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

Diplomová práce

Modulární detekce procesních anti-patternů v projektových datech

Plzeň 2024 Bc. Petr Štěpánek

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petr ŠTĚPÁNEK**
Osobní číslo:	**A22N0059P**
Studijní program:	**N0613A140040 Softwarové a informační systémy**
Téma práce:	**Modulární detekce procesních anti-patternů v projektových datech**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou anti-vzorů v softwarových procesech a projektovém řízení, nástrojů pro řízení projektů a nástrojovou sadou SPADe.
2. Navrhněte vylepšení schopností SPADe pro detekci anti-vzorů modulárním způsobem, jež umožní definovat jednotlivé metriky a prahové hodnoty dat a následnou kompozici indikátorů a modelů samotných anti-vzorů.
3. Implementujte změny v současné podobě systému SPADe tak, aby reflektoval modulární návrh z předchozího bodu.
4. Použitelnost a správnost nového systému detekcí, jakož i jeho přínos, ověřte převodem současných detekcí a operacionalizací několika nově vybraných anti-vzorů na určené sadě projektových dat.
5. Funkčnost změn v aplikaci a jejich integraci se zbytkem systému ověřte patřičnými testy.

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

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Pícha**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **8. září 2023**
Termín odevzdání diplomové práce: **16. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 11. října 2023

Prohlášení

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

V Plzni dne 16. května 2024

Bc. Petr Štěpánek

Abstract

The aim of this master thesis is to extend and improve the detection mechanism of the Software Process Anti-Pattern Detector (SPADe) toolkit, which is used to analyse project data to identify anti-patterns and bad practices. To achieve these goals, a detection microservice was designed, implemented and integrated to allow modular definition of models of indicators, metrics and parameters and their combination into more complex structures such as anti-pattern models. Within the newly proposed solution, all existing detections were transferred and validated, while a set of new detections was selected, designed and implemented.

Abstrakt

Cílem této diplomové práce je rozšířit a zdokonalit detekční mechanismus nástrojové sady Software Process Anti-Pattern Detector (SPADe), která slouží k analýze projektových dat za účelem identifikace anti-vzorů a špatných praktik. K dosažení těchto cílů byla navržena, implementována a integrována detekční mikroslužba, která umožňuje modulární definici modelů indikátorů, metrik a parametrů a jejich kombinování do složitějších struktur, jako jsou modely anti-vzorů. V rámci nově navrženého řešení byly převedeny a ověřeny všechny stávající detekce a zároveň byla vybrána, navržena a implementována sada nových detekcí.

Poděkování

Rád bych poděkoval vedoucímu této diplomové práce Ing. Petru Píchovi za cenné rady a veškerý čas, který mi v průběhu celé práce věnoval.

Obsah

1 Úvod

V rámci řízení softwarových projektů často dochází k opakování určitých chyb, které mohou mít fatální dopady na úspěšné dokončení projektů. Tyto chyby, běžně označované jako anti-vzory, mohou být zvláště problematické, jelikož jejich důsledky se často projevují až ve chvíli, kdy je již příliš pozdě na jejich efektivní řešení. Projekt Software Process Anti-Pattern Detector (SPADe), realizovaný na Fakultě aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU), se zaměřuje na včasné odhalování těchto anti-vzorů. V rámci tohoto projektu byla vyvinuta stejnojmenná sada nástrojů, která tvoří komplexní ekosystém určený k extrakci potřebných dat z Application Lifecycle Management (ALM) nástrojů a jejich analýze na přítomnost anti-vzorů, jejichž příznaky jsou definovány v rámci katalogu, který je součástí nástrojové sady. Systém SPADe je aktuálně ve vývojové, experimentální fázi a slouží pro vědecké účely, avšak vykazuje značný potenciál pro širší praktické využití, například jako pomocník projektového manažera.

Přestože nástroj umožňuje provádět detekci anti-vzorů na množině získaných projektových dat, jeho implementace není v některých aspektech ideální. Cílem této práce je proto zdokonalit detekční část systému SPADe tak, aby byla schopna podporovat modularitu, umožňovala jednoduchou definici metrik, prahových hodnot a indikátorů a jejich kompozici do složitějších struktur, jako jsou například anti-vzory. Tímto způsobem bude aplikace flexibilnější a umožní snadné vytváření anti-vzorů pomocí znovupoužitelných komponent. Součástí této práce bude také převod existujících detekcí do nově navržené části, aby byla zachována stávající funkcionalita, a implementace nových detekcí, které ověří a rozšíří detekční schopnosti systému.

Tato diplomová práce je strukturována do osmi hlavních kapitol. Úvodní kapitola se zabývá projektovým a procesním řízením v rámci softwarových projektů a nástroji používanými k těmto účelům (kapitola 2). Dále je rozebrána problematika získání a zpracování projektových dat z těchto nástrojů (kapitola 3) a klíčové koncepty použité pro popis a detekci anti-vzorů (kapitola 4). Následně je proveden návrh architektury nového řešení (kapitola 5), jeho implementace (kapitola 6) a integrace do nástrojové sady SPADe (kapitola 7). V další kapitole jsou přidány jak původní, tak nové detekce (kapitola 8) a poslední kapitola předkládá náměty pro další rozvoj implementované a integrované detekční části (kapitola 9).

2 Proces a řízení vývoje softwaru

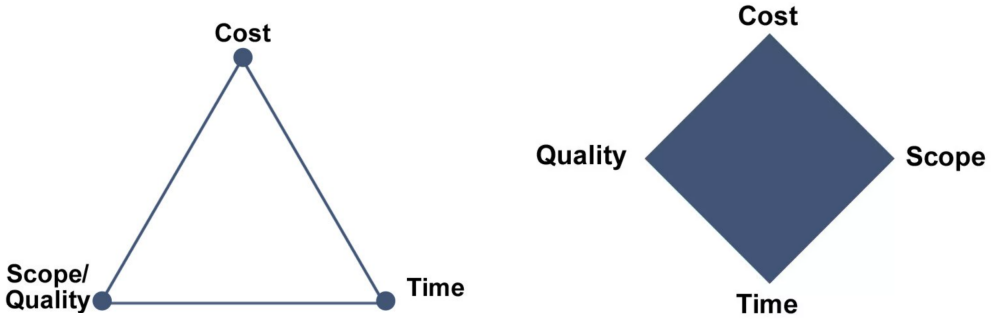
Tato kapitola se zaměřuje na klíčové aspekty procesu a řízení projektu vývoje softwaru, které jsou zásadní pro jeho úspěšné dokončení. Procesy a metody řízení vývoje softwaru formují způsob, jakým jsou projekty plánovány, realizovány a hodnoceny, přičemž zahrnují široké spektrum aktivit od definování požadavků, přes návrh, vývoj a testování, až po nasazení a údržbu finálního produktu.

V následujících podkapitolách budou detailně definovány a rozlišeny termíny projektového a procesního řízení. Zároveň bude představena role a zodpovědnosti klíčové osoby projektu, tedy projektového manažera. Budou také rozebrány často se vyskytující praktiky a metodiky, které se v praxi využívají k efektivnímu vývoji softwarových řešeních. Závěr této kapitoly bude věnován podpůrným nástrojům, které usnadní procesy v rámci projektu a všech jeho částí.

2.1 Projektové a procesní řízení

Projektové řízení představuje základní disciplínu, která je zcela nezbytná pro úspěšné dokončení softwarového projektu. Projekt samotný může být definován jako dočasné úsilí o vytvoření jedinečného produktu, služby nebo výsledku, přičemž může být samostatný nebo může být součástí programu či portfolia [1]. Projektové řízení pak může být chápáno jakožto aplikace znalostí, dovedností, nástrojů a technik za účelem splnění požadavků projektu [1]. Hlavním cílem je dosáhnout stanovených projektových cílů v určeném časovém rámci a rozpočtu, přičemž splňuje nebo dokonce překračuje očekávané požadavky a standardy kvality. Tyto omezující podmínky mohou být vizualizovány pomocí tzv. projektového trojúhelníku, známého také jako projektový trojimperativ či trojité omezení, který ukazuje vzájemné propojení mezi časem, náklady a rozsahem (či kvalitou). Pokud projekt nesplní kterýkoliv z těchto faktorů, může být označen za neúspěšný, i když splní ostatní dva [10]. Někdy je možné začlenění kvality jako čtvrtého klíčového parametru. Tím vzniká tzv. projektový čtverec či projektový diamant [7]. Koncepty projektového trojúhelníku i čtverce jsou ilustrovány na obrázku 2.1.

Na rozdíl od projektového řízení, procesní řízení se zaměřuje na neustálé



Obrázek 2.1: Projektový trojúhelník a projektový čtverec [3]

zlepšování a optimalizaci stávajících procesů v organizaci, které vycházejí z jasně definované strategie [20]. Jeho hlavním cílem je dosáhnout vyšší efektivity, produktivity a kvality výsledků prostřednictvím běžných operací a rutinních činností. Tento přístup zdůrazňuje význam standardizace, dokumentace a měření procesů, aby bylo možné identifikovat oblasti pro zlepšení. V rámci softwarových procesů se tento koncept často objevuje pod pojmem Software Process Improvement (SPI), což je metodika navržená k plánování a provádění aktivit s cílem dosáhnout specifických výsledků, kterými mohou být zvýšení rychlosti vývoje nebo snížení nákladů [28].

Klíčový rozdíl mezi projektovým a procesním řízením spočívá v jejich cílech a trvání. Zatímco projektové řízení, ve smyslu řízení jednoho určitého projektu, je definováno jasným cílem a končí po jeho dosažení, procesní řízení představuje nekonečný cyklus, který hledá cesty pro neustálé zlepšování pracovních postupů a rutinních činností.

Integrace obou přístupů do praxe organizace přináší synergie, které zvyšují efektivitu a účinnost při dosahování jak krátkodobých, tak dlouhodobých cílů organizace. Efektivní projektové řízení zajišťuje, že projekty jsou dokončeny včas a v rámci rozpočtu s požadovanou úrovní kvality. Současně procesní řízení optimalizuje každodenní operace, čímž přispívá k dlouhodobému úspěchu organizace.

Ve vývoji softwaru jsou tyto přístupy zásadní pro řízení komplexních projektů a neustálé zlepšování procesů. Projektové řízení v tomto kontextu vyžaduje specifické znalosti softwarového inženýrství, včetně běžně používaných metodik vývoje softwaru, které definují fáze vývoje, role v týmu, metody testování apod. Procesní řízení ve vývoji softwaru se pak zaměřuje na optimalizaci těchto metodik a procesů, aby byla zajištěna nejen efektivita a produktivita, ale i adaptabilita a schopnost rychle reagovat na změny požadavků zákazníka nebo trhu. Využití obou přístupů umožňuje týmům úspěšně realizovat jednotlivé projekty a adaptovat se na nové výzvy ve velmi

dynamickém prostředí vývoje softwaru.

2.2 Role projektového manažera

Projektový manažer je klíčovou osobou zodpovědnou za dosažení cílů projektu v rámci daných omezení. Tato role vyžaduje kombinaci technických dovedností a schopnosti efektivně řídit změny, komunikovat, prezentovat a poskytovat zpětnou vazbu. Efektivní projektový manažer plánuje, dohlíží na průběh projektu a koordinuje práci týmu k zajištění optimálních výsledků. Adaptabilita, schopnost prioritizace a efektivní komunikace s různými zainteresovanými stranami jsou klíčové dovednosti každého úspěšného projektového manažera. Důležitou součástí jeho role je také podpora týmové spolupráce a využívání zkušeností z minulých projektů pro zlepšení budoucích praxí, což je uznáváno v průvodci PMBOK [1], standardu pro projektové řízení.

2.3 Praktiky a vzory

V rámci projektového a procesního řízení se často opakují určité aktivity, metody a přístupy k řešení problémů a výzev, které mohou pramenit jak ze záměrného využití osvědčených postupů na základě předchozích úspěšných zkušeností, tak i nezáměrně, z tradice či zvyklosti. Tyto metody, známé jako dobré praktiky, jsou ve správně definovaném kontextu považovány za efektivnější a účinnější než jiné metody [17]. Praktiky v projektovém a procesním řízení tedy představují soubor osvědčených metod, postupů a akcí, které usnadňují efektivní dosahování cílů a řešení problémů, a jsou založeny jak na obecně známých principech, tak i na individuálních zkušenostech projektových manažerů [31]. U dobrých praktik je často požadováno jejich cílené zapojení do projektů pro jejich přínos a následné benefitování. Příkladem dobré praktiky může být pravidelné pořádání stand-up schůzek v rámci týmu.

Opačným případem dobrých praktik jsou špatné (neefektivní) praktiky, u kterých je snaha je z projektu vyloučit kvůli jejich potenciálně škodlivým dopadům na projekt. Špatné praktiky jsou definovány jako metody nebo postupy, které se historicky ukázaly jako neefektivní nebo vedoucí k nechtěným výsledkům, a proto jsou považovány za nežádoucí v řízení projektů a procesů [31]. Jako příklad špatné praktiky lze uvést například podceňování správného testování a inspekcí kvality [5].

Specifickou podkategorií praktik jsou vzory (patterns), které představují

osvědčené řešení opakujících se problémů v konkrétním kontextu. Lze je definovat jako praxí ověřené řešení problému, opakovaně použitelné v daném kontextu [30]. Jedná se tedy o formalizovaná řešení pro běžné problémy nebo výzvy s cílem zvýšit efektivitu a účinnost projektového a procesního řízení. Vzory nejsou přísně určené k nepřetržitému dodržování bez úprav, ale spíše k adaptaci a využití jejich klíčových principů pro zlepšení řízení a dosahování cílů. Příkladem může být vzor s názvem Size the Schedule, který pojednává o vytvoření realistického časového plánu projektu a jeho částí [12]. V kontextu softwarového vývoje se také můžeme setkat s takzvanými návrhovými vzory. Jedná se o doporučené postupy řešení běžně se vyskytujících úloh [23]. Příkladem návrhového vzoru může být Singleton, tedy způsob vytvoření třídy, která bude mít pouze jednu instanci.

Na druhé straně, anti-vzory (anti-patterns) jsou přístupy, které se zdají být řešením problému, ale ve skutečnosti vedou k dalším komplikacím nebo jsou neúčinné [31]. Anti-vzory jsou definovány jako běžné, často se vyskytující, ale škodlivé praktiky a řešení, které se mohou zdát jako účinné, ale v praxi jsou neefektivní nebo kontraproduktivní v určitém kontextu [11]. Jejich rozpoznání a odstranění je klíčové pro úspěšné řízení projektů, přestože příznaky anti-vzorů se mohou projevit až později, což komplikuje jejich včasné identifikování a řešení. Příkladem anti-vzoru v projektovém řízení může být Business As Usual [8]. Tento anti-vzor je charakterizován tím, že tým setrvává u zavedených postupů a rutin, které mu v minulosti zajišťovaly úspěch. Přitom však zanedbává význam pravidelných retrospektiv a nutnosti být adaptabilní a flexibilní. S anti-vzory se můžeme setkat také přímo u softwarového vývoje, kde typickým příkladem může být tzv. Špagetový kód, tedy programový kód, který je nestrukturovaný a z toho důvodu také špatně čitelný.

2.4 Metodiky projektového řízení

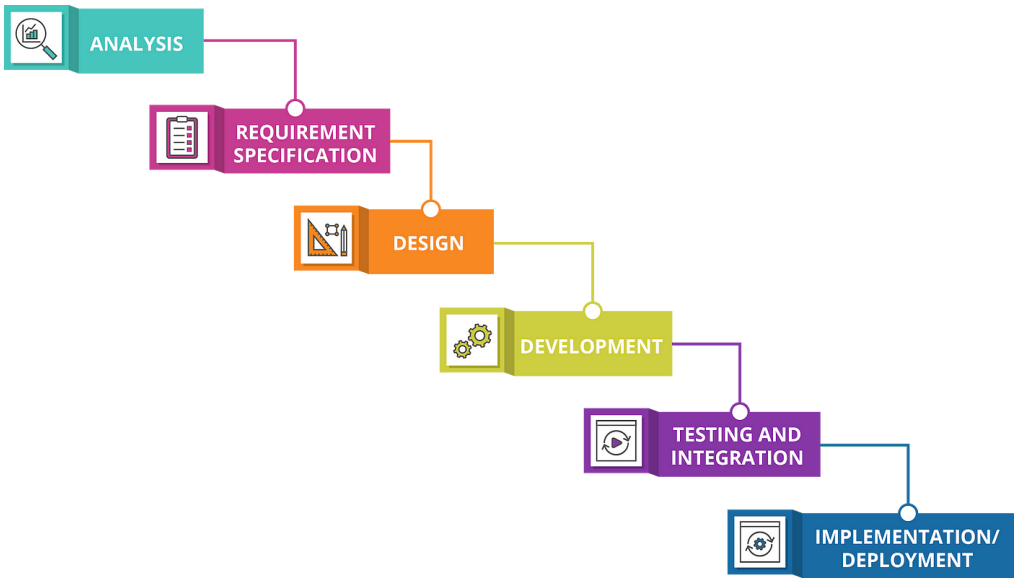
Na rozdíl od praktik, které se zaměřují na přístupy k řešení specifických problémů, metodiky představují komplexní soubory praktik a dalších elementů, které se využívají v rámci celého životního cyklu projektu. Metodiky v kontextu řízení projektů, zejména těch softwarových, lze definovat jako soubory pravidel, postupů, technik a nástrojů, které se používají k plánování, vývoji, implementaci a testování v rámci projektů, tedy lze říci, že pokrývají celý životní cyklus vývoje softwaru [30]. Tyto metodiky poskytují rámec, který projektovým týmům umožňuje systematicky dosahovat cílů projektu, řídit rozsah práce, času a nákladů a zároveň zajišťovat kvalitu výsledného

výstupu.

V rámci softwarových projektů mimo jiné existují skupiny metodik sekvenčních, iterativních a agilních, které budou nyní krátce popsány.

Sekvenční metodiky

Sekvenční metodiky charakterizuje předem definovaná sada fází, které projekt musí postupně projít od začátku do konce. Tyto fáze zahrnují analýzu, plánování, návrh, implementaci, testování, nasazení a údržbu [18]. Každá fáze se skládá z určitého souboru činností a výstupů, které musí být dokončeny před zahájením následující fáze [6], což omezuje možnost zpětné vazby a úprav v pozdějších fázích projektu. Sekvenční metodiky jsou vhodné pro projekty s jasně definovanými požadavky, u kterých se předpokládá nízká míra změn. Příkladem sekvenční metodiky je tradiční Vodopádový model, jehož fáze jsou znázorněny na obrázku 2.2.



Obrázek 2.2: Fáze Vodopádového modelu [18]

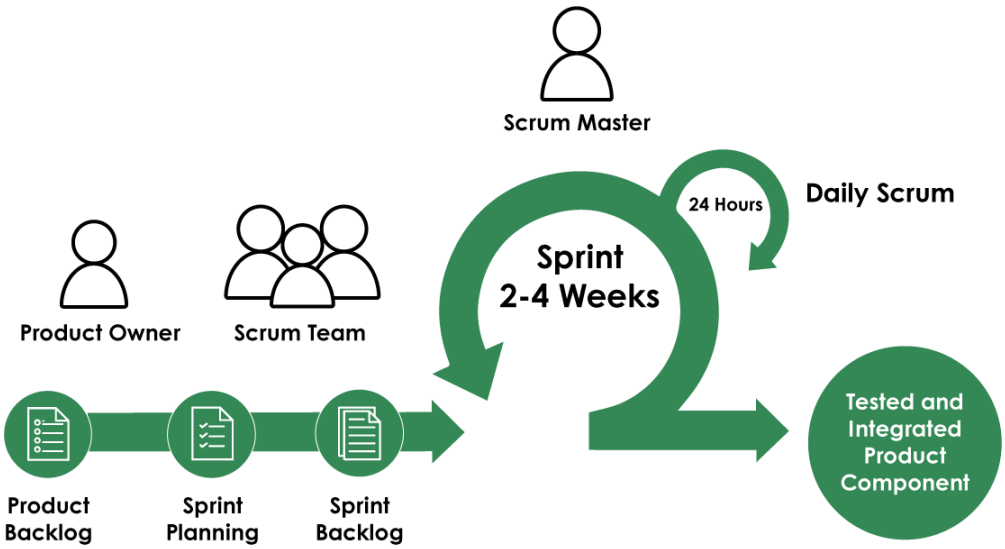
Iterativní metodiky

Iterativní metodiky se na rozdíl od sekvenčních vyznačují cyklickým vývojem, kde produkt prochází sérií iterací, během nichž se postupně vyvíjí, testuje a validuje u zákazníka [27]. Tyto iterace umožňují projektovému týmu pružně reagovat na případné změny požadavků a zpětnou vazbu od zákazníků nebo uživatelů, což vede k lepšímu přizpůsobení konečného produktu potřebám. Iterativní metodiky jsou vhodné pro projekty, kde jsou požadavky

v počáteční fázi nejasné a očekává se jejich vývoj v průběhu projektu. Dále jsou často přítomny u komplikovaných a implementačně náročných projektů. Příkladem iterativní metodiky je standardizovaný Rational Unified Process (RUP).

Agilní metodiky

Speciálním případem iterativních metodik jsou agilní metodiky, které kladou důraz na rychlé a pružné řešení změn, intenzivní spolupráci s klientem a časté vydávání funkčních verzí produktu. Agilní přístupy se vyznačují krátkými vývojovými cykly (tzv. sprinty), během kterých týmy pracují na malých částech celkového projektu, což usnadňuje rychlou adaptaci na změny. Agilní metodiky jsou ideální pro projekty v dynamickém a nejistém prostředí, kde jsou rychlost, flexibilita a zákaznická spokojenost klíčové. Jedná se o odlehčené procesy, které umožňují efektivnější práci a snadnější řízení změn [6]. Příkladem agilních metodik může být tzv. Scrum, jehož iterace je znázorněna včetně příslušných rolí na obrázku 2.3.



Obrázek 2.3: Iterace v rámci metodiky Scrum [4]

2.5 Nástroje pro řízení projektů

V současném dynamickém prostředí vývoje softwaru hrají nástroje pro Application Lifecycle Management (ALM) klíčovou roli v zajištění efektivity a úspěšnosti projektů. Tyto nástroje pokrývají celý životní cyklus aplikací, od konceptu až po uvedení na trh a následnou údržbu. Umožňují týmům efektivně spolupracovat, spravovat požadavky, sledovat změny, automatizovat sestavení a deployment nebo udržovat kvalitu kódu [31]. V této podkapitole jsou vybrány klíčové kategorie ALM nástrojů, které podporují tyto procesy.

Nástroje pro řízení změn

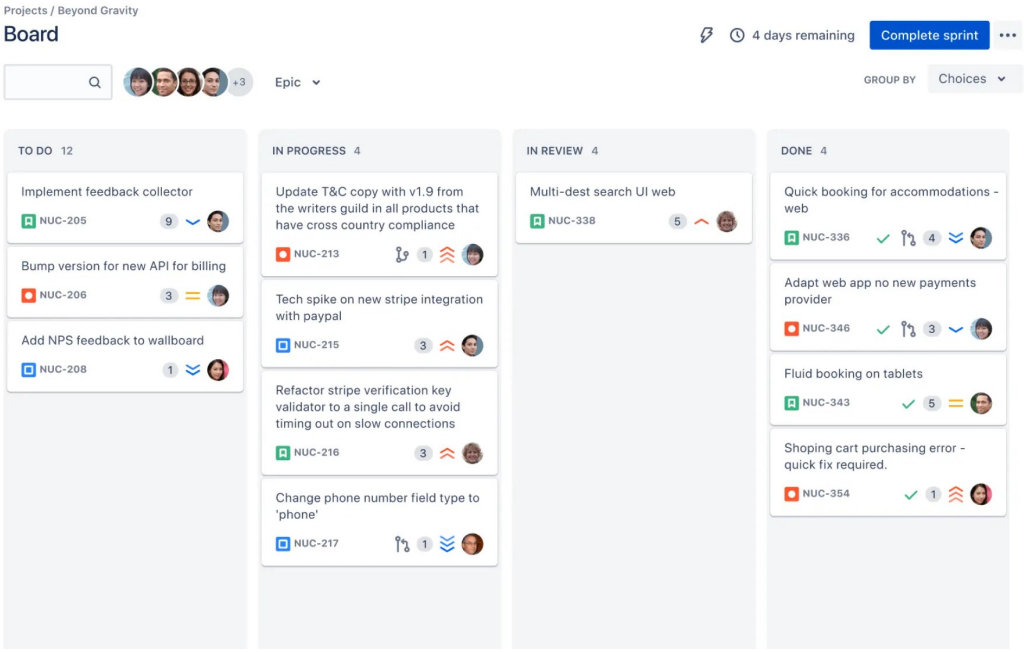
Tyto nástroje, známé také jako bug-tracking nebo issue-tracking systémy, umožňují efektivní správu a sledování požadavků na změny, opravy chyb a další úkoly související s projektem a s vyvíjeným produktem. Umožňují týmům zachytávat, prioritizovat a sledovat postup jednotlivých úloh. Zároveň jsou veškeré úkoly centralizované na jednom místě, což zajistí přehlednost a také snadnou dohledatelnost. Kromě aktuálních informací umožňují udržovat také historii všech změn v rámci obsažených úkolů. Mezi často používané nástroje pro správu úloh patří například Atlassian Jira[1], Redmine[2] či Mozilla Bugzilla[3]. Na přiloženém obrázku 2.4 je znázorněný Scrum board v rámci nástroje Atlassian Jira.

Nástroje pro správu verzí

Systémy pro správu verzí jsou nezbytným nástrojem pro efektivní řízení zdrojového kódu v rámci softwarového vývoje. Tyto systémy poskytují vývojářům možnost pečlivě sledovat a spravovat změny provedené v kódu, což zároveň usnadňuje kooperaci mezi členy týmu a umožňuje reverzi k dřívějším verzím kódu, kdykoli je to nezbytné. Tato funkcionalita je kritická pro koordinaci práce na projektech, kde na kódu pracuje více lidí, protože zajišťuje, že všechny změny jsou řádně dokumentovány a přístupné všem zúčastněným.

Kromě základních funkcí jako je sledování změn, podpora týmové spolupráce a možnost reverze, moderní nástroje pro správu verzí často integrují další pokročilé funkce. Mezi ty patří větvení a slučování kódu, což umožňuje týmům pracovat paralelně na různých funkcionalitách bez rizika vzájemného ovlivnění práce. Dále tyto systémy mohou nabízet integrované nástroje pro

[1] https://www.atlassian.com/software/jira
[2] https://www.redmine.org
[3] https://bugzilla.mozilla.org



Obrázek 2.4: Scrum board v nástroji Attlasian Jira [2]

revizi kódu, což umožňuje systematické přezkoumání změn kódem ostatními vývojáři před jejich začleněním do hlavního projektu. Tím se zvyšuje kvalita kódu a snižuje pravděpodobnost chyb.

Nejčastěji se v praxi využívá nástroj Git, zejména v úložištích jako GitHub[4] a GitLab[5]. Dalším známým systémem je pak Apache Subversion[6].

Nástroje pro znalostní management

V rámci vývojového týmu hraje klíčovou roli efektivní šíření znalostí a informací mezi členy týmu. Je nezbytné, aby se know-how každého jednotlivce systematicky sdílelo s ostatními, což umožňuje optimální využití těchto znalostí pro společné cíle. Když jsou informace volně dostupné všem členům týmu, umožňuje to rychlejší řešení problémů a efektivnější spolupráci. Toto platí zejména v kontextu softwarového vývoje, na kterém pracuje tým různě zaměřených lidí, přičemž každý má specifické znalosti o určité části vyvíjeného produktu nebo o určité technologii. Příkladem nástrojů pro sdílení znalostí a informací může být Attlasian Confluence[7] nebo Microsoft One-

[4]https://github.com
[5]https://gitlab.com
[6]https://subversion.apache.org/
[7]https://www.atlassian.com/software/confluence

Note[8].

Nástroje pro komunikaci

Základem spolupráce členů týmu je bezesporu správná komunikace. Nástroje pro komunikaci umožňují týmům udržovat spojení bez ohledu na fyzickou vzdálenost. Tyto nástroje podporují různé formy komunikace, od textových zpráv a emailů, po hovory a videokonference. Zajišťují tak, že komunikace mezi členy týmu je hladká a efektivní. Častou funkcí je například také integrace osobních kalendářů, což umožní přehled o aktivitách a schůzkách jednotlivých členů týmu. Mezi často využívané nástroje pro komunikaci může patřit například Microsoft Teams[9], Microsoft Outlook[10] či Slack[11].

Další nástroje

Mimo výše uvedené kategorie existuje i řada dalších nástrojů, které podporují specifické aspekty vývoje softwaru. Může se jednat o nástroje jako jsou systémy pro automatickou integraci změn a nasazení produktu (například Jenkins[12]) nebo cloudová úložiště dokumentů (například Microsoft OneDrive[13]). Řada nástrojů také neslouží pouze pro jeden účel, ale integruje více funkcionalit dohromady a nabízí tak multifukční platformu. Příkladem může být již dříve zmíněný nástroj GitHub, který kromě funkce kolaborativní spolupráce a správy zdrojového kódu umožňuje také správu změn a požadavků nebo průběžnou integraci a nasazení produktu.

[8] https://www.onenote.com
[9] https://www.microsoft.com/cs-cz/microsoft-teams
[10] https://www.microsoft.com/cs-cz/microsoft-365/outlook/outlook-for-business
[11] https://slack.com
[12] https://www.jenkins.io
[13] https://www.microsoft.com/cs-cz/microsoft-365/onedrive/onedrive-for-business

3 Zpracování projektových dat

Zpracování projektových dat, včetně těch pocházejících z ALM nástrojů, je pro řízení a úspěšné dokončení softwarových projektů klíčové. Tato data představují bohatý zdroj cenných informací, které, když jsou správně zpracovány a využity, mohou firmám poskytnout konkurenční výhodu. Efektivní zpracování dat umožňuje získat přesnější přehledy o průběhu a výkonnosti projektů, což napomáhá k lepšímu rozhodování a optimalizaci procesů. Různé metody vizualizace těchto dat jsou pak cenným nástrojem pro projektové manažery v řízení projektů.

V následujících podkapitolách bude podrobněji popsáno, jak se data získávají, analyzují a využívají k posílení výsledků projektů. Také bude představen nástroj Software Process Anti-Pattern Detector (SPADe), který je specializovaný na detekci anti-vzorů a špatných praktik v projektových datech.

3.1 Získání projektových dat

V rámci efektivního řízení projektů hrají Application Lifecycle Management nástroje, popsané v předešlé kapitole, klíčovou roli ve správě a uchovávání projektových dat. Tyto nástroje shromažďují a uchovávají data po celou dobu trvání projektu, od jeho zahájení až po úspěšné dokončení. Přístupnost těchto dat se liší v závislosti na typu projektu. Otevřené projekty (open-source) obvykle umožňují širokou přístupnost dat veřejnosti. To se týká zejména softwaru s kódem uloženým ve veřejném repozitáři. Na druhou stranu projekty realizované v rámci organizací nebo školních prací mohou mít přístup omezen pouze pro členy týmu nebo zaměstnance firmy, a to z důvodu ochrany zdrojového kódu a citlivých dat. V tomto případě se jedná o takzvaný uzavřený (closed-source) projekt.

Shromažďování projektových dat z podpůrných nástrojů může probíhat různými způsoby. Některé z těchto nástrojů poskytují různorodé pohledy na data a nabízejí předpřipravené integrace s dalšími systémy, což umožňuje agregaci dat z více zdrojů. Je však třeba poznamenat, že tyto funkce nejsou standardem ve všech nástrojích a kde jsou dostupné, mohou se metody jejich implementace lišit.

Dalším ze způsobu získávání dat z ALM nástrojů je využití Application Programming Interface (API) daného nástroje, pokud je tato možnost dostupná. Další metoda spočívá v exportu dat do běžně užívaného formátu, jako je Extensible Markup Language (XML), JavaScript Object Notation (JSON) a další, a jejich následném strojovém zpracování. Avšak ne všechny nástroje nabízejí možnost exportu dat. Možnou metodou je také přímý přístup do databáze, ve kterém jsou data pro daný ALM nástroj uložena. Poslední, poněkud náročnější, ale stále proveditelný způsob představuje crawling, tedy parsování jednotlivých obrazovek nástroje a extrakce dat na nich zobrazených [25].

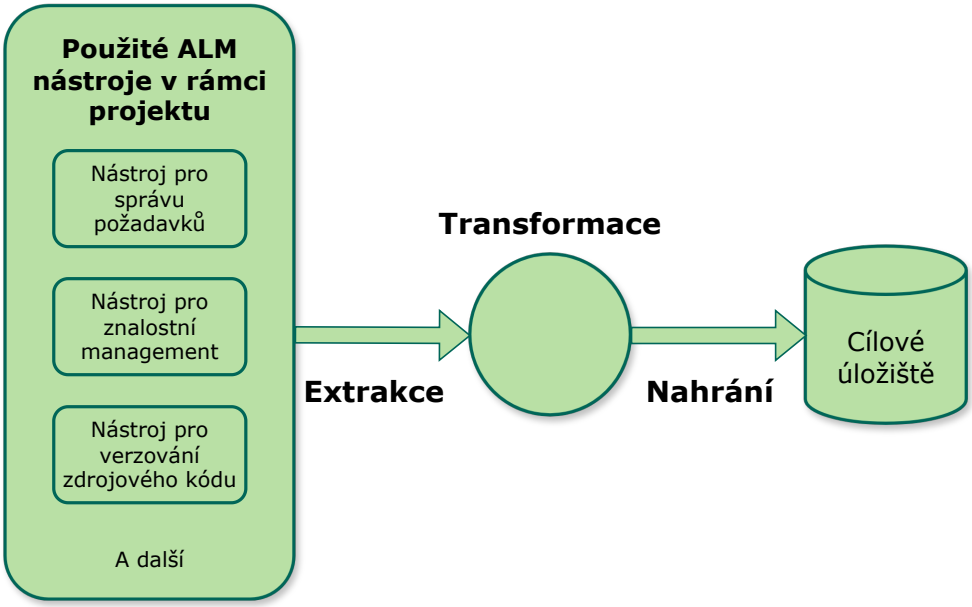
Při využití kteréhokoliv způsobu, zejména pak použití API se vynořuje potřeba vytvoření takzvané datové pumpy, která umožní efektivní extrakci, transformaci a nahrání (ETL) dat. Vytvoření datové pumpy přináší tradiční výzvy spojené s ETL procesy, jakými jsou například zachování integrity dat či optimalizace výkonu. Kromě samotné extrakce dat je třeba řešit, jak bude vypadat cílová destinace pro extrahovaná data - zda to bude databáze, datový sklad, nebo jiné úložiště. Je také nezbytné zvážit, jak budou data zpracována do vhodného formátu pro jejich uložení v této nové destinaci.

Projekty často využívají více ALM nástrojů, což znamená, že pro získání úplného přehledu o projektu je nezbytné extrahovat data ze všech použitých nástrojů. To vyžaduje integraci dat do jednotného, unifikovaného modelu, což může být výzvou vzhledem k rozdílným formátům a strukturám dat, které jednotlivé nástroje poskytují. Efektivní řešení této problematiky zahrnuje detailní plánování a výběr vhodných nástrojů a technologií, které podporují integraci a transformaci dat do konzistentního a analyzovatelného formátu. Proces získání projektových dat z ALM nástrojů je znázorněn na obrázku 3.1.

3.2 Analýzy projektových dat a jejich využití

Analýza projektových dat představuje klíčový nástroj pro zvyšování efektivity, konkurenceschopnosti a úspěšnosti projektů v dnešní datově orientované době. Přestože analýza dat může na první pohled působit jako nákladná a časově náročná aktivita, která by mohla být nahrazena jinou, zdánlivě důležitější, její hodnota a přínosy nemohou být přehlíženy.

Data obsahují nejen zjevné informace, ale i množství skrytých poznatků, které, jsou-li správně interpretovány, mohou přinést zásadní vhledy pro zlepšení procesů, efektivitu a inovace. Využití dat k získání těchto informací je možné jak na konci projektu, kdy jsou data kompletní (v tomto kontextu se



Obrázek 3.1: Proces získání dat z ALM nástrojů

jedná o tzv. post-mortem analýzu [21]), tak i v průběhu jeho realizace, což umožňuje téměř real-time reakci na vznikající situace a trendy.

Po extrakci dat se nabízí široká škála metod pro jejich analýzu, od základní statistické analýzy po pokročilé techniky strojového učení, v závislosti na charakteru a struktuře dat. Důležitým aspektem analýzy jsou metriky, které slouží jako kvantitativní ukazatele různých aspektů projektu. Tyto metriky mohou zahrnovat širokou škálu dimenzí, od výkonnosti a produktivity po kvalitu a spokojenost zákazníků.

Kromě metrik můžeme v datech hledat také opakující se vzory či anti-vzory, které mohou být specifické pro daný projekt nebo se vyskytovat obecně napříč různými projekty. Identifikace těchto vzorů umožňuje odhalit osvědčené postupy vedoucí k úspěchu, stejně jako varovné signály, které signalizují možné problémy. Zatímco pozitivní vzory (ty, které přinášejí benefit projektu) jsou žádoucí a jejich identifikace a implementace mohou projekt posílit, negativní vzory, tedy anti-vzory (ty, které projektu přinášejí zátěž) jsou varováním, které ukazuje na neefektivitu, ztrátu kontroly, času, či finančních prostředků. Tyto nežádoucí postupy je třeba co nejdříve identifikovat a z projektu eliminovat.

Výsledkem analýzy projektových dat by tedy nemělo být pouze získání hlubšího porozumění pro aktuální stav projektu, ale také odhalení příležitostí pro zlepšení a inovace. Tím se z analýzy dat stává investice, která může významně přispět k úspěchu projektu, optimalizaci zdrojů a dosažení lepších

výsledků. Analýza dat tedy představuje cenný nástroj pro řízení projektů, který umožňuje organizacím pracovat chytřeji, reagovat pružněji na změny a udržet si konkurenceschopnost v rychle se měnícím prostředí.

3.3 Nástrojová sada SPADe

Vývoj nástroje Software Process Anti-Pattern Detector (SPADe) na Katedře informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) Západočeské univerzity v Plzni (ZČU) představuje zásadní krok vpřed ve zkoumání projektových dat. Tento nástroj, který je momentálně ve fázi vývoje experimentální verze, se již používá pro vědecké účely, avšak vykazuje výrazný potenciál pro uplatnění v praxi projektových manažerů.

SPADe je tvořený sadou nástrojů pro definici a detekci anti-vzorů a špatných praktik v procesním a projektovém řízení softwarových projektů prostřednictvím analýzy projektových dat. Tento nástroj je schopen nejen analyzovat data, ale i je efektivně sbírat, a to z různých ALM nástrojů pomocí datových pump a procesu ETL s uložením do unifikovaného datového úložiště. Integrace dat z různých zdrojů, tedy z vícero využitých ALM nástrojů, přináší realističtější přehled o projektu a zvyšuje přesnost výsledků analýz.

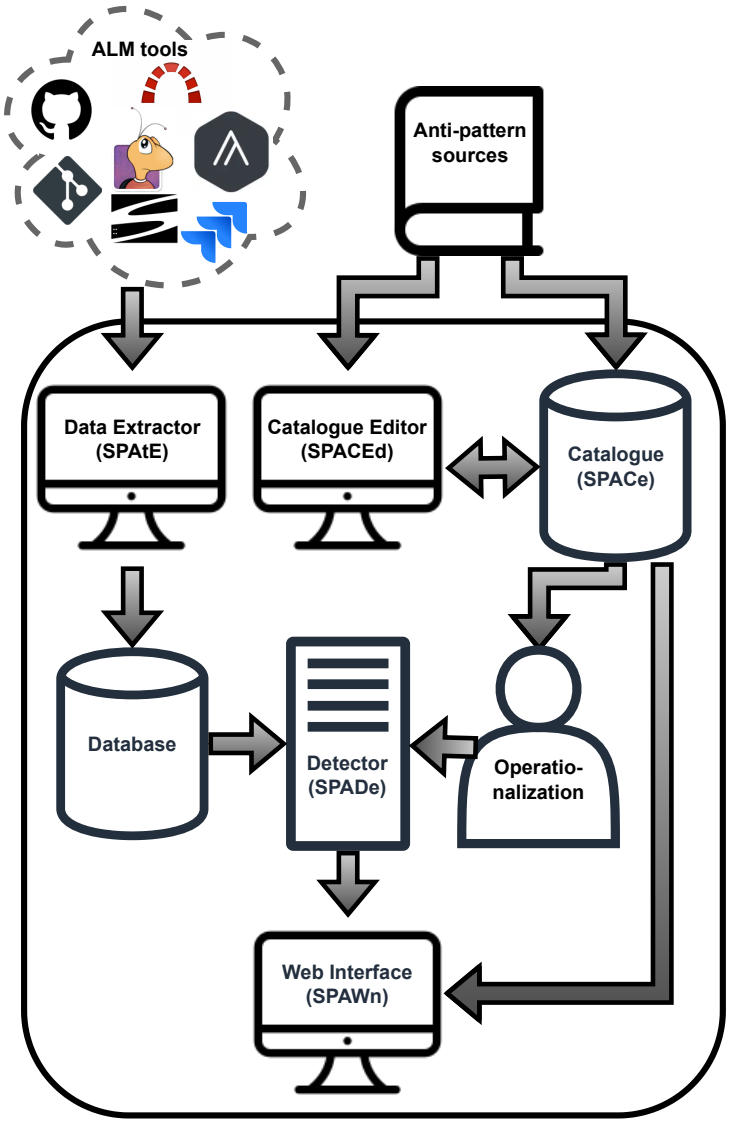
Vize projektu SPADe je vytvoření robustního nástroje, který by byl schopný projektového manažera pravidelně upozorňovat na potenciálně škodlivé vzory nebo znaky v projektových datech. Tyto informace by mohly být, v ideálním případě, předány manažerovi dříve, než se negativní dopady těchto problémů skutečně projeví, čímž by se minimalizovala možná škoda, jelikož by bylo umožněno včas přijmout nápravná či preventivní opatření.

Nástroj by měl být užitečný nejen během trvání projektu, ale i po jeho dokončení, a to prostřednictvím již zmíněné post-mortem analýzy. Taková analýza umožní manažerům a celému týmu provést retrospektivní hodnocení projektu, identifikovat příčiny neúspěchu a rozpoznat možnosti pro zlepšení v budoucích projektech. Tímto způsobem se získává cenné know-how, které je klíčové pro úspěch budoucích iniciativ.

3.3.1 Podoba současného řešení

Nástrojová sada SPADe je tvořena několika synergicky propojenými částmi, které zajišťují komplexní proces od sběru projektových dat, až po zobrazení výsledků analýzy uživateli, tedy projektovému manažerovi. Celý systém a všechny jeho komponenty jsou pro přehlednost znázorněny na obrázku 3.2.

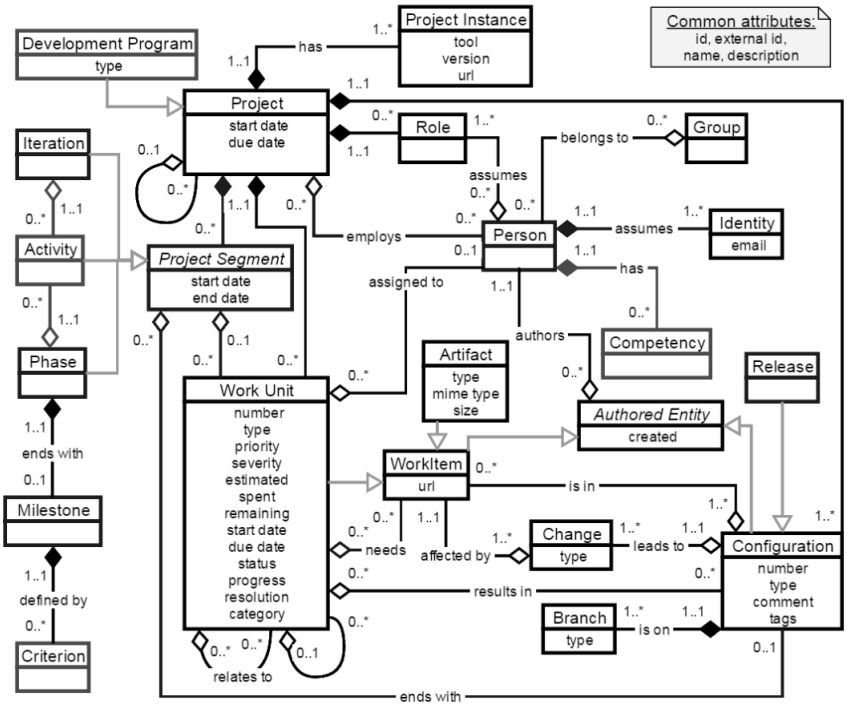
Celý proces začíná fází získávání dat, kterou zajišťují datové pumpy (Data Extractor). Tyto pumpy využívají různých rozhraní ALM nástrojů. V



Obrázek 3.2: Nástrojová sada SPADe a její komponenty [31]

současné době SPADe implementuje datové pumpy pro sedm nástrojů: Assembla[1], Mozilla Bugzilla, Git, GitHub, Atlassian Jira, Redmine a Apache Subversion [26].

Natěžená data jsou uložena do datového úložiště (Database) založeném na unifikovaném datovém modelu. Jedná se o unikátní datový model, který byl pečlivě navržený tak, aby do něj bylo možné ukládat data z různých ALM nástrojů [25]. Je tedy dostatečně obecný, nicméně umožňuje uchování všech možných detailů a vztahů mezi různými entitami. Přítomnost tohoto unifikovaného úložiště významně usnadní další práci s daty. Úložiště je momentálně realizováno jako relační databáze, nicméně vizí projektu je jeho nahrazení nebo doplnění o datový sklad pro rozsáhlejší analýzy. Doménový model databáze SPADe je uveden na obrázku 3.3.



Obrázek 3.3: Doménový model databáze SPADe [25]

Další částí SPADe je unikátní katalog anti-vzorů (Catalogue) [8]. Ten obsahuje soubor anti-vzorů nashromážděných z různých zdrojů (Anti-pattern sources), uložených v jednotně formátované textové struktuře. Vytvoření katalogu znamená tvorbu centralizovaného úložiště, které adresuje problém roztříštěnosti anti-vzorů a jejich popisů napříč literaturou a internetem.

[1]https://get.assembla.com

V rámci katalogu je také vytvořena aplikace pro jejich snadnou správu (Catalogue Editor).

Samotná detekce anti-vzorů v projektových datech je zajišťována aplikací s názvem Detector (SPADe), což je jádro celé nástrojové sady. Aplikace je implementovaná v Javě s využitím frameworku Spring. Tato aplikace aktuálně podporuje detekci deseti anti-vzorů a umožňuje nastavení prahových hodnot pro detekce. Průběh detekce je dán operacionalizací (Operationalization), tedy překladem textového popisu projevů anti-vzorů do metrik měřitelných nad daty z ALM nástrojů. Jejím výsledkem je pak případné upozornění na opodstatněné podezření výskytu anti-vzorů v analyzovaných datech.

Jako uživatelské rozhraní poté slouží webová aplikace s názvem Web Interface (SPAWn). Jedná se o prostředníka, pomocí kterého může uživatel zadávat požadavky na provedení detekcí spolu s nastavením jednotlivých prahových hodnot. Zároveň jsou mu skrz toto rozhraní zobrazeny komplexní výsledky provedené analýzy na přítomnost anti-vzorů v projektových datech.

3.3.2 Implementace detekční části

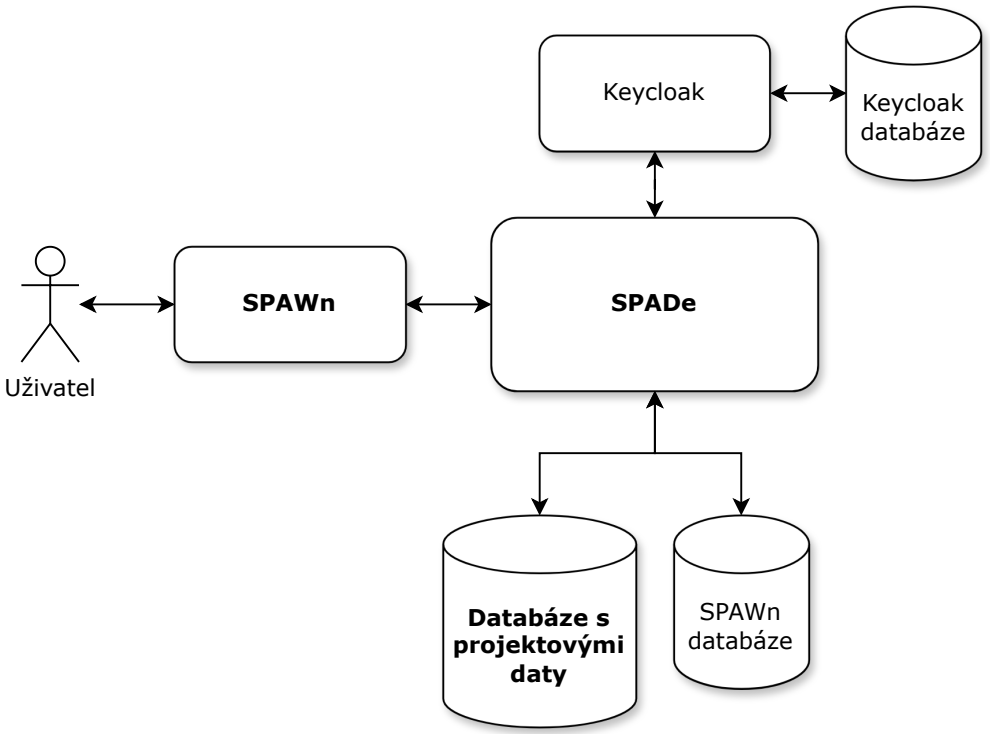
Klíčovou částí systému, která je podstatná pro tuto práci, je modul zajišťující analýzu projektových dat za účelem detekce anti-vzorů. Tato část obsahuje především detekční aplikaci SPADe, která tvoří jádro systému. Dále zahrnuje nezbytný datový zdroj, což je databáze s projektovými daty, a uživatelské rozhraní SPAWn. Aplikace SPAWn má vlastní databázi, obsahující například informace o uživatelých nebo další potřebná metadata. Zabezpečení a správa uživatelů je zajištěna pomocí Keycloaku, který pracuje s vlastní databází. Celkové schéma zkoumané části je zobrazeno na diagramu 3.4.

Aplikace SPADe je implementovaná v jazyce Java s využitím frameworku Spring. Detekce je prováděna na základě vytvořených detektorů. Jedná se o třídy implementující jednotné rozhraní, kdy pro každý anti-vzor je vytvořena právě jedna třída. Data jsou získávána z databáze za pomoci předem definované sady dotazů, jejichž výsledky jsou využívány v jednotlivých detektorech.

Uživatelské rozhraní SPAWn je postaveno na knihovně React a jazyku TypeScript. Tato aplikace využívá komponentový design, který je charakteristický pro technologii React a zahrnuje standardní nástroje a metody, které tato technologie nabízí. Komunikace s aplikací SPADe je zajištěna pomocí standardního Hypertext Transfer Protocol (HTTP) protokolu.

Databázová část je tvořena hned třemi různými technologiemi. Zatímco hlavní databáze obsahující projektová data je založena na MySQL, data-

báze pro metadata SPAWn využívá Microsoft SQL Server. Implementovaný Keycloak využívá databázi PostgreSQL.



Obrázek 3.4: Podoba detekční části systému SPADe

3.3.3 Nedostatky současného řešení

I přes poměrně dobře řešený návrh a rozložení jednotlivých částí, nástroj SPADe vykazuje několik nedostatků, které komplikují jeho používání a další rozvoj. Tyto nedostatky jsou zkoumány především ve vztahu k hlavní části, tedy detekčnímu jádru.

Jedním z hlavních nedostatků je složitá implementace nových detektorů, která vyžaduje vytvoření nové Java třídy implementující specifické rozhraní. Veškeré získávání potřebných dat a jejich zpracování probíhá přímo v této třídě a vyžaduje vytváření složitých programových struktur. Kód nového detektoru je tedy poměrně nepřehledný a může být pro nezasvěcené nepříliš dobře čitelný.

Přidání nového detektoru vyžaduje doplnění nových souborů na server a úpravu existujících souborů, například těch, které uchovávají prahové hodnoty pro další použití. Jedná se tedy o opětovné přeložení a sestavení (rebuild) projektu a jeho znovunasazení (redeploy) s následným restartem. Imple-

mentace jakéhokoli nového detektoru musí být proto prováděna programátorem s hlubokými znalostmi architektury a implementace aplikace a klade také velké časové nároky.

Dalším klíčovým nedostatkem je snížená možnost modularizace detekčních entit. Tedy oddělení a znovupoužití detektorů či jejich částí v dalších případech. Toto omezení výrazně snižuje další možnosti implementace nových detektorů a zároveň ztěžuje přidání dalších funkčních rozšířeních této aplikace.

V kódu aplikace také v některých případech nejsou dodržovány programátorské standardy, jako jsou SOLID (Single Responsibility Principle, Open Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle [19]) principy pro čistou implementaci a čistý kód. Kritické části aplikace navíc nejsou dostatečně otestovány jednotkovými testy, což znamená, že není ověřena a zajištěna jejich správná funkčnost.

Zmíněné nedostatky tvoří základ pro tuto práci, ve které se budou zmíněné problémy řešit. Zejména klíčovým je problém modularizace, tedy rozčlenění procesu detekce na znovupoužitelné části. Zároveň je cílem poskytnout možnost snadno vytvářet pravidla pro nové detekce anti-vzorů bez hlubokých znalostí implementace aplikace a také bez nutnosti složitého procesu integrace nově vytvořeného detektoru do běžící aplikace. Tím bude zvýšena celková efektivita nástroje a také uživatelská přívětivost celého systému SPADe.

4 Koncepty detekce anti-vzorů

Na úvod této kapitoly, která se věnuje konceptům detekce anti-vzorů, je klíčové připomenout, že důkladný průzkum této problematiky a pochopení klíčových konceptů jsou základem pro návrh efektivního řešení. Znalost těchto konceptů umožní navrhnout a implementovat změny do současné podoby systému, které povedou k zajištění požadovaných výsledků a efektivity při řešení výzev spojených s anti-vzory a jejich detekcí v rámci projektového a procesního řízení. Tato kapitola nejprve zkoumá modelování a popis anti-vzorů a všech jejich složek. Poté se zaměří na metody identifikace těchto anti-vzorů ve shromážděných projektových datech.

4.1 Modelování anti-vzorů

Anti-vzory v projektovém a procesním řízení představují různorodou skupinu, přičemž každý z nich se může lišit svým zaměřením. Zatímco některé anti-vzory se specializují na kvantitativní analýzu průběhu projektu a jeho jednotlivých fází, jiné mohou klást důraz na mezilidskou dynamiku a spolupráci v týmu. Tato variabilita komplikuje jednoduché stanovení univerzálního postupu nebo konvence pro jejich popis či modelování, což znesnadňuje systematické a automatické strojové zpracování těchto fenoménů.

V oblasti modelování anti-vzorů do jednotného schématu se již někteří autoři pokusili o vývoj svých přístupů. Jedním z nich může být systém SPARSE [29]. V této práci autoři navrhli systém k podpoře reprezentace a odhalování anti-vzorů v softwarových projektech s využitím technologie sémantického webu a znalostních systémů. Tento systém je založen na ontologickém modelu, který umožňuje definovat, spravovat a identifikovat anti-vzory a jejich vzájemné vztahy na základě jejich příznaků, příčin a důsledků. Další ukázkou může být systém SODA-BP [22], který je navržený k identifikaci a řešení opakujících se chyb v návrhu strukturovaných obchodních procesů, což zlepšuje jejich údržbu a kvalitu. Systém využívá vlastní doménově specifický jazyk pro formulaci pravidel detekce antipatternů. Problematice modelování anti-vzorů se také věnovali autoři systému SPADe. Ve své práci [8] shromáždili více než 160 antipatternů z různých zdrojů, které byly popsány s použitím jednotné šablony ve vytvořeném katalogu. V další

práci [9] popisují způsoby formalizace modelování anti-vzorů pomocí metrik a indikátorů, které jsou použity v detekčních algoritmech pro datový model SPADe.

Ačkoliv existují vědecké přístupy pro detekci a modelování anti-vzorů, dosud neexistuje žádné komerční a široce používané řešení. Tento fakt poukazuje na potřebu dalšího výzkumu a zdokonalení, který by mohl přinést efektivnější metody pro popis a identifikaci anti-vzorů v datech shromážděných z různých ALM nástrojů.

4.1.1 Návrh nového modelu

V této sekci bude diskutován, navržen a podrobně popsán model pro charakterizaci anti-vzorů a jejich komponent. Modelované anti-vzory budou využívány pro identifikaci v rámci unifikovaného datového schématu SPADe. Nový způsob modelování je inspirován dříve zmíněnou prací [9] autorů tohoto schématu, což zajistí konzistentnost práce v rámci celé projektu souvisejícího s nástrojem SPADe.

Základní struktura popisu anti-vzorů vychází z textové formy, která je pro člověka nejpřirozenější a nejintuitivnější. Tento popis je systematicky rozdělen do několika klíčových částí. Začíná názvem anti-vzoru, který je doplněn o další často používané alternativní názvy, jež umožňují různé perspektivy na stejný fenomén. Dále obsahuje shrnutí, které poskytuje stručný přehled o podstatě anti-vzoru. Další část je věnována příznakům (symptomům), které detailně popisují specifické projevy nebo chyby, které anti-vzor charakterizují. Tyto příznaky jsou klíčové pro identifikaci anti-vzoru v reálných situacích.

Je-li prostor všech možných příznaků S, potom každý anti-vzor p může být modelován jakožto podmnožina S' množiny S:

$$S' \models p, S' \subseteq S.$$

Navzdory intuitivnosti textové formy popisu, je pro účely automatického zpracování a detekce nezbytné tyto textové popisy převést do formalizované reprezentace platné přímo nad projektovými daty získanými z ALM nástrojů. Příznaky jsou mapovány na indikátory, které fungují jako výrazy nad těmito daty.

Je-li prostor všech možných indikátorů I, potom platí:

$$\forall s \in S \; \exists \; I' \subseteq I, I' \models s, I' = \{i_0, i_1, i_2, \ldots, i_n\}.$$

Indikátory se dále skládají z metrik, které hodnotí numerické a logické

hodnoty získané z projektových dat. Numerické metriky zahrnují různé kvantitativní údaje a parametry projektu. Logické metriky naopak sledují přítomnost nebo nepřítomnost určitých jevů nebo aspektů v rámci projektových dat.

Je-li prostor všech možných metrik M, potom platí:

$$\forall i \in I \; \exists \; M' \subseteq M, M' \models i, M' = \{m_0, m_1, m_2, \ldots, m_n\}.$$

Na nejvyšší úrovni hierarchie jsou anti-vzory definovány jako soubory indikátorů, které jsou propojeny pomocí logických operátorů nebo složitějších logických vztahů. Tento přístup umožňuje, aby anti-vzor byl chápán jako množina indikátorů, které jsou formalizovanými příznaky specifických aspektů projektů nebo procesů.

Je-li prostor všech možných anti-vzorů P, potom platí:

$$\forall p \in P \; \exists \; I' \subseteq I, I' \models p, I' = \{i_0, i_1, i_2, \ldots, i_n\}.$$

Je možné si všimnout, že uvedená definice anti-vzoru odpovídá výše uvedené definici příznaku. Nicméně vycházíme-li z předpokladu, že anti-vzor se stává z alespoň jednoho příznaku, je vztah mezi anti-vzorem a příznaky následující:

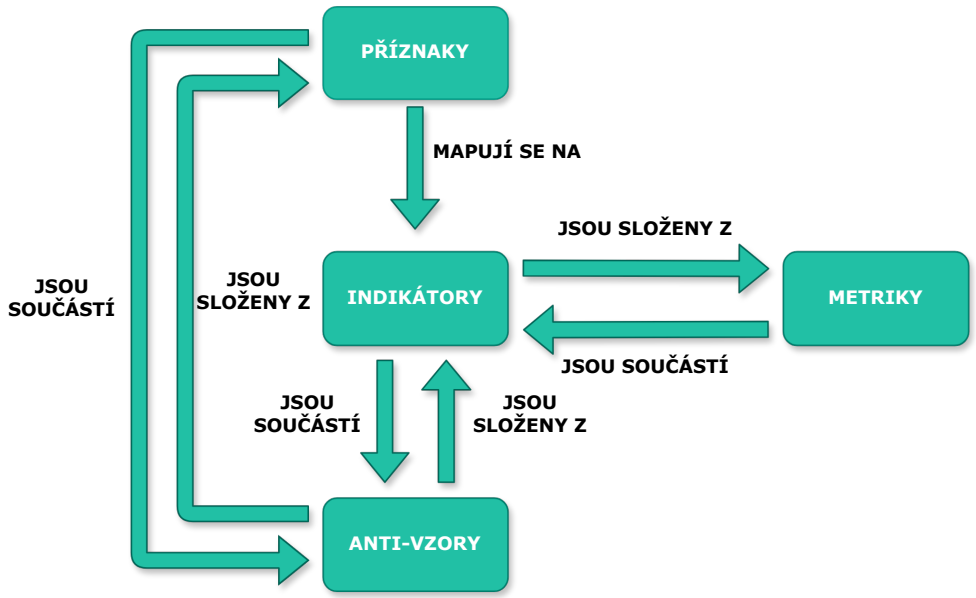
$$\forall p \in P \; \exists \; S' \subseteq S, S' \models p, S' = \{s_0, s_1, s_2, \ldots, s_n\}.$$

Vztahy mezi příznaky, indikátory, metrikami a anti-vzory jsou graficky znázorněny na obrázku 4.1.

4.1.2 Klíčová pozorování

Zajímavým pozorováním vycházejícím z konkrétních anti-vzorů je, že jeden příznak může figurovat ve více anti-vzorech. Příkladem může být příznak přidání nových vývojářů do projektu před koncem projektu s úmyslem práci na projektu urychlit. S tímto příznakem se můžeme setkat například u anti-vzorů Nine Pregnant Women a Brooks' Law [8]. Analogicky i jedna metrika může být obsažena ve více indikátorech. Metrikou může být například přírůstek vývojářů za dané časové období. Tento fakt je zásadní pro definování architektury aplikace, která by měla obsahovat modulární prvky, které umožní jejich znovupoužitelnost.

Dalším pozorováním, které je klíčové pro návrh architektury řešení, je fakt, že anti-vzor může být tvořen jediným indikátorem, který je pak označen jako anti-vzor. Příkladem může být indikátor přítomnosti přispěvatele



Obrázek 4.1: Znázornění vztahů mezi fenomény

bez známé reálné identity. Tento indikátor je pak jako jediný součástí anti-vzoru s názvem Unknown Poster [31]. Jinými slovy, na anti-vzor je možné nahlížet jako na indikátor, který byl jakožto anti-vzor specificky označen. Nicméně, tento způsob definice už nelze aplikovat na vztah mezi indikátorem a metrikou. I když indikátor může zahrnovat pouze jednu metriku, nelze tvrdit, že v tomto případě metrika sama představuje indikátor. Indikátor je v tomto kontextu prvek, který používá danou metriku a zahrnuje logiku pro vyhodnocení svého stavu na základě této metriky.

4.2 Detekce anti-vzorů

V předchozí podkapitole byly představeny základní koncepty anti-vzoru, indikátoru a metriky. Následující text se zabývá procesem detekce anti-vzorů, tedy vysvětlením, co znamená, že anti-vzor je ve zkoumaých projektových datech detekován. Podle definice, která je užívána ve zkoumaných zdrojích, detekce spočívá v aplikaci binárních funkcí. Tyto funkce porovnávají každou naměřenou hodnotu s určenou prahovou hodnotou. Pokud dojde k překročení prahové hodnoty (jedním nebo druhým směrem), funkce vrátí hodnotu `true` a zkoumaný fenomén považujeme za detekovaný či aktivovaný. V opačném případě funkce vrátí hodnotu `false`.

Je-li x_i naměřená hodnota a t_i prahová hodnota, pak pro binární detekční funkci f platí, že:

$$f(x_i, t_i) \in \{\text{true}, \text{false}\}$$

V kontextu indikátoru se zkoumá, jak jsou aktivovány jednotlivé metriky, respektive zda dochází k překročení nastavených prahových hodnot pro každou z nich. Zkoumaný zdroj uvádí, že indikátor je považován za detekovaný pouze tehdy, když jsou překročeny prahové hodnoty všech metrik v něm obsažených. Toto tvrzení však není zcela přesné, což ilustruje anti-vzor nazvaný Road To Nowhere. Tento anti-vzor se projevuje například příznakem nedostatku plánovacích aktivit či artefaktů [8]. Detekce tohoto příznaku může využívat různé metriky:

- přítomnost či absence plánovacích aktivit,
- počet plánovacích aktivit,
- čas strávený na plánovacích aktivitách,
- přítomnost artefaktů obsahujících informace o plánování,
- počet artefaktů s informacemi o plánování,
- délka (počet znaků/slov) artefaktů s informacemi o plánování.

Z výčtu potenciálních metrik je patrné, že indikátor nemusí být nutně označen za detekovaný pouze na základě splnění všech prahových hodnot. Například, pokud tým provádí dostatečné množství plánovacích aktivit, ale každé z nich nevěnuje dostatek času a nemá žádné relevantní plánovací artefakty, je zřejmé, že indikátor nedostatku plánovacích aktivit či artefaktů může být považován za detekovaný, i když nebyla splněna jedna z metrik.

Indikátor tedy může být označen za detekovaný, pokud je splněna určitá kombinace metrik, jejichž hodnoty překračují nastavené prahové hodnoty. Indikátor je v podstatě logikou, která pracuje s množinou metrik a jejich prahovými hodnotami a určuje, za jakých podmínek může být daný indikátor považován za detekovaný. Pro flexibilitu detekce na různých typech projektových dat by u této logiky měla být umožněna případná parametrizace pomocí množiny vstupních parametrů.

Je-li X' množina všech naměřených hodnot metrik v rámci jednoho indikátoru, T' množina všech prahových hodnot pro tyto metriky a Q' množina vstupních parametrů, pak pro indikátorovou detekční funkci g platí:

$$g(X', T', Q') \in \{\text{true}, \text{false}\},$$
$$X' = \{x_0, \ldots, x_n\}, T' = \{t_0, \ldots, t_n\}, Q' = \{q_0, \ldots, q_m\}.$$

Velmi podobný případ nastává u detekční funkce pro anti-vzory. Dle zkoumaného zdroje platí, že anti-vzor je detekovaný právě tehdy, když dojde k detekovaní všech indikátorů v něm obsažených. I na tento výrok lze najít protipříklad. Ukázkou, která toto vyvrací, je anti-vzor s názvem Bystander Apathy. Ten nastává v softwarových projektech, když týmoví členové nejsou ochotni pomoci kolegům s jejich problémy. Tento jev vede k tomu, že osoby potřebující pomoc zbytečně dlouho čekají na odpověď nebo musí problém řešit samy, což způsobuje zdržení a snižuje efektivitu [31]. U tohoto anti-vzoru se můžeme setkat s indikátory jako:

- nevyřešené nebo neuzavřené aktivity zůstávají po dlouhou dobu beze změny,
- dotazy a žádosti pracovníků jsou dlouhou dobu bez reakce,
- pracovník se potýká s jedním úkolem a tráví na něm až příliš mnoho času [8].

I v tomto případě je patrné, že pro detekci anti-vzoru nemusí být nezbytně detekovány všechny uvedené indikátory. Pokud v projektu nejsou evidovány žádné nevyřešené či neuzavřené aktivity dlouho beze změny, nicméně někteří pracovníci na jejich řešení opakovaně trávili až přehnaně mnoho času a všechny jeho žádosti o pomoc zůstaly bez odpovědi, můžeme tento anti-vzor považovat za detekovaný, i když nebyly striktně naplněny všechny definované indikátory.

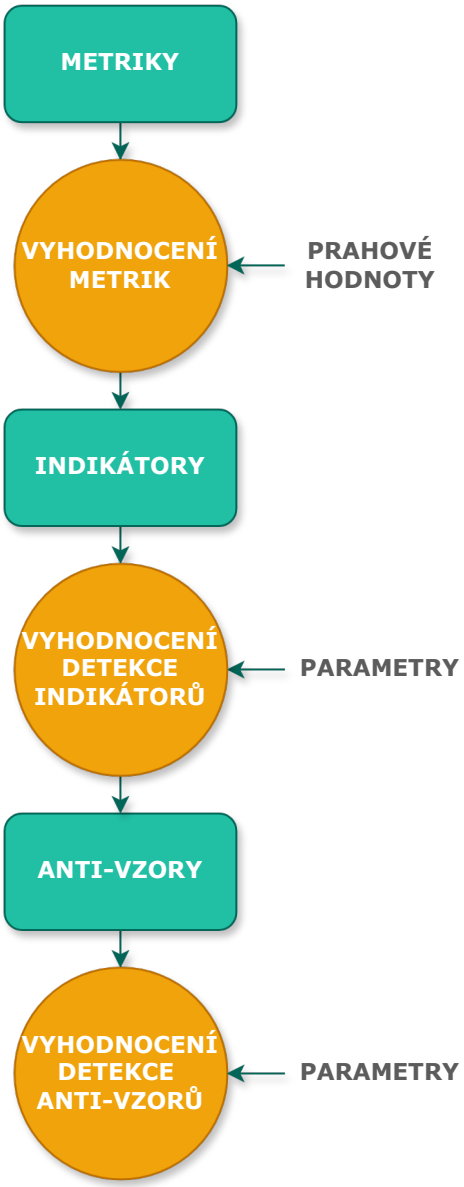
Anti-vzor je tedy možné chápat jako parametrizovatelnou logiku, která řídí množinu indikátorů a určuje, za jakých okolností je anti-vzor považován za detekovaný. Tento pohled podtrhuje podobnost s konceptem indikátoru, což podporuje dříve popsanou myšlenku, že anti-vzor je ve skutečnosti speciálně označený indikátor.

Je-li I' množina indikátorů daného anti-vzoru p a R' množina vstupních parametrů, pak pro detekční funkci anti-vzoru h platí:

$$h(I', R') \in \{\text{true}, \text{false}\}, I' \models p, I' = \{i_0, \ldots, i_n\}, R' = \{r_0, \ldots, r_m\}.$$

Celý proces identifikace anti-vzoru tedy začíná měřením projektových dat pomocí metrik, které na základě stanovených prahových hodnot určují přítomnost nebo nepřítomnost specifických aspektů. Tyto metriky poskytují informace pro indikátory, jež na základě svých metrik rozhodují o existenci určitých příznaků v datech. Dále se od indikátorů postupuje k anti-vzorům,

které využívají logiku nad svými indikátory k určení své přítomnosti v datech. Pro zjednodušení pochopení byl tento komplexní detekční proces znázorněn na obrázku 4.2.



Obrázek 4.2: Znázornění procesu detekce

5 Návrh nové architektury

V této kapitole je popsán proces a principy, které byly zvoleny pro návrh nové architektury detekčního systému. Vzhledem k různým výzvám, kterým čelí stávající systém, je důležité pečlivě plánovat jakoukoliv architektonickou změnu. Návrh musí zohlednit nejen technické požadavky, ale také budoucí rozšiřitelnost, modularitu a integraci s existujícími částmi nástrojové sady SPADe.

Tato kapitola proto zkoumá různé aspekty návrhu, od obecných principů, přes identifikaci komponent a volbu technologií až po strategii testování. Je zde kladen důraz na zajištění, že výsledné řešení bude robustní, škálovatelné a snadno udržovatelné.

5.1 Principy návrhu

Klíčovým principem úspěšného návrhu systému je jeho soulad s definovanými cíli, což vyžaduje přesné vymezení účelů, které má nová architektura plnit. Tyto cíle přímo vycházejí z identifikovaných nedostatků stávajícího řešení, jak bylo podrobně popsáno v předchozích částech této práce.

Hlavním cílem je vytvořit modulární řešení, které bude odrazem jednotlivých modelů anti-vzorů, indikátorů a metrik a umožní jejich efektivní kompozici do funkčních celků a zároveň jejich snadnou detekci na množině projektových dat uložených v databázi. Tento přístup nejenže usnadní znovupoužitelnost jednotlivých komponent v různých kontextech, například při opakovaném využívání stejné metriky v různých indikátorech, ale také zvýší celkovou efektivitu aplikace tím, že omezí redundanci. Redundance může vést nejen k nadbytečnému zabírání prostoru v datových úložištích, ale i k potenciálnímu zmatku způsobenému nekonzistencí v popisu stejné funkcionality na různých místech aplikace.

Dalším cílem je umožnit snadné vytváření a implementaci nových detektorů anti-vzorů a indikátorů bez nutnosti rozsáhlých zásahů do kódu aplikace, což povede k výraznému zjednodušení procesu přidávání nových funkcionalit do produkčního prostředí. Kromě možnosti snadného rozšíření o další detektory by měl být systém navržen s přihlédnutím k potenciálním budoucím funkčním rozšířením. Je také nezbytné, aby byla architektura od počátku koncipována s ohledem na budoucí škálovatelnost aplikace, aby mohla efektivně reagovat na rostoucí požadavky a objem dat.

Výsledné řešení musí být kompatibilní s nástrojovou sadou SPADe, což

vyžaduje účinnou integraci s ostatními komponentami a soulad s celkovou vizí projektu. Pouze tak bude možné zaručit, že implementace přinese požadovanou efektivitu a přinese celému projektu požadované přínosy.

V průběhu návrhu bude využitý přístup top-down, což je metodika, při které se postupuje od obecných koncepčních rámů a architektonických rozhodnutí k detailním specifikacím jednotlivých komponent [13].

5.2 Vlastnosti a integrace nového řešení

Tato podkapitola je věnována klíčovým vlastnostem navrhovaného řešení, tedy popisu funkcí, které toto řešení zastává. Vzhledem k tomu, že je řešení integrováno do již existujícího systému, je také nezbytné detailně specifikovat, jak se zapojuje k ostatním komponentám systému a jaké role v něm plní.

Vlastnosti

Nově vytvořené řešení by mělo být navrženo v souladu s principy zmíněnými v předchozí podkapitole a z toho plynou jeho základní vlastnosti, a těmi jsou možnosti definice a detekce fenoménů, jako jsou anti-vzory, indikátory a metriky. Řešení by také mělo podporovat definici a detekci špatných praktik, které jsou, stejně jako anti-vzory, indikátory specificky označené jako špatné praktiky. Hlavním účelem řešení je umožnit uživatelům efektivně definovat tyto fenomény a následně detekovat jejich přítomnost v projektech uchovávaných v databázi.

- **Definice fenoménů** – Tento aspekt zahrnuje schopnost definovat jednotlivé fenomény samostatně, jejich seskupování do komplexnějších struktur, a také možnost těmto strukturám přidělovat jména, popisky a ukládat je pro budoucí použití. Tato flexibilita je klíčová pro zajištění modularity, a tím pádem efektivní práce s fenomény.
- **Detekce fenoménů** – Po definici se fenomény testují na projektových datech uložených v databázi, aby se zjistila jejich přítomnost nebo absence. Řešení by tedy mělo být schopné zpracovávat požadavky na provedení detekce a následně poskytnout výsledky.

Těsné propojení definice a detekce je zásadní pro udržení konzistence a efektivity procesu, což vyžaduje, aby tyto dvě funkce byly umístěny blízko sebe v rámci architektury.

Integrace do systému

Klíčovou otázkou je optimální začlenění nového řešení do celkové architektury systému. Možnosti zahrnují integraci jako nový modul do aplikace SPADe nebo vytvoření nezávislé mikroslužby, která by s aplikací komunikovala. Volba mezi těmito alternativami vyžaduje analýzu jejich výhod a omezení:

- **Modul v rámci aplikace SPADe** – Toto řešení umožňuje těsnější integraci s existujícími funkcemi a zjednodušenou správu, což usnadňuje nejen nasazení aplikace. Mezi nevýhody patří nižší spolehlivost, přičemž chyba v jakémkoliv modulu může způsobit pád celé aplikace. Navíc se objevuje problém s omezenými možnostmi škálování [16].
- **Samostatná mikroslužba** – Poskytuje lepší škálovatelnost a izolaci, což usnadňuje údržbu a rozvoj. Mikroslužby jsou obecně více adaptabilní a nezávislé. Výzvou však může být zajištění integrity všech dat, náročnější nasazení a také dodatečná komunikace, která může ovlivnit celkový výkon systému [16].

Je zásadní vzít také v úvahu současný stav aplikace SPADe, jejíž detekční část bude nahrazována. Přestože byla aplikace původně koncipována jako modularizovaná, v praxi se oddělení modulů často nedodržuje adekvátně, což komplikuje orientaci v implementaci aplikace. Před začleněním nového řešení by proto bylo vhodné provést revizi a nezbytné úpravy stávající architektury a všech modulů.

Paralelně s realizací této diplomové práce probíhá další vývoj, který si klade za cíl rekonstrukci části architektury SPADe a její rozšíření o nové funkcionality. Současný průběh těchto dvou projektů, proto může vést ke značným komplikacím ve vývoji a vyžaduje by přesnou součinnost všech stran.

Faktorem klíčového významu je v tomto kontextu možnost škálování. Detekce představuje poměrně komplexní analytický proces zpracování rozsáhlých datových sad. Tento detekční proces je charakteristický vysokými výkonnostními nároky. V případě implementace jako modul by mohlo dojít k negativnímu ovlivnění výkonu a efektivity ostatních částí aplikace. Kromě toho jsou možnosti škálování v reakci na rostoucí objem projektových dat a počet fenoménů pro detekci v tomto případě omezené.

Vzhledem k těmto faktorům a po konzultaci s vedoucím této práce, který je zároveň vlastníkem projektu SPADe, bylo rozhodnuto implementovat nové řešení jako mikroslužbu, která bude komunikovat s hlavní aplikací SPADe.

Toto rozhodnutí bylo učiněno především kvůli jednoduššímu a nezávislému vývoji, budoucí škálovatelnosti a možnosti využití této mikroslužby v různých projektech nebo při zavádění nových klientů.

Potřeba datového úložiště

Je zásadní zvážit, jakým způsobem bude navrhovaná služba využívat datové úložiště. V současné podobě je detekční část nástroje SPADe navržena tak, že informace o anti-vzorech jsou částečně zakódovány přímo v programu a částečně uloženy v JSON souborech v adresáři aplikace. Tento model by měl být přepracován tak, aby se využívala databáze, která oddělí data od logiky aplikace a zároveň je centralizuje na jednom místě.

V současné aplikaci jsou využívány tři technologie relačních databází. Cílem není přidávat novou technologii, ale využít jednu z již integrovaných. Toho bude dosaženo vytvořením nového databázového schématu, které bude specificky určeno pro tuto mikroslužbu. V tomto novém schématu budou uložena všechna relevantní data týkající se jednotlivých fenoménů a jejich skládání do vyšších celků.

Shrnutí

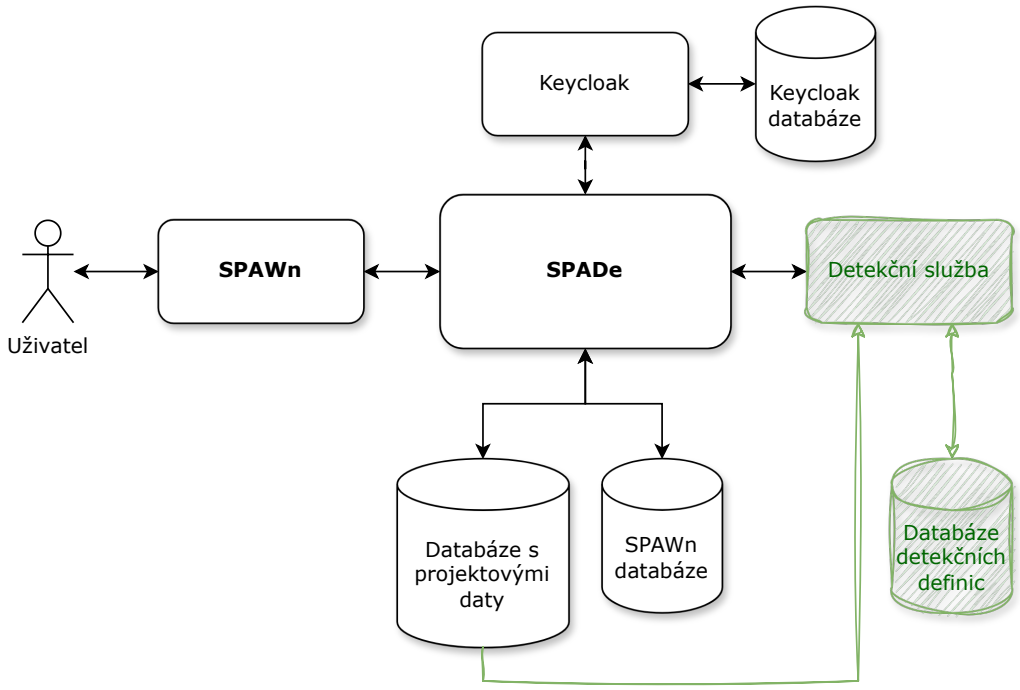
Jak již bylo uvedeno, nové řešení bude začleněno do stávající detekční infrastruktury jako samostatná mikroslužba, využívající vlastní datové schéma v rámci stávající databáze. Tato služba bude interagovat s aplikací SPADe, která bude fungovat jako centrální entita celého detekčního schématu. Součástí integrace bude také vzájemná komunikace s databází obsahující projektová data, na kterých budou prováděné analýzy. Začlenění mikroslužby do detekční části nástroje SPADe je ilustrováno na obrázku 5.1.

5.3 Identifikace a interakce komponent

Na úrovni návrhu architektury detekční mikroslužby se nyní přechází od high-level pohledu k detailnějšímu rozboru. Prvním důležitým krokem je identifikace všech potřebných komponent a stanovení způsobu jejich vzájemné interakce.

Komponenty pro uchování dat

Nejprve bude nezbytné vytvořit komponenty, které budou reprezentovat jednotlivé fenomény, jako jsou metriky, indikátory a anti-vzory.



Obrázek 5.1: Nová služba v kontextu detekční části nástroje

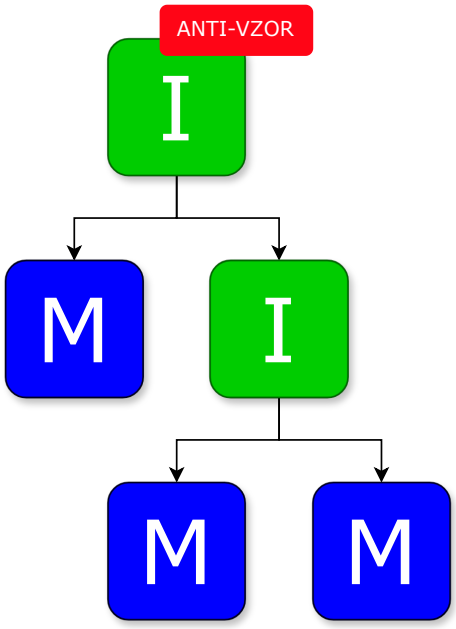
Jako první bude navržena komponenta reprezentující metriku. Tato komponenta bude obsahovat základní údaje, jako je název a popis. Metriky budou implementovány formou dotazů do projektových dat. Dále bude nutné zahrnout mechanismus, který umožní parametrizaci metrik prostřednictvím prahových hodnot, což zahrnuje způsob zadávání těchto hodnot do dotazů spojených s metrikami.

Dalším krokem bude vytvoření komponenty pro indikátory. Podobně jako metrika, i indikátor bude obsahovat základní údaje a bude vyžadovat možnost parametrizace. Logika indikátorů bude definována specifickým programovacím jazykem nebo gramatikou, která zahrne odkazy na používané metriky.

Anti-vzor, který je ve své podstatě speciálně označeným indikátorem, bude zpracován rozšířením stávající komponenty pro indikátory. Toto rozšíření zahrne informace, zda se jedná o běžný indikátor nebo anti-vzor. Logika zápisu anti-vzoru bude navíc umožňovat odkazy nejen na používané metriky, ale i na další indikátory, které jsou v procesu detekce využívány.

Znázornění možného vztah anti-vzoru, indikátorů a metrik může být je na obrázku 5.2. Na vrcholu se nachází indikátor označený jako anti-vzor. Ten ve svém těle využívá napřímo jednu metriku a jeden indikátor. Tento indikátor dále využívá další dvě metriky. Tímto způsobem je možné sestavovat

libovolné modely anti-vzorů a indikátorů.



Obrázek 5.2: Možný vztah anti-vzoru, indikátorů a metrik

Komponenty pro procesy

V rámci této služby lze rozlišit dva základní procesy odpovídající jejím klíčovým funkcím: definici a detekci.

Proces definice zahrnuje vytváření a kompozici nových anti-vzorů, indikátorů a metrik. Tento proces obnáší tvorbu nových instancí modelových tříd, jejich konfiguraci a následné ukládání do databáze. Cílem je poskytnout uživatelům nástroje pro definování a strukturování vlastních kritérií pro analýzu.

Proces detekce je sofistikovanější a zahrnuje analyzování projektových dat na základě definovaných parametrů uložených v modelových komponentách. Aby bylo zajištěno, že detekce proběhne správně, musí být proces řízen s důrazem na správné pořadí jednotlivých kroků. Výsledky analýzy musí být přesné a spolehlivé, aby mohly být prezentovány uživateli, který požádal o provedení detekce.

Komponenty pro integraci

Důležitým aspektem jsou komponenty odpovědné za integraci interních prvků služby s jejím okolím. Zásadní bude zajištění interakce s aplikací SPADe,

která bude zasílat požadavky na detekci, úpravu nebo vytvoření nových definic a obdrží výsledky provedených detekcí.

Dalším klíčovým bodem je komunikace s databázemi. Je nezbytné zajistit spojení s databází, která bude uchovávat definice fenoménů. Současně je potřeba komunikovat s databází obsahující projektová data určená k analýze, toto spojení však bude pouze jednosměrné, tedy pro čtení. Tato spojení umožní efektivní správu a přístup k datům nezbytným pro operace detekce a definice, což je klíčové pro správnou funkčnost celého systému.

Výsledný návrh komponent detekční služby a jejich vzájemného propojení i propojení s okolím je vizualizován na obrázku 5.3.

5.4 Proces a datový tok detekce

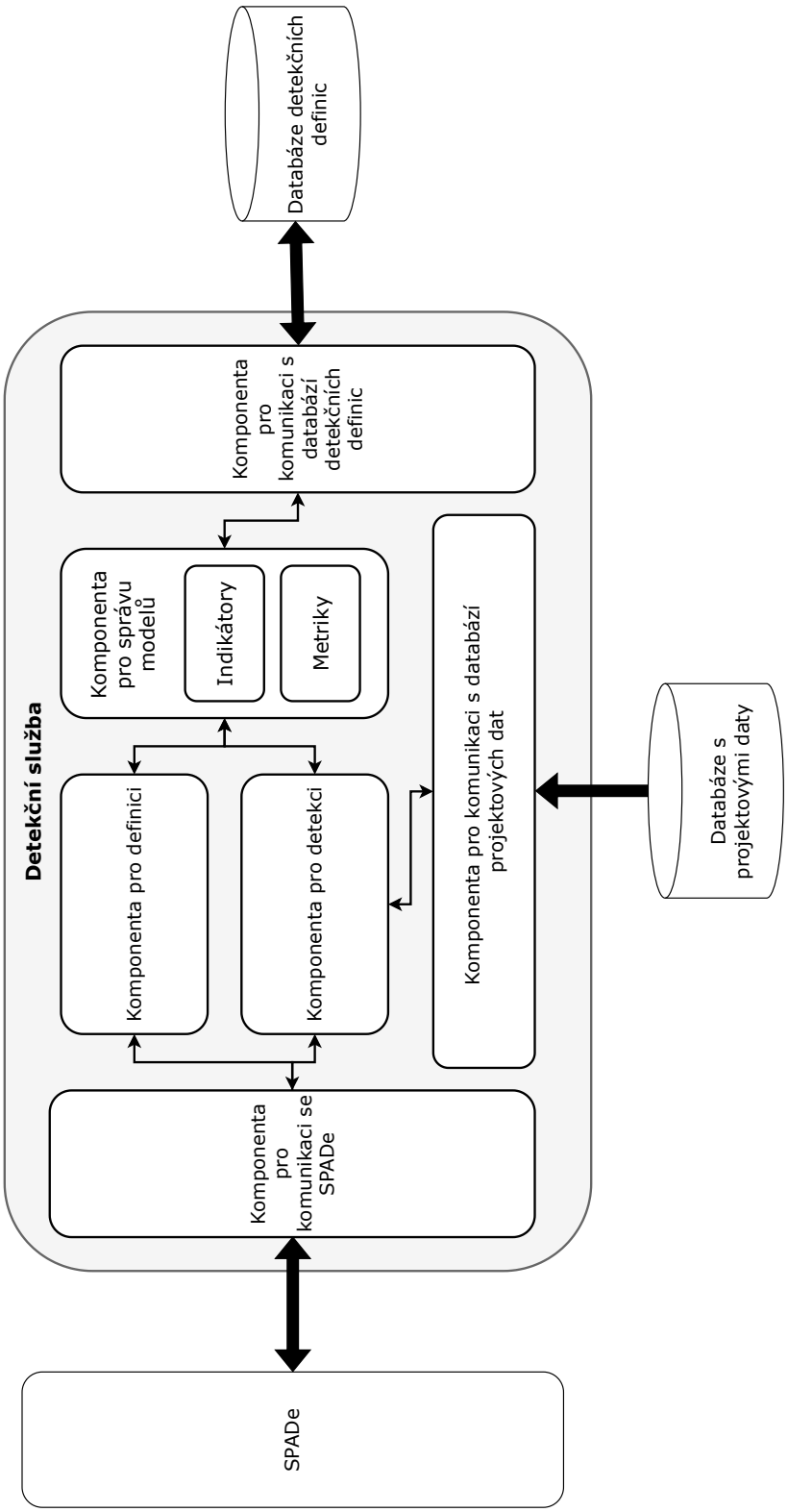
Samotná detekce dat představuje hlavní a zároveň nejsložitější proces celého systému, což vyžaduje pečlivě navržený proces a v něm tok dat. Nejde pouze o projektová data a data metrik a indikátorů vstupujících do procesu, ale také o prahové hodnoty pro nastavení metrik a parametry pro indikátory.

Proces začíná příjmem požadavku od SPAWn, kde uživatel specifikuje, které anti-vzory chce detekovat a na kterých projektech. Pro každý projekt je následně vyhodnocována sada těchto anti-vzorů, a to postupně.

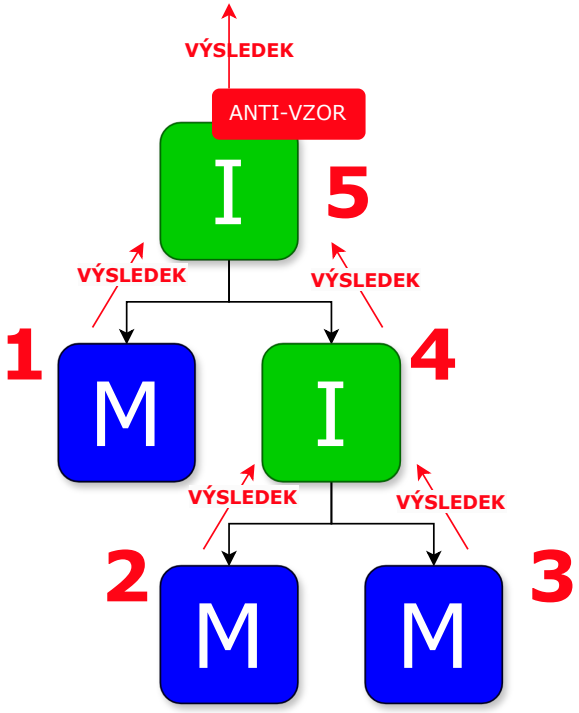
V každé iteraci se nejprve z databáze vybírá logika jednotlivých anti-vzorů. Tato logika je procedurálně vyhodnocována. Pokud logika anti-vzoru obsahuje volání indikátoru, zavolá se pro daný indikátor rekurzivně stejný proces vyhodnocení. Výsledky těchto vyhodnocení jsou poté zpracovávány dál. Pokud dojde k volání metriky, vybírají se relevantní data z databáze podle dotazu dané metriky a následně se vyhodnocují společně s její prahovou hodnotou. Ať už jde o soubor dat nebo logický výsledek, výsledky jsou zpětně integrovány do zpracovávaného anti-vzoru nebo indikátoru. Tento postup pokračuje, dokud není dokončen běh hlavního těla anti-vzoru, jehož výsledky pak představují konečný výsledek detekce daného anti-vzoru v projektu.

Vyhodnocování anti-vzoru a všech jeho komponent je znázorněno na obrázku 5.4, který znázorňuje již dříve zmíněný příkladový anti-vzor. Čísla na obrázku znázorňují, v jakém pořadí bude dokončeno vyhodnocování jednotlivých fenoménů (předpokládáme-li, že jejich pořadí v tomto diagramu je stejné jako v jejich zápisech) a jak budou předávány veškeré výsledky.

Důležitým aspektem je také efektivní předávání a správná distribuce prahových hodnot a parametrů. Pro zajištění správné funkčnosti systému je klí-



Obrázek 5.3: Komponenty a jejich interakce v rámci detekční služby



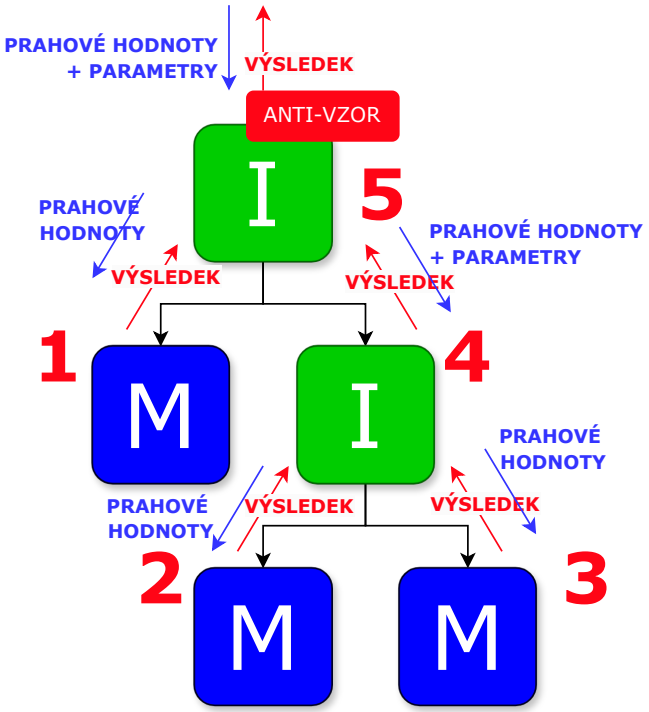
Obrázek 5.4: Vyhodnocení anti-vzoru a jeho částí

čové, aby každý fenomén disponoval všemi potřebnými parametry. Uživatel proto musí při zadávání požadavku na detekci zadat všechny potřebné parametry, včetně těch, které jsou aplikovány na vnořené indikátory a metriky. Existují však výjimky, kdy například vnořený indikátor může metrice přiřadit prahovou hodnotu staticky. V takovém případě není nutné ani možné, aby byla tato hodnota nastavena z vyšší vrstvy, tedy nadřazeným indikátorem. Distribuce parametrů a prahových hodnot je ilustrována na příkladu anti-vzoru na obrázku 5.5.

Předávání parametrů a výsledků detekcí bude probíhat v rámci detekční komponenty. V této komponentě budou dílčí moduly pro vyhodnocování metrik a indikátorů, které jako vstup budou přijímat příslušné parametry a na výstupu budou vracet výsledky.

5.5 Výběr technologií

V této podkapitole budou diskutovány a vybrány klíčové technologie pro navrhovaný detekční systém, od databáze a mikroslužby až po technologii řešící problém zápisu a vyhodnocování indikátorů.



Obrázek 5.5: Vyhodnocení anti-vzoru a jeho částí

5.5.1 Databáze a mikroslužba

Výběr technologií pro implementaci mikroslužby a databáze bude primárně založen na technologiích, které jsou již v projektu využívány.

Databáze

Pro správu databázových požadavků detekční služby bylo rozhodnuto použít MySQL [1]. Tato volba byla motivována několika důležitými faktory. Především MySQL již úspěšně slouží pro správu projektových dat v rámci aplikace SPADe, což potvrzuje jeho spolehlivost a efektivitu v reálném provozu. Využití MySQL pro nové schéma navíc umožňuje jednodušší integraci a správu dat, jelikož se může spoléhat na stávající infrastrukturu. V praxi bude přidáno pouze nové databázové schéma.

Mikroslužba

Pro vývoj detekční mikroslužby byl vybrán jazyk Java a Spring Framework [2], což je rozhodnutí podložené několika klíčovými důvody. Spring nabízí ro-

[1] https://www.mysql.com/
[2] https://spring.io/projects/spring-framework

bustní podporu pro vývoj mikroslužeb a umožňuje budoucí efektivní škálování a správu. Díky své modulární architektuře a širokému spektru dostupných modulů, Spring usnadňuje rychlý vývoj a snadnou údržbu aplikací.

Dalším významným faktorem je, že Spring je již používán v aplikaci SPADe. To znamená, že účastníci na projektu jsou s tímto frameworkem seznámeni, což zkracuje čas potřebný pro další rozšiřující vývoj. Stejně tak použití již známé technologie přispívá k jednotnosti a soudržnosti celého softwarového stacku, což je zásadní pro udržení stabilního a efektivního prostředí.

5.5.2 Zápis a vyhodnocení indikátorů

Zápis a vyhodnocení indikátorů představují dva klíčové procesy, které jsou zásadní pro funkčnost a efektivitu detekčního systému. Je proto nezbytné pečlivě zvážit dostupné možnosti a zvolit řešení, které nejlépe vyhovuje technickým požadavkům projektu.

Požadavky

Při výběru technologie pro zápis a vyhodnocení indikátorů je nezbytné stanovit specifické požadavky, které musí technologie splňovat:

- Zápis bude realizován ve formě skriptu (série příkazů).
- Výsledek skriptu bude logická hodnota true/false doplněná o dodatečné informace.
- Skript umožní vkládání parametrů.
- Musí podporovat volání a vyhodnocování vnořených indikátorů a metrik.
- Skript musí podporovat aritmetické a logické operace, práci s proměnnými, cykly, podmínky a tvorbu struktur/objektů.
- Zápis by měl být maximálně jednoduchý a čitelný.
- V případě použití externí knihovny musí být tato knihovna dobře dokumentovaná, udržovaná a plně kompatibilní s technologií použitou v mikroslužbě.
- Použití musí být maximálně bezpečné.

Důležitost bezpečnosti nemůže být podceňována, zejména když systém umožňuje uživatelům vytvářet a spouštět vlastní kód. Tento přístup přináší několik potenciálních rizik, které je třeba řádně ošetřit:

- I/O a síťové operace,
- dynamické načítání tříd a knihoven,
- systémové příkazy a přístup k systémovým prostředkům,
- nekonečné smyčky a rekurze,
- přístup k datům z jiných částí aplikace,
- změny v runtime,
- přístup ke globálním proměnným nebo kontextu.

Tento seznam bezpečnostních rizik ukazuje, jaké opatření je třeba zvážit pro zajištění bezpečnosti a integrity systému. Tyto požadavky jsou nejen klíčové pro funkčnost, ale také pro zajištění důvěry uživatelů.

Vzhledem k tomu, že řešení bezpečnostních požadavků může být vysoce komplexním úkolem, který vyžaduje podrobnou analýzu všech potenciálních přístupů, bylo po konzultaci s vedoucím práce rozhodnuto, že tento aspekt nebude v rámci této práce zahrnut. Bezpečnost bude dočasně zajištěna tak, že přístup k používání těchto mechanismů bude omezen pouze na ověřené uživatele. V budoucím rozvoji systému SPADe je však nezbytné tento aspekt řádně adresovat.

Výběr řešení

Při rozhodování o implementaci zápisu a vyhodnocení indikátorů se nabízí dvě hlavní možnosti: vývoj vlastní gramatiky nebo využití existujícího skriptovacího jazyka prostřednictvím externí knihovny. Ačkoliv by vlastní gramatika poskytla řešení přizpůsobené specifickým potřebám projektu, bylo by toto řešení spojeno s vysokými náklady na čas a zdroje pro vývoj a testování. Vzhledem k tomu, že základní požadavky pro skriptovací jazyk korespondují s funkcemi dostupných jazyků, je efektivnější volbou využití stabilní a osvědčené externí knihovny.

Další alternativou by mohlo být začlenění doménově specifického jazyka nebo pravidlově orientovaného jazyka. Tyto možnosti by však vyžadovaly výrazně složitější procesy ve fázích jejich definice, implementace a interpretace. Vzhledem k tomu, že tyto jazyky pro účely zápisu a vyhodnocení

indikátorů neposkytují výrazné výhody oproti existujícím skriptovacím jazykům, po pečlivém zhodnocení se jako nejvhodnější řešení ukázalo využití externí knihovny pro jeden z běžně používaných skriptovacích jazyků.

Ideálním řešením se jeví knihovna *javax.script*, která umožňuje spouštění skriptů v různých jazycích, jako jsou Python, Groovy, JavaScript a Java. Tato knihovna nabízí nejen flexibilitu a širokou podporu pro různá skriptovací prostředí, ale také poskytuje tzv. binding [3], což umožňuje zpřístupnění Java objektů skriptům jako pojmenované proměnné. To je klíčové pro snadné vkládání parametrů a efektivní volání metod, což zajišťuje správné zpracování metrik a indikátorů. Díky široké dostupnosti dokumentace a čistotě zápisu jazyků podporovaných touto knihovnou je tento výběr vhodný pro potřeby projektu.

Ohledně konkrétního skriptovacího jazyka byl zvolen Groovy pro jeho syntaktickou blízkost k Javě a schopnost integrace s Javovskými knihovnami. Groovy je ideální pro skriptovací účely díky své flexibilitě a efektivnímu zpracování skriptů. Přesto bylo rozhodnuto, že řešení bude vyvíjeno s možností snadného rozšíření o podporu dalších skriptovacích jazyků, což uživatelům v budoucnu umožní zvolit preferovaný jazyk pro zápis skriptů. Tento přístup zvyšuje adaptabilitu a budoucí využitelnost aplikace v různých scénářích.

5.6 Testovací strategie

Důležitým aspektem vývoje každé softwarové aplikace je ověření její kvality. Pro tuto mikroslužbu bude kvalita ověřována především prostřednictvím jednotkových testů. Hlavním cílem těchto testů není nutně pokrýt celou nebo většinu aplikace, ale zajistit, že klíčové funkce fungují správně podle očekávání a jsou spolehlivé.

Kromě jednotkových testů je klíčová i správná integrace s ostatními částmi nástrojové sady SPADe. Tato integrace bude ověřována pomocí integračních testů, které zajistí, že komponenty systému efektivně spolupracují.

Dalším krokem v testovacím procesu je akceptační testování prováděné ve spolupráci s vedoucím této práce. Akceptační testy systematicky ověřují, že vývoj řešení postupuje správným směrem a splňuje očekávání a požadavky vlastníka projektu.

Nakonec se uskuteční i uživatelské testování, které má za úkol ověřit, že aplikace je z pohledu koncových uživatelů přívětivá a splňuje všechny jejich očekávání. Toto testování je nezbytné pro zajištění, že konečný produkt je

[3] https://docs.oracle.com/en/java/javase/17/docs/api/java.scripting/javax/script/package-summary.html

nejen funkční, ale také intuitivní a uživatelsky přívětivý.

5.7 Metodika vývoje

Vývoj navrhovaného řešení započne konstrukcí detekční mikroslužby, přičemž souběžně dojde k vytvoření struktury databáze pro detekční definice. Následovat bude integrace této mikroslužby do existující sady nástrojů SPADe, která zahrnuje vývoj nového uživatelského rozhraní specificky navrženého pro detekční účely. V závěrečné fázi dojde k začlenění stávajících detekcí do nové architektury a k vytvoření detekcí nových.

6 Implementace mikroslužby

Tato kapitola se zabývá klíčovými aspekty realizace detekční mikroslužby, jejíž design byl pečlivě navržen v předchozí kapitole. Jsou zde podrobně rozebrány hlavní principy, které byly při implementaci uplatněny, s důrazem na modulárnost a budoucí rozšiřitelnost aplikace.

Popsány jsou použité technologie a struktura projektu. Součástí je rovněž detailní vysvětlení implementace detekčního procesu, potřebných modelových tříd a databázového schématu. Na konci jsou rozebrány dva další důležité aspekty implementace, a sice testování a dokumentace.

6.1 Principy implementace

V této podkapitole jsou definovány základní principy, které byly během implementace aplikace dodrženy s cílem zajistit její spolehlivost, snadnou udržitelnost a možnost dalšího rozvoje.

- **SOLID principy a čistý kód** – Aplikace je navržena a implementována s ohledem na základní SOLID principy, což podporuje snadnější rozšiřitelnost a refaktoring kódu v budoucnosti. Kromě toho byl kladen důraz na čitelnost a možnost znovupoužití kódu.
- **Oddělení zájmů** – Aplikace je modulární s jasně strukturovanými komponentami, každá s definovanými odpovědnostmi. Tento přístup minimalizuje vzájemnou závislost komponent a usnadňuje jejich čitelnost a možnost znovupoužití.
- **Bezstavovost a škálovatelnost** – Jako klíčový cíl mikroslužební architektury byla zajištěna bezstavovost aplikace pro vyšší škálovatelnost. Absence stavovosti mezi požadavky umožňuje aplikaci efektivně škálovat horizontálně přidáváním dalších instancí bez nutnosti synchronizace stavu.
- **Jednoduchá konfigurace** – Aplikace je jednoduše konfigurovatelná skrze konfigurační soubory, což zjednodušuje její nastavení a flexibilní nasazení v různých prostředích.
- **Dostatečné testování** – Aplikace prošla rozsáhlým testováním s důrazem na kritické části systému, aby bylo zajištěno jejich správné fungování.

- **Komentování kódu** – I přes snahu psát kód tak, aby byl co nejvíce samodokumentující, byl také věnován zvláštní důraz na důkladné komentování tříd a metod, což zvyšuje srozumitelnost kódu pro současné i budoucí vývojáře.

6.2 Použité technologie

Samotná implementace mikroslužby využívá technologie, které byly pečlivě vybrány na základě předchozí analýzy. Následuje stručný přehled hlavních technologií a jejich konkrétních verzí použitých při implementaci:

- Java – verze 17,
- Spring Boot – verze 3.2.0,
- MySQL – verze 8.0.29.

6.3 Struktura projektu

Pro správu řízení sestavení projektu byl použitý nástoj Maven. Struktura projektu tedy odpovídá standardní adresářové struktuře Maven projektu, která zahrnuje základní adresáře `src/main/java` pro zdrojové kódy aplikace, `src/main/resources` pro konfigurační soubory a `src/test/java` pro testovací kódy. Dále je přítomen konfigurační soubor `pom.xml`, který obsahuje informace například o závislostech a pluginech projektu a také adresář `db` obsahující soubory pro obnovu databáze.

V kontextu rozložení balíků tříd uvnitř složky se zdrojovými soubory (konkrétně `src/main/java/cz/fav/kiv/detector`) je použita standardní struktura, která se obvykle využívá pro Spring projekty. Tato struktura zahrnuje oddělené balíky pro kontroléry, modelové třídy, servisní třídy a další. Kompletní rozvržení balíků tříd je následující:

- **config** – konfigurační třídy,
- **controller** – kontroléry přijímající požadavky,
- **exception** – výjimky,
- **model** – modelové třídy,
- **repository** – přístup k datům,

- **service** – servisní třídy obsahující logiku aplikace,
- **utils** – pomocné utility třídy.

6.4 Architektura aplikace

Implementovaná architektura aplikace je odvozena ze dvou klíčových zdrojů: z komponentového návrhu prezentovaného v předchozí kapitole a ze standardních postupů používaných v Spring Frameworku, který do určité míry předurčuje strukturu architektury.

Architektura mikroslužby je ilustrována na obrázku 6.1. Interakce s aplikací začíná příchozím požadavkem z aplikace SPADe, který je následně předán kontrolérům. Tyto kontroléry zodpovídají za delegaci zpracování požadavků na servisní vrstvu, jež je nositelem aplikační obchodní logiky, včetně samotného algoritmu pro provedení detekce.

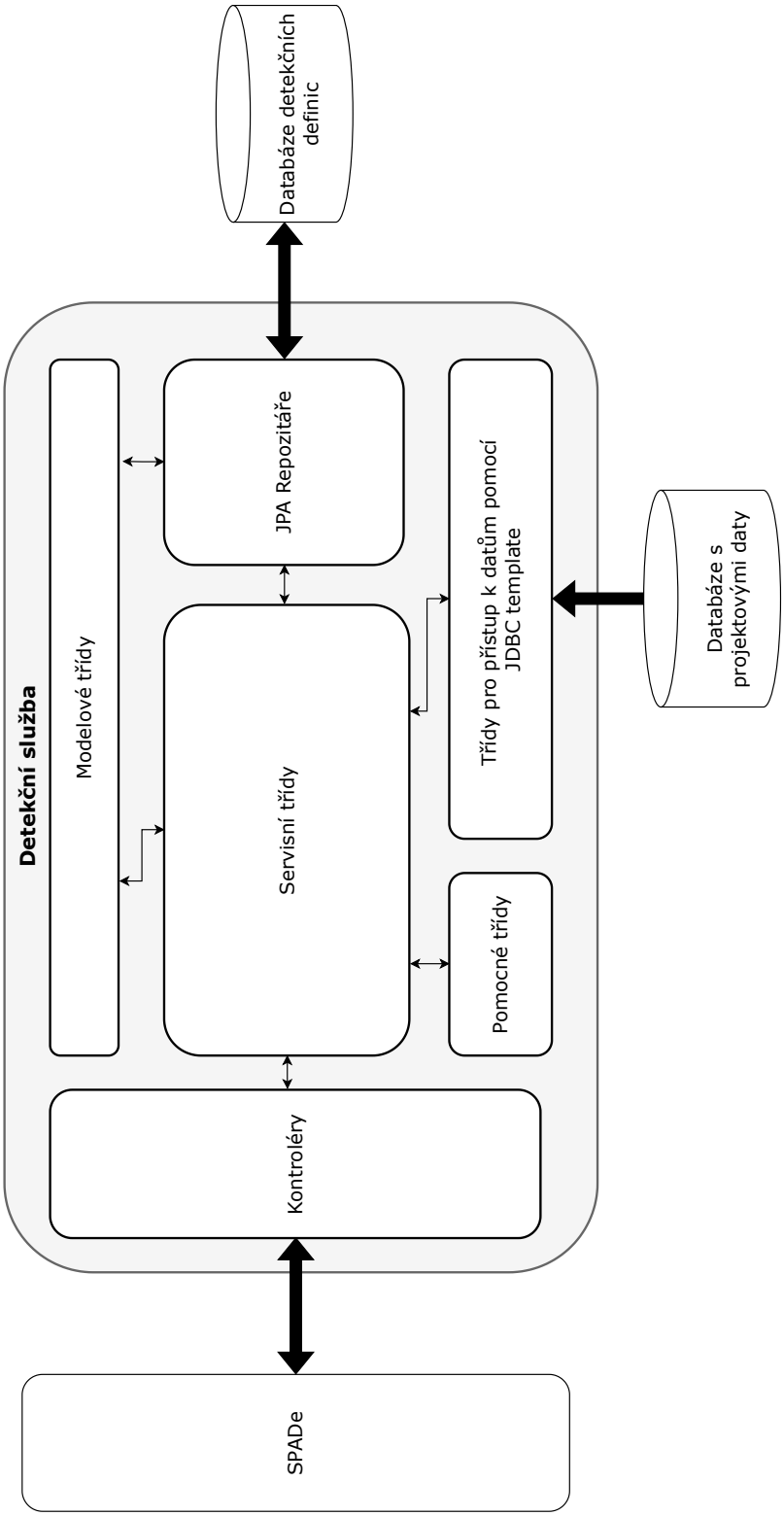
Servisní vrstva dynamicky interaguje s modelovými třídami, které reprezentují abstrakci entit, jako jsou metriky a indikátory. Persistenci těchto entit zajišťují Java Persistence API (JPA) Repozitáře, jež komunikují s databází definic detekcí skrze Java Persistence API, která slouží jakožto primární úložiště aplikačních dat.

Významnou částí servisní vrstvy jsou třídy zodpovědné za vyhodnocení metrik, které si vyžadují přístup k sekundárnímu zdroji dat obsahujícímu informace o projektech. Tento přístup je realizován použitím Java Database Connectivity Template (JDBC), což umožňuje robustní a flexibilní manipulaci s daty. Navíc servisní vrstva využívá pomocné třídy, které poskytují sdílené metody pro často opakované operace.

Po zpracování požadavku servisní vrstvou je odpověď prostřednictvím kontrolérů předána zpět do aplikace SPADe. Tímto způsobem architektura podporuje čisté oddělení odpovědností, což usnadňuje údržbu a rozšiřitelnost aplikace.

6.5 Modelové třídy

Modelové třídy představují nejen abstrakci databázových entit, ale také základní stavební prvky pro přenos a manipulaci s daty v rámci systému. Tyto třídy jsou zásadní pro mapování obchodní logiky na databázové schéma a zároveň poskytují robustní strukturu pro přenos dat mezi různými vrstvami aplikace.



Obrázek 6.1: Architektura detekční mikroslužby

Entitní modelové třídy jsou specificky navrženy pro reprezentaci fenoménů popsaných v předchozích kapitolách. Například, třída `Metric` je zásadní pro reprezentaci metrik a zahrnuje nejen základní popisné atributy jako název a popis, ale také řetězec obsahující databázový dotaz, který je používán pro získání a agregaci dat z projektové databáze.

Dalším z klíčových modelů je třída `Indicator`, která je strukturována tak, aby reflektovala indikátory a jejich důležité aspekty. Tato třída obsahuje základní charakteristiky, jako jsou název a popis, a navíc podrobnosti o typu indikátoru představovaného pomocí modelové třídy `IndicatorType`, tedy jinými slovy, zda se jedná o anti-vzor, špatnou praktiku či pouhý indikátor. Dále je přítomen atribut pro samotný skript, které obsahuje logiku pro detekci indikátorů. Jeho typ je reprezentován jako atribut typu `ScriptType`, což umožňuje identifikovat programovací jazyk skriptu. Tím v současné verzi může být pouze Groovy, nicméně díky rozšiřitelné architektuře lze systém snadno doplnit o další jazyky, včetně vlastní gramatiky.

Nezbytnou součástí je i třída modelující parametry pro tyto indikátory, tedy třída `Parameter`. Ta popisuje parametr s atributy zahrnujícími název, popis, identifikátor spojeného indikátoru, datový typ parametru představovaný třídou `ParameterType` a výchozí hodnotu. Mezi implementované typy parametrů se řadí text, číslo, procento a ne/pravda, nicméně tento výčet může být v budoucnu snadno rozšířen.

V rámci procesu detekce, byla implementována třída `DetectionRequest`, která jako atributy nese seznam indikátorů určených k detekci, sadu jejich parametrů a seznam projektů, na nichž má být detekce prováděna. Na druhé straně procesu figuruje třída `ScriptExecutionResult` (viz Příloha D), která poskytuje rámec pro uložení výsledků detekce, obsahující údaje o úspěšnosti provedení, konkrétní výsledky detekce a dodatečné informace pro prezentaci uživatelům, například mezivýsledky či hodnoty odůvodňující výsledky.

Vzhledem k potřebě zefektivnit vývoj a zvýšit přehlednost kódu, byla začleněna knihovna *Lombok*. Tato knihovna umožnila minimalizovat opakující se kód spojený s definicí metod jako jsou gettery, settery a konstruktory, prostřednictvím anotací, které tyto funkce generují automaticky. Díky tomu byly modelové třídy udrženy konzistentní a čitelné, zatímco byl zjednodušen proces vývoje a udržování kódu.

6.6 Provedení detekcí

Celý proces detekce začíná příchodem požadavku strukturovaného v objektu `DetectionRequest`. Tento objekt obsahuje seznam indikátorů, jejich para-

metry a seznam projektů, na kterých mají být tyto indikátory vyhodnoceny. Detekce probíhá iterativně ve dvou zanořených smyčkách: první iteruje přes projekty a druhá, vnořená, iteruje přes indikátory. Pro každou kombinaci indikátoru a projektu je ze třídy `DetectionService` zavolán specifický proces detekce.

Vyhodnocení indikátorů a metrik

Proces detekce začíná vykonáváním skriptu indikátoru s vloženými odpovídajícími parametry z příchozí sady. Skript je spouštěn pomocí třídy implementující rozhraní `ScriptExecutor`, která je specifická pro jazyk, v němž je skript napsán. V současném řešení je implemenován pouze jazyk Groovy prostřednictvím třídy `GroovyScriptExecutor`. Nicméně díky přítomnosti rozhraní může dojít ke snadnému rozšíření o další, nejen skriptovací, jazyky, nebo dokonce o třídy zpracovávající vlastní gramatiku.

Skript může během svého provádění volat vyhodnocení dalších indikátorů či metrik pomocí již vložených objektů a jejich metod. Konkrétní provedení v případě indikátoů je pomocí metody:

```
scriptExecutor.executeScript(<indicatorName>, <params>,
                             <projectId>);
```

Volané indikátory jsou zpracovány rekurzivně a asynchronně, přičemž výsledky jsou vráceny do volajícího skriptu a dále zpracovávány.

Volané metriky jsou rovněž asynchronně vyhodnocovány, a to pomocí metody:

```
metricExecutor.executeMetric(<metricName>, ...<params>);
```

Metoda pro danou metriku vykoná odpovídající databázový dotaz na zvoleném projektu a výsledky vrátí zpět do skriptu.

Celý skript je takto postupně zpracován a výsledky jsou zabaleny a vráceny v objektu `ScriptExecutionResult`.

Konkrétní příklad skriptu indikátoru je uvedený v příloze E.

Předávání parametrů

Parametry jsou předávány do skriptu z externí sady, která je součástí požadavku na detekci. Vnitřní skripty, které jsou volány hlavním skriptem, přebírají parametry přímo z něj. Pokud autor hlavního skriptu požaduje specifické parametrizace pro vnitřní skripty, je nutné, aby tyto parametry byly

definovány už v hlavním skriptu. Parametry jsou distribuovány do vnitřních skriptů podle jejich názvů.

Speciálním parametrem důležitým zejména pro databázové dotazy metrik je identifikátor právě analyzovaného projektu. Tento parametr je pro usnadnění potřeb uživatelů vkládán do každého indikátoru automaticky. Uživatel vytvářející kód skriptu indikátoru tak může tento parametr libovolně používat, aniž by byl povinný jej předem deklarovat.

6.7 Komunikace s okolím

Detekční mikroslužba je navržena tak, aby komunikovala s vnějšími systémy a databázemi prostřednictvím bezpečných a efektivních protokolů.

Komunikace s aplikací SPADe

Mikroslužba komunikuje s aplikací SPADe prostřednictvím HTTP protokolu, kde využívá standardní HTTP kódy pro signalizaci stavů operací. Tato komunikace umožňuje výměnu požadavků a odpovědí mezi službou a aplikací a je základem pro interakci s uživatelským rozhraním.

Komunikace s databázemi

Komunikace s databázovým úložištěm pro detekční definice je realizována pomocí Spring Data JPA, což je rozšíření standardního Java Persistence API (JPA). Toto propojení, zajištěné pomocí tříd `Repository` a anotací modelových tříd, umožňuje efektivní čtení i zápis dat.

Pro projektová data je zvoleno připojení pomocí JDBC template, což umožňuje provádění specifikovaných dotazů s parametrizací. Tento přístup byl zvolen kvůli složitosti databázového modelu, kde parametrizované dotazy zajišťují flexibilitu a zároveň bezpečnost proti útokům typu SQL-injection. JDBC template automaticky řetězcově escapuje všechny parametry, čímž zabraňuje vykonávání potenciálně škodlivého kódu. Databáze uchovávající projektová data je konfigurována pouze pro čtení, což je specifikováno nastavením `spring.projects-datasource.readOnly=true`, které se nachází v souboru `application.properties`, aby uživatelé neměli možnost data jakkoliv modifikovat.

6.8 Databázové schéma

Tato kapitola se věnuje podrobnému popisu databázového schématu, které je součástí databáze pro detekčních definic. Podrobný Entity–Relationship–Attribute (ERA) diagram, generovaný pomocí nástroje MySQL Workbench, je zobrazen na přiloženém obrázku 6.2 a poskytuje vizuální reprezentaci databázové struktury.

Hlavními prvky schématu jsou tabulky `metrics` a `indicators`, přičemž každá hraje klíčovou roli ve fungování mikroslužby. Tabulka `metrics` je samostatná a neobsahuje žádné přímé relace s ostatními tabulkami. Naopak, tabulka `indicators` je úzce propojena s dalšími tabulkami. Pro každý indikátor mohou existovat jeden či více parametrů, což je reprezentováno vazbou na tabulku `parameters`, která obsahuje odkaz na identifikátor indikátoru formou cizího klíče. Typ každého parametru je definován vztahem k tabulce `parametertype`.

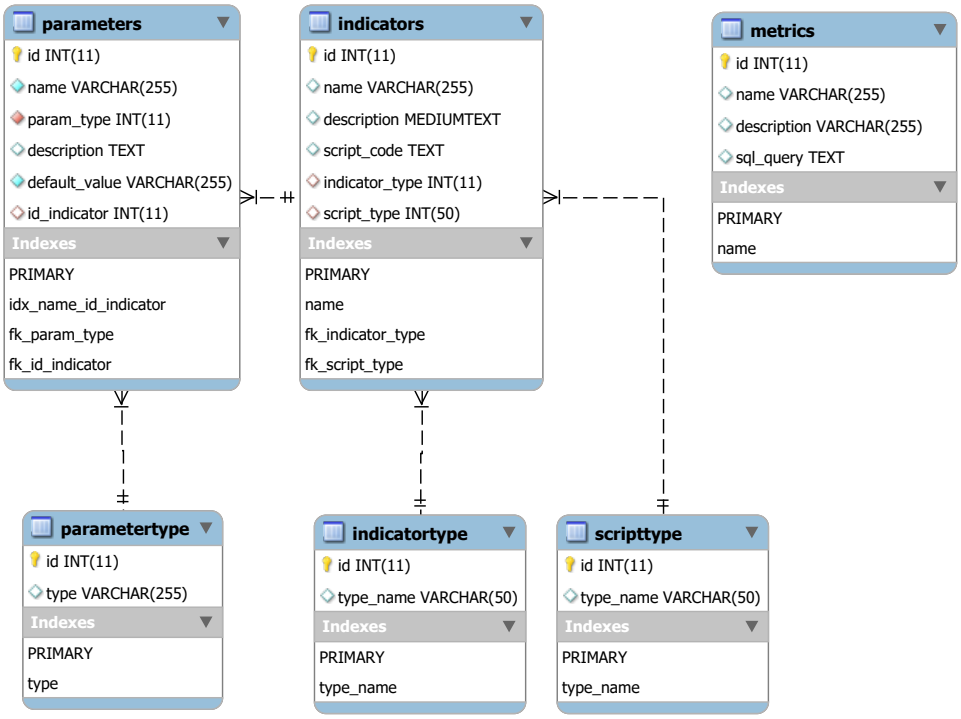
Dále tabulka `indicators` zahrnuje cizí klíče odkazující na dvojici tabulek `indicatortype` a `scripttype`. Tabulka `indicatortype` klasifikuje indikátory do kategorií jako anti-vzory nebo špatné praktiky, zatímco `scripttype` určuje možné programovací jazyky pro skripty indikátorů.

Je důležité poznamenat, že vazba mezi metrikami a indikátory není v databázi přímo reprezentována, což je důsledek rozhodnutí nezakládat explicitní vztah M:N v databázi, ale zachovat tuto informaci v textu skriptů. Tím odpadá proces složitého parsování těl skriptů po každé změně a dohledávání metrik s následným zanášením vztahů do tabulky. Ačkoliv by vazba mezi metrikami a indikátory mohla být v budoucnu využívána, v současném stadiu aplikace nepřináší významné přínosy.

Kromě toho detekční mikroslužba interaguje s databází projektových dat. Tato komunikace využívá stávající datový model, který zůstal během vývoje této práce nezměněn. Struktura tohoto modelu je ilustrována na obrázku 3.3, který poskytuje přehled o doménovém modelu použitém v rámci mikroslužby.

6.9 Testování aplikace

Kritickým prvkem vývoje mikroslužby je její důkladné testování. Aby bylo zajištěno, že všechny komponenty fungují správně, byla implementována sada automatizovaných jednotkových a integračních testů. Navíc bylo během vývoje průběžně prováděno akceptační testování ve spolupráci s vedoucím této práce.



Obrázek 6.2: Databázové schéma databáze pro detekční definice

6.9.1 Jednotkové testování

Jednotkové testy byly implementovány s použitím frameworku JUnit a mockovací knihovny Mockito pro simulaci externích závislostí. Celkem bylo napsáno 100 jednotkových testů, které pokrývají 91% kódu mikroslužby, s 98% pokrytím pro servisní vrstvu obsahující klíčovou obchodní logiku. Tyto testy jsou automaticky spouštěny během Maven fáze `test` pomocí *maven-surefire-plugin* a následují jmennou konvenci `*Test.java`. Veškeré jednotkové testy jsou zároveň nezávislé na připojení k databázím.

6.9.2 Integrační testování

Integrační testování bylo zavedeno pro ověření správnosti komunikace aplikace s jejím okolím, tedy s oběmi databázemi. Nejprve byly vytvoření testy pro ověření konfiguračních souborů pro obě databáze, respektive pro ověření, zda došlo k připojení a správné inicializaci všech komponent potřebných pro komunikaci s databázemi.

K ověření JPA komunikace byla využita in-memory H2 databáze, která emuluje produkční databázové prostředí. Tyto testy se zaměřují na třídy v balíčku `repository`. V rámci fáze integračního testování bylo vytvořeno 23

testů, které se spouštějí v Maven fázi `verify` pomocí *maven-failsafe-plugin* pluginu, s testy pojmenovanými podle konvence `*IT.java`.

6.9.3 Akceptační testování

Akceptační testování bylo prováděno ve spolupráci s vedoucím práce a bylo zaměřeno na ověření, že aplikace splňuje všechny požadavky a představy. Během těchto testů byly pravidelně zaznamenávány a zapracovávány připomínky, což zajistilo, že konečný produkt odpovídá očekávání.

6.10 Dokumentace

Pro zajištění snadné orientace ve zdrojovém kódu mikroslužby byla implementována dokumentace využívající standard JavaDoc. Tato dokumentace je generována přímo z dokumentačních komentářů nad třídami, jejich metodami a atributy a poskytuje ucelený přehled o funkcionalitě jednotlivých komponent. Kompletní JavaDoc dokumentace je součástí projektu s mikroslužbou a nachází se ve vlastním adresáři s názvem `doc`.

K dokumentaci API byl aplikován framework Swagger, který je založen na standardu OpenAPI a poskytuje uživatelsky přívětivé grafické rozhraní pro prohlížení všech dostupných endpointů aplikace. Swagger umožňuje nejen zobrazit detaily a strukturu každého endpointu, ale také poskytuje funkci interaktivního testování jednotlivých volání přímo z prohlížeče. Díky tomu mohou vývojáři i koncoví uživatelé snadno experimentovat s funkcemi API a ověřovat jejich chování. Dokumentace Swagger je přístupná po spuštění služby na URL adrese:

```
http://<ip-adresa/doménové-jméno>:<port>/swagger-ui/index.html
```

7 Integrace mikroslužby do frameworku SPADe

Tato kapitola se věnuje detailnímu popisu procesu integrace nově vyvinuté detekční mikroslužby do stávajícího softwarového ekosystému SPADe. Vzhledem k tomu, že SPADe je rozsáhlý nástroj používaný pro analýzu softwarových projektů, klíčovým aspektem této integrace je zajištění, že nová mikroslužba se systémem efektivně komunikuje a uživatelé ji budou moci jednoduše využívat při svých analýzách.

Kapitola postupně pokrývá přípravu aplikace SPADe pro integraci mikroslužby, detailní popis změn v uživatelském rozhraní a samotné nastavení komunikace v rámci integrace. Závěr je věnován popisu integračního testování pro ověření komunikace s mikroslužbou.

7.1 Příprava aplikací před integrací

V rámci této práce, probíhající současně s jinými akademickými projekty, byla nutná koordinace a synchronizace vývoje. Jedním z těchto projektů byla modifikace architektury aplikace SPADe, kde hlavní změnou bylo přesunutí uživatelského rozhraní, implementované pomocí šablonovacího nástroje Thymeleaf, do samostatné aplikace v Reactu pojmenované SPAWn. Tato aplikace byla během trvání diplomové práce ve vývojové fázi. Změny byly prováděny i na aplikaci SPADe, což vyžadovalo pečlivé sledování všech úprav v průběhu těchto projektů, aby byla integrace nově vyvinuté detekční služby do nástrojové sady SPADe co možná nejhladší. Přesto bylo pro zjednodušení integrace a podporu budoucích vývojových aktivit nezbytné provést určité úpravy stávajících aplikací a databází.

Prvním důležitým krokem bylo odstranění původní detekční logiky v aplikaci SPADe, která zahrnovala definice detektorů, provádění detekcí, nastavení prahových hodnot a další související funkce. Rozhraní a všechny implementované detektory byly z aplikace odstraněny, stejně jako relevantní servisní třídy a nepotřebné části kontroléru. Dále došlo k reorganizaci adresářové struktury a eliminaci adresáře `v2`, který měl původně oddělit nový vývoj od stávajícího. Soubory z tohoto adresáře byly přesunuty do příslušných balíků, čímž se aplikace SPADe dostala do konzistentního stavu a byla očištěna od předchozí detekční logiky.

Také v uživatelském rozhraní aplikace SPAWn bylo nutné provést odstranění některých funkcí. Vzhledem k tomu, že aplikace byla stále ve vývoji, dosud neposkytovala uživatelské rozhraní pro všechny detekční funkce, například pro detailní zobrazení výsledků detekcí či úpravy detailních informací o detektorech. Ostatní detekční funkce byly z uživatelského rozhraní odstraněny, včetně stránek pro zadávání detekcí, nastavení prahových hodnot, ukládání souborů s prahovými hodnotami a zjednodušeného zobrazení výsledků.

Poslední úprava před integrací se týkala změny databázové technologie pro databázi aplikace SPAWn. Původně používaná databáze Microsoft SQL Server byla v důsledku fragmentace mezi různými databázovými technologiemi a komplexního nasazení po konzultaci s vedoucím této práce převedena do nového schématu v rámci stávající MySQL databáze, což výrazně zjednodušilo budoucí proces nasazení.

7.2 Tvorba uživatelského rozhraní

Vzhledem k tomu, že původní uživatelské rozhraní aplikace SPAWn nebylo kompatibilní s funkcemi nově vyvinuté detekční služby a zároveň se lišilo v logice detekce a definici fenoménů, bylo nutné vytvořit v aplikaci SPAWn zcela nové uživatelské rozhraní. To bylo navrženo tak, aby přesně vyhovovalo potřebám mikroslužby a zároveň poskytovalo maximální uživatelský zážitek a funkčnost.

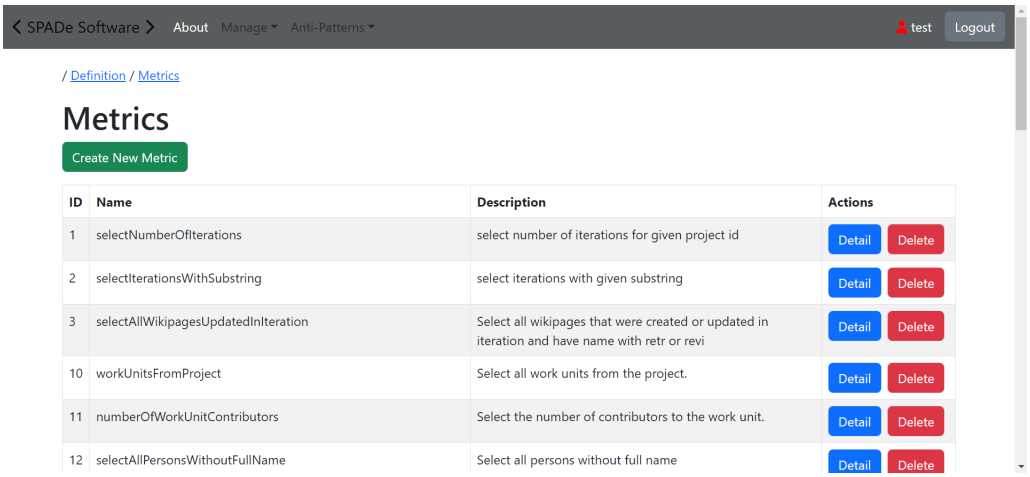
Nový vývoj v rámci SPAWn respektoval stávající konvence, jako jsou struktury adresářů, volání API a obecný vizuální styl aplikace. Dále byly do aplikace implementovány nové obrazovky a funkcionality, které umožňují plné využití možností detekční mikroslužby. Tento vývoj můžeme rozdělit do dvou hlavních oblastí: definice a detekce. V oblasti definice se jedná o nástroje umožňující tvorbu a úpravu jednotlivých fenoménů, zatímco oblast detekce se zabývá spouštěním detekčních procesů a vizualizací výsledků.

Definice

V rámci oblasti definice (Definition) bylo nutné navrhnout obrazovky umožňující komplexní správu metrik a indikátorů. Tato funkcionalita zahrnuje možnosti pro zobrazení seznamu existujících fenoménů, jejich vytváření, úpravy a odstraňování.

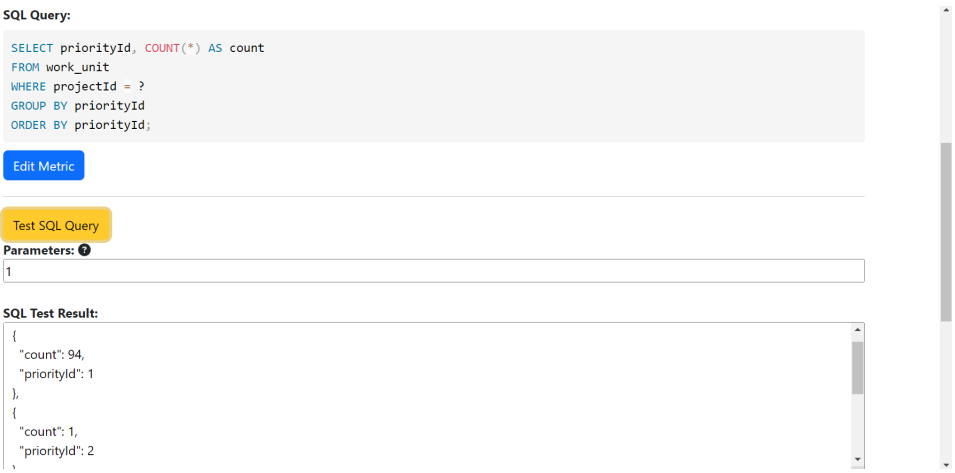
Na obrázku 7.1 je zobrazen snímek obrazovky pro správu metrik, kde uživatelé najdou seznam již vytvořených metrik. Na této stránce mají také možnost vytvářet nové metriky, odstraňovat existující a přecházet na de-

taily konkrétních metrik. Obrazovka pro správu indikátorů je navržena analogicky.



Obrázek 7.1: Obrazovka správy metrik

Detailní zobrazení jednotlivých metrik a indikátorů poskytuje přístup k základním informacím, které je možné upravovat a ukládat. Specifickým prvkem v detailu metrik je pole pro zadání SQL dotazu, jehož syntaxe je zvýrazněna pro lepší orientaci. Uživatelé mohou z této obrazovky dotaz také přímo spustit, a to i s možností zadání libovolných parametrů v případě parametrizovaných dotazů. Tím je možné otestovat jeho funkčnost nad projektovou databází. Tato část obrazovky detailu metriky je ukázána na obrázku 7.2.



Obrázek 7.2: Test SQL dotazu na obrazovce metriky

Na obrazovce detailu indikátoru je možné přidávat parametry, které jsou zobrazovány před spuštěním detekce, což uživatelům umožňuje lépe pocho-

pit, které parametry je třeba zadat, jaký mají datový typ a jaká je jejich výchozí hodnota. Kromě toho je zde umožněno vytváření skriptů, zatím pouze v jazyce Groovy. Syntaxe tohoto jazyka je po uložení editace zvýrazněna, jak je ilustrováno na obrázku 7.3. To přispívá k lepší orientaci při tvorbě skriptu.

Obrázek 7.3: Zvýraznění syntaxe na obrazovce detailu indikátoru

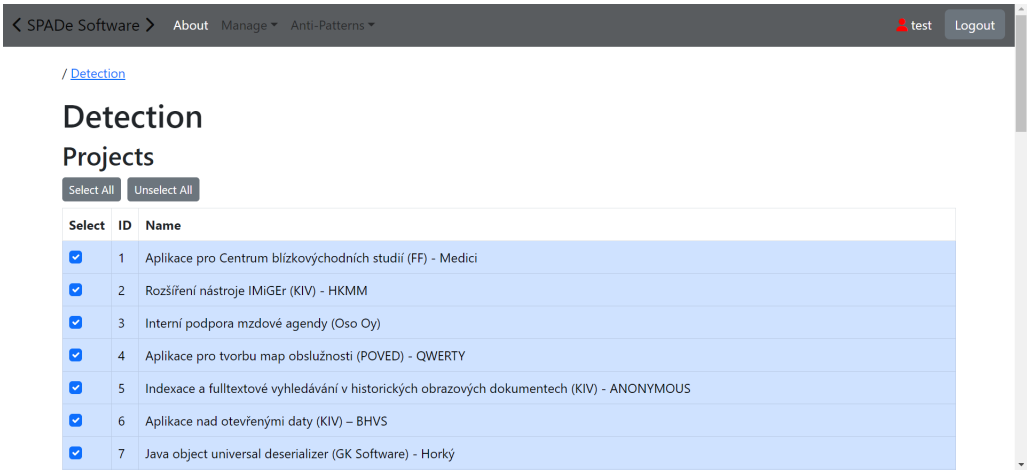
Detekce

V rámci detekční oblasti (Detection) bylo vytvořeno několik obrazovek usnadňujících zadání požadavků na detekce a zobrazení výsledků již provedených detekcí.

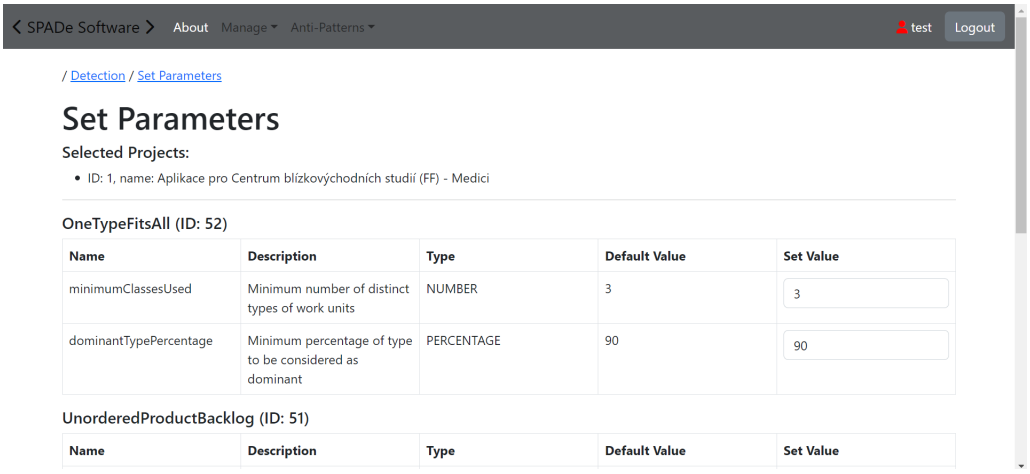
Proces začíná na obrazovce, kde uživatelé vidí seznam dostupných projektů v projektové databázi spolu se seznamem implementovaných indikátorů. Indikátory je možné filtrovat, například za účelem výběru pouze antivzroů. Pro oba seznamy byla vytvořena tlačítka umožňující snadné označení nebo odznačení všech položek. Tato obrazovka je znázorněna na obrázku 7.4.

Následuje obrazovka, na které se zadávají parametry pro vybrané indikátory. Jsou zde zobrazeny veškeré parametry pro indikátory, které byly vybrány v předchozím kroku. Parametry jsou nastaveny na výchozí hodnoty, ale uživatelé mají možnost tyto hodnoty upravit. Příklad této obrazovky se nachází na obrázku 7.5.

Poté, co je požadavek uživatelem odeslán, zobrazují se výsledky detekce. Byly implementovány tři různé pohledy na získané výsledky. Prvním z nich je maticové zobrazení, které poskytuje přehlednou tabulku s kombinacemi indikátorů a projektů a příslušných výsledků. To je pro názornost ilustrováno na obrázku 7.6. Dva další pohledy seskupují výsledky dle projektů a dle in-

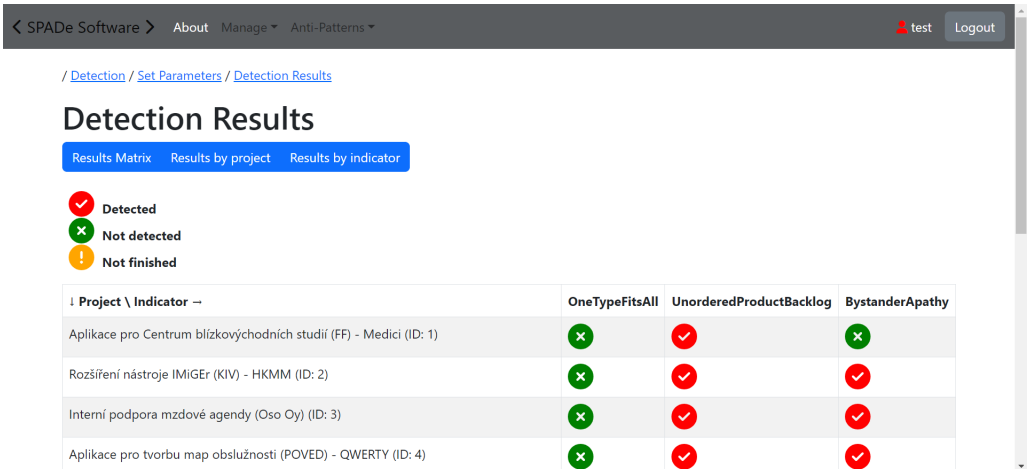


Obrázek 7.4: Obrazovka pro výběr entit k detekci



Obrázek 7.5: Obrazovka pro nastavení parametrů

dikátorů. Možnost přepínání mezi těmito zobrazeními usnadňuje uživatelům další analýzu a práci s výsledky.



Obrázek 7.6: Maticové zobrazení výsledků detekce

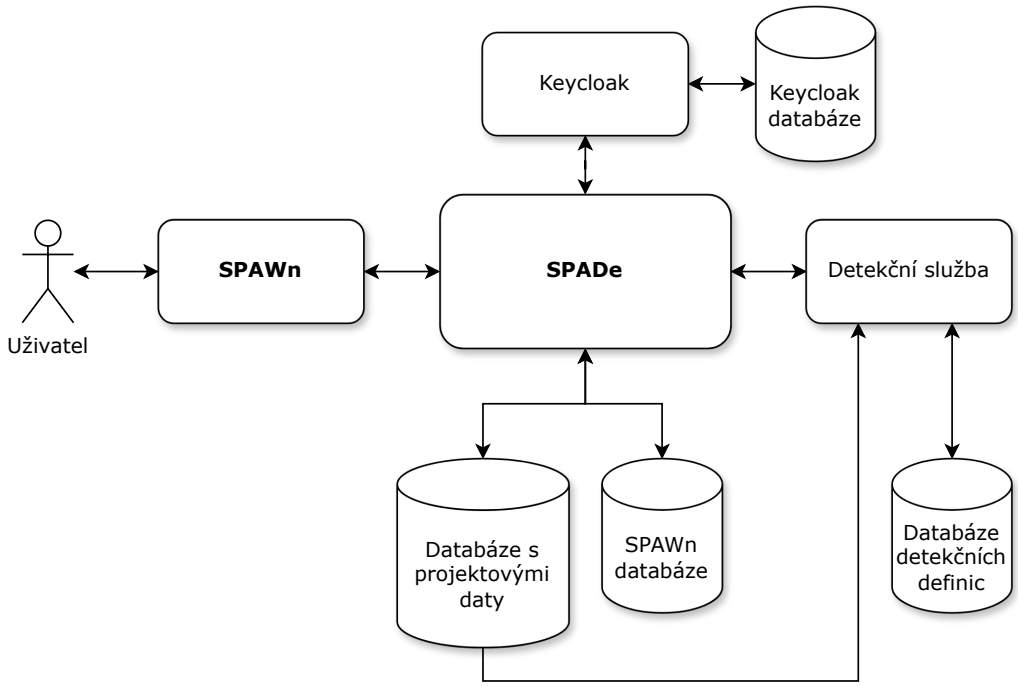
7.3 Integrace mikroslužby

Detekční mikroslužba byla začleněna do stávajícího detekčního segmentu nástroje SPADe, jak bylo navrženo v předchozích kapitolách. Začlenění je ilustrováno na obrázku 7.7. Tato mikroslužba nyní komunikuje výhradně s aplikací SPADe. Tato integrace byla realizována tak, že SPADe nyní funguje jako prostředník mezi uživatelským rozhraním a detekční mikroslužbou. Jednou z klíčových funkcí, které SPADe přidává, je zajištění autentizace a autorizace všech požadavků pocházejících z uživatelského rozhraní.

Pro tyto účely byl do aplikace SPADe přidán nový kontrolér s endpointem `detecting/**` a přidružená servisní třída. Tyto komponenty jsou navrženy tak, aby přijímaly požadavky z uživatelského rozhraní a po jejich autentizaci a autorizaci je následně přeposílaly detekční mikroslužbě. Odpověď od mikroslužby je poté s příslušným HTTP kódem odesílána zpět do uživatelského rozhraní, kde je možné ji dále zpracovat. V případě, že mikroslužba není dostupná, aplikace SPADe vrátí odpověď se standardním HTTP kódem 500.

7.4 Testování integrace

Tato podkapitola popisuje ověřování správnosti integrace a přívětivosti uživatelského rozhraní.



Obrázek 7.7: Detekční mikroslužba v rámci systému SPADe

7.4.1 Integrační testování

Do aplikace SPADe bylo přidáno integrační testování, které zaručí, že komunikace s detekční mikroslužbou probíhá podle očekávání. Konkrétně byly přidané 3 integrační testy, přičemž dva z nich posílají požadavky na existující endpointy běžící mikroslužby a kontrolují, zda se vrátí odpověď s HTTP kódem 200. Zbylý test zasílá požadavek na neexistující endpoint v rámci mikroslužby a testuje, zda ze služby dorazí odpověď s HTTP kódem 404.

I v tomto případě byl pro integrační testování využití standardní Maven plugin *maven-failsafe-plugin*, který spouští testy v Maven fázi `verify`. Tyto testy jsou pojmenovány podle konvence pluginu, tedy `*IT.java`.

7.4.2 Uživatelské testování

Proběhlo také uživatelské testování, a to s cílem ověřit uživatelskou přívětivost implementovaného rozhraní a zkontrolovat, zda původní funkcionalita systému před nasazením nové mikroslužby zůstává nezměněna a plně funkční. Testování se zúčastnili tři testeři, kteří měli předchozí zkušenosti s vývojem aplikací SPADe a uživatelským rozhraním SPAWn a dva další testeři, kteří s tímto systémem nebyli předtím seznámeni. Zkušení testeři byli pověřeni především ověřením stávající funkčnosti a hodnocením nového

uživatelského rozhraní z hlediska detekčních potřeb. Nově příchozí testeři nejprve obdrželi instruktáž o systému a následně testovali jeho uživatelské rozhraní.

V průběhu uživatelského testování byla potvrzena nejen správná funkčnost existujících funkcionalit, ale také byla shromážděna pozitivní zpětná vazba ohledně intuitivnosti nově implementovaného uživatelského rozhraní. Z testování také vyplynuly drobné připomínky k úpravám, jako je například zavedení možnosti přímého přesměrování na detailní stránku indikátoru ze seznamu projektů a indikátorů pro detekci. Na základě těchto připomínek byly do systému implementovány příslušné úpravy.

8 Převod detektorů do nového řešení

Aby bylo možné plně využívat nově vyvinutý detekční mechanismus realizovaný prostřednictvím implementace nové mikroslužby, je nezbytné do systému integrovat relevantní data. Tato data zahrnují indikátory, respektive anti-vzory, a tedy i jejich detektory.

Tato kapitola se zaměřuje na přenos detektorů anti-vzorů, které byly vyvinuty a implementovány v předchozí architektury, do databáze nového systému. Kromě převodu stávajících detektorů budou identifikovány a implementovány také nové anti-vzory a jejich detekce. Tento krok umožní ověřit modularitu systému a rozšířit funkční schopnosti nástroje SPADe.

8.1 Původní detektory

V původní verzi aplikace SPADe bylo implementováno deset detektorů anti-vzorů a špatných praktik. Tyto detektory byly převedeny do nového zápisu v rámci nové architektury, což znamenalo jejich transformaci do skriptovacího jazyka Groovy. Logika detekce těchto původně implementovaných detektorů byla již dříve důkladně prozkoumána, proto nebyla potřeba další kontrola ani úprava této logiky při jejich převodu. Nicméně, bylo klíčové této logice porozumět, a to prostřednictvím seznámení se s pracemi, ve kterých byly původně vytvořeny [30][31].

Konverze zahrnovala přepsání logiky detektorů do Groovy skriptů indikátorů označených jako anti-vzory, přičemž SQL dotazy byly integrovány jako nové metriky, které jsou volány v rámci těchto skriptů. Prahové hodnoty, které byly původně nastavitelné, byly začleněny jako parametry indikátorů. Tímto způsobem bylo převedeno všech deset detektorů, mezi něž patří:

- Business As Usual,
- Long Or Non-Existant Feedback Loops,
- Ninety-Ninety Rule,
- Road To Nowhere,
- Specify Nothing,

- Too Long Sprint,
- Varying Sprint Length,
- Unknown Poster,
- Bystander Apathy,
- Yet Another Programmer.

Nová struktura skriptů výrazně zlepšila přehlednost kódu. Například v případě metrik byly v původní implementaci nejprve vypsané názvy souborů s uloženými dotazy a poté z detektoru volány jednotlivé SQL dotazy přímo z databázového připojení, což následně vyžadovalo iteraci přes výsledky a výběr relevantních dat. Původní implementace, zahrnující volání dotazů a následné zpracování výsledků, tedy byla v případě zápisu poměrně komplexní a méně efektivní. Nový způsob implementace metrik v Groovy skriptu je však mnohem čitelnější a uživatelsky přívětivější, což zjednodušuje jak psaní, tak revizi kódu. Díky této změně je celý proces tvorby, kontrolního přezkoumání a úpravy kódu rychlejší a efektivnější.

Správnost převodu původních detektorů do nové architektury byla ověřena srovnáním nově získaných výsledků spolu s výsledky získanými v rámci původní architektury na stejné množině projektových dat. Ve všech případech došlo ke shodě výsledků, tudíž je možné nový zápis původních detektorů považovat za věrohodný.

8.2 Nové detektory

Do nově navržené architektury s detekční mikroslužbou byly vybrány nové indikátory, respektive anti-vzory pro implementaci jejich detektorů. Cílem tohoto výběru bylo nejen rozšířit schopnosti frameworku SPADe v detekování širší množiny fenoménů, ale také ověřit novou architekturu implementovanou v rámci této práce. Z toho důvodu byly k implementaci záměrně vybrané anti-vzory, které v logice své detekce obsahují další vnořené anti-vzory. Tímto přístupem bude postupně využita a ověřena možnost modularity. Tedy možnost vytvářet indikátory a metriky jako samostatné fenomény a umožnit jejich kompozici do širších celků, kterými mohou být například anti-vzory či špatné praktiky.

Důležité je zmínit, že nové anti-vzory a jejich detekce nebudou navrženy striktně korektně či kompletně. Cílem není nutně vytvořit komplexní detekci zahrnující všechny možné faktory, ale spíš ověřit již zmíněnou modularitu.

8.2.1 Analýza vybraných anti-vzorů

V rámci této sekce jsou blíže prozkoumány vybrané anti-vzory k implementace jejich detekce. Dále je navržena logika této detekce a také je provedeno zhodnocení výsledků těchto detekcí na vybrané množině projektových dat. Konkrétně se jedná o anti-vzory Unordered Product Backlog, One Type Fits All, Backlog not DEEP.

Unordered Product Backlog

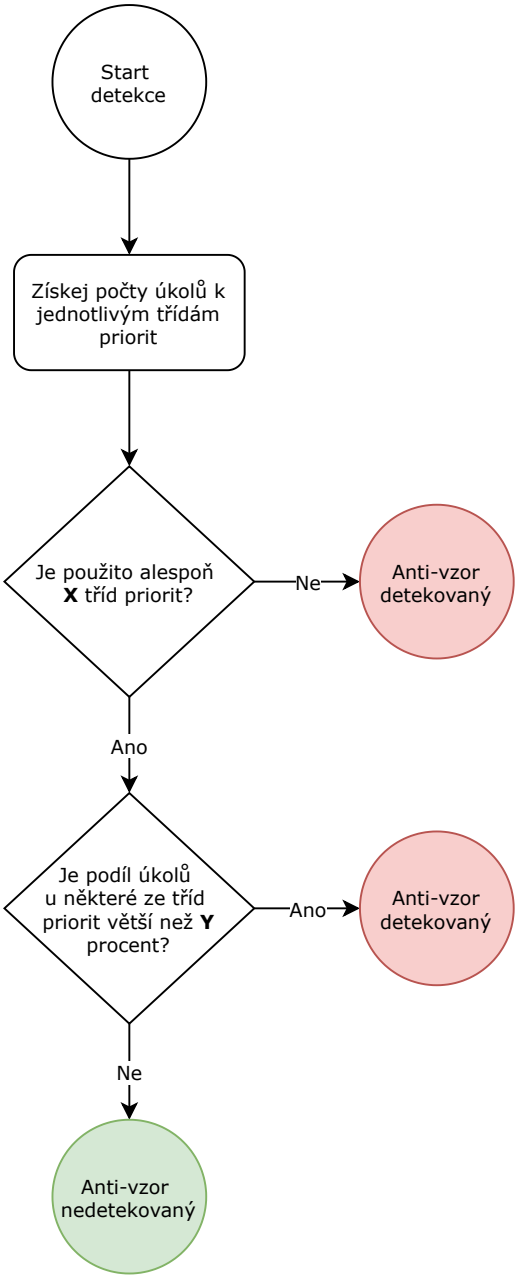
Anti-vzor Unordered Product Backlog poukazuje na problémy spojené s neuspořádaným produktovým backlogem, což je seznam úkolů nutných k úspěšnému vývoji softwarového produktu. Klíčovým problémem tohoto anti-vzoru je nedostatečná diferenciace priorit mezi úkoly, což může způsobit nejistotu ohledně pořadí, v jakém by měly být úkoly řešeny. Například kritické úkoly, jako jsou opravy závažných chyb, se mohou objevit vedle méně důležitých úkolů, jako jsou nepodstatná vylepšení, která nejsou bezprostředně požadována zákazníkem [15]. Správná prioritizace a jasné rozlišení naléhavosti úkolů jsou zásadní pro efektivní rozdělení zdrojů, plánování a provedení úkolů, což výrazně ovlivňuje efektivitu a plynulost vývojových procesů.

Logika pro detekci tohoto anti-vzoru využívá dvě klíčové metriky:

- **Počet úrovní priorit** – Tato metrika zkoumá, zda a jak jsou v projektu využívány různé úrovně priorit. Specificky se zaměřuje na počet použitých úrovní priorit v rámci projektu, což naznačuje, jak dobře je systém prioritizace strukturován.
- **Proporce priorit** – Analyzuje, jaký procentuální podíl z celkového počtu úkolů má přiřazenu nejčastěji využívanou prioritu. Tato metrika ukazuje, zda není většina úkolů automaticky zařazována do jedné kategorie priority, což může signalizovat, že zadavatelé úkolů nemají jasnou představu o důležitosti jednotlivých úkolů a ve většině případů jim přisuzují například střední prioritu.

Detekce tohoto anti-vzoru je navržena jako parametrizovatelná, což umožňuje její adaptaci na specifika různých projektů. Parametry zahrnují nastavení minimálního počtu použitých úrovní priorit a maximálního poměru úkolů přiřazených k jedné dominantní prioritě. Překročení kterékoli z těchto prahových hodnot způsobí, že bude anti-vzor považován za identifikovaný.

Celková logika detekce je ilustrována na obrázku 8.1. Parametr minimálního počtu použitých tříd priorit je označený jako `X`, zatímco parametr maximálního poměru úkolů v rámci dominantní priority je označený jako `Y`.



Obrázek 8.1: Postup detekce anti-vzoru Unordered Product Backlog

One Type Fits All

Druhý vybraný anti-vzor, One Type Fits All, se zaměřuje na nedostatek rozmanitosti v typech úkolů nejen v rámci produktového backlogu, ale i v rámci projektu celkově. Tento anti-vzor vychází z obecně používaného sousloví „one size fits all“, které vystihuje vlastnost použitelnou pro všechny situace. Univerzální řešení však ne vždy přináší výhody a není vždy použitelné pro každou situaci.

Typy úkolů v rámci projektu, jako jsou například Feature (nové funkce), Bug (opravy chyb) nebo Administration (administrativní úkony), by měly být dostatečně diverzifikované. Pokud vývojový tým nebo používaný ALM nástroj neumožňuje efektivní diferenciaci a přizpůsobení těchto typů, může to vést k neefektivnímu řízení a plánování práce.

Tento anti-vzor může být identifikován pomocí následujících metrik:

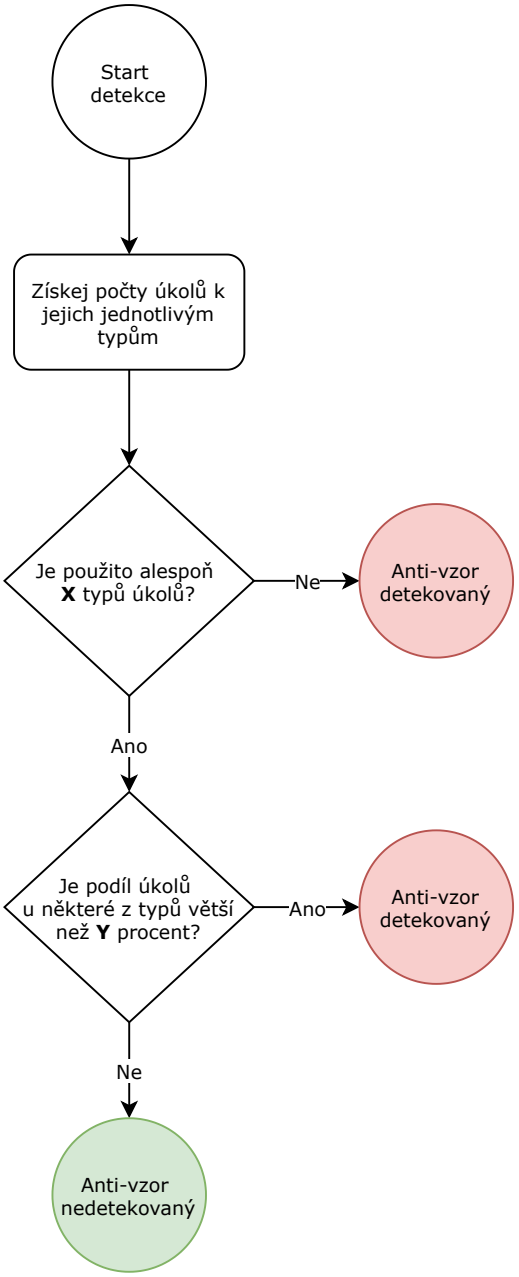
- **Počet typů úkolů** – Tato metrika zkoumá, kolik různých typů úkolů je v projektu aktivně využíváno. Ideální je, aby bylo rozlišení úkolů dostatečně široké pro efektivní kategorizaci, ale zároveň přehledné a ne příliš fragmentované.
- **Proporce typů** – Analyzuje, jaký podíl úkolů připadá na nejčastěji využívaný typ. Dominance jednoho typu úkolu nad ostatními může ukazovat na potřebu revize a lepšího rozdělení úkolů do relevantních kategorií.

Podobně jako předchozí anti-vzor, i One Type Fits All je plně parametrizovatelný. Lze nastavit minimální počet typů úkolů, které by měly být v projektu používány a maximální povolený podíl úkolů pro dominantní typ. Tyto parametry umožňují přizpůsobení detekce anti-vzoru specifickým potřebám a charakteristikám projektu.

Celková logika detekce je ilustrována na obrázku 8.2. Parametr minimálního počtu typů úkolů je označený jako `X` a parametr maximálního podílu úkolů u dominantního typu je označený jako `Y`.

Backlog not DEEP

Dalším z vybraných anti-vzorů, Backlog not DEEP, vychází z konceptu DEEP backlogu. DEEP je akronym, který popisuje ideální stav produktového backlogu, jehož efektivní využití maximálně přispívá k úspěchu týmu. Tento termín zahrnuje čtyři hlavní aspekty, které by měl backlog a úkoly v něm splňovat:



Obrázek 8.2: Postup detekce anti-vzoru One Type Fits All

- Detailed appropriately – dostatečně detailní,
- Estimated – s odhadem náročnosti,
- Emergent – vyvíjející se,
- Prioritized – s přiřazenou prioritou [24].

Vzhledem k tomu, že předchozí dva anti-vzory – Unordered Product Backlog a One Type Fits All – se přímo týkají řízení produktového backlogu, jejich detekce bude klíčová i pro identifikaci anti-vzoru Backlog not DEEP. Pokud dojde k detekci alespoň jednoho z těchto anti-vzorů, automaticky bude považován za detekovaný i nadřazený Backlog not DEEP.

Pokud jsou tyto problémy identifikovány, lze téměř jistě předpokládat, že produktový backlog není správně spravován podle zásad DEEP, což vedle k narušení zmíněných principů může vést i k dalším komplikacím v projektovém řízení. Detekce těchto anti-vzorů tedy může sloužit jako významný indikátor potřeby revize a zlepšení procesů spojených s vedením a aktualizací produktového backlogu.

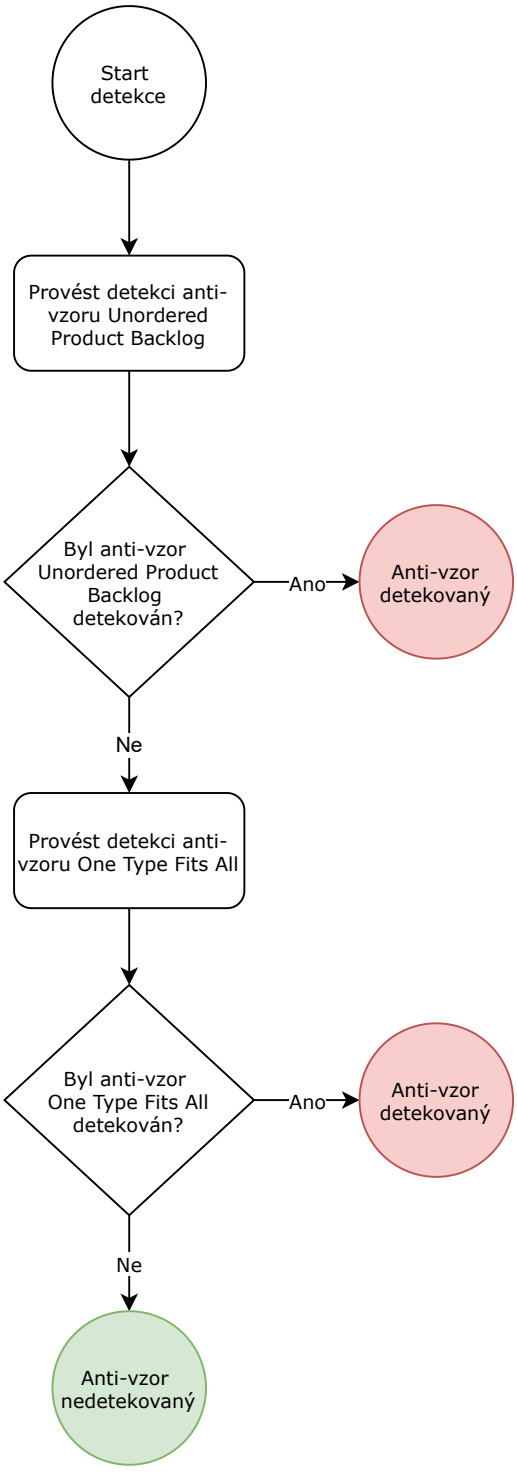
Celková logika detekce je ilustrována na obrázku 8.3.

Process Disintegration

Posledním z vybraných anti-vzorů je Process Disintegration. Tento anti-vzor poukazuje na výskyt selhávajících procesů v projektu, které jsou důsledkem poklesu celkové spolupráce a morálky týmu [14]. Jedná se o celkový úpadek v rámci projektového týmu, který může mít zásadní a často kritický dopad na průběh a úspěšné dokončení projektu. Tento anti-vzor se může projevovat různými způsoby a jeho detekce je často komplikovaná.

Pro identifikaci Process Disintegration lze využít již implementované detektory anti-vzorů, zejména ty, které mohou indikovat pokles morálky týmu v dodržování procesů. Po analýze implementovaných anti-vzorů byly pro detekci Process Disintegration vybrány následující:

- Business As Usual,
- Long Or Non-Existant Feedback Loops,
- Ninety-Ninety Rule,
- Road To Nowhere,
- Specify Nothing,



Obrázek 8.3: Postup detekce anti-vzoru Backlog not Deep

- Too Long Sprint,
- Varying Sprint Length,
- Backlog not DEEP.

Pro potvrzení přítomnosti nadřazeného anti-vzoru Process Disintegration není nutné, aby byly přítomny všechny výše uvedené anti-vzory. Rovněž není vyžadována konkrétní kombinace těchto anti-vzorů. Detekce tohoto nadřazeného anti-vzoru je však vždy založena na nějaké kombinaci nebo počtu z těchto anti-vzorů, což umožňuje její parametrizaci. Parametrizace proběhne nastavením minimálního počtu detekovaných anti-vzorů, který je nutný pro prohlášení Process Disintegration za detekovaný. Tento přístup zajišťuje flexibilitu detekce a umožňuje její přizpůsobení různým druhům projektů.

Celková logika detekce je ilustrována na obrázku 8.4. Parametr minimálního počtu detekovaných anti-vzorů je označený jako `X`.

8.2.2 Implementace a kontrola nových detektorů

Nové detektory pro zmíněné anti-vzory byly začleněny do frameworku SPADe prostřednictvím nově implementované mikroslužby. Při implementaci byly využity již dříve popsané metody, konkrétně skriptování v jazyce Groovy a volání metrik prostřednictvím příslušné metody. Ve srovnání s původními detektory byla využita funkcionalita pro volání vnořených indikátorů.

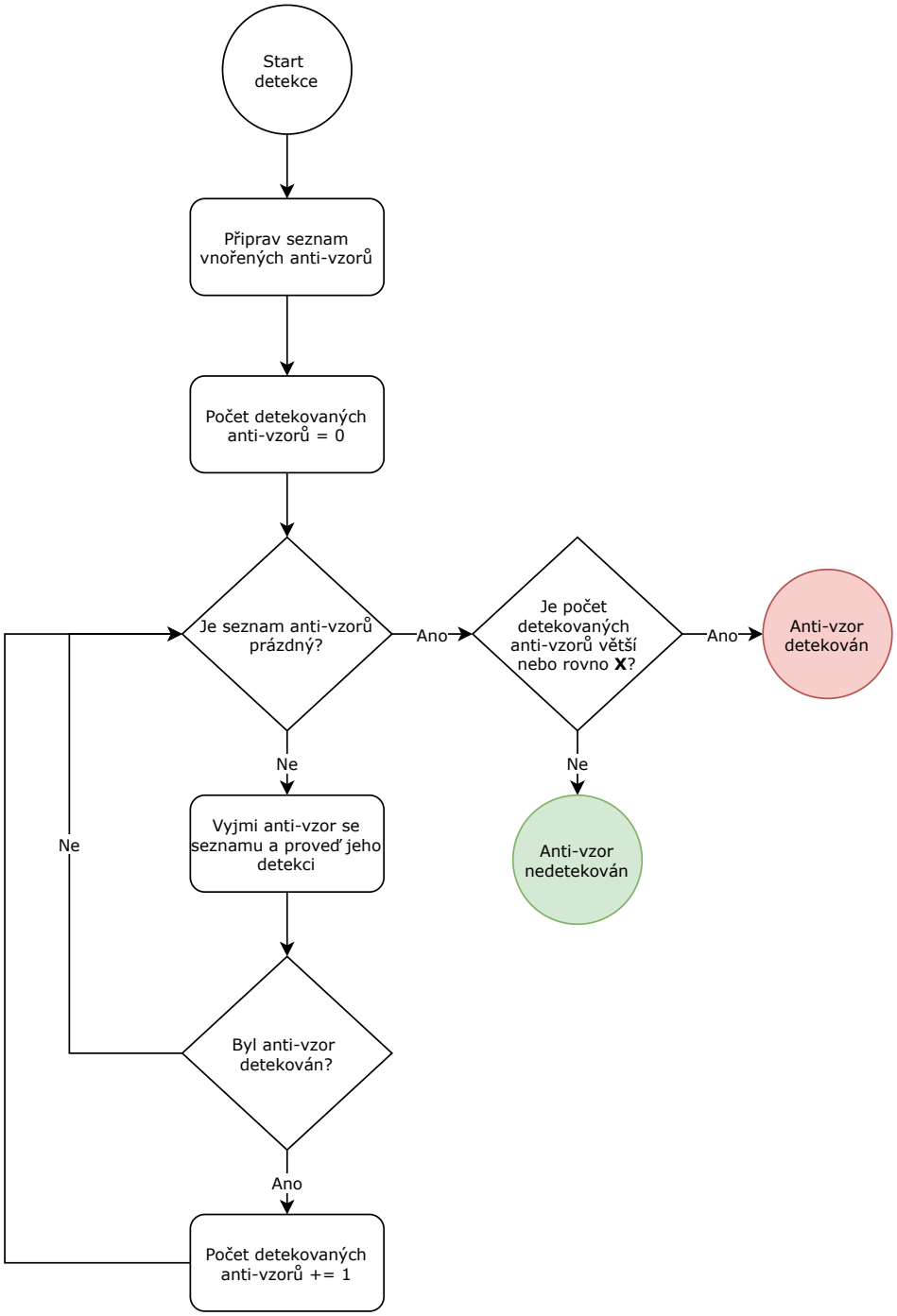
Co se týče parametrizace indikátorů (anti-vzorů), nezbytné parametry pro vnitřní indikátory byly integrovány jako parametry hlavního, vnějšího indikátoru. Tyto parametry byly označeny prefixem, který odpovídá indikátoru, ke kterému patří, čímž se zlepšuje organizace a usnadňuje správa a distribuce jednotlivých parametrů.

Správnost výsledků nově implementovaných detektorů byla ověřena ruční kontrolou jednotlivých kroků detekce na vybrané množině projektových dat. Ve všech případech byla ověřena správnost získaných výsledků.

8.2.3 Provedení detekce a zhodnocení výsledků

Implementace nových detektorů byla testována na množině projektových dat. Aby bylo možné správně interpretovat výsledky, je nezbytné pochopení kontextu analyzovaných projektů.

V rámci analýzy bylo zahrnuto celkem 15 projektů. Z tohoto počtu devět projektů představuje studentské práce realizované v rámci kurzu KIV/ASWI na Fakultě aplikovaných věd ZČU. Zbývajících šest projektů tvoří



Obrázek 8.4: Postup detekce anti-vzoru Process Disintegration

open-source projekty, které byly vybrány na základě kritérií stanovených v předchozí bakalářské práci. Tato kritéria primárně adresovala dostupnost relevantních dat pro analýzu [31].

Dalším důležitým faktorem je také nastavení parametrů pro tyto indikátory. Anti-vzory Unordered Product Backlog a One Type Fits All byly parametrizovány podobným způsobem. Byly vyžadovány minimálně tři třídy priorit, respektive tři typy úkolů. Tento počet by mohl být dostačující pro základní kategorizaci úkolů. V rámci vyhodnocení dominantní třídy či typu byla zvolena hranice v případě priorit 80 % a v případě typů úkolů 90 % z celkového počtu úkolů.

Anti-vzor Backlog not DEEP využíval pouze parametry pro své vnitřní anti-vzory, jejichž hodnoty byly stejné jako výše uvedené.

Anti-vzor Process Disintegration byl parametrizován tak, aby pro jeho detekci bylo detekováno alespoň 5 z 8 vnitřních anti-vzorů, tedy nadpoloviční počet. Veškeré vnitřní anti-vzory využívaly výchozí parametry nastavené pro nové detektory jako výše uvedené a pro původní detektory hodnoty nastavené v rámci předchozích prací [30][31].

Unordered Product Backlog

Anti-vzor Unordered Product Backlog byl detekován ve všech zkoumaných open-source projektech. Převážně se projevoval nedostatkem diferencovaných priorit úkolů a ve specifickém případě byla většina úkolů přiřazena pouze jedné prioritě.

Tato situace reflektuje charakter vybraných open-source projektů, které nebyly vyvíjeny stabilním vývojovým týmem, ale spíše se jednalo o příležitostné příspěvky různých osob ve spíše neorganizované formě, zejména v pozdějších fázích. V důsledku toho nebyl řízený projektový proces, a tedy ani prioritizace úkolů se nejevila jako nezbytná. Avšak i přes tento fakt by určitá forma prioritizace měla své opodstatnění. V případě, že by například zkušení nebo noví přispěvatelé identifikovali kritickou chybu ohrožující funkčnost produktu, bylo by vhodné takový úkol označit vysokou prioritou. To by následně umožnilo ostatním přispěvatelům snáze rozeznat, které úkoly by měly být řešeny s předností.

Situace u studentských projektů byla odlišná. V třech projektech nebyl tento anti-vzor zaznamenán. U ostatních bylo sice dostatek prioritních tříd, ale většina úkolů spadala do jedné dominantní priority, která překročila stanovenou prahovou hodnotu pro detekci tohoto anti-vzoru.

Fakt, že u studentských projektů nedošlo k detekci anti-vzoru z důvodu nedostatečného množství prioritních tříd, odráží záměr těchto projektů, kde

byl kladen důraz na využití prioritizace jako součást výuky. V případech, kde převládala jedna priorita, se nemusí nutně jednat o negativní jev. Každý projekt je jedinečný, a stejně tak jedinečné může být uspořádání priorit. Nicméně, výskyt tohoto anti-vzoru by měl sloužit jako upozornění a motivace k revizi způsobu, jakým je prioritizace v projektu prováděna.

One Type Fits All

V rámci studentských projektů nebyl anti-vzor One Type Fits All detekován v žádném případě. Tento výsledek je opět dán charakterem těchto projektů, kde rozmanité typy úkolů byly záměrně zapojeny jako součást výuky předmětu, v rámci něhož byly projekty realizovány.

U open-source projektů byl tento anti-vzor identifikován ve většině případech, s výjimkou jednoho. V případech kdy došlo k odhalení anti-vzoru bylo obvyklé, že úkoly nebyly dostatečně rozčleněny podle typu, což by mohlo vést k zhoršení přehlednosti a celkové organizaci.

Rozdělení úkolů na různé typy, jako jsou nový vývoj, opravy chyb, úpravy funkcionalit, testování nebo analýzy, je zásadní pro efektivní správu projektů. Každý projekt je komplexní a vyžaduje různorodé činnosti, a proto je výskyt tohoto anti-vzoru vhodnou příležitostí k revizi a potenciálnímu zlepšení kategorizace typů úkolů, aby byla zajištěna lepší a efektivnější organizace práce.

Backlog not DEEP

Výsledky detekce tohoto anti-vzoru reflektují zjištění z předchozích dvou analyzovaných anti-vzorů. V rámci studentských projektů nebyl tento anti-vzor zaznamenán v žádném případě. Na druhé straně, u open-source projektů byl tento anti-vzor detekován ve většině případů, s výjimkou dvou projektů.

Identifikace kteréhokoli z vnitřních anti-vzorů vyžaduje provedení důkladné revize procesů spojených s tvorbou a správou produktového backlogu, včetně přiřazování priorit a kategorizace úkolů podle jejich typů. Využití správné praxe v těchto oblastech zajišťuje efektivní správu projektových úkolů, což přináší prospěch všem zainteresovaným stranám. Důkladné zvážení a potenciální úpravy pak mohou výrazně přispět k lepší organizaci a průběhu projektových aktivit.

Process Disintegration

Detekce anti-vzoru Process Disintegration je závislá na výsledcích detekce vnořených anti-vzorů. V rámci zkoumaných projektů nebyl tento anti-vzor

identifikován v žádném případě, což naznačuje, že žádný z projektů nevykazoval známky výrazného úpadku v organizaci projektové práce.

Identifikace tohoto anti-vzoru v projektových datech by vyžadovala okamžitou a důkladnou revizi organizace projektových činností. Tento krok je nezbytný pro zajištění, že projekt bude moci pokračovat bez rizika selhání. Intervence na základě výskytu Process Disintegration je klíčová pro zachování integrity a úspěšnosti projektu, protože tento anti-vzor signalizuje závažné problémy v řízení a koordinaci týmu, které mohou ohrozit celkový výsledek.

9 Náměty na další rozšíření

Ačkoliv byla navržená mikroslužba úspěšně implementována, zaintegrována a je funkční, stále existuje prostor pro její další rozvoj a zdokonalení existujících funkcionalit.

V rámci návrhu a implementace byly zavedeny prvky, které usnadňují budoucí rozšíření aplikace. Konkrétním příkladem je použití rozhraní, které umožňuje snadné přidání dalších skriptovacích jazyků pro zápis logiky detekce. Možností je také implementace pravidlově orientovaných jazyků či vlastní gramatiky. To umožní uživatelům vybírat z více programovacích jazyků při tvorbě nových indikátorů, což zvýší flexibilitu a uživatelský komfort.

Je možné rozšířit i spektrum typů indikátorů, a to o opačně laděné fenomény, jako jsou dobré praktiky a vzory. Přístup pro jejich tvorbu a identifikaci v projektových datech by mohl být totožný, jako tomu je u současně implementovaných anti-vzorů. Tímto by se výrazně rozšířily možnosti využití nástroje.

Další potenciální rozšíření se týká datových typů parametrů. Současná implementace umožňuje volbu datového typu, ale tento výběr není striktně vynucen a má pouze informativní charakter. To může vést k chybám, pokud uživatel zadá parametr ve špatném formátu. Zavedení přísnějšího ověřování na uživatelském rozhraní i na straně serveru by zlepšilo robustnost systému. Možné je i rozšíření stávající nabídky datových typů.

V kontextu detekční mikroslužby je také nezbytné vyřešit bezpečnostní aspekty spojené se spouštěním skriptů, jak bylo dříve zmíněno. Řádné ošetření těchto bezpečnostních rizik je klíčové pro to, aby byl systém použitelný nejen pro současné experimentální a vědecké účely, ale aby bylo možné ho efektivně a bezpečně využívat i v běžné praxi.

Vylepšení může proběhnout i na úrovni uživatelského rozhraní. Přidání grafických reprezentací výsledků, možnosti srovnání výsledků z různých projektů na jednom grafu nebo možnosti ukládání a exportu výsledků by znatelně zvýšilo uživatelský komfort a přehlednost.

V původní verzi aplikace byla implementována funkcionalita, která uživatelům umožňovala ukládat si vlastní sady použitých prahových hodnot (parametrů) pro jejich budoucí opětovné využití. Tato funkce v nové architektuře zatím chybí, ale systém je na její přidání připraven a implementace by neměla být komplikovaná.

Současně je přístup k detekční sekci omezen pouze na přihlášené uživatele. Vytváření nových detekčních definic by mělo být povoleno pouze ově-

řeným uživatelům s odpovídajícími znalostmi o struktuře databáze s projektovými daty. Na druhou stranu, samotné provádění detekcí by mohlo být dostupné širšímu okruhu uživatelů. Tento přístup by bylo možné regulovat pomocí rolí uživatelů v systému Keycloak.

Inovativní rozšíření by mohlo zahrnovat využití umělé inteligence při tvorbě skriptů indikátorů a použitých metrik. Tato inteligence by byla seznámena se strukturou databáze projektových dat a na základě verbálního popisu uživatele by sestavovala skripty pro detekci indikátorů. Tento přístup by významně zjednodušil a zefektivnil proces tvorby indikátorů.

10 Závěr

Tato diplomová práce byla zaměřena na rozšíření a zlepšení detekčních schopností nástrojové sady Software Process Anti-Pattern Detector (SPADe) s primárním cílem zvýšit její modularitu. Specificky práce usilovala o vytvoření možnosti definice jednotlivých metrik, jejich prahových hodnot, indikátorů a dále o umožnění složitější kompozice těchto prvků do rozsáhlejších struktur, jako jsou anti-vzory.

Realizace těchto cílů si vyžádala důkladné seznámení s oblastí projektového a procesního řízení softwarových projektů a s nástroji používanými v tomto kontextu (kapitola 2). Poté bylo prozkoumáno zpracování projektových dat a byla provedena podrobná analýza nástroje SPADe (kapitola 3). Následně bylo navrženo nové řešení detekčního mechanismu (kapitoly 4, 5), které bylo implementováno a integrováno do existujícího systému (kapitoly 6, 7). Nové příspěvky byly důkladně otestovány (podkapitoly 6.9, 7.4), stávající detekce byly převedeny a doplněny o nové (kapitola 8). V závěrečné části byly navrženy směry pro další rozvoj implementovaného detekčního mechanismu (kapitola 9).

Klíčovým výstupem této práce je nově vyvinutý modulární detekční mechanismus ve formě mikroslužby pro definici fenoménů, jako jsou metriky, indikátory, parametry a anti-vzory. Tento přístup výrazně zvyšuje flexibilitu aplikace, umožňuje efektivní znovuvyužití již implementovaných komponent a jejich integraci do složitějších struktur. Proces vytváření těchto komponent je nyní realizovatelný přímo během provozu aplikace, což eliminuje potřebu složitých implementačních postupů a opakovaného restartování systému, který byl dříve nezbytný. Implementace je navíc intuitivnější a umožňuje snadné přidávání komponent i uživatelům bez hlubokých technických znalostí celého systému. Také struktura zápisu detekcí je nyní přehlednější než v předchozích verzích. Zásadní novinkou je rovněž integrace nových detekcí, které rozšiřují spektrum detekovatelných fenoménů systémem.

Tato práce představuje významný přínos k rozšíření a zdokonalení funkcionalit systému SPADe, čímž podstatně zlepšuje jeho budoucí využitelnost. Současně byly rozšířeny možnosti škálování systému, což zlepšuje jeho připravenost na očekávaný nárůst zpracovávaných dat, jak z důvodu přidávání nových detekcí, tak i zvýšeného využívání systému pro analýzy různých projektů.

Zadání této práce bylo splněno v plném rozsahu.

Literatura

[1] *A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Seventh Edition and The Standard for Project Management*. Project Management Institute, Inc., 2021. ISBN 978-1-62825-664-2.

[2] *Jira Software features* [online]. Atlassian. [cit. 2024/04/20]. Dostupné z: `https://www.atlassian.com/software/jira/features`.

[3] *Project management triangle: overview of the triple constraints* [online]. PRINCE2.com, 2019. [cit. 2024/05/8]. Dostupné z: `https://www.prince2.com/eur/blog/project-triangle-constraints`.

[4] *Agile Software Development Methodology* [online]. One Beyond Limited. [cit. 2024/04/20]. Dostupné z: `https://www.one-beyond.com/process/agile-software-development-methodology/`.

[5] Attarzadeh, I. – Ow, S. H. Project management practices: The criteria for success or failure. *Communications of the IBIMA*. 2008, 1, 28, s. 234–241.

[6] Awad, M. A comparison between agile and traditional software development methodologies. *University of Western Australia*. 2005, 30, s. 1–69.

[7] Bannerman, P. L. Defining project success: a multilevel framework. Paper presented at PMI® Research Conference: Defining the Future of Project Management, Warsaw, Poland. *Project Management Institute*. 2008. Dostupné z: `https://www.pmi.org/learning/library/defining-project-success-multilevel-framework-7096`.

[8] Brada, P. – Pícha, P. Software Process Anti-pattern Catalogue. *EuroPLoP '19: Proceedings of the 24th European Conference on Pattern Languages of Programs*. July 2019. doi: https://doi.org/10.1145/3361149.3361178.

[9] Brada, P. – Pícha, P. Software Process Anti-pattern Detection in Project Data. *EuroPLOP '19: 24th European Conference on Pattern Lan- guages of Programs, July 03–07, 2019, Kloster Irsee, Germany.* 2019.

[10] Bronte-Stewart, M. Beyond the iron triangle: Evaluating aspects of success and failure using a project status model. *Computing & Information Systems*. 2015, 19, 2, s. 19–36.

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

[12] Coplien, J. O. – Harrison, N. B. *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc., 2004. ISBN 978-0-13-146740-8.

[13] Bruin, H. – Vliet, H. Top-down composition of software architectures. 2002, s. 147–156. doi: 10.1109/ECBS.2002.999833.

[14] Silva, P. D. *Categorization of Software Project Management Anti-Patterns. Master Thesis.* Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid, 2021. Dostupné z: `https://oa.upm.es/32705/7/TESIS_MASTER_PEDRO_DIAS_E_SILVA_A.pdf`.

[15] Eloranta, V.-P. – Koskimies, K. – Mikkonen, T. Exploring ScrumBut—An empirical study of Scrum anti-patterns. *Information and Software Technology*. 2016, 74, s. 194–203. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2015.12.003. Dostupné z: `https://www.sciencedirect.com/science/article/pii/S0950584915002050`.

[16] Gos, K. – Zabierowski, W. The Comparison of Microservice and Monolithic Architecture. 2020, s. 150–153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[17] Ilieş, E. – Crişan, E. – Mureşan, I. N. Best Practices in Project Management. *Review of International Comparative Management*. March 2010, 11, s. 9. Dostupné z: `http://www.rmci.ase.ro/no11vol1/Vol11_No1_Article4.pdf`.

[18] Jones, J. *The Cascading Costs of Waterfall* [online]. Medium, 2019. [cit. 2024/04/20]. Dostupné z: `https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec`.

[19] Madasu, V. K. et al. Solid principles in software architecture and introduction to resm concept in oop. *Studies*. 2015, 35, 38, s. 8.

[20] Motalík, P. *Procesní řízení a jeho optimalizace v organizaci. Diplomová práce*. Katedra informačních technologií a elektronického bankovnictví, Bankovní institut vysoká škola Praha, 2021. Dostupné z: `https://is.ambis.cz/th/10726/bivs_m/DP_PetrMotalik.pdf`.

[21] Myllyaho, M. et al. A review of small and large post-mortem analysis methods. *Proceedings of the ICSSEA, Paris*. 2004, s. 1–8.

[22] Palma, F. – Moha, N. – Guéhéneuc, Y.-G. Specification and Detection of Business Process Antipatterns. *Benyoucef, M., Weiss, M., Mili, H. (eds)*

E-Technologies. MCETECH 2015. Lecture Notes in Business Information Processing, vol 209. 2015. doi: https://doi.org/10.1007/978-3-319-17957-5_3.

[23] Pecinovský, R. *Návrhové vzory.* Computer Press, 2015. ISBN 978-80-251-1582-4.

[24] Pichler, R. *Make Your Product Backlog DEEP* [online]. Pichler Consulting Limited, 2022. [cit. 2024/05/04]. Dostupné z: `https://www.romanpichler.com/blog/make-the-product-backlog-deep/`.

[25] 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: `https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2019/Rigo_P%c3%adcha_2019_4.pdf`.

[26] Pícha, P. *Datový model nástroje SPADe a jeho mapování na projektová data z ALM nástrojů. Práce ke zkoušce z předmětu Moderní programovací styly a metody.* Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2016.

[27] Saeed, S. et al. Analysis of software development methodologies. *International Journal of Computing and Digital Systems.* 2019, 8, 5, s. 446–460.

[28] Sami, M. *The Software Process Improvement (SPI) – Reward or Risk* [online]. Mohamed Sami, 2018. [cit. 2024/05/8]. Dostupné z: `https://melsatar.blog/2018/06/26/the-software-process-improvement-spi-reward-or-risk/`.

[29] Settas, D. L. et al. *SPARSE: A symptom-based antipattern retrieval knowledge-based system using Semantic Web technologies.* Department of Informatics, Aristotle University, Greece, 2010. doi: https://doi.org/10.1016/j.eswa.2010.12.097.

[30] Váně, O. *Analýza přítomnosti anti-vzorů v datech nástrojů pro řízení projektů. Diplomová práce.* Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2021. Dostupné z: `http://hdl.handle.net/11025/44774`.

[31] Štěpánek, P. *Detekce špatných praktik projektového řízení v open-source projektech.* Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2022. Dostupné z: `http://hdl.handle.net/11025/49934`.

Přehled zkratek

- **ALM** – Application Lifecycle Management
- **API** – Application Programming Interface
- **ASWI** – Pokročilé softwarové inženýrství
- **DEEP** – Detailed appropriately, Estimated, Emergent, Prioritized
- **ERA** – Entity-Relationship Attribute
- **ETL** – Extract-Transform-Load
- **FAV** – Fakulta aplikovaných věd
- **HTTP** – Hypertext Transfer Protocol
- **JDBC** – Java Database Connectivity
- **JPA** – Java Persistence API
- **JSON** – JavaScript Object Notation
- **RUP** – Rational Unified Process
- **SOLID** – Single responsibility, Open–closed, Liskov substitution, Interface segregation Dependency inversion
- **SPI** – Software Process Improvement
- **SPADe** – Software Process Anti-pattern Detector
- **SPAWn** – SPADe Web Interface
- **SQL** – Structured Query Language
- **XML** – Extensible Markup Language
- **ZČU** – Západočeská univerzita v Plzni

Seznam obrázků

A Obsah ZIP souboru

Tato příloha popisuje strukturu přiloženého ZIP souboru.

- `Text_prace`
  - **zdroj** – zdrojové soubory textu diplomové práce
  - **diplomova_prace.pdf** – text diplomové práce
- `Aplikace_a_knihovny`
  - **spade_tools**
    - **detector** – zdrojové kódy detekční služby
    - **imports** – importní soubory pro Keycloak
    - **spade** – zdrojové kódy aplikace SPADe
    - **spawn** – zdrojové kódy aplikace SPAWn
    - **docker-compose.yml** – konfigurační soubor pro Docker Compose
- `Poster`
  - **stepanek_petr_2024.pub** – poster ve formátu PUB
  - **stepanek_petr_2024.pdf** – poster ve formátu PDF
- `Readme.txt` – detailní popis adresářové struktury

B Instalační příručka

Tato příručka nabízí podrobný návod na instalaci všech součástí nástrojové sady na operační systém Windows.

B.1 Potřebné nástroje

Pro spuštění aplikační sady SPADe jsou nezbytné následující nástroje:

- Docker[1],
- Docker Compose[2].

B.2 Konfigurace

Všechny komponenty nástrojové sady jsou předem plně nakonfigurovány, takže není potřeba žádná další konfigurace.

Konfigurační soubory pro spojení s databází se v rámci detekční služby a aplikace SPADe nachází ve složce `src/main/resources`.

B.3 Instalace a spuštění

Instalace a spuštění všech komponent nástrojové sady probíhá následujícím způsobem:

1. Ujistit se, zda jsou porty 8080, 8081, 3000, 3306, 9080 a 5432 dostupné.
2. Zkopírovat složku `Aplikace_a_knihovny/spade_tools` z přiloženého ZIP souboru na libovolné místo.
3. Přesunout se do kořenové složky `spade_tools` na zvoleném místě a zde otevřít příkazový řádek.
4. Vytvořit potřebné obrazy služeb pomocí příkazu:

```
docker-compose build
```

[1] https://docs.docker.com/get-docker
[2] https://docs.docker.com/compose/install

5. Spustit MySQL databázi pomocí příkazu:

```
docker-compose up db
```

   Nyní dochází k vytvoření a spuštění databáze, vytvoření databázových struktur a vkládání dat. To může trvat několik minut. Proces bude dokončen, když se v logu objeví hláška:

   MySQL init process done. Ready for start up.

6. Minimalizovat otevřenou příkazovou řádku a otevřít další příkazovou řádku ve stejném místě.

7. Spustit ostatní komponenty pomocí příkazu:

```
docker-compose up
```

8. Veškeré komponenty nástrojové sady jsou nyní spuštěny. Uživatelské rozhraní je dostupné v prohlížeči na adrese:

```
http://localhost:3000
```

B.4 Spuštění testů

Jednotkové a integrační testy vytvořené pro aplikace SPADe a nově vzniklou detekční službu lze spustit v rámci Dockeru následujícím způsobem:

1. Nainstalovat a spustit veškeré komponenty nástrojové sady podle návodu B.3.

2. Otevřít příkazovou řádku v libovolném místě.

3. Připojit se do běžícího kontejneru detekční služby pomocí příkazu:

```
docker exec -it detector /bin/bash
```

4. Spustit sestavení aplikace, a poté jednotkové a integrační testy detekční služby pomocí příkazu:

```
mvn verify
```

5. Počkat na dokončení sestavení detekční služby a proběhnutí všech jejích testů.

6. Připojit se do běžícího kontejneru aplikace SPADe pomocí příkazu:

```
docker exec -it spade /bin/bash
```

7. Spustit sestavení aplikace, a poté jednotkové a integrační testy aplikace SPADe pomocí příkazu:

```
mvn verify
```

8. Počkat na dokončení sestavení aplikace SPADe a proběhnutí všech jejích testů.

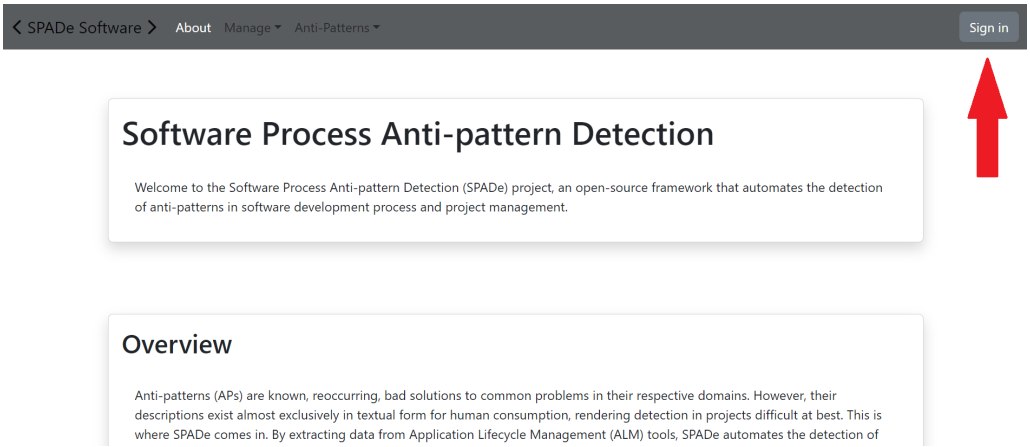
C Uživatelská příručka

Tato příručka popisuje ovládání nově přidaných částí uživatelského rozhraní nástrojové sady SPADe.

C.1 Registrace a přihlášení

Nově přidané části uživatelského rozhraní jsou dostupné pouze pro přihlášené uživatele. Proto je nutné se nejprve zaregistrovat a poté přihlásit.

Obrazovku pro přihlášení a registraci lze otevřít z úvodní obrazovky uživatelského rozhraní kliknutím na tlačítko *Sign in* (viz Obrázek C.1).

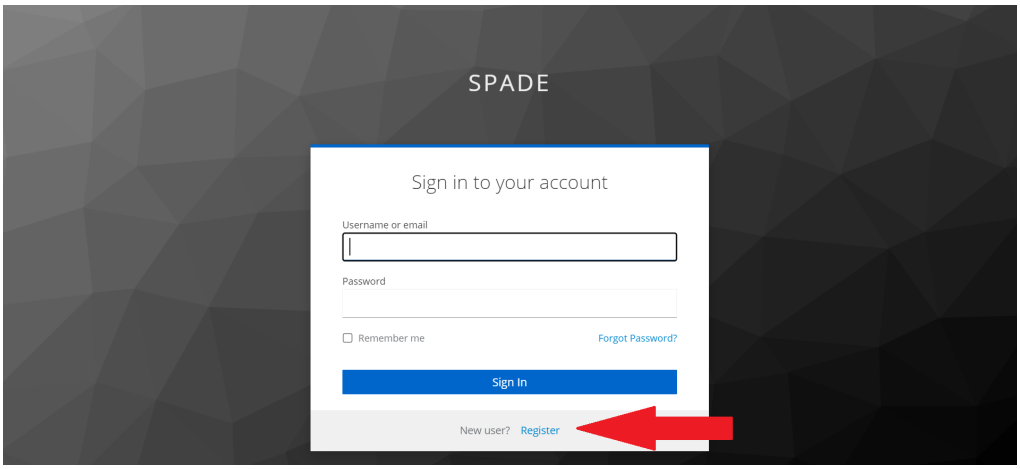


Obrázek C.1: Tlačítko pro přihlášení

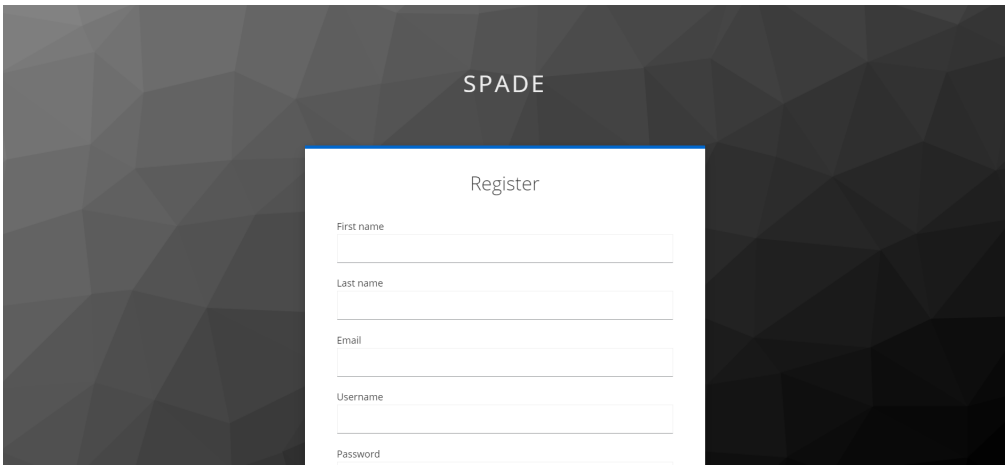
Po stisknutí tlačítka dojde k přesměrování na přihlašovací stránku systému SPADe. Nicméně, nejprve je nutné se zaregistrovat. To lze provést kliknutím na tlačítko *Register* (viz Obrázek C.2).

Stiskem tlačítka dojde k přesměrování na registrační formulář (viz Obrázek C.3). Zde je nutné vyplnit všechny potřebné údaje a formulář odeslat stisknutím tlačítka *Register*, které se nachází pod formulářem.

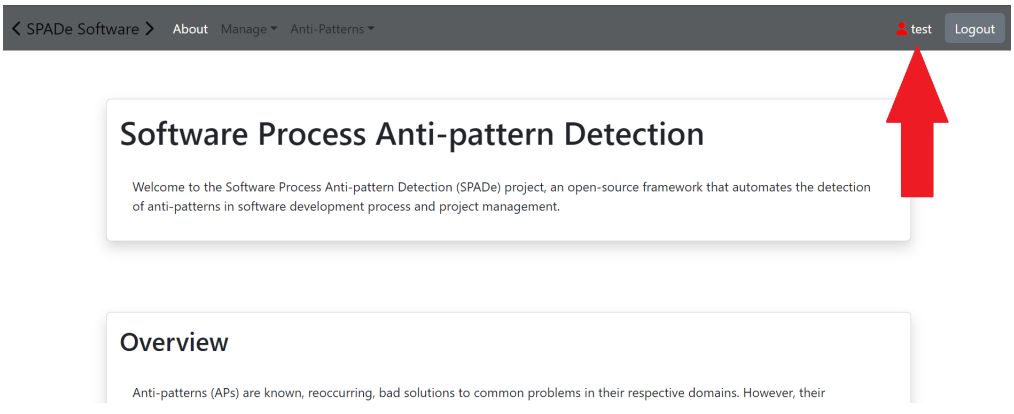
Po úspěšné registraci dojde k přesměrování zpět na úvodní obrazovku. V pravém horním rohu je nyní zobrazena ikona se jménem uživatele (viz Obrázek C.4). Pro opětovné přihlášení již registrovaného uživatele lze použít přihlašovací formulář (viz Obrázek C.2).



Obrázek C.2: Tlačítko pro registraci



Obrázek C.3: Formulář pro registraci



Obrázek C.4: Přihlášený uživatel

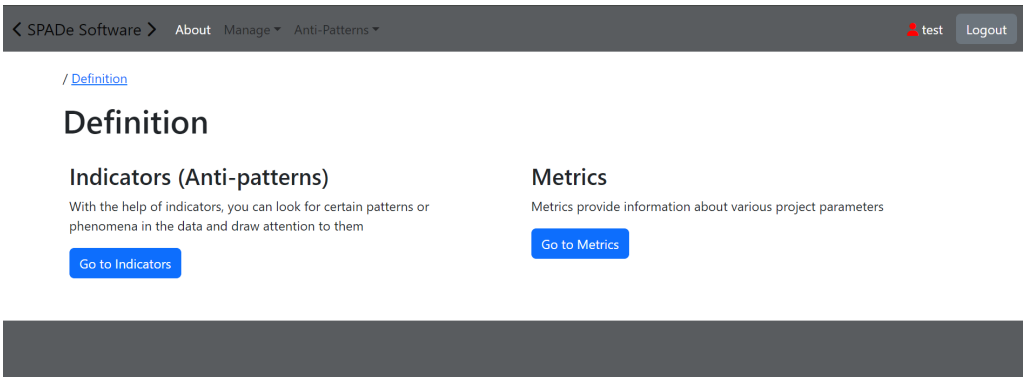
C.2 Definice fenoménů

Tato část popisuje použití uživatelského rozhraní pro definici fenoménů, kterými jsou metriky, indikátory, anti-vzory a jejich parametry.

Na obrazovku pro definice lze přejít kliknutím na záložku *Anti-Patterns* v hlavním menu. Tím se vyvolá kontextová nabídka, ve které se vybere položka *Definition* (viz Obrázek C.5).

Obrázek C.5: Kontextová nabídka

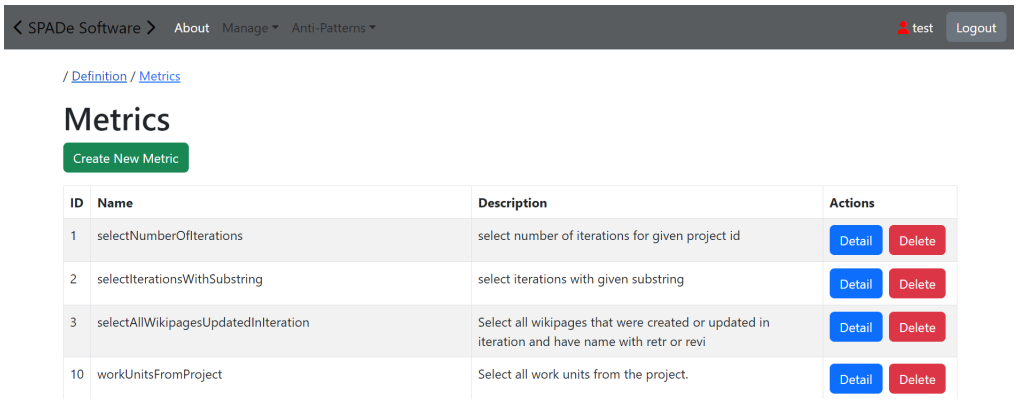
Po stisknutí tlačítka *Definition* dojde k přesměrování na obrazovku pro výběr fenoménů, konkrétně metrik a indikátorů (viz Obrázek C.6).



Obrázek C.6: Obrazovka výběru fenoménů

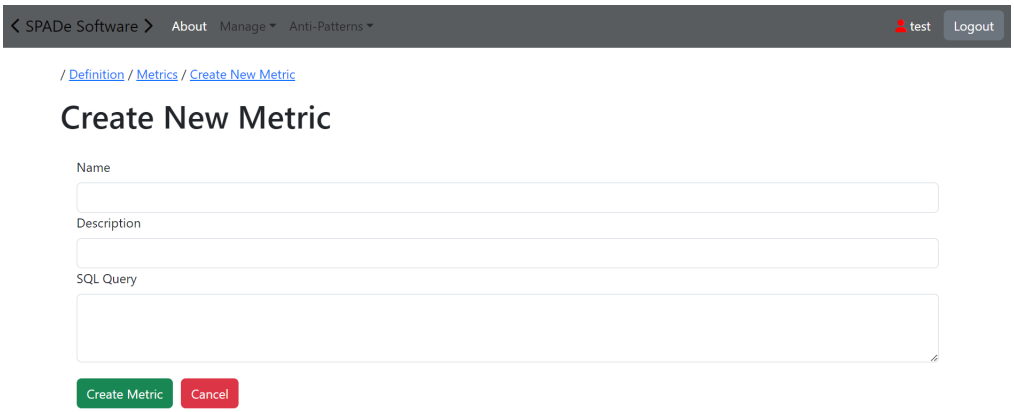
C.2.1 Metriky

Po stisknutí tlačítka *Go to Metrics* dojde k přesměrování na obrazovku s výčtem všech již vytvořených metrik (viz Obrázek C.7). Na této obrazovce je možné vytvořit novou metriku (tlačítko *Create New Metric*), zobrazit detail konkrétní metriky (tlačítko *Detail*) a odstraňovat již vytvořené metriky (tlačítko *Delete*).



Obrázek C.7: Obrazovka metrik

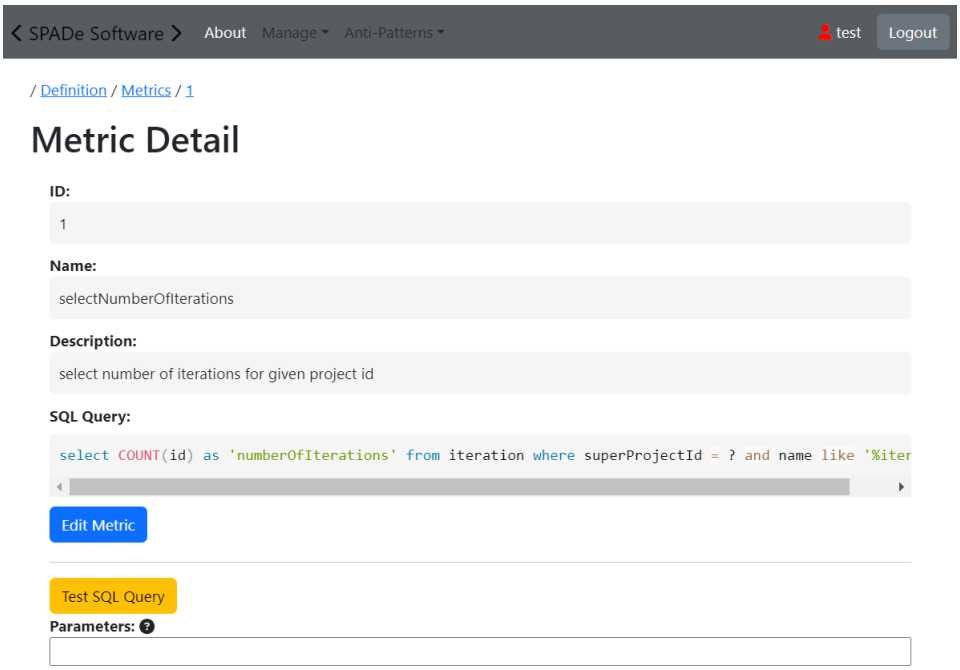
Stisknutím tlačítka *Create New Metric* dojde k přesměrování na stránku s formulářem pro vytvoření nové metriky (viz Obrázek C.8). Stisknutím tlačítka *Create Metric* na této obrazovce dojde k vytvoření nové metriky a přesměrování na její detail.



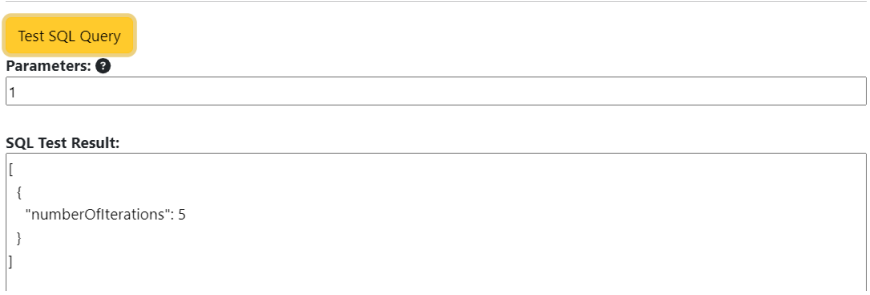
Obrázek C.8: Vytvoření nové metriky

Obrazovka detailu metriky (viz Obrázek C.9) obsahuje veškeré informace o dané metrice. Tyto informace lze editovat stisknutím tlačítka *Edit Metric*.

Na obrazovce detailu metriky je možné otestovat přidaný SQL dotaz pomocí tlačítka *Test SQL Query*. Pokud je dotaz parametrizovatelný, parametry se zadají do pole *Parameters* v pořadí, v jakém se do dotazu vkládají, a oddělené čárkou. V případě úspěšného vykonání dotazu se zobrazí pole s výsledky (viz Obrázek C.10). V opačném případě se zobrazí upozornění o chybě.



Obrázek C.9: Detail metriky

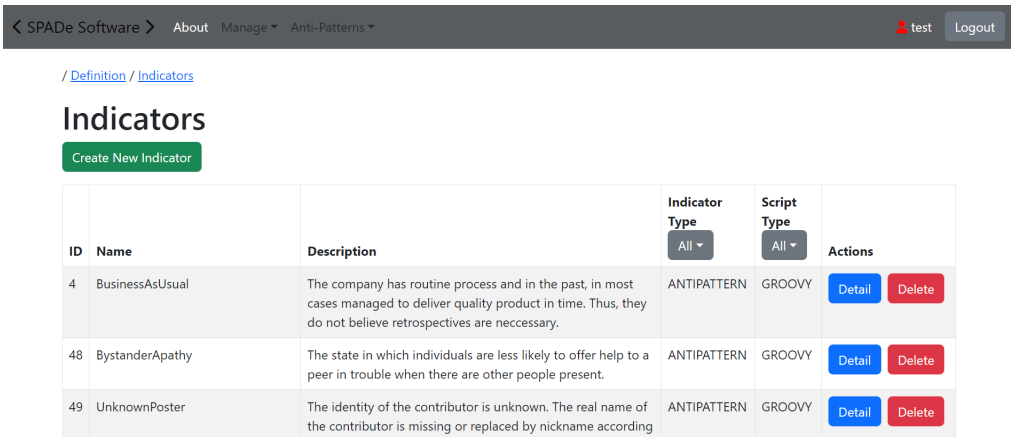


Obrázek C.10: Test metriky

C.2.2 Indikátory

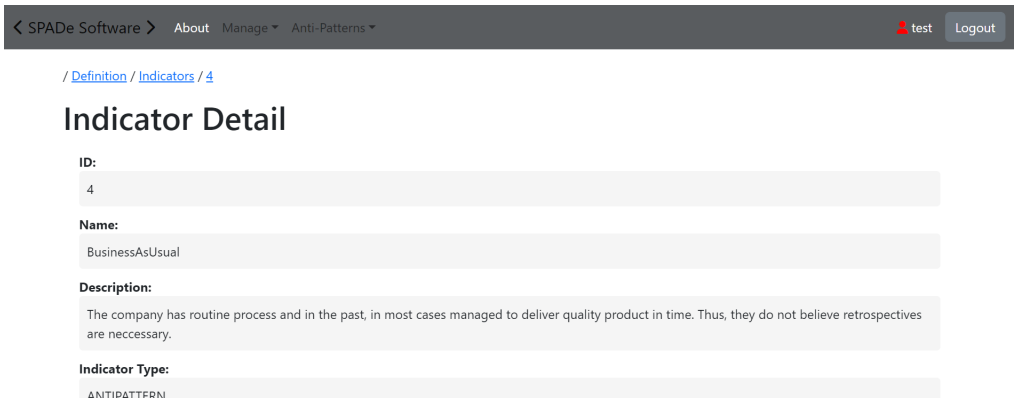
Po stisknutí tlačítka *Go To Indicators* na obrazovce výběru fenoménů (viz Obrázek C.6) dojde k přesměrování na obrazovku indikátorů (viz Obrázek C.11). Tato obrazovka poskytuje obdobné funkcionality jako obrazovka metrik, tedy vytvoření nového indikátoru (*Create New Indicator*), přechod na detail konkrétního indikátoru (*Detail*) a odstranění indikátoru (*Delete*). Navíc je možné filtrovat indikátory podle jejich typu (*Indicator Type*) nebo typu jejich skriptu (*Script Type*).

Obrazovka detailu indikátoru (viz Obrázek C.12) poskytuje veškeré informace o vybraném indikátoru, včetně jeho výkonného skriptu. Tyto informace



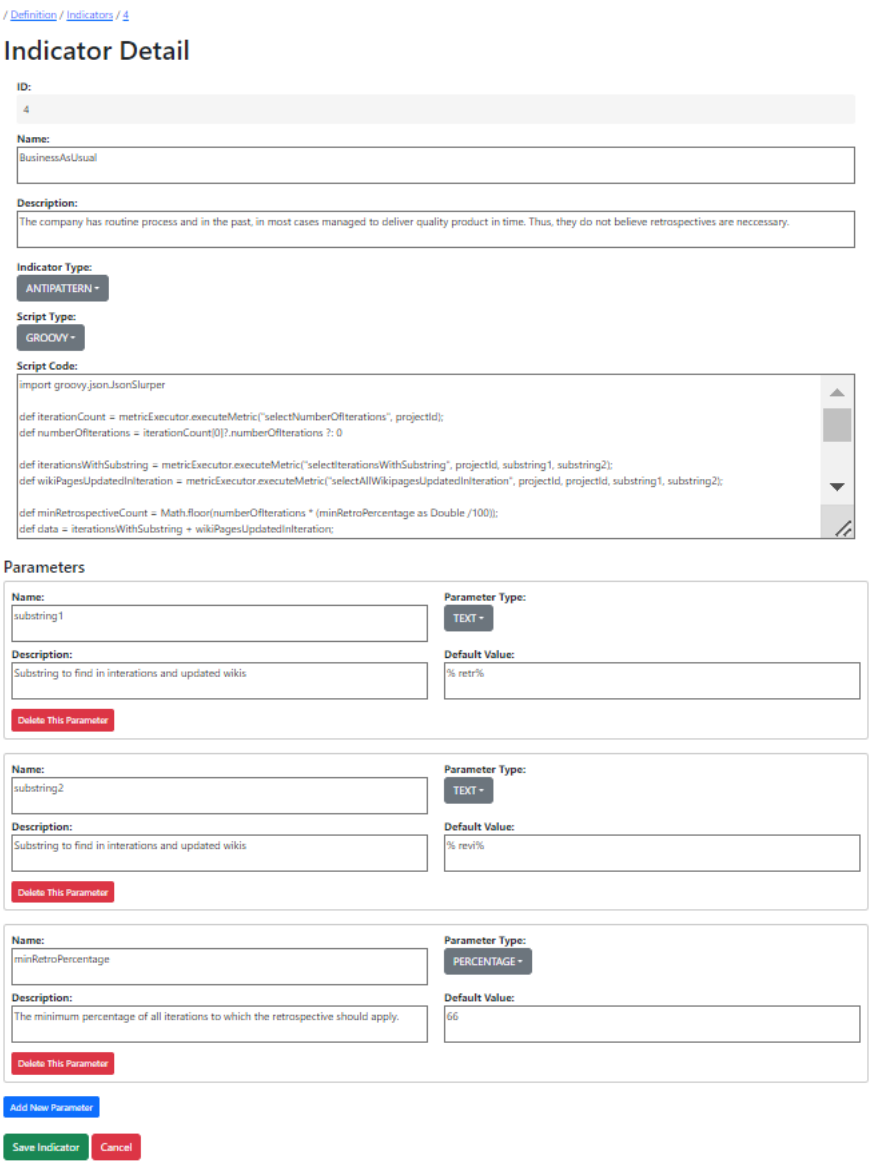
Obrázek C.11: Obrazovka indikátorů

lze editovat stisknutím tlačítka *Edit* v dolní části obrazovky.



Obrázek C.12: Obrazovka detailu indikátoru

Obrazovka editace indikátoru (viz Obrázek C.13) umožňuje upravovat veškeré informace o vybraném indikátoru. Za zmínku stojí možnost libovolně přidávat a odstraňovat parametry v sekci *Parameters*, které budou vyžadovány od uživatele před spuštěním detekce.



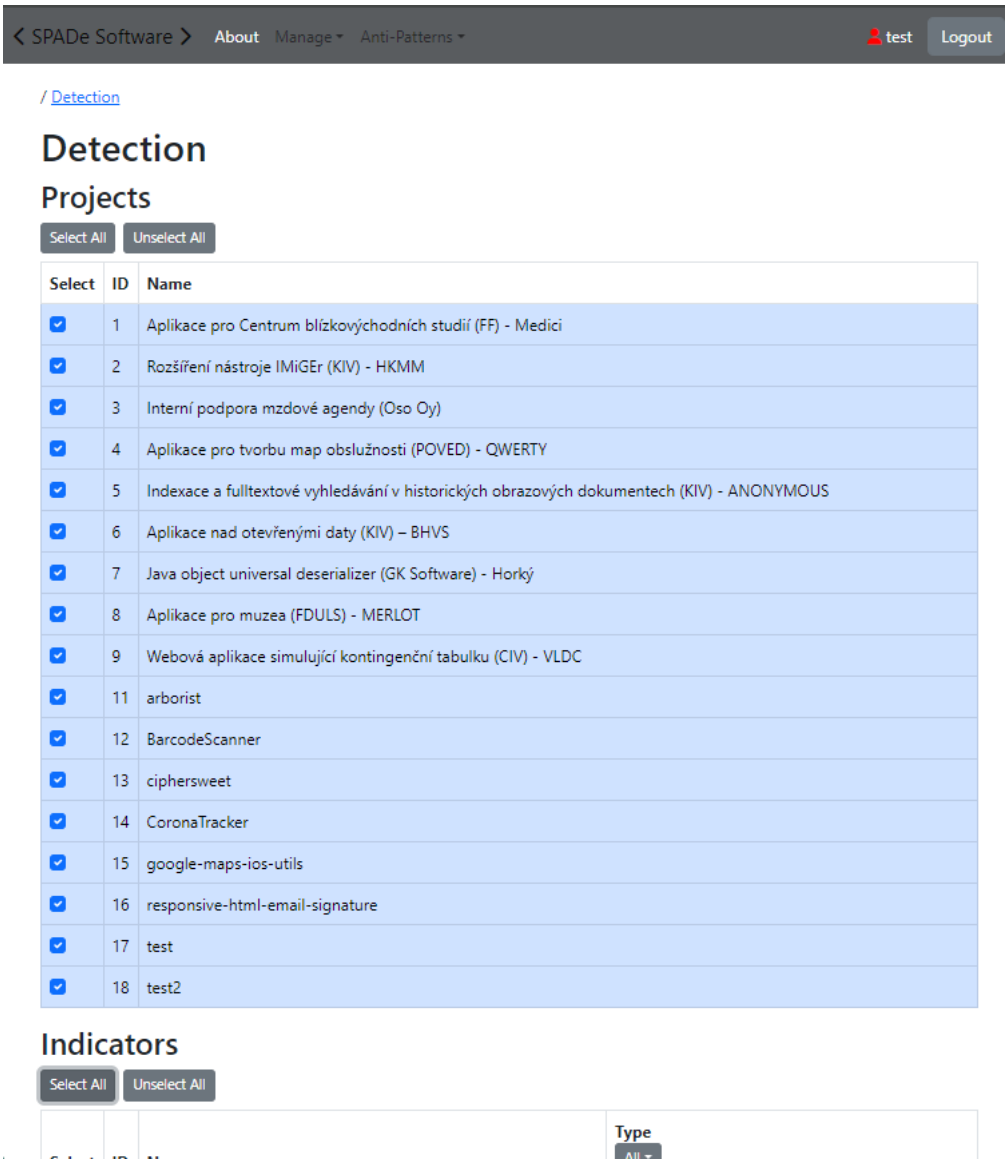
Obrázek C.13: Editace indikátoru

C.3 Detekce na projektových datech

Pro přechod na obrazovku detekcí je nutné zvolit položku *Detection* v kontextové nabídce záložky *Anti-Patterns* (viz Obrázek C.5).

Po stisknutí tlačítka dojde k přesměrování na obrazovku výběru projektů a indikátorů (anti-vzorů) k detekci (viz Obrázek C.14). Na této obrazovce je nutné vybrat minimálně jeden projekt a jeden indikátor. Na další krok lze přejít pomocí tlačítka *Next Step* v dolní části obrazovky.

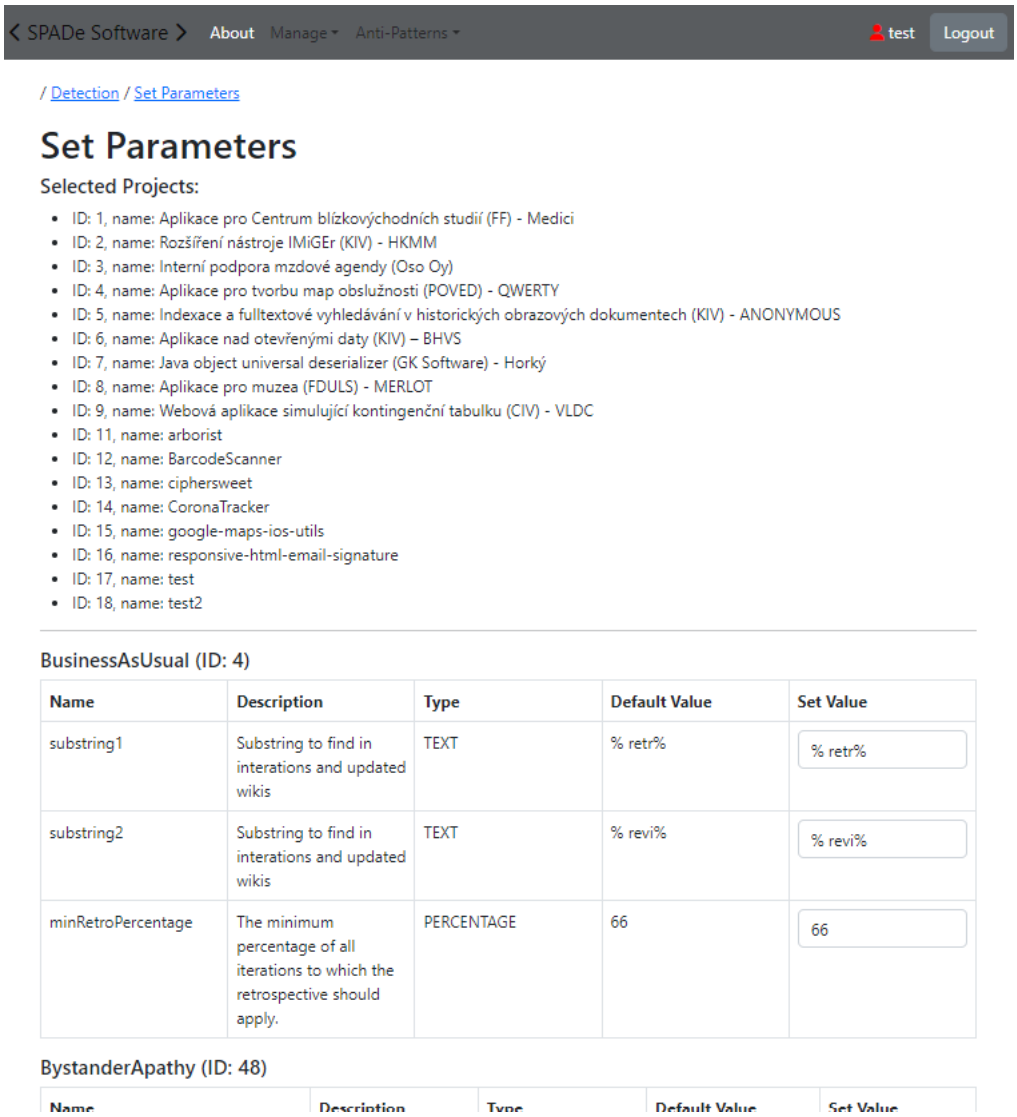
Další obrazovkou je nastavení parametrů, včetně prahových hodnot (viz Obrázek C.15). Na této obrazovce jsou uvedeny vybrané projekty a jednot-



Obrázek C.14: Výběr projektů a indikátorů

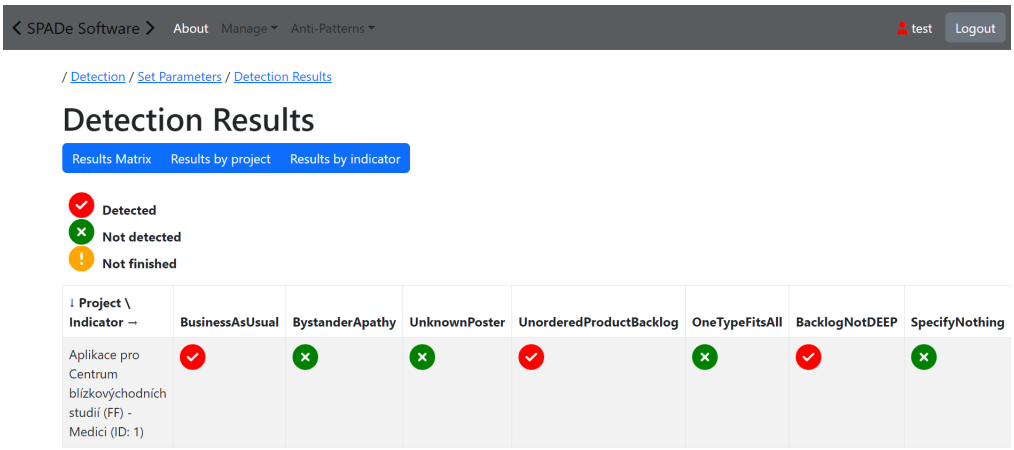
livé indikátory s příslušnými parametry. Parametry jsou přednastaveny na výchozí hodnoty, ale je možné je libovolně změnit. Poté lze spustit detekci stisknutím tlačítka *Start Detection* v dolní části obrazovky, čímž se spustí proces detekce.

Po dokončení procesu detekce dojde k přesměrování na stránku s výsledky. Stránka s výsledky nabízí tři různá zobrazení. Prvním je maticové zobrazení (viz Obrázek C.16), které poskytuje přehled výsledků pro všechny kombinace projektů a indikátorů. Další dvě zobrazení seskupují výsledky podle projektů (viz Obrázek C.17) a podle indikátorů (viz Obrázek C.18). V

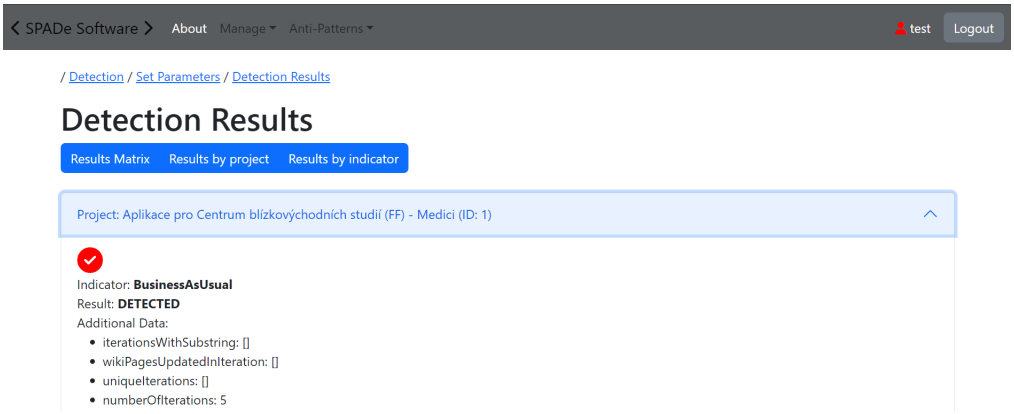


Obrázek C.15: Nastavení parametrů

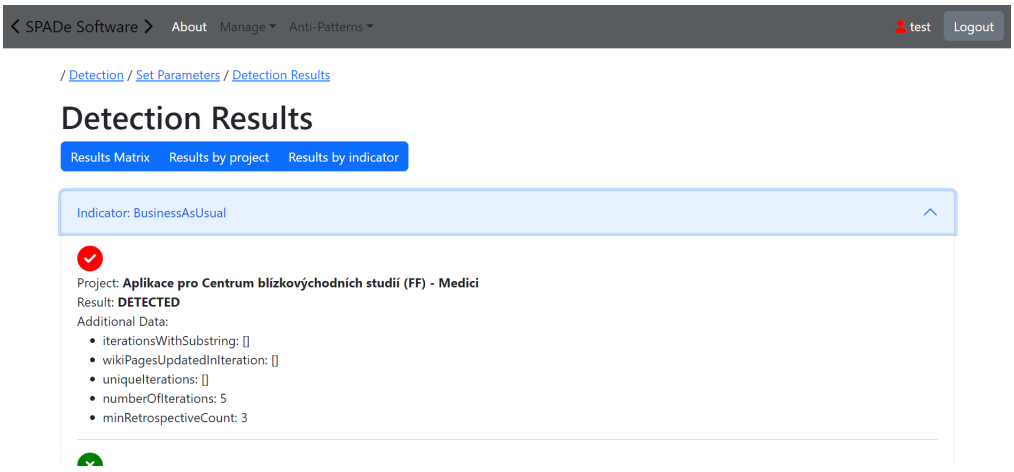
těchto zobrazeních jsou rovněž uvedeny dílčí mezivýsledky a další informace.



Obrázek C.16: Maticové zobrazení výsledků



Obrázek C.17: Výsledky podle projektů



Obrázek C.18: Výsledky podle indikátorů

C.4 Tvorba skriptu

V této části bude popsán postup tvorby skriptu indikátoru s využitím implementovaného jazyka Groovy.

Při zápisu skriptu je možné využívat veškeré prostředky jazyka Groovy. Nejlepším zdrojem informací je oficiální dokumentace[1].

Parametry

Do prostředí, ve kterém bude skript spouštěn, jsou automaticky přidávány všechny parametry indikátoru. Tyto parametry lze využívat jako proměnné, které mají stejné názvy jako odpovídající parametry.

Speciálním parametrem, který je automaticky přidán do prostředí vykonávání skriptu, je identifikátor projektu. Ten je přidán automaticky dle právě zkoumaného projektu. Ve skriptu s ním lze pracovat jako s proměnnou s názvem `projectId`.

Volání metrik

Metriky lze volat pomocí vložené metody, a sice:

```
metricExecutor.executeMetric(<metricName>, <params>);
```

Prvním parametrem je název metriky a dalšími, volitelnými parametry jsou hodnoty vkládané do dotazu dané metriky.

Výsledek volání metriky je vracen jako JSON objekt, proto je nutné s ním takto dále pracovat. Konkrétní výsledky metrik je možné získat v detailu dané metriky

Volání indikátorů

Uvnitř indikátorů je možné volat vnořené indikátory, a to pomocí metody:

```
scriptExecutor.executeScript(<indicatorName>, <params>,
                             <projectId>);
```

Prvním parametrem je `indicatorName`, dalším je mapa parametrů a posledním je identifikátor projektu, který může být zadaný buď staticky jako konkrétní identifikátor projektu, nebo dynamicky dosazením proměnné `projectId`.

[1] https://groovy-lang.org/documentation.html

Uvedená mapa parametrů musí obsahovat veškeré parametry daného indikátoru a musí být typu `Map<String, Object>`. V této mapě jsou názvy parametrů použity jako klíče a hodnoty jako odpovídající hodnoty k těmto klíčům.

Výsledkem volání skriptu je objekt typu `ScriptExecutionResult`. Veškeré atributy a metody tohoto objektu jsou uvedeny v rámci přílohy D.

Vrácení výsledků

Výsledkem skriptu by měl být naplněný objekt typu `ScriptExecutionResult` (viz Příloha D). Je nutné zejména nastavit atribut `result`, který by měl být `true`, pokud byl anti-vzor (indikátor) detekován, nebo `false`, pokud detekován nebyl. Do výsledného objektu je rovněž možné vkládat další libovolné informace pomocí metody `addAdditionalData(<key>, <value>)`.

D Třída ScriptExecutionResult

```
import lombok.*;

import java.util.HashMap;
import java.util.Map;

/**
 * Represents a Script Execution Result object
 * which is obtained after detection is processed
 */
@Getter
@Setter
public class ScriptExecutionResult {
    /**
     * Result of the detection analysis
     * True, if an indicator is detected in the project
     */
    private boolean result;

    /**
     * Result of the detection process
     * True, if the detection process was successfully
     * processed
     */
    private boolean finished;
    private Project project;
    private String indicatorName;
    private Map<String, Object> additionalData;

    public ScriptExecutionResult() {
        this.result = false;
        this.finished = true;
        this.additionalData = new HashMap<>();
    }
```

```
    public ScriptExecutionResult(boolean result,
    boolean finished, Project project, String indicatorName,
    Map<String, Object> additionalData) {
        this.result = result;
        this.finished = finished;
        this.project = project;
        this.indicatorName = indicatorName;
        this.additionalData = additionalData != null ?
            additionalData : new HashMap<>();
    }

    public void addAdditionalData(String key, Object value) {
        additionalData.put(key, value);
    }

    // It is necessary to explicitly define this getResult
    // method to be able to use it in scripts
    public boolean getResult() {
        return this.result;
    }
}
```

E Příklad skriptu indikátoru

Níže uvedený příklad skriptu je konkrétně skript pro anti-vzor Backlog not DEEP. V rámci tohoto skriptu je znázorněno, jakým způsobem jsou volány vnořené indikátory (anti-vzory) spolu s příslušnými parametry.

```
def result = new ScriptExecutionResult();

Map<String, Object> params1 = [
    'dominantPriorityPercentage':
        UnorderedProductBacklog_dominantPriorityPercentage,
    'minimumClassesUsed':
        UnorderedProductBacklog_minimumClassesUsed
]
def UnorderedProductBacklogResult =
    scriptExecutor.executeScript("UnorderedProductBacklog",
        params1, projectId);

Map<String, Object> params2 = [
    'minimumClassesUsed':
        OneTypeFitsAll_minimumClassesUsed,
    'dominantTypePercentage':
        OneTypeFitsAll_dominantTypePercentage
]
def OneTypeFitsAllResult =
    scriptExecutor.executeScript("OneTypeFitsAll",
        params2, projectId);

if (UnorderedProductBacklogResult.getResult() == true
|| OneTypeFitsAllResult.getResult() == true) {
   result.setResult(true);
}

result.addAdditionalData("Unordered Product Backlog detected",
    UnorderedProductBacklogResult.getResult());
result.addAdditionalData("One Type Fits All detected",
    OneTypeFitsAllResult.getResult());

return result;
```