžeme rozdělit na interní a externí. Externí komunikace je prováděna například se zákazníkem nebo dodavatelem. Interní komunikace je prováděna mezi kolegy, v rámci vývojového týmu nebo v rámci celého podniku.

Mezi nejrozšířenější nástroje pro komunikaci patří Microsoft Teams[13] (nástroj nahrazující Skype), Slack[14] nebo Google Meet[15]. Nesmíme ale opomenout také elektronickou poštu, která je stále jedním z nejpoužívanějších nástrojů pro komunikaci. Typickým zástupcem nástroje pro elektronickou poštu v podnikové sféře je Microsoft Outlook[16].

[10] `https://docs.google.com`
[11] `https://www.dropbox.com`
[12] `https://www.krystaldms.in/`
[13] `https://www.microsoft.com/microsoft-teams`
[14] `https://slack.com`
[15] `https://meet.google.com`
[16] `https://www.microsoft.com/microsoft-365/outlook`

3.6 SPADe

V této sekci je stručně popsán nástroj Software Process Anti-pattern Detector (SPADe), který je vyvíjen na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU). Porozumět tomuto nástroji je z hlediska této práce stěžejní, jelikož bude využit pro analýzu a detekci přítomnosti anti-vzorů v datech nástrojů pro řízení projektů.

3.6.1 Popis nástroje SPADe

Hlavním cílem nástroje SPADe je dolovat data z různých ALM nástrojů a analyzovat je s ohledem na potenciální přítomnost vzorů a anti-vzorů, jejichž výskyt může negativně ovlivnit průběh projektu nebo výsledného produktu.

V jednotlivých ALM nástrojích jsou uloženy veliké objemy projektových dat, která mezi sebou nejsou vždy logicky propojena. Cílem nástroje SPADe je dolovat data z jednotlivých ALM nástrojů, následně je mezi sebou logicky uspořádat a uložit do předdefinované struktury v datovém skladu. Nad výsledným datovým skladem by bylo následně umožněno provádět nejrůznější datové analýzy, zobrazovat statistiky projektů nebo například porovnávat vlastnosti podobných projektů, které mohou pomoci ke zvýšení úspěchu daného projektu [21].

Struktura nástroje SPADe

Strukturu nástroje SPADe lze rozdělit do čtyř základních částí:

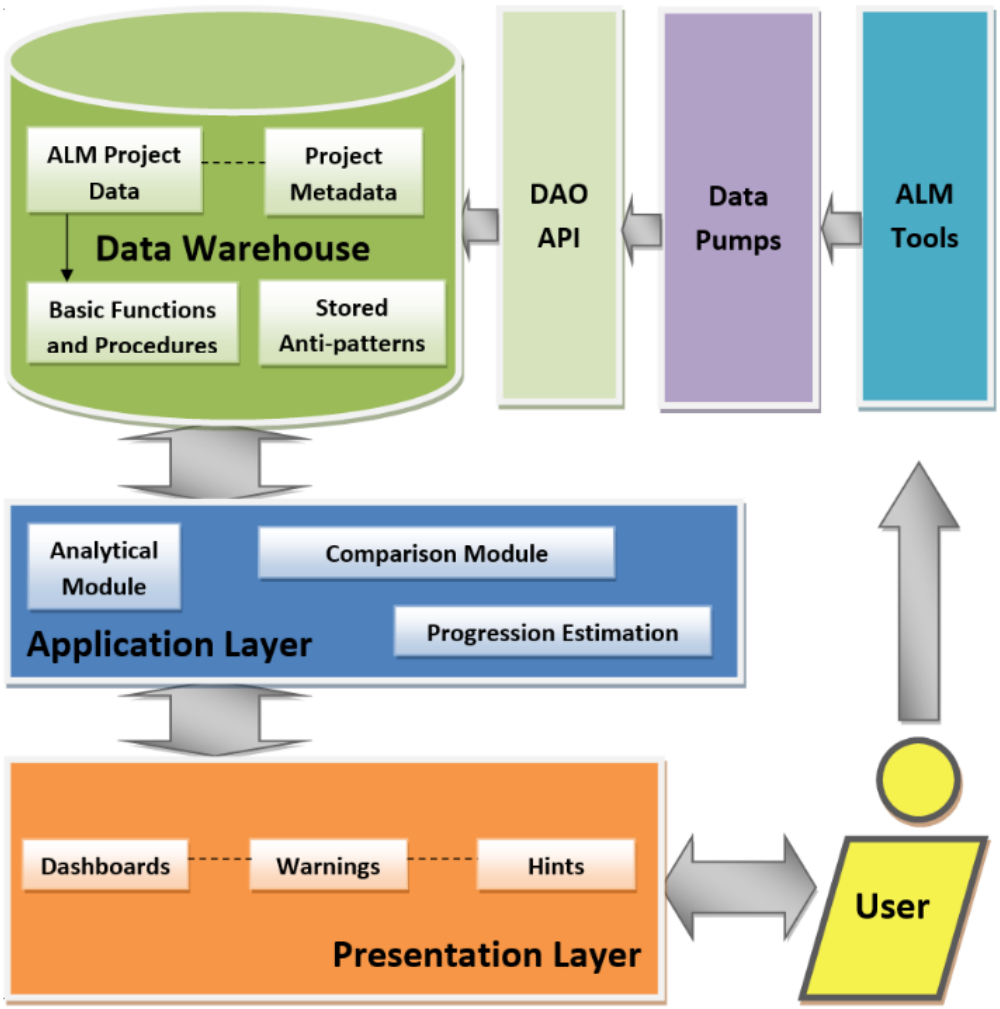
1. Dolování dat – První část pro dolování dat získává z jednotlivých ALM nástrojů pomocí standardního procesu extrakce, transformace a nahrání (Extract-Transform-Load; ETL) potřebné informace. ETL je proces, při kterém jsou dolována data z jednoho nebo více systémů a jsou transformována do jednotné formy, ve které jsou následně uložena [13]. Každý nástroj má nadefinovanou svoji vlastní ETL datovou pumpu, jelikož rozhraní jednotlivých ALM nástrojů není jednotné, viz podkapitola číslo 3.6.1.

2. Datový sklad – Pro okamžitou ale i pozdější analýzu získaných dat je zapotřebí uložit natěžená projektová data do datového úložiště. K této funkci slouží datový sklad, který má nadefinovanou strukturu uložených dat a poskytuje získané informace. Struktura datového skladu je popsána v podkapitole číslo 3.6.2.

3. Aplikační vrstva – Mezivrstva komunikující s prezentační vrstvou a datovým skladem. Zde jsou posílány jednotlivé dotazy do datového skladu a výsledky jsou touto vrstvou následně zpracovány a připraveny pro zobrazení na prezentační vrstvě.

4. Prezentační vrstva – S touto částí přímo interaguje uživatel a pomocí této části může uživatel procházet uložená data a zobrazovat si nejrůznější statistiky uložených projektů.

Výše popisovaná struktura nástroje SPADe je graficky zobrazena na obrázku číslo 3.2.



Obrázek 3.2: Architektura nástroje SPADe [21]

Dolování dat

Data o každém projektu jsou rozprostřena v různých ALM nástrojích, ze kterých je nutno tato data získat, zpracovat a uložit do datového skladu nástroje SPADe. Nástroj SPADe umožňuje dolovat data z následujících nástrojů:

- Git,
- SVN,
- GitHub,
- Bugzilla,
- Redmine,
- Jira,
- Assembla [21].

Dolování dat z jednotlivých nástrojů není vždy unifikované a nelze využít stejný přístup u všech nástrojů. Nejjednodušší přístup dolování dat je přímé připojení do databáze, kde si nástroj ukládá informace. Dalším přístupem je použít aplikačního rozhraní (Application Programming Interface; API) a pomocí různých dotazů získat potřebná data. Třetím přístupem je export dat z jednotlivých nástrojů do různých datových formátů (JSON[17], XML[18]). Následně pak data zpracovat a uložit do datového skladu. Posledním a nejsložitějším přístupem je přímo přistupovat na uživatelské rozhraní webových ALM nástrojů a pomocí web crawlingu (procházení a parsování jednotlivých HTML stránek webového rozhraní) dolovat potřebná data [21].

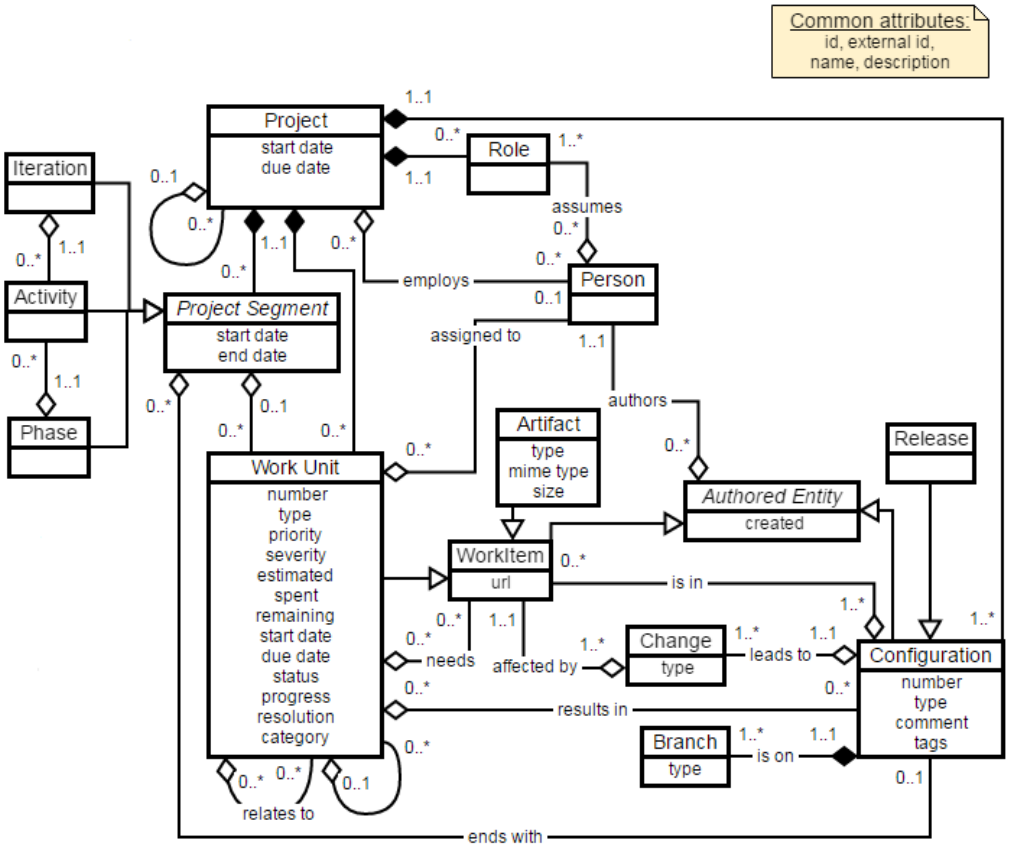
3.6.2 Datový sklad

Datový sklad je systém správy dat, který je navržen k uchovávání velkého množství dat, tzv. big data z různých zdrojů. Datový sklad lze následně použít k nejrůznější datovým analýzám. Jeden z největších rozdílů od klasické relační databáze je, že v datovém skladu jsou uložena i historická data například změny dané instance v průběhu času. [30].

[17] JavaScript Object Notation – způsob zápisu dat nezávislý na počítačové platformě, určený pro přenos dat.

[18] Extensible Markup Language – obecný značkovací jazyk, který se používá pro přenos a serializaci dat.

Datový sklad nástroje SPADe obsahuje celkem 47 entit, které obsahují komplexní data o jednotlivých projektech, získaná z ALM nástrojů zmíněných v kapitole číslo 3.6.1. V momentální verzi nástroje SPADe se datový sklad spíše blíží standardní relační databázi, ale z důvodu konzistence textu bude nadále používán pojem datový sklad. Základní struktura datového skladu je vyobrazena pomocí zjednodušeného doménového metamodelu datového skladu nástroje SPADe na obrázku číslo 3.3. Kompletní fyzický datový model je příliš komplexní, nepřehledný a není důležitý k porozumění fungování nástroje SPADe.



Obrázek 3.3: Doménový metamodel datového skladu nástroje SPADe [21]

Entity

Jak již bylo zmíněno výše, datový sklad obsahuje celkem 47 entit. Některé nejdůležitější entity jsou popsány v následujícím seznamu:

- Project – Entita představující základní informace o jednotlivých projektech. Na tuto entitu je přímo či nepřímo napojena většina entit

v datovém skladu. Obsahuje například název projektu, popis projektu, začátek nebo konec projektu.

- Work Unit – Entita obsahující informace o úkolu získané z ALM nástrojů. Zde jsou uloženy základní údaje o úkolu jako například název, autor, odhadovaný a strávený čas.
- Artifact – Entita obsahující základní informace o souborech či adresářích uložených v nástroji pro správu verzí nebo o wiki stránkách[19]. Dále také může obsahovat soubory, které byly přiloženy k jednotlivým wiki stránkám nebo úkolům.
- Configuration – Entita obsahující jakýkoliv stav projektu, respektive představující sadu změn provedených v jednom čase jednou osobou nad artefakty nebo tickety (Work Unit). Například commit, nahrání souboru nebo úprava wiki stránky.
- Iteration – Entita obsahující informace o jednotlivých iteracích daného projektu. Obsahuje například název iterace, popis iterace, začátek a konec iterace.
- Person – Entita obsahující informace o osobě pracující na daném projektu. Dále zde máme entitu Role, která udává roli příslušné osoby v daném projektu.
- Work Item – Rodičovská entita pro entity Artifact, Configuration a Work Unit. Obsahuje přímou URL adresu pro daný záznam například wiki stránku nebo soubor v nástroji pro správu verzí.

Pohledy

Samotný datový sklad obsahuje mnoho entit, které jsou mezi sebou propojeny mnoha vazbami. Pro zjednodušení a přehlednější práci obsahuje datový sklad osm pohledů, které spojují několik tabulek do logických celků. Tato sada pohledů je součástí SPADe a byla vytvořena v rámci předešlé diplomové práce [4]. Jednotlivé pohledy jsou popsány v následující seznamu.

- `artifactView` – Tento pohled zobrazuje všechny artefakty k danému projektu. Důležitým atributem, který je obsažen v tomto pohledu je atribut `artifactClass`. Tento atribut slouží ke kategorizaci artefaktů podle typu, např. wiki stránka, soubor, atd. Záznam obsahuje všechny

[19] Jednoduché textové stránky umožňující týmu zaznamenávat potřebné údaje například poznámky ze schůzek.

Tabulka 3.1: Atributy pohledu `artifactView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	název daného artefaktu
`description`	longtext	obsah daného artefaktu
`create`	datetime	datum a čas vytvoření artefaktu
`url`	varchar(255)	umístění artefaktu v projektovém repozitáři
`artifactClass`	varchar(255)	zda se jedná o soubor nebo o wiki stránku
`mimeType`	varchar(255)	typ souboru (php, png, text, html,...)
`size`	bigint(20)	velikost artefaktu
`authodId`	bigint(20)	id autora, který artefakt vytvořil
`authorName`	varchar(255)	jméno autora, který artefakt vytvořil
`projectId`	bigint(20)	reference na projekt

Tabulka 3.2: Atributy pohledu `personView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	jméno osoby
`projectId`	bigint(20)	reference na projekt

wiki stránky projektu. U každého záznamu je dále uvedeno například, kdo je autorem, k jakému projektu záznam patří, velikost nebo obsah. Kompletní výpis jednotlivých atributů pohledu je zobrazen v tabulce číslo 3.1.

- `personView` – Jelikož osoba nemá přímou vazbu na projekt ve fyzickém datovém modelu, existuje pohled `personView`. Pohled umožňuje zobrazit jména lidí, která pracují na daném projektu. Kompletní výpis jednotlivých atributů je zobrazen v tabulce číslo 3.2.

- `personWithRolesView` – Stejný jako předchozí pohled `personView`, ale navíc je zde uvedena role každé osoby včetně její klasifikace nutné pro vzájemné porovnání rolí napříč daty z různých nástrojů. Například vývojář, mentor nebo zákazník. Kompletní výpis jednotlivých atributů je zobrazen v tabulce číslo 3.3.

- `workUnitView` – Nejdůležitějším pohledem je `workUnitView`. Tento pohled zobrazuje všechny úkoly, které byly v rámci každého projektu vykonány. Je zde uveden název, popis, autor, řešitel nebo také odhado-

Tabulka 3.3: Atributy pohledu `personWithRolesView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	jméno osoby
`projectId`	bigint(20)	reference na projekt
`role`	varchar(255)	název role dané osoby
`roleClass`	varchar(255)	třída role dané osoby
`roleSuperClass`	varchar(255)	nadřazená třída role dané osoby

Tabulka 3.4: Atributy pohledu `workUnitView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	název úkolu
`description`	longtext	popis úkolu
`dueDate`	date	předpokládané datum splnění úkolu
`estimatedTime`	double	odhadovaný čas úkolu v hodinách
`spentTime`	double	strávený čas na úkolu v hodinách
`projectId`	bigint(20)	reference na projekt
`autorId`	bigint(20)	id autora úkolu
`assigneeId`	bigint(20)	id vykonavatele úkolu
`iterationName`	varchar(255)	název iterace, ve které je úkol naplánována
`statusName`	varchar(255)	status daného úkolu
`priorityName`	varchar(255)	název priority daného úkolu

vaný a skutečný čas úkolu. Výpis nejdůležitějších atributů je zobrazen v tabulce číslo 3.4.

- `configurationView` – Pohled poskytující informace o konfiguracích v jednotlivých projektech a o tom, kdo změnu konfigurace provedl. Kompletní výpis jednotlivých atributů je zobrazen v tabulce číslo 3.5.

- `commitedConfigView` – Pohled slouží k výpisu jen těch konfigurací, které obsahují druhý časový údaj. Konkrétně vykázání času stráveného prací na projektu. Kompletní výpis jednotlivých atributů je zobrazen v tabulce číslo 3.6.

- `commitView` – Pohled, který rozšiřuje pohled `commitedConfigView`. Pohled obsahuje informace o všech změnách, které byly provedeny v repozitáři každého projektu. U každého záznamu je uveden napří-

Tabulka 3.5: Atributy pohledu `configurationView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	název konfigurace
`type`	varchar(31)	o jaký typ konfigurace se jedná (Configuration/Committed/Commit)
`description`	longtext	popis provedené změny (komentář)
`created`	datetime	čas a datum provedení změny
`authorId`	bigint(20)	cizí klíč představující id autora
`authorName`	varchar(255)	jméno autora
`relationName`	varchar(255)	vztah mezi další osobou (mimo autora) a změnou
`relatedId`	bigint(20)	identifikátor spřízněné osoby
`relatedName`	varchar(255)	jméno spřízněné osoby
`projectId`	bigint(20)	přiřazení konfigurace k projektu

Tabulka 3.6: Atributy pohledu `commitedConfigView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`name`	varchar(255)	název konfigurace
`type`	varchar(31)	o jaký typ konfigurace se jedná
`description`	longtext	popis provedené konfigurace
`created`	datetime	čas a datum vytvoření záznamu
`authorId`	bigint(20)	cizí klíč představující id autora
`authorName`	varchar(255)	jméno autora
`relationName`	varchar(255)	vztah mezi `authorName` a `relatedName`
`relatedId`	bigint(20)	identifikátor odpovídající osoby
`relatedName`	varchar(255)	jméno odpovídající osoby
`projectId`	bigint(20)	přiřazení konfigurace k projektu
`commited`	datetime	datum uložení konfigurace tzv. „commit“

Tabulka 3.7: Atributy pohledu `commitView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`type`	varchar(31)	o jaký typ záznamu se jedná
`name`	varchar(255)	jednoznačný idetifikátor commitu
`description`	longtext	popis daného commitu
`created`	datetime	datum a čas vytvoření záznamu
`authorId`	bigint(20)	id autora commitu
`authorName`	varchar(255)	jméno autora commitu
`relationName`	varchar(255)	jméno odpovídající osoby
`relatedId`	bigint(20)	identifikátor odpovídající osoby
`relatedName`	varchar(255)	vztah mezi `authorName` a `relatedName`
`projectId`	bigint(20)	reference na projekt
`committed`	datetime	datum a čas provedení commitu
`isRelease`	bit(1)	příznak udávající uvolnění nové verze produktu
`tag`	varchar(255)	pokud má daný commit tag (označení významného commitu vývojáři), tak je uložený zde
`branch`	varchar(255)	název větvě, do které je commitováno
`main`	bit(1)	příznak udávající commit do hlavní větve

klad popis změny (commit message), autor změny nebo do jaké větve (branche) byla změna provedena. Kompletní výpis jednotlivých atributů je zobrazen v tabulce číslo 3.7.

- `fieldChangeView` – Poslední pohled obsahuje záznamy o konkrétní změně jednoho atributu artefaktu nebo úkolu. Z těchto změn je složena každá konfigurace, z tohoto důvodu je v tomto pohledu zahrnuto i několik jejích atributů (čas, autor atd.). Výpis nejdůležitějších atributů je zobrazen v tabulce číslo 3.8.

Tabulka 3.8: Atributy pohledu `fieldChangeView`

Název atributu	Datový typ	Popis atributu
`id`	bigint(20)	jednoznačný identifikátor záznamu
`type`	varchar(31)	o jaký typ konfigurace se jedná
`description`	longtext	komentář příslušné změny
`field`	varchar(255)	jaký atribut byl změněn
`itemId`	bigint(20)	id udávající k jakému artefaktu/úkolu/konfiguraci se změna váže
`newValue`	longtext	nová hodnota příslušného pole
`oldValue`	longtext	stará hodnota příslušného pole
`projectId`	bigint(20)	jednoznačný identifikátor projektu
`changeName`	bigint(20)	popis změny (vytvoření, úprava apod.)

3.7 Shrnutí

V této kapitole byl nejprve představen pojem ALM nástrojů, základní skupiny těchto nástrojů a konkrétní produkty. Dále byl popsán nástroj SPADe, jeho struktura, datový sklad atd. Pochopení tohoto nástroje a struktury uložených dat je pro tuto práci klíčové, jelikož s využitím tohoto nástroje budou detekovány jednotlivé anti-vzory. V dalších částech práce se bude nejčastěji pracovat s jednotlivými pohledy, které zde byly popsány.

4 Návrh detekce anti-vzorů

V této kapitole jsou nejprve popsány základní faktory pro výběr vhodných anti-vzorů, které by bylo možné detekovat pomocí nástroje SPADe. Na základě těchto faktorů je vybrána vhodná sada kandidátních anti-vzorů. Jednotlivé anti-vzory jsou zde podrobně popsány a analyzovány pro jejich detekci v projektových datech později v práci.

4.1 Výběr vhodných anti-vzorů

V literatuře lze nalézt velké množství nejrůznějších anti-vzorů a jejich definic, například v rámci výzkumu na ZČU FAV KIV byl vytvořen katalog obsahující několik desítek definic anti-vzorů [6]. Nejdůležitějším faktorem pro výběr vhodných anti-vzorů je přítomnost potřebných dat v nástroji SPADe. V případě že nástroj SPADe neobsahuje potřebné informace k vybranému anti-vzoru, je nemožné ho detekovat. Jako příklad zde může sloužit anti-vzor s názvem Absentee Manager. Tento anti-vzor popisuje negativní dopady absence nebo nedostatečnou komunikaci projektového manažera. Následně mohou nastat situace, kdy musí samotný vývojový tým rozhodovat o něčem, na co nemá kvalifikaci nebo zkušenosti [17]. Pro detekci tohoto anti-vzoru by bylo nutné mít v datovém skladu uložena data z komunikačních nástrojů, která SPADe v současné době netěží, viz kapitola číslo 3.6.

Dalším faktorem mohou být nekonzistentní data u jednotlivých projektů. Každý tým si v rámci projektu volí své vlastní konvence například používání kategorií, tagů, popis úkolů apod. Může tedy dojít ke kolizi při používání některých atributů, které jednotlivé týmy používají. Jako příklad zde poslouží anti-vzor s názvem Indeferent Specialist známý také jako Domain Specialist. Anti-vzor popisuje člena týmu, který je specialistou pouze na jednu věc a nechce se učit novým věcem mimo jeho specializaci. Často znevažuje ostatní technologie a snižují vážnost věcí, které se nezaobírají jeho specializací [2]. V tomto případě je velice těžké rozřadit jednotlivé úkoly do doménových specializací, jelikož neexistuje žádný standardní atribut napříč ALM nástroji, který by používaly všechny týmy konzistentně. Navíc většina ALM nástrojů umožňuje dodefinovat vlastní atributy, na jejichž třídění a sémantickou analýzu nejsou datové pumpy SPADe v současnosti vybaveny.

Dalším faktorem pro výběr vhodných anti-vzorů je absence potřebných dat nejen v nástroji SPADe, ale v jakémkoliv jiném ALM nástroji. Jedná se zde o anti-vzory, které lze zařadit do skupiny osobnostních anti-vzorů

(Human Anti-patterns). Toto je skupina anti-vzorů, které popisují špatné či nevhodné osobnostní vlastnosti nebo předpoklady dané osoby [17]. Pro detekci takovýchto anti-vzorů/osob by bylo zapotřebí využít některých sociologických metod a analýzu těchto výsledků pro zjištění takovéto osoby a detekci anti-vzoru.

Dle analýzy známých anti-vzorů na základě předchozích faktorů byla vybrána sada sedmi anti-vzorů, které se budou dále analyzovat v rámci této práce. Jedná se konkrétně o následující anti-vzory:

- Business As Usual,
- Long Or Non-Existant Feedback Loops,
- Ninety-Ninety Rule,
- Road To Nowhere,
- Specify Nothing,
- Too Long Sprint,
- Varying Sprint Length.

4.2 Popis a návrh detekce pro vybrané anti-vzory

V rámci této části bude popsána základní charakteristika každého anti-vzoru, jeho následky a poté bude popsána jeho možná detekce v projektových datech. V rámci každého návrhu detekce bude vždy uvedena jedna či více prahových hodnot. Prahové hodnoty si lze představit jako nastavení striktnosti detekce. Dále bude u každé detekce zobrazen diagram, který přehledně znázorňuje postup detekce daného anti-vzoru.

4.2.1 Business As Usual

Popis

Anti-vzor s názvem Business As Usual, také známý jako No Sprint Retrospective, se zabývá absencí týmové schůzky, nazývané retrospektiva, po každé iteraci. Po uplynulé a před plánováním následující iterace by mělo dojít ke schůzce celého týmu, kde se celý tým zamyslí na minulou iterací a vytvoří návrhy na zlepšení užívaných postupů a praktik. Zlepšení jsou představena

při plánování následující iterace a na závěr iterace jsou opět vyhodnoceny [11].

Následky

Pokud nedochází k retrospektivám po každé iterací, tak nemůže docházet k žádnému vylepšení a změnám při vývoji softwaru. Tím se nemůže zvýšit produktivita vývojového týmu, naopak může spíše klesat. Dále se eliminuje jedna ze základních složek agilního vývoje a to je přizpůsobivost. V případě výskytu tohoto anti-vzoru může vést k několika dalším anti-vzorům, jelikož neexistuje žádný systematický způsob, jak zjistit co funguje a co nefunguje [11].

Detekce

Pro detekci tohoto anti-vzoru je nutné se zaměřit na informace o konaných retrospektivách v daném projektu. První variantou je zjistit konání jednotlivých retrospektiv pomocí jednotlivých úkolů. Budeme tedy hledat úkoly, které mohou představovat revizní schůzku týmu pomocí příslušných podřetězců v názvu úkolu. V případě neúspěchu hledání podřetězců by bylo možné zjistit, jestli si na daném úkolu zaznamenávají čas všichni členové a zda je tento úkol bez commitu do repozitáře. Na základě další analýzy sady projektových dat však bylo zjištěno, že každý tým má zvolené jiné konvence a zaznamenávat čas na úkolu může pouze jeden člen týmu za všechny. Tím by se docílilo velkého množství falešných negativních nálezů anti-vzoru. Proto byla zvolena varianta pouze s hledáním retrospektiv pomocí podřetězce v názvu úkolu.

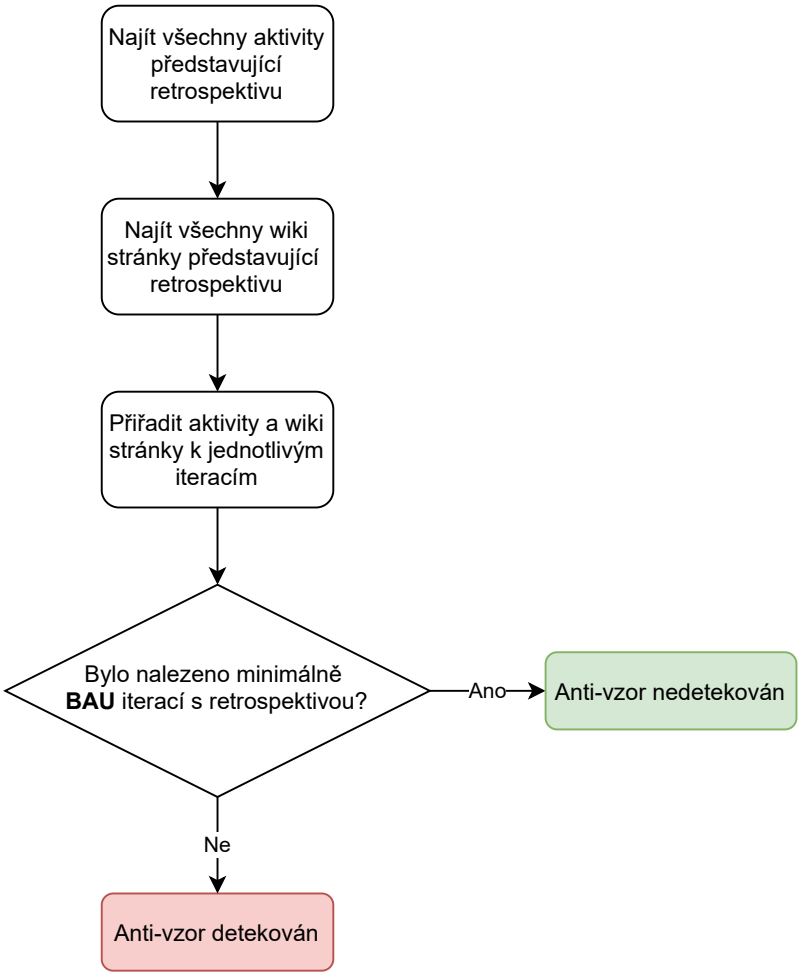
Dále si může tým zaznamenávat poznámky z jednotlivých retrospektiv pouze do wiki stránek. Je tedy zapotřebí nalézt všechny wiki stránky, které v sobě obsahují zmínku o retrospektivě. Jedna wiki stránka může obsahovat poznámky z jedné nebo více retrospektiv. Je tedy nutné se zaměřit na to, kdy byla daná stránka upravena. Tím eliminujeme problém s jednou stránkou pro více retrospektiv.

Nalezené úkoly a wiki stránky následně přiřadíme do jednotlivých iterací. Úkoly jsou přiřazeny podle toho, na jakou iteraci byly naplánovány a wiki stránky jsou přiřazeny k jednotlivým iteracím podle toho, kdy byly upravovány. V ideálním případě by každá iterace měla obsahovat alespoň jednu zmínku o konání retrospektivy. Dále musí být zvolena prahová hodnota (s označením `BAU`), která bude představovat minimální počet iterací s detekovanou retrospektivou. Pokud bude minimální počet iterací menší

než prahová hodnota, je anti-vzor detekován. V opačném případě není detekován. Diagram detekce je zobrazen na obrázku číslo 4.1.

Prahové hodnoty

- `BAU` – minimální počet iterací s retrospektivou



Obrázek 4.1: Stavový diagram návrhu detekce pro anti-vzor Business As Usual

4.2.2 Long Or Non-Existant Feedback Loops

Popis

Tento anti-vzor se zabývá příliš dlouhou nebo vůbec žádnou zpětnou vazbou od zákazníka. Zpětná vazba je jednou z nejdůležitějších faktorů pro úspěšný

softwarový produkt. Zpětná vazba by měla být ideálně prováděna po dokončení nějaké větší ucelené části. Nejčastěji může docházet ke komunikaci se zákazníkem po každé vykonané iteraci. Tento problém může být způsoben zhotovitelem nebo zadavatelem projektu. Ze strany zhotovitele se jedná o nedostatečnou nebo nedbalou iniciativu jednotlivých schůzek se zákazníkem. Ze strany zadavatele může nastat problém, kde zadavatel projektu nemá čas nebo zájem vykonávat jakoukoliv zpětnou vazbu zhotoviteli projektu [11].

Následky

V případě dlouhé nebo žádné zpětné vazby může docházet k velkým změnám v pozdní fázi vývoje, jelikož zadavatel není s výslednou prací spokojen. Detekce tohoto anti-vzoru může potenciálně představovat spoustu přepracování produktu z důvodu špatného pochopení potřeb zákazníka na začátku projektu. Také může vést k vytvoření nesprávných funkcí, které pro zákazníka nemají žádnou hodnotu.

Detekce

Pro detekci tohoto anti-vzoru je nutné určit, kdy byly konány schůzky se zákazníkem. Schůzky lze detekovat ze dvou zdrojů. Prvním zdrojem jsou úkoly a druhým zdrojem jsou wiki stránky.

Nejprve jsou nalezeny všechny úkoly, které mohou představovat, dle názvu, schůzku se zákazníkem. Dále tyto úkoly přiřadíme k jednotlivým iteracím. Pokud každá iterace obsahuje alespoň jednu schůzku se zákazníkem, tak není anti-vzor detekován. Toto je ideální případ. V případě, že nemá každá iterace detekovanou schůzku se zákazníkem, tak mohou nastat dva případy. Prvním případ je, že tým nezaznamenává schůzky jako úkoly a druhý případ je, že se schůzky konaly s velkými rozestupy. Je tedy nutné se rozhodnout, od jakého počtu nalezených úkolů se bude hledat ve wiki stránkách, a kdy budeme měřit rozestupy mezi schůzkami. Pro tento účel bude sloužit prahová hodnota s označením $\texttt{LON}_1$. Pokud alespoň v $\texttt{LON}_1$ iteracích nastala schůzka se zákazníkem, tak dojde k měření rozestupů těchto úkolů. Rozestupy jednotlivých úkolů nesmějí přesáhnout prahovou hodnotu s označením $\texttt{LON}_2$. Stejný rozestup je kontrolovaný mezi začátkem projektu a první schůzkou se zákazníkem. Pokud je překročen tento rozestup, tak je anti-vzor detekován.

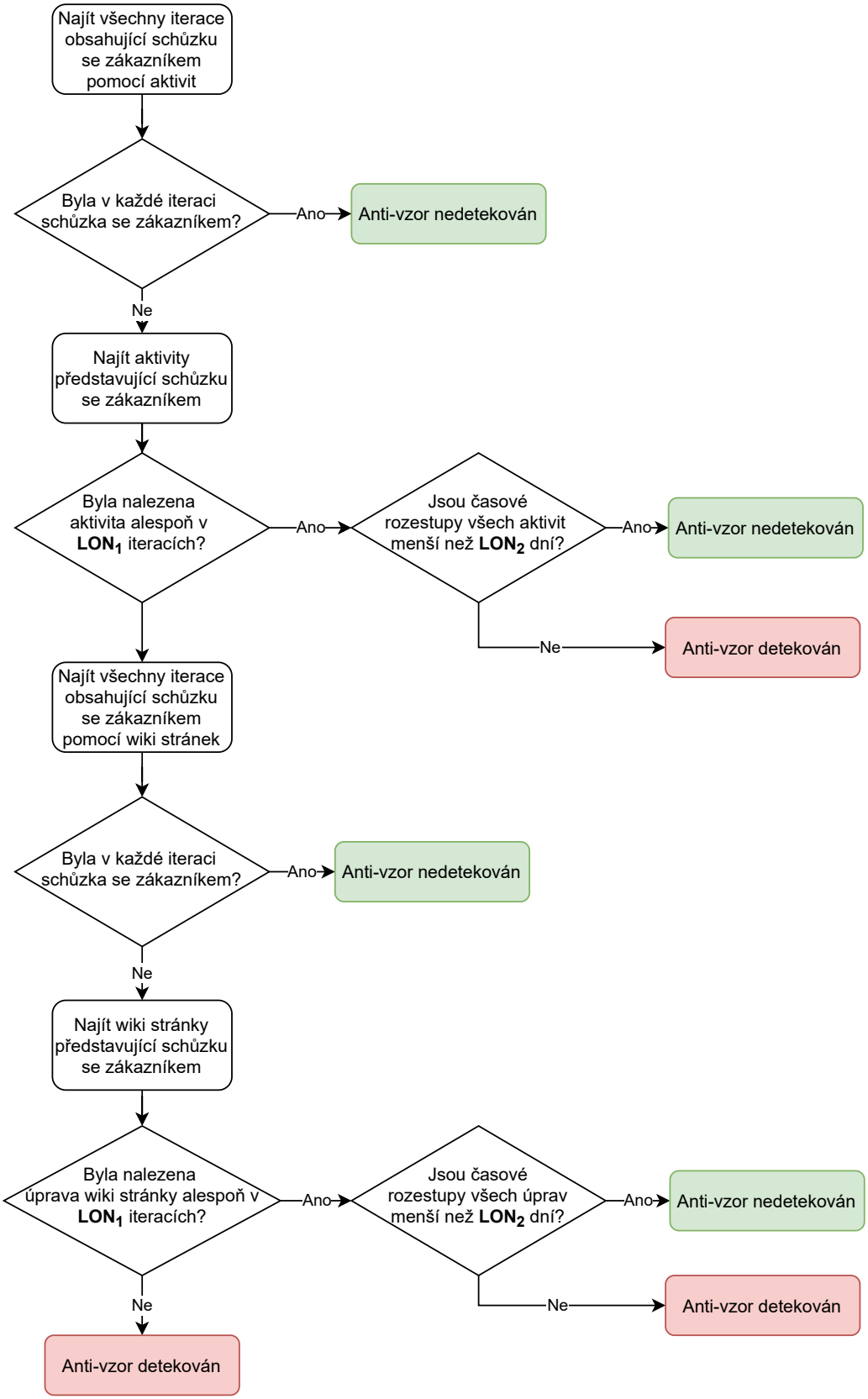
V případě že nastala schůzka méně jak v $\texttt{LON}_1$ iteracích, dojde k hledání ve wiki stránkách. Musíme tedy najít všechny wiki stránky, které by mohly představovat záznam ze schůzky se zákazníkem (pomocí názvu a obsahu). Jednotlivé wiki stránky přiřadíme k iterací podle data vytvoření nebo upravení. Pokud je u každé iterace nalezena alespoň jedna wiki stránka, není

anti-vzor detekován. Dále se zjistí, zda byly nalezeny wiki stránky alespoň v $\texttt{LON}_1$ iteracích. Pokud nebyl nalezen dostatečný počet wiki stránek, tak je anti-vzor detekován. Pokud byl nalezen dostatečný počet wiki stránek, přichází na řadu měření časového rozestupu mezi úpravami nalezených wiki stránek. Pokud tyto rozestupy nepřesahují délku $\texttt{LON}_2$ dní, není anti-vzor detekován.

Diagram detekce je zobrazen na obrázku číslo 4.2.

Prahové hodnoty

- $\texttt{LON}_1$ – minimální počet iterací, kde se konala alespoň jedna schůzka se zákazníkem,
- $\texttt{LON}_2$ – maximální rozestup schůzek se zákazníkem (počet dní).



Obrázek 4.2: Stavový diagram návrhu detekce pro anti-vzor Long Or Non-Existant Feedback Loops

4.2.3 Ninety-Ninety Rule

Popis

Anti-vzor Ninety-Ninety Rule se zabývá množstvím stráveného času na hotových úkolech v závislosti na množství času pro dokončení zbylých úkolů. Konkrétní definice tohoto anti-vzoru říká, že 90 % implementačních úkolů zabere 90 % času a zbývajících 10 % implementačních úkolů zabere další 90 % času. Může také nastat situace, že funkcionalita už je skoro hotová, některé úkoly už jsou uzavřené, ale stále se čeká pouze na uzavření jednoho úkolu, který je však dlouhou dobu otevřen [18].

Tento anti-vzor může nastat v projektech, kde byla na začátku projektu podceněna nebo chybně provedena analýza náročnosti implementace jednotlivých funkcionalit. Implementace jednotlivých funkcionalit zabere tedy mnohem více času, než bylo odhadováno.

Následky

Z definice tohoto anti-vzoru vyplývá, že pro finální dokončení produktu bude zapotřebí přibližně o 80 % více času, než bylo plánováno (90 % + 90 % - 100 %). Následkem toho dojde tedy k masivnímu opoždění projektu.

Detekce

Detekce tohoto anti-vzoru je založena na analýze vzniklých následků. Konkrétně následku, který je založen na špatných odhadech složitosti jednotlivých implementačních úkolů.

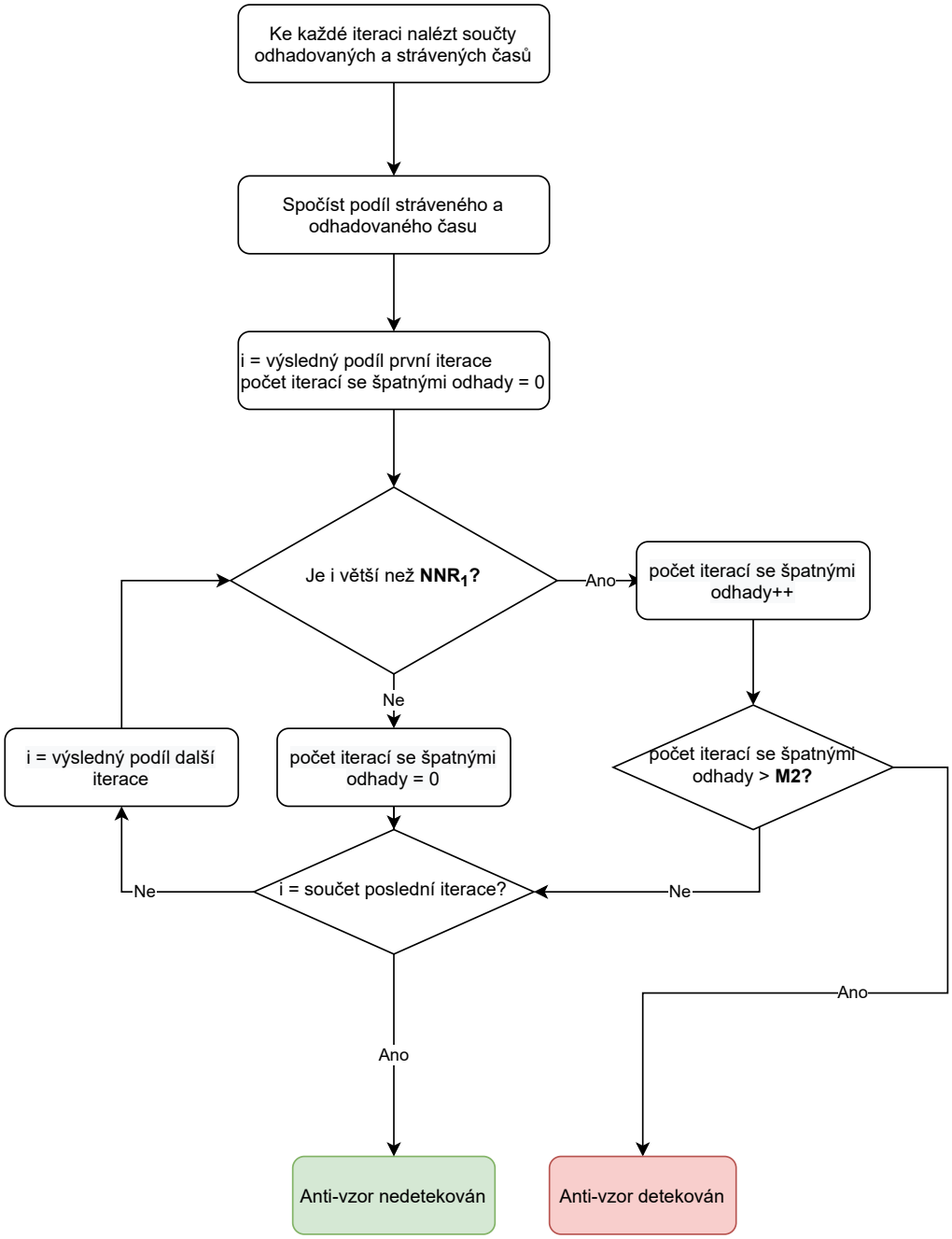
Nejprve nalezneme všechny implementační úkoly, které přiřadíme do jednotlivých iterací. Následně spočteme podíly stráveného a odhadovaného času na těchto úkolech v jednotlivých iteracích. Následně dojde ke kontrole jednotlivých podílů, které nesmějí přesáhnout prahovou hodnotu $\texttt{NNR}_1$ (prahová hodnota udávající maximální hodnotu podílu stráveného a odhadovaného času). Pokud podíl překročí prahovou hodnotu $\texttt{NNR}_1$, je tato iterace označena za signifikantní. V případě že nalezne $\texttt{NNR}_2$ počet signifikantních iterací po sobě, je anti-vzor detekován. V opačném případě není detekován.

Detekce odpovídá scénáři, kde by měl tým vyhodnotit špatné odhady a v následujících iteracích tyto odhady zlepšit. V případě, že bude několik po sobě jdoucích iterací ignorovat tento problém, tak může následně dojít k neúspěchu projektu.

Diagram detekce tohoto anti-vzoru je zobrazen na obrázku číslo 4.3.

Prahové hodnoty

- $\texttt{NNR}_1$ – maximální hodnota podílu stráveného a odhadovaného času v jednotlivých iteracích,
- $\texttt{NNR}_2$ – maximální počet po sobě jdoucích iteracích, kde podíl stráveného a odhadovaného času překročí prahovou hodnotu $\texttt{NNR}_1$.



Obrázek 4.3: Stavový diagram návrhu detekce pro anti-vzor Ninety-Ninety Rule

4.2.4 Road To Nowhere

Popis

Jedná se o anti-vzor, který se zabývá absencí projektového plánu a samotného plánování [8]. Projektový plán je dokument, který se sepisuje ve vět-

šině případů na začátku projektu. Dokument obsahuje jednotlivé cíle a výstupy projektu a jak se k těmto výstupům máme dostat. Projektový plán lze sepsat podle jasně definovaných standardů například PMBOK[1] [23] nebo PRINCE2[2] [24]. Hlavně by měl obsahovat čtyři základní otázky důležité pro projekt a jeho řízení:

1. Proč se daný projekt realizuje?
2. Co je výstupem nebo cílem projektu?
3. Kdo se na realizaci projektu bude podílet a jaké má povinnosti?
4. Kdy by měl být projekt dokončen a jaké budou jednotlivé milníky [9]?

Následky

V případě absence projektového plánu nebo jakéhokoliv plánování může v průběhu projektu docházet ke zmatkům a krizím ve vedení. Pokud není jasně definovaný projektový plán s průběžnými termíny, nejsme schopni vyhodnotit, zda je projekt ve skluzu nebo není. Tím může následně docházet k velikému zpoždění v závěru projektu. Dalším problémem, který může nastat při absenci projektového plánu, je zmatek při vedení projektu. Projektový management neví, v jaké části se projekt nachází a na čem se v dané fází má pracovat. Zmatek a chaos se následně může přenést i na vývojový tým, který neví, co má dělat, a tím klesá i produktivita práce [8].

Detekce

Hlavním zdrojem informací pro detekci tohoto anti-vzoru budou informace o úkolech a wiki stránkách. V prvním kroku detekce je třeba nalézt všechny wiki stránky, které by mohly obsahovat projektový plán. Toho docílíme prohledáváním názvu a obsahu wiki stránek, kde budeme hledat podřetězce připomínající projektový plán. Pokud počet nalezených wiki stránek představující projektový plán bude větší nebo roven prahové hodnotě s názvem $\texttt{RTN}_1$, není anti-vzor detekován.

Může také nastat situace, kde tým neuložil projektový plán do wiki stránek a vede ho v nějakém externím nástroji. Musíme tedy zjistit, zda existuje nějaký úkol, který by představoval vytvoření projektového plánu. Projektový plán by měl být ideálně vytvořený na začátku projektu, takže se zaměříme

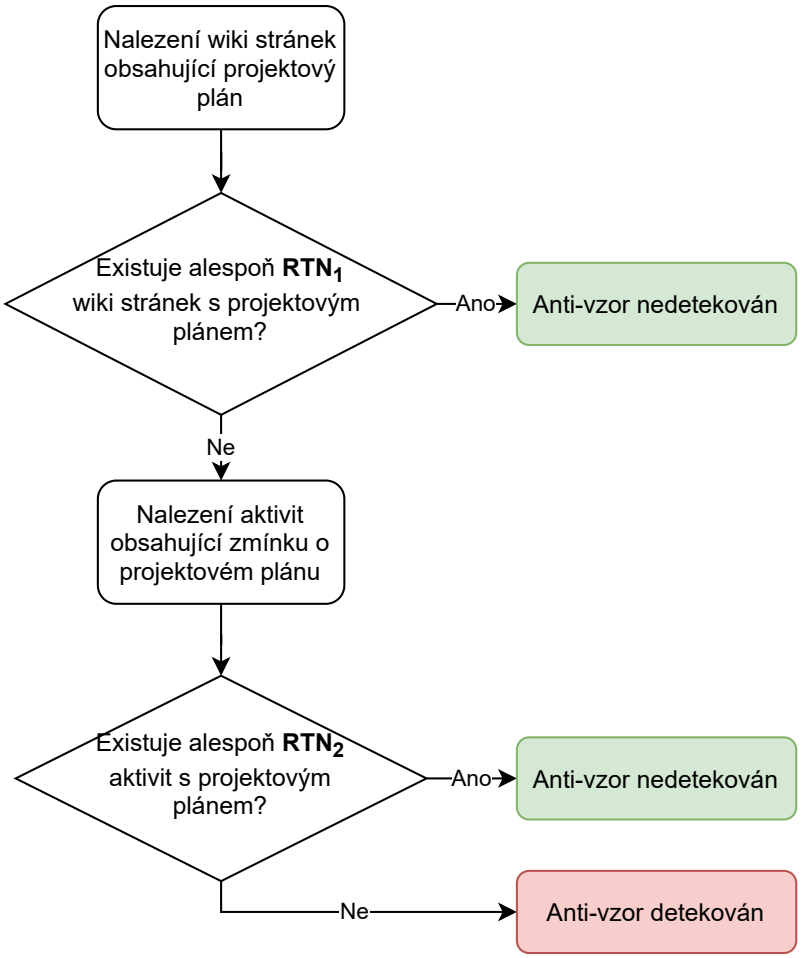
[1] Project Management Body of Knowledge – Jedná se o mezinárodně uznávaný standard řízení projektů, který vydává institut Project Management Institute (PMI).

[2] Projects in Controlled Environments 2nd version – Flexibilní metodika řízení projektů použitelná pro řízení jakéhokoli projektu bez ohledu na jeho velikost, složitost, odvětví.

na úkoly v prvních dvou iteracích, které obsahují podřetězec představující vytvoření projektového plánu. Pokud bude počet nalezených úkolů připomínající vytvoření projektového plánu větší nebo roven prahové hodnotě s názvem $\texttt{RTN}_2$, není anti-vzor detekován. V opačném případě je anti-vzor detekován. Diagram detekce je zobrazen na obrázku číslo 4.4.

Prahové hodnoty

- $\texttt{RTN}_1$ – minimální počet wiki stránek obsahující projektový plán,
- $\texttt{RTN}_2$ – minimální počet úkolů obsahující projektový plán.



Obrázek 4.4: Stavový diagram návrhu detekce pro anti-vzor Road To Nowhere

4.2.5 Specify Nothing

Popis

Tento anti-vzor se zaobírá problematikou písemné specifikace projektu. V některých organizacích jsou považovány písemné specifikace projektu za naprosto nedůležité a předpokládá se, že členové týmu znají specifikaci projektu zpaměti a není ji nutné sepisovat. V rámci malých projektů a vývojových týmů může tento přístup fungovat. Pokud se však jedná o rozsáhlejší produkty či vývojové týmy, je zcela nemožné vést specifikaci projektu pouze v hlavách členů projektového týmu [22]. V některých případech je navíc specifikace projektu závazným dokumentem, na kterém se shodne zadavatel a dodavatel projektu. Proto je písemná forma tohoto dokumenty považována za velice důležitou.

Následky

Při absenci písemné verze dokumentu specifikace projektu může docházek k rozporům v názorech jednotlivých členů týmu na implementaci dané části produktu. Jelikož neexistuje žádná centrální verze tohoto dokumentu, není možné určit, kdo má pravdu. Výsledkem může být implementace části produktu, která není pro zákazníka vůbec důležitá, nebo si ji zákazník představoval jinak.

Dalším problémem, který může nastat, jsou personální změny v týmu. Pokud zná specifikaci pouze jeden člen týmu, který se rozhodne tým opustit, je nutné specifikaci předat někomu jinému. Pokud k předání nedojde, musí dojít ke schůzce se zákazníkem a specifikaci znovu nadefinovat. Pro zákazníka to může značit neprofesionalitu dodavatele a ztrátu důvěry v něj.

Detekce

Detekce tohoto anti-vzoru bude rozdělena do třech hlavních částí. V první části se pokusíme nalézt wiki stránky, které představují specifikaci projektu. Toho docílíme hledáním různých podřetězců v názvu či obsahu stránky. Pokud bude počet nalezených úkolů větší nebo roven prahové hodnotě s označením $\texttt{SPN}_1$, není anti-vzor detekován.

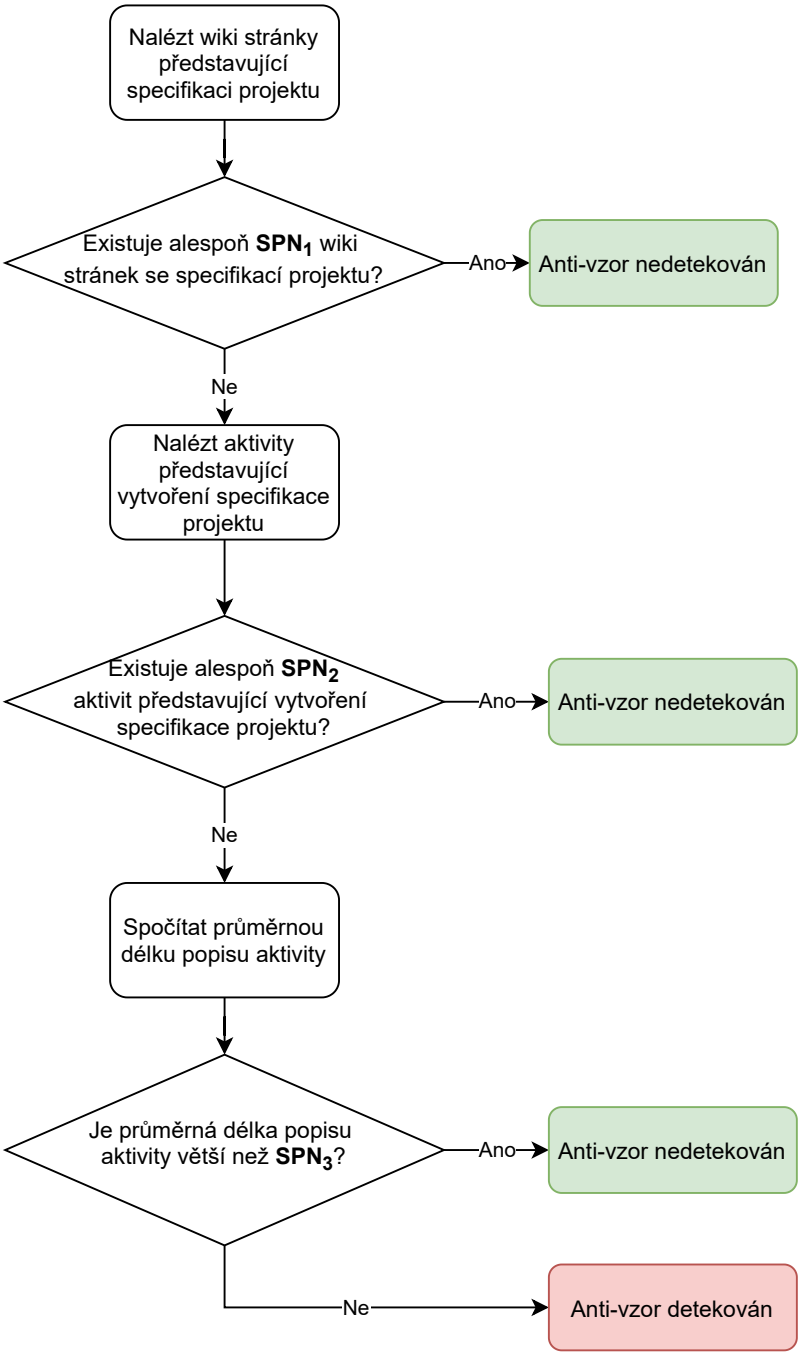
V další části se zaměříme na jednotlivé úkoly projektu. Je tedy možné, že byla specifikace vytvořena, ale není o ní žádná zmínka ve wiki stránkách. Toto je možné zjistit pomocí úkolů. Budeme hledat úkoly, které by mohly představovat vytvoření dokumentu specifikace projektu. Specifikace by měla být vytvořena na začátku projektu, omezíme se tedy pouze na úkoly v první

a druhé iteraci. Pokud je počet nalezených úkolů větší nebo roven prahové hodnotě s označením $\texttt{SPN}_2$, není anti-vzor detekován.

V poslední části se zaměříme na průměrnou délku popisu jednotlivých úkolů. Je možné, že si tým nevede strukturovaný dokument, ale specifikaci si uchovává v podobě popisu jednotlivých (forma blížící se user stories) úkolů. Pokud tedy průměrná délka popisu všech úkolů překročí prahovou hodnotu s označením $\texttt{SPN}_3$, není anti-vzor detekován, v opačném případě bude detekován. Diagram detekce je zobrazen na obrázku číslo 4.5.

Prahové hodnoty

- $\texttt{SPN}_1$ – minimální počet wiki stránek představující specifikaci projektu,
- $\texttt{SPN}_2$ – minimální počet úkolů představující tvorbu specifikace projektu,
- $\texttt{SPN}_3$ – minimální průměrná délka popisku úkolů (počet znaků).



Obrázek 4.5: Stavový diagram návrhu detekce pro anti-vzor Specify Nothing

4.2.6 Too Long Sprint

Popis

Anti-vzor s názvem Too Long Sprint se zabývá příliš dlouhými iteracemi. Dle doporučení metodiky Scrum by měla být délka iterací od dvou do čtyřech

týdnů. Pokud bude tato délka překročena, je anti-vzor detekován [11].

Následky

Příliš dlouhé iterace svádí k odkládání jednotlivých úkolů na pozdější fázi iterace. Následně je možné, že se na konci iterace nahromadí velké množství nedokončených úkolů, které již není možné dokončit. Dalším problémem může být špatná dekompozice jednotlivých úkolů. Pro delší iterace se naplánují velké úkoly, které se v průběhu dlouhých iteracích špatně korigují. Na konci iterace se může stát, že byla vytvořena funkcionalita, která neodpovídá zadání od zákazníka. Posledním zásadním problémem může být pomalá reakce na měnící se priority zákazníka v průběhu projektu [11], které plynou z nedostatečné komunikace se zákazníkem po každé iteraci. V ideálním případě by mělo dojít ke komunikaci se zákazníkem na konci každé iterace, viz sekce číslo 2.3.3.

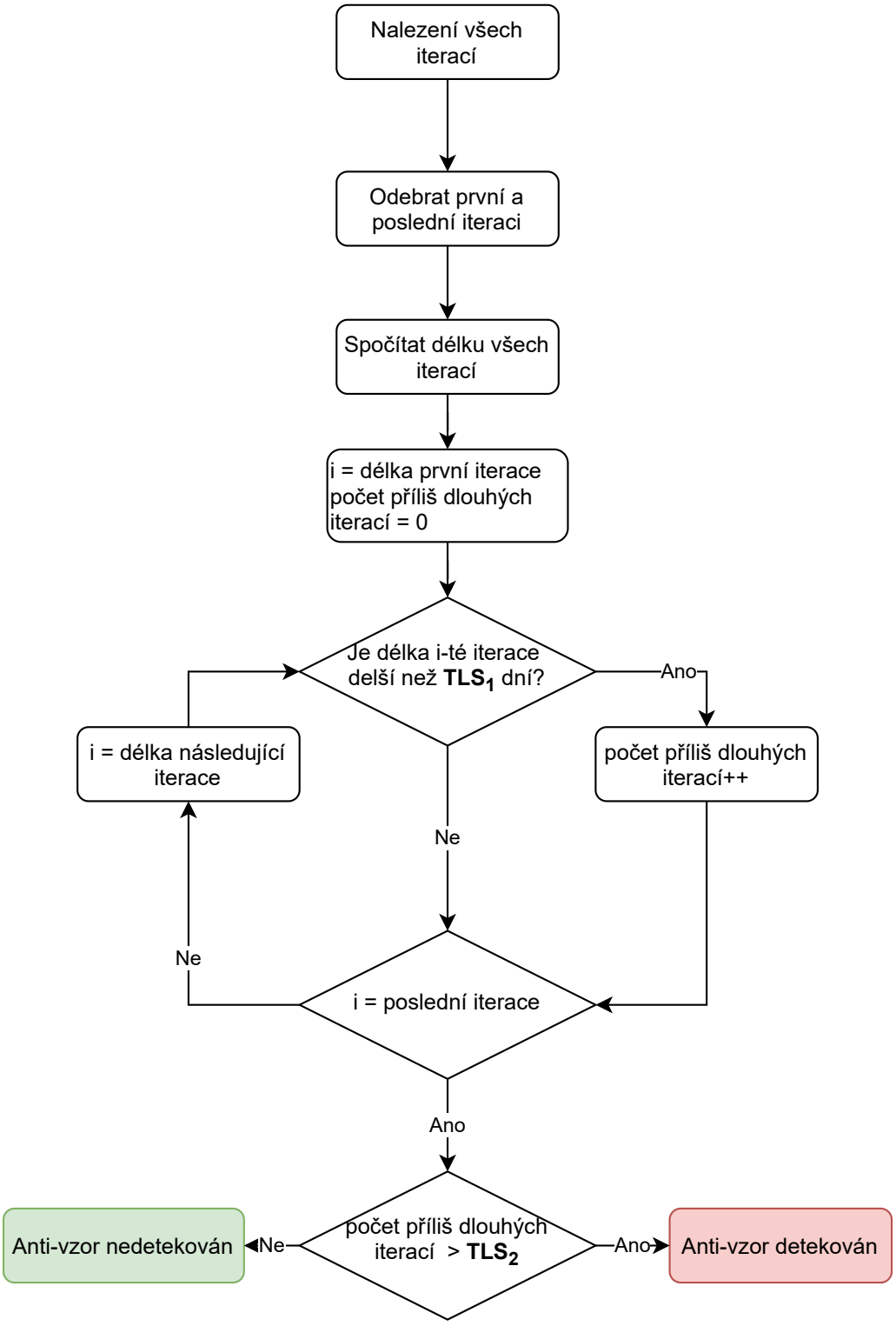
Detekce

Pro detekci tohoto anti-vzoru se bude pracovat s informacemi o jednotlivých iteracích daného projektu. Nejprve je nutné nalézt všechny iterace pro daný projekt. Dále odebereme první a poslední iteraci z důvodů výkyvů na začátku a na konci projektu. K těmto výkyvům může dojít z časových možností vývojového týmu nebo zadavatele. Výkyvy na začátku a na konci projektu jsou v rámci tohoto anti-vzoru přípustné. Na začátku dochází k rozběhnutí infrastruktury celého projektu a vlivem toho může být první iterace zkrácena či prodloužena dle potřeby týmu. Na konci projektu mohla být poslední iterace zkrácena z formálního důvodu nebo prodloužena z důvodu nutných oprav před finální předáním produktu.

Následně spočteme délku všech iterací pomocí rozdílu data konce a začátku iterace. Pokud jedna ze spočtených délek iterací překročí prahovou hodnotu s označením $\texttt{TLS}_1$, je inkrementována proměnná pro počet příliš dlouhých iterací. Pokud počet příliš dlouhých iterací přesáhne prahovou hodnotu $\texttt{TLS}_2$, je anti-vzor detekován. Diagram detekce je zobrazen na obrázku číslo 4.6.

Prahové hodnoty

- $\texttt{TLS}_1$ – maximální délka iterace,
- $\texttt{TLS}_2$ – maximální počet příliš dlouhých iterací.



Obrázek 4.6: Stavový diagram návrhu detekce pro anti-vzor Too Long Sprint

4.2.7 Varying Sprint Length

Popis

Anti-vzor s názvem Varying Sprint Length se zabývá proměnlivou délkou iterace/sprintu. Délka iterace se v průběhu projektu může měnit, například na základě požadavku vývojového týmu z retrospektivy. Například může dojít k příliš krátké iteraci při inicializaci projektu. Po dokončení první iterace tým následně zjistí, že je iterace příliš krátká a bude lepší jí prodloužit. Toto se může zejména vyskytnout v raných fázích projektu. Opačný případ může nastat před ukončením projektu. Tým zjistí, že dle plánu nestíhá dodat výsledný produkt a dojde ke zkrácení délky iterací. Délka iterace by se však neměla měnit příliš často v průběhu projektu (mimo začátek a konec) [11].

Následky

Pokud dochází k příliš častým změnám délky jednotlivých iterací v průběhu projektu, je velice obtížné identifikovat dosažení pokroku. Délka jednotlivých iterací může být také prodlužována z důvodu dosažení cílů dané iterace, to může vést ke zpoždění dodání části nebo celého produktu. Zákazník tedy přesně neví, kdy dostane slíbenou část produktu.

Následek proměnlivé délky iterace také může být odlišná doba konání retrospektivy a plánování iterace. To způsobuje zbytečný čas navíc, který bychom mohli věnovat vývoji samotného produktu [11].

Detekce

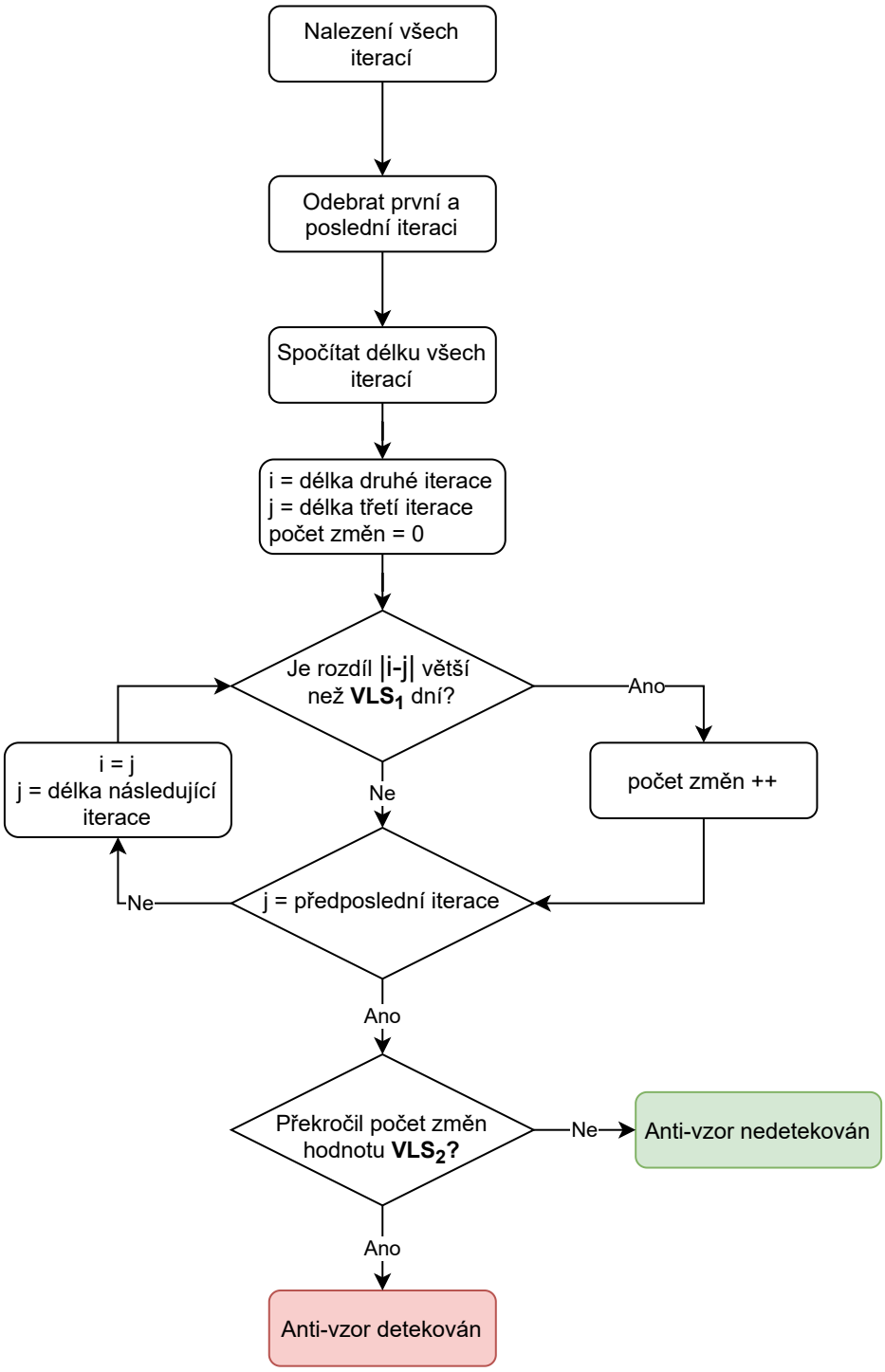
Pro detekci tohoto anti-vzoru se nejprve naleznou všechny iterace pro daný projekt. Následně se odebere první a poslední iterace kvůli výkyvům, které mohou nastat na začátku a na konci projektu.

Pro zbylé iterace se spočte délka pomocí rozdílu dat konce a začátku iterace. Poté se zkontrolují vždy dvě po sobě jdoucí iterace, pokud bude jejich rozdíl délek větší než prahová hodnota s označení $\texttt{VSL}_1$, tak dojde k navýšení proměnné pro počet signifikantních změn délek iterace. Po porovnání všech po sobě jdoucích iteracích dojde ke kontrole proměnné pro počet změn. Pokud překročí počet změn prahovou hodnotu s označením $\texttt{VSL}_2$, je anti-vzor detekován. Diagram detekce je zobrazen na obrázku číslo 4.7.

Prahové hodnoty

- $\texttt{VSL}_1$ – maximální rozdíl délek dvou po sobě jdoucích iteracích (počet dní),

- $\texttt{VSL}_2$ – maximální počet signifikantních změn délek iterace.



Obrázek 4.7: Stavový diagram návrhu detekce pro anti-vzor Varying Sprint Length

4.3 Návrh pomocné aplikace

V rámci detekce jednotlivých anti-vzorů budou získána data z datového skladu SPADe pomocí standardního dotazovacího jazyka Structured Query Language (SQL). V některých případech by bylo obtížné nebo dokonce i nemožné se omezit pouze na dotazování pomocí SQL díky komplexitě operací nutných k detekci některých anti-vzorů. Proto bude detekce anti-vzorů probíhat vždy ve dvou fázích:

1. získání potřebných informací z datového skladu SPADe pomocí SQL,
2. procedurální vyhodnocení získaných informací.

Pro sjednocení rozhraní detekcí všech anti-vzorů a přehledné zobrazení výsledků detekce bude vytvořena pomocná aplikace. Návrh této pomocné aplikace bude popsán v této části.

4.3.1 Klíčové vlastnosti

Čtení z datového skladu SPADe

Pro detekci jednotlivých anti-vzorů bude zapotřebí získat data, která jsou uložena v datovém skladu, viz podkapitola číslo 3.6.2. Data budou získávána za pomocí standardních SQL dotazů. Za tímto účelem bude muset být vytvořená komponenta, která zasílá potřebné dotazy do datového skladu a následně zpracuje výsledky dotazů pro následnou analýzu.

Snadná rozšiřitelnost o další anti-vzory

V rámci této práce bude implementována detekce pro sedm vybraných anti-vzorů, viz podkapitola číslo 4.1. Cílem vytvoření tohoto nástroje je však snadná rozšiřitelnost o další možné anti-vzory v budoucnu. Proto musí být docíleno co nejsnadnější rozšiřitelnosti o další případné anti-vzory.

Načítání SQL dotazů ze souborů

Pro zajištění přehlednosti aplikace bude vhodné SQL dotazy uchovávat v samostatných souborech, které budou součástí výsledné aplikace. V těchto souborech může být uložen jeden čí více SQL dotazů. Při spuštění detekce dojde k načtení jednotlivých SQL dotazů, do kterých se vloží potřebné atributy. Po připravení dotazu pro detekci dojde k odeslání pomocí modulu pro komunikaci s datovým skladem SPADe. Tato funkcionalita částečně zajišťuje snadnou rozšiřitelnost aplikace, která byla popsána v předchozí podkapitole.

Výběr anti-vzorů a projektů pro analýzu

Pomocí uživatelského rozhraní se přehledně zobrazí všechny dostupné projekty, které jsou uložené v datovém skladu SPADe a všechny implementované anti-vzory. Následně umožní vybrat uživateli kombinaci projektů a anti-vzorů, které chce detekovat.

Přehledné zobrazení výsledků

Hlavním účelem je přehledně zobrazit výsledky detekce implementovaných anti-vzorů u jednotlivých projektů do tabulky. Dále bude vhodné u každé detekce zobrazit podrobnější informace, na základě kterých byl anti-vzor detekován či nedetekován.

Nastavení prahových hodnot

U každého anti-vzoru lze konfigurovat prahové hodnoty, které udávají práh detekce daného anti-vzoru. Nastavení těchto prahových hodnot se může měnit v závislosti na charakteru projektu, aplikace by tedy měla umožňovat jednoduše měnit nastavení těchto hodnot pomocí uživatelského rozhraní nebo pomocí konfiguračního souboru.

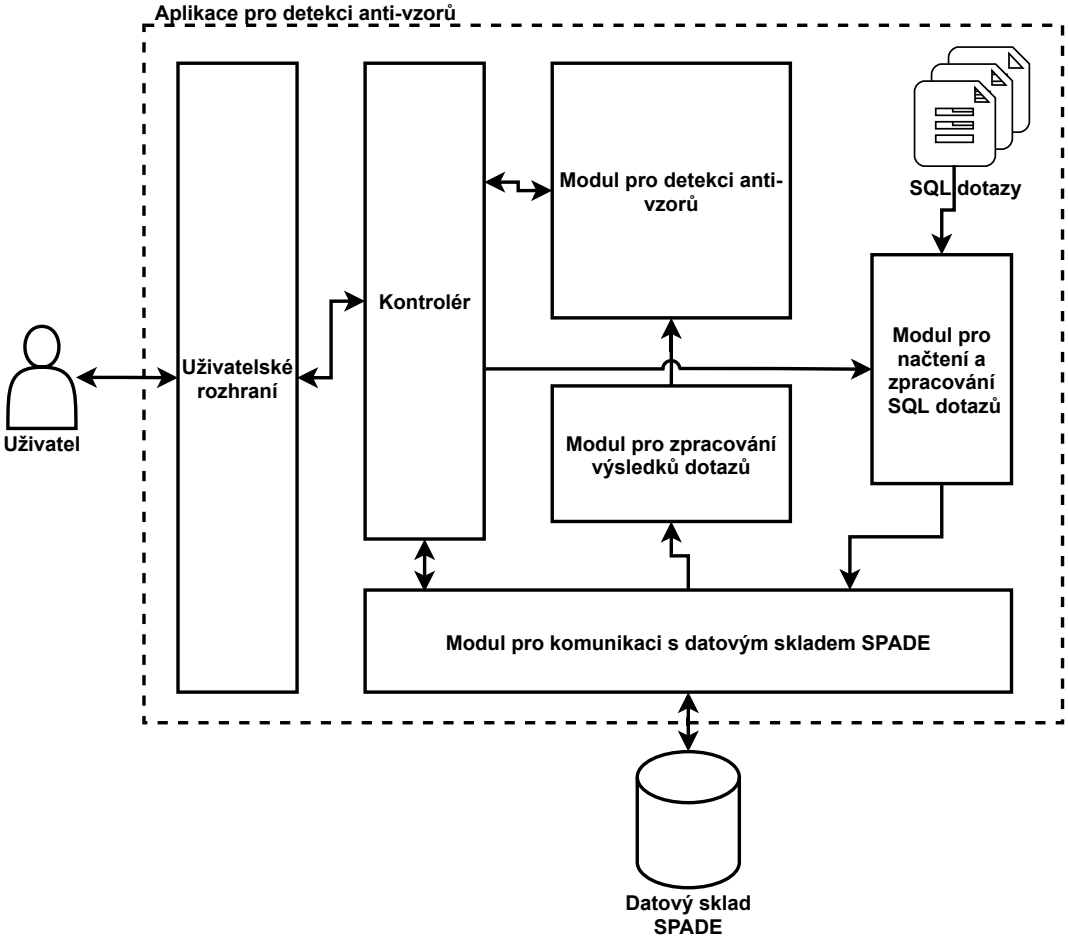
4.3.2 Struktura aplikace

Struktura samotné aplikace byla rozdělena do šesti hlavních částí, které se starají o jednotlivé funkce aplikace.

- První část je uživatelské rozhraní, pomocí kterého bude umožněno vybírat jednotlivé projekty a anti-vzory. Následně bude zobrazovat výsledky detekce uživateli.
- Druhá část aplikace je tzv. kontrolér, který zpracovává požadavky jednotlivých uživatelů a komunikuje s ostatními moduly. Připravuje data, která budou zobrazena v uživatelském rozhraní.
- Další částí je modul pro načítání a přípravu SQL dotazů. Tento modul zajišťuje načítání potřebných SQL dotazů z příslušných souborů pro detekci. Po načtení vloží do dotazů potřebné atributy jako například identifikační číslo analyzovaného projektu apod.
- Po načtení SQL dotazů ze souboru budou dotazy zpracovány a odeslány do datového skladu pomocí modulu pro komunikaci s datovým skladem SPADe.

- Další část aplikace je modul pro zpracování výsledků jednotlivých dotazů. Tento modul bude transformovat příchozí data z datového skladu do přívětivějšího formátu, který bude využit dále v aplikaci.
- Poslední částí aplikace bude modul pro detekci anti-vzorů, který analyzuje výsledná data přijatá od modulu pro zpracování výsledku dotazů. Na základě analýzy získaných dat bude modul rozhodovat o výskytu příslušného anti-vzoru v projektových datech.

Výsledná struktura pomocné aplikace pro detekci anti-vzorů je zobrazena na obrázku číslo 4.8.



Obrázek 4.8: Návrh struktury aplikace

5 Implementace

V této kapitole bude popsána implementace detekce jednotlivých anti-vzorů na základě analýzy v kapitole číslo 4. Dále zde bude stručně popsána implementace pomocné aplikace pro detekci anti-vzorů.

5.1 Implementace detekce anti-vzorů

Jak již bylo zmíněno v sekci číslo 4.3, detekce anti-vzorů bude probíhat ve dvou krocích:

1. získání potřebných informací z datového skladu SPADe pomocí SQL,
2. procedurální vyhodnocení získaných informací pomocí jazyka Java.

V této části bude popsáno, jaké konkrétní pohledy nebo tabulky z datového skladu (popsány v sekci číslo 3.6.2) jsou k detekci využívány a k jakému následnému zpracování informací dochází.

Business As Usual

Pro detekci tohoto anti-vzoru je nejprve nutné spočíst počet iterací daného projektu. Počet iterací lze zjistit z tabulky s názvem `iteration`. Jednotlivé iterace jsou vybírány pomocí atributu `superProjectId`, který odpovídá identifikátoru daného projektu.

V dalším kroku se použije pohled `workUnitView`, kde jsou vybrány pouze ty úkoly, které mají v názvu podřetězec připomínající retrospektivu, například „retr“, „revi“ nebo „week scrum“. Následně jednotlivé nalezené úkoly seskupíme pomocí atributu `iterationName`. Tím dostaneme seznam všech iterací, ve kterých byly nalezeny úkoly připomínající retrospektivu a jejich počet v každé iteraci.

Dalším krokem je pokusit se nalézt jednotlivé wiki stránky, které by mohly obsahovat záznamy z retrospektiv. K tomu bude zapotřebí pohled `artifactView`, tabulka `iteration` a pro detekci změn v jednotlivých wiki stránkách bude zapotřebí pohled `fieldChangeView`, který je propojen s pohledem pomocí cizího klíče `itemId`. Artefakty týkající se pouze wiki stránek lze vyfiltrovat pomocí atributu `artifactClass`, který se rovná hodnotě „WIKIPAGE“. Všechny wiki stránky, které by se mohly týkat retrospektiv, jsou detekovány stejným způsobem jako úkoly, s rozdílem, že dochází

i ke kontrole obsahu wiki stránky. Tím dostaneme seznam všech změn ve wiki stránkách, které připomínají retrospektivu. Následně pomocí atributu `fieldChangeView.created` přiřadíme změny k jednotlivým iteracím. Opět tedy získáme seznam všech iterací, ve kterých byla nalezena změna wiki stránky připomínající retrospektivu a počet těchto nálezů v každé iteraci.

Následně jsou oba výsledné seznamy sloučeny do jednoho. V ideálním případě by každá iterace měla obsahovat alespoň jeden záznam retrospektivy. Počet iterací ve výsledném seznamu je následně porovnán s prahovou hodnotou, která udává minimální počet iterací s retrospektivou. Pokud je počet nalezených iterací s retrospektivou menší než prahová hodnota, je anti-vzor detekován. V opačném případě není detekován.

Long Or Non-Existant Feedback Loops

V prvním kroku detekce dojde k hledání všech iterací, ve kterých proběhla schůzka se zákazníkem. Nejprve jsou nalezeny všechny úkoly pomocí pohledu `workUnitView`, které by mohly představovat schůzku se zákazníkem. Detekce úkolů probíhá pomocí hledání různých podřetězců v názvu, například „zákazn“, „zadavatel“ apod. Následně jsou úkoly seskupeny pomocí atributu `workUnitView.iterationName`. Pokud byl nalezen úkol představující schůzku se zákazníkem ve všech iteracích, není anti-vzor detekován.

Pokud nebyla nalezena schůzka se zákazníkem ve všech iteracích, ale alespoň v $\texttt{LON}_1$ (minimální počet iterací, kde se konala alespoň jedna schůzka se zákazníkem) iteracích, dochází ke kontrole rozestupů těchto úkolů/schůzek. U první a poslední nalezené schůzky navíc probíhá kontrola rozestupu od začátku a konce projektu. Začátek a konec projektu je zjištěn z tabulky `iteration`. Začátek projektu je detekován pomocí atributu `startDate` u první iterace a konec projektu je detekován pomocí atributu `endDate` u poslední iterace. Pokud překročí rozestup schůzek se zákazníkem prahovou hodnotu $\texttt{LON}_2$ (maximální rozestup schůzek se zákazníkem), tak je anti-vzor detekován.

Pokud není nalezen potřebný počet úkolů, je možné, že tým zaznamenává schůzky pouze do wiki stránek. Za pomocí pohledu `artifactView` jsou nalezeny všechny úkoly, které by mohly připomínat schůzku se zákazníkem. V případě, že má tým vytvořenou pouze jednu wiki stránku pro všechny schůzky, je navíc využito pohledu `fieldChagneView`, pomocí kterého lze zjistit změny v nalezených wiki stránkách v průběhu projektu. Akceptovatelné změny jsou pouze ty, kde je délka nového obsahu wiki stránky delší než starého obsahu (`length(newValue) > length(oldValue)`).

Pokud jsou nalezené wiki stránky ve všech iteracích, není anti-vzor dete-

kován. V opačném případě dochází ke kontrole rozestupů úprav jednotlivých wiki stránek. Kontrola rozestupu opět probíhá i s porovnáním vůči začátku a konci projektu. V případě, že některý rozestup překročí prahovou hodnotu $\texttt{LON}_2$ (maximální rozestup schůzek se zákazníkem), je anti-vzor detekován.

Ninety-Ninety Rule

Při implementaci byl nalezen problém s detekcí implementačních úkolů. Prvotní cíl, jak detekovat implementační úkoly, byl vybrat všechny úkoly, které mají commit do repozitáře. Bohužel bylo zjištěno, že v dostupné verzi datového skladu SPADe tato vazba chybí. Druhou možností, jak detekovat tyto úkoly, by bylo pomocí názvu. Po další analýze bylo od této varianty upuštěno a dále se bude pracovat se všemi úkoly.

Pomocí pohledu `workUnitView` vybereme všechny úkoly pro daný projekt. Následně pomocí klauzule `GROUP BY` rozdělíme úkoly do jednotlivých iterací pomocí atributu s názvem `workUnitView.iteration`. V další fázi spočteme podíl stráveného a odhadovaného času pomocí atributů s názvem `workUnitView.spentTime` a `workUnitView.estimatedTime`.

Dle definice tohoto anti-vzoru se budeme zaměřovat pouze na iterace, kde byl strávený čas větší než odhadovaný. Maximální hodnotu podílu stráveného a odhadovaného času definuje prahová hodnota. Budeme procházet výsledné podíly a porovnávat je s prahovou hodnotou. Pokud bude překročena prahová hodnota udávající maximální počet po sobě jdoucích iterací se zhoršujícími se odhady, je anti-vzor detekován.

Road To Nowhere

Pro detekci tohoto anti-vzoru je nejprve nutné u každého projektu nalézt první dvě iterace. Toho docílíme pomocí tabulky `iteration`, kde jednotlivé iterace seřadíme pomocí atributu `iteration.startDate` a poté vybereme operátorem `LIMIT` a `OFFSET` první dvě iterace.

Dalším krokem je nalezení všech wiki stránek představující projektový plán, který byl vytvořen v první nebo druhé iteraci. S využitím pohledu `artifactView` nalezneme všechny wiki stránky pomocí atributu s názvem `artifactClass` = „WIKIPAGE“. Následně dojde k hledání podřetězce v názvu nebo obsahu wiki stránky představující projektový plán.

V dalším kroku je nutné nalézt všechny úkoly, které byly dokončeny v první nebo druhé iteraci a mohly by představovat vytvoření projektového plánu. Pomocí pohledu `workUnitView` nalezneme všechny úkoly v první a druhé iteraci. Pohled `workUnitView` neobsahuje id iterace, kontrola, zda úkol spadá do první nebo druhé iterace, je tedy prováděna za pomocí atributu

`workUnitView.iterationStartDate`. Dále je zapotřebí detekovat úkoly týkající se projektového plánu. Tyto úkoly jsou detekovány pomocí hledání podřetězce v názvu nebo popisku.

V poslední fázi dochází nejprve k porovnání počtu nalezených wiki stránek s prahovou hodnotou, která udává minimální počet těchto stránek. Pokud není nalezena žádná stránka, je porovnán počet nalezených úkolů s prahovou hodnotu, která udává minimální počet těchto úkolů. Pokud není ani v jednom případě detekován potřebný počet, je anti-vzor detekován. V opačném případě není detekován.

Specify Nothing

Detekci anti-vzoru Specify Nothing lze rozdělit do čtyřech hlavních částí. V první části je nutné zjistit, zda se v projektových datech vyskytuje samotný dokument specifikace projektu. Toho můžeme docílit pomocí pohledu `artifactView`, pomocí kterého vyhledáme všechny wiki stránky potenciálně představující specifikaci projektu. Wiki stránky jsou detekovány pomocí podřetězce v názvu například „DSP“ nebo „specification“.

Jak již bylo zmíněno v analýze, samotný dokument specifikace projektu může být veden v některém externím nástroji pro správu dokumentů. Proto je nutné se dále zaměřit na pohled `workUnitView`, který představuje úkoly projektu. Je nutné detekovat úkoly, které by mohly představovat vytvoření specifikačního dokumentu.

Dále je nutné se zaměřit na průměrnou délku popisku jednotlivých úkolů. Popisky úkolů jsou uloženy v atributu `workUnitView.description` a pomocí standardní SQL funkce `AVG()` je spočtena průměrná délka popisku.

Po získání všech potřebných informací z datového skladu jsou jednotlivé počty nalezených wiki stránek, úkolů a průměrné délky textu popisku úkolů porovnány s prahovými hodnotami. V případě, že jsou všechny nalezené položky pod prahovými hodnotami, je anti-vzor detekován.

Too Long Sprint

Pro detekci tohoto anti-vzoru je zapotřebí pouze jedné tabulky s názvem `iteration`. Nejprve dojde k detekci první a poslední iterace, které v detekci nebudou figurovat z důvodu možných výkyvů na začátku a konci projektu.

Následně dojde ke spočtení délek všech zbylých iterací pomocí atributů `iteration.startDate` a `iteration.endDate` s využitím standardní SQL funkce `DATEDIFF()`. Tím získáme délku jednotlivých iterací ve dnech.

Dále dochází ke kontrole každé délky iterace, a pokud překročí délka jedné iterace prahovou hodnotu (prahová hodnota udává maximální délku

iterace), je navýšena proměnná `numberOfLongIterations`. Pokud překročí proměnná `numberOfLongIterations` prahovou hodnotu udávající maximální počet příliš dlouhých iterací, je anti-vzor detekován.

Varying Sprint Length

K detekci tohoto anti-vzoru bude zapotřebí získat informace o délkách jednotlivých iteracích. Proto bude první fáze implementace detekce tohoto anti-vzoru totožná s detekcí anti-vzoru Too Long Sprint.

Po získání délek všech iterací (kromě první a poslední) dochází k porovnání vždy dvou po sobě jdoucích iteracích. Pokud rozdíl jejich délek překročí prahovou hodnotu (maximální změna délky iterace), je navýšena proměnná `iterationLengthChanged` o jedničku. Pokud po porovnání všech párů po sobě jdoucích iterací bude proměnná `iterationLengthChanged` větší než prahová hodnota udávající maximální počet změn délek iterací, je anti-vzor detekován. V opačném případě není detekován.

5.2 Implementace pomocné aplikace

V této části bude popsána implementace pomocné aplikace pro detekci anti-vzorů, která byla navrhnuta v předchozí kapitole.

5.2.1 Zvolené technologie

Aplikace pro detekci anti-vzorů je implementována jako jednoduchá webová aplikace. Důvody pro vytvoření webové aplikace byly následovné:

- trend dnešní doby,
- snadný přístup k aplikaci bez nutnosti instalace na každé zařízení,
- snadná implementace REST API v případě potřeby propojení aplikace s ostatními webovými službami.

Jelikož se jedná o velmi malou aplikaci, je implementována monolitickou architekturou (uživatelské rozhraní není samostatně stojící služba).

Pro implementaci bylo využito populárního open-source frameworku s názvem Spring. Jedná se o aplikační rámec neboli framework pro vývoj Java Enterprise Edition (JEE) aplikací. Uživatelské rozhraní je implementováno pomocí jednoduchého šablonovacího systému s názvem Thymeleaf.

Pro komunikaci s datovým skladem, který je uložen v MySQL databázi bylo využito standardního Java Database Connectivity (JDBC) konektoru pro komunikaci s MySQL databázemi.

5.2.2 Struktura projektu

Struktura a popis jednotlivých složek či souborů aplikace jsou zobrazeny v tabulce číslo 5.1. Znak * nahrazuje cestu `src/main` a znak `#` nahrazuje složku `resources` z důvodu úspory místa v tabulce.

Tabulka 5.1: Struktura projektu aplikace

Složka/Soubor	*Obsah*
`*/java`	zdrojové kódy aplikace
`*/#/application.properties`	nastavení aplikace
`*/webapp/queries`	definice dotazů pro anti-vzory
`*/webapp/WEB-INF/templates`	html šablony uživatelského rozhraní
`pom.xml`	soubor pro sestavení aplikace
`db_dump.sql`	soubor pro obnovu datového skladu
`docker-compose.yml`	soubor pro spuštění aplikace v Dockeru

5.2.3 Balíky tříd

Jednotlivé třídy jsou děleny do balíků. Balíky tříd jsou umístěny ve složce `cz.zcu.fav.kiv.antipatterndetectionapp`. V této složce je také umístěna hlavní třída s názvem `AntiPatternDetectionAppApplication`, která zajišťuje spuštění celé aplikace. Třídy jsou do jednotlivých balíků rozděleny podle funkce dané třídy. Celkem je projekt tvořen sedmi balíky, které jsou níže popsány.

- `controller` – Balík obsahuje pouze jednu třídu, která představuje kontrolér aplikace. Tento kontrolér zpracovává požadavky jednotlivých uživatelů a následně zobrazuje data do uživatelského rozhraní.
- `detecting` – Balík obsahující všechny potřebné třídy pro detekci jednotlivých anti-vzorů a připojení k datovému skladu SPADe.
- `model` – Balík obsahující modelové třídy nebo také přepravky pro snadné předávání dat v rámci jednotlivých komponent aplikace.
- `repository` – Balík obsahující jednotlivé komponenty, které se starají o načítání a ukládání načtených SQL dotazů ze souborů a inicializaci jednotlivých detektorů anti-vzorů.
- `service` – Balík obsahující jednotlivé služby pro práci s instancemi jednotlivých projektů a anti-vzorů.

- `spring` – Třídy obsahující konfiguraci frameworku Spring pro korektní fungování webové aplikace. Například ve třídě `AppConfig` je definováno, kde jsou uloženy šablony pro uživatelské rozhraní.
- `utils` – Balík obsahující pouze jednu třídu `Utils`, která obsahuje pomocné statické metody používané na různých místech v aplikaci.

5.2.4 Rozšíření o další anti-vzory

Pro zajištění co nejjednoduššího rozšíření o další detekce anti-vzorů jsou pomocí reflexe automaticky načítány všechny třídy, které implementují rozhraní s názvem `AntiPatternDetector`. Poté je volána přímo metoda s názvem `analyze`, která analyzuje vybraný anti-vzor.

Pro rozšíření aplikace o další anti-vzor je nutné provést následující dva kroky:

1. implementovat rozhraní `AntiPatternDetector` a všechny jeho metody,
2. vytvořit soubor s jedním nebo více SQL dotazy ve složce `/src/main/webapp/queries`.

Rozhraní obsahuje čtyři metody, které je nutné implementovat. Funkce jednotlivých metod je popsána v následujícím seznamu:

- `getAntiPatternModel` – metoda vracející instanci modelové třídy anti-vzoru pro zobrazení v uživatelském rozhraní (v této modelové třídě lze také definovat prahové hodnoty daného anti-vzoru),
- `getAntiPatternSqlFileName` – metoda, která vrací název souboru s SQL dotazy pro daný anti-vzor,
- `setSqlQueries` – metoda, která nastavuje SQL dotazy načtené ze souboru k tomuto anti-vzoru (k načítání dochází pouze jednou, a to pří startu aplikace),
- `analyze` – metoda, která pracuje s výsledky SQL dotazů a provádí konečnou detekci anti-vzoru.

Pro správné fungování detekce je nutné zachovat jednotnou formu souborů s SQL dotazy. Každý dotaz musí být na samostatné řádce a musí být ukončen středníkem.

5.2.5 Testování

Ověření správné konstrukce jednotlivých SQL dotazů bylo prováděno na bázi manuálního ověření předpokládaných výsledků. Před sestrojením potřebného dotazu byla vytvořena předpokládaná sada výsledků, která by měla odpovídat výsledku vytvořeného dotazu. Po implementaci potřebného SQL dotazu byl dotaz spuštěn nad sadou vybraných dat. Výsledek dotazu byl následně porovnán na základě předpokládaných výstupů, a tím byla určena jeho věrohodnost.

Pomocná aplikace pro detekci anti-vzorů byla vytvořena nad rámec a nebyla cílem této práce. Z tohoto důvodu nebyla podrobena důslednému testování. Funkce, ovládání a vzhled výsledné aplikace je uveden v příloze C a v příloze B je uveden postup pro její nasazení.

6 Experiment

V této kapitole je proveden experiment detekce implementovaných anti-vzorů na vybrané sadě projektových dat. Výsledky jsou následně diskutovány a jsou zde uvedena možná rozšíření jednotlivých detekcí.

6.1 Postup experimentu

Experiment bude rozdělen do následujících částí:

1. výběr vhodné sady projektů,
2. nastavení prahových hodnot pro jednotlivé anti-vzory,
3. provedení detekce,
4. vyhodnocení výsledků,
5. diskuze výsledků.

6.2 Výběr vhodné sady projektů

Výběr vhodných projektů je pro úspěch detekce anti-vzorů velice důležitý. Musí být k dispozici dostatečné množství informací o každém projektu, o jednotlivých úkolech, iteracích ale také o různých artefaktech. V případě, že by projektová data obsahovala malé množství těchto informací, může dojít ke špatné detekci jednotlivých anti-vzorů. K dispozici se nabízejí tři varianty zdrojů projektů, respektive projektových dat:

1. komerční projekty,
2. open source projekty,
3. studentské projekty KIV na ZČU.

První zmíněnou skupinou jsou komerční projekty. Jedná se o softwarové projekty tvořené softwarovými společnostmi, které produkují buď vlastní produkty, nebo produkty na základě zákaznické poptávky. Každá softwarová společnost používá některé metodiky pro vývoj softwaru, které si přizpůsobuje svým vlastním potřebám. Tím si v podstatě tvoří svoje vlastní

know-how pro vývoj softwaru. Tímto know-how si vytváří konkurenční výhodu oproti ostatním společnostem, které jsou na trhu. Proto je velice složité tato data od komerčních společností získat a pro účely této práce nebudou využita.

Další skupinou jsou tzv. open source projekty. Jedná se o projekty s veřejně přístupným zdrojový kódem a každý vývojář může buď přispět k vývoji tohoto produktu, nebo si může produktu upravit pro své vlastní potřeby. Tento typ projektů je sice veřejně dostupný, ale ve většině případů je dostupný pouze zdrojový kód aplikace. V lepším případě je dostupný přímo repozitář daného projektu, takže můžeme vidět jednotlivé změny v průběhu času tzv. commity. Z těchto informací by bylo možné detekovat jen velice malou omezenou část anti-vzorů, a proto nebudou pro detekci v této práci vybrány. K potřebám této práce bychom potřebovali znát více informací z dalších ALM nástrojů, jako například z nástrojů pro správu změn a problémů popsaných v kapitole číslo 3.3.

Poslední skupinou jsou interní projekty, které jsou vytvářeny na KIV FAV ZČU. Konkrétně se jedná o projekty, které jsou vytvářeny v rámci předmětu Pokročilé softwarové inženýrství (ASWI). V rámci těchto projektů se využívá metodika přizpůsobená pro tyto krátké projekty, ale je založená na metodice RUP se značným vlivem Scrumu a jiných agilních metodik, která byla představena v sekci číslo 2.3.2 a 2.3.3. Modifikovaná metodika pro projekty v rámci předmětu ASWI je představena v další sekci číslo 6.2.2. Informace o jednotlivých úkolech, iteracích a artefaktech se zaznamenávají do nástroje Redmine, která je zdrojem největšího množství informací pro detekci anti-vzorů. Tyto projekty jsou nejvhodnějším kandidátem z důvodu velkého množství informací, které jsou o nich uchovávány.

6.2.1 ASWI projekty

Jak již bylo zmíněno, pro detekci anti-vzorů budou využita data z projektů vytvářených v rámci předmětu ASWI. V rámci této práce se budou analyzovat projektová data za poslední dva roky (2019 a 2020) z důvodu konzistence dat. Všechna tato projektová data jsou uložena v nástroji SPADe, který byl popsán v kapitole 3.6. Výsledná sada projektů z předmětu ASWI se základními informacemi je zobrazena v tabulce číslo 6.1.

Tabulka 6.1: Výsledná sada projektů

Id	Počet členů týmu	Počet iterací	Začátek	Konec
1	4	5	13. 3. 2019	14. 6. 2019
2	4	7	5. 4. 2019	9. 6. 2019
3	4	7	18. 3. 2019	3. 6. 2019
4	4	7	19. 3. 2019	14. 6. 2019
5	4	6	3. 3. 2020	18. 5. 2020
6	4	6	21. 3. 2020	29. 5. 2020
7	2	4	12. 3. 2020	11. 5. 2020
8	5	6	5. 3. 2020	5. 6. 2020
9	4	6	19. 3. 2020	7. 8. 2020

6.2.2 ASWI proces

Softwarový proces pro studentské projekty ASWI je iterativní, agilně orientovaný proces pro řízení tvorby malých až středně velkých softwarových systémů. Je založený na kombinaci některých praktik z metodik Scrum a RUP (viz podkapitola číslo 2.6). Délka projektu je odvozena od délky jednoho semestru, projekt tedy většinou trvá od dvou do třech měsíců.

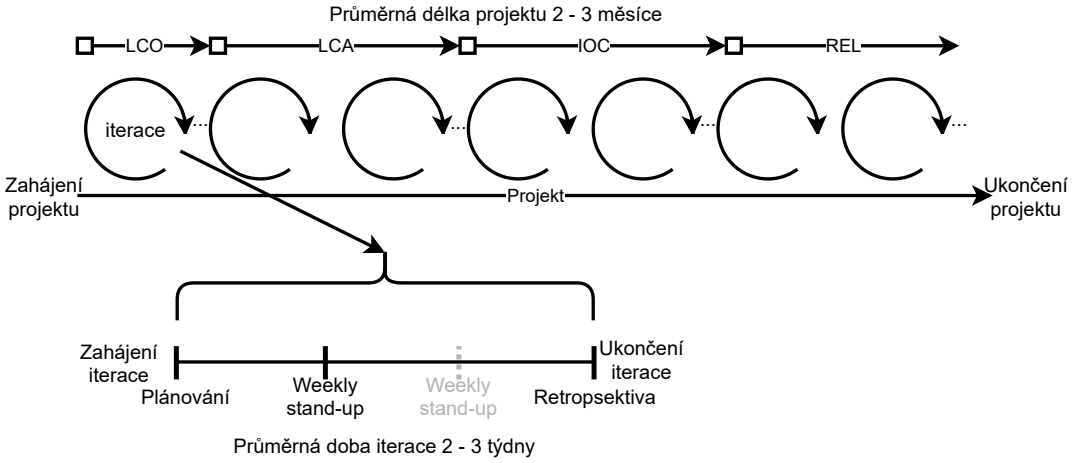
Celý proces vývoje probíhá v jednotlivých iteracích. Délku jednotlivých iterací si tým zvolí na základě jejich zkušeností či preferencí. Standardně, jako v metodice Scrum, by mělo probíhat na začátku každé iterace plánování a na konci retrospektiva, kde by měli být přítomni všichni členové týmu. Výjimkou, oproti metodice Scrum, je schůzka Daily Scrum, která se nekoná každý den ale pouze jednou za týden.

Celý projekt se řídí čtyřmi základními fázemi, které jsou definované v metodice RUP (kapitola číslo 2.3.2). V rámci každé fáze by měl tým dokončit odpovídající milník. Jednotlivé milníky jsou definovány následovně:

- Lifecycle Objectives (LCO) — ukončuje fázi zahájení projektu a stanovení jeho vize,
- Lifecycle Architecture (LCA) — ukončuje fázi určení návrhu architektury řešení,
- Initial Operational Capability (IOC) — ukončuje hlavní realizační práce,
- Product Release (REL) — završuje předání produktu do rutinního provozu a celý semestrální projekt.

Celý projektový tým se skládá z interních a externích rolí. Mezi externí role patří mentor a zákazník. Mentor pomáhá s nejasnostmi ohledně vývo-

jového procesu a práce s podpůrnými nástroji, provádí pedagogický dohled nad týmem. Role mentora lze částečně připodobnit roli Scrum Mastera, která je definovaná v metodice Scrum (sekce 2.3.3). Zákazník se podílí na tvorbě vize produktu, definuje jeho požadavky a hodnotí jejich splnění. Mezi interní role patří analytik, architekt, systémový inženýr, tester, vedoucí a vývojář. Rozdělení interních rolí mezi jednotlivé osoby v projektovém týmu je v pravomoci samotných členů týmu. Velikost projektového týmu (bez externích rolích) je většinou okolo čtyř až pěti členů. Průběh popisovaného projektu je zobrazen na obrázku číslo 6.1. Kompletní popis ASWI procesu je uveden v dokumentu [7].



Obrázek 6.1: Schéma průběhu ASWI procesu

6.3 Nastavení prahových hodnot

Některé prahové hodnoty lze nastavit na konkrétní hodnotu, jelikož nejsou závislé na vlastnostech jednotlivých projektů. Avšak většina prahových hodnot je určena výrazem, jehož výsledek určí prahovou hodnotu pro každý projekt individuálně na základě jeho pozorovaných vlastností. Nastavení prahových hodnot bude probíhat na základě charakteru ASWI procesu popsaného v sekci 6.2.2. Konkrétní konfigurace prahových hodnot slouží k ověření funkčnosti detekce v porovnání s reálnými daty. Konkrétní hodnoty, které jsou zde použity, jsou přímo závislé na použitém procesu. Jednotlivé prahové hodnoty lze ručně konfigurovat pomocí výsledné aplikace, viz sekce číslo 4.3.1 a podkapitola přílohy C.2.

Business As Usual

- BAU – minimální počet iterací s retrospektivou = *2/3 * počet iterací*

Každý projekt může mít jiný počet iterací, je nutné, aby byl minimální počet iterací s retrospektivou závislý na celkovém počtu. Prahová hodnota udávající minimální počet iterací s retrospektivou je vypočtena pomocí vztahu výše. Tato prahová hodnota vyžaduje retrospektivu alespoň u 66,6 % iterací.

Long Or Non-Existant Feedback Loops

- $\texttt{LON}_1$ – minimální počet iterací, kde se konala alespoň jedna schůzka se zákazníkem = *1/2 * počet iterací*

- $\texttt{LON}_2$ – maximální rozestup schůzek se zákazníkem (počet dní) = *2 * průměrná délka iterace*

V ideálním případě by měla schůzka se zákazníkem probíhat každou iteraci. V některých případech však může dojít k odložení schůzky z důvodu malého posunu v implementaci projektu nebo z důvodu nedostatečného času zákazníka pro schůzku, proto byla zvolena maximální hodnota rozestupu jako dvojnásobek průměrné délky iterace projektu.

Ninety-Ninety Rule

- $\texttt{NNR}_1$ – maximální hodnota podílu stráveného a odhadovaného času v jednotlivých iteracích = *1.25*

Hodnota 1,25 představuje v podstatě odchylku o 25 % od plánovaného času, což lze považovat jako signifikantní odchylku. Tato hodnota představuje v ASWI procesu průměrně deset hodin, a to odpovídá u standardního případu času jedné osoby na jednu dvoutýdenní iteraci.

- $\texttt{NNR}_2$ – maximální počet po sobě jdoucích iterací, kde podíl stráveného a odhadovaného času překročí prahovou hodnotu $\texttt{NNR}_1$ = *2 iterace*

V rámci průběhu projektu může dojít ke špatným odhadům jednotlivých úkolů, ale tým by se z nich měl ponaučit a v následující iteraci tyto odhady zlepšit. Proto zde byl nastaven počet dvou iterací, kde má tým ve dvou iteracích na špatné odhady zareagovat. V případě, že budou špatné odhady stále pokračovat, je anti-vzor detekován.

Road To Nowhere

- $\texttt{RTN}_1$ – minimální počet wiki stránek obsahující projektový plán = *1*

Většina dostupných projektů vede projektový plán ve wiki stránkách pouze na jedné stránce, je tedy postačující nalézt pouze jednu wiki stránku s projektovým plánem.

- $\texttt{RTN}_2$ – minimální počet úkolů obsahující projektový plán = *1*

Projektový plán může vzniknout na základě jednoho úkolu, je tedy postačující nalézt pouze jeden úkol s projektovým plánem.

Specify Nothing

- $\texttt{SPN}_1$ – minimální počet wiki stránek představující specifikaci daného projektu = *1*

V případě vedení specifikace projektu ve wiki stránkách je vedena pouze na jedné stránce. Proto je postačující nalézt jednu wiki stránku představující specifikaci.

- $\texttt{SPN}_2$ – minimální počet úkolů představující tvorbu specifikace daného projektu = *1*

Specifikaci projektu většinou vytváří samotný projektový tým a následně ji konzultuje se zadavatelem. Je tedy možné, že bude specifikace vytvářena-/upravována vícekrát. Může však dojít k tomu, že je specifikace vytvořena podle představ zákazníka ihned na poprvé. Proto je postačující nalézt alespoň jeden úkol, který zmiňuje vytvoření specifikace projektu.

- $\texttt{SPN}_3$ – minimální průměrná délka popisku úkolu (počet znaků) = *150*

Aby představoval popisek úkolu nějakou vypovídající hodnotu o daném úkolu, musí mít alespoň délku 150 znaků. Tato hodnota byla zvolena na základě analýzy délek popisků jednotlivých úkolů, které jsou uloženy ve SPADe.

Too Long Sprint

- $\texttt{TLS}_1$ – maximální délka iterace = *21 dní*.

V agilní komunitě neexistuje shoda ohledně ideální délky iterace. Metoda Scrum navrhuje délku iterace 3 - 4 týdny, zatímco Extrémní programování navrhuje délku iterace na 1 - 2 týdny [1]. Délka iterace je závislá nejen na

daném procesu vývoje softwaru, ale také na velikosti, délce nebo složitosti projektu. Na základě projektů v rámci ASWI procesu, který trvá v průměru 2 - 3 měsíce zvolíme maximální délku iterace na 3 týdny (21 dní). Pokud by byly v rámci těchto projektů nastaveny délky iterací na 4 týdny, došlo by v některých případech pouze ke dvěma iteracím, což je velice málo.

- $\texttt{TLS}_2$ – maximální počet příliš dlouhých iterací = *0*.

Dle charakteru anti-vzoru Too Long Sprint byla zvolena hodnota této prahové hodnoty na nulu. Jakmile bude jedna iterace (vyjma první a poslední) delší než prahová hodnota, je anti-vzor detekován.

Varying Sprint Length

- $\texttt{VSL}_1$ – maximální rozdíl délek dvou po sobě jdoucích iterací = *7 dní*

Pro detekci rozdílných délek iterací je nutné počítat například s výkyvy v řádech jednotek dnů (státní svátek apod.), které mohou ovlivnit délku iterace. Proto byl zvolen jako signifikantní změna délky iterace jeden týden.

- $\texttt{VSL}_2$ – maximální počet signifikantních změn délek iterace = *1*

V rámci projektu je přirozené, občas i žádané, měnit délku iterace, jelikož prvotní délka iterace nebyla vhodně zvolena. Z tohoto důvodu byla pro detekci odebrána první iterace. Dále byla odebrána poslední iterace, kde může docházet ke zkrácení či prodloužení iterace z důvodu termínu odevzdání projektu. Po odebrání těchto iterací zůstaly iterace z prostředku projektu, kde by měla být délka relativně stálá. Prahová hodnota tedy byla zvolena na hodnotu jedna. Jakmile se tedy signifikantně mění délka iterace ve dvou a více iteracích, je anti-vzor detekován.

6.4 Výsledky

Spolehlivost výsledného systému pro detekci anti-vzorů bude vypočtena pomocí vztahu níže.

$$Spolehlivost[\%] = \frac{počet\ úspěšných\ detekcí}{celkový\ počet\ detekcí} * 100$$

Celkový počet úspěšných detekcí je vypočten na základě manuálního detekování jednotlivých anti-vzorů v projektových datech. Výsledky automatické detekce jsou zobrazeny v tabulce číslo 6.2. Následně budou detekce porovnány s manuální detekcí vybraných anti-vzorů. Výsledky manuální detekce jsou zobrazeny v tabulce číslo 6.3.

Na základě těchto tabulek je výsledná spolehlivost detekce automatického systému nad daty z ASWI projektů vyčíslena na 93.65 %. Postup výpočtu je uveden níže. V diskuzi bude dále uvedena úspěšnost detekcí jednotlivých anti-vzorů, která bude vypočtena stejný postupem.

$$Spolehlivost\ [\%] = \frac{59}{63} * 100 = 93,65\ \%$$

Tabulka 6.2: Výsledky detekce anti-vzorů pomocí realizovaného nástroje

Anti-vzor	Proj. 1	Proj. 2	Proj. 3	Proj. 4	Proj. 5	Proj. 6	Proj. 7	Proj. 8	Proj. 9
Business As Usual	✗	✗	✗	✗	✗	✗	✓	✗	✗
Long Or Non-Existant Feedback Loops	✓	✗	✓	✓	✗	✗	✗	✗	✗
Ninety-Ninety Rule	✗	✗	✗	✗	✗	✗	✗	✗	✗
Road To Nowhere	✗	✗	✓	✓	✗	✗	✗	✗	✗
Specify Nothing	✗	✗	✗	✓	✗	✗	✗	✗	✗
Too Long Sprint	✓	✗	✓	✗	✗	✗	✗	✓	✓
Varying Sprint Length	✓	✗	✓	✗	✗	✗	✗	✓	✗

Legenda:

- ✓– anti-vzor detekován
- ✗– anti-vzor nedetekován
- ▪ – rozpor oproti manuální detekci

Tabulka 6.3: Výsledky manuální detekce anti-vzorů

Anti-vzor	Proj. 1	Proj. 2	Proj. 3	Proj. 4	Proj. 5	Proj. 6	Proj. 7	Proj. 8	Proj. 9
Business As Usual	✗	✗	✗	✗	✗	✗	✗	✗	✗
Long Or Non-Existant Feedback Loops	✓	✗	✗	✓	✗	✗	✗	✗	✗
Ninety-Ninety Rule	✗	✗	✗	✗	✗	✗	✗	✗	✗
Road To Nowhere	✗	✗	✗	✓	✗	✗	✗	✗	✗
Specify Nothing	✗	✗	✗	?	✗	✗	✗	✗	✗
Too Long Sprint	✓	✗	✓	✗	✗	✗	✗	✓	✓
Varying Sprint Length	✓	✗	✓	✗	✗	✗	✗	✓	✗

Legenda:

- ✓– anti-vzor detekován
- ✗– anti-vzor nedetekován
- ? – nelze určit

6.5 Diskuze

Při automatické detekci jednotlivých anti-vzorů došlo k určitým rozporům oproti manuální detekci. Konkrétně u anti-vzorů s názvem Business As Usual, Long Or Non-Existant Feedback Loops, Road To Nowhere a Specify Nothing. Většina těchto rozporů je způsobena absencí některých informací v datovém skladu SPADe, nebo špatným používání ALM nástrojů ze strany projektového týmu. V následujících sekcích jsou diskutovány výsledky detekce jednotlivých anti-vzorů.

Business As Usual

Dle manuální detekce bylo zjištěno, že se retrospektiva konala pravidelně ve všech projektech. S porovnáním oproti automatické detekci byl detekován projekt číslo 7, který byl detekován pro příliš malý počet uskutečněných retrospektiv. Je nutné zmínit, že tento tým nezaznamenával retrospektivy jako úkoly, retrospektivy zaznamenával pouze ve formě poznámek z konání z jednotlivých schůzek. Ve dvou z celkových čtyř iterací byly poznámky z retrospektiv uloženy ve formátu `pdf` a přiloženy jako příloha k wiki stránce. Bohužel tyto informace nejsou obsaženy v datové skladu SPADe a nebylo tedy možné tyto soubory detekovat. Implementace je ale nastavena tak, že pokud by byly informace obsaženy ve SPADe, byl by anti-vzor detekován úspěšně. Následně tedy došlo pouze k detekci dvou retrospektiv z celkových čtyř. Tím nebylo dosaženo minimálního počtu retrospektiv a anti-vzor byl u tohoto projektu detekován.

U všech ostatních projektů došlo k nalezení retrospektiv buď ve formě úkolu, nebo záznamů ve wiki stránkách. Výsledná úspěšnost detekce anti-vzoru Business As Usual byla vypočtena na $88,8\overline{8}\%$.

Long Or Non-Existant Feedback Loops

Tento anti-vzor byl ve dvou případech detekován správně. U projektu číslo 1 bylo detekováno celkem pět schůzek se zákazníkem. Tři schůzky byly konány v rané fázi projektu a dvě další se konaly těsně před odevzdáním finálního produktu. Mezi třetí a čtvrtou schůzkou však vznikl rozestup bezmála dvou měsíců, což bylo v tomto případě považováno za příliš dlouhý rozestup. Časové rozložení schůzek tedy naznačuje tomu, že tým si převzal na začátku projektu informace od zákazníka a na konci projektu přinesl hotový produkt. Toto je velice nebezpečná praktika, jelikož může při finálním předání dojít k velkému množství připomínek ze strany zákazníka a následně může být celý projekt výrazně zpožděn (prodloužen). Podobný postup byl detekován i

u projektu číslo 4, kde se konala schůzka v prvních dvou iteracích a následně až v poslední.

U projektu číslo 3 se konala schůzka se zákazníkem v každé iteraci vyjma jedné. Maximální rozestup, který nastal mezi schůzkami se zákazníkem, byl 27 dní. Avšak průměrná délka iterace v tomto projektu byla vypočtena na 13 dní, prahová hodnota pro detekci byla nastavena na 26 dní. Následně byl tedy anti-vzor detekován.

V ostatních případech byly schůzky detekovány buď v každé iteraci, anebo byl jejich rozestup akceptovatelný. Výsledná úspěšnost detekce anti-vzoru Long Or Non-Existant Feedback Loops byla vypočtena na $88,8\overline{8}\%$.

Ninety-Ninety Rule

Tento anti-vzor nebyl detekován u žádného z devíti vybraných projektů a tento výsledek se shoduje i s manuální detekcí.

Hlavní důvod absence tohoto anti-vzoru ve vybrané sadě projektů je primárně způsoben tím, že na každý průběh projektu dohlíží mentor. Mentor kontroluje průběh projektu a upozorňuje samotný tým na některé problémy v řízení projektu. Tím se následně předchází některým špatným praktikám, které mohou v průběhu projektu vzniknout. Výsledná úspěšnost detekce anti-vzoru Ninety-Ninety Rule byla vypočtena na 100%.

Road To Nowhere

Při automatické detekci tohoto anti-vzoru byly označeny dva projekty. Projekt číslo 4 byl označen korektně, jelikož v projektu není žádný úkol ani wiki stránka, která by naznačovala existenci projektového plánu. Musíme však brát na vědomí, že tým mohl vést projektový plán v nějakém externím nástroji a tato data nejsou dostupná v ALM nástrojích, a tudíž ani ve SPADe. Avšak v rámci dostupných informací je tento anti-vzor detekován správně.

Dalším detekovaným projektem byl projekt číslo 3, kde nebyl nalezen žádný úkol ani wiki stránka připomínající projektový plán. Po manuální detekci bylo však zjištěno, že projekt obsahuje wiki stránku, která popisuje jednotlivé milníky projektu. Tento dokument lze označit jako projektový plán, a tudíž byla detekce anti-vzoru chybná. Důvod neodhalení tohoto dokumentu byl způsoben použitím názvu, se kterým analýza možností vedení plánu nepočítala.

V ostatních projektech byl projektový plán detekován korektně. Výsledná úspěšnost detekce anti-vzoru Road To Nowhere byla vypočtena na $88,8\overline{8}\%$.

Specify Nothing

V rámci této detekce bylo zjištěno, že specifikace byla vytvořena ve všech projektech vyjma jednoho. V rámci tohoto projektu neexistuje žádný úkol ani wiki stránka, která by popisovala vytvoření specifikace. Je zde pouze odkaz do externího nástroje, kde jsou uchovávány všechny dokumenty k projektu. Bohužel k tomuto nástroji není umožněn přístup, tudíž není možné určit, zda byla specifikace vytvořena či nikoli. Pokud by však byla zde specifikace uložena, výsledek detekce by byl opět stejný, jelikož datový sklad SPADe neobsahuje informace z těchto externích nástrojů. Nezabralo by zde ani kritérium minimální průměrné délky popisku úkolu, které je zde pouze 55 znaků.

Výsledná úspěšnost detekce anti-vzoru Specify Nothing byla vypočtena na $88,8\overline{8}\%$.

Too Long Sprint

U tohoto anti-vzoru je implementace detekce anti-vzoru relativně přímočará, a proto byl ve všech případech detekován správně.

Avšak při manuální kontrole dat přímo v ALM nástrojích bylo zjištěno, že některé týmy měly špatně nadefinované termíny jednotlivých iterací. Například u projektu číslo osm bylo zjištěno, že dvě po sobě jdoucí iterace měly rozdílná data počátku iterace, ale data konce iterací se shodovala. Následkem toho se tyto iterace překrývaly. Toto poukazuje na špatné použití ALM nástroje a není možné zjistit, které datum je opravdu správné. I přes tyto chyby zde byly některé další iterace, které přesahovaly maximální prahovou hodnotu pro délku iterace.

Výsledná úspěšnost detekce anti-vzoru Too Long Sprint byla vypočtena na 100%.

Varying Sprint Length

Jako v předchozím případě byl tento anti-vzor detekován ve všech případech správně. Jedná se opět o poměrně přímočarou detekční techniku.

Dále je nutné zmínit, že sada vybraných projektů obsahuje poměrně krátké projekty. V každém projektu je průměrně okolo pěti až šesti iterací. To může být také důvod k výskytu tohoto anti-vzoru, jelikož týmy musejí pracovat s krátkým časovým intervalem a jednotlivé iterace přizpůsobit podle objemu práce, který je ještě nutné dokončit.

Výsledná úspěšnost detekce anti-vzoru Varying Sprint Length byla vypočtena na 100%.

6.6 Možná rozšíření

V této sekci budou navržena možná rozšíření a vylepšení této práce.

Hlubší textová analýza

U většiny anti-vzorů v rámci této práce není možné využít sumarizované hodnoty, ale jsme odkázáni pouze na hledání v dostupných textech jednotlivých projektů. Proto se jako jedno z hlavních vylepšení detekce anti-vzoru nabízí použití některé pokročilé metody pro zpracování textu a na základě těchto metod rozhodovat o přítomnosti některých anti-vzorů.

Využití strojového učení

Trendem dnešním doby je využití umělé inteligence. Proto by bylo vhodným rozšířením této práce vyzkoušet detekci anti-vzorů s využitím některých známých modelů strojového učení například neuronových sítí. Pro detekci s využitím umělé inteligence je klíčové mít větší sadu dostupných dat, kterou by bylo nutné získat z některých dostupných zdrojů.

Detekce dalších anti-vzorů

Teorie okolo problematiky anti-vzorů je velice rozsáhlá a je dostupné velké množství nejrůznějších definic anti-vzorů. Proto by bylo dalším vhodným rozšířením analyzovat některé další anti-vzory, které by bylo možno implementovat do tohoto systému. Podpůrný systém, který byl vytvořen v rámci této práce, byl navrhnut tak, aby bylo co nejjednodušší implementovat případné další anti-vzory.

Rozšíření vytvořené aplikace

Nad rámec této práce vznikla pomocná aplikace pro detekci anti-vzorů, která by mohla být dále rozšířena nebo vylepšena. Možným rozšířením by mohlo být například zobrazení některých významných statistik o projektech do přehledných grafů nebo vytvoření rozhraní pro skládání jednotlivých částí existujících SQL dotazů pro detekci některých dalších anti-vzorů. Jako možné vylepšení pak přichází k úvaze striktnější kontrola při nastavování prahových hodnot, které nejsou v současné době příliš validovány.

7 Závěr

Cílem této diplomové práce byla analýza anti-vzorů v datech nástrojů pro projektové řízení a vytvoření automatické detekce pomocí nástroje Software Process Anti-pattern Detector (SPADe). Za tímto účelem bylo nutné prostudovat problematiku procesu vývoje softwaru a anti-vzorů (kapitola 2). Dále bylo nutné se seznámit s Application Lifecycle Management (ALM) nástroji a samotným nástrojem SPADe, zejména se strukturou uložených dat (kapitola 3). Tím byl splněn bod zadání číslo jedna.

Další část práce byla zaměřena na výběr vhodných anti-vzorů pro automatickou detekci a jejich analýzu (kapitola 4). V této části byl splněn bod zadání číslo dvě.

V poslední části práce byla provedena implementace detekce vybraných anti-vzorů na základě předchozí analýzy. Navíc byla vytvořena pomocná aplikace pro automatickou detekci anti-vzorů, která komunikuje se SPADe a umožňuje konfigurovat detekci analyzovaných anti-vzorů. Tento nástroj byl vytvořen s kladeným důrazem na jednoduchou rozšiřitelnost o další anti-vzory. Úspěšnost vytvořené detekce byla následně ověřena pomocí manuálního procházení dat v ALM nástroji a na závěr této části byly popsány výsledky a diskuze nad nimi (kapitola 5 a 6). Touto částí byly splněny body zadání číslo tři a čtyři.

V rámci experimentu prováděném v této práci bylo zkoumáno sedm vybraných anti-vzorů, jejichž detekce byla vyzkoušena na sadě devíti projektů. Úspěšnost detekce implementovaných anti-vzorů byla vyčíslena na 93,65 %. V některých případech byly anti-vzory chybně detekovány. Tato skutečnost byla nejčastěji způsobena nedostatečnými daty v datovém skladu SPADe a v textu práce byl každý takovýto výskyt řádně analyzován a zdůvodněn.

Výstupy této práce představují sadu automatických detekcí anti-vzorů pro hledání špatných praktik ve vybraných projektech a implementaci pomocné webové aplikace pro detekci anti-vzorů, která byla vytvořena nad rámce této práce. Výsledný nástroj lze použít pro detekci anti-vzorů při vedení jednotlivých softwarových projektů, a tím předejít jejich selhání. Vzhledem k jeho snadné rozšiřitelnosti má velký potenciál pro budoucí užití. V současném stavu může detekce anti-vzorů pomoci včasnému odhalování chyb ve studentských projektech podobných těm, na nichž byl prováděn experiment v rámci této práce. Zároveň ji po budoucím rozšíření bude možné využít k dosažení významných výzkumných výsledků při analýzách open source a průmyslových projektů.

Literatura

[1] Aguanno, K. *The Ideal Iteration Length Revealed* [online]. Agile PM, 2016. [cit. 2021/04/21]. Dostupné z: https://agilepm.com/the-ideal-iteration-length-revealed.

[2] Ambler, S. W. *Common Role Anti-Patterns in Online Discussion Forums* [online]. Ambysoft, 2016. [cit. 2021/04/21]. Dostupné z: http://www.ambysoft.com/essays/discussionListAntiPatterns.html.

[3] Beck, K. *Manifest Agilního vývoje software* [online]. The Agile Alliance, 2001. [cit. 2021/03/10]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html.

[4] Bezděk, P. Identifikace a modelování špatných praktik v projektovém řízení. Diplomová práce, Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2019. Dostupné z: http://hdl.handle.net/11025/39185.

[5] Brada, J. *Projekt vs. proces* [online]. ApuTime blog, 2020. [cit. 2021/03/10]. Dostupné z: https://blog.aputime.com/post/projekt-vs-proces.

[6] Brada, P. – Picha, P. Software Process Anti-pattern Catalogue. In *EuroPLoP '19: Proceedings of the 24th European Conference on Pattern Languages of Programs*, New York, NY, USA, 2019. ACM.

[7] Brada, P. Agilní proces pro výuku softwarového inženýrství v předmětu ASWI. Technická zpráva, Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2011. Dostupné z: http://www.kiv.zcu.cz/~brada/files/aswi/aswi-proces.2020/cz.zcu.kiv.aswi_plug-in_cz/guidances/whitepapers/resources/aswi-proces-2011-spec.pdf.

[8] Brown, W. J. et al. *AntiPatterns, Refactoring Software, Architectures, and Projects in Crisis*. Robert Ipsen, 1998. ISBN 0-471-19713-0.

[9] *Plán projektu (Project Plan)* [online]. ManagementMania.com, 2019. [cit. 2021/04/21]. Dostupné z: https://managementmania.com/cs/plan-projektu.

[10] Duda, J. *Procesní a projektové řízení* [online]. management.cz, 2014. [cit. 2021/03/10]. Dostupné z: http://www.management.cz/procesni-a-projektove-rizeni/.

[11] Eloranta, V.-P. – Koskimies, K. – Mikkonen, T. Exploring ScrumBut—An empirical study of Scrum anti-patterns. *Information and Software Technology*. 2016, 74. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2015.12.003. Dostupné z: `https://www.sciencedirect.com/science/article/pii/S0950584915002050`.

[12] Eneh, T. *Most popular CI/CD pipelines and tools* [online]. medium.com, 2019. [cit. 2021/03/10]. Dostupné z: `https://medium.com/faun/most-popular-ci-cd-pipelines-and-tools-ccfdce429867`.

[13] *ETL Process in Data Warehouse* [online]. Geeks for Geeks, 2019. [cit. 2021/04/21]. Dostupné z: `https://www.geeksforgeeks.org/etl-process-in-data-warehouse/`.

[14] Grossmann, L. *Metodiky vývoje softwaru* [online]. itnetwork.cz, 2020. [cit. 2021/03/10]. Dostupné z: `https://www.itnetwork.cz/navrh/metodiky`.

[15] Humphrey, W. S. Introduction to Software Process Improvement. Technická zpráva ESC-TR-92-007, Software Engineering Institute, 1993. Dostupné z: `https://resources.sei.cmu.edu/asset_files/TechnicalReport/1992_005_001_16049.pdf`.

[16] Kroll, P. – Kruchten, P. *The Rational Unified Process made easy: A practitioner's guide to the RUP*. Addison-Wesley, 2003. ISBN 0-321-16609-4.

[17] Laplante, P. A. – Neill, C. J. *Antipatterns, Identification, Refactoring, and Management*. CRC Press, 2006. ISBN 0-8493-2994-9.

[18] Mantle, M. – Lichty, R. *Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams*. Addison-Wesley Professional, 2012. ISBN 978-0-321-82203-1.

[19] Mersino, A. *Best Agile Team Size for High Performance?* [online]. VITALITY CHICAGO INC., 2017. [cit. 2021/04/21]. Dostupné z: `https://vitalitychicago.com/blog/best-size-for-my-agile-team-high-performing-teams-magic-number/`.

[20] Preston-Werner, T. *Semantic Versioning 2.0.0* [online]. www.semver.org. [cit. 2021/04/21]. Dostupné z: `https://semver.org/`.

[21] Pícha, P. Detecting software development process patterns in project data. Technická zpráva DCSE/TR-2019-04, Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2019. Dostupné z: `http://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2019/Rigo_P%C3%ADcha_2019_4.pdf`.

[22] Schrader, M. *Specify Nothing* [online]. 2010. [cit. 2021/04/21]. Dostupné z: http://wiki.c2.com/?SpecifyNothing.

[23] *PMBOK (Project Management Body of Knowledge)* [online]. Management Mania, 2016. [cit. 2021/04/21]. Dostupné z: https://managementmania.com/cs/project-management-body-of-knowledge.

[24] *Co je a co není PRINCE2, PRINCE2 Agile a Scrum?* [online]. POTIFOB. [cit. 2021/04/21]. Dostupné z: https://potifob.cz/Co-je-a-co-neni-PRINCE2-PRINCE2-Agile-a-Scrum.

[25] *Software Architecture AntiPatterns* [online]. SourceMaking, 2018. [cit. 2021/04/21]. Dostupné z: https://sourcemaking.com/antipatterns/software-architecture-antipatterns.

[26] Sutherland, J. – Schwaber, K. *The Scrum methodology. In Business object design and implementation: OOPSLA workshop.* Springer-Verlag, 1995.

[27] *The Kanban method in IT development projects* [online]. www.bocasay.com. [cit. 2021/04/21]. Dostupné z: https://www.bocasay.com/kanban-method-it-development-projects/.

[28] *Understanding Agile Scrum in 10 minutes* [online]. Tuleap, 2019. [cit. 2021/03/10]. Dostupné z: https://www.tuleap.org/agile/agile-scrum-in-10-minutes/.

[29] *Vodopádový model (Waterfall model)* [online]. Management Mania, 2015. [cit. 2021/03/10]. Dostupné z: https://managementmania.com/cs/vodopadovy-model-waterfall-model.

[30] *What is a data warehouse?* [online]. Amazon, 2019. [cit. 2021/04/21]. Dostupné z: https://aws.amazon.com/data-warehouse/.

Seznam zkratek

- **ALM** – Application Lifecycle Management – jedná se o nástroje, které jsou využívány k podpoře vývoji softwaru.
- **API** – Application Programming Interface – rozhraní pro programování aplikací.
- **CI/CD** – Continuous Integration/Continuous Deployment – představuje nástroje a postupy pro správné dodání nové verze softwaru.
- **CVS** – Concurrent Version System – systém sloužící ke správě verzí projektu.
- **DMS** – Document management system – systém určený ke správě elektronických dokumentů.
- **ETL** – Extract, transform, load – proces extrakce, transformace a nahrání dat z jednoho či více zdrojů do datového skladu.
- **JDBC** – Java Database Connectivity – rozhraní pro přístup k relačním databázím pomocí jazyka Java.
- **JEE** – Java Enterprise Edition – nástroj součástí platformy Java určený pro vývoj a provoz podnikových aplikací a informačních systémů.
- **JSON** – JavaScript Object Notation – způsob zápisu dat nezávislý na počítačové platformě, určený pro přenos dat.
- **PDF** – Portable Document Format – formát souborů pro výměnu elektronických dokumentů.
- **PMBOK** – Project Management Body of Knowledge – mezinárodně uznávaný standard řízení projektů.
- **PRINCE2** – Projects in Controlled Environments 2nd version – flexibilní metodika pro řízení projektů.
- **REST** – Representational state transfer – architektura rozhraní navržená pro distribuované systémy.
- **RUP** – Rational Unified Process – metodiky vývoje softwaru pro rozsáhlejší projekty a větší vývojové týmy.

- **SQL** – Structured Query Language – standardizovaný strukturovaný dotazovací jazyk, který je používán pro práci s daty v relačních databázích.
- **XML** – Extensible Markup Language – obecný značkovací jazyk, který se používá pro přenos a serializaci dat.

Seznam obrázků

2.1 Vodopádový model [29] 5
2.2 Fáze RUP [16] 7
2.3 Milníky metodiky RUP 8
2.4 Scrum proces [28] 10
2.5 Proces vývoje v extrémní programování [14] 12
2.6 Ukázka Kanban tabule [27] 14

3.1 Proces CI/CD [12] 19
3.2 Architektura nástroje SPADe [21] 23
3.3 Doménový metamodel datového skladu nástroje SPADe [21] 25

4.1 Stavový diagram návrhu detekce pro anti-vzor Business As Usual 35
4.2 Stavový diagram návrhu detekce pro anti-vzor Long Or Non-Existant Feedback Loops 38
4.3 Stavový diagram návrhu detekce pro anti-vzor Ninety-Ninety Rule 41
4.4 Stavový diagram návrhu detekce pro anti-vzor Road To Nowhere 43
4.5 Stavový diagram návrhu detekce pro anti-vzor Specify Nothing 46
4.6 Stavový diagram návrhu detekce pro anti-vzor Too Long Sprint 48
4.7 Stavový diagram návrhu detekce pro anti-vzor Varying Sprint Length 50
4.8 Návrh struktury aplikace 53

6.1 Schéma průběhu ASWI procesu 65

B.1 Vytvoření databáze 87
B.2 Obnova databáze ze zálohy 88
B.3 Dodatečná konfigurace databáze 88
C.4 Hlavní obrazovka aplikace 93
C.5 Výběr projektů 94
C.6 Chybová hláška – žádný vybraný projekt 94
C.7 Výběr anti-vzorů 95
C.8 Chybová hláška – žádný vybraný anti-vzor 95
C.9 Výsledná tabulka 96
C.10 Zobrazení podrobností o detekci 96
C.11 Návrat na hlavní stránku 97

C.12 Zobrazení seznamu prahových hodnot 98
C.13 Zobrazení prahových hodnot 99
C.14 Uložení prahových hodnot 99
C.15 Hláška o úspěšném uložení prahových hodnot 100
C.16 Hláška o neplatném nastavení prahových hodnot 100
C.17 Zobrazení podrobností o projektu 100
C.18 Podrobnosti o vybraném projektu 101
C.19 Zobrazení podrobností o anti-vzoru 101
C.20 Podrobnosti o vybraném anti-vzoru 102

Seznam tabulek

3.1 Atributy pohledu `artifactView` 27
3.2 Atributy pohledu `personView` 27
3.3 Atributy pohledu `personWithRolesView` 28
3.4 Atributy pohledu `workUnitView` 28
3.5 Atributy pohledu `configurationView` 29
3.6 Atributy pohledu `commitedConfigView` 29
3.7 Atributy pohledu `commitView` 30
3.8 Atributy pohledu `fieldChangeView` 31

5.1 Struktura projektu aplikace 59

6.1 Výsledná sada projektů 64
6.2 Výsledky detekce anti-vzorů pomocí realizovaného nástroje . 70
6.3 Výsledky manuální detekce anti-vzorů 71

Přílohy

A Obsah ZIP souboru

Adresářová struktura a obsah ZIP souboru je popsán v následujícím seznamu:

- `Text_prace` – Složka obsahující text diplomové práce, obrázky a zdrojové kódy textu v LaTeXu.
- `Poster` – Adresář obsahující výsledný poster ve formátu `pdf` a `pub`.
- `Aplikace_a_knihovny` – Složka obsahující zdrojové kódy aplikace.
- `Vstupni_data` – Adresář obsahující zálohu datového skladu SPADe.
- `Vysledky` – Adresář obsahující soubor s výsledky automatické detekce anti-vzorů.
- `Readme.txt` – Textový soubor popisující obsah zip souboru a adresářovou strukturu.

B Příručka nasazení

Pro usnadnění nasazení aplikace je celkový projekt nakonfigurován pro spuštění v Dockeru. V této příručce je popsáno, jak pomocí nástroje Docker spustit tuto aplikaci.

B.1 Potřebné nástroje

Ke spuštění aplikace jsou zapotřebí následující nástroje:

- Docker [1]
- Docker Compose [2]
- GIT [3] (pouze v případě absence přiloženého ZIP souboru)

B.2 Postup spuštění aplikace

Postup jednotlivých kroků ke spuštění aplikace je popsán v následujícím seznamu:

1. Zkopírovat obsah složky `AntiPatternDetectionApp` z přiloženého ZIP souboru nebo pomocí nástroje GIT udělat klon veřejně dostupného repozitáře pomocí následujícího příkazu.

   ```
   git clone https://github.com/OndrejVane/
   AntiPatternDetectionApp.git
   ```

2. V případě, že chcete aplikaci napojit na běžící instanci datového skladu SPADe, pokračujte sekcí B.5.

3. Otevřít příkazový řádek a přesunout se do kořenové složky projektu s názvem `AntiPatternDetectionApp`. V této složce by se měl nacházet soubor `docker-compose.yml`.

4. V této složce spustit následující příkaz.

   ```
   docker-compose build
   ```

[1] Příručku pro instalaci nástroje Docker naleznete na adrese `https://docs.docker.com/get-docker/`

[2] Příručku pro instalaci nástroje Docker Compose naleznete na adrese `https://docs.docker.com/compose/install/`

[3] Příručku pro instalaci nástroje GIT naleznete na adrese `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

Tento příkaz vytvoří potřebné komponenty aplikace, které se v terminologii Dockeru nazývají jako image.

5. Po vytvoření potřebných komponent aplikace spustit následující příkaz.

```
docker-compose up -d
```

Tento příkaz spustí aplikaci na portu `8080`. Dále také spustí MySQL databázi na portu `3306` a rozhraní phpMyAdmin pro administraci databáze na portu `8082`. Pro správnou funkčnost aplikace je ještě zapotřebí obnovit databázi ze zálohy a databázi nakonfigurovat.

B.3 Obnova a konfigurace databáze

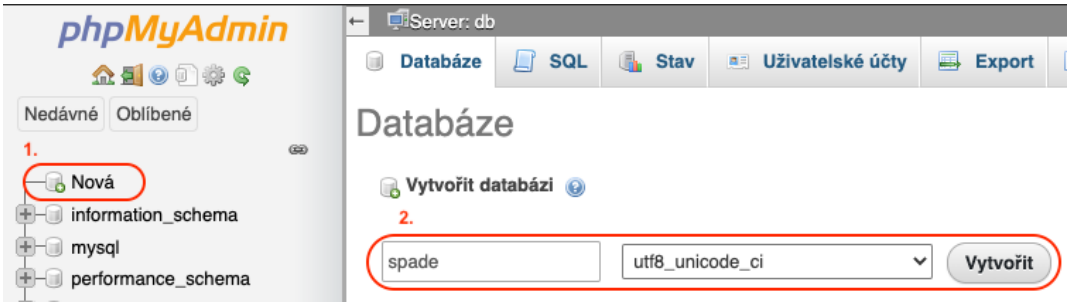
V této části jsou popsány dva postupy pro nastavení a obnovení databáze ze zálohy.

Pomocí phpMyAdmin

1. Otevřít rozhraní phpMyAdmin, které běží na portu `8082`.
2. Přihlásit se do rozhraní phpMyAdmin pomocí následujících údajů.

   Jméno: `root` a Heslo: `testtest`

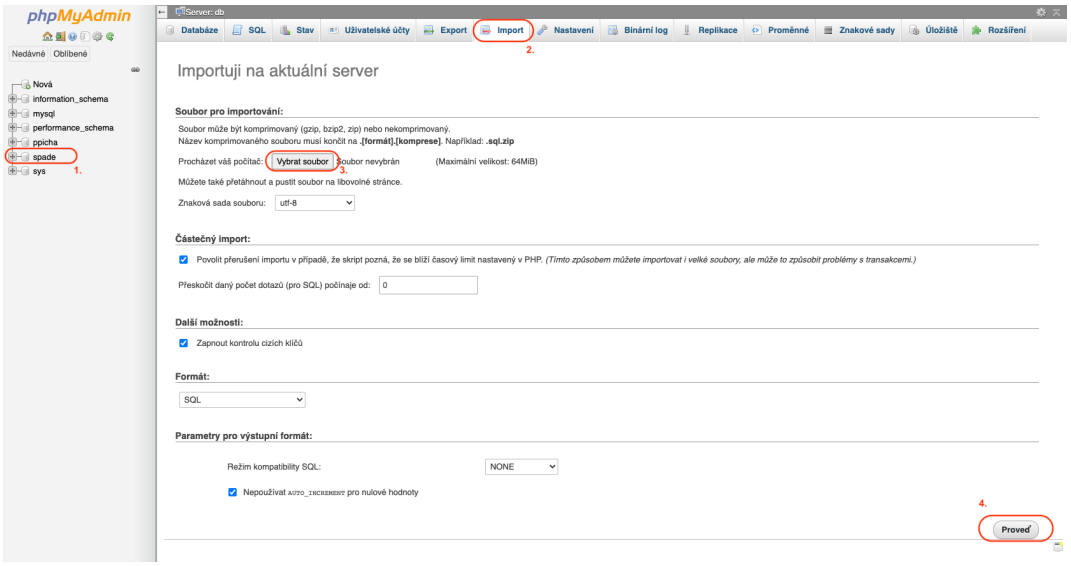
3. Vytvořit novou databázi s názvem `spade` a kódováním `utf8_unicode_ci`, viz obrázek číslo B.1.



Obrázek B.1: Vytvoření databáze

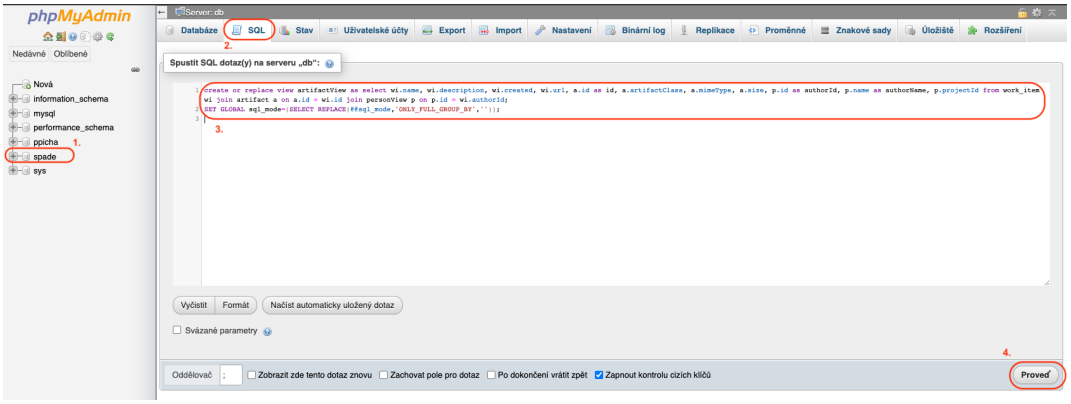
4. Otevřít vytvořenou databázi a přejít do sekce *Import*. Na této obrazovce stisknout tlačítko *Vybrat soubor*. Soubor se zálohou databáze je uložen v kořenové složce projektu s názvem `db_dump.sql`. Po vybrání

souboru se zálohou stisknout tlačítko *Proveď* v pravém dolním rohu. Postup je znázorněn na obrázku číslo B.2. Tím dojde k vytvoření všech potřebných tabulek a pohledů. Pozor tato akce může trvat i několik minut.



Obrázek B.2: Obnova databáze ze zálohy

5. Posledním krokem je konfigurace vytvořené databáze. Pro konfiguraci databáze je zapotřebí obsah souboru `config.sql`, který je opět uložen v kořenové složce projektu. V phpMyAdmin přejdeme do záložky *SQL*, vložíme obsah souboru do pole pro SQL a stiskneme tlačítko *Proveď*, viz obrázek číslo B.3. Tím by měla být databáze úspěšně nakonfigurována a aplikace plně funkční.



Obrázek B.3: Dodatečná konfigurace databáze

Pomocí příkazového řádku

1. Spustit v příkazovém řádku následující příkaz.

   ```
   docker exec -it mysql-db bin/bash
   ```

   Tím dojde k připojení terminálu k běžícímu kontejneru MySQL databáze.

2. Přihlásit se do konzole databáze následujícím příkazem.

   ```
   mysql -u root -p
   ```

   Následně je uživatel vyzván pro vložení hesla pro uživatele *root*.

   Heslo: `testtest`

3. V příkazovém řádku MySQL vytvoříme databázi následujícím příkazem.

   ```
   create database spade;
   ```

4. Přepnout se do vytvořené databáze následujícím příkazem.

   ```
   use spade;
   ```

5. Dále je zapotřebí načíst databázi ze zálohy. Soubor se zálohou je již nakopírován na databázovém serveru. Stačí tedy spustit následující příkaz.

   ```
   source /usr/local/etc/db_dump.sql
   ```

   Tím dojde ke kompletní obnově všech tabulek a pohledů. Pozor tato operace může trvat několik minut.

6. Posledním krokem je konfigurace databáze. Soubor pro konfiguraci je opět nakopírován na databázový server. Stačí spustit následující příkaz.

   ```
   source /usr/local/etc/config.sql
   ```

   Tím dojde ke konfiguraci databází a po tomto kroku by měla být aplikace plně funkční.

B.4 Konfigurace aplikace

Konfigurace celkové aplikace lze provádět v souboru `docker-compose.yml`. V tomto souboru je možné nakonfigurovat, na kterých portech budou služby spuštěny a také heslo pro připojení do databáze. Pro aplikování změn je nutné restartovat aplikaci. Aktuální soubor je zobrazen níže.

```
version: '3.3'

services:
  #service 1: definition of mysql database
  db:
    image: mysql:latest
    container_name: mysql-db
    environment:
      - MYSQL_ROOT_PASSWORD=testtest
    ports:
      - "3306:3306"
    restart: always
    volumes:
      - ./db_dump.sql:/usr/local/etc/db_dump.sql
      - ./config.sql:/usr/local/etc/config.sql

  #service 2: definition of phpMyAdmin
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: my-php-myadmin
    ports:
      - "8082:80"
    restart: always
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: testtest
    volumes:
      - ./uploads.ini:/usr/local/etc/php/conf.d/uploads.ini

  #service 3: definition of your spring-boot app
  antipatterndetection:
    image: anti-pattern-detection
    container_name: anti-pattern-detection-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
```

```
41      restart: always
42      depends_on:
43        - db
44      environment:
45        SPRING_DATASOURCE_URL: jdbc:mysql://mysql-db:3306/spade
46        SPRING_DATASOURCE_USERNAME: root
47        SPRING_DATASOURCE_PASSWORD: testtest
```

Změna portu

Port je uváděn ve formátu `<veřejný port>:<port uvnitř kontejneru>`. Pro změnu portu stačí přepsat pouze část nalevo od dvojtečky, tedy veřejný port. Pokud dojde ke změně veřejného portu MySQL databáze, je nutné provést úpravu i na řádku 45, kde je definován JDBC konektor pro aplikaci.

Změna hesla databáze

Heslo pro uživatele `root` je definováno na řádce číslo devět s označením `MYSQL_ROOT_PASSWORD`. Pokud dojde ke změně tohoto hesla, je nutné ho upravit také na dalších příslušných místech. Konkrétně se jedná o řádky 29 a 47 s označením `SPRING_DATASOURCE_PASSWORD`.

B.5 Spuštění aplikace s napojením na SPADe

Tato sekce je určena pro spuštění aplikace v případě, že je k dispozici běžící instance datového skladu SPADe. Postup spuštění a připojení aplikace k běžící instanci SPADe je popsána v následujícím seznamu. Potřebné nástroje pro spuštění jsou uvedeny v sekci B.1.

1. Zkopírovat obsah složky `AntiPatternDetectionApp` z přiloženého ZIP souboru, nebo pomocí nástroje GIT udělat klon veřejně dostupného repozitáře pomocí následujícího příkazu. Pokud byl již tento krok proveden, tak přejděte k dalšímu bodu.

   ```
   git clone https://github.com/OndrejVane/
   AntiPatternDetectionApp.git
   ```

2. Otevřít kořenový adresář projektu s názvem `AntiPatternDetectionApp`.

3. Otevřít soubor s názvem `application.properties`, který se nachází ve složce `./src/main/resources`.

4. V tomto souboru se nachází údaje pro připojení k MySQL databázi, které je nutné nastavit pro korektní připojení ke SPADe.

```
spring.datasource.url=jdbc:mysql://<SPADe url>:<port SPADe>/<SPADe db>
spring.datasource.username=<login k db SPADe>
spring.datasource.password=<heslo k db SPADe>
```

5. Po korektní nastavení těchto hodnot otevřeme příkazový řádek a přejdeme do kořenové složky projektu `AntiPatternDetectionApp`. V této složce spustíme následující příkaz.

```
docker build -t <název image> .
```

   Tento příkaz vytvoří komponentu aplikace s příslušným názvem, která se v terminologii Dockeru nazývají image.

6. Vytvořený image spustíme následujícím příkazem.

```
docker run <název image> -p 8080:<číslo portu>
```

   Číslo portu označuje, na jakém portu bude aplikace dostupná.

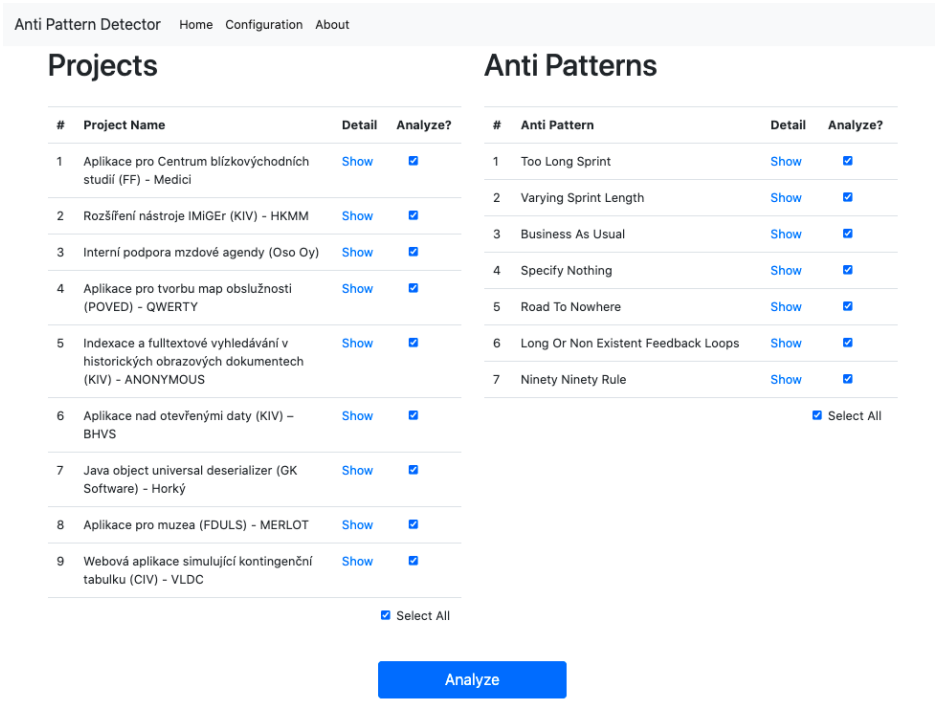
C Uživatelská příručka

Aplikace je určena pro přehledné a rychlé analyzování jednotlivých projektů a anti-vzorů.

C.1 Detekce anti-vzorů

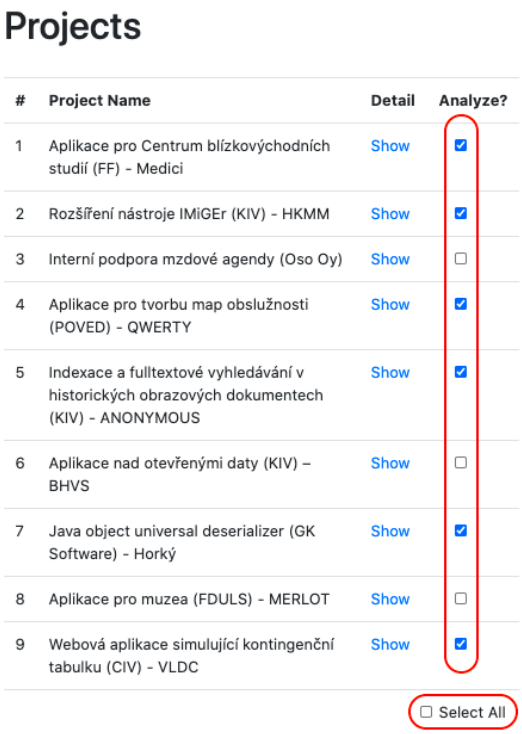
Postup detekce anti-vzorů je popsán v následujícím seznamu.

1. Otevřít prohlížeč a přejít na hlavní stránku aplikace. V horní části obrazovky se nachází hlavní navigační menu, v levé části se nachází tabulka s projekty a v pravé části tabulka s anti-vzory. Hlavní obrazovka je zobrazena na obrázku číslo C.4.

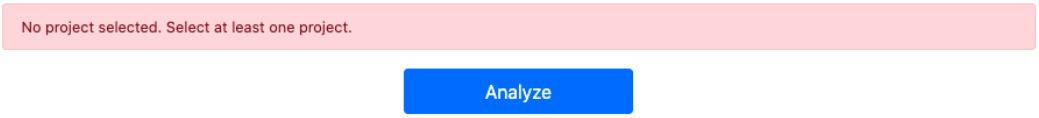


Obrázek C.4: Hlavní obrazovka aplikace

2. Vybrat všechny projekty, které chcete analyzovat. Pozor musí být vybrán alespoň jeden projekt (v případě nevybrání žádného projektu se zobrazí hláška, viz obrázek číslo C.6). Pro výběr daného projektu použijte zaškrtávací políčko, viz obrázek číslo C.5. Pro označení nebo odznačení všech projektů použijte zaškrtávací políčko `Select All`.

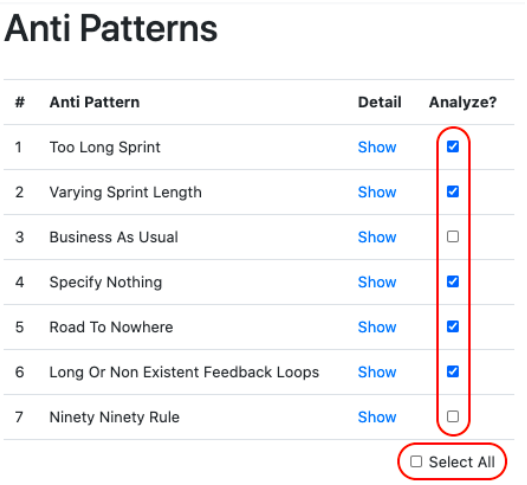


Obrázek C.5: Výběr projektů

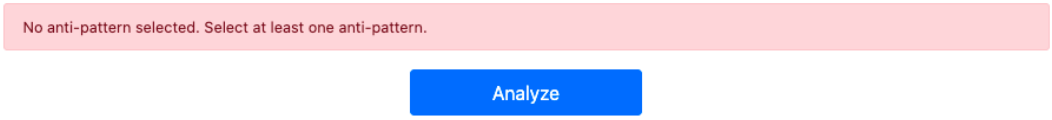


Obrázek C.6: Chybová hláška – žádný vybraný projekt

3. Vybrat všechny anti-vzory, které chcete ve vybraných projektech detekovat. Pozor musí být vybrán alespoň jeden anti-vzor (v případě nevybrání žádného anti-vzoru se zobrazí hláška, viz obrázek číslo C.8). Pro výběr daného anti-vzoru použijte zaškrtávací políčko, viz obrázek číslo C.7. Pro označení nebo odznačení všech anti-vzorů použijte zaškrtávací políčko `Select All`.



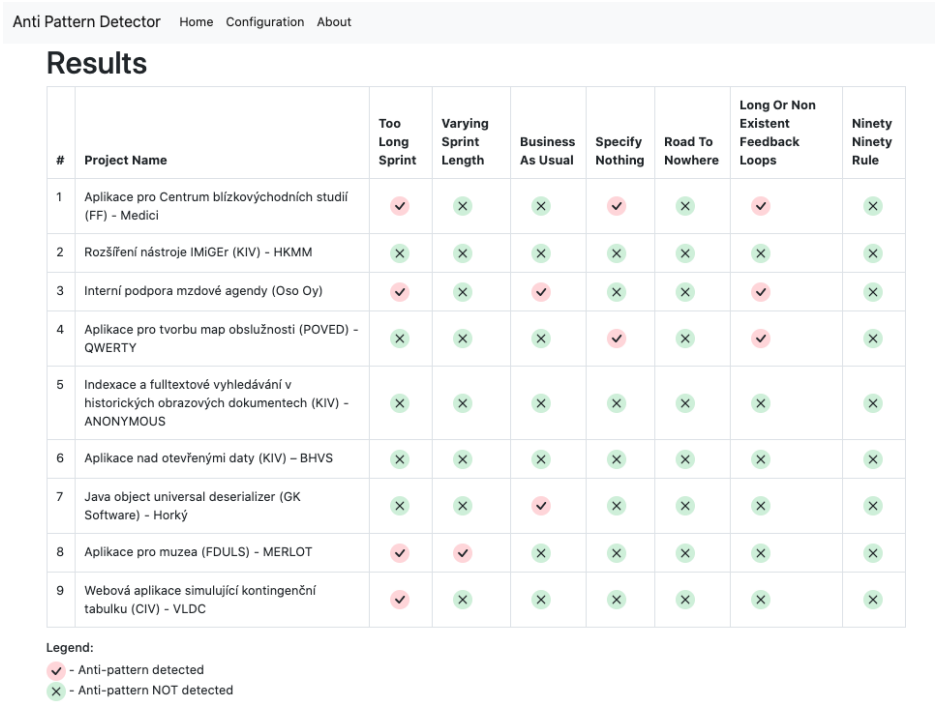
Obrázek C.7: Výběr anti-vzorů



Obrázek C.8: Chybová hláška – žádný vybraný anti-vzor

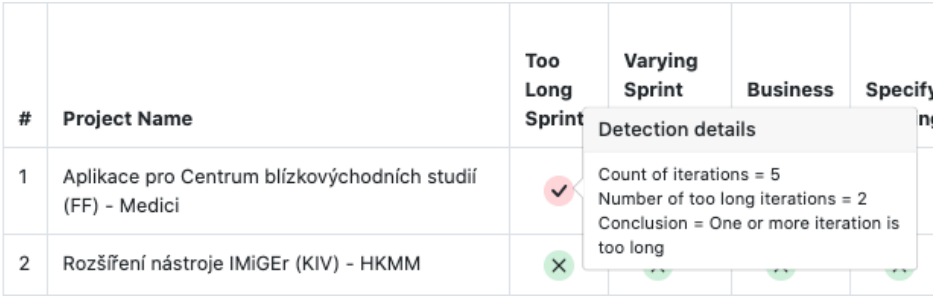
4. Po vybrání příslušných projektů a anti-vzorů stačí stisknout tlačítko `Analyze`.

5. Po dokončení analýzy dojde k zobrazení tabulky s výsledky detekce a legendy, viz obrázek číslo C.9.



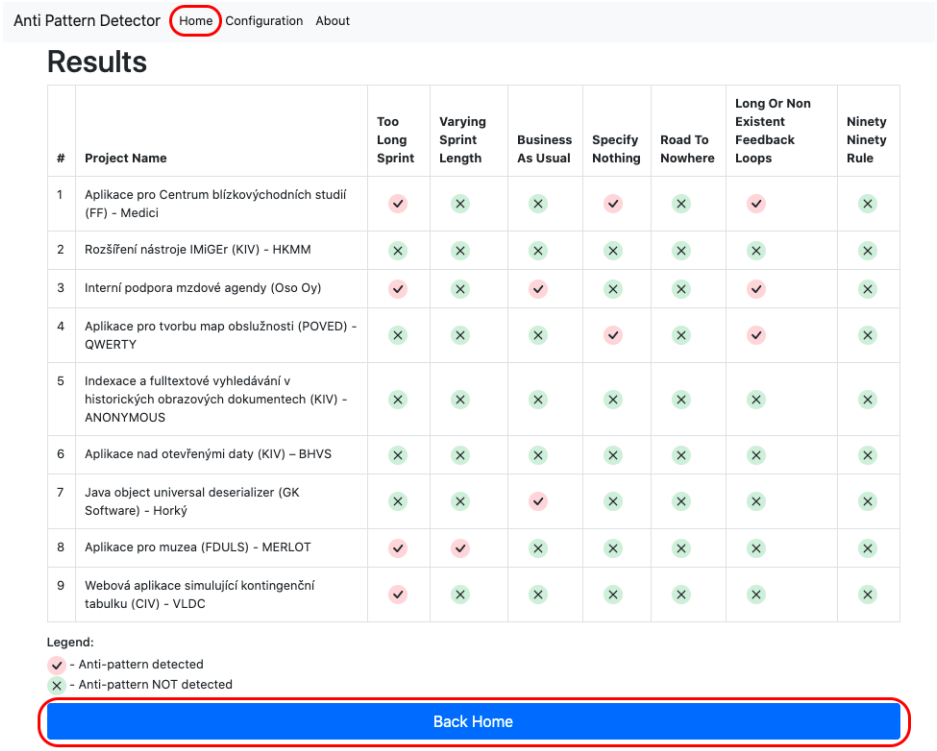
Obrázek C.9: Výsledná tabulka

6. U každého výsledku detekce lze zobrazit dialogové okno s podrobnostmi o detekci. Dialogové okno lze zobrazit po kliknutí na ikonu, která znázorňuje výsledek detekce anti-vzoru u příslušného projektu, viz obrázek číslo C.10.



Obrázek C.10: Zobrazení podrobností o detekci

7. Pro opětovný výběr projektů nebo anti-vzorů stiskněte tlačítko `Back Home` nebo tlačítko `Home` v hlavním menu, viz obrázek číslo C.11. Tím dojde k otevření hlavní stránky aplikace.

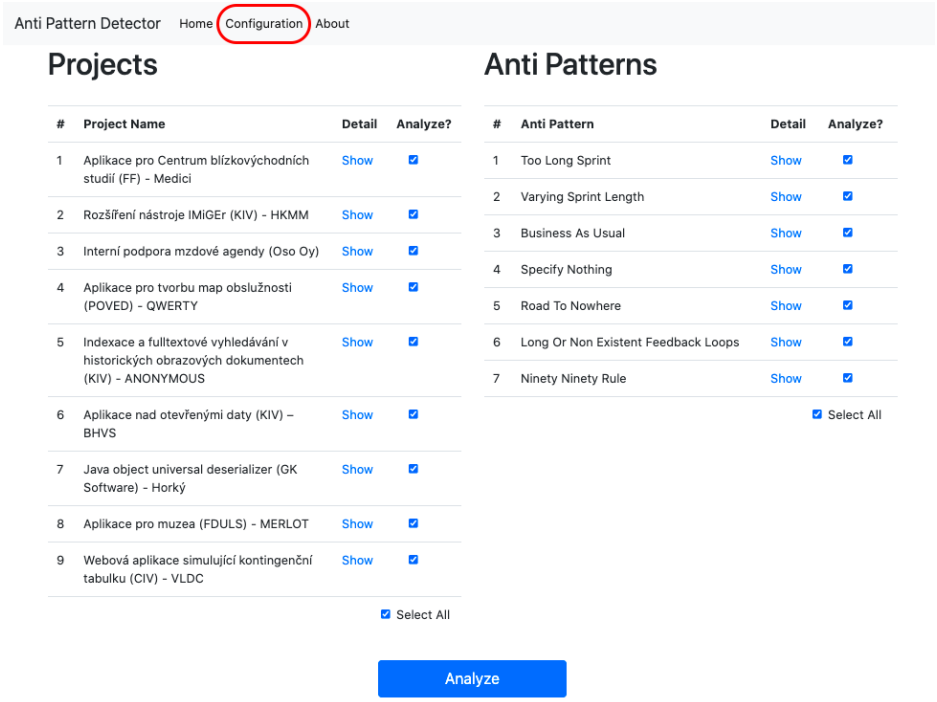


Obrázek C.11: Návrat na hlavní stránku

C.2 Konfigurace prahových hodnot

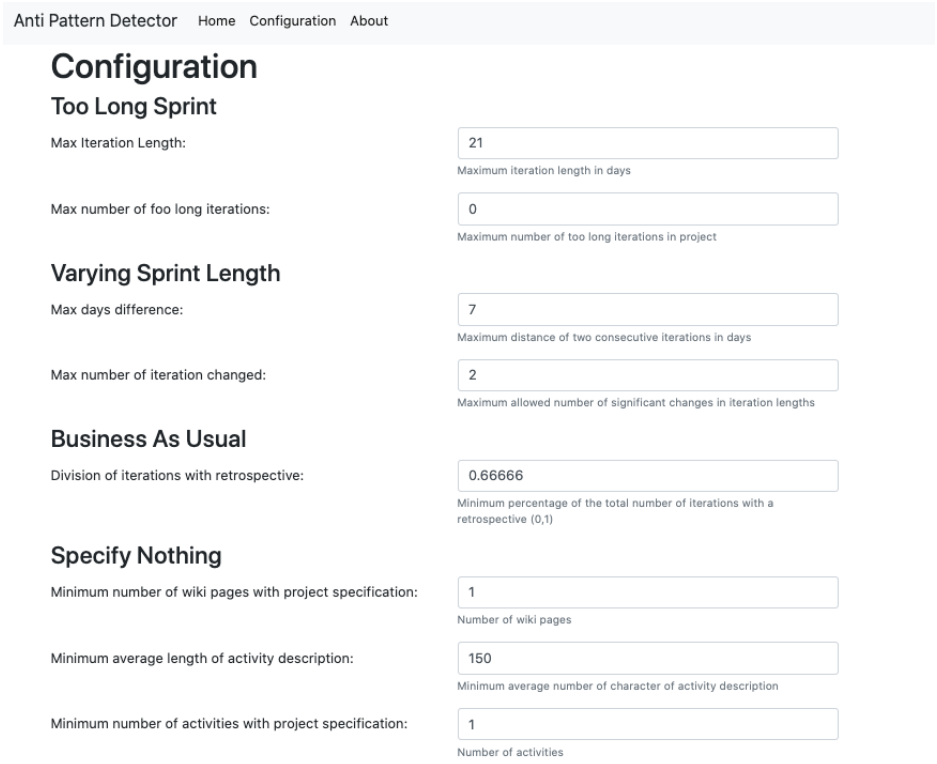
Pomocí výsledné aplikace lze konfigurovat jednotlivé prahové hodnoty, které byly definovány v kapitole číslo 4.2. Postup pro nastavení prahových hodnot je popsán v následujícím seznamu.

1. Přejít na stránku konfigurace pomocí tlačítka `Configuration` v hlavním menu, viz obrázek číslo C.12.



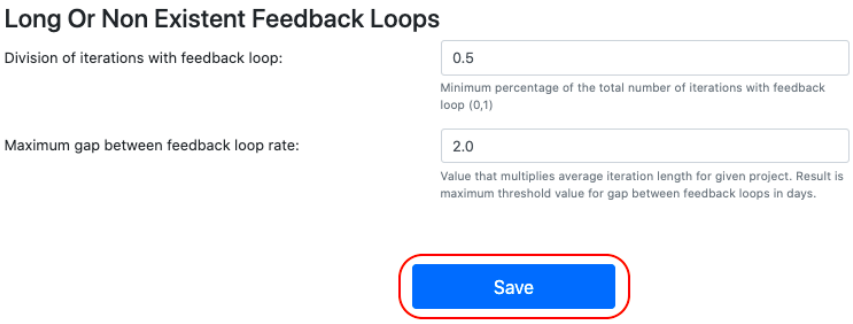
Obrázek C.12: Zobrazení seznamu prahových hodnot

2. Následně se zobrazí seznam všech implementovaných anti-vzorů a jejich aktuálně nastavených prahových hodnot, viz obrázek číslo C.13.



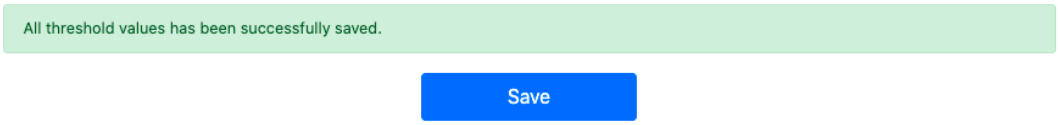
Obrázek C.13: Zobrazení prahových hodnot

3. Pro editaci jedné nebo více prahových hodnot stačí přepsat číselnou hodnotu u dané prahové hodnoty a následně stisknout tlačítko `Save` ve spodní části obrazovky, viz obrázek číslo C.14.



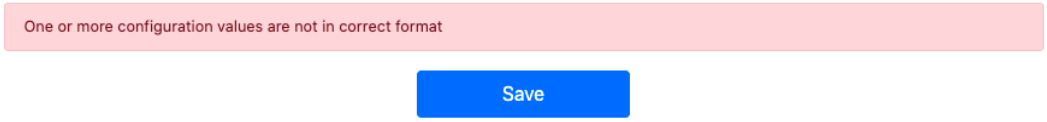
Obrázek C.14: Uložení prahových hodnot

4. V případě, že byly hodnoty správně nastaveny, dojde k zobrazení hlášky, viz obrázek číslo C.15.



Obrázek C.15: Hláška o úspěšném uložení prahových hodnot

5. V případě neúspěšného uložení dojde k výpisu chybové hlášky, viz obrázek číslo C.16, a je nutné nastavení prahových hodnot upravit a zopakovat.

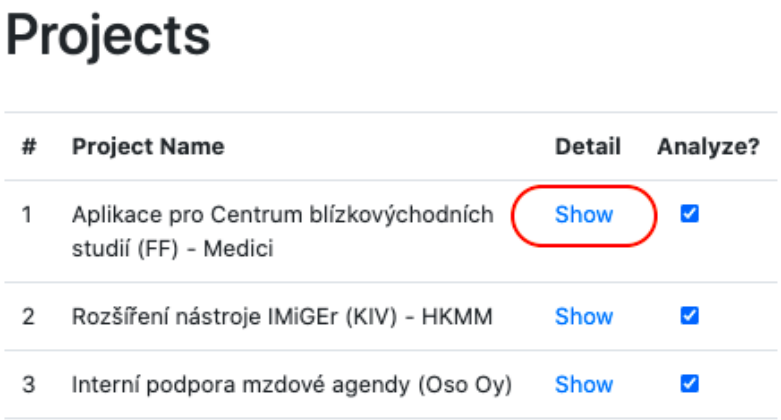


Obrázek C.16: Hláška o neplatném nastavení prahových hodnot

C.3 Zobrazení informací o projektu

Aplikace umožňuje zobrazení základních informací o vybraném projektu. Postup pro zobrazení informací o projektu je popsán v následujícím seznamu.

1. Přejít na hlavní stránku aplikace.
2. Stisknout tlačítko `Show` vedle projektu, u kterého chcete zobrazit podrobnosti, viz obrázek číslo C.17.



Obrázek C.17: Zobrazení podrobností o projektu

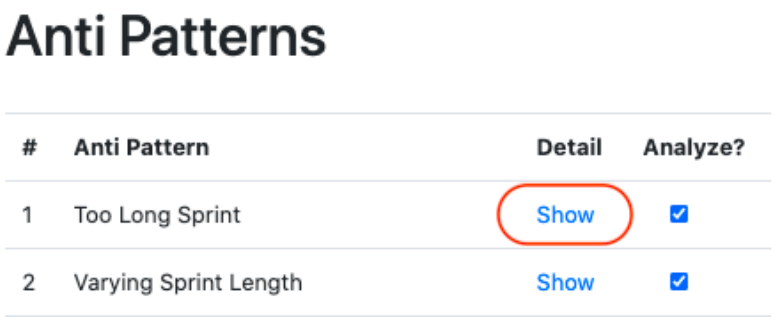
3. Následně se zobrazí obrazovka s informacemi o projektu, viz obrázek číslo C.18.

Obrázek C.18: Podrobnosti o vybraném projektu

C.4 Zobrazení informací o anti-vzoru

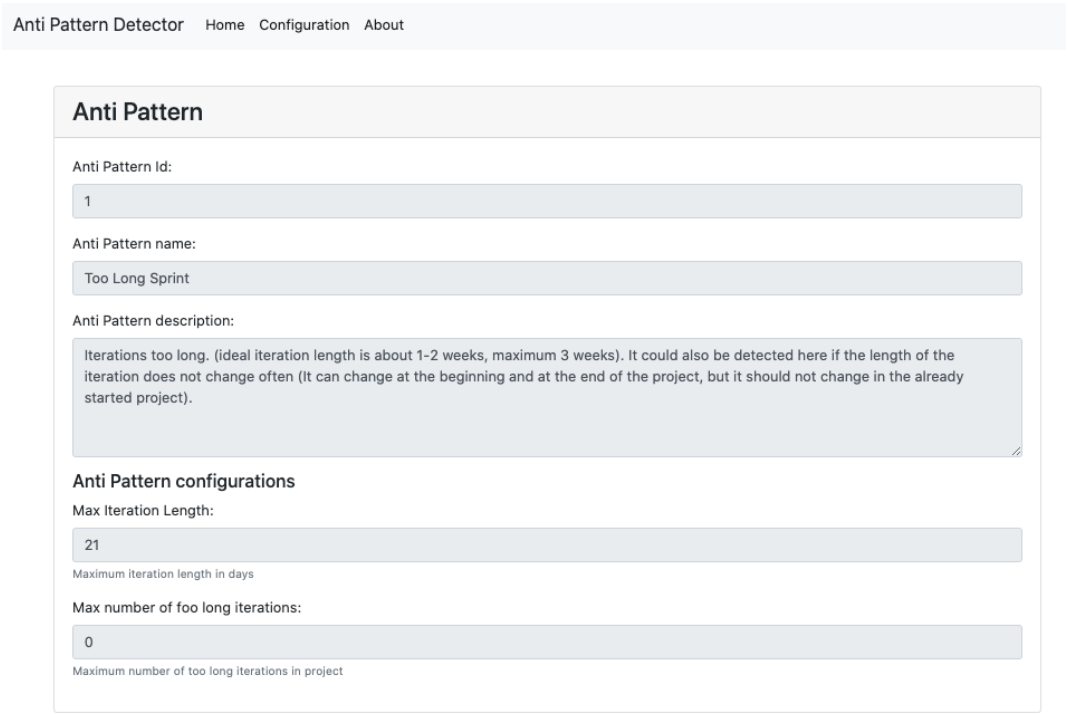
Aplikace umožňuje zobrazení základních informací o vybraném anti-vzoru. Postup pro zobrazení informací o anti-vzoru je popsán v následujícím seznamu.

1. Přejít na hlavní stránku aplikace.
2. Stisknout tlačítko `Show` vedle anti-vzoru, u kterého chcete zobrazit podrobnosti, viz obrázek číslo C.19.



Obrázek C.19: Zobrazení podrobností o anti-vzoru

3. Následně se zobrazí obrazovka s informacemi o anti-vzoru a všechny jeho prahové hodnoty, viz obrázek číslo C.20.



Obrázek C.20: Podrobnosti o vybraném anti-vzoru